Guide des meilleures pratiques

Amazon EKS

Copyright © 2026 2024. Review license at https://github.com/aws/aws-eks-best-practices/blob/master/LICENSE

Amazon EKS: Guide des meilleures pratiques

Copyright © 2026 2024. Review license at https://github.com/aws/aws-eks-best-practices/blob/master/LICENSE

Les marques et la présentation commerciale d’Amazon ne peuvent être utilisées en relation avec un produit ou un service qui n’est pas d’Amazon, d’une manière susceptible de créer une confusion parmi les clients, ou d’une manière qui dénigre ou discrédite Amazon. Toutes les autres marques commerciales n'appartenant pas à Amazon sont la propriété de leurs propriétaires respectifs, qui peuvent ou non être affiliés, connectés ou sponsorisés par Amazon

Table of Contents

Introduction ... 1
Guides associés ... 2
Contribution ... 2
Sécurité ... 3
Comment utiliser ce guide ... 3
Comprendre le modèle de responsabilité partagée ... 3
Introduction ... 1
Commentaires ... 6
Suggestions de lecture ... 6
Outils et ressources ... 6
Mode automatique EKS - Sécurité ... 6
Architecture de sécurité ... 8
Questions fréquentes (FAQ) ... 17
Gestion de l'identité et des accès ... 20
Contrôle de l'accès aux clusters EKS ... 21
Recommandations relatives à l'accès aux clusters ... 28
Identités et informations d'identification pour les pods EKS ... 35
Recommandations relatives aux identités et aux informations d'identification pour les pods EKS ... 44
Outils et ressources ... 52
Sécurité du pod ... 53
Fonctionnalités de Linux ... 53
Solutions de sécurité pour les pods ... 55
Recommandations ... 64
Outils et ressources ... 52
Multilocataire ... 71
Multi-location souple ... 72
Constructions Kubernetes ... 74
Contrôles d'atténuation ... 77
Multi-location rigide ... 85
Orientations futures ... 85
Outils et ressources de gestion multi-clusters ... 86
Detective Controls ... 86
Recommandations ... 64

Outils et ressources ... 52
Sécurité du réseau ... 94
Contrôle du trafic ... 95
Chiffrement de réseau ... 95
Politique du réseau ... 95
Recommandations ... 64
Groupes de sécurité ... 101
Quand utiliser la politique réseau par rapport au groupe de sécurité pour les pods ? ... 103
Application de la politique Service Mesh ou politique réseau Kubernetes ... 105
ThirdParty Moteurs de politique réseau ... 106
Chiffrement en transit ... 107
Outils et ressources ... 52
Chiffrement des données et gestion des secrets ... 118
Chiffrement au repos ... 118
Gestion des secrets ... 120
Outils et ressources ... 52
Sécurité de l'exécution ... 123
Contextes de sécurité et contrôles Kubernetes intégrés ... 124
Recommandations ... 64
Outils et ressources ... 128
Sécurité de l’infrastructure ... 128
Recommandations ... 64
Solutions de rechange ... 133
Outils et ressources ... 52
Conformité réglementaire ... 136
Déplacement vers la gauche ... 138
Outils et ressources ... 52
Réponse aux incidents et criminalistique ... 140
Exemple de plan de réponse aux incidents ... 140
Recommandations ... 64
Outils et ressources ... 52
Sécurité de l'image ... 145
Recommandations ... 64
Outils et ressources ... 52
Stratégie multi-comptes ... 156

Planification d'une stratégie de comptes à charges de travail multiples pour les clusters à locataires multiples ... 157
Cluster EKS centralisé ... 157
Clusters EKS décentralisés ... 164
Clusters EKS centralisés ou décentralisés ... 166
Gestion des accès aux clusters ... 169
Options de gestion des accès EKS ... 169
Option 1 : centre d'identité AWS IAM avec API de gestion des accès aux clusters (CAM) ... 169
Option 2 : AWS IAM Users/Roles mappé aux groupes Kubernetes ... 172
Option 3 : fournisseurs OIDC ... 172
AWS EKS Pod Identity et IRSA pour les charges de travail ... 173
Recommendation ... 174
Mise à l'échelle automatique du cluster ... 175
Mode automatique EKS ... 175
Raisons d'utiliser le mode automatique ... 176
FAQ ... 176
Karpenter ... 182
Recommandations ... 64
Les meilleures pratiques de Karpenter ... 184
Création NodePools ... 187
Pods de planification ... 192
Recommandations du CoreDNS ... 195
Plans de charpentier ... 196
Ressources supplémentaires ... 94
Cluster Autoscaler ... 196
Présentation de ... 196
Optimisation des performances et de l'évolutivité ... 201
Optimisation des coûts et de la disponibilité ... 204
Cas d’utilisation avancés ... 207
Paramètres supplémentaires ... 209
Ressources supplémentaires ... 94
Références ... 85
Fiabilité ... 215
Comment utiliser ce guide ... 3
Introduction ... 1
Commentaires ... 6

Applications ... 218
Recommandations ... 64
Autoscaler à nacelle horizontale (HPA) ... 223
Autoscaler à nacelle verticale (VPA) ... 224
Mise à jour des applications ... 225
Bilans de santé et auto-guérison ... 227
Recommandations ... 228
Gérer les perturbations ... 230
Recommandations ... 231
Observabilité ... 233
Recommandations ... 233
Plan de contrôle ... 236
Architecture d'EKS ... 236
Recommandations ... 238
Surveiller les métriques du plan de contrôle ... 238
Authentification par cluster ... 241
Webhooks d'admission ... 242
Gestion des mises à niveau de clusters ... 244
Connectivité des terminaux du cluster ... 244
Gestion de grands clusters ... 245
Ressources supplémentaires : ... 246
Plan de données ... 246
Recommandations ... 64
Réseaux ... 253
Modèle de réseau Kubernetes ... 253
Interface réseau de conteneurs (CNI) ... 254
Amazon Virtual Private Cloud (VPC) CNI ... 255
Calculateur de sous-réseau ... 256
Considérations relatives au VPC et aux sous-réseaux ... 257
Présentation de ... 196
Recommandations ... 64
CNI Amazon VPC ... 271
Présentation de ... 196
Recommandations ... 64
Optimisation de l'utilisation des adresses IP ... 286
Optimisation de la consommation IP au niveau des nœuds ... 286

Atténuer l'épuisement des IP ... 286
IPv6 Clusters en cours ... 296
Présentation de ... 196
Recommandations ... 64
Mise en réseau personnalisée ... 314
Exemple de configuration ... 314
Recommandations ... 64
Mode préfixe pour Linux ... 322
Recommandations ... 64
Mode préfixe pour Windows ... 328
Recommandations ... 64
Groupes de sécurité par pod ... 334
Recommandations ... 64
Équilibrage de charge ... 344
Choix du type de Load Balancer ... 344
Provisionnement d'équilibreurs de charge ... 345
Choisir le type de cible du Load Balancer ... 346
Configuration des vérifications de l'état de santé du Load Balancer ... 349
Disponibilité et cycle de vie du pod ... 350
Références ... 85
Annexe ... 355
Surveillance des problèmes de performance du réseau ... 357
Surveillance du trafic CoreDNS pour détecter les problèmes de limitation du DNS ... 357
Surveillance des délais de requêtes DNS à l'aide des métriques Conntrack ... 358
Autres indicateurs de performance réseau importants ... 358
Capture des métriques pour surveiller les charges de travail et détecter les problèmes de performance du réseau ... 359
Exécution de kube-proxy en mode IPVS ... 367
Présentation de ... 196
Capacité de mise à l’échelle ... 372
Comment utiliser ce guide ... 372
Comprendre les dimensions d'échelle ... 372
Très grande mise à l'échelle ... 373
Plan de contrôle ... 374
Limitez la charge de travail et l'éclatement des nœuds ... 374
Diminuez les nœuds et les pods en toute sécurité ... 375

Utiliser le cache côté client lors de l'exécution de Kubectl ... 375
Désactiver la compression kubectl ... 376
Autoscaler Shard Cluster ... 376
Priorité et équité des API ... 378
Récupération de ressources dans le serveur d'API ... 386
Plan de données ... 393
Mise à l'échelle automatique des nœuds ... 393
Utilisez de nombreux types d'instances EC2 ... 393
Préférez des nœuds plus grands pour réduire la charge du serveur d'API ... 394
Utilisez des tailles de nœuds similaires pour des performances de charge de travail cohérentes ... 395
Utiliser efficacement les ressources informatiques ... 396
Automatisez les mises à jour d'Amazon Machine Image (AMI) ... 396
Utiliser plusieurs volumes EBS pour les conteneurs ... 397
Évitez les instances avec de faibles limites d'attachement EBS si les charges de travail utilisent des volumes EBS ... 399
Désactiver la journalisation inutile sur le disque ... 399
Instances de correctifs en place lorsque la vitesse de mise à jour du système d'exploitation est nécessaire ... 400
Services de cluster ... 401
CoreDNS à l'échelle ... 401
Faites dimensionner le serveur de métriques Kubernetes à la verticale ... 403
Durée du lameduck CoreDNS ... 403
Sonde de préparation CoreDNS ... 404
Agents de journalisation et de surveillance ... 404
Charges de travail ... 405
Utilisation IPv6 pour la mise en réseau des pods ... 406
Limiter le nombre de services par espace de noms ... 406
Comprendre les quotas d'Elastic Load Balancer ... 407
Utilisez Route 53, Global Accelerator ou CloudFront ... 407
À utiliser à la EndpointSlices place des points de terminaison ... 408
Utilisez des secrets immuables et externes si possible ... 408
Historique des déploiements limités ... 409
Désactiver enableServiceLinks par défaut ... 409
Limiter les webhooks d'admission dynamique par ressource ... 410
Comparez les charges de travail entre plusieurs clusters ... 410

La théorie de la mise à l'échelle ... 411
Nœuds par rapport au taux de désabonnement ... 411
Réflexion en requêtes par seconde ... 412
Mise à l'échelle des composants distribués ... 412
Goulets d'étranglement en amont et en aval ... 413
Échelle par métriques ... 415
Surveillance du plan de contrôle ... 418
Serveur d'API ... 418
Où est le problème ? ... 418
Planificateur ... 424
Gestionnaire de contrôleurs Kube ... 426
ETCD ... 428
Efficacité et évolutivité des nœuds ... 429
Sélection du nœud ... 429
Emballage Node Bin ... 431
Utilisation ou saturation ... 439
Configuration des limites du processeur ... 447
Kubernetes SLOs ... 451
Kubernetes SLOs ... 451
Métriques SLI de Kubernetes ... 454
SLOs sur votre cluster ... 458
Limites connues et Quotas de Service ... 459
Autres quotas de Service AWS ... 460
Limitation des demandes AWS ... 467
Autres limites connues ... 468
Améliorations de clusters ... 469
Présentation de ... 469
Avant la mise à niveau ... 469
Conservez votre cluster up-to-date ... 470
Consultez le calendrier de publication d'EKS ... 471
Comprendre comment le modèle de responsabilité partagée s'applique aux mises à niveau de clusters ... 471
Mettez à niveau les clusters sur place ... 472
Améliorez votre plan de contrôle et votre plan de données en séquence ... 472
Utilisez la documentation EKS pour créer une liste de contrôle de mise à niveau ... 473

Mettez à niveau les modules complémentaires et les composants à l'aide de l'API Kubernetes ... 474
Vérifiez les exigences de base d'EKS avant la mise à niveau ... 475
Vérifiez les adresses IP disponibles ... 476
Vérifier le rôle EKS IAM ... 477
Migrer vers les modules complémentaires EKS ... 477
Identifiez et corrigez l'utilisation supprimée de l'API avant de mettre à niveau le plan de contrôle ... 478
Informations sur les clusters ... 479
K ube-no-trouble ... 481
Pluton ... 481
Ressources ... 482
Mettez à jour les charges de travail Kubernetes. Utilisez kubectl-convert pour mettre à jour les manifestes ... 483
Configurez PodDisruptionBudgets et topologySpreadConstraints garantissez la disponibilité de vos charges de travail pendant la mise à niveau du plan de données ... 483
Utilisez Managed Node Groups ou Karpenter pour simplifier les mises à niveau du plan de données ... 485
Confirmer la compatibilité des versions avec les nœuds existants et le plan de contrôle ... 486
Activer l'expiration des nœuds pour les nœuds gérés par Karpenter ... 486
Utiliser la fonctionnalité Drift pour les nœuds gérés par Karpenter ... 487
Utilisez eksctl pour automatiser les mises à niveau pour les groupes de nœuds autogérés ... 487
Backup le cluster avant de procéder à la mise à niveau ... 488
Redémarrer les déploiements de Fargate après la mise à niveau du plan de contrôle ... 488
Évaluez les Blue/Green clusters comme alternative aux mises à niveau de clusters sur place ... 488
Suivez les modifications majeures prévues dans le projet Kubernetes — Think ahead ... 489
Conseils spécifiques sur les suppressions de fonctionnalités ... 490
Suppression de Dockershim en 1.25 - Utilisez le détecteur pour Docker Socket (DDS) ... 490
Suppression de PodSecurityPolicy la version 1.25 - Migrer vers les normes de sécurité des pods ou vers une solution policy-as-code ... 490
Obsolète du pilote de stockage intégré dans la version 1.23 - Migrer vers des pilotes CSI (Container Storage Interface) ... 490
Ressources supplémentaires ... 491
ClowdHaus Conseils de mise à niveau EKS ... 491
GoNoGo ... 491
Optimisation des coûts ... 492

Consignes générales 492
Meilleures pratiques d'optimisation des coûts chez EKS 492
Comment utiliser ce guide 3
Principaux services AWS et fonctionnalités de Kubernetes 493
Commentaires 6
Cadre 494
Le pilier See : mesure et responsabilité 494
Le pilier de l'épargne : optimisation des coûts 495
Le pilier du plan : planification et prévisions 496
Le pilier Run 496
Références 85
Sensibilisation 497
Recommandations 64
Autres outils 505
Calcul 506
Dimensionnez correctement vos charges de travail 507
Réduire la consommation 508
Réduction de la capacité inutilisée 509
Optimisation des types de capacité de calcul 515
Optimisation de l'utilisation du calcul 519
Réseau 521
Communication d'un pod à un autre 521
Communication entre le Load Balancer et le Pod 534
Transfert de données depuis le registre des conteneurs 536
Transfert de données vers Internet et les services AWS 537
Transfert de données entre VPCs 540
Utilisation d'un Service Mesh 543
Ressources supplémentaires 94
Stockage 552
Présentation 196
Volumes éphémères 552
Volumes persistants 553
Autres considérations 561
Observabilité 562
Introduction 562
Journalisation 562

Plan de contrôle EKS ... 562
Plan de données EKS ... 563
Métriques ... 568
Tracing ... 572
Ressources supplémentaires : ... 94
Windows ... 575
Gestion des AMI ... 575
Gestion de votre propre AMI Windows optimisée pour Amazon EKS ... 576
Configuration d'un lancement plus rapide pour une optimisation EKS personnalisée AMIs ... 577
Mise en cache des couches de base Windows sur mesure AMIs ... 578
Billet de blog ... 579
GMSA pour conteneurs Windows ... 579
Qu'est-ce qu'un compte GMSA ... 579
Conteneur Windows et cas d'utilisation du GMSA ... 580
Renforcement de Windows Server ... 582
Réduction de la surface d'attaque avec Windows Server Core ... 582
Éviter les connexions RDP ... 583
Amazon Inspector ... 584
Amazon GuardDuty ... 585
Sécurité dans Amazon EC2 pour Windows ... 585
Numérisation d'images Windows ... 586
Versions et licences Windows ... 586
Version Windows Server ... 586
Licences ... 587
Logging ... 587
Recommandations en matière de journalisation ... 587
Surveillance des conteneurs Windows ... 588
Réseau Windows ... 593
Présentation de Windows Container Networking ... 593
Gestion des adresses IP ... 594
Options d'interface réseau de conteneurs (CNI) ... 598
Politiques du réseau ... 598
Gestion de la mémoire et des systèmes ... 598
Système de réservation et mémoire Kubelet ... 599
Exigences relatives à la mémoire des conteneurs Windows ... 599
Conclusion ... 600

Gestion de l'infrastructure ... 600
Pousser et extraire des images Windows ... 601
Référence ... 603
Planification ... 603
Meilleures pratiques PODs d'attribution aux nœuds ... 603
Veiller à ce que les charges de travail spécifiques au système d'exploitation arrivent sur le conteneur hôte approprié ... 604
Gestion de plusieurs versions de Windows dans le même cluster ... 605
Simplification NodeSelector et tolérance dans les manifestes Pod à l'aide de RuntimeClass ... 606
Support pour les groupes de nœuds gérés ... 607
Documentations supplémentaires ... 608
Pod Security pour les conteneurs Windows ... 608
Options de stockage ... 611
Qu'est-ce qu'un plugin intégré à l'arbre par rapport à un plugin out-of-tree de volume ? ... 611
Plug-in In-Tree Volume pour Windows ... 612
Out-of-tree pour Windows ... 616
Serveur FSx de fichiers Amazon pour Windows ... 616
Renforcement des images des conteneurs Windows ... 617
1. Configurer les politiques de compte (mot de passe ou verrouillage) à l'aide des politiques de sécurité locales et du registre ... 619
2. Politiques d'audit ... 621
3. Bonnes pratiques de sécurité IIS pour les conteneurs Windows ... 624
4. Principe du moindre privilège ... 630
Réflexions finales : pourquoi la sécurisation de vos conteneurs Windows est indispensable dans le contexte actuel des menaces ... 633
Hybride ... 635
Déconnexion du réseau ... 635
Bonnes pratiques ... 636
Kubernetes pod Failover ... 641
Trafic réseau des applications ... 648
Identifiants de l'hôte ... 653
AI/ML ... 656
Commentaires ... 656
Calcul ... 656
Optimisation des ressources GPU et gestion des coûts ... 656

Résilience des nœuds et gestion des tâches de formation ... 672
Mise à l'échelle et performances des applications ... 675
Allocation dynamique des ressources pour une gestion avancée du GPU ... 677
Réseaux ... 724
Envisagez une bande passante réseau plus élevée ou un adaptateur Elastic Fabric pour les applications nécessitant une communication inter-nœuds élevée ... 724
Augmentez le nombre d'adresses IP disponibles pour accélérer le lancement des pods ... 725
Sécurité ... 726
Sécurité et conformité ... 726
Stockage ... 727
Gestion et stockage des données ... 728
Observabilité ... 741
Surveillance et observabilité ... 742
Observabilité et métriques ... 743
Performance ... 760
Gestion des artefacts ML, des frameworks de service et optimisation du démarrage ... 760
Optimisation des temps d'extraction des images des conteneurs ... 763
Pensez NVMe au stockage kubelet et containerd ... 766
Contribuez ... 768
Résumé pour les contributeurs existants ... 768
Configuration d'un environnement d'édition local ... 768
Forker et cloner le dépôt ... 768
Ouvrez l'espace de travail VS Code ... 769
Modifier un fichier ... 769
Soumettre une pull request ... 769
Utilisez l'éditeur Web github.dev ... 770
Modifier une seule page ... 770
Afficher et définir l'ID d'une page ... 770
Définissez l'ID de page ... 771
Création d’une nouvelle page ... 771
Création de métadonnées de page ... 771
Ajouter à la table des matières ... 772
Insérer une image ... 772
Vérifiez le style avec Vale ... 773
Création d'un aperçu local ... 773
AsciiDoc Aide-mémoire ... 774

Formatage de base 774
En-têtes 774
Lists 774
Liens 774
Images 775
Blocs de code 775
Tables 775
Des admonestations 775
Inclut 776
..... dcclxxvii

Guide des meilleures pratiques Amazon EKS

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Bienvenue dans les guides des meilleures pratiques d'EKS. L'objectif principal de ce projet est de proposer un ensemble de bonnes pratiques pour les opérations du deuxième jour pour Amazon EKS. Nous avons choisi de publier ces directives GitHub afin de pouvoir les itérer rapidement, de fournir des recommandations rapides et efficaces pour diverses préoccupations et d'intégrer facilement les suggestions de l'ensemble de la communauté.

Nous avons actuellement publié des guides sur les sujets suivants :

- Bonnes pratiques en matière de sécurité
- Meilleures pratiques en matière de fiabilité
- Meilleures pratiques pour la mise à l'échelle automatique des clusters : Karpenter
- Meilleures pratiques pour la mise à l'échelle automatique des clusters : cluster-autoscaler
- Meilleures pratiques pour la mise à l'échelle automatique des clusters : mode automatique EKS
- Bonnes pratiques en matière de mise en réseau
- Meilleures pratiques en matière d'évolutivité
- Meilleures pratiques pour les mises à niveau de clusters
- Meilleures pratiques en matière d'optimisation des coûts
- Bonnes pratiques pour l'exécution de conteneurs Windows
- Bonnes pratiques pour les déploiements hybrides
- Bonnes pratiques en matière d'exécution des charges AI/ML de travail

Nous avons également ouvert le code source d'une CLI (interface de ligne de commande) basée sur Python appelée hardeneks pour vérifier certaines des recommandations de ce guide.

À l'avenir, nous publierons des conseils sur les meilleures pratiques en matière de performance, d'optimisation des coûts et d'excellence opérationnelle.

Guides associés

Outre le guide de l'utilisateur d'EKS, AWS a publié plusieurs autres guides susceptibles de vous aider à implémenter EKS.

- Guides des meilleures pratiques en matière de conteneurs EMR
- Données sur EKS
- Bonnes pratiques d'observabilité d'AWS
- Blueprints Amazon EKS pour Terraform
- Démarrage rapide d'Amazon EKS Blueprints

Contribution

Nous vous encourageons à contribuer à ces guides. Si vous avez mis en place une pratique qui s'est révélée efficace, veuillez nous en faire part en signalant un problème ou en lançant une pull request. De même, si vous découvrez une erreur ou une faille dans les directives que nous avons déjà publiées, veuillez envoyer un PR pour la corriger. Les directives de soumission se PRs trouvent dans nos directives de contribution.

Meilleures pratiques en matière de sécurité

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Ce guide fournit des conseils sur la protection des informations, des systèmes et des actifs qui dépendent d'EKS tout en apportant une valeur commerciale grâce à des évaluations des risques et à des stratégies d'atténuation. Le présent guide fait partie d'une série de guides de bonnes pratiques publiés par AWS pour aider les clients à implémenter EKS conformément aux meilleures pratiques. Des guides sur les performances, l'excellence opérationnelle, l'optimisation des coûts et la fiabilité seront disponibles dans les prochains mois.

Comment utiliser ce guide

Ce guide est destiné aux professionnels de la sécurité chargés de mettre en œuvre et de surveiller l'efficacité des contrôles de sécurité pour les clusters EKS et les charges de travail qu'ils prennent en charge. Le guide est organisé en différents domaines thématiques pour en faciliter la consommation. Chaque rubrique commence par un bref aperçu, suivi d'une liste de recommandations et de bonnes pratiques pour sécuriser vos clusters EKS. Il n'est pas nécessaire de lire les sujets dans un ordre particulier.

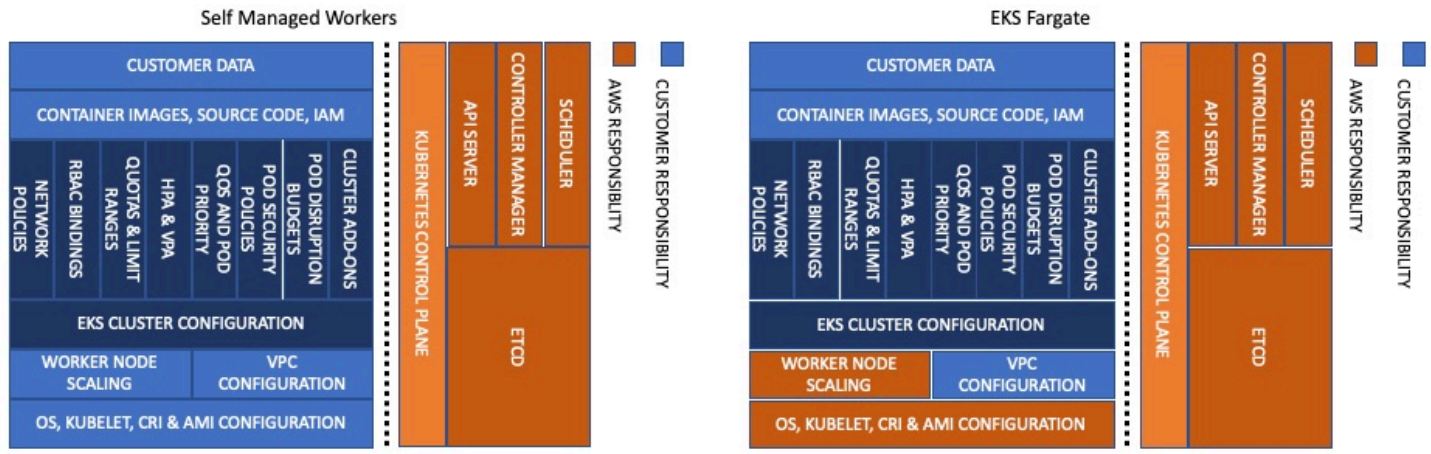
Comprendre le modèle de responsabilité partagée

La sécurité et la conformité sont considérées comme des responsabilités partagées lors de l'utilisation d'un service géré tel que EKS. D'une manière générale, AWS est responsable de la sécurité « du » cloud alors que vous, le client, êtes responsable de la sécurité « dans » le cloud. Avec EKS, AWS est responsable de la gestion du plan de contrôle Kubernetes géré par EKS. Cela inclut les nœuds du plan de contrôle Kubernetes, la base de données ETCD et les autres infrastructures nécessaires à AWS pour fournir un service sécurisé et fiable. En tant que consommateur d'EKS, vous êtes en grande partie responsable des sujets abordés dans ce guide, par exemple l'IAM, la sécurité des modules, la sécurité des environnements d'exécution, la sécurité du réseau, etc.

En matière de sécurité de l'infrastructure, AWS assumera des responsabilités supplémentaires à mesure que vous passerez de travailleurs autogérés à des groupes de nœuds gérés, puis à Fargate.

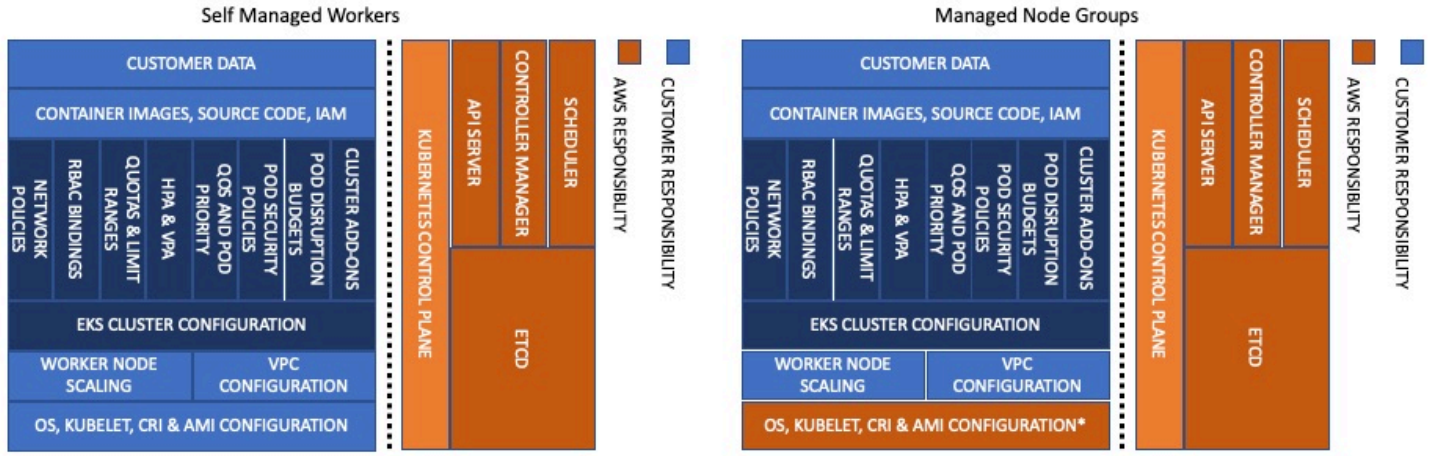
Par exemple, avec Fargate, AWS devient responsable de la sécurisation instance/runtime du sous-jacent utilisé pour exécuter vos pods.

Modèle de responsabilité partagée - Fargate



AWS assumera également la responsabilité de maintenir l'AMI optimisée pour EKS à jour avec les versions des correctifs et des correctifs de sécurité de Kubernetes. Les clients utilisant des groupes de nœuds gérés (MNG) sont responsables de la mise à niveau de leurs groupes de nœuds vers la dernière AMI via l'API EKS, la CLI, Cloudformation ou la console AWS. De plus, contrairement à Fargate MNGs , il ne fera pas automatiquement évoluer votre infrastructure/cluster. [Cela peut être géré par le cluster-autoscaler ou par d'autres technologies telles que Karpenter, le scalage automatique natif d'AWS, Ocean ou Atlassian Escalator SpotInst.]

Modèle de responsabilité partagée - MNG



Avant de concevoir votre système, il est important de savoir où se situe la ligne de démarcation entre vos responsabilités et le fournisseur du service (AWS).

Pour plus d'informations sur le modèle de responsabilité partagée, voir https://aws.amazon.com/compliance/shared-responsibility-model/

Introduction

Plusieurs domaines de bonnes pratiques en matière de sécurité sont pertinents lors de l'utilisation d'un service Kubernetes géré tel que EKS :

- Gestion de l’identité et des accès
- Sécurité du pod
- Sécurité de l'exécution
- Sécurité du réseau
- Multilocataire
- Compte multiple pour location multiple
- Detective Controls
- Sécurité de l’infrastructure
- Chiffrement des données et gestion des secrets
- Conformité réglementaire
- Réponse aux incidents et criminalistique
- Sécurité de l'image

Lors de la conception de tout système, vous devez réfléchir à ses implications en matière de sécurité et aux pratiques susceptibles d'affecter votre posture de sécurité. Par exemple, vous devez contrôler qui peut effectuer des actions sur un ensemble de ressources. Vous devez également être en mesure d'identifier rapidement les incidents de sécurité, de protéger vos systèmes et services contre tout accès non autorisé et de préserver la confidentialité et l'intégrité des données grâce à la protection des données. Le fait de disposer d'un ensemble de processus bien définis et répétés pour répondre aux incidents de sécurité améliorera également votre posture de sécurité. Ces outils et techniques sont importants, car ils soutiennent des objectifs tels que la prévention des pertes financières ou le respect des obligations réglementaires.

AWS aide les entreprises à atteindre leurs objectifs de sécurité et de conformité en proposant un ensemble complet de services de sécurité qui ont évolué en fonction des commentaires d'un large éventail de clients soucieux de la sécurité. En offrant une base hautement sécurisée, les clients peuvent consacrer moins de temps à « soulever des objets lourds de manière indifférenciée » et plus de temps à atteindre leurs objectifs commerciaux.

Commentaires

Ce guide est publié GitHub afin de recueillir les commentaires et suggestions directs de l'ensemble de la EKS/Kubernetes communauté. Si vous avez une bonne pratique que vous pensez que nous devrions inclure dans le guide, veuillez signaler un problème ou soumettre un PR dans le GitHub référentiel. Notre intention est de mettre à jour le guide périodiquement à mesure que de nouvelles fonctionnalités sont ajoutées au service ou lorsqu'une nouvelle bonne pratique évolue.

Suggestions de lecture

Livre blanc sur la sécurité Kubernetes, sponsorisé par le groupe de travail sur les audits de sécurité, ce livre blanc décrit les principaux aspects de la surface d'attaque et de l'architecture de sécurité de Kubernetes dans le but d'aider les professionnels de la sécurité à prendre des décisions judicieuses en matière de conception et de mise en œuvre.

La CNCF a également publié un white paper sur la sécurité native du cloud. Le paper examine l'évolution du paysage technologique et préconise l'adoption de pratiques de sécurité conformes aux DevOps processus et aux méthodologies agiles.

Outils et ressources

Atelier d'immersion sur la sécurité Amazon EKS

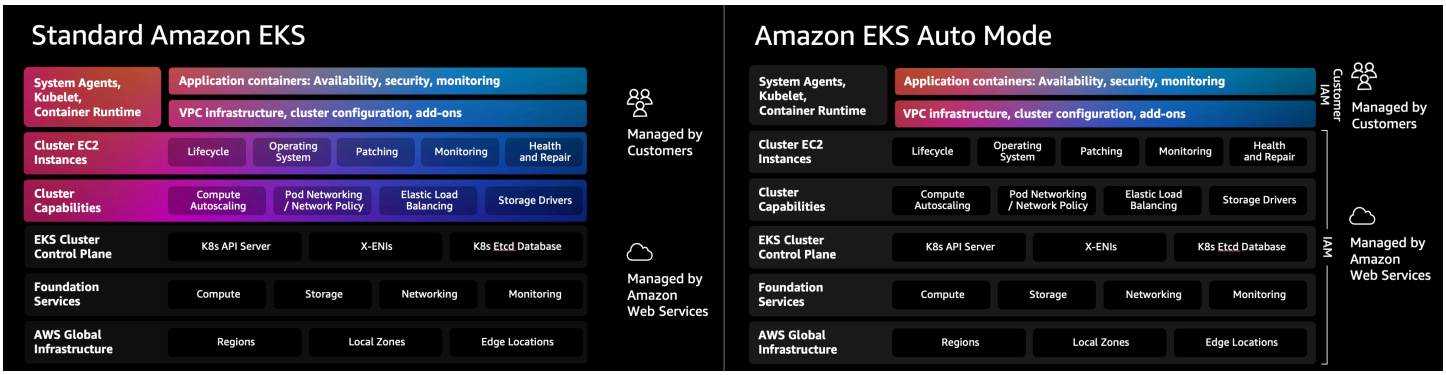
Mode automatique EKS - Sécurité

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Amazon EKS Auto Mode introduit des fonctionnalités de sécurité améliorées en étendant la gestion de la sécurité d'AWS au-delà du plan de contrôle pour inclure les nœuds de travail et les composants

principaux du cluster. Ce modèle de sécurité complet aide les entreprises à maintenir une solide posture de sécurité tout en réduisant les frais opérationnels.

Modèle de responsabilité partagée - Mode automatique EKS



Les principales améliorations apportées à la sécurité dans le mode automatique d'EKS sont les suivantes :

- Système d'exploitation minimal optimisé pour les conteneurs avec une surface d'attaque réduite
- Meilleures pratiques de sécurité appliquées grâce aux instances gérées EC2
- Gestion automatisée des correctifs de sécurité avec rotation obligatoire des nœuds
- Isolation des nœuds et limites de sécurité intégrées
- Intégration IAM rationalisée avec un minimum d'autorisations requises

Le mode automatique d'EKS applique ces contrôles de sécurité par défaut, aidant ainsi les entreprises à répondre à leurs exigences de sécurité et de conformité tout en simplifiant les opérations des clusters. Cette approche est conforme defense-in-depth aux principes et fournit plusieurs niveaux de contrôles de sécurité au niveau de l'infrastructure, des nœuds et de la charge de travail.

> **Important**
>
> Bien que le mode automatique d'EKS offre des fonctionnalités de sécurité améliorées, les entreprises doivent tout de même mettre en œuvre des contrôles de sécurité appropriés au niveau de la couche application et suivre les meilleures pratiques de sécurité pour leurs charges de travail exécutées sur le cluster.

Architecture de sécurité

Le mode automatique EKS met en œuvre des contrôles de sécurité sur plusieurs couches de l'infrastructure EKS, du plan de contrôle aux nœuds individuels. Comprendre cette architecture est essentiel pour gérer et sécuriser efficacement vos clusters EKS.

Sécurité du plan de contrôle

Le plan de contrôle EKS en mode automatique EKS respecte les mêmes normes de sécurité élevées que les clusters EKS traditionnels tout en ajoutant de nouvelles fonctionnalités de sécurité :

- Chiffrement des enveloppes : toutes les données de l'API Kubernetes sont automatiquement chiffrées à l'aide du chiffrement des enveloppes.
- Intégration KMS : utilise AWS KMS avec le fournisseur Kubernetes KMS v2, avec des options pour les clés détenues par AWS ou les clés gérées par le client (CMK).
- Gestion améliorée des composants : les composants critiques tels que l'auto-scaling, la gestion ENI et les contrôleurs EBS sont déplacés hors du cluster et gérés par AWS.
- Contrôles de sécurité et capacités d'audit améliorés : les autorisations requises en mode automatique d'EKS, au-delà des clusters EKS standard, sont entièrement gérées via le rôle IAM du cluster plutôt que par des rôles de nœud individuels.

Intégration IAM et gestion des accès

Le mode automatique d'EKS fournit une intégration améliorée avec AWS Identity and Access Management (IAM) via EKS Access Entries et EKS Pod Identity.

Gestion de l'accès au cluster

Le mode automatique EKS apporte des améliorations à la gestion de l'accès aux clusters via l'API de gestion des accès aux clusters (CAM) :

- Modes d'authentification standardisés via `EKS_API`
- Sécurité renforcée grâce à une gestion des accès basée sur des API
- Contrôle d'accès simplifié à l'aide des entrées d'accès et des politiques d'accès

Des entrées d'accès peuvent être créées pour gérer l'accès au cluster :

```
aws eks create-access-entry \
    --cluster-name ${EKS_CLUSTER_NAME} \
    --principal-arn arn:aws:iam::${ACCOUNT_ID}:role/${IAM_ROLE_NAME} \
    --type STANDARD
```

> **Important**
>
> Bien qu'il soit toujours possible de créer un cluster en mode automatique EKS avec le mode `CONFIG_MAP_AND_API` d'authentification, il ne s'agit pas de l'approche standard et il est fortement recommandé d'utiliser le mode `API` d'authentification par défaut pour les nouveaux clusters. `API`l'authentification basée sur l'authentification améliore la sécurité et simplifie la gestion des accès par rapport à ConfigMap l'approche traditionnelle.

Identité du pod EKS

Le mode automatique d'EKS est fourni avec l'agent d'identité Pod déjà déployé, ce qui permet d'accorder des autorisations AWS IAM aux pods de manière simplifiée :

- Gestion simplifiée des autorisations IAM sans configuration du fournisseur OIDC
- Frais d'exploitation réduits par rapport à l'IRSA
- Sécurité améliorée grâce au balisage des sessions et à la prise en charge de l'ABAC

```
aws eks create-pod-identity-association \
  --cluster-name ${EKS_CLUSTER_NAME} \
  --role-arn arn:aws:iam::${AWS_ACCOUNT_ID}:role/${IAM_ROLE_NAME} \
  --namespace ${NAMESPACE} \
  --service-account ${SERVICE_ACCOUNT_NAME}
```

> **Important**
>
> Pod Identity est l'approche recommandée pour les autorisations IAM en mode automatique d'EKS, car elle fournit des fonctionnalités de sécurité améliorées et une gestion simplifiée par rapport à l'IRSA.

Rôle IAM de nœud

Le mode automatique EKS utilise un nouveau mode `AmazonEKSWorkerNodeMinimalPolicy` qui fournit uniquement les autorisations nécessaires au fonctionnement des nœuds du mode automatique EKS. Ces autorisations :

- Fournir un ensemble d'autorisations réduit par rapport aux politiques de nœud traditionnelles
- Adhérer au principe du moindre privilège
- Sont automatiquement gérés et mis à jour par AWS

Cette approche de politique minimale permet d'améliorer le niveau de sécurité en limitant les autorisations accordées au nœud et à ses charges de travail.

Sécurité des nœuds

Le mode automatique EKS introduit plusieurs améliorations de sécurité importantes au niveau du nœud :

Sécurité des instances gérées EC2

Les nœuds EKS Auto Mode utilisent des instances gérées Amazon EC2 dotées de propriétés de sécurité améliorées :

- Restrictions imposées par l'IAM qui empêchent les opérations susceptibles de compromettre la capacité d'AWS à exploiter les nœuds
- Modèles d'infrastructure immuables dans lesquels les modifications de configuration nécessitent le remplacement de nœuds
- Remplacement obligatoire du nœud dans les 21 jours pour garantir des mises à jour de sécurité régulières
- Accès restreint aux métadonnées de l'instance à l' IMDSv2 aide de limites de sauts contrôlées

Sécurité du système d'exploitation

Le système d'exploitation est une variante personnalisée de Bottlerocket, optimisée pour le mode automatique EKS, qui inclut :

- Système de fichiers racine en lecture seule

- SELinux activé par défaut avec des contrôles d'accès obligatoires
- Isolation automatique du pod à l'aide d'étiquettes SELinux MCS uniques
- Accès SSH désactivé et suppression des services inutiles
- Correctifs de sécurité automatisés grâce à la rotation des nœuds

Sécurité des composants du nœud

Les composants du nœud sont configurés conformément aux meilleures pratiques de sécurité :

- Kubelet configuré avec des valeurs par défaut sécurisées
- Configuration renforcée du conteneur lors de l'exécution
- Gestion et rotation automatisées des certificats
- node-to-control-planeCommunication restreinte

Sécurité du réseau

Le mode automatique EKS met en œuvre plusieurs fonctionnalités de sécurité réseau pour garantir une communication sécurisée au sein du cluster et avec les ressources externes :

Politique du réseau VPC CNI

Le mode automatique d'EKS tire parti de la prise en charge native de la politique réseau Kubernetes du plug-in Amazon VPC CNI :

- S'intègre à l'API Kubernetes Network Policy en amont
- Permet un contrôle précis de la communication pod-to-pod
- Supporte à la fois les règles d'entrée et de sortie

Pour activer la prise en charge des politiques réseau en mode automatique EKS, vous devez configurer le module complémentaire VPC CNI avec un manifeste. `configMap` Voici un exemple :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
```

```
data:
  enable-network-policy: "true"
```

Il est également nécessaire de définir si le support de la politique réseau est configuré dans la classe Node, comme illustré ici :

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: example-node-class
spec:
  networkPolicy: DefaultAllow
  networkPolicyEventLogs: Enabled
```

Une fois activé, vous pouvez créer des politiques réseau pour contrôler le trafic :

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

Gestion améliorée de l'ENI

Le mode automatique EKS fournit une sécurité améliorée pour la gestion de l'Elastic Network Interface (ENI) :

- Connexion et configuration ENI gérées par AWS
- Séparation du trafic de contrôle du trafic de données
- Gestion automatisée des adresses IP avec des privilèges réduits requis sur les nœuds

Sécurité du stockage

Le mode automatique EKS fournit des fonctionnalités de sécurité améliorées pour le stockage éphémère et persistant :

Stockage éphémère

- Toutes les données écrites dans des volumes éphémères sont automatiquement cryptées
- Utilise l'algorithme cryptographique AES-256 standard
- Chiffrement et déchiffrement gérés de manière fluide par le service

Volumes EBS

- Les volumes EBS root et de données sont toujours chiffrés
- Les volumes sont configurés pour être supprimés à la fin de l'instance
- Il existe une option permettant de spécifier des clés KMS personnalisées pour le chiffrement

Intégration EFS

- Support pour le chiffrement en transit avec EFS
- Chiffrement automatique au repos pour les systèmes de fichiers EFS
- Intégration aux points d'accès EFS pour un contrôle d'accès amélioré

> **Important**
>
> Lorsque vous utilisez EFS avec le mode automatique EKS, assurez-vous que les paramètres de chiffrement appropriés sont configurés au niveau du système de fichiers EFS, car le mode automatique EKS ne gère pas directement le chiffrement EFS.

Surveillance et journalisation

Le mode automatique EKS fournit des fonctionnalités de surveillance et de journalisation améliorées pour vous aider à conserver une visibilité sur le niveau de sécurité et la santé opérationnelle de votre cluster.

Journalisation de plan de contrôle

Le mode automatique EKS conserve les mêmes fonctionnalités de journalisation sur le plan de contrôle que l'EKS standard, mais il active tous les journaux par défaut pour une surveillance améliorée.

- Les journaux sont envoyés à Amazon CloudWatch Logs
- Par défaut, le mode automatique d'EKS active tous les journaux du plan de contrôle : serveur d'API, audit, authentificateur, gestionnaire de contrôleurs et planificateur
- Le mode automatique EKS permet une visibilité détaillée des opérations du cluster et des événements de sécurité

> **Important**
>
> La journalisation sur le plan de contrôle entraîne des coûts supplémentaires pour le stockage des journaux. CloudWatch Réfléchissez bien à votre stratégie de journalisation afin de trouver un équilibre entre les besoins de sécurité et la gestion des coûts.

Journalisation au niveau des nœuds

Le mode automatique EKS améliore la journalisation au niveau des nœuds :

- Les journaux du système sont automatiquement collectés et sont accessibles via CloudWatch Logs
- Les journaux des nœuds sont conservés même après la fermeture du nœud, ce qui facilite l'analyse après l'incident
- Visibilité améliorée sur les événements de sécurité et les problèmes opérationnels au niveau des nœuds

GuardDuty Intégration avec Amazon

Les clusters EKS Auto Mode s'intègrent parfaitement à Amazon GuardDuty pour une meilleure détection des menaces. Ses caractéristiques sont les suivantes :

- Numérisation automatique des journaux d'audit du plan de contrôle
- Surveillance du temps d'exécution pouvant être activée pour la surveillance des charges de travail
- Intégration aux GuardDuty résultats et aux mécanismes d'alerte existants

Pour activer la protection en mode automatique d'EKS sur Amazon GuardDuty pour les journaux d'audit Kubernetes, vous pouvez exécuter la commande suivante :

```
aws guardduty update-detector \
```

```
    --detector-id 12abc34d567e8fa901bc2d34e56789f0 \
    --data-sources '{"Kubernetes":{"AuditLogs":{"Enable":true}}}'
```

GuardDuty Intégration à Amazon pour la sécurité des environnements d'exécution

Amazon GuardDuty fournit une surveillance essentielle de la sécurité d'exécution pour les clusters en mode automatique EKS, offrant des fonctionnalités complètes de détection des menaces et de surveillance de la sécurité. Cette intégration est particulièrement importante car elle permet d'identifier les menaces de sécurité potentielles et les activités malveillantes en temps réel.

GuardDuty Caractéristiques principales du mode automatique EKS

- Surveillance du temps d'exécution :
  - Surveillance continue du comportement d'exécution
  - Détection d'activités malveillantes ou suspectes
  - Identification des tentatives d'évasion potentielles du conteneur
  - Surveillance de l'exécution inhabituelle des processus ou des connexions réseau
- Détection des menaces spécifiques à Kubernetes :
  - Identification des tentatives suspectes de déploiement de pods
  - Détection des conteneurs endommagés
  - Surveillance des lancements de conteneurs privilégiés
  - Identification d'une utilisation suspecte du compte de service
- Types de recherche complets :
  - Policy:Kubernetes/* - Détecte les violations des meilleures pratiques de sécurité
  - Impact:Kubernetes/* - Identifie les ressources potentiellement affectées
  - Discovery : Kubernetes/* - Détecte les activités de reconnaissance
  - Execution:Kubernetes/* - Identifie les modèles d'exécution suspects
  - Persistence:Kubernetes/* - Détecte les menaces persistantes potentielles

Pour activer la protection en mode automatique d'EKS sur Amazon GuardDuty pour les journaux d'audit Kubernetes et la surveillance du temps d'exécution, vous pouvez exécuter la commande suivante :

```
aws guardduty update-detector \
    --detector-id 12abc34d567e8fa901bc2d34e56789f0 \
```

```
    --data-sources '{
        "Kubernetes": {
            "AuditLogs": {"Enable": true},
            "RuntimeMonitoring": {"Enable": true}
        }
    }'
```

> **Important**
>
> GuardDuty La surveillance du temps d'exécution est automatiquement prise en charge dans les clusters en mode automatique EKS, offrant une visibilité de sécurité améliorée sans configuration supplémentaire au niveau du nœud.

GuardDuty Intégration des résultats

GuardDuty les résultats peuvent être intégrés à d'autres services AWS pour une réponse automatique :

- EventBridge Règles :

```
{
  "source": ["aws.guardduty"],
  "detail-type": ["GuardDuty Finding"],
  "detail": {
    "type": ["Runtime:Container/*", "Runtime:Kubernetes/*"],
    "severity": [4, 5, 6, 7, 8]
  }
}
```

- Intégration au Security Hub :

```
# Enable Security Hub integration
aws securityhub enable-security-hub \
    --enable-default-standards \
    --tags '{"Environment":"Production"}' \
    --region us-west-2
```

Bonnes pratiques pour le GuardDuty mode automatique d'EKS

1. Activez tous les types de recherche :
   - Activez à la fois la surveillance du journal d'audit Kubernetes et la surveillance de l'exécution
   - Configurer les résultats pour tous les niveaux de gravité
2. Mettre en œuvre une réponse automatisée :
   - Créez des EventBridge règles pour les résultats très graves
   - Intégrez AWS Security Hub pour une gestion centralisée de la sécurité
   - Configurez des actions de correction automatisées le cas échéant
3. Révision et réglage réguliers :
   - Examiner régulièrement GuardDuty les résultats
   - Ajustez les seuils de détection en fonction de votre environnement
   - Mettre à jour les procédures de réponse en fonction des nouveaux types de résultats
4. Gestion entre comptes :
   - Envisagez d'utiliser un compte GuardDuty administrateur pour une gestion centralisée
   - Permettre l'agrégation des résultats sur plusieurs comptes

> **Warning**
>
> Tout en GuardDuty fournissant une surveillance complète de la sécurité, elle doit faire partie d'une defense-in-depth stratégie incluant d'autres contrôles de sécurité tels que les politiques réseau, les normes de sécurité des pods et une configuration RBAC appropriée.

Questions fréquentes (FAQ)

Q : En quoi le mode automatique d'EKS diffère-t-il du mode EKS standard en termes de sécurité ?
R : Le mode automatique d'EKS améliore la sécurité grâce aux instances gérées EC2, à l'application automatique de correctifs, à la rotation obligatoire des nœuds et aux contrôles de sécurité intégrés. Cela réduit les frais opérationnels tout en maintenant une solide posture de sécurité en permettant à AWS de gérer une plus grande partie des aspects liés à la sécurité.

Q : Puis-je toujours utiliser les outils et politiques de sécurité existants avec le mode automatique d'EKS ? R : Oui, le mode automatique d'EKS est compatible avec la plupart des outils et politiques de

sécurité existants. Cependant, certains outils de sécurité au niveau des nœuds peuvent nécessiter une adaptation en raison de la nature gérée des nœuds EKS Auto Mode.

Q : Comment déployer des agents de sécurité et des outils de surveillance en mode automatique EKS ? R : En mode automatique d'EKS, les agents de sécurité et les outils de surveillance doivent être déployés sous forme de charges de travail Kubernetes (généralement DaemonSets, une instance du Pod est déployée par défaut sur chaque nœud) plutôt que d'être installés directement sur le système d'exploitation du nœud. Cette approche s'aligne sur le modèle d'infrastructure immuable d'EKS Auto Mode. Exemple :

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: security-agent
  namespace: security
spec:
  selector:
    matchLabels:
      app: security-agent
  template:
    metadata:
      labels:
        app: security-agent
    spec:
      containers:
      - name: security-agent
        image: security-vendor/agent:latest
        securityContext:
          privileged: false
          # Use specific capabilities instead of privileged mode
          capabilities:
            add: ["NET_ADMIN", "SYS_ADMIN"]
```

Q : Les solutions de sécurité tierces sont-elles compatibles avec le mode automatique EKS ? R : De nombreuses solutions de sécurité tierces populaires ont été mises à jour pour prendre en charge le mode automatique EKS, mais il est toujours recommandé de vérifier les exigences spécifiques en matière de version et de déploiement auprès de votre fournisseur de sécurité, car la prise en charge du mode automatique EKS peut nécessiter des versions mises à jour ou des configurations de déploiement spécifiques.

Q : Quelles sont les limites imposées aux agents de sécurité en mode automatique d'EKS ? R : Les principales limites sont les suivantes :

- Aucun accès direct pour modifier le système d'exploitation du nœud
- Aucune persistance entre les rotations de nœuds
- Doit être compatible avec le déploiement basé sur des conteneurs
- Nécessité de respecter l'immuabilité des nœuds
- Peut nécessiter différentes configurations de privilèges
- Toute modification persistante des nœuds doit être effectuée par le biais `NodePools` des `NodeClasses` ressources.

> **Note**
>
> Bien que le mode automatique d'EKS puisse nécessiter des ajustements de votre stratégie de déploiement d'outils de sécurité, ces modifications se traduisent souvent par des configurations plus faciles à maintenir et plus sécurisées, conformes aux meilleures pratiques natives du cloud. Le mode automatique d'EKS prévoit de prendre complètement en charge la plupart des fonctionnalités qu'il gère. Par conséquent, toutes les modifications manuelles que vous apportez à ces fonctionnalités, si vous pouvez y accéder, peuvent être annulées ou annulées par le mode automatique d'EKS.

Q : Puis-je utiliser le mode personnalisé AMIs avec le mode automatique EKS ? R : Pour le moment, le mode automatique d'EKS ne prend pas en charge la personnalisation AMIs. Cela est intentionnel, car AWS gère la sécurité, les correctifs et la maintenance des nœuds dans le cadre du modèle de responsabilité partagée. Les nœuds EKS Auto Mode utilisent une variante spécialisée de Bottlerocket optimisée et gérée par AWS.

Q : À quelle fréquence les nœuds sont-ils automatiquement pivotés en mode automatique EKS ? R : Les nœuds en mode automatique EKS ont une durée de vie maximale de 21 jours. Ils seront automatiquement remplacés avant cette limite, garantissant ainsi des mises à jour de sécurité régulières et l'application de correctifs.

Q : Puis-je me connecter par SSH aux nœuds du mode automatique d'EKS à des fins de dépannage ? R : Non, l'accès SSH direct n'est pas disponible en mode automatique d'EKS. Vous pouvez plutôt utiliser la définition de ressource NodeDiagnostic personnalisée (CRD) pour collecter les journaux du système et les informations de débogage.

Q : La prise en charge des politiques réseau est-elle activée par défaut en mode automatique d'EKS ? R : Pour le moment, la prise en charge de la politique réseau doit être explicitement activée via la configuration du module complémentaire VPC CNI. Une fois activée, vous pouvez utiliser les politiques réseau Kubernetes standard.

Q : Dois-je utiliser IRSA ou Pod Identity avec le mode automatique d'EKS ? R : Bien que les deux soient compatibles, Pod Identity est l'approche recommandée en mode automatique d'EKS, car elle inclut déjà le module complémentaire de l'agent Pod Identity Security et fournit des fonctionnalités de sécurité améliorées et une gestion simplifiée.

Q : Puis-je toujours utiliser l'aws-auth ConfigMap en mode automatique EKS ? R : `aws-auth` ConfigMap Il s'agit d'une fonctionnalité obsolète. Il est recommandé d'utiliser l'approche par défaut de l'authentification basée sur les API pour améliorer la sécurité et simplifier la gestion des accès.

Q : Comment puis-je surveiller les événements de sécurité en mode automatique EKS ? R : Le mode automatique EKS s'intègre à plusieurs solutions de surveillance GuardDuty, notamment CloudWatch, et CloudTrail. GuardDuty fournit une surveillance améliorée de la sécurité d'exécution spécifiquement pour les charges de travail EKS.

Q : Comment puis-je collecter les journaux à partir des nœuds du mode automatique EKS ? R : Utilisez le NodeDiagnostic CRD, qui télécharge automatiquement les journaux dans un compartiment S3. Vous pouvez également utiliser CloudWatch Container Insights et AWS Distro pour OpenTelemetry.

> **Note**
>
> Cette section FAQ est régulièrement mise à jour au fur et à mesure que de nouvelles fonctionnalités sont ajoutées au mode automatique d'EKS et que nous recevons des questions fréquentes de la part des clients.

Gestion de l’identité et des accès

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Identity and Access Management (IAM) est un service AWS qui exécute deux fonctions essentielles : l'authentification et l'autorisation. L'authentification implique la vérification d'une identité, tandis que l'autorisation régit les actions qui peuvent être effectuées par les ressources AWS. Au sein d'AWS, une ressource peut être un autre service AWS, par exemple EC2, ou un principal AWS tel qu'un utilisateur ou un rôle IAM. Les règles régissant les actions qu'une ressource est autorisée à effectuer sont exprimées sous forme de politiques IAM.

Contrôle de l'accès aux clusters EKS

Le projet Kubernetes prend en charge différentes stratégies pour authentifier les demandes adressées au service kube-apiserver, par exemple les Bearer Tokens, les certificats X.509, OIDC, etc. EKS prend actuellement en charge de manière native l'authentification par jeton Webhook, les jetons de compte de service et, depuis le 21 février 2021, l'authentification OIDC.

La stratégie d'authentification du webhook fait appel à un webhook qui vérifie les jetons porteurs. Sur EKS, ces jetons porteurs sont générés par la CLI AWS ou par le aws-iam-authenticatorclient lorsque vous exécutez des `kubectl` commandes. Lorsque vous exécutez des commandes, le jeton est transmis au serveur kube-apiserver qui le transmet au webhook d'authentification. Si la demande est bien formée, le webhook appelle une URL pré-signée intégrée dans le corps du jeton. Cette URL valide la signature de la demande et renvoie des informations sur l'utilisateur, par exemple le compte de l'utilisateur, Arn, et UserId au kube-apiserver.

Pour générer manuellement un jeton d'authentification, tapez la commande suivante dans une fenêtre de terminal :

```
aws eks get-token --cluster-name <cluster_name> --region <region>
```

La sortie doit ressembler à ceci :

```
{
    "kind": "ExecCredential",
    "apiVersion": "client.authentication.k8s.io/v1alpha1",
    "spec": {},
    "status": {
        "expirationTimestamp": "2024-12-20T17:38:48Z",
        "token": "k8s-aws-
v1.aHR0cHM6Ly9zdHMudXMtd2VzdC0yLmFtYXpvbmF3cy5jb20vP0FjdGlvbj1HZ...."
    }
}
```

Vous pouvez également obtenir un jeton par programmation. Voici un exemple écrit en Go :

```
package main

import (
  "fmt"
  "log"
  "sigs.k8s.io/aws-iam-authenticator/pkg/token"
)

func main()  {
  g, _ := token.NewGenerator(false, false)
  tk, err := g.Get("<cluster_name>")
  if err != nil {
    log.Fatal(err)
  }
  fmt.Println(tk)
}
```

La sortie doit ressembler à ceci :


```
{
  "kind": "ExecCredential",
  "apiVersion": "client.authentication.k8s.io/v1alpha1",
  "spec": {},
  "status": {
    "expirationTimestamp": "2020-02-19T16:08:27Z",
    "token": "k8s-aws-
v1.aHR0cHM6Ly9zdHMuYW1hem9uYXdzLmNvbS8_QWN0aW9uPUdldENhbGxlcklkZW50aXR5JlZlcnNpb249MjAxMS0wNi0x
  }
}
```


Chaque jeton commence par `k8s-aws-v1.` suivi d'une chaîne codée en base64. La chaîne, une fois décodée, doit ressembler à ceci :

```
https://sts.amazonaws.com/?Action=GetCallerIdentity&Version=2011-06-15&X-
Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=XXXXJPFRILKNSRC2W5QA
%2F20200219%2Fus-xxxx-1%2Fsts%2Faws4_request&X-Amz-Date=20200219T155427Z&X-
Amz-Expires=60&X-Amz-SignedHeaders=host%3Bx-k8s-aws-id&X-Amz-
Signature=XXXf8f3285e320ddb5e683a5c9a405301ad76546f24f28111fdad09cf648a393
```

Le jeton consiste en une URL pré-signée qui inclut un identifiant et une signature Amazon. Pour plus de détails, voir https://docs.aws.amazon.com/STS/ latest/APIReference/API _ GetCallerIdentity .html.

Le jeton a une durée de vie (TTL) de 15 minutes, après quoi un nouveau jeton devra être généré. Cela est géré automatiquement lorsque vous utilisez un client, par exemple`kubectl`, si vous utilisez le tableau de bord Kubernetes, vous devrez générer un nouveau jeton et vous authentifier à chaque fois que le jeton expire.

Une fois que l'identité de l'utilisateur a été authentifiée par le service AWS IAM, le kube-apiserver la lit `aws-auth` ConfigMap dans l'`kube-system`espace de noms pour déterminer le groupe RBAC à associer à l'utilisateur. `aws-auth` ConfigMap Il est utilisé pour créer un mappage statique entre les principaux IAM, c'est-à-dire les utilisateurs et les rôles IAM, et les groupes RBAC Kubernetes. Les groupes RBAC peuvent être référencés dans RoleBindings Kubernetes ou. ClusterRoleBindings Ils sont similaires aux rôles IAM dans la mesure où ils définissent un ensemble d'actions (verbes) qui peuvent être effectuées sur un ensemble de ressources Kubernetes (objets).

CloudWatch requête pour aider les utilisateurs à identifier les clients envoyant des demandes au point de terminaison STS global

Exécutez CloudWatch la requête ci-dessous pour obtenir le point de terminaison sts. Si stsendpoint est égal à « sts.amazonaws.com », il s'agit d'un point de terminaison STS global. Si stsendpoint est égal à « sts ». <region>.amazonaws.com », il s'agit alors d'un point de terminaison STS régional.

```
fields @timestamp, @message, @logStream, @log,stsendpoint
| filter @logStream like /authenticator/
| filter @message like /stsendpoint/
| sort @timestamp desc
| limit 10000
```

Gestionnaire d'accès au cluster

Cluster Access Manager, désormais la méthode préférée pour gérer l'accès des principaux AWS IAM aux clusters Amazon EKS, est une fonctionnalité de l'API AWS et une fonctionnalité optionnelle pour les clusters EKS v1.23 et versions ultérieures (nouveaux ou existants). Il simplifie le mappage des identités entre AWS IAM et KubernetesRBACs, éliminant ainsi le besoin de passer d'AWS à Kubernetes APIs ou de modifier le code `aws-auth` ConfigMap pour la gestion des accès, réduisant ainsi les frais opérationnels et aidant à corriger les erreurs de configuration. L'outil permet également aux administrateurs de cluster de révoquer ou d'affiner `cluster-admin` les autorisations accordées automatiquement au principal AWS IAM utilisé pour créer le cluster.

Cette API repose sur deux concepts :

- Entrées d'accès : une identité de cluster directement liée à un AWS IAM principal (utilisateur ou rôle) autorisé à s'authentifier auprès d'un cluster Amazon EKS.
- Politiques d'accès : les politiques spécifiques à Amazon EKS autorisent-elles une entrée d'accès à effectuer des actions dans le cluster Amazon EKS.

Au lancement, Amazon EKS prend uniquement en charge les politiques prédéfinies et gérées par AWS. Les politiques d'accès ne sont pas des entités IAM et sont définies et gérées par Amazon EKS.

Cluster Access Manager permet de combiner le RBAC en amont avec des politiques d'accès permettant d'autoriser et de transmettre (mais pas de refuser) les décisions de Kubernetes AuthZ concernant les demandes du serveur d'API. Une décision de refus sera prise lorsque le RBAC en amont et les autorisateurs Amazon EKS ne peuvent pas déterminer le résultat de l'évaluation d'une demande.

Grâce à cette fonctionnalité, Amazon EKS prend en charge trois modes d'authentification :

1. `CONFIG_MAP`pour continuer à utiliser `aws-auth` ConfigMap exclusivement.
2. `API_AND_CONFIG_MAP`pour obtenir des principes IAM authentifiés à la fois à partir d'EKS Access Entry et de `aws-auth` ConfigMap, en hiérarchisant APIs les entrées d'accès. Idéal pour migrer les `aws-auth` autorisations existantes vers Access Entries.
3. `API`de s'appuyer exclusivement sur EKS Access Entry APIs. Il s'agit de la nouvelle approche recommandée.

Pour commencer, les administrateurs de clusters peuvent créer ou mettre à jour des clusters Amazon EKS, en définissant l'authentification `API_AND_CONFIG_MAP` ou la `API` méthode d'authentification préférée et en définissant les entrées d'accès pour accorder l'accès aux principaux AWS IAM souhaités.

```
$ aws eks create-cluster \
    --name <CLUSTER_NAME> \
    --role-arn <CLUSTER_ROLE_ARN> \
    --resources-vpc-config
 subnetIds=<value>,endpointPublicAccess=true,endpointPrivateAccess=true \
    --logging '{"clusterLogging":[{"types":
["api","audit","authenticator","controllerManager","scheduler"],"enabled":true}]}' \
```

```
    --access-config
 authenticationMode=API_AND_CONFIG_MAP,bootstrapClusterCreatorAdminPermissions=false
```

La commande ci-dessus est un exemple de création d'un cluster Amazon EKS déjà sans les autorisations d'administrateur du créateur du cluster.

Il est possible de mettre à jour la configuration des clusters Amazon EKS pour activer `API` AuthenticationMode à l'aide de la `update-cluster-config` commande. Pour ce faire, sur les clusters existants, `CONFIG_MAP` vous devrez d'abord effectuer une mise à jour vers, `API_AND_CONFIG_MAP` puis vers. `API` Ces opérations ne peuvent pas être annulées, ce qui signifie qu'il n'est pas possible de `API` passer de `API_AND_CONFIG_MAP` ou `CONFIG_MAP` `API_AND_CONFIG_MAP` à`CONFIG_MAP`.

```
$ aws eks update-cluster-config \
    --name <CLUSTER_NAME> \
    --access-config authenticationMode=API
```

L'API prend en charge les commandes permettant d'ajouter et de révoquer l'accès au cluster, ainsi que de valider les politiques d'accès et les entrées d'accès existantes pour le cluster spécifié. Les politiques par défaut sont créées pour correspondre à Kubernetes comme suit. RBACs

Politique d'accès EKS	RBAC Kubernetes
Amazon EKSCluster AdminPolicy	administrateur du cluster
EKSAdminPolitique d'Amazon	admin
EKSEditPolitique d'Amazon	modifier
EKSViewPolitique d'Amazon	afficher

```
$ aws eks list-access-policies
{
    "accessPolicies": [
        {
            "name": "AmazonEKSAdminPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy"
        },
        {
```

```
            "name": "AmazonEKSClusterAdminPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy"
        },
        {
            "name": "AmazonEKSEditPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSEditPolicy"
        },
        {
            "name": "AmazonEKSViewPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSViewPolicy"
        }
    ]
}

$ aws eks list-access-entries --cluster-name <CLUSTER_NAME>

{
    "accessEntries": []
}
```

Aucune entrée d'accès n'est disponible lorsque le cluster est créé sans l'autorisation d'administrateur du créateur du cluster, qui est la seule entrée créée par défaut.

Le **`aws-auth`** ConfigMap (obsolète)

L'intégration de Kubernetes à l'authentification AWS peut être effectuée via le `aws-auth` ConfigMap, qui réside dans l'espace de noms. `kube-system` Il est chargé de mapper l'authentification des identités AWS IAM (utilisateurs, groupes et rôles) à l'autorisation de contrôle d'accès basé sur les rôles (RBAC) de Kubernetes. `aws-auth` ConfigMap Il est automatiquement créé dans votre cluster Amazon EKS lors de sa phase de provisionnement. Il a été initialement créé pour permettre aux nœuds de rejoindre votre cluster, mais comme indiqué, vous pouvez également l'utiliser ConfigMap pour ajouter RBACs un accès aux principaux IAM.

Pour vérifier celui de votre cluster `aws-auth` ConfigMap, vous pouvez utiliser la commande suivante.

```
kubectl -n kube-system get configmap aws-auth -o yaml
```

Il s'agit d'un exemple de configuration par défaut du `aws-auth`ConfigMap.

```
apiVersion: v1
data:
  mapRoles: |
```

```
    - groups:
      - system:bootstrappers
      - system:nodes
      - system:node-proxier
      rolearn: arn:aws:iam::<AWS_ACCOUNT_ID>:role/kube-system-<SELF_GENERATED_UUID>
      username: system:node:{{SessionName}}
kind: ConfigMap
metadata:
  creationTimestamp: "2023-10-22T18:19:30Z"
  name: aws-auth
  namespace: kube-system
```

La session principale se trouve `data` en dessous du `mapRoles` bloc, qui est essentiellement composé de 3 paramètres. ConfigMap

- groupes : les Kubernetes auxquels group/groups mapper le rôle IAM. Il peut s'agir d'un groupe par défaut ou d'un groupe personnalisé spécifié dans un `clusterrolebinding` ou`rolebinding`. Dans l'exemple ci-dessus, seuls les groupes de systèmes sont déclarés.
- rolearn : L'ARN du rôle AWS IAM doit être mappé au groupe/aux groupes Kubernetes à ajouter, en utilisant le format suivant. `arn:<PARTITION>:iam::<AWS_ACCOUNT_ID>:role/role-name`
- nom d'utilisateur : nom d'utilisateur dans Kubernetes à associer au rôle AWS IAM. Il peut s'agir de n'importe quel nom personnalisé.

Il est également possible de mapper les autorisations pour les utilisateurs d'AWS IAM, en définissant un nouveau bloc de configuration pour`mapUsers`, sous le `aws-auth`ConfigMap, `data` en remplaçant le paramètre rolearn pour userarn. Toutefois, en tant que bonne pratique, il est toujours recommandé d'utiliser plutôt l'utilisateur. `mapRoles`

Pour gérer les autorisations, vous pouvez modifier l'`aws-auth` ConfigMap ajout ou la suppression de l'accès à votre cluster Amazon EKS. Bien qu'il soit possible de le modifier `aws-auth` ConfigMap manuellement, il est recommandé d'utiliser des outils tels que`eksctl`, car il s'agit d'une configuration très sensible et une configuration inexacte peut vous empêcher de sortir de votre cluster Amazon EKS. Consultez la sous-section Utiliser des outils pour apporter des modifications à l'aws-auth ConfigMap ci-dessous pour plus de détails.

Avantages par rapport ConfigMap à la gestion des accès basée

1. Réduction des risques de mauvaise configuration : la gestion directe basée sur l'API élimine les erreurs courantes associées à l'édition manuelle. ConfigMap Cela permet d'éviter les suppressions

accidentelles ou les erreurs de syntaxe susceptibles d'empêcher les utilisateurs d'accéder au cluster.

2. Principe du moindre privilège amélioré : élimine le besoin d'une autorisation d'administrateur de cluster pour l'identité du créateur du cluster et permet une attribution des autorisations plus précise et plus appropriée. Vous pouvez choisir d'ajouter cette autorisation pour les cas d'utilisation de break-glass.
3. Modèle de sécurité amélioré : fournit une validation intégrée des entrées d'accès avant leur application. En outre, offre une intégration plus étroite avec AWS IAM pour l'authentification.
4. Opérations rationalisées : offre un moyen plus intuitif de gérer les autorisations grâce à des outils natifs d'AWS.

Recommandations relatives à l'accès aux clusters

Combinez IAM Identity Center avec l'API CAM

- Gestion simplifiée : en utilisant l'API de gestion des accès aux clusters conjointement avec IAM Identity Center, les administrateurs peuvent gérer l'accès au cluster EKS parallèlement aux autres services AWS, ce qui réduit le besoin de passer d'une interface à l'autre ou de procéder à des modifications ConfigMaps manuelles.
- Utilisez les entrées d’accès pour gérer les autorisations Kubernetes des principaux IAM depuis l’extérieur du cluster. Vous pouvez ajouter et gérer l'accès au cluster à l'aide de l'API EKS, de l'interface de ligne de commande AWS, d'AWS SDKs, d'AWS CloudFormation et de la console de gestion AWS. Cela signifie que vous pouvez gérer les utilisateurs avec les mêmes outils que ceux avec lesquels vous avez créé le cluster.
- Tirez parti de l'automatisation, comme illustré dans cet exemple, pour déployer des clusters avec AWS IAM Identity Center comme IdP et l'API CAM comme point d'entrée.
- Les autorisations Kubernetes granulaires peuvent être appliquées en mappant les utilisateurs ou les groupes Kubernetes aux principaux IAM associés aux identités SSO via des entrées d'accès et des politiques d'accès.
- Pour commencer, suivez Changer le mode d'authentification pour utiliser les entrées d'accès, puis Migration des entrées aws-auth existantes pour accéder aux ConfigMap entrées.

Rendre le point de terminaison du cluster EKS privé

Par défaut, lorsque vous provisionnez un cluster EKS, le point de terminaison du cluster d'API est défini sur public, c'est-à-dire qu'il est accessible depuis Internet. Bien qu'il soit accessible depuis Internet, le point de terminaison est toujours considéré comme sécurisé car il nécessite que toutes les demandes d'API soient authentifiées par IAM puis autorisées par Kubernetes RBAC. Cela dit, si la politique de sécurité de votre entreprise impose de restreindre l'accès à l'API depuis Internet ou vous empêche de router le trafic en dehors du VPC du cluster, vous pouvez :

- Configurez le point de terminaison du cluster EKS pour qu'il soit privé. Voir Modification de l'accès aux points de terminaison du cluster pour plus d'informations à ce sujet.
- Laissez le point de terminaison du cluster public et spécifiez quels blocs CIDR peuvent communiquer avec le point de terminaison du cluster. Les blocs sont en fait un ensemble d'adresses IP publiques sur liste blanche autorisées à accéder au point de terminaison du cluster.
- Configurez l'accès public avec un ensemble de blocs CIDR sur liste blanche et définissez l'accès aux terminaux privés sur Activé. Cela permettra un accès public à partir d'un éventail spécifique de publics IPs tout en forçant tout le trafic réseau entre les kubelets (travailleurs) et l'API Kubernetes via les comptes croisés ENIs qui sont provisionnés dans le VPC du cluster lorsque le plan de contrôle est provisionné.

N'utilisez pas de jeton de compte de service pour l'authentification

Un jeton de compte de service est un identifiant statique de longue durée. S'il est compromis, perdu ou volé, un attaquant peut être en mesure d'effectuer toutes les actions associées à ce jeton jusqu'à ce que le compte de service soit supprimé. Parfois, vous devrez peut-être accorder une exception pour les applications qui doivent utiliser l'API Kubernetes depuis l'extérieur du cluster, par exemple une application de pipeline CI/CD. Si de telles applications s'exécutent sur une infrastructure AWS, comme les instances EC2, envisagez d'utiliser un profil d'instance et de le mapper à un rôle RBAC Kubernetes.

Utilisez l'accès le moins privilégié aux ressources AWS

Il n'est pas nécessaire d'attribuer des privilèges aux ressources AWS pour accéder à l'API Kubernetes. Si vous devez accorder à un utilisateur IAM l'accès à un cluster EKS, créez une entrée dans le `aws-auth` ConfigMap pour cet utilisateur qui correspond à un groupe Kubernetes RBAC spécifique.

Supprimer les autorisations d'administrateur du cluster auprès du créateur principal du cluster

Par défaut, les clusters Amazon EKS sont créés avec une `cluster-admin` autorisation permanente liée au créateur principal du cluster. Avec l'API Cluster Access Manager, il est possible de créer des clusters sans que cette autorisation ne définisse le mode `--access-config bootstrapClusterCreatorAdminPermissions` to`false`, lors de l'utilisation `API_AND_CONFIG_MAP` ou le mode `API` d'authentification. Révoquer cet accès est considéré comme une bonne pratique pour éviter toute modification indésirable de la configuration du cluster. Le processus de révocation de cet accès suit le même processus pour révoquer tout autre accès au cluster.

L'API vous donne la flexibilité de dissocier uniquement un principal IAM d'une politique d'accès, dans ce cas le. `AmazonEKSClusterAdminPolicy`

```
$ aws eks list-associated-access-policies \
    --cluster-name <CLUSTER_NAME> \
    --principal-arn <IAM_PRINCIPAL_ARN>

$ aws eks disassociate-access-policy --cluster-name <CLUSTER_NAME> \
    --principal-arn <IAM_PRINCIPAL_ARN. \
    --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy
```

Ou en supprimant complètement l'entrée d'accès associée à l'`cluster-admin`autorisation.

```
$ aws eks list-access-entries --cluster-name <CLUSTER_NAME>

{
    "accessEntries": []
}

$ aws eks delete-access-entry --cluster-name <CLUSTER_NAME> \
  --principal-arn <IAM_PRINCIPAL_ARN>
```

Cet accès peut être accordé à nouveau si nécessaire lors d'un incident, d'une urgence ou d'un scénario de rupture de vitre dans lequel le cluster est autrement inaccessible.

Si le cluster `aws-auth` ConfigMap est toujours configuré avec la méthode `CONFIG_MAP` d'authentification, tous les utilisateurs supplémentaires doivent avoir accès au cluster par le biais du `aws-auth` ConfigMap, et une fois configuré, le rôle attribué à l'entité qui a créé le cluster peut être

supprimé et recréé uniquement en cas d'incident, d'urgence ou de bris de vitre, ou lorsque le cluster `aws-auth` ConfigMap est endommagé et que le cluster est inaccessible pour une autre raison. Cela peut être particulièrement utile dans les clusters de production.

Utiliser les rôles IAM lorsque plusieurs utilisateurs ont besoin d'un accès identique au cluster

Plutôt que de créer une entrée pour chaque utilisateur IAM individuel, autorisez ces utilisateurs à assumer un rôle IAM et à associer ce rôle à un groupe Kubernetes RBAC. Cela sera plus facile à gérer, d'autant plus que le nombre d'utilisateurs ayant besoin d'un accès augmente.

> **Important**
>
> Lorsque vous accédez au cluster EKS avec l'entité IAM mappée par `aws-auth` ConfigMap, le nom d'utilisateur décrit est enregistré dans le champ utilisateur du journal d'audit Kubernetes. Si vous utilisez un rôle IAM, les utilisateurs réels qui assument ce rôle ne sont pas enregistrés et ne peuvent pas être audités.

Si vous utilisez toujours le `aws-auth` ConfigMap comme méthode d'authentification, lorsque vous attribuez des autorisations RBAC K8s à un rôle IAM, vous devez inclure \ {{}} dans votre nom d'utilisateur. SessionName Ainsi, le journal d'audit enregistrera le nom de la session afin que vous puissiez savoir qui est l'utilisateur réel qui assume ce rôle avec le CloudTrail journal.

```
- rolearn: arn:aws:iam::XXXXXXXXXXXX:role/testRole
  username: testRole:{{SessionName}}
  groups:
    - system:masters
```

> Dans Kubernetes 1.20 et versions ultérieures, cette modification n'est plus requise, car elle a `user.extra.sessionName.0` été ajoutée au journal d'audit de Kubernetes.

Utilisez l'accès le moins privilégié lors de la création RoleBindings et ClusterRoleBindings

Comme le point précédent concernant l'octroi de l'accès aux ressources AWS, RoleBindings ClusterRoleBindings il ne devrait inclure que l'ensemble des autorisations nécessaires pour exécuter une fonction spécifique. Évitez de l'utiliser `["*"]` dans vos rôles et ClusterRoles sauf si cela est absolument nécessaire. Si vous ne savez pas quelles autorisations attribuer, pensez à utiliser un outil

tel que audit2rbac pour générer automatiquement des rôles et des liaisons en fonction des appels d'API observés dans le journal d'audit Kubernetes.

Création d'un cluster à l'aide d'un processus automatisé

Comme indiqué dans les étapes précédentes, lors de la création d'un cluster Amazon EKS, si vous n'utilisez pas le mode `API` d'utilisation `API_AND_CONFIG_MAP` ou d'authentification, et si vous ne choisissez pas de déléguer des `cluster-admin` autorisations au créateur du cluster, l'utilisateur ou le rôle de l'entité IAM, tel qu'un utilisateur fédéré qui crée le cluster, reçoit automatiquement des `system:masters` autorisations dans la configuration RBAC du cluster. Même s'il s'agit d'une bonne pratique consistant à supprimer cette autorisation, comme décrit ici si vous utilisez la méthode `CONFIG_MAP` d'authentification `aws-auth` ConfigMap, cet accès ne peut pas être révoqué. Il est donc judicieux de créer le cluster avec un pipeline d'automatisation de l'infrastructure lié à un rôle IAM dédié, sans aucune autorisation à assumer par d'autres utilisateurs ou entités, et de vérifier régulièrement les autorisations et les politiques de ce rôle, ainsi que les personnes habilitées à déclencher le pipeline. De plus, ce rôle ne doit pas être utilisé pour effectuer des actions de routine sur le cluster, et être exclusivement utilisé pour des actions au niveau du cluster déclenchées par le pipeline, via des modifications de code SCM par exemple.

Créez le cluster avec un rôle IAM dédié

Lorsque vous créez un cluster Amazon EKS, l'utilisateur ou le rôle de l'entité IAM, tel qu'un utilisateur fédéré qui crée le cluster, reçoit automatiquement des `system:masters` autorisations dans la configuration RBAC du cluster. Cet accès ne peut pas être supprimé et n'est pas géré via le `aws-auth` ConfigMap. Il est donc judicieux de créer le cluster avec un rôle IAM dédié et de vérifier régulièrement qui peut assumer ce rôle. Ce rôle ne doit pas être utilisé pour effectuer des actions de routine sur le cluster. À cette fin, des utilisateurs supplémentaires doivent avoir accès au cluster via le `aws-auth` ConfigMap . Une fois `aws-auth` ConfigMap configuré, le rôle doit être sécurisé et utilisé uniquement en mode temporaire à privilèges élevés/break glass pour les scénarios dans lesquels le cluster est autrement inaccessible. Cela peut être particulièrement utile dans les clusters pour lesquels l'accès direct des utilisateurs n'est pas configuré.

Auditez régulièrement l'accès au cluster

Les personnes qui ont besoin d'un accès sont susceptibles de changer au fil du temps. Prévoyez de vérifier régulièrement `aws-auth` ConfigMap les personnes qui ont obtenu l'accès et les droits qui leur ont été attribués. Vous pouvez également utiliser des outils open source tels kubectl-who-canque rbac-lookup pour examiner les rôles liés à un compte de service, un utilisateur ou un groupe

en particulier. Nous explorerons ce sujet plus en détail lorsque nous arriverons à la section sur l'audit. D'autres idées peuvent être trouvées dans cet article de NCC Group.

Si vous vous fiez à **`aws-auth`** ConfigMap, utilisez des outils pour apporter des modifications

Un aws-auth mal formaté ConfigMap peut vous faire perdre l'accès au cluster. Si vous devez apporter des modifications au ConfigMap, utilisez un outil.

eksctl La `eksctl` CLI inclut une commande pour ajouter des mappages d'identité à l'aws-auth. ConfigMap

Afficher l'aide de la CLI :

```
$ eksctl create iamidentitymapping --help
...
```

Vérifiez les identités mappées à votre cluster Amazon EKS.

```
$ eksctl get iamidentitymapping --cluster $CLUSTER_NAME --region $AWS_REGION
ARN                                                                     USERNAME
               GROUPS                                                    ACCOUNT
arn:aws:iam::788355785855:role/kube-system-<SELF_GENERATED_UUID>      system:node:
{{SessionName}}     system:bootstrappers,system:nodes,system:node-proxier
```

Définissez un rôle IAM en tant qu'administrateur de cluster :

```
$ eksctl create iamidentitymapping --cluster  <CLUSTER_NAME> --region=<region> --arn
 arn:aws:iam::123456:role/testing --group system:masters --username admin
...
```

Pour plus d'informations, consultez les `eksctl`documents

aws-auth par keikoproj

`aws-auth`by keikoproj inclut à la fois une bibliothèque cli et une bibliothèque go.

Téléchargez et consultez l'aide de la CLI d'aide :

```
$ go get github.com/keikoproj/aws-auth
...
```

```
$ aws-auth help
...
```

Vous pouvez également l'installer `aws-auth` avec le gestionnaire de plugins Krew pour kubectl.

```
$ kubectl krew install aws-auth
...
$ kubectl aws-auth
...
```

Consultez la documentation aws-auth GitHub pour plus d'informations, y compris la bibliothèque go.

CLI d'authentification AWS IAM

Le `aws-iam-authenticator` projet inclut une CLI pour mettre à jour leConfigMap.

Téléchargez un communiqué sur GitHub.

Ajoutez des autorisations de cluster à un rôle IAM :

```
$ ./aws-iam-authenticator add role --rolearn arn:aws:iam::185309785115:role/lil-dev-
role-cluster --username lil-dev-user --groups system:masters --kubeconfig ~/.kube/
config
...
```

Autres approches en matière d'authentification et de gestion des accès

Bien que l'IAM soit le moyen préféré pour authentifier les utilisateurs qui ont besoin d'accéder à un cluster EKS, il est possible d'utiliser un fournisseur d'identité OIDC, par exemple en GitHub utilisant un proxy d'authentification et en se faisant passer pour Kubernetes. Des articles relatifs à deux de ces solutions ont été publiés sur le blog Open Source d'AWS :

- Authentification auprès d'EKS à l'aide des GitHub informations d'identification avec téléportation
- Authentification OIDC cohérente sur plusieurs clusters EKS à l'aide de kube-oidc-proxy

> **Important**
>
> EKS prend en charge nativement l'authentification OIDC sans utiliser de proxy. Pour plus d'informations, consultez le blog de lancement, Présentation de l'authentification du fournisseur d'identité OIDC pour Amazon EKS. Pour un exemple montrant comment

configurer EKS avec Dex, un fournisseur OIDC open source populaire doté de connecteurs pour différentes méthodes d'authentification, consultez Utiliser l'authentificateur Dex & dex-k8s-k8s-authenticator pour vous authentifier auprès d'Amazon EKS. Comme décrit dans les blogs, les username/group utilisateurs authentifiés par un fournisseur OIDC apparaîtront dans le journal d'audit Kubernetes.

Vous pouvez également utiliser AWS SSO pour fédérer AWS avec un fournisseur d'identité externe, par exemple Azure AD. Si vous décidez de l'utiliser, la version 2.0 de l'interface de ligne de commande AWS inclut une option permettant de créer un profil nommé qui permet d'associer facilement une session SSO à votre session d'interface de ligne de commande actuelle et d'assumer un rôle IAM. Sachez que vous devez assumer un rôle avant de lancer l'exécution, `kubectl` car le rôle IAM est utilisé pour déterminer le groupe Kubernetes RBAC de l'utilisateur.

Identités et informations d'identification pour les pods EKS

Certaines applications qui s'exécutent au sein d'un cluster Kubernetes doivent être autorisées à appeler l'API Kubernetes pour fonctionner correctement. Par exemple, le contrôleur AWS Load Balancer doit être en mesure de répertorier les points de terminaison d'un service. Le contrôleur doit également être en mesure d'invoquer AWS APIs pour approvisionner et configurer un ALB. Dans cette section, nous explorerons les meilleures pratiques en matière d'attribution de droits et de privilèges aux Pods.

Comptes de service Kubernetes

Un compte de service est un type d'objet spécial qui vous permet d'attribuer un rôle Kubernetes RBAC à un pod. Un compte de service par défaut est créé automatiquement pour chaque espace de noms au sein d'un cluster. Lorsque vous déployez un pod dans un espace de noms sans faire référence à un compte de service spécifique, le compte de service par défaut pour cet espace de noms est automatiquement attribué au pod et le secret, c'est-à-dire le jeton de compte de service (JWT) pour ce compte de service, est monté sur le pod en tant que volume à. `/var/run/secrets/kubernetes.io/serviceaccount` Le décodage du jeton du compte de service dans ce répertoire révélera les métadonnées suivantes :

```
{
  "iss": "kubernetes/serviceaccount",
  "kubernetes.io/serviceaccount/namespace": "default",
  "kubernetes.io/serviceaccount/secret.name": "default-token-5pv4z",
  "kubernetes.io/serviceaccount/service-account.name": "default",
```

```
  "kubernetes.io/serviceaccount/service-account.uid":
 "3b36ddb5-438c-11ea-9438-063a49b60fba",
  "sub": "system:serviceaccount:default:default"
}
```

Le compte de service par défaut dispose des autorisations suivantes pour accéder à l'API Kubernetes.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  creationTimestamp: "2020-01-30T18:13:25Z"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:discovery
  resourceVersion: "43"
  selfLink: /apis/rbac.authorization.k8s.io/v1/clusterroles/system%3Adiscovery
  uid: 350d2ab8-438c-11ea-9438-063a49b60fba
rules:
- nonResourceURLs:
  - /api
  - /api/*
  - /apis
  - /apis/*
  - /healthz
  - /openapi
  - /openapi/*
  - /version
  - /version/
  verbs:
  - get
```

Ce rôle autorise les utilisateurs authentifiés et non authentifiés à lire les informations de l'API et est considéré comme sûr pour être accessible au public.

Lorsqu'une application exécutée dans un Pod appelle les Kubernetes APIs, le Pod doit se voir attribuer un compte de service qui lui accorde explicitement l'autorisation de les appeler. APIs À l'instar des directives relatives à l'accès des utilisateurs, le rôle ou le lien ClusterRole lié à un compte de service doivent être limités aux ressources et méthodes d'API dont l'application a besoin pour fonctionner et rien d'autre. Pour utiliser un compte de service autre que celui par défaut, définissez

simplement le `spec.serviceAccountName` champ d'un Pod avec le nom du compte de service que vous souhaitez utiliser. Pour plus d'informations sur la création de comptes de service, consultez https://kubernetes. io/docs/reference/access-authn-authz/rbac/#. service-account-permissions

> **Note**
>
> Avant Kubernetes 1.24, Kubernetes créait automatiquement un secret pour chaque compte de service. Ce secret a été monté sur le pod à l'adresse/var/run/secrets/kubernetes.io/serviceaccountet serait utilisé par le pod pour s'authentifier auprès du serveur d'API Kubernetes. Dans Kubernetes 1.24, un jeton de compte de service est généré dynamiquement lorsque le pod fonctionne et n'est valide que pendant une heure par défaut. Aucun secret pour le compte de service ne sera créé. Si vous avez une application qui s'exécute en dehors du cluster et qui doit s'authentifier auprès de l'API Kubernetes, par exemple Jenkins, vous devrez créer un secret de type `kubernetes.io/service-account-token` ainsi qu'une annotation faisant référence au compte de service, telle que. `metadata.annotations.kubernetes.io/service-account.name: <SERVICE_ACCOUNT_NAME>` Les secrets ainsi créés n'expirent pas.

Rôles IAM pour les comptes de service (IRSA)

L'IRSA est une fonctionnalité qui vous permet d'attribuer un rôle IAM à un compte de service Kubernetes. Il fonctionne en tirant parti d'une fonctionnalité de Kubernetes connue sous le nom de Service Account Token Volume Projection. Lorsque les pods sont configurés avec un compte de service qui fait référence à un rôle IAM, le serveur d'API Kubernetes appelle le point de terminaison de découverte OIDC public du cluster au démarrage. Le point de terminaison signe cryptographiquement le jeton OIDC émis par Kubernetes et le jeton obtenu est monté sous forme de volume. Ce jeton signé permet au Pod d'appeler le rôle IAM APIs associé à AWS. Lorsqu'une API AWS est invoquée, l'AWS SDKs appelle`sts:AssumeRoleWithWebIdentity`. Après avoir validé la signature du jeton, IAM échange le jeton émis par Kubernetes contre un identifiant de rôle AWS temporaire.

Lorsque vous utilisez IRSA, il est important de réutiliser les sessions du SDK AWS afin d'éviter les appels inutiles à AWS STS.

Le décodage du jeton (JWT) pour IRSA produira un résultat similaire à l'exemple ci-dessous :

```
{
```

```
  "aud": [
    "sts.amazonaws.com"
  ],
  "exp": 1582306514,
  "iat": 1582220114,
  "iss": "https://oidc.eks.us-west-2.amazonaws.com/id/
D43CF17C27A865933144EA99A26FB128",
  "kubernetes.io": {
    "namespace": "default",
    "pod": {
      "name": "alpine-57b5664646-rf966",
      "uid": "5a20f883-5407-11ea-a85c-0e62b7a4a436"
    },
    "serviceaccount": {
      "name": "s3-read-only",
      "uid": "a720ba5c-5406-11ea-9438-063a49b60fba"
    }
  },
  "nbf": 1582220114,
  "sub": "system:serviceaccount:default:s3-read-only"
}
```

Ce jeton particulier accorde au Pod des privilèges d'affichage uniquement à S3 en assumant un rôle IAM. Lorsque l'application tente de lire depuis S3, le jeton est échangé contre un ensemble temporaire d'informations d'identification IAM semblable à ceci :

```
{
    "AssumedRoleUser": {
        "AssumedRoleId": "AROA36C6WWEJULFUYMPB6:abc",
        "Arn": "arn:aws:sts::123456789012:assumed-role/eksctl-winterfell-addon-
iamserviceaccount-de-Role1-1D61LT75JH3MB/abc"
    },
    "Audience": "sts.amazonaws.com",
    "Provider": "arn:aws:iam::123456789012:oidc-provider/oidc.eks.us-
west-2.amazonaws.com/id/D43CF17C27A865933144EA99A26FB128",
    "SubjectFromWebIdentityToken": "system:serviceaccount:default:s3-read-only",
    "Credentials": {
        "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
        "SessionToken": "FwoGZXIvYXdzEGMaDMLxAZkuLpmSwYXShiL9A1S0X87VBC1mHCrRe/
pB2oesl1eXxUYnPJyC9ayOoXMvqXQsomq0xs6OqZ3vaa5Iw1HIyA4Cv1suLaOCoU3hNvOIJ6C94H1vU0siQYk7DIq9Av5RZ
Zv2A6zp5xGz9cWj2f0aD9v66vX4bexOs5t/YYhwuwAvkkJPSIGvxja0xRThnceHyFHKtj0Hbi/
PWAtlI8YJcDX69cM30JAHDdQHltm/4scFptW1hlvMaPWReCAaCrsHrATyka7ttw5YlUyvZ8EPogj6fwHlxmrXM9h1Bqdiko
LgkWsCTGlYcY8z3zkigJMbYn07ewTL5Ss7LazTJJa758I7PZan/
```

```
v3xQHd5DEc5WBneiV3iOznDFgup0VAMkIviVjVCkszaPSVEdK2NU7jtrh6Jfm7bU/3P6ZGCkyDLIa8MBn9KPXeJd/
yjTk5IifIwO/mDpGNUribg6TPxhzZ8b/XdZO1kS1gVgqjXyVCM+BRBh6C4H21w/eMzjCtDIpoxt5rGKL6Nu/
IFMipoC4fgx6LIIHwtGYMG7SWQi7OsMAkiwZRg0n68/RqWgLzBt/4pfjSRYuk=",
        "Expiration": "2020-02-20T18:49:50Z",
        "AccessKeyId": "ASIAIOSFODNN7EXAMPLE"
    }
}
```

Un webhook mutant qui s'exécute dans le cadre du plan de contrôle EKS injecte l'ARN du rôle AWS et le chemin d'accès à un fichier de jeton d'identité Web dans le Pod en tant que variables d'environnement. Ces valeurs peuvent également être fournies manuellement.

```
AWS_ROLE_ARN=arn:aws:iam::AWS_ACCOUNT_ID:role/IAM_ROLE_NAME
AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
```

Le kubelet fera automatiquement pivoter le jeton projeté lorsqu'il est supérieur à 80 % de son TTL total, ou après 24 heures. Les AWS SDKs sont chargés de recharger le jeton lors de sa rotation. Pour plus d'informations sur l'IRSA, voir https://docs.aws.amazon.com/eks/ latest/userguide/iam - roles-for-service-accounts -technical-overview.html.

Identités du pod EKS

EKS Pod Identities est une fonctionnalité lancée lors de re:Invent 2023 qui vous permet d'attribuer un rôle IAM à un compte de service Kubernetes, sans avoir à configurer un fournisseur d'identité (IDP) Open Id Connect (OIDC) pour chaque cluster de votre compte AWS. Pour utiliser EKS Pod Identity, vous devez déployer un agent qui s'exécute en tant que DaemonSet pod sur chaque nœud de travail éligible. Cet agent est mis à votre disposition en tant que module complémentaire EKS et constitue une condition préalable à l'utilisation de la fonction EKS Pod Identity. Vos applications doivent utiliser une version prise en charge du SDK AWS pour utiliser cette fonctionnalité.

Lorsque les identités de pod EKS sont configurées pour un pod, EKS monte et actualise un jeton d'identité de pod sur`/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token`. Ce jeton sera utilisé par le SDK AWS pour communiquer avec l'agent EKS Pod Identity, qui utilise le jeton d'identité du pod et le rôle IAM de l'agent pour créer des informations d'identification temporaires pour vos pods en appelant l'AssumeRoleForPodIdentityAPI. Le jeton d'identité du pod livré à vos pods est un JWT émis par votre cluster EKS et signé cryptographiquement, avec les déclarations JWT appropriées à utiliser avec EKS Pod Identities.

Pour en savoir plus sur EKS Pod Identities, consultez ce blog.

Il n'est pas nécessaire de modifier le code de votre application pour utiliser EKS Pod Identities. Les versions du SDK AWS prises en charge découvriront automatiquement les informations d'identification mises à disposition avec EKS Pod Identities en utilisant la chaîne de fournisseurs d'informations d'identification. À l'instar de l'IRSA, EKS pod identity définit des variables dans vos pods pour leur indiquer comment trouver les informations d'identification AWS.

Utilisation des rôles IAM pour EKS Pod Identities

- EKS Pod Identities ne peut assumer directement qu'un rôle IAM appartenant au même compte AWS que le cluster EKS. Pour accéder à un rôle IAM dans un autre compte AWS, vous devez assumer ce rôle en configurant un profil dans la configuration de votre SDK ou dans le code de votre application.
- Lorsque les identités du module EKS sont configurées pour les comptes de service, la personne ou le processus qui configure l'association d'identité du pod doit avoir les `iam:PassRole` droits requis pour ce rôle.
- Chaque compte de service ne peut être associé qu'à un seul rôle IAM via EKS Pod Identities, mais vous pouvez associer le même rôle IAM à plusieurs comptes de service.
- Les rôles IAM utilisés avec EKS Pod Identities doivent autoriser le principal de `pods.eks.amazonaws.com` service à les assumer et à définir des balises de session. Voici un exemple de politique de confiance des rôles qui permet à EKS Pod Identities d'utiliser un rôle IAM :

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "pods.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceOrgId": "${aws:ResourceOrgId}"
        }
      }
    }
```

```
  ]
}
```

AWS recommande d'utiliser des clés de condition `aws:SourceOrgId` pour se protéger contre le problème de confusion entre les services adjoints. Dans l'exemple de politique de confiance des rôles ci-dessus, `ResourceOrgId` il s'agit d'une variable égale à l'ID d'organisation AWS Organizations de l'organisation AWS à laquelle appartient le compte AWS. EKS transmettra une valeur `aws:SourceOrgId` égale à cette valeur lorsqu'il assumera un rôle auprès d'EKS Pod Identities.

Identités des pods ABAC et EKS

Lorsque EKS Pod Identities assume un rôle IAM, il définit les balises de session suivantes :

Tag de session EKS Pod Identities	Value
espace de noms kubernetes	L'espace de noms dans lequel s'exécute le pod associé à EKS Pod Identities.
kubernetes-service-account	Le nom du compte de service Kubernetes associé à EKS Pod Identities
eks-cluster-arn	L'ARN du cluster EKS, par `arn:${Partition}:eks:${Region}:${Account}:cluster/${ClusterName}` ex. L'ARN du cluster est unique, mais si un cluster est supprimé et recréé dans la même région avec le même nom, au sein du même compte AWS, il aura le même ARN.
eks-cluster-name	Nom du cluster EKS. Notez que les noms des clusters EKS peuvent être identiques dans votre compte AWS et ceux des clusters EKS dans d'autres comptes AWS.
kubernetes-pod-name	Nom du module dans EKS.
kubernetes-pod-uid	L'UID du pod dans EKS.

Ces balises de session vous permettent d'utiliser le contrôle d'accès basé sur les attributs (ABAC) pour accorder l'accès à vos ressources AWS uniquement à des comptes de service Kubernetes spécifiques. Ce faisant, il est très important de comprendre que les comptes de service Kubernetes ne sont uniques qu'au sein d'un espace de noms, et que les espaces de noms Kubernetes ne sont uniques qu'au sein d'un cluster EKS. Ces balises de session sont accessibles dans les politiques AWS à l'aide de la clé de condition `aws:PrincipalTag/<tag-key>` globale, telle que `aws:PrincipalTag/eks-cluster-arn`

Par exemple, si vous souhaitez accorder l'accès à un compte de service spécifique uniquement pour accéder à une ressource AWS de votre compte avec un IAM ou une politique de ressources, vous devez vérifier `eks-cluster-arn` et `kubernetes-namespace` étiqueter ainsi que `kubernetes-service-account` pour vous assurer que seuls les comptes de service du cluster prévu ont accès à cette ressource, car d'autres clusters peuvent avoir un identifiant `kubernetes-service-accounts` et`kubernetes-namespaces`.

Cet exemple de politique de compartiment S3 accorde uniquement l'accès aux objets du compartiment S3 auquel il est attaché, uniquement si `kubernetes-service-account` `eks-cluster-arn` tous répondent aux valeurs attendues, lorsque le cluster EKS est hébergé dans le compte AWS`111122223333`. `kubernetes-namespace`

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "s3:*",
            "Resource":            [
                "arn:aws:s3:::ExampleBucket/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalTag/kubernetes-service-account": "s3objectservice",
                    "aws:PrincipalTag/eks-cluster-arn": "arn:aws:eks:us-
west-2:111122223333:cluster/ProductionCluster",
                    "aws:PrincipalTag/kubernetes-namespace": "s3datanamespace"
                }
            }
        }
```

```
    ]
}
```

Comparaison entre EKS Pod Identities et IRSA

EKS Pod Identities et IRSA sont les méthodes préférées pour fournir des informations d'identification AWS temporaires à vos pods EKS. À moins que vous n'ayez des cas d'utilisation spécifiques pour IRSA, nous vous recommandons d'utiliser EKS Pod Identities lorsque vous utilisez EKS. Ce tableau permet de comparer les deux fonctionnalités.

#	Identités du pod EKS	IRSA
Avez-vous besoin d'une autorisation pour créer un IDP OIDC dans vos comptes AWS ?	Non	Oui
Nécessite une configuration IDP unique par cluster	Non	Oui
Définit les balises de session pertinentes à utiliser avec ABAC	Oui	Non
Nécessite un IAM : PassRole Check ?	Oui	Non
Utilise le quota AWS STS de votre compte AWS ?	Non	Oui
Peut accéder à d'autres comptes AWS	Indirectement avec le chaînage de rôles	Directement avec Sts : AssumeRoleWithWebIdentity
Compatible avec AWS SDKs	Oui	Oui
Nécessite le daemonset de l'agent Pod Identity sur les nœuds ?	Oui	Non

Recommandations relatives aux identités et aux informations d'identification pour les pods EKS

Mettez à jour le daemonset aws-node pour utiliser IRSA

À l'heure actuelle, le daemonset aws-node est configuré pour utiliser un rôle attribué aux instances EC2 à attribuer aux pods. IPs Ce rôle inclut plusieurs politiques gérées par AWS, par exemple Amazoneks_CNI_Policy, qui autorisent effectivement tous les pods exécutés sur un nœud à accéder à des adresses EC2 ContainerRegistryReadOnly IP ou à extraire des images depuis ECR. attach/detach ENIs assign/unassign Étant donné que cela présente un risque pour votre cluster, il est recommandé de mettre à jour le daemonset aws-node pour utiliser IRSA. Un script permettant de le faire se trouve dans le dépôt de ce guide.

Le daemonset aws-node prend en charge EKS Pod Identities dans les versions v1.15.5 et ultérieures.

Restreindre l'accès au profil d'instance attribué au nœud de travail

Lorsque vous utilisez IRSA ou EKS Pod Identities, cela met à jour la chaîne d'identification du pod pour utiliser d'abord IRSA ou EKS Pod Identities. Toutefois, le pod peut toujours hériter des droits du profil d'instance attribué au nœud de travail. Pour les pods qui n'ont pas besoin de ces autorisations, vous pouvez bloquer l'accès aux métadonnées de l'instance afin de garantir que vos applications disposent uniquement des autorisations dont elles ont besoin, et non de leurs nœuds.

> **Warning**
>
> Le blocage de l'accès aux métadonnées de l'instance empêchera les pods qui n'utilisent pas IRSA ou EKS Pod Identities d'hériter du rôle attribué au nœud de travail.

Vous pouvez bloquer l'accès aux métadonnées de l'instance en demandant à l'instance de IMDSv2 n'utiliser que celle-ci et en mettant à jour le nombre de sauts à 1, comme dans l'exemple ci-dessous. Vous pouvez également inclure ces paramètres dans le modèle de lancement du groupe de nœuds. Ne désactivez pas les métadonnées d'instance, car cela empêcherait des composants tels que le gestionnaire de terminaison de nœud et d'autres éléments qui dépendent des métadonnées d'instance de fonctionner correctement.

```
$ aws ec2 modify-instance-metadata-options --instance-id <value> --http-tokens required
 --http-put-response-hop-limit 1
...
```

Si vous utilisez Terraform pour créer des modèles de lancement à utiliser avec des groupes de nœuds gérés, ajoutez le bloc de métadonnées pour configurer le nombre de sauts, comme indiqué dans cet extrait de code :

```
tf hl_lines="7" resource "aws_launch_template" "foo" { name = "foo" …
 metadata_options { http_endpoint = "enabled" http_tokens = "required"
http_put_response_hop_limit = 1 instance_metadata_tags = "enabled" } …
```

Vous pouvez également bloquer l'accès d'un pod aux métadonnées EC2 en manipulant iptables sur le nœud. Pour plus d'informations sur cette méthode, consultez Limiter l'accès au service de métadonnées d'instance.

Si votre application utilise une ancienne version du SDK AWS qui ne prend pas en charge les identités IRSA ou EKS Pod, vous devez mettre à jour la version du SDK.

Étendre la politique de confiance des rôles IAM pour les rôles IRSA au nom du compte de service, à l'espace de noms et au cluster

La politique de confiance peut être étendue à un espace de noms ou à un compte de service spécifique au sein d'un espace de noms. Lorsque vous utilisez IRSA, il est préférable de rendre la politique de confiance des rôles aussi explicite que possible en incluant le nom du compte de service. Cela empêchera efficacement d'autres pods du même espace de noms d'assumer le rôle. La CLI le `eksctl` fera automatiquement lorsque vous l'utiliserez pour créer des accounts/IAM rôles de service. Voir https://eksctl. io/usage/iamserviceaccounts/pour de plus amples informations.

Lorsque vous travaillez directement avec IAM, cela ajoute une condition à la politique de confiance du rôle qui utilise des conditions pour garantir que la `:sub` réclamation correspond à l'espace de noms et au compte de service que vous attendez. Par exemple, nous avions auparavant un jeton IRSA avec une sous-réclamation « system:serviceaccount:default:s3-read-only ». Il s'agit de l'`default`espace de noms et du compte de service. `s3-read-only` Vous devez utiliser une condition comme la suivante pour vous assurer que seul votre compte de service dans un espace de noms donné de votre cluster peut assumer ce rôle :

```
  "Condition": {
      "StringEquals": {
          "oidc.eks.us-west-2.amazonaws.com/id/D43CF17C27A865933144EA99A26FB128:aud":
 "sts.amazonaws.com",
          "oidc.eks.us-west-2.amazonaws.com/id/D43CF17C27A865933144EA99A26FB128:sub":
 "system:serviceaccount:default:s3-read-only"
      }
```

```
}
```

Utiliser un rôle IAM par application

Avec IRSA et EKS Pod Identity, il est recommandé de donner à chaque application son propre rôle IAM. Vous bénéficiez ainsi d'une meilleure isolation, car vous pouvez modifier une application sans en affecter une autre, et vous pouvez appliquer le principe du moindre privilège en n'accordant à une application que les autorisations dont elle a besoin.

Lorsque vous utilisez ABAC avec EKS Pod Identity, vous pouvez utiliser un rôle IAM commun à plusieurs comptes de service et vous fier à leurs attributs de session pour le contrôle d'accès. Cela est particulièrement utile lorsque vous travaillez à grande échelle, car ABAC vous permet de travailler avec moins de rôles IAM.

Lorsque votre application a besoin d'accéder à IMDS, utilisez IMDSv2 et augmentez la limite de sauts sur les instances EC2 à 2

IMDSv2nécessite que vous utilisiez une requête PUT pour obtenir un jeton de session. La demande PUT initiale doit inclure un TTL pour le jeton de session. Les nouvelles versions des kits SDK AWS s'en chargeront automatiquement, ainsi que le renouvellement de ce jeton. Il est également important de savoir que la limite de sauts par défaut sur les instances EC2 est intentionnellement définie sur 1 pour empêcher le transfert IP. Par conséquent, les pods qui demandent un jeton de session exécuté sur des instances EC2 peuvent éventuellement expirer et recommencer à utiliser le flux de IMDSv1 données. EKS ajoute du support IMDSv2 en activant les versions v1 et v2 et en modifiant la limite de sauts à 2 sur les nœuds approvisionnés par eksctl ou avec les modèles officiels. CloudFormation

Désactiver le montage automatique des jetons de compte de service

Si votre application n'a pas besoin d'appeler l'API Kubernetes, définissez l'`automountServiceAccountToken`attribut sur « `false` in » PodSpec pour votre application ou corrigez le compte de service par défaut dans chaque espace de noms afin qu'il ne soit plus automatiquement monté sur les pods. Par exemple :

```
kubectl patch serviceaccount default -p $'automountServiceAccountToken: false'
```

Utilisez des comptes de service dédiés pour chaque application

Chaque application doit disposer de son propre compte de service dédié. Cela s'applique aux comptes de service pour l'API Kubernetes ainsi qu'à IRSA et EKS Pod Identity.

> **Important**
>
> Si vous utilisez une blue/green approche de mise à niveau de cluster au lieu d'effectuer une mise à niveau de cluster sur place lorsque vous utilisez IRSA, vous devrez mettre à jour la politique de confiance de chacun des rôles IRSA IAM avec le point de terminaison OIDC du nouveau cluster. Lors d'une mise à niveau de blue/green cluster, vous créez un cluster exécutant une version plus récente de Kubernetes parallèlement à l'ancien cluster et vous utilisez un équilibreur de charge ou un maillage de services pour transférer facilement le trafic des services exécutés sur l'ancien cluster vers le nouveau cluster. Lorsque vous utilisez des mises à niveau de blue/green cluster avec EKS Pod Identity, vous devez créer des associations d'identité d'espace entre les rôles IAM et les comptes de service dans le nouveau cluster. Et mettez à jour la politique de confiance des rôles IAM si vous rencontrez une `sourceArn` condition.

Exécuter l'application en tant qu'utilisateur non root

Les conteneurs s'exécutent en tant que root par défaut. Bien que cela leur permette de lire le fichier du jeton d'identité Web, l'exécution d'un conteneur en tant que root n'est pas considérée comme une bonne pratique. Vous pouvez également envisager d'ajouter l'`spec.securityContext.runAsUser`attribut au PodSpec. La valeur de `runAsUser` est une valeur arbitraire.

Dans l'exemple suivant, tous les processus du Pod s'exécuteront sous l'ID utilisateur spécifié dans le `runAsUser` champ.

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
```

Lorsque vous exécutez un conteneur en tant qu'utilisateur non root, cela empêche le conteneur de lire le jeton du compte de service IRSA car les autorisations 0600 [root] sont attribuées au jeton par défaut. Si vous mettez à jour le SecurityContext de votre conteneur pour inclure fsgroup=65534 [Nobody], cela permettra au conteneur de lire le jeton.

```
spec:
  securityContext:
    fsGroup: 65534
```

Dans Kubernetes 1.19 et versions ultérieures, cette modification n'est plus requise et les applications peuvent lire le jeton du compte de service IRSA sans l'ajouter au groupe Nobody.

Accorder l'accès le moins privilégié aux applications

Action Hero est un utilitaire que vous pouvez exécuter parallèlement à votre application pour identifier les appels d'API AWS et les autorisations IAM correspondantes dont votre application a besoin pour fonctionner correctement. Il est similaire à IAM Access Advisor dans la mesure où il vous permet de limiter progressivement l'étendue des rôles IAM attribués aux applications. Consultez la documentation sur l'octroi de l'accès le moins privilégié aux ressources AWS pour plus d'informations.

Envisagez de définir une limite d'autorisations pour les rôles IAM utilisés avec IRSA et Pod Identities. Vous pouvez utiliser la limite d'autorisations pour vous assurer que les rôles utilisés par IRSA ou Pod Identities ne peuvent pas dépasser un niveau maximum d'autorisations. Pour un exemple de guide sur la prise en main des limites d'autorisations avec un exemple de politique de limites d'autorisations, veuillez consulter ce dépôt github.

Vérifiez et révoquez les accès anonymes inutiles à votre cluster EKS

Idéalement, l'accès anonyme doit être désactivé pour toutes les actions d'API. L'accès anonyme est accordé en créant un RoleBinding ou ClusterRoleBinding pour le système utilisateur intégré de Kubernetes : anonymous. Vous pouvez utiliser l'outil rbac-lookup pour identifier les autorisations que l'utilisateur system:anonymous possède sur votre cluster :

```
./rbac-lookup | grep -P 'system:(anonymous)|(unauthenticated)'
system:anonymous              cluster-wide        ClusterRole/system:discovery
system:unauthenticated        cluster-wide        ClusterRole/system:discovery
system:unauthenticated        cluster-wide        ClusterRole/system:public-info-
viewer
```

Aucun rôle ClusterRole autre que system : ne public-info-viewer doit être lié à system:anonymous user ou system:unauthenticated group.

Il peut y avoir des raisons légitimes d'autoriser l'accès anonyme à des informations spécifiques APIs. Si tel est le cas pour votre cluster, assurez-vous que seules les informations spécifiques APIs sont accessibles aux utilisateurs anonymes et le fait de les exposer APIs sans authentification ne rend pas votre cluster vulnérable.

Avant la Kubernetes/EKS version 1.14, le groupe system:unauthenticated était associé par défaut à system:discovery et system:basic-user. ClusterRoles Notez que même si vous avez mis à jour votre cluster vers la version 1.14 ou supérieure, ces autorisations peuvent toujours être activées sur votre cluster, car les mises à jour du cluster ne les révoquent pas. Pour vérifier ceux qui ClusterRoles ont « system:unauthenticated » sauf system : public-info-viewer vous pouvez exécuter la commande suivante (nécessite jq util) :

```
kubectl get ClusterRoleBinding -o json | jq -r '.items[] | select(.subjects[]?.name
 =="system:unauthenticated") | select(.metadata.name != "system:public-info-viewer")
 | .metadata.name'
```

Et « system:unauthenticated » peut être supprimé de tous les rôles sauf « system : » en utilisant : public-info-viewer

```
kubectl get ClusterRoleBinding -o json | jq -r '.items[] | select(.subjects[]?.name
 =="system:unauthenticated") | select(.metadata.name != "system:public-info-viewer") |
 del(.subjects[] | select(.name =="system:unauthenticated"))' | kubectl apply -f -
```

Vous pouvez également le vérifier et le supprimer manuellement en utilisant kubectl describe et kubectl edit. Pour vérifier si system:unauthenticated group dispose des autorisations system:discovery sur votre cluster, exécutez la commande suivante :

```
kubectl describe clusterrolebindings system:discovery

Name:         system:discovery
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind:  ClusterRole
  Name:  system:discovery
Subjects:
  Kind   Name                     Namespace
```

```
  ----   ----                    ---------
  Group  system:authenticated
  Group  system:unauthenticated
```

Pour vérifier si system:unauthenticated group dispose de l'autorisation system:basic-user sur votre cluster, exécutez la commande suivante :

```
kubectl describe clusterrolebindings system:basic-user

Name:         system:basic-user
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind:  ClusterRole
  Name:  system:basic-user
Subjects:
  Kind   Name                    Namespace
  ----   ----                    ---------
  Group  system:authenticated
  Group  system:unauthenticated
```

Si system:unauthenticated group est lié à system:discovery et/ou system:basic-user ClusterRoles sur votre cluster, vous devez dissocier ces rôles de system:unauthenticated group. Modifiez system:discovery à ClusterRoleBinding l'aide de la commande suivante :

```
kubectl edit clusterrolebindings system:discovery
```

La commande ci-dessus ouvrira la définition actuelle de system:discovery ClusterRoleBinding dans un éditeur, comme indiqué ci-dessous :

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will
 be
# reopened with the relevant failures.
#
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  creationTimestamp: "2021-06-17T20:50:49Z"
  labels:
```

```
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:discovery
  resourceVersion: "24502985"
  selfLink: /apis/rbac.authorization.k8s.io/v1/clusterrolebindings/system%3Adiscovery
  uid: b7936268-5043-431a-a0e1-171a423abeb6
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:discovery
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:authenticated
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:unauthenticated
```

Supprimez l'entrée pour system:unauthenticated group dans la section « Sujets » de l'écran de l'éditeur ci-dessus.

Répétez les mêmes étapes pour ClusterRoleBinding system:basic-user.

Réutilisez les sessions du SDK AWS avec IRSA

Lorsque vous utilisez IRSA, les applications écrites à l'aide du SDK AWS utilisent le jeton fourni à vos pods pour appeler afin de générer des informations `sts:AssumeRoleWithWebIdentity` d'identification AWS temporaires. Cela est différent des autres services de calcul AWS, dans lesquels le service de calcul fournit des informations d'identification AWS temporaires directement à la ressource de calcul AWS, comme une fonction lambda. Cela signifie que chaque fois qu'une session du SDK AWS est initialisée, un appel `AssumeRoleWithWebIdentity` est effectué vers AWS STS for. Si votre application évolue rapidement et initialise de nombreuses sessions du SDK AWS, il se peut que vous soyez confronté à une limitation de la part d'AWS STS, car votre code devra faire l'objet de nombreux appels. `AssumeRoleWithWebIdentity`

Pour éviter ce scénario, nous vous recommandons de réutiliser les sessions du SDK AWS au sein de votre application afin d'éviter `AssumeRoleWithWebIdentity` les appels inutiles.

Dans l'exemple de code suivant, une session est créée à l'aide du SDK boto3 python, et cette même session est utilisée pour créer des clients et interagir avec Amazon S3 et Amazon SQS. `AssumeRoleWithWebIdentity`n'est appelé qu'une seule fois, et le SDK AWS actualise les informations d'identification `my_session` lorsqu'elles expirent automatiquement.

```
import boto3

= Create your own session

my_session = boto3.session.Session()

= Now we can create low-level clients from our session

sqs = my_session.client('`sqs`') s3 = my_session.client('`s3`')

s3response = s3.list_buckets() sqsresponse = sqs.list_queues()

#print the response from the S3 and SQS APIs print("`s3 response:`")
print(s3response) print("`—`") print("`sqs response:`")
print(sqsresponse)
```

Si vous migrez une application depuis un autre service de calcul AWS, tel qu'EC2, vers EKS avec IRSA, ce détail est particulièrement important. Sur les autres services de calcul, l'initialisation d'une session du SDK AWS n'appelle pas AWS STS sauf si vous le lui demandez.

Approches alternatives

Bien que les identités de pod IRSA et EKS soient les méthodes préférées pour attribuer une identité AWS à un pod, elles nécessitent que vous incluiez une version récente d'AWS SDKs dans votre application. Pour une liste complète de ceux SDKs qui prennent actuellement en charge l'IRSA, voir https://docs.aws.amazon.com/eks/ latest/userguide/iam-roles-for-service-accounts-minimum-sdk.html, for EKS Pod Identities, see https://docs.aws.amazon.com/eks/latest/userguide/ pod - id-minimum-sdk .html. Si vous avez une application que vous ne pouvez pas mettre à jour immédiatement avec un SDK compatible, plusieurs solutions communautaires sont disponibles pour attribuer des rôles IAM aux pods Kubernetes, notamment kube2iam et kiam. Bien qu'AWS n'approuve, ne tolère ni ne soutienne l'utilisation de ces solutions, celles-ci sont fréquemment utilisées par l'ensemble de la communauté pour obtenir des résultats similaires à ceux d'IRSA et d'EKS Pod Identities.

Si vous devez utiliser l'une de ces solutions non fournies par AWS, veuillez faire preuve de diligence raisonnable et vous assurer que vous comprenez les implications en matière de sécurité.

Outils et ressources

- Atelier d'immersion sur la sécurité Amazon EKS - Identity and Access Management

- Modèle de plans Terraform EKS - Cluster Amazon EKS entièrement privé
- Modèle de plans Terraform EKS - Authentification unique au centre d'identité IAM pour le cluster Amazon EKS
- Modèle de plans Terraform EKS - Authentification unique Okta pour Amazon EKS Cluster
- audit2rbac
- rbac.dev Une liste de ressources supplémentaires, notamment des blogs et des outils, pour Kubernetes RBAC
- Héros d'action
- kube2iam
- kiam

Sécurité du pod

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

La spécification du module inclut une variété d'attributs différents qui peuvent renforcer ou affaiblir votre posture de sécurité globale. En tant que praticien de Kubernetes, votre principale préoccupation devrait être d'empêcher un processus exécuté dans un conteneur d'échapper aux limites d'isolation du runtime du conteneur et d'accéder à l'hôte sous-jacent.

Fonctionnalités de Linux

Les processus exécutés dans un conteneur s'exécutent par défaut dans le contexte de l'utilisateur root [Linux]. Bien que les actions du root au sein d'un conteneur soient partiellement limitées par l'ensemble des fonctionnalités Linux que le moteur d'exécution du conteneur attribue aux conteneurs, ces privilèges par défaut peuvent permettre à un attaquant d'augmenter ses privilèges .7.html. and/or gain access to sensitive information bound to the host, including Secrets and ConfigMaps. Below is a list of the default capabilities assigned to containers. For additional information about each capability, see http://man7.org/linux/man-pages/man7/capabilities

```
CAP_AUDIT_WRITE, CAP_CHOWN, CAP_DAC_OVERRIDE, CAP_FOWNER, CAP_FSETID,
CAP_KILL, CAP_MKNOD, CAP_NET_BIND_SERVICE, CAP_NET_RAW, CAP_SETGID,
CAP_SETUID, CAP_SETFCAP, CAP_SETPCAP, CAP_SYS_CHROOT
```

Example

Les fonctionnalités susmentionnées sont attribuées par défaut aux pods EC2 et Fargate. De plus, les fonctionnalités Linux ne peuvent être supprimées que depuis les pods Fargate.

Les pods exécutés en tant que privilèges héritent de toutes les fonctionnalités Linux associées au root sur l'hôte. Cela doit être évité dans la mesure du possible.

Autorisation du nœud

Tous les nœuds de travail Kubernetes utilisent un mode d'autorisation appelé Node Authorization. L'autorisation de nœud autorise toutes les demandes d'API provenant du kubelet et permet aux nœuds d'effectuer les actions suivantes :

Opérations de lecture :

- services
- points de terminaison
- nœuds
- pods
- secrets, configmaps, réclamations de volumes persistants et volumes persistants liés aux pods liés au nœud du kubelet

Opérations d'écriture :

- nœuds et état des nœuds (activez le plugin `NodeRestriction` d'admission pour limiter un kubelet à modifier son propre nœud)
- pods et statut des pods (activez le plugin `NodeRestriction` d'admission pour limiter un kubelet afin de modifier les pods liés à lui-même)
- events

Opérations liées à l'authentification :

- Accès en lecture/écriture à l'API CertificateSigningRequest (CSR) pour le démarrage du protocole TLS
- la capacité de créer TokenReview et de SubjectAccessReview déléguer authentication/authorization des contrôles

EKS utilise le contrôleur d'admission par restriction de nœud qui permet uniquement au nœud de modifier un ensemble limité d'attributs de nœud et d'objets de pod liés au nœud. Néanmoins, un attaquant parvenant à accéder à l'hôte sera toujours en mesure de glaner des informations sensibles sur l'environnement à partir de l'API Kubernetes, ce qui pourrait lui permettre de se déplacer latéralement au sein du cluster.

Solutions de sécurité pour les pods

Politique de sécurité des pods (PSP)

Dans le passé, les ressources de la politique de sécurité des pods (PSP) étaient utilisées pour spécifier un ensemble d'exigences auxquelles les pods devaient répondre avant de pouvoir être créés. Depuis la version 1.21 de Kubernetes, les PSP sont devenues obsolètes. Leur suppression est prévue dans la version 1.25 de Kubernetes.

> **Important**
>
> PSPs sont obsolètes dans la version 1.21 de Kubernetes. Vous aurez jusqu'à la version 1.25, soit environ 2 ans, pour passer à une alternative. Ce document explique les raisons de cette dépréciation.

Migration vers une nouvelle solution de sécurité des pods

Comme PSPs ils ont été supprimés à partir de Kubernetes v1.25, les administrateurs et les opérateurs de clusters doivent remplacer ces contrôles de sécurité. Deux solutions peuvent répondre à ce besoin :

- Policy-as-code solutions (PAC) issues de l'écosystème Kubernetes
- Normes de sécurité (PSS) de Kubernetes Pod

Les solutions PAC et PSS peuvent coexister avec la PSP ; elles peuvent être utilisées dans des clusters avant le retrait de la PSP. Cela facilite l'adoption lors de la migration depuis PSP. Veuillez consulter ce document lorsque vous envisagez de migrer de PSP vers PSS.

Kyverno, l'une des solutions PAC décrites ci-dessous, propose des conseils spécifiques décrits dans un article de blog lors de la migration PSPs vers sa solution, notamment des politiques analogues, des comparaisons de fonctionnalités et une procédure de migration. Des informations et des conseils

supplémentaires sur la migration vers Kyverno en ce qui concerne le Pod Security Admission (PSA) ont été publiés sur le blog AWS ici.

Policy-as-code (PAC)

Policy-as-code Les solutions (PAC) fournissent des garde-fous pour guider les utilisateurs du cluster et empêcher les comportements indésirables, grâce à des contrôles prescrits et automatisés. Le PAC utilise les contrôleurs d'admission dynamiques Kubernetes pour intercepter le flux de demandes du serveur d'API Kubernetes, via un appel webhook, et pour muter et valider les charges utiles des demandes, sur la base de politiques écrites et stockées sous forme de code. La mutation et la validation ont lieu avant que la demande du serveur d'API n'entraîne une modification du cluster. Les solutions PAC utilisent des politiques pour faire correspondre et agir sur les charges utiles des demandes du serveur d'API, en fonction de la taxonomie et des valeurs.

Plusieurs solutions PAC open source sont disponibles pour Kubernetes. Ces solutions ne font pas partie du projet Kubernetes ; elles proviennent de l'écosystème Kubernetes. Certaines solutions PAC sont répertoriées ci-dessous.

- OPA/Gardien
- Agent de politique ouverte (OPA)
- Kyverno
- Kubewarden
- Politique JS

Pour plus d'informations sur les solutions PAC et sur la manière de vous aider à sélectionner la solution adaptée à vos besoins, consultez les liens ci-dessous.

- Contre-mesures basées sur des politiques pour Kubernetes — Partie 1
- Contre-mesures basées sur des politiques pour Kubernetes — Partie 2

Normes de sécurité des pods (PSS) et entrée de sécurité des pods (PSA)

En réponse à la dépréciation de la PSP et au besoin permanent de contrôler la sécurité des pods out-of-the-box, le Kubernetes Auth Special Interest Group a créé les normes de sécurité des pods (PSS) et Pod Security Admission (PSA) à l'aide d'une solution Kubernetes intégrée. L'effort du PSA inclut un projet de webhook pour le contrôleur d'admission qui implémente les contrôles définis dans le PSS. Cette approche du contrôleur d'admission ressemble à celle utilisée dans les solutions PAC.

Selon la documentation de Kubernetes, le PSS « définit trois politiques différentes pour couvrir largement le spectre de la sécurité. Ces politiques sont cumulatives et vont de très permissives à très restrictives. `»

Ces politiques sont définies comme suit :

- Privilégié : politique illimitée (non sécurisée), fournissant le niveau d'autorisations le plus large possible. Cette politique autorise les augmentations de privilèges connues. C'est l'absence de politique. Cela convient aux applications telles que les agents de journalisation CNIs, les pilotes de stockage et les autres applications à l'échelle du système qui nécessitent un accès privilégié.
- Base de référence : politique minimalement restrictive qui empêche les augmentations de privilèges connues. Permet la configuration par défaut (spécifiée au minimum) du pod. La politique de base interdit l'utilisation de HostNetwork, HostPID, HostPC, HostPath, HostPort, l'impossibilité d'ajouter des fonctionnalités Linux, ainsi que plusieurs autres restrictions.
- Restreint : politique de restriction stricte, conforme aux meilleures pratiques actuelles en matière de renforcement des pods. Cette politique hérite de la ligne de base et ajoute des restrictions supplémentaires, telles que l'impossibilité de s'exécuter en tant que root ou en tant que groupe root. Les politiques restreintes peuvent avoir un impact sur le fonctionnement d'une application. Ils sont principalement destinés à exécuter des applications critiques en matière de sécurité.

Ces politiques définissent des profils pour l'exécution des pods, organisés en trois niveaux d'accès privilégié ou restreint.

Pour implémenter les contrôles définis par le PSS, le PSA fonctionne selon trois modes :

- appliquer : les violations des politiques entraîneront le rejet du pod.
- audit : les violations des politiques déclencheront l'ajout d'une annotation d'audit à l'événement enregistré dans le journal d'audit, mais elles sont par ailleurs autorisées.
- avertissement : les violations des politiques déclencheront un avertissement destiné à l'utilisateur, mais elles sont autorisées dans le cas contraire.

Ces modes et les niveaux de profil (restriction) sont configurés au niveau de l'espace de noms Kubernetes, à l'aide d'étiquettes, comme indiqué dans l'exemple ci-dessous.

```
apiVersion: v1
kind: Namespace
metadata:
```

```
  name: policy-test
  labels:
    pod-security.kubernetes.io/enforce: restricted
```

Lorsqu'ils sont utilisés indépendamment, ces modes de fonctionnement ont des réponses différentes qui se traduisent par des expériences utilisateur différentes. Le mode d'application empêchera la création de pods si les PodSpecs respectifs enfreignent le niveau de restriction configuré. Toutefois, dans ce mode, les objets Kubernetes autres que des pods qui créent des pods, tels que les déploiements, ne seront pas empêchés d'être appliqués au cluster, même si le PodSpec qu'ils contiennent viole le PSS appliqué. Dans ce cas, le déploiement sera appliqué, tandis que le ou les pods ne pourront pas être appliqués.

Il s'agit d'une expérience utilisateur difficile, car rien n'indique dans l'immédiat que l'objet de déploiement correctement appliqué contredit l'échec de la création du pod. Les PodSpecs incriminés ne créeront pas de pods. L'inspection de la ressource de déploiement avec `kubectl get deploy <DEPLOYMENT_NAME> -oyaml` exposera le message provenant de l'`.status.conditions`élément pod (s) défaillant, comme indiqué ci-dessous.

```
...
status:
  conditions:
    - lastTransitionTime: "2022-01-20T01:02:08Z"
      lastUpdateTime: "2022-01-20T01:02:08Z"
      message: 'pods "test-688f68dc87-tw587" is forbidden: violates PodSecurity
 "restricted:latest":
        allowPrivilegeEscalation != false (container "test" must set
 securityContext.allowPrivilegeEscalation=false),
        unrestricted capabilities (container "test" must set
 securityContext.capabilities.drop=["ALL"]),
        runAsNonRoot != true (pod or container "test" must set
 securityContext.runAsNonRoot=true),
        seccompProfile (pod or container "test" must set
 securityContext.seccompProfile.type
        to "RuntimeDefault" or "Localhost")'
      reason: FailedCreate
      status: "True"
      type: ReplicaFailure
...
```

Dans les modes audit et avertissement, les restrictions relatives aux modules n'empêchent pas la création et le démarrage des modules non conformes. Toutefois, dans ces modes, les annotations

d'audit sur les événements du journal d'audit du serveur d'API et les avertissements adressés aux clients du serveur d'API, tels que kubectl, sont déclenchés, respectivement, lorsque les pods, ainsi que les objets qui créent des pods, contiennent des PodSpecs présentant des violations. Un message `kubectl` d'avertissement est affiché ci-dessous.

```
Warning: would violate PodSecurity "restricted:latest":
 allowPrivilegeEscalation != false (container "test" must set
 securityContext.allowPrivilegeEscalation=false), unrestricted capabilities (container
 "test" must set securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod
 or container "test" must set securityContext.runAsNonRoot=true), seccompProfile (pod
 or container "test" must set securityContext.seccompProfile.type to "RuntimeDefault"
 or "Localhost")
deployment.apps/test created
```

Les modes d'audit et d'avertissement du PSA sont utiles lors de l'introduction du PSS sans avoir d'impact négatif sur les opérations du cluster.

Les modes de fonctionnement du PSA ne s'excluent pas mutuellement et peuvent être utilisés de manière cumulative. Comme indiqué ci-dessous, les modes multiples peuvent être configurés dans un seul espace de noms.

```
apiVersion: v1
kind: Namespace
metadata:
  name: policy-test
  labels:
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/warn: restricted
```

Dans l'exemple ci-dessus, les avertissements et les annotations d'audit faciles à utiliser sont fournis lors de l'application des déploiements, tandis que l'application des violations est également fournie au niveau du module. En fait, plusieurs étiquettes PSA peuvent utiliser différents niveaux de profil, comme indiqué ci-dessous.

```
apiVersion: v1
kind: Namespace
metadata:
  name: policy-test
  labels:
```

```
    pod-security.kubernetes.io/enforce: baseline
    pod-security.kubernetes.io/warn: restricted
```

Dans l'exemple ci-dessus, le PSA est configuré pour autoriser la création de tous les pods qui répondent au niveau de profil de référence, puis pour avertir les pods (et les objets qui créent des pods) qui enfreignent le niveau de profil restreint. Il s'agit d'une approche utile pour déterminer les impacts possibles lors du passage de profils de référence à des profils restreints.

Pods existants

Si un espace de noms contenant des pods existants est modifié pour utiliser un profil PSS plus restrictif, les modes audit et avertissement produiront les messages appropriés ; toutefois, le mode Enforce ne supprimera pas les pods. Les messages d'avertissement sont affichés ci-dessous.

```
Warning: existing pods in namespace "policy-test" violate the new PodSecurity enforce
 level "restricted:latest"
Warning: test-688f68dc87-htm8x: allowPrivilegeEscalation != false, unrestricted
 capabilities, runAsNonRoot != true, seccompProfile
namespace/policy-test configured
```

Dérogations

PSA utilise des exemptions pour exclure l'application de violations à l'encontre des pods qui auraient autrement été appliquées. Ces exemptions sont énumérées ci-dessous.

- Noms d'utilisateur : les demandes des utilisateurs dont le nom d'utilisateur authentifié (ou usurpé) est exempté sont ignorées.
- RuntimeClassNames: les pods et les ressources de charge de travail spécifiant un nom de classe d'exécution exempté sont ignorés.
- Espaces de noms : les pods et les ressources de charge de travail dans un espace de noms exempté sont ignorés.

Ces exemptions sont appliquées de manière statique dans la configuration du contrôleur d'admission PSA dans le cadre de la configuration du serveur d'API.

Dans l'implémentation de Validating Webhook, les exemptions peuvent être configurées au sein d'une ConfigMapressource Kubernetes montée sous forme de volume dans le conteneur. pod-security-webhook

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: pod-security-webhook
  namespace: pod-security-webhook
data:
  podsecurityconfiguration.yaml: |
    apiVersion: pod-security.admission.config.k8s.io/v1
    kind: PodSecurityConfiguration
    defaults:
      enforce: "restricted"
      enforce-version: "latest"
      audit: "restricted"
      audit-version: "latest"
      warn: "restricted"
      warn-version: "latest"
    exemptions:
      # Array of authenticated usernames to exempt.
      usernames: []
      # Array of runtime class names to exempt.
      runtimeClasses: []
      # Array of namespaces to exempt.
      namespaces: ["kube-system","policy-test1"]
```

Comme indiqué dans le ConfigMap YAML ci-dessus, le niveau PSS par défaut à l'échelle du cluster a été défini comme restreint pour tous les modes PSA, audit, application et avertissement. Cela affecte tous les espaces de noms, à l'exception de ceux exemptés :. `namespaces: ["kube-system","policy-test1"]` De plus, dans la ValidatingWebhookConfigurationressource présentée ci-dessous, l'espace de pod-security-webhooknoms est également exempté du PSS configuré.

```
...
webhooks:
  # Audit annotations will be prefixed with this name
  - name: "pod-security-webhook.kubernetes.io"
    # Fail-closed admission webhooks can present operational challenges.
    # You may want to consider using a failure policy of Ignore, but should
    # consider the security tradeoffs.
    failurePolicy: Fail
    namespaceSelector:
      # Exempt the webhook itself to avoid a circular dependency.
      matchExpressions:
```

```
        - key: kubernetes.io/metadata.name
          operator: NotIn
          values: ["pod-security-webhook"]
...
```

> **Important**
>
> Pod Security Admissions est devenue stable dans Kubernetes v1.25. Si vous vouliez utiliser la fonction Pod Security Admission avant qu'elle ne soit activée par défaut, vous deviez installer le contrôleur d'admission dynamique (webhook mutant). Les instructions d'installation et de configuration du webhook se trouvent ici.

Choisir entre les normes de sécurité policy-as-code et celles des pods

Les Pod Security Standards (PSS) ont été développés pour remplacer la Pod Security Policy (PSP), en fournissant une solution intégrée à Kubernetes et ne nécessitant pas de solutions issues de l'écosystème Kubernetes. Cela étant dit, les solutions policy-as-code (PAC) sont considérablement plus flexibles.

La liste suivante des avantages et des inconvénients est conçue pour vous aider à prendre une décision plus éclairée concernant la solution de sécurité de votre pod.

Policy-as-code (par rapport aux normes de sécurité des pods)

Avantages :

- Plus flexible et plus granulaire (jusqu'aux attributs des ressources si nécessaire)
- Non seulement axé sur les pods, il peut être utilisé contre différentes ressources et actions
- Pas seulement appliqué au niveau de l'espace de noms
- Plus mature que les normes de sécurité du Pod
- Les décisions peuvent être basées sur n'importe quel élément de la charge utile des demandes du serveur d'API, ainsi que sur les ressources du cluster existantes et les données externes (en fonction de la solution)
- Prend en charge les requêtes mutantes du serveur d'API avant validation (dépend de la solution)
- Peut générer des politiques complémentaires et des ressources Kubernetes (en fonction de la solution). À partir des politiques des modules, Kyverno peut générer automatiquement des politiques pour les contrôleurs de niveau supérieur, tels que les déploiements. Kyverno peut

également générer des ressources Kubernetes supplémentaires « lorsqu'une nouvelle ressource est créée ou lorsque la source est mise à jour` » en utilisant Generate Rules.)

- Peut être utilisé pour passer à gauche, dans les pipelines CICD, avant de passer des appels au serveur d'API Kubernetes (selon la solution)
- Peut être utilisé pour mettre en œuvre des comportements qui ne sont pas nécessairement liés à la sécurité, tels que les meilleures pratiques, les normes organisationnelles, etc.
- Peut être utilisé dans des cas d'utilisation autres que Kubernetes (selon la solution)
- Grâce à la flexibilité, l'expérience utilisateur peut être adaptée aux besoins des utilisateurs

Inconvénients :

- Non intégré à Kubernetes
- Plus complexe à apprendre, à configurer et à prendre en charge
- La création de politiques peut nécessiter de nouvelles skills/languages/capabilities

Accès sécurisé au pod (par rapport à policy-as-code)

Avantages :

- Intégré à Kubernetes
- Plus simple à configurer
- Aucune nouvelle langue à utiliser ni aucune nouvelle politique à rédiger
- Si le niveau d'admission par défaut du cluster est configuré sur privilégié, les libellés d'espace de noms peuvent être utilisés pour ajouter des espaces de noms aux profils de sécurité du module.

Inconvénients :

- Pas aussi souple ou granuleux que policy-as-code
- Seulement 3 niveaux de restrictions
- Principalement axé sur les pods

Résumé

Si vous ne disposez pas actuellement d'une solution de sécurité pour les pods, autre que la PSP, et que la posture de sécurité requise correspond au modèle défini dans les normes de sécurité

des pods (PSS), il est peut-être plus facile d'adopter le PSS au lieu d'une solution. policy-as-code Toutefois, si la posture de sécurité de votre module ne correspond pas au modèle PSS, ou si vous envisagez d'ajouter des contrôles supplémentaires, au-delà de ceux définis par le PSS, une policy-as-code solution semble mieux adaptée.

Recommandations

Utilisez plusieurs modes Pod Security Admission (PSA) pour une meilleure expérience utilisateur

Comme indiqué précédemment, le mode d'application du PSA empêche l'application des pods présentant des violations du PSS, mais n'arrête pas les contrôleurs de niveau supérieur, tels que les déploiements. En fait, le déploiement sera appliqué avec succès sans aucune indication que les pods n'ont pas été appliqués. Bien que vous puissiez utiliser kubectl pour inspecter l'objet de déploiement et découvrir le message du PSA indiquant que les pods ont échoué, l'expérience utilisateur pourrait être meilleure. Pour améliorer l'expérience utilisateur, plusieurs modes PSA (audit, appliquer, avertir) doivent être utilisés.

```
apiVersion: v1
kind: Namespace
metadata:
  name: policy-test
  labels:
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/warn: restricted
```

Dans l'exemple ci-dessus, avec le mode d'application défini, lorsqu'un manifeste de déploiement contenant des violations du PSS dans le PodSpec correspondant est tenté d'être appliqué au serveur d'API Kubernetes, le déploiement sera correctement appliqué, mais pas les pods. Et comme les modes audit et avertissement sont également activés, le client du serveur d'API recevra un message d'avertissement et l'événement du journal d'audit du serveur d'API sera également annoté par un message.

Restreindre les conteneurs qui peuvent fonctionner en tant que privilégiés

Comme indiqué, les conteneurs qui s'exécutent en tant que privilèges héritent de toutes les fonctionnalités Linux attribuées au root sur l'hôte. Les conteneurs ont rarement besoin de ce type de

privilèges pour fonctionner correctement. Plusieurs méthodes peuvent être utilisées pour restreindre les autorisations et les capacités des conteneurs.

> **Important**
>
> Fargate est un type de lancement qui vous permet d'exécuter des conteneurs « sans serveur » dans lesquels les conteneurs d'un pod sont exécutés sur une infrastructure gérée par AWS. Avec Fargate, vous ne pouvez pas exécuter un conteneur privilégié ni configurer votre pod pour utiliser HostNetwork ou HostPort.

Ne pas exécuter de processus dans des conteneurs en tant qu'utilisateur root

Tous les conteneurs s'exécutent en tant que root par défaut. Cela peut être problématique si un attaquant parvient à exploiter une vulnérabilité de l'application et à obtenir un accès shell au conteneur en cours d'exécution. Vous pouvez atténuer ce risque de différentes manières. Tout d'abord, en retirant la coque de l'image du conteneur. Ensuite, ajoutez la directive USER à votre Dockerfile ou exécutez les conteneurs du pod en tant qu'utilisateur non root. Le Kubernetes PodSpec inclut un ensemble de champs, sous`spec.securityContext`, qui vous permettent de spécifier le and/or groupe d'utilisateurs sous lequel exécuter votre application. Ces champs sont `runAsUser` `runAsGroup` respectivement.

Pour renforcer l'utilisation du`spec.securityContext`, et de ses éléments associés, dans le cadre de Kubernetes, PodSpec policy-as-code ou Pod Security Standards peuvent être ajoutés aux clusters. Ces solutions vous permettent d'écrire et/ou d'utiliser des politiques ou des profils capables de valider les charges utiles des demandes du serveur d'API Kubernetes entrantes, avant qu'elles ne soient conservées dans etcd. En outre, policy-as-code les solutions peuvent modifier les demandes entrantes et, dans certains cas, générer de nouvelles demandes.

N'exécutez jamais Docker dans Docker et ne montez jamais le socket dans le conteneur

Bien que cela vous permette facilement d'accéder à build/run des images dans des conteneurs Docker, vous abandonnez essentiellement le contrôle total du nœud au profit du processus exécuté dans le conteneur. Si vous avez besoin de créer des images de conteneur sur Kubernetes, utilisez plutôt Kaniko, buildah ou un service de génération similaire. CodeBuild

> **Note**
>
> Les clusters Kubernetes utilisés pour le traitement CICD, tels que la création d'images de conteneurs, doivent être isolés des clusters exécutant des charges de travail plus généralisées.

Restreignez l'utilisation de HostPath ou, si HostPath est nécessaire, limitez les préfixes pouvant être utilisés et configurez le volume en lecture seule

`hostPath`est un volume qui monte un répertoire depuis l'hôte directement vers le conteneur. Les capsules auront rarement besoin de ce type d'accès, mais si c'est le cas, vous devez être conscient des risques. Par défaut, les pods qui s'exécutent en tant que root auront un accès en écriture au système de fichiers exposé par HostPath. Cela pourrait permettre à un attaquant de modifier les paramètres du kubelet, de créer des liens symboliques vers des répertoires ou des fichiers qui ne sont pas directement exposés par HostPath, par exemple /etc/shadow, d'installer des clés ssh, de lire des secrets montés sur l'hôte et d'autres éléments malveillants. Pour atténuer les risques liés à HostPath, configurez le `spec.containers.volumeMounts` comme suit`readOnly`, par exemple :

```
volumeMounts:
- name: hostPath-volume
    readOnly: true
    mountPath: /host-path
```

Vous devez également utiliser policy-as-code des solutions pour restreindre les répertoires qui peuvent être utilisés par les `hostPath` volumes ou empêcher complètement `hostPath` leur utilisation. Vous pouvez utiliser les politiques de base ou de restriction des normes de sécurité du Pod pour empêcher l'utilisation de`hostPath`.

Pour plus d'informations sur les dangers de l'escalade des privilèges, consultez le blog de Seth Art, Bad Pods : Kubernetes Pod Privilege Escalation.

Définissez des demandes et des limites pour chaque conteneur afin d'éviter les contentions de ressources et les attaques DoS

Un pod sans demandes ni limites peut théoriquement consommer toutes les ressources disponibles sur un hôte. Lorsque des pods supplémentaires sont programmés sur un nœud, le nœud peut subir une pression sur le processeur ou la mémoire, ce qui peut entraîner l'arrêt du Kubelet ou

l'expulsion des pods du nœud. Bien que vous ne puissiez pas empêcher que cela se produise complètement, la définition de demandes et de limites permet de minimiser la contention des ressources et d'atténuer les risques liés à des applications mal conçues qui consomment une quantité excessive de ressources.

Vous `podSpec` permet de spécifier les demandes et les limites du processeur et de la mémoire. Le processeur est considéré comme une ressource compressible car il peut être sursouscrit. La mémoire est incompressible, c'est-à-dire qu'elle ne peut pas être partagée entre plusieurs conteneurs.

Lorsque vous spécifiez des demandes de processeur ou de mémoire, vous désignez essentiellement la quantité de mémoire garantie aux conteneurs. Kubernetes agrège les demandes de tous les conteneurs d'un pod afin de déterminer sur quel nœud planifier le pod. Si un conteneur dépasse la quantité de mémoire demandée, il peut être mis hors service en cas de pression de mémoire sur le nœud.

Les limites sont la quantité maximale de ressources de processeur et de mémoire qu'un conteneur est autorisé à consommer et correspondent directement à la `memory.limit_in_bytes` valeur du cgroup créé pour le conteneur. Un conteneur qui dépasse la limite de mémoire sera détruit par OOM. Si un conteneur dépasse sa limite de CPU, il sera limité.

> **Note**
>
> Lors de l'utilisation d'un conteneur, `resources.limits` il est fortement recommandé que l'utilisation des ressources du conteneur (alias Resource Footprints) soit basée sur les données et précise, sur la base de tests de charge. En l'absence d'une empreinte de ressources précise et fiable, le conteneur `resources.limits` peut être rembourré. Par exemple, il `resources.limits.memory` peut être augmenté de 20 à 30 % par rapport aux maximums observables, afin de tenir compte d'éventuelles inexactitudes dans les limites des ressources de mémoire.

Kubernetes utilise trois classes de qualité de service (QoS) pour hiérarchiser les charges de travail exécutées sur un nœud. Il s’agit des licences suivantes :

- garanti
- éclatable
- meilleur effort

Si les limites et les demandes ne sont pas définies, le pod est configuré comme le meilleur effort (priorité la plus basse). Les pods Best-Effort sont les premiers à être détruits lorsque la mémoire est insuffisante. Si des limites sont définies pour tous les conteneurs du pod, ou si les demandes et les limites sont définies sur les mêmes valeurs et non égales à 0, le pod est configuré comme garanti (priorité la plus élevée). Les pods garantis ne seront pas détruits à moins qu'ils ne dépassent les limites de mémoire configurées. Si les limites et les demandes sont configurées avec des valeurs différentes et différentes de 0, ou si un conteneur du pod définit des limites et les autres non, ou si des limites sont définies pour différentes ressources, les pods sont configurés en mode rafale (priorité moyenne). Ces pods ont certaines garanties en matière de ressources, mais ils peuvent être détruits une fois qu'ils dépassent la mémoire demandée.

> **Important**
>
> Les demandes n'affectent pas la `memory_limit_in_bytes` valeur du cgroup du conteneur ; la limite du cgroup est définie en fonction de la quantité de mémoire disponible sur l'hôte. Néanmoins, si la valeur des requêtes est trop faible, le kubelet peut cibler le pod pour être arrêté par le kubelet si le nœud subit une pression de mémoire.

Classe	Priority	Condition	État de mise à mort
Garantie	le plus élevé	limite = demande ! = 0	Dépassez uniquement les limites de mémoire
Éclatable	medium	limite ! = demande ! = 0	Peut être tué en cas de dépassement de la mémoire de demande
Meilleur effort	le plus bas	limite et demande non définies	Premier à être tué lorsque la mémoire est insuffisante

Pour plus d'informations sur la QoS des ressources, consultez la documentation de Kubernetes.

Vous pouvez forcer l'utilisation de demandes et de limites en définissant un quota de ressources sur un espace de noms ou en créant une plage de limites. Un quota de ressources vous permet de spécifier la quantité totale de ressources, par exemple le processeur et la RAM, allouées à un espace

de noms. Lorsqu'il est appliqué à un espace de noms, il vous oblige à spécifier des demandes et des limites pour tous les conteneurs déployés dans cet espace de noms. En revanche, les plages de limites vous permettent de contrôler de manière plus précise l'allocation des ressources. Avec les plages limites, vous pouvez définir min/max les ressources du processeur et de la mémoire par pod ou par conteneur au sein d'un espace de noms. Vous pouvez également les utiliser pour définir les valeurs de demande/limite par défaut si aucune n'est fournie.

Policy-as-code les solutions peuvent être utilisées pour appliquer les demandes et les limites, ou même pour créer des quotas de ressources et des plages de limites lors de la création d'espaces de noms.

Ne pas autoriser l'escalade privilégiée

L'escalade privilégiée permet à un processus de modifier le contexte de sécurité dans lequel il s'exécute. Sudo en est un bon exemple, tout comme les binaires avec le bit SUID ou SGID. L'escalade de privilèges est essentiellement un moyen pour les utilisateurs d'exécuter un fichier avec les autorisations d'un autre utilisateur ou groupe. Vous pouvez empêcher un conteneur d'utiliser l'escalade privilégiée en policy-as-code implémentant une politique mutante qui définit `allowPrivilegeEscalation` `false` ou `securityContext.allowPrivilegeEscalation` en définissant le`podSpec`. Policy-as-code les politiques peuvent également être utilisées pour empêcher les demandes du serveur d'API de réussir si des paramètres incorrects sont détectés. Les normes de sécurité des pods peuvent également être utilisées pour empêcher les pods d'utiliser l'escalade de privilèges.

Désactiver les montages de ServiceAccount jetons

Pour les pods qui n'ont pas besoin d'accéder à l'API Kubernetes, vous pouvez désactiver le montage automatique d'un ServiceAccount jeton sur une spécification de pod, ou pour tous les pods utilisant un jeton en particulier. ServiceAccount

Example

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-no-automount
spec:
  automountServiceAccountToken: false
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa-no-automount
automountServiceAccountToken: false
```

Désactiver la découverte de services

Pour les pods qui n'ont pas besoin de rechercher ou d'appeler des services intégrés au cluster, vous pouvez réduire la quantité d'informations fournies à un pod. Vous pouvez définir la politique DNS du pod de manière à ne pas utiliser CoreDNS et à ne pas exposer les services de l'espace de noms du pod en tant que variables d'environnement. Consultez la documentation Kubernetes sur les variables d'environnement pour plus d'informations sur les liens de service. La valeur par défaut de la politique DNS d'un pod est « ClusterFirst » qui utilise le DNS intégré au cluster, tandis que la valeur autre que la valeur par défaut « Default » utilise la résolution DNS du nœud sous-jacent. Consultez la documentation de Kubernetes sur la politique Pod DNS pour plus d'informations.

Example

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-no-service-info
spec:
    dnsPolicy: Default # "Default" is not the true default value
    enableServiceLinks: false
```

Configurez vos images avec un système de fichiers racine en lecture seule

La configuration de vos images avec un système de fichiers racine en lecture seule empêche un attaquant de remplacer un fichier binaire sur le système de fichiers utilisé par votre application. Si votre application doit écrire dans le système de fichiers, pensez à écrire dans un répertoire temporaire ou à attacher et monter un volume. Vous pouvez appliquer cela en configurant les modules SecurityContext comme suit :

```
...
securityContext:
  readOnlyRootFilesystem: true
...
```

Policy-as-code et les normes de sécurité des pods peuvent être utilisées pour appliquer ce comportement.

Example

Conformément à Windows, les conteneurs de Kubernetes `securityContext.readOnlyRootFilesystem` ne peuvent pas être définis sur un conteneur exécuté sous Windows car un accès en écriture est requis pour que les processus du registre et du système s'exécutent à l'intérieur du conteneur. `true`

Outils et ressources

- Atelier d'immersion sur la sécurité Amazon EKS - Pod Security
- open-policy-agent/gatekeeper-library : La bibliothèque de politiques OPA Gatekeeper est une bibliothèque de OPA/Gatekeeper politiques que vous pouvez utiliser comme substitut. PSPs
- Bibliothèque des politiques de Kyverno
- Une collection de politiques communes de l'OPA et de Kyverno pour EKS.
- Contre-mesures basées sur les politiques : partie 1
- Contre-mesures basées sur les politiques : partie 2
- Pod Security Policy Migrator est un outil qui se convertit en PSPs politiques OPA/Gatekeeper ou Kyverno KubeWarden
- NeuVector de SUSE Open Source, plate-forme de sécurité des conteneurs Zero-Trust, fournit des politiques de processus et de système de fichiers ainsi que des règles de contrôle d'admission.

Isolement des locataires

Lorsque nous pensons à la mutualisation, nous voulons souvent isoler un utilisateur ou une application des autres utilisateurs ou des applications exécutées sur une infrastructure partagée.

Kubernetes est un orchestrateur à locataire unique, c'est-à-dire qu'une seule instance du plan de contrôle est partagée entre tous les locataires d'un cluster. Il existe cependant différents objets Kubernetes que vous pouvez utiliser pour créer un semblant de mutualisation. Par exemple, les espaces de noms et les contrôles d'accès basés sur les rôles (RBAC) peuvent être mis en œuvre pour isoler logiquement les locataires les uns des autres. De même, les quotas et les plages de limites peuvent être utilisés pour contrôler la quantité de ressources du cluster que chaque locataire peut consommer. Néanmoins, le cluster est la seule construction qui fournit une limite de sécurité

solide. En effet, un attaquant qui parvient à accéder à un hôte au sein du cluster peut récupérer tous ConfigMaps les secrets et volumes montés sur cet hôte. Ils pourraient également se faire passer pour le Kubelet, ce qui leur permettrait de manipuler les attributs du nœud et de and/or se déplacer latéralement au sein du cluster.

Les sections suivantes expliquent comment implémenter l'isolation des locataires tout en atténuant les risques liés à l'utilisation d'un seul orchestrateur de locataires tel que Kubernetes.

Multi-location souple

Avec la mutualisation souple, vous utilisez des structures Kubernetes natives, par exemple des espaces de noms, des rôles et des liaisons de rôles, ainsi que des politiques réseau, pour créer une séparation logique entre les locataires. Le RBAC, par exemple, peut empêcher les locataires d'accéder aux ressources des autres ou de les manipuler. Les quotas et les plages de limites contrôlent la quantité de ressources du cluster que chaque locataire peut consommer, tandis que les politiques réseau peuvent empêcher les applications déployées dans différents espaces de noms de communiquer entre elles.

Cependant, aucun de ces contrôles n'empêche les pods de différents locataires de partager un nœud. Si une isolation renforcée est requise, vous pouvez utiliser un sélecteur de nœuds, des règles anti-affinité, des and/or contraintes et des tolérances pour obliger les pods de différents locataires à être planifiés sur des nœuds distincts, souvent appelés nœuds à locataire unique. Cela peut devenir assez compliqué et coûteux dans un environnement comptant de nombreux locataires.

> **Important**
>
> La mutualisation souple mise en œuvre avec les espaces de noms ne vous permet pas de fournir aux locataires une liste filtrée d'espaces de noms, car les espaces de noms sont un type à portée globale. Si un locataire a la possibilité de consulter un espace de noms particulier, il peut afficher tous les espaces de noms du cluster.

> **Warning**
>
> Avec soft-multi-tenancy, les locataires conservent la possibilité d'interroger CoreDNS pour tous les services exécutés par défaut au sein du cluster. Un attaquant pourrait exploiter cela en exécutant dig SRV `..svc.cluster.local` à partir de n'importe quel pod du cluster. Si vous devez restreindre l'accès aux enregistrements DNS des services exécutés au sein

de vos clusters, pensez à utiliser les plug-ins Firewall ou Policy pour CoreDNS. Pour plus d'informations, consultez https://github.com/coredns/ policy# kubernetes-metadata-multi-tenancy -policy.

Kiosk est un projet open source qui peut aider à la mise en œuvre de la mutualisation souple. Il est implémenté sous la forme d'une série de contrôleurs CRDs et fournissant les fonctionnalités suivantes :

- Comptes et utilisateurs de comptes pour séparer les locataires d'un cluster Kubernetes partagé
- Provisionnement d'espaces de noms en libre-service pour les utilisateurs du compte
- Limites de compte pour garantir la qualité de service et l'équité lors du partage d'un cluster
- Modèles d'espaces de noms pour l'isolation sécurisée des locataires et l'initialisation d'espaces de noms en libre-service

Loft est une offre commerciale des mainteneurs de Kiosk DevSpacequi ajoute les fonctionnalités suivantes :

- Accès multi-clusters pour accorder l'accès aux espaces de différents clusters
- Le mode veille réduit les déploiements dans un espace pendant les périodes d'inactivité
- Authentification unique avec des fournisseurs d'authentification OIDC tels que GitHub

Trois principaux cas d'utilisation peuvent être traités par la mutualisation souple.

Paramètre d'entreprise

Le premier est dans un environnement d'entreprise où les « locataires » sont semi-fiables dans la mesure où ils sont des employés, des sous-traitants ou qu'ils sont autrement autorisés par l'organisation. Chaque locataire s'alignera généralement sur une division administrative telle qu'un département ou une équipe.

Dans ce type de configuration, un administrateur de cluster est généralement chargé de créer des espaces de noms et de gérer les politiques. Ils peuvent également mettre en œuvre un modèle d'administration déléguée dans lequel certaines personnes sont chargées de superviser un espace de noms, leur permettant d'effectuer des opérations CRUD pour des objets non liés aux politiques, tels que des déploiements, des services, des modules, des tâches, etc.

L'isolation fournie par l'environnement d'exécution d'un conteneur peut être acceptable dans ce paramètre ou il peut être nécessaire de la renforcer par des contrôles supplémentaires pour la sécurité du pod. Il peut également être nécessaire de restreindre la communication entre les services de différents espaces de noms si une isolation plus stricte est requise.

Kubernetes en tant que service

En revanche, la mutualisation souple peut être utilisée dans les environnements où vous souhaitez proposer Kubernetes en tant que service (KaaS). Avec le KaaS, votre application est hébergée dans un cluster partagé avec un ensemble de contrôleurs CRDs fournissant un ensemble de services PaaS. Les locataires interagissent directement avec le serveur d'API Kubernetes et sont autorisés à effectuer des opérations CRUD sur des objets non liés aux politiques. Il existe également un élément de libre-service dans la mesure où les locataires peuvent être autorisés à créer et à gérer leurs propres espaces de noms. Dans ce type d'environnement, les locataires sont supposés exécuter du code non fiable.

Pour isoler les locataires dans ce type d'environnement, vous devrez probablement mettre en œuvre des politiques réseau strictes ainsi que le sandboxing des pods. Le sandboxing consiste à exécuter les conteneurs d'un pod dans une micro-machine virtuelle telle que Firecracker ou dans un noyau d'espace utilisateur. Aujourd'hui, vous pouvez créer des pods en bac à sable avec EKS Fargate.

Logiciel en tant que service (SaaS)

Le dernier cas d'utilisation de la mutualisation souple se situe dans un environnement (Software-as-a-ServiceSaaS). Dans cet environnement, chaque locataire est associé à une instance particulière d'une application exécutée au sein du cluster. Chaque instance possède souvent ses propres données et utilise des contrôles d'accès distincts qui sont généralement indépendants du RBAC de Kubernetes.

Contrairement aux autres cas d'utilisation, le locataire d'un environnement SaaS n'interagit pas directement avec l'API Kubernetes. L'application SaaS est plutôt chargée de l'interfaçage avec l'API Kubernetes afin de créer les objets nécessaires pour prendre en charge chaque locataire.

Constructions Kubernetes

Dans chacun de ces cas, les constructions suivantes sont utilisées pour isoler les locataires les uns des autres :

Espaces de noms

Les espaces de noms sont essentiels à la mise en œuvre de la mutualisation souple. Ils vous permettent de diviser le cluster en partitions logiques. Les quotas, les politiques réseau, les comptes de service et les autres objets nécessaires à la mise en œuvre de la mutualisation sont limités à un espace de noms.

Stratégies réseau

Par défaut, tous les pods d'un cluster Kubernetes sont autorisés à communiquer entre eux. Ce comportement peut être modifié à l'aide de politiques réseau.

Les politiques réseau limitent la communication entre les pods à l'aide d'étiquettes ou de plages d'adresses IP. Dans un environnement multi-tenant où une isolation réseau stricte entre les locataires est requise, nous recommandons de commencer par une règle par défaut qui refuse la communication entre les pods, et une autre règle qui autorise tous les pods à interroger le serveur DNS pour la résolution de noms. Une fois cela en place, vous pouvez commencer à ajouter des règles plus permissives qui permettent la communication au sein d'un espace de noms. Cela peut être affiné selon les besoins.

> **Note**
>
> Amazon VPC CNI prend désormais en charge les politiques réseau Kubernetes afin de créer des politiques capables d'isoler les charges de travail sensibles et de les protéger contre tout accès non autorisé lors de l'exécution de Kubernetes sur AWS. Cela signifie que vous pouvez utiliser toutes les fonctionnalités de l'API Network Policy au sein de votre cluster Amazon EKS. Ce niveau de contrôle granulaire vous permet de mettre en œuvre le principe du moindre privilège, qui garantit que seuls les pods autorisés sont autorisés à communiquer entre eux.

> **Important**
>
> Les politiques de réseau sont nécessaires mais ne sont pas suffisantes. L'application des politiques réseau nécessite un moteur de politiques tel que Calico ou Cilium.

Contrôle d’accès basé sur les rôles (RBAC)

Les rôles et les liaisons de rôles sont les objets Kubernetes utilisés pour appliquer le contrôle d'accès basé sur les rôles (RBAC) dans Kubernetes. Les rôles contiennent des listes d'actions qui peuvent être effectuées sur des objets de votre cluster. Les liaisons de rôles spécifient les individus ou les groupes auxquels les rôles s'appliquent. Dans les paramètres d'entreprise et KaaS, le RBAC peut être utilisé pour permettre l'administration d'objets par des groupes ou des individus sélectionnés.

Quotas

Les quotas sont utilisés pour définir les limites des charges de travail hébergées dans votre cluster. Avec les quotas, vous pouvez spécifier la quantité maximale de processeur et de mémoire qu'un pod peut consommer, ou vous pouvez limiter le nombre de ressources pouvant être allouées dans un cluster ou un espace de noms. Les plages de limites vous permettent de déclarer des valeurs minimales, maximales et par défaut pour chaque limite.

Il est souvent avantageux de surengager des ressources dans un cluster partagé, car cela vous permet de maximiser vos ressources. Cependant, un accès illimité à un cluster peut entraîner une privation de ressources, ce qui peut entraîner une dégradation des performances et une perte de disponibilité des applications. Si les demandes d'un pod sont définies à un niveau trop faible et que l'utilisation réelle des ressources dépasse la capacité du nœud, le nœud commencera à subir une pression sur le processeur ou la mémoire. Dans ce cas, les pods peuvent être redémarrés et and/or expulsés du nœud.

Pour éviter que cela ne se produise, vous devez prévoir d'imposer des quotas sur les espaces de noms dans un environnement multi-locataires afin de forcer les locataires à spécifier des demandes et des limites lors de la planification de leurs pods sur le cluster. Cela permettra également d'atténuer un éventuel déni de service en limitant la quantité de ressources qu'un pod peut consommer.

Vous pouvez également utiliser des quotas pour répartir les ressources du cluster afin de les aligner sur les dépenses du locataire. Cela est particulièrement utile dans le scénario KaaS.

Priorité et préemption des pods

La priorité et la préemption des pods peuvent être utiles lorsque vous souhaitez accorder plus d'importance à un pod par rapport aux autres pods. Par exemple, avec la priorité des pods, vous pouvez configurer les pods du client A pour qu'ils fonctionnent avec une priorité plus élevée que le client B. Lorsque la capacité disponible est insuffisante, le planificateur expulse les pods les moins prioritaires du client B pour accueillir les pods les plus prioritaires du client A. Cela peut être

particulièrement pratique dans un environnement SaaS où les clients prêts à payer un supplément reçoivent une priorité plus élevée.

> **Important**
>
> La priorité des pods peut avoir un effet indésirable sur les autres pods moins prioritaires. Par exemple, bien que les modules concernés soient résiliés gracieusement mais que cela ne PodDisruptionBudget soit pas garanti, cela pourrait endommager une application de moindre priorité qui dépend d'un quorum de pods, voir Limitations de préemption.

Contrôles d'atténuation

En tant qu'administrateur d'un environnement mutualisé, votre principale préoccupation est d'empêcher un attaquant d'accéder à l'hôte sous-jacent. Les contrôles suivants doivent être envisagés pour atténuer ce risque :

Environnements d'exécution en sandbox pour les conteneurs

Le sandboxing est une technique par laquelle chaque conteneur est exécuté sur sa propre machine virtuelle isolée. Les technologies qui permettent de réaliser le sandboxing en nacelle incluent Firecracker et Weave's Firekube.

Pour plus d'informations sur les efforts visant à faire de Firecracker un environnement d'exécution compatible pour EKS, consultez le fichier 1238496944684597248.html. https://threadreaderapp.com/thread/

Agent de politique ouverte (OPA) et contrôleur d'accès

Gatekeeper est un contrôleur d'admission Kubernetes qui applique les politiques créées avec OPA. Avec OPA, vous pouvez créer une politique qui exécute les pods provenant de locataires sur des instances distinctes ou avec une priorité plus élevée que celle des autres locataires. Une collection de politiques OPA communes se trouve dans le GitHub référentiel de ce projet.

Il existe également un plugin OPA expérimental pour CoreDNS qui vous permet d'utiliser OPA filter/control pour les enregistrements renvoyés par CoreDNS.

Kyverno

Kyverno est un moteur de politique natif de Kubernetes capable de valider, de muter et de générer des configurations avec des politiques en tant que ressources Kubernetes. Kyverno utilise des superpositions de type Kustomize pour la validation, prend en charge le correctif JSON et le correctif de fusion stratégique pour les mutations, et peut cloner des ressources dans des espaces de noms en fonction de déclencheurs flexibles.

Vous pouvez utiliser Kyverno pour isoler les espaces de noms, renforcer la sécurité des pods et appliquer d'autres bonnes pratiques, et générer des configurations par défaut telles que des politiques réseau. Plusieurs exemples sont inclus dans le GitHub référentiel de ce projet. De nombreuses autres politiques sont incluses dans la bibliothèque des politiques sur le site Web de Kyverno.

Isolation des charges de travail des locataires sur des nœuds spécifiques

Le fait de restreindre les charges de travail des locataires pour qu'elles s'exécutent sur des nœuds spécifiques peut être utilisé pour renforcer l'isolation dans le modèle de mutualisation souple. Avec cette approche, les charges de travail spécifiques aux locataires ne sont exécutées que sur des nœuds provisionnés pour les locataires respectifs. Pour obtenir cette isolation, les propriétés natives de Kubernetes (affinité des nœuds, contraintes et tolérances) sont utilisées pour cibler des nœuds spécifiques pour la planification des pods et empêcher les pods, provenant d'autres locataires, d'être planifiés sur les nœuds spécifiques aux locataires.

Partie 1 - Affinité des nœuds

L'affinité des nœuds Kubernetes est utilisée pour cibler les nœuds à des fins de planification, en fonction des étiquettes des nœuds. Avec les règles d'affinité des nœuds, les pods sont attirés par des nœuds spécifiques qui correspondent aux termes du sélecteur. Dans la spécification du pod ci-dessous, l'affinité des `requiredDuringSchedulingIgnoredDuringExecution` nœuds est appliquée au pod correspondant. Le résultat est que le pod ciblera les nœuds étiquetés avec la clé/valeur suivante :. `node-restriction.kubernetes.io/tenant: tenants-x`

```
...
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
```

```
        - matchExpressions:
          - key: node-restriction.kubernetes.io/tenant
            operator: In
            values:
            - tenants-x
...
```

Avec cette affinité de nœud, l'étiquette est requise lors de la planification, mais pas pendant l'exécution ; si les étiquettes des nœuds sous-jacents changent, les pods ne seront pas expulsés uniquement en raison de ce changement d'étiquette. Cependant, la planification future pourrait être affectée.

> **Warning**
>
> Le préfixe d'étiquette `node-restriction.kubernetes.io/` a une signification particulière dans Kubernetes. NodeRestrictionqui est activé pour les clusters EKS, `kubelet` empêche adding/removing/updating les étiquettes comportant ce préfixe. Les attaquants ne peuvent pas utiliser le `kubelet's credentials to update the node object or modify the system setup to pass these labels into` `kubelet` car ils `kubelet` ne sont pas autorisés à modifier ces étiquettes. Si ce préfixe est utilisé pour toute la planification entre un pod et un nœud, il permet d'éviter les scénarios dans lesquels un attaquant pourrait vouloir attirer un ensemble différent de charges de travail vers un nœud en modifiant les étiquettes des nœuds.

Example

Au lieu de l'affinité des nœuds, nous aurions pu utiliser le sélecteur de nœuds. Cependant, l'affinité des nœuds est plus expressive et permet de prendre en compte davantage de conditions lors de la planification des pods. Pour plus d'informations sur les différences et les choix de planification plus avancés, consultez ce billet de blog de la CNCF sur la planification avancée d'un pod à un nœud Kubernetes.

Partie 2 - Taches et tolérances

Attirer les pods vers les nœuds n'est que la première partie de cette approche en trois parties. Pour que cette approche fonctionne, nous devons empêcher les pods de planifier sur des nœuds pour lesquels les pods ne sont pas autorisés. Pour repousser les pods indésirables ou non autorisés, Kubernetes utilise des nœuds souillés. Les taches sont utilisées pour créer des conditions sur les

nœuds qui empêchent la planification des pods. La tache ci-dessous utilise une paire clé-valeur de. `tenant: tenants-x`

```
...
    taints:
      - key: tenant
        value: tenants-x
        effect: NoSchedule
...
```

Compte tenu du nœud ci-dessus`taint`, seuls les pods qui tolèrent cette odeur seront autorisés à être planifiés sur le nœud. Pour permettre de programmer des pods autorisés sur le nœud, les spécifications des pods respectifs doivent inclure un `toleration` point de contact, comme indiqué ci-dessous.

```
...
  tolerations:
  - effect: NoSchedule
    key: tenant
    operator: Equal
    value: tenants-x
...
```

Les pods présentant les caractéristiques ci-dessus ne `toleration` seront pas empêchés de planifier sur le nœud, du moins pas à cause de cette odeur spécifique. Kubernetes utilise également Taints pour arrêter temporairement la planification des pods dans certaines conditions, telles que la pression sur les ressources des nœuds. Grâce à l'affinité des nœuds, aux taches et aux tolérances, nous pouvons attirer efficacement les pods souhaités vers des nœuds spécifiques et repousser les pods indésirables.

> **Important**
>
> Certains pods Kubernetes sont nécessaires pour fonctionner sur tous les nœuds. Des exemples de ces pods sont ceux démarrés par le Container Network Interface (CNI) et les daemonsets kube-proxy. À cette fin, les spécifications de ces capsules contiennent des tolérances très permissives, afin de tolérer différentes odeurs. Il faut veiller à ne pas modifier ces tolérances. La modification de ces tolérances peut entraîner un fonctionnement incorrect du cluster. En outre, les outils de gestion des politiques, tels que OPA/Gatekeeper

et Kyverno, peuvent être utilisés pour rédiger des politiques de validation qui empêchent les pods non autorisés d'utiliser ces tolérances permissives.

Partie 3 - Gestion basée sur des politiques pour la sélection des nœuds

Plusieurs outils peuvent être utilisés pour aider à gérer l'affinité des nœuds et les tolérances des spécifications des pods, y compris l'application des règles dans les pipelines CICD. Cependant, l'application de l'isolation doit également être effectuée au niveau du cluster Kubernetes. À cette fin, les outils de gestion des politiques peuvent être utilisés pour modifier les demandes entrantes du serveur d'API Kubernetes, en fonction des charges utiles des demandes, afin d'appliquer les règles d'affinité des nœuds et les tolérances respectives mentionnées ci-dessus.

Par exemple, les pods destinés à l'espace de noms tenants-x peuvent être estampillés avec l'affinité et la tolérance de nœuds correctes afin de permettre la planification sur les nœuds tenants-x. À l'aide des outils de gestion des politiques configurés à l'aide du Kubernetes Mutating Admission Webhook, les politiques peuvent être utilisées pour modifier les spécifications des pods entrants. Les mutations ajoutent les éléments nécessaires pour permettre la planification souhaitée. Un exemple OPA/Gatekeeper de politique qui ajoute une affinité de nœud est présenté ci-dessous.

```
apiVersion: mutations.gatekeeper.sh/v1alpha1
kind: Assign
metadata:
  name: mutator-add-nodeaffinity-pod
  annotations:
    aws-eks-best-practices/description: >-
      Adds Node affinity - https://kubernetes.io/docs/concepts/scheduling-eviction/
assign-pod-node/#node-affinity
spec:
  applyTo:
  - groups: [""]
    kinds: ["Pod"]
    versions: ["v1"]
  match:
    namespaces: ["tenants-x"]
  location:
 "spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.nodeSelectorTerms"
  parameters:
    assign:
      value:
        - matchExpressions:
```

```
          - key: "tenant"
            operator: In
            values:
            - "tenants-x"
```

La politique ci-dessus est appliquée à une demande du serveur d'API Kubernetes, afin d'appliquer un pod à l'espace de noms tenants-x. La politique ajoute la règle d'affinité des `requiredDuringSchedulingIgnoredDuringExecution` nœuds, afin que les pods soient attirés par les nœuds portant l'`tenant: tenants-x`étiquette.

Une deuxième politique, présentée ci-dessous, ajoute la tolérance à la même spécification de pod, en utilisant les mêmes critères de correspondance pour l'espace de noms cible et les groupes, types et versions.

```
apiVersion: mutations.gatekeeper.sh/v1alpha1
kind: Assign
metadata:
  name: mutator-add-toleration-pod
  annotations:
    aws-eks-best-practices/description: >-
      Adds toleration - https://kubernetes.io/docs/concepts/scheduling-eviction/taint-
and-toleration/
spec:
  applyTo:
  - groups: [""]
    kinds: ["Pod"]
    versions: ["v1"]
  match:
    namespaces: ["tenants-x"]
  location: "spec.tolerations"
  parameters:
    assign:
      value:
      - key: "tenant"
        operator: "Equal"
        value: "tenants-x"
        effect: "NoSchedule"
```

Les politiques ci-dessus sont spécifiques aux modules ; cela est dû aux chemins d'accès aux éléments mutés dans les éléments des politiques. `location` Des politiques supplémentaires pourraient être rédigées pour gérer les ressources qui créent des pods, telles que les ressources

Deployment et Job. Les politiques répertoriées et d'autres exemples peuvent être consultés dans le GitHubprojet complémentaire de ce guide.

Le résultat de ces deux mutations est que les gousses sont attirées par le nœud souhaité, sans pour autant être repoussées par la souillure spécifique du nœud. Pour vérifier cela, nous pouvons voir les extraits de sortie de deux `kubectl` appels pour obtenir les nœuds étiquetés avec et pour obtenir `tenant=tenants-x` les pods dans l'`tenants-x`espace de noms.

```
kubectl get nodes -l tenant=tenants-x
NAME
ip-10-0-11-255...
ip-10-0-28-81...
ip-10-0-43-107...

kubectl -n tenants-x get pods -owide
NAME                                  READY   STATUS    RESTARTS   AGE   IP
 NODE
tenant-test-deploy-58b895ff87-2q7xw   1/1     Running   0          13s   10.0.42.143
 ip-10-0-43-107...
tenant-test-deploy-58b895ff87-9b6hg   1/1     Running   0          13s   10.0.18.145
 ip-10-0-28-81...
tenant-test-deploy-58b895ff87-nxvw5   1/1     Running   0          13s   10.0.30.117
 ip-10-0-28-81...
tenant-test-deploy-58b895ff87-vw796   1/1     Running   0          13s   10.0.3.113
 ip-10-0-11-255...
tenant-test-pod                       1/1     Running   0          13s   10.0.35.83
 ip-10-0-43-107...
```

Comme nous pouvons le voir sur les résultats ci-dessus, tous les pods sont planifiés sur les nœuds étiquetés avec`tenant=tenants-x`. En termes simples, les pods ne fonctionneront que sur les nœuds souhaités, tandis que les autres pods (sans l'affinité et les tolérances requises) ne le feront pas. Les charges de travail des locataires sont efficacement isolées.

Un exemple de spécification de pod muté est présenté ci-dessous.

```
apiVersion: v1
kind: Pod
metadata:
  name: tenant-test-pod
  namespace: tenants-x
spec:
  affinity:
```

```
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: tenant
            operator: In
            values:
            - tenants-x
...
  tolerations:
  - effect: NoSchedule
    key: tenant
    operator: Equal
    value: tenants-x
...
```

> **Important**
>
> Les outils de gestion des politiques intégrés au flux de demandes du serveur d'API Kubernetes, à l'aide de webhooks d'admission mutants et validants, sont conçus pour répondre à la demande du serveur d'API dans un délai spécifié. Cette durée est généralement inférieure ou égale à 3 secondes. Si l'appel webhook ne renvoie pas de réponse dans le délai configuré, la and/or validation de la mutation de la demande du serveur d'API entrante peut avoir lieu ou non. Ce comportement dépend du fait que les configurations du webhook d'admission sont définies sur Fail Open ou Fail Close.

Dans les exemples ci-dessus, nous avons utilisé des politiques écrites pour OPA/Gatekeeper. Cependant, il existe d'autres outils de gestion des politiques qui gèrent également notre cas d'utilisation de la sélection de nœuds. Par exemple, cette politique Kyverno pourrait être utilisée pour gérer la mutation d'affinité des nœuds.

> **Note**
>
> Si elles fonctionnent correctement, les politiques mutantes apporteront les modifications souhaitées aux charges utiles des demandes du serveur d'API entrantes. Cependant, des politiques de validation doivent également être incluses pour vérifier que les modifications souhaitées se produisent, avant que les modifications ne soient autorisées à persister. Cela est particulièrement important lorsque vous utilisez ces politiques à des fins tenant-

to-node d'isolation. Il est également conseillé d'inclure des politiques d'audit pour vérifier régulièrement la présence de configurations indésirables dans votre cluster.

Références

- k-rail Conçu pour vous aider à sécuriser un environnement multi-locataires grâce à l'application de certaines politiques.
- Pratiques de sécurité pour les applications MultiTenant SaaS utilisant Amazon EKS

Multi-location rigide

La mutualisation stricte peut être mise en œuvre en provisionnant des clusters distincts pour chaque locataire. Bien que cela assure un très fort isolement entre les locataires, cela présente plusieurs inconvénients.

Tout d'abord, lorsque vous avez de nombreux locataires, cette approche peut rapidement devenir coûteuse. Non seulement vous devrez payer les coûts du plan de contrôle pour chaque cluster, mais vous ne pourrez pas partager les ressources de calcul entre les clusters. Cela finira par provoquer une fragmentation dans laquelle un sous-ensemble de vos clusters sera sous-utilisé tandis que d'autres seront surutilisés.

Ensuite, vous devrez probablement acheter ou créer des outils spéciaux pour gérer tous ces clusters. Avec le temps, la gestion de centaines ou de milliers de clusters peut tout simplement devenir trop complexe.

Enfin, la création d'un cluster par locataire sera lente par rapport à la création d'un espace de noms. Néanmoins, une approche de location rigide peut être nécessaire dans les secteurs hautement réglementés ou dans les environnements SaaS où une forte isolation est requise.

Orientations futures

La communauté Kubernetes a reconnu les lacunes actuelles de la mutualisation souple et les défis liés à la mutualisation stricte. Le Multi-Tenancy Special Interest Group (SIG) tente de remédier à ces lacunes par le biais de plusieurs projets d'incubation, notamment le contrôleur hiérarchique des espaces de noms (HNC) et le cluster virtuel.

La proposition HNC (KEP) décrit un moyen de créer des relations parent-enfant entre les espaces de noms grâce à l'héritage d'objets [policy] ainsi qu'à la possibilité pour les administrateurs locataires de créer des sous-espaces de noms.

La proposition de cluster virtuel décrit un mécanisme permettant de créer des instances distinctes des services du plan de contrôle, y compris le serveur d'API, le gestionnaire de contrôleurs et le planificateur, pour chaque locataire du cluster (également connu sous le nom de « Kubernetes on Kubernetes »).

La proposition Multi-Tenancy Benchmarks fournit des directives pour le partage de clusters à l'aide d'espaces de noms pour l'isolation et la segmentation, ainsi qu'un outil de ligne de commande kubectl-mtb pour valider la conformité aux directives.

Outils et ressources de gestion multi-clusters

- Cloud Banzaï
- Commandant
- Lentille
- Nirmata
- Rafay
- éleveur
- Flux de tissage

Audit et journalisation

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

La collecte et l'analyse des journaux [d'audit] sont utiles pour différentes raisons. Les journaux peuvent aider à analyser et à attribuer les causes premières, c'est-à-dire à attribuer un changement à un utilisateur en particulier. Lorsque suffisamment de journaux ont été collectés, ils peuvent également être utilisés pour détecter les comportements anormaux. Sur EKS, les journaux d'audit sont envoyés à Amazon Cloudwatch Logs. La politique d'audit d'EKS est la suivante :

```
apiVersion: audit.k8s.io/v1beta1
```

```
kind: Policy
rules:
  # Log full request and response for changes to aws-auth ConfigMap in kube-system
 namespace
  - level: RequestResponse
    namespaces: ["kube-system"]
    verbs: ["update", "patch", "delete"]
    resources:
      - group: "" # core
        resources: ["configmaps"]
        resourceNames: ["aws-auth"]
    omitStages:
      - "RequestReceived"
  # Do not log watch operations performed by kube-proxy on endpoints and services
  - level: None
    users: ["system:kube-proxy"]
    verbs: ["watch"]
    resources:
      - group: "" # core
        resources: ["endpoints", "services", "services/status"]
  # Do not log get operations performed by kubelet on nodes and their statuses
  - level: None
    users: ["kubelet"] # legacy kubelet identity
    verbs: ["get"]
    resources:
      - group: "" # core
        resources: ["nodes", "nodes/status"]
  # Do not log get operations performed by the system:nodes group on nodes and their
 statuses
  - level: None
    userGroups: ["system:nodes"]
    verbs: ["get"]
    resources:
      - group: "" # core
        resources: ["nodes", "nodes/status"]
  # Do not log get and update operations performed by controller manager, scheduler,
 and endpoint-controller on endpoints in kube-system namespace
  - level: None
    users:
      - system:kube-controller-manager
      - system:kube-scheduler
      - system:serviceaccount:kube-system:endpoint-controller
    verbs: ["get", "update"]
    namespaces: ["kube-system"]
```

```
    resources:
      - group: "" # core
        resources: ["endpoints"]
  # Do not log get operations performed by apiserver on namespaces and their statuses/
finalizations
  - level: None
    users: ["system:apiserver"]
    verbs: ["get"]
    resources:
      - group: "" # core
        resources: ["namespaces", "namespaces/status", "namespaces/finalize"]
  # Do not log get and list operations performed by controller manager on
 metrics.k8s.io resources
  - level: None
    users:
      - system:kube-controller-manager
    verbs: ["get", "list"]
    resources:
      - group: "metrics.k8s.io"
  # Do not log access to health, version, and swagger non-resource URLs
  - level: None
    nonResourceURLs:
      - /healthz*
      - /version
      - /swagger*
  # Do not log events resources
  - level: None
    resources:
      - group: "" # core
        resources: ["events"]
  # Log request for updates/patches to nodes and pods statuses by kubelet and node
 problem detector
  - level: Request
    users: ["kubelet", "system:node-problem-detector", "system:serviceaccount:kube-
system:node-problem-detector"]
    verbs: ["update", "patch"]
    resources:
      - group: "" # core
        resources: ["nodes/status", "pods/status"]
    omitStages:
      - "RequestReceived"
  # Log request for updates/patches to nodes and pods statuses by system:nodes group
  - level: Request
    userGroups: ["system:nodes"]
```

```
    verbs: ["update", "patch"]
    resources:
      - group: "" # core
        resources: ["nodes/status", "pods/status"]
    omitStages:
      - "RequestReceived"
  # Log delete collection requests by namespace-controller in kube-system namespace
  - level: Request
    users: ["system:serviceaccount:kube-system:namespace-controller"]
    verbs: ["deletecollection"]
    omitStages:
      - "RequestReceived"
  # Log metadata for secrets, configmaps, and tokenreviews to protect sensitive data
  - level: Metadata
    resources:
      - group: "" # core
        resources: ["secrets", "configmaps"]
      - group: authentication.k8s.io
        resources: ["tokenreviews"]
    omitStages:
      - "RequestReceived"
  # Log requests for serviceaccounts/token resources
  - level: Request
    resources:
      - group: "" # core
        resources: ["serviceaccounts/token"]
  # Log get, list, and watch requests for various resource groups
  - level: Request
    verbs: ["get", "list", "watch"]
    resources:
      - group: "" # core
      - group: "admissionregistration.k8s.io"
      - group: "apiextensions.k8s.io"
      - group: "apiregistration.k8s.io"
      - group: "apps"
      - group: "authentication.k8s.io"
      - group: "authorization.k8s.io"
      - group: "autoscaling"
      - group: "batch"
      - group: "certificates.k8s.io"
      - group: "extensions"
      - group: "metrics.k8s.io"
      - group: "networking.k8s.io"
      - group: "policy"
```

```
      - group: "rbac.authorization.k8s.io"
      - group: "scheduling.k8s.io"
      - group: "settings.k8s.io"
      - group: "storage.k8s.io"
    omitStages:
      - "RequestReceived"
  # Default logging level for known APIs to log request and response
  - level: RequestResponse
    resources:
      - group: "" # core
      - group: "admissionregistration.k8s.io"
      - group: "apiextensions.k8s.io"
      - group: "apiregistration.k8s.io"
      - group: "apps"
      - group: "authentication.k8s.io"
      - group: "authorization.k8s.io"
      - group: "autoscaling"
      - group: "batch"
      - group: "certificates.k8s.io"
      - group: "extensions"
      - group: "metrics.k8s.io"
      - group: "networking.k8s.io"
      - group: "policy"
      - group: "rbac.authorization.k8s.io"
      - group: "scheduling.k8s.io"
      - group: "settings.k8s.io"
      - group: "storage.k8s.io"
    omitStages:
      - "RequestReceived"
  # Default logging level for all other requests to log metadata only
  - level: Metadata
    omitStages:
      - "RequestReceived"
```

Recommandations

Activer les journaux d'audit

Les journaux d'audit font partie des journaux du plan de contrôle Kubernetes gérés par EKS et gérés par EKS. enabling/disabling Les instructions relatives aux journaux du plan de contrôle, qui incluent les journaux du serveur d'API Kubernetes, du gestionnaire de contrôleurs et du planificateur, ainsi

que le journal d'audit, se trouvent ici, -plane-logs.html# -export. https://docs.aws.amazon.com/eks/latest/userguide/control enabling-control-plane-log

> **Note**
>
> Lorsque vous activez la journalisation sur le plan de contrôle, le stockage des connexions entraîne des frais. CloudWatch Cela soulève un problème plus général concernant le coût permanent de la sécurité. En fin de compte, vous devrez évaluer ces coûts par rapport au coût d'une faille de sécurité, par exemple une perte financière, une atteinte à votre réputation, etc. Vous constaterez peut-être que vous pouvez sécuriser correctement votre environnement en ne mettant en œuvre que certaines des recommandations de ce guide.

> **Warning**
>
> La taille maximale d'une entrée CloudWatch Logs est de 1 Mo, tandis que la taille maximale des demandes d'API Kubernetes est de 1,5 Mo. Les entrées de journal supérieures à 1 Mo seront tronquées ou incluront uniquement les métadonnées de la demande.

Utiliser les métadonnées d'audit

Les journaux d'audit Kubernetes incluent deux annotations qui indiquent si une demande a été autorisée ou non `authorization.k8s.io/decision` et le motif de la décision. `authorization.k8s.io/reason` Utilisez ces attributs pour déterminer pourquoi un appel d'API spécifique a été autorisé.

Créez des alarmes en cas d'événements suspects

Créez une alarme pour vous avertir automatiquement en cas d'augmentation du nombre de 403 réponses interdites et 401 réponses non autorisées, puis utilisez des attributs tels que `hostsourceIPs`, et `k8s_user.username` pour savoir d'où proviennent ces demandes.

Analysez les journaux avec Log Insights

Utilisez CloudWatch Log Insights pour surveiller les modifications apportées aux objets RBAC, par exemple les rôles RoleBindings, ClusterRoles, et ClusterRoleBindings. Quelques exemples de requêtes apparaissent ci-dessous :

Répertorie les mises à jour apportées aux `aws-auth` ConfigMap :

```
fields @timestamp, @message
| filter @logStream like "kube-apiserver-audit"
| filter verb in ["update", "patch"]
| filter objectRef.resource = "configmaps" and objectRef.name = "aws-auth" and
 objectRef.namespace = "kube-system"
| sort @timestamp desc
```

Répertorie les créations de nouveaux webhooks ou les modifications apportées à ces webhooks :

```
fields @timestamp, @message
| filter @logStream like "kube-apiserver-audit"
| filter verb in ["create", "update", "patch"] and responseStatus.code = 201
| filter objectRef.resource = "validatingwebhookconfigurations"
| sort @timestamp desc
```

Répertorie les opérations de création, de mise à jour et de suppression dans les rôles :

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| filter objectRef.resource="roles" and verb in ["create", "update", "patch", "delete"]
```

Répertorie les opérations de création, de mise à jour et de suppression pour RoleBindings :

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| filter objectRef.resource="rolebindings" and verb in ["create", "update", "patch",
 "delete"]
```

Répertorie les opérations de création, de mise à jour et de suppression pour ClusterRoles :

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| filter objectRef.resource="clusterroles" and verb in ["create", "update", "patch",
 "delete"]
```

Répertorie les opérations de création, de mise à jour et de suppression pour ClusterRoleBindings :

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| filter objectRef.resource="clusterrolebindings" and verb in ["create", "update",
 "patch", "delete"]
```

Propose des opérations de lecture non autorisées à l'encontre de Secrets :

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| filter objectRef.resource="secrets" and verb in ["get", "watch", "list"] and
 responseStatus.code="401"
| stats count() by bin(1m)
```

Liste des demandes anonymes ayant échoué :

```
fields @timestamp, @message, sourceIPs.0
| sort @timestamp desc
| limit 100
| filter user.username="system:anonymous" and responseStatus.code in ["401", "403"]
```

Auditez vos CloudTrail journaux

Les AWS APIs appelés par des pods qui utilisent des rôles IAM pour les comptes de service (IRSA) sont automatiquement connectés CloudTrail avec le nom du compte de service. Si le nom d'un compte de service qui n'était pas explicitement autorisé à appeler une API apparaît dans le journal, cela peut indiquer que la politique de confiance du rôle IAM a été mal configurée. D'une manière générale, Cloudtrail est un excellent moyen d'attribuer des appels d'API AWS à des principes IAM spécifiques.

Utilisez CloudTrail Insights pour détecter les activités suspectes

CloudTrail Insights analyse automatiquement les événements de gestion de l'écriture issus des CloudTrail sentiers et vous alerte en cas d'activité inhabituelle. Cela peut vous aider à identifier les cas d'augmentation du volume d'appels lors de l'écriture APIs dans votre compte AWS, notamment en provenance de pods utilisant IRSA pour assumer un rôle IAM. Pour plus d'informations, consultez Annancing CloudTrail Insights : Identification et réponse à une activité d'API inhabituelle.

Ressources supplémentaires

À mesure que le volume des journaux augmente, leur analyse et leur filtrage avec Log Insights ou un autre outil d'analyse des journaux peuvent devenir inefficaces. Comme alternative, vous pouvez envisager d'exécuter Sysdig Falco et ekscloudwatch. Falco analyse les journaux d'audit et signale les anomalies ou les abus sur une longue période. Le projet ekscloudwatch transmet les événements du journal d'audit CloudWatch à Falco pour analyse. Falco fournit un ensemble de règles d'audit par défaut ainsi que la possibilité d'ajouter les vôtres.

Une autre option pourrait être de stocker les journaux d'audit dans S3 et d'utiliser l'algorithme SageMaker Random Cut Forest pour détecter les comportements anormaux nécessitant une enquête plus approfondie.

Outils et ressources

Les projets commerciaux et open source suivants peuvent être utilisés pour évaluer l'alignement de votre cluster sur les meilleures pratiques établies :

- Atelier d'immersion sur la sécurité Amazon EKS - Detective Controls
- kubeaudit
- kube-scan Attribue un score de risque aux charges de travail exécutées dans votre cluster conformément au framework Kubernetes Common Configuration Scoring System
- kubesec.io
- polaris
- tribord
- Snyk
- Kubescape Kubescape est un outil de sécurité Kubernetes open source qui analyse les clusters, les fichiers YAML et les graphiques Helm. Il détecte les erreurs de configuration selon plusieurs frameworks (notamment NSA-CISA et MITRE ATT&CK®).

Sécurité du réseau

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

La sécurité du réseau comporte plusieurs facettes. Le premier concerne l'application de règles qui limitent le flux du trafic réseau entre les services. Le second concerne le chiffrement du trafic pendant son transit. Les mécanismes permettant de mettre en œuvre ces mesures de sécurité sur EKS sont variés mais incluent souvent les éléments suivants :

Contrôle du trafic

- Politiques réseau
- Groupes de sécurité

Chiffrement de réseau

- Maillage de service
- Interfaces réseau de conteneurs (CNIs)
- Contrôleurs d'entrée et équilibreurs de charge
- Instances Nitro
- ACM Private CA avec gestionnaire de certificats

Politique du réseau

Au sein d'un cluster Kubernetes, toutes les communications Pod to Pod sont autorisées par défaut. Bien que cette flexibilité puisse contribuer à promouvoir l'expérimentation, elle n'est pas considérée comme sûre. Les politiques réseau de Kubernetes vous offrent un mécanisme permettant de restreindre le trafic réseau entre les pods (souvent appelé trafic est/ouest) ainsi qu'entre les pods et les services externes. Les politiques réseau Kubernetes s'appliquent aux couches 3 et 4 du modèle OSI. Les politiques réseau utilisent des modules, des sélecteurs d'espace de noms et des étiquettes pour identifier les pods source et de destination, mais peuvent également inclure des adresses IP, des numéros de port, des protocoles ou une combinaison de ces éléments. Les politiques réseau peuvent être appliquées aux connexions entrantes ou sortantes au pod, souvent appelées règles d'entrée et de sortie.

Grâce à la prise en charge native des politiques réseau par le plug-in Amazon VPC CNI, vous pouvez implémenter des politiques réseau pour sécuriser le trafic réseau dans les clusters Kubernetes. Cela s'intègre à l'API Kubernetes Network Policy en amont, garantissant ainsi la compatibilité et le respect des normes Kubernetes. Vous pouvez définir des politiques à l'aide de différents identifiants

pris en charge par l'API en amont. Par défaut, tout le trafic entrant et sortant est autorisé vers un module. Lorsqu'une politique réseau avec une entrée PolicyType est spécifiée, seules les connexions autorisées dans le pod sont celles provenant du nœud du pod et celles autorisées par les règles d'entrée. Il en va de même pour les règles de sortie. Si plusieurs règles sont définies, l'union de toutes les règles est prise en compte lors de la prise de décision. Ainsi, l'ordre d'évaluation n'a aucune incidence sur le résultat de la politique.

> **Important**
>
> Lorsque vous provisionnez un cluster EKS pour la première fois, la fonctionnalité VPC CNI Network Policy n'est pas activée par défaut. Assurez-vous d'avoir déployé la version du module complémentaire VPC CNI prise en charge et définissez l'`ENABLE_NETWORK_POLICY`indicateur `true` sur le module complémentaire vpc-cni pour l'activer. Consultez le guide de l'utilisateur Amazon EKS pour obtenir des instructions détaillées.

Recommandations

Débuter avec les politiques réseau - Suivez le principe du moindre privilège

Création d'une politique de refus par défaut

Comme pour les politiques RBAC, il est recommandé de suivre les principes d'accès les moins privilégiés en matière de politiques réseau. Commencez par créer une politique de refus total qui restreint tout le trafic entrant et sortant dans un espace de noms.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
  namespace: default
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

default-deny (refus par défaut)

		Target		Source				
#	Policy name	Namespace	Pods	Namespace	Pods	Subnet	Ports	
1	default-deny	default	Any	Any	Any	Any	Any	

> **Note**
>
> L'image ci-dessus a été créée par le visualiseur de politiques réseau de Tufin.

Création d'une règle pour autoriser les requêtes DNS

Une fois que la règle par défaut de tout refuser est en place, vous pouvez commencer à appliquer des règles supplémentaires, telles qu'une règle qui permet aux pods d'interroger CoreDNS pour la résolution de noms.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-dns-access
  namespace: default
spec:
  podSelector:
    matchLabels: {}
  policyTypes:
  - Egress
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: kube-system
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - protocol: UDP
      port: 53
```

allow-dns-access

		Target		Destination				
#	Policy name	Namespace	Pods	Namespace	Pods	Subnet	Ports	
1	allow-dns-access	default	Any	name: kube-system	Any		UDP: 53	

Ajoutez progressivement des règles pour autoriser de manière sélective le flux de trafic entre les espaces de noms/pods

Comprenez les exigences de l'application et créez des règles d'entrée et de sortie précises selon les besoins. L'exemple ci-dessous montre comment restreindre le trafic entrant sur le port 80 à `app-one` partir de`client-one`. Cela permet de minimiser la surface d'attaque et de réduire le risque d'accès non autorisé.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-ingress-app-one
  namespace: default
spec:
  podSelector:
    matchLabels:
      k8s-app: app-one
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          k8s-app: client-one
    ports:
    - protocol: TCP
      port: 80
```

allow-ingress-app-one

		Target		Source				
#	Policy name	Namespace	Pods	Namespace	Pods	Subnet	Ports	
1	allow-ingress-app-one	default	k8s-app: app-one	default	k8s-app: client-one		TCP: 80	

Surveillance de l'application des politiques du réseau

- Utiliser l'éditeur de stratégie réseau
  - L'éditeur de politique réseau facilite les visualisations, le score de sécurité et la génération automatique à partir des journaux de flux réseau
  - Élaborez des politiques réseau de manière interactive
- Journaux d'audit
  - Consultez régulièrement les journaux d'audit de votre cluster EKS
  - Les journaux d'audit fournissent de nombreuses informations sur les actions effectuées sur votre cluster, notamment les modifications apportées aux politiques réseau.
  - Utilisez ces informations pour suivre les modifications apportées à vos politiques réseau au fil du temps et détecter toute modification non autorisée ou inattendue
- Tests automatisés
  - Mettez en œuvre des tests automatisés en créant un environnement de test qui reflète votre environnement de production et déployez régulièrement des charges de travail qui tentent de violer les politiques de votre réseau.
- Surveillance des métriques
  - Configurez vos agents d'observabilité pour extraire les métriques Prometheus des agents du nœud VPC CNI, afin de surveiller l'état de santé des agents et les erreurs du SDK.
- Auditez régulièrement les politiques du réseau
  - Vérifiez régulièrement vos politiques réseau pour vous assurer qu'elles répondent aux exigences actuelles de votre application. Au fur et à mesure que votre application évolue, un audit vous permet de supprimer les règles d'entrée et de sortie redondantes et de vous assurer que vos applications ne disposent pas d'autorisations excessives.
- Assurez-vous que les politiques réseau existent à l'aide de l'Open Policy Agent (OPA)
  - Utilisez la politique OPA comme indiqué ci-dessous pour vous assurer que la politique réseau existe toujours avant d'intégrer les modules d'application. Cette politique interdit l'intégration de pods k8s portant une étiquette `k8s-app: sample-app` s'il n'existe pas de politique réseau correspondante.

```
package kubernetes.admission
import data.kubernetes.networkpolicies

deny[msg] {
```

```
    input.request.kind.kind == "Pod"
    pod_label_value := {v["k8s-app"] | v := input.request.object.metadata.labels}
    contains_label(pod_label_value, "sample-app")
    np_label_value := {v["k8s-app"] | v :=
 networkpolicies[_].spec.podSelector.matchLabels}
    not contains_label(np_label_value, "sample-app")
    msg:= sprintf("The Pod %v could not be created because it is missing an associated
 Network Policy.", [input.request.object.metadata.name])
}
contains_label(arr, val) {
    arr[_] == val
}
```

Résolution des problèmes

Surveillez les journaux vpc-network-policy-controller des agents des nœuds

Activez les journaux du gestionnaire du plan de contrôle EKS pour diagnostiquer la fonctionnalité de politique réseau. Vous pouvez diffuser les journaux du plan de contrôle vers un groupe de CloudWatch journaux et utiliser CloudWatchLog Insights pour effectuer des requêtes avancées. À partir des journaux, vous pouvez voir quels objets de point de terminaison du module sont résolus selon une politique réseau, l'état de réconciliation des politiques et vérifier si la politique fonctionne comme prévu.

En outre, Amazon VPC CNI vous permet d'activer la collecte et l'exportation des journaux d'application des politiques vers Amazon Cloudwatch à partir des nœuds de travail EKS. Une fois activé, vous pouvez tirer parti de CloudWatchContainer Insights pour obtenir des informations sur votre utilisation liée aux politiques du réseau.

Amazon VPC CNI fournit également un SDK qui fournit une interface permettant d'interagir avec les programmes eBPF sur le nœud. Le SDK est installé lorsqu'il `aws-node` est déployé sur les nœuds. Vous pouvez trouver le binaire du SDK installé dans le `/opt/cni/bin` répertoire du nœud. Au lancement, le SDK prend en charge les fonctionnalités fondamentales telles que l'inspection des programmes et des cartes eBPF.

```
sudo /opt/cni/bin/aws-eks-na-cli ebpf progs
```

Enregistrer les métadonnées du trafic réseau

AWS VPC Flow Logs capture les métadonnées relatives au trafic transitant par un VPC, telles que l'adresse IP et le port source et de destination, ainsi que les paquets acceptés/abandonnés. Ces

informations peuvent être analysées pour détecter toute activité suspecte ou inhabituelle entre les ressources du VPC, y compris les pods. Toutefois, étant donné que les adresses IP des pods changent fréquemment lorsqu'ils sont remplacés, les journaux de flux peuvent ne pas être suffisants à eux seuls. Calico Enterprise complète les journaux de flux avec des étiquettes de modules et d'autres métadonnées, ce qui facilite le déchiffrement des flux de trafic entre les modules.

Groupes de sécurité

EKS utilise les groupes de sécurité VPC AWS (SGs) pour contrôler le trafic entre le plan de contrôle Kubernetes et les nœuds de travail du cluster. Les groupes de sécurité sont également utilisés pour contrôler le trafic entre les nœuds de travail, les autres ressources VPC et les adresses IP externes. Lorsque vous provisionnez un cluster EKS (avec Kubernetes version 1.14-eks.3 ou supérieure), un groupe de sécurité de cluster est automatiquement créé pour vous. Ce groupe de sécurité permet une communication sans entrave entre le plan de contrôle EKS et les nœuds des groupes de nœuds gérés. Pour des raisons de simplicité, il est recommandé d'ajouter le cluster SG à tous les groupes de nœuds, y compris les groupes de nœuds non gérés.

Avant la version 1.14 de Kubernetes et la version eks.3 d'EKS, des groupes de sécurité distincts étaient configurés pour le plan de contrôle EKS et les groupes de nœuds. Les règles minimales et suggérées pour le plan de contrôle et les groupes de sécurité des groupes de nœuds se trouvent dans le https://docs.aws.amazon.com/eks/ latest/userguide/sec fichier -group-reqs.html. Les règles minimales pour le groupe de sécurité du plan de contrôle autorisent le port 443 en provenance du nœud de travail SG. Cette règle permet aux kubelets de communiquer avec le serveur d'API Kubernetes. Il inclut également le port 10250 pour le trafic sortant vers le nœud de travail SG ; 10250 est le port que les kubelets écoutent. De même, les règles de groupe de nœuds minimum autorisent le port 10250 entrant depuis le plan de contrôle SG et le port 443 sortant vers le plan de contrôle SG. Enfin, il existe une règle qui permet une communication sans entrave entre les nœuds d'un groupe de nœuds.

Si vous devez contrôler la communication entre les services exécutés au sein du cluster et les services exécutés en dehors du cluster, tels qu'une base de données RDS, envisagez de créer des groupes de sécurité pour les pods. Avec les groupes de sécurité pour les modules, vous pouvez attribuer un groupe de sécurité existant à un ensemble de modules.

> **Warning**
>
> Si vous faites référence à un groupe de sécurité qui n'existe pas avant la création des modules, ceux-ci ne seront pas planifiés.

Vous pouvez contrôler les modules affectés à un groupe de sécurité en créant un `SecurityGroupPolicy` objet et en spécifiant a `PodSelector` ou `ServiceAccountSelector` a. Si les sélecteurs sont définis sur, le SGs référencé `{}` sera attribué `SecurityGroupPolicy` à tous les pods d'un espace de noms ou à tous les comptes de service d'un espace de noms. Assurez-vous d'avoir pris connaissance de toutes les considérations avant d'implémenter des groupes de sécurité pour les pods.

> **Important**
>
> Si vous utilisez SGs des pods, vous devez créer un port 53 SGs qui autorise le port 53 à destination du groupe de sécurité du cluster. De même, vous devez mettre à jour le groupe de sécurité du cluster pour accepter le trafic entrant du port 53 en provenance du groupe de sécurité du pod.

> **Important**
>
> Les limites relatives aux groupes de sécurité s'appliquent toujours lors de l'utilisation de groupes de sécurité pour les pods. Utilisez-les donc judicieusement.

> **Important**
>
> Vous devez créer des règles pour le trafic entrant depuis le groupe de sécurité du cluster (kubelet) pour toutes les sondes configurées pour le pod.

> **Important**
>
> Les groupes de sécurité pour les pods reposent sur une fonctionnalité connue sous le nom de jonction ENI qui a été créée pour augmenter la densité ENI d'une instance EC2. Lorsqu'un pod est attribué à un SG, un contrôleur VPC associe une branche ENI du groupe de nœuds au pod. S'il n'y a pas suffisamment de branches ENIs disponibles dans un groupe de nœuds au moment de la planification du pod, le pod restera en état d'attente. Le nombre de branches ENIs qu'une instance peut prendre en charge varie d'une instance à l'autre type/family. See https://docs.aws.amazon.com/eks/latest/userguide/security : groups-for-pods .html# supported-instance-types pour plus de détails.

Bien que les groupes de sécurité pour les pods constituent un moyen natif d'AWS de contrôler le trafic réseau à l'intérieur et à l'extérieur de votre cluster sans la surcharge d'un démon de politique, d'autres options sont disponibles. Par exemple, le moteur de politique Cilium vous permet de référencer un nom DNS dans une politique réseau. Calico Enterprise inclut une option permettant de mapper les politiques réseau aux groupes de sécurité AWS. Si vous avez implémenté un maillage de services tel qu'Istio, vous pouvez utiliser une passerelle de sortie pour restreindre la sortie du réseau à des domaines ou adresses IP spécifiques et entièrement qualifiés. Pour plus d'informations sur cette option, consultez la série en trois parties sur le contrôle du trafic de sortie dans Istio.

Quand utiliser la politique réseau par rapport au groupe de sécurité pour les pods ?

Quand utiliser la politique réseau Kubernetes

- Contrôle pod-to-pod du trafic
  - Convient pour contrôler le trafic réseau entre les pods d'un cluster (trafic est-ouest)
- Contrôlez le trafic au niveau de l'adresse IP ou du port (couche OSI 3 ou 4)

Quand utiliser les groupes de sécurité AWS pour les pods (SGP)

- Tirez parti des configurations AWS existantes
  - Si vous disposez déjà d'un ensemble complexe de groupes de sécurité EC2 qui gèrent l'accès aux services AWS et que vous migrez des applications d'instances EC2 vers EKS, cela SGPs peut être un très bon choix vous permettant de réutiliser les ressources des groupes de sécurité et de les appliquer à vos pods.
- Contrôlez l'accès aux services AWS
  - Vos applications exécutées au sein d'un cluster EKS souhaitent communiquer avec d'autres services AWS (base de données RDS), ce qui constitue un mécanisme efficace pour contrôler le trafic entre les pods et les services AWS. SGPs
- Isolation du trafic des pods et des nœuds
  - Si vous souhaitez séparer complètement le trafic des pods du reste du trafic des nœuds, utilisez le mode SGP en `POD_SECURITY_GROUP_ENFORCING_MODE=strict` mode.

Meilleures pratiques en matière d'utilisation des groupes de sécurité pour les pods et de la politique réseau

- Sécurité multicouche
  - Utilisez une combinaison de politiques réseau SGP et Kubernetes pour une approche de sécurité à plusieurs niveaux
  - SGPs À utiliser pour limiter l'accès au niveau du réseau aux services AWS qui ne font pas partie d'un cluster, tandis que les politiques réseau de Kubernetes peuvent restreindre le trafic réseau entre les pods du cluster
- Principe du moindre privilège
  - Autoriser uniquement le trafic nécessaire entre les pods ou les espaces de noms
- Segmentez vos applications
  - Dans la mesure du possible, segmentez les applications en fonction de la politique du réseau afin de réduire le rayon de propagation en cas de compromission d'une application
- Maintenir des politiques simples et claires
  - Les politiques réseau de Kubernetes peuvent être très détaillées et complexes, il est préférable de les garder aussi simples que possible pour réduire le risque de mauvaise configuration et alléger les frais de gestion
- Réduire la surface d'attaque
  - Minimisez la surface d'attaque en limitant l'exposition de vos applications

> **Important**
>
> Les groupes de sécurité pour les pods proposent deux modes d'application : `strict` et`standard`. Vous devez utiliser `standard` le mode lorsque vous utilisez à la fois la stratégie réseau et les groupes de sécurité pour les fonctionnalités des pods dans un cluster EKS.

En matière de sécurité du réseau, une approche multicouche est souvent la solution la plus efficace. L'utilisation combinée de la politique réseau Kubernetes et du SGP peut fournir une defense-in-depth stratégie robuste pour vos applications exécutées dans EKS.

Application de la politique Service Mesh ou politique réseau Kubernetes

A `service mesh` est une couche d'infrastructure dédiée que vous pouvez ajouter à vos applications. Il vous permet d'ajouter de manière transparente des fonctionnalités telles que l'observabilité, la gestion du trafic et la sécurité, sans les ajouter à votre propre code.

Le maillage de service applique les politiques à la couche 7 (application) du modèle OSI, tandis que les politiques réseau Kubernetes fonctionnent à la couche 3 (réseau) et à la couche 4 (transport). Il existe de nombreuses offres dans cet espace, comme AWSAppMesh, Istio, Linkerd, etc.

Quand utiliser Service Mesh pour appliquer les politiques

- Vous avez déjà investi dans un maillage de services
- Vous avez besoin de fonctionnalités plus avancées telles que la gestion du trafic, l'observabilité et la sécurité
  - Contrôle du trafic, équilibrage de charge, coupure de circuit, limitation de débit, délais d'attente, etc.
  - Informations détaillées sur les performances de vos services (latence, taux d'erreur, demandes par seconde, volumes de demandes, etc.)
  - Vous souhaitez implémenter et exploiter le maillage de services pour les fonctionnalités de sécurité telles que les MTL

Choisissez la politique réseau Kubernetes pour des cas d'utilisation plus simples

- Limitez les pods qui peuvent communiquer entre eux
- Les politiques réseau nécessitent moins de ressources qu'un maillage de services, ce qui les rend idéales pour des cas d'utilisation plus simples ou pour des clusters plus petits où les frais liés à l'exécution et à la gestion d'un maillage de services peuvent ne pas être justifiés

> **Note**
>
> Les politiques réseau et le maillage de services peuvent également être utilisés conjointement. Utilisez des politiques réseau pour fournir un niveau de sécurité et d'isolation de base entre vos pods, puis utilisez un maillage de services pour ajouter des fonctionnalités supplémentaires telles que la gestion du trafic, l'observabilité et la sécurité.

ThirdParty Moteurs de politique réseau

Envisagez un moteur de politique réseau tiers lorsque vous avez des exigences de politique avancées, telles que des politiques de réseau globales, la prise en charge des règles basées sur les noms d'hôte DNS, des règles de couche 7, des règles ServiceAccount basées et des deny/log actions explicites, etc. Calico est un moteur de politique open source de Tigera qui fonctionne bien avec EKS. Outre la mise en œuvre de l'ensemble complet des fonctionnalités de politique réseau de Kubernetes, Calico prend en charge les politiques réseau étendues avec un ensemble de fonctionnalités plus riche, notamment la prise en charge des règles de couche 7, par exemple HTTP, lorsqu'il est intégré à Istio. Les politiques de Calico peuvent être étendues aux espaces de noms, aux pods, aux comptes de service ou à l'échelle mondiale. Lorsque les politiques sont étendues à un compte de service, celui-ci associe un ensemble de ingress/egress règles à ce compte de service. Lorsque les règles RBAC appropriées sont en place, vous pouvez empêcher les équipes de contourner ces règles, ce qui permet aux professionnels de la sécurité informatique de déléguer l'administration des espaces de noms en toute sécurité. Isovalent, les responsables de Cilium, ont également étendu les politiques du réseau pour inclure la prise en charge partielle des règles de la couche 7, par exemple HTTP. Cilium prend également en charge les noms d'hôte DNS, ce qui peut être utile pour restreindre le trafic entre Kubernetes Services/Pods et les ressources exécutées à l'intérieur ou à l'extérieur de votre VPC. En revanche, Calico Enterprise inclut une fonctionnalité qui vous permet de mapper une politique réseau Kubernetes à un groupe de sécurité AWS, ainsi qu'à des noms d'hôte DNS.

Vous trouverez une liste des politiques réseau Kubernetes courantes à l'adresse. https://github.com/ahmetb/ kubernetes-network-policy-recipes Un ensemble de règles similaires pour Calico est disponible sur https://docs.projectcalico. org/security/calico-politique-réseau.

Migration vers le moteur de politique réseau Amazon VPC CNI

Pour garantir la cohérence et éviter tout comportement de communication inattendu entre les pods, il est recommandé de ne déployer qu'un seul moteur de stratégie réseau dans votre cluster. Si vous souhaitez migrer du moteur de politique réseau 3P vers le moteur de politique réseau VPC CNI, nous vous recommandons de convertir vos ressources NetworkPolicy CRDs 3P existantes en ressources NetworkPolicy Kubernetes avant d'activer la prise en charge de la politique réseau VPC CNI. Testez également les politiques migrées dans un cluster de test distinct avant de les appliquer dans votre environnement de production. Cela vous permet d'identifier et de résoudre tout problème ou incohérence potentiel dans le comportement de communication du pod.

Outil de migration

Pour faciliter votre processus de migration, nous avons développé un outil appelé K8s Network Policy Migrator qui convertit votre politique réseau existante en politiques Calico/Cilium réseau natives de CRDs Kubernetes. Après la conversion, vous pouvez directement tester les politiques réseau converties sur vos nouveaux clusters exécutant le contrôleur de politique réseau VPC CNI. L'outil est conçu pour vous aider à rationaliser le processus de migration et à garantir une transition harmonieuse.

> **Important**
>
> L'outil de migration ne convertira que les politiques 3P compatibles avec l'API native de politique réseau Kubernetes. Si vous utilisez les fonctionnalités avancées de politique réseau proposées par les plugins 3P, l'outil de migration les ignorera et les signalera.

Veuillez noter que l'outil de migration n'est actuellement pas pris en charge par l'équipe d'ingénierie des politiques du réseau AWS VPC CNI. Il est mis à la disposition des clients dans la mesure du possible. Nous vous encourageons à utiliser cet outil pour faciliter votre processus de migration. Si vous rencontrez des problèmes ou des bogues avec l'outil, nous vous demandons de bien vouloir créer un GitHubproblème. Vos commentaires sont précieux pour nous et nous aideront à améliorer continuellement nos services.

Ressources supplémentaires

- Kubernetes et Tigera : politiques réseau, sécurité et audit
- Entreprise Calico
- Cilium
- NetworkPolicyEditeur : un éditeur de politiques interactif de Cilium
- Inspektor Gadget conseille un gadget de politique réseau Suggère des politiques réseau basées sur une analyse du trafic réseau

Chiffrement en transit

Les applications qui doivent être conformes aux réglementations PCI, HIPAA ou autres peuvent avoir besoin de chiffrer les données pendant leur transit. De nos jours, le TLS est le choix de facto pour chiffrer le trafic sur le fil. Le protocole TLS, comme son prédécesseur SSL, fournit des

communications sécurisées sur un réseau à l'aide de protocoles cryptographiques. Le protocole TLS utilise un chiffrement symétrique dans lequel les clés de chiffrement des données sont générées sur la base d'un secret partagé négocié au début de la session. Voici quelques méthodes permettant de chiffrer des données dans un environnement Kubernetes.

Instances Nitro

Le trafic échangé entre les types d'instances Nitro suivants, par exemple C5n, G4, i3en, M5dn, M5n, P3dn, R5dn et R5n, est automatiquement chiffré par défaut. Lorsqu'il y a un saut intermédiaire, comme une passerelle de transit ou un équilibreur de charge, le trafic n'est pas chiffré. Consultez la section Chiffrement en transit pour plus de détails sur le chiffrement en transit ainsi que la liste complète des types d'instances qui prennent en charge le chiffrement réseau par défaut.

Interfaces réseau de conteneurs (CNIs)

WeaveNetpeut être configuré pour chiffrer automatiquement l'ensemble du trafic à l'aide du NaCl chiffrement pour le trafic de couverture et de l' IPsec ESP pour le trafic rapide des chemins de données.

Maillage de service

Le chiffrement en transit peut également être mis en œuvre avec un maillage de services tel que App Mesh, Linkerd v2 et Istio. AppMesh prend en charge les MTL avec des certificats X.509 ou le Secret Discovery Service (SDS) d'Envoy. Linkerd et Istio prennent tous deux en charge les MTL.

Le aws-app-mesh-examplesGitHub référentiel fournit des instructions détaillées pour configurer les MTL à l'aide de certificats X.509 et de SPIRE en tant que fournisseur de SDS avec votre conteneur Envoy :

- Configuration de MTL à l'aide de certificats X.509
- Configuration de TLS à l'aide de SPIRE (SDS)

App Mesh prend également en charge le chiffrement TLS avec un certificat privé émis par AWS Certificate Manager (ACM) ou un certificat stocké sur le système de fichiers local du nœud virtuel.

Le aws-app-mesh-examplesGitHub référentiel fournit des instructions détaillées pour configurer le protocole TLS à l'aide de certificats émis par ACM et de certificats fournis avec votre conteneur Envoy :

- Configuration du protocole TLS avec des certificats TLS fournis par fichier

- Configuration du protocole TLS avec AWS Certificate Manager

Contrôleurs d'entrée et équilibreurs de charge

Les contrôleurs d'entrée sont un moyen pour vous d'acheminer intelligemment le HTTP/S traffic that emanates from outside the cluster to services running inside the cluster. Oftentimes, these Ingresses are fronted by a layer 4 load balancer, like the Classic Load Balancer or the Network Load Balancer (NLB). Encrypted traffic can be terminated at different places within the network, e.g. at the load balancer, at the ingress resource, or the Pod. How and where you terminate your SSL connection will ultimately be dictated by your organization's network security policy. For instance, if you have a policy that requires end-to-end encryption, you will have to decrypt the traffic at the Pod. This will place additional burden on your Pod as it will have to spend cycles establishing the initial handshake. Overall SSL/TLS traitement qui consomme beaucoup de CPU. Par conséquent, si vous en avez la flexibilité, essayez d'effectuer le déchargement SSL à l'entrée ou à l'équilibreur de charge.

Utiliser le chiffrement avec les équilibreurs de charge élastiques AWS

L'AWS Application Load Balancer (ALB) et le Network Load Balancer (NLB) prennent tous deux en charge le chiffrement des transports (SSL et TLS). L'`alb.ingress.kubernetes.io/certificate-arn`annotation pour l'ALB vous permet de spécifier les certificats à ajouter à l'ALB. Si vous omettez l'annotation, le contrôleur tentera d'ajouter des certificats aux écouteurs qui en ont besoin en faisant correspondre les certificats AWS Certificate Manager (ACM) disponibles à l'aide du champ host. À partir d'EKS v1.15, vous pouvez utiliser l'`service.beta.kubernetes.io/aws-load-balancer-ssl-cert`annotation avec le NLB comme indiqué dans l'exemple ci-dessous.

```
apiVersion: v1
kind: Service
metadata:
  name: demo-app
  namespace: default
  labels:
    app: demo-app
  annotations:
     service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
     service.beta.kubernetes.io/aws-load-balancer-ssl-cert: "<certificate ARN>"
     service.beta.kubernetes.io/aws-load-balancer-ssl-ports: "443"
     service.beta.kubernetes.io/aws-load-balancer-backend-protocol: "http"
spec:
  type: LoadBalancer
  ports:
```

```
  - port: 443
    targetPort: 80
    protocol: TCP
  selector:
    app: demo-app
//---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx
  namespace: default
  labels:
    app: demo-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demo-app
  template:
    metadata:
      labels:
        app: demo-app
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 443
              protocol: TCP
            - containerPort: 80
              protocol: TCP
```

Vous trouverez ci-dessous d'autres exemples de SSL/TLS résiliation.

- Sécuriser EKS Ingress avec Contour et Let's Encrypt The Way GitOps
- Comment arrêter le trafic HTTPS sur les charges de travail Amazon EKS avec ACM ?

> **Important**
>
> Certaines entrées, comme le contrôleur AWS LB, implémentent les annotations SSL/TLS d'utilisation plutôt que dans le cadre de la spécification d'entrée.

ACM Private CA avec gestionnaire de certificats

Vous pouvez activer les protocoles TLS et MTL pour sécuriser les charges de travail de vos applications EKS à l'entrée, sur le pod et entre les pods à l'aide d'ACM Private Certificate Authority (CA) et de cert-manager, un module complémentaire populaire de Kubernetes permettant de distribuer, de renouveler et de révoquer des certificats. ACM Private CA est une autorité de certification hautement disponible, sécurisée et gérée sans les coûts initiaux et de maintenance liés à la gestion de votre propre autorité de certification. Si vous utilisez l'autorité de certification Kubernetes par défaut, il est possible d'améliorer votre sécurité et de répondre aux exigences de conformité avec ACM Private CA. L'autorité de certification privée ACM sécurise les clés privées dans les modules de sécurité matériels FIPS 140-2 de niveau 3 (très sécurisés), alors que l'autorité de certification par défaut stocke les clés codées en mémoire (moins sécurisée). Une autorité de certification centralisée vous permet également de mieux contrôler et d'améliorer l'auditabilité des certificats privés à la fois à l'intérieur et à l'extérieur d'un environnement Kubernetes.

Mode CA de courte durée pour le protocole TLS mutuel entre les charges de travail

Lorsque vous utilisez ACM Private CA pour mTLS dans EKS, il est recommandé d'utiliser des certificats de courte durée avec un mode CA de courte durée. Bien qu'il soit possible de délivrer des certificats de courte durée en mode CA à usage général, l'utilisation du mode CA de courte durée s'avère plus rentable (environ 75 % moins cher que le mode général) dans les cas d'utilisation où de nouveaux certificats doivent être émis fréquemment. En outre, vous devez essayer d'aligner la période de validité des certificats privés sur la durée de vie des pods de votre cluster EKS. Pour en savoir plus sur ACM Private CA et ses avantages, cliquez ici.

Instructions de configuration d'ACM

Commencez par créer une autorité de certification privée en suivant les procédures décrites dans les documents techniques d'ACM Private CA. Une fois que vous avez une autorité de certification privée, installez cert-manager en suivant les instructions d'installation habituelles. Après avoir installé cert-manager, installez le plugin Private CA Kubernetes cert-manager en suivant les instructions de configuration indiquées dans. GitHub Le plugin permet à cert-manager de demander des certificats privés à ACM Private CA.

Maintenant que vous disposez d'une autorité de certification privée et d'un cluster EKS sur lesquels le gestionnaire de certificats et le plugin sont installés, il est temps de définir les autorisations et de créer l'émetteur. Mettez à jour les autorisations IAM du rôle de nœud EKS pour autoriser l'accès à ACM Private CA. Remplacez le `<CA_ARN>` par la valeur de votre autorité de certification privée :

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "awspcaissuer",
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:IssueCertificate"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:acm-pca:us-west-2:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012"
        }
    ]
}
```

Les rôles de service pour les comptes IAM ou IRSA peuvent également être utilisés. Consultez la section Ressources supplémentaires ci-dessous pour obtenir des exemples complets.

Créez un émetteur dans Amazon EKS en créant un fichier de définition de ressource personnalisée nommé cluster-issuer.yaml contenant le texte suivant, en remplaçant `<CA_ARN>` les informations par votre autorité de certification privée. `<Region>`

```
apiVersion: awspca.cert-manager.io/v1beta1
kind: AWSPCAClusterIssuer
metadata:
          name: demo-test-root-ca
spec:
          arn: <CA_ARN>
          region: <Region>
```

Déployez l'émetteur que vous avez créé.

```
kubectl apply -f cluster-issuer.yaml
```

Votre cluster EKS est configuré pour demander des certificats à Private CA. Vous pouvez désormais utiliser la `Certificate` ressource de cert-manager pour émettre des certificats en remplaçant les valeurs du `issuerRef` champ par l'émetteur privé CA que vous avez créé ci-dessus. Pour plus de détails sur la façon de spécifier et de demander des ressources de certificat, veuillez consulter le guide des ressources de certificats de cert-manager. Voir des exemples ici.

ACM Private CA avec Istio et cert-manager

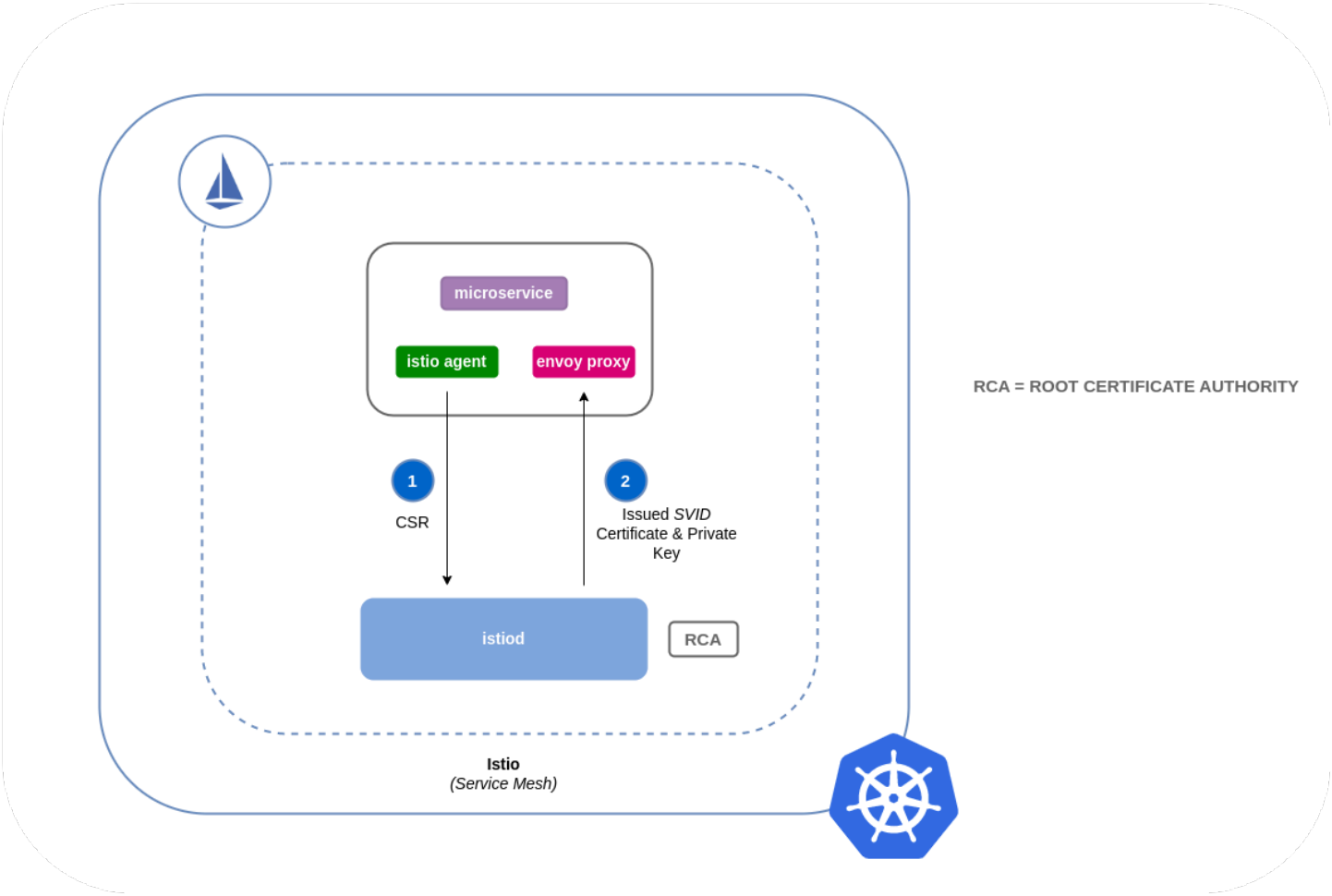
Si vous exécutez Istio dans votre cluster EKS, vous pouvez empêcher le plan de contrôle Istio (en particulier`istiod`) de fonctionner en tant qu'autorité de certification (CA) racine et configurer l'autorité de certification privée ACM en tant qu'autorité de certification racine pour les MTL entre les charges de travail. Si vous optez pour cette approche, pensez à utiliser le mode CA éphémère dans ACM Private CA. Reportez-vous à la section précédente et à ce billet de blog pour plus de détails.

Comment fonctionne la signature des certificats dans Istio (par défaut)

Les charges de travail dans Kubernetes sont identifiées à l'aide de comptes de service. Si vous ne spécifiez aucun compte de service, Kubernetes en attribuera automatiquement un à votre charge de travail. De plus, les comptes de service installent automatiquement un jeton associé. Ce jeton est utilisé par le compte de service pour les charges de travail afin de s'authentifier auprès de l'API Kubernetes. Le compte de service peut être suffisant comme identité pour Kubernetes, mais Istio possède son propre système de gestion des identités et une autorité de certification. Lorsqu'une charge de travail démarre avec son proxy annexe envoyé, elle a besoin d'une identité attribuée par Istio pour être considérée comme fiable et autorisée à communiquer avec les autres services du maillage.

Pour obtenir cette identité auprès d'Istio, `istio-agent` envoie une demande connue sous le nom de demande de signature de certificat (ou CSR) au plan de contrôle Istio. Ce CSR contient le jeton du compte de service afin que l'identité de la charge de travail puisse être vérifiée avant d'être traitée. Ce processus de vérification est géré par`istiod`, qui agit à la fois en tant qu'autorité d'enregistrement (ou RA) et en tant que CA. Le RA fait office de gardien qui s'assure que seule la CSR vérifiée parvient à l'AC. Une fois le CSR vérifié, il sera transmis au CA qui délivrera alors un certificat contenant une identité SPIFFE avec le compte de service. Ce certificat est appelé document d'identité vérifiable SPIFFE (ou SVID). Le SVID est attribué au service demandeur à des fins d'identification et pour chiffrer le trafic en transit entre les services communicants.

Flux par défaut pour les demandes de signature de certificats Istio :



Comment fonctionne la signature de certificats dans Istio avec ACM Private CA

Vous pouvez utiliser un module complémentaire de gestionnaire de certificats appelé agent Istio Certificate Signing Request (istio-csr) pour intégrer Istio à ACM Private CA. Cet agent permet de sécuriser les charges de travail Istio et les composants du plan de contrôle auprès des émetteurs de gestionnaires de certificats, en l'occurrence ACM Private CA. L'agent istio-csr expose le même service que celui fourni par istiod dans la configuration par défaut de validation des données entrantes. CSRs Sauf qu'après vérification, il convertira les demandes en ressources prises en charge par le gestionnaire de certificats (c'est-à-dire des intégrations avec des émetteurs de CA externes).

Chaque fois qu'un CSR provient d'une charge de travail, il est transmis à istio-csr, qui demande des certificats à ACM Private CA. Cette communication entre istio-csr et ACM Private CA est activée par le plugin AWS Private CA issuer. Le gestionnaire de certificats utilise ce plugin pour demander des certificats TLS à ACM Private CA. Le plugin émetteur communiquera avec le service ACM Private CA

pour demander un certificat signé pour la charge de travail. Une fois le certificat signé, il sera renvoyé à istio-csr, qui lira la demande signée et la renverra à la charge de travail à l'origine du CSR.

Flux pour les demandes de signature de certificats Istio avec istio-csr

image : : istio-csr-with-acm -private-ca.png [Flux pour les demandes de signature de certificats Istio avec istio-csr]

Instructions de configuration d'Istio avec CA privée

1. Commencez par suivre les mêmes instructions de configuration que dans cette section pour effectuer les opérations suivantes :
2. Créer une autorité de certification privée
3. Installer cert-manager
4. Installez le plugin de l'émetteur
5. Définissez les autorisations et créez un émetteur. L'émetteur représente l'autorité de certification et est utilisé pour signer `istiod` et mailler les certificats de charge de travail. Il communiquera avec ACM Private CA.
6. Créez un `istio-system` espace de noms. C'est là que les ressources `istiod certificate` et les autres ressources d'Istio seront déployées.
7. Installez Istio CSR configuré avec le plug-in AWS Private CA Issuer. Vous pouvez conserver les demandes de signature de certificat pour les charges de travail afin de vérifier qu'elles sont approuvées et signées (`preserveCertificateRequests=true`).

```
helm install -n cert-manager cert-manager-istio-csr jetstack/cert-manager-istio-csr \
--set "app.certmanager.issuer.group=awspca.cert-manager.io" \
--set "app.certmanager.issuer.kind=AWSPCAClusterIssuer" \
--set "app.certmanager.issuer.name=<the-name-of-the-issuer-you-created>" \
--set "app.certmanager.preserveCertificateRequests=true" \
--set "app.server.maxCertificateDuration=48h" \
--set "app.tls.certificateDuration=24h" \
--set "app.tls.istiodCertificateDuration=24h" \
--set "app.tls.rootCAFile=/var/run/secrets/istio-csr/ca.pem" \
--set "volumeMounts[0].name=root-ca" \
--set "volumeMounts[0].mountPath=/var/run/secrets/istio-csr" \
--set "volumes[0].name=root-ca" \
--set "volumes[0].secret.secretName=istio-root-ca"
```

8. Installez Istio avec des configurations personnalisées à remplacer par `istiod` `cert-manager` `istio-csr` en tant que fournisseur de certificats pour le maillage. Ce processus peut être effectué à l'aide de l'opérateur Istio.

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
metadata:
  name: istio
  namespace: istio-system
spec:
  profile: "demo"
  hub: gcr.io/istio-release
  values:
  global:
    # Change certificate provider to cert-manager istio agent for istio agent
    caAddress: cert-manager-istio-csr.cert-manager.svc:443
  components:
    pilot:
      k8s:
        env:
          # Disable istiod CA Sever functionality
        - name: ENABLE_CA_SERVER
          value: "false"
        overlays:
        - apiVersion: apps/v1
          kind: Deployment
          name: istiod
          patches:

            # Mount istiod serving and webhook certificate from Secret mount
          - path: spec.template.spec.containers.[name:discovery].args[7]
            value: "--tlsCertFile=/etc/cert-manager/tls/tls.crt"
          - path: spec.template.spec.containers.[name:discovery].args[8]
            value: "--tlsKeyFile=/etc/cert-manager/tls/tls.key"
          - path: spec.template.spec.containers.[name:discovery].args[9]
            value: "--caCertFile=/etc/cert-manager/ca/root-cert.pem"

          - path: spec.template.spec.containers.[name:discovery].volumeMounts[6]
            value:
              name: cert-manager
              mountPath: "/etc/cert-manager/tls"
              readOnly: true
          - path: spec.template.spec.containers.[name:discovery].volumeMounts[7]
```

```
            value:
              name: ca-root-cert
              mountPath: "/etc/cert-manager/ca"
              readOnly: true

          - path: spec.template.spec.volumes[6]
            value:
              name: cert-manager
              secret:
                secretName: istiod-tls
          - path: spec.template.spec.volumes[7]
            value:
              name: ca-root-cert
              configMap:
                defaultMode: 420
                name: istio-ca-root-cert
```

9. Déployez la ressource personnalisée que vous avez créée ci-dessus.

```
istioctl operator init
kubectl apply -f istio-custom-config.yaml
```

10. Vous pouvez désormais déployer une charge de travail sur le maillage de votre cluster EKS et appliquer les MTL.

Demandes de signature de certificats Istio

image : istio-csr-requests .png [Demandes de signature de certificats Istio]

Outils et ressources

- Atelier d'immersion sur la sécurité Amazon EKS - Sécurité du réseau
- Comment implémenter le gestionnaire de certificats et le plugin ACM Private CA pour activer le protocole TLS dans EKS.
- Configuration du chiffrement end-to-end TLS sur Amazon EKS avec le nouveau AWS Load Balancer Controller et ACM Private CA.
- Le plugin privé de gestion de certificats CA Kubernetes est activé. GitHub
- Guide de l'utilisateur du plugin privé CA Kubernetes cert-manager.
- Comment utiliser le mode de certificat éphémère de l'autorité de certification privée AWS

- egress-operator Un opérateur et un plugin DNS pour contrôler le trafic sortant de votre cluster sans inspection du protocole
- NeuVector de SUSE, plate-forme open source de sécurité des conteneurs Zero-Trust, fournit des règles réseau politiques, la prévention des pertes de données (DLP), un pare-feu pour applications Web (WAF) et des signatures de menaces réseau.

Chiffrement des données et gestion des secrets

Chiffrement au repos

Il existe trois options de stockage natives AWS différentes que vous pouvez utiliser avec Kubernetes : EBS, EFS et pour Lustre. FSx Tous les trois proposent un chiffrement au repos à l'aide d'une clé gérée par le service ou d'une clé principale du client (CMK). Pour EBS, vous pouvez utiliser le pilote de stockage intégré ou le pilote EBS CSI. Les deux incluent des paramètres permettant de chiffrer les volumes et de fournir une clé CMK. Pour EFS, vous pouvez utiliser le pilote EFS CSI, mais contrairement à EBS, le pilote EFS CSI ne prend pas en charge le provisionnement dynamique. Si vous souhaitez utiliser EFS avec EKS, vous devez fournir et configurer le chiffrement au repos pour le système de fichiers avant de créer un PV. Pour plus d'informations sur le chiffrement des fichiers EFS, reportez-vous à la section Chiffrement des données au repos. Outre le chiffrement au repos, EFS et FSx Lustre incluent une option de chiffrement des données en transit. FSx for Lustre le fait par défaut. Pour EFS, vous pouvez ajouter le chiffrement du transport en ajoutant le `tls` paramètre à `mountOptions` dans votre PV, comme dans cet exemple :

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: efs-pv
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: efs-sc
  mountOptions:
    - tls
  csi:
    driver: efs.csi.aws.com
```

```
    volumeHandle: <file_system_id>
```

Le pilote FSx CSI prend en charge le provisionnement dynamique des systèmes de fichiers Lustre. Il chiffre les données à l'aide d'une clé gérée par le service par défaut, bien qu'il soit possible de fournir votre propre clé CMK, comme dans cet exemple :

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: fsx-sc
provisioner: fsx.csi.aws.com
parameters:
  subnetId: subnet-056da83524edbe641
  securityGroupIds: sg-086f61ea73388fb6b
  deploymentType: PERSISTENT_1
  kmsKeyId: <kms_arn>
```

> **Important**
>
> Depuis le 28 mai 2020, toutes les données écrites sur le volume éphémère des pods EKS Fargate sont cryptées par défaut à l'aide d'un algorithme cryptographique AES-256 standard. Aucune modification de votre application n'est nécessaire car le chiffrement et le déchiffrement sont gérés sans problème par le service.

Chiffrer les données au repos

Le chiffrement des données au repos est considéré comme une bonne pratique. Si vous ne savez pas si le chiffrement est nécessaire, chiffrez vos données.

Faites pivoter CMKs votre

Configurez KMS pour qu'il fasse automatiquement pivoter votre CMKs. Cela fera pivoter vos clés une fois par an tout en sauvegardant les anciennes clés indéfiniment afin que vos données puissent toujours être déchiffrées. Pour plus d'informations, voir Rotation des clés principales du client

Utilisez les points d'accès EFS pour simplifier l'accès aux ensembles de données partagés

Si vous avez partagé des ensembles de données avec différentes autorisations de fichiers POSIX ou si vous souhaitez restreindre l'accès à une partie du système de fichiers partagé en créant différents points de montage, pensez à utiliser des points d'accès EFS. Pour en savoir plus sur l'utilisation des points d'accès, consultez le https://docs.aws.amazon.com/efs/ latest/ug/efs fichier -access-points.html. Aujourd'hui, si vous souhaitez utiliser un point d'accès (AP), vous devez référencer le point d'accès dans les `volumeHandle` paramètres du PV.

> **Important**
>
> Depuis le 23 mars 2021, le pilote EFS CSI prend en charge le provisionnement dynamique des points d'accès EFS. Les points d'accès sont des points d'entrée spécifiques à une application dans un système de fichiers EFS qui facilitent le partage d'un système de fichiers entre plusieurs pods. Chaque système de fichiers EFS peut en contenir jusqu'à 120 PVs. Consultez Présentation du provisionnement dynamique Amazon EFS CSI pour plus d'informations.

Gestion des secrets

Les secrets Kubernetes sont utilisés pour stocker des informations sensibles, telles que des certificats utilisateur, des mots de passe ou des clés d'API. Ils sont conservés dans etcd sous forme de chaînes codées en base64. Sur EKS, les volumes EBS pour les nœuds etcd sont chiffrés avec le chiffrement EBS. Un pod peut récupérer un objet secret Kubernetes en faisant référence au secret dans le. `podSpec` Ces secrets peuvent être mappés à une variable d'environnement ou montés en tant que volume. Pour plus d'informations sur la création de secrets, consultez https://kubernetes. io/docs/concepts/configuration/secret/.

> **Warning**
>
> Les secrets d'un espace de noms particulier peuvent être référencés par tous les modules de l'espace de noms du secret.

> **Warning**
>
> L'autorisateur de nœud permet au Kubelet de lire tous les secrets montés sur le nœud.

Utiliser AWS KMS pour le chiffrement de l'enveloppe des secrets Kubernetes

Cela vous permet de chiffrer vos secrets à l'aide d'une clé de chiffrement des données (DEK) unique. Le DEK est ensuite chiffré à l'aide d'une clé de chiffrement (KEK) d'AWS KMS, qui peut être automatiquement modifiée selon un calendrier récurrent. Avec le plugin KMS pour Kubernetes, tous les secrets Kubernetes sont stockés dans etcd sous forme de texte chiffré au lieu de texte brut et ne peuvent être déchiffrés que par le serveur d'API Kubernetes. Pour plus de détails, consultez la section Utilisation du support du fournisseur de chiffrement EKS pour une défense approfondie

Auditez l'utilisation de Kubernetes Secrets

Sur EKS, activez la journalisation des audits et créez un filtre de CloudWatch métriques et une alarme pour vous avertir lorsqu'un secret est utilisé (facultatif). Voici un exemple de filtre de métriques pour le journal d'audit de Kubernetes. `{($.verb="get") && ($.objectRef.resource="secret")}` Vous pouvez également utiliser les requêtes suivantes avec CloudWatch Log Insights :

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| stats count(*) by objectRef.name as secret
| filter verb="get" and objectRef.resource="secrets"
```

La requête ci-dessus affichera le nombre de fois qu'un secret a été consulté au cours d'une période donnée.

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| filter verb="get" and objectRef.resource="secrets"
| display objectRef.namespace, objectRef.name, user.username, responseStatus.code
```

Cette requête affichera le secret, ainsi que l'espace de noms et le nom d'utilisateur de l'utilisateur qui a tenté d'accéder au secret et le code de réponse.

Faites régulièrement alterner vos secrets

Kubernetes ne fait pas automatiquement pivoter les secrets. Si vous devez alterner les secrets, pensez à utiliser un magasin de secrets externe, par exemple Vault ou AWS Secrets Manager.

Utilisez des espaces de noms distincts pour isoler les secrets des différentes applications

Si vous avez des secrets qui ne peuvent pas être partagés entre les applications d'un espace de noms, créez un espace de noms distinct pour ces applications.

Utiliser des montages de volume plutôt que des variables d'environnement

Les valeurs des variables d'environnement peuvent apparaître involontairement dans les journaux. Les secrets montés sous forme de volumes sont instanciés sous forme de volumes tmpfs (un système de fichiers sauvegardé par de la RAM) qui sont automatiquement supprimés du nœud lorsque le pod est supprimé.

Utiliser un fournisseur de secrets externe

Il existe plusieurs alternatives viables à l'utilisation des secrets Kubernetes, notamment AWS Secrets Manager et Hashicorp's Vault. Ces services offrent des fonctionnalités telles que des contrôles d'accès précis, un cryptage renforcé et une rotation automatique des secrets qui ne sont pas disponibles avec Kubernetes Secrets. Les secrets scellés de Bitnami sont une autre approche qui utilise le cryptage asymétrique pour créer des « secrets scellés ». Une clé publique est utilisée pour chiffrer le secret tandis que la clé privée utilisée pour le déchiffrer est conservée dans le cluster, ce qui vous permet de stocker des secrets scellés en toute sécurité dans des systèmes de contrôle de source tels que Git. Consultez Gérer le déploiement des secrets dans Kubernetes à l'aide de Sealed Secrets pour plus d'informations.

À mesure que l'utilisation de magasins de secrets externes s'est accrue, le besoin de les intégrer à Kubernetes s'est accru. Le pilote CSI Secret Store est un projet communautaire qui utilise le modèle du pilote CSI pour récupérer des secrets dans des magasins de secrets externes. Actuellement, le pilote prend en charge AWS Secrets Manager, Azure, Vault et GCP. Le fournisseur AWS prend en charge à la fois AWS Secrets Manager et AWS Parameter Store. Il peut également être configuré pour alterner les secrets lorsqu'ils expirent et pour synchroniser les secrets d'AWS Secrets Manager avec ceux de Kubernetes. La synchronisation des secrets peut être utile lorsque vous devez référencer un secret en tant que variable d'environnement au lieu de le lire à partir d'un volume.

> **Note**
>
> Lorsque le pilote CSI de la banque secrète doit récupérer un secret, il assume le rôle IRSA attribué au module qui fait référence à un secret. Le code de cette opération se trouve ici.

Pour plus d'informations sur le fournisseur de secrets et de configuration AWS (ASCP), consultez les ressources suivantes :

- Comment utiliser le fournisseur de configuration AWS Secrets avec le pilote CSI Kubernetes Secret Store
- Intégration des secrets de Secrets Manager au pilote CSI Kubernetes Secrets Store

external-secrets est une autre façon d'utiliser un magasin de secrets externe avec Kubernetes. À l'instar du pilote CSI, external-secrets fonctionne avec différents backends, notamment AWS Secrets Manager. La différence est qu'au lieu de récupérer des secrets dans le magasin de secrets externe, external-secrets copie les secrets de ces backends vers Kubernetes en tant que secrets. Cela vous permet de gérer les secrets à l'aide de votre magasin de secrets préféré et d'interagir avec les secrets de manière native à Kubernetes.

Outils et ressources

- Atelier d'immersion sur la sécurité Amazon EKS - Chiffrement des données et gestion des secrets

Sécurité de l'exécution

La sécurité d'exécution fournit une protection active à vos conteneurs lorsqu'ils sont en cours d'exécution. L'idée est de détecter et d' and/or empêcher toute activité malveillante de se produire à l'intérieur du conteneur. Cela peut être réalisé à l'aide d'un certain nombre de mécanismes du noyau Linux ou d'extensions de noyau intégrées à Kubernetes, tels que les fonctionnalités Linux, l'informatique sécurisée (seccomp) ou. AppArmor SELinux Il existe également des options telles qu'Amazon GuardDuty et des outils tiers qui peuvent aider à établir des bases de référence et à détecter les activités anormales en réduisant la configuration manuelle des mécanismes du noyau Linux.

> **Important**
>
> Kubernetes ne fournit actuellement aucun mécanisme natif pour le chargement de seccomp ou de profils sur AppArmor les nœuds. SELinux Ils doivent être chargés manuellement ou installés sur les nœuds lors de leur démarrage. Cela doit être fait avant de les référencer dans vos pods, car le planificateur ne sait pas quels nœuds ont des profils. Découvrez ci-dessous comment des outils tels que Security Profiles Operator peuvent aider à automatiser le provisionnement des profils sur les nœuds.

Contextes de sécurité et contrôles Kubernetes intégrés

De nombreux mécanismes de sécurité d'exécution Linux sont étroitement intégrés à Kubernetes et peuvent être configurés via des contextes de sécurité Kubernetes. L'une de ces options est le `privileged` drapeau, qui est `false` par défaut et, s'il est activé, équivaut essentiellement à root sur l'hôte. Il est presque toujours inapproprié d'activer le mode privilégié dans les charges de travail de production, mais il existe de nombreux autres contrôles qui peuvent fournir des privilèges plus précis aux conteneurs, le cas échéant.

Fonctionnalités de Linux

Les fonctionnalités de Linux vous permettent d'octroyer certaines fonctionnalités à un pod ou à un conteneur sans fournir toutes les capacités de l'utilisateur root. Les exemples incluent `CAP_NET_ADMIN` ce qui permet de configurer des interfaces réseau ou des pare-feux`CAP_SYS_TIME`, ou qui permet de manipuler l'horloge du système.

Seccomp

Grâce à l'informatique sécurisée (seccomp), vous pouvez empêcher une application conteneurisée d'effectuer certains appels système vers le noyau du système d'exploitation hôte sous-jacent. Bien que le système d'exploitation Linux compte quelques centaines d'appels système, la plupart d'entre eux ne sont pas nécessaires pour exécuter des conteneurs. En limitant les appels système pouvant être effectués par un conteneur, vous pouvez réduire efficacement la surface d'attaque de votre application.

Seccomp fonctionne en interceptant les appels système et en n'autorisant que ceux qui ont été autorisés à passer. Docker possède un profil seccomp par défaut qui convient à la majorité des charges de travail générales, et d'autres environnements d'exécution de conteneurs tels que

containerd fournissent des valeurs par défaut comparables. Vous pouvez configurer votre conteneur ou votre Pod pour utiliser le profil seccomp par défaut de l'environnement d'exécution du conteneur en ajoutant ce qui suit à la `securityContext` section de la spécification du Pod :

```
securityContext:
  seccompProfile:
    type: RuntimeDefault
```

À partir de la version 1.22 (en alpha, stable à partir de la version 1.27), ce qui précède `RuntimeDefault` peut être utilisé pour tous les pods d'un nœud en utilisant un seul drapeau kubelet,. `--seccomp-default` Ensuite, le profil spécifié dans n'`securityContext`est nécessaire que pour les autres profils.

Il est également possible de créer vos propres profils pour les choses qui nécessitent des privilèges supplémentaires. Cela peut être très fastidieux à effectuer manuellement, mais il existe des outils tels qu'Inspektor Gadget (également recommandé dans la section sur la sécurité du réseau pour générer des politiques réseau) et Security Profiles Operator qui permettent d'utiliser des outils tels que eBPF ou des journaux pour enregistrer les exigences de base en matière de privilèges sous forme de profils seccomp. L'opérateur de profils de sécurité permet en outre d'automatiser le déploiement des profils enregistrés sur les nœuds à l'usage des pods et des conteneurs.

AppArmor et SELinux

AppArmor et SELinux sont connus sous le nom de contrôle d'accès obligatoire ou systèmes MAC. Leur concept est similaire à celui de seccomp, mais avec des fonctionnalités différentes, permettant le contrôle APIs d'accès, par exemple, à des chemins de système de fichiers ou à des ports réseau spécifiques. Le support de ces outils dépend de la distribution Linux, avec Debian/Ubuntu prise en charge AppArmor et prise en charge de RHEL/CentOS/Bottlerocket/Amazon Linux 2023 SELinux. Consultez également la section sur la sécurité de l'infrastructure pour une discussion plus approfondie à ce sujet SELinux.

Les deux AppArmor SELinux sont intégrés à Kubernetes, mais à partir de Kubernetes 1.28, les AppArmor profils doivent être spécifiés via des annotations, tandis que les SELinux étiquettes peuvent être définies directement via le champ Options du SELinux contexte de sécurité.

Comme pour les profils seccomp, l'opérateur de profils de sécurité mentionné ci-dessus peut vous aider à déployer des profils sur les nœuds du cluster. (À l'avenir, le projet vise également à générer des profils pour AppArmor et SELinux comme il le fait pour seccomp.)

Recommandations

Utilisez Amazon GuardDuty pour surveiller le temps d'exécution et détecter les menaces qui pèsent sur vos environnements EKS

Si vous ne disposez pas actuellement d'une solution permettant de surveiller en permanence les temps d'exécution d'EKS, d'analyser les journaux d'audit d'EKS et de détecter les logiciels malveillants et autres activités suspectes, Amazon GuardDuty recommande vivement d'utiliser Amazon aux clients qui recherchent un moyen simple, rapide, sécurisé, évolutif et rentable en un clic de protéger leurs environnements AWS. Amazon GuardDuty est un service de surveillance de la sécurité qui analyse et traite les sources de données fondamentales, telles que les événements de CloudTrail gestion AWS, les journaux d' CloudTrail événements AWS, les journaux de flux VPC (provenant d'instances Amazon EC2), les journaux d'audit Kubernetes et les journaux DNS. Il inclut également la surveillance du temps d'exécution d'EKS. Il utilise des flux de renseignements sur les menaces constamment mis à jour, tels que des listes d'adresses IP et de domaines malveillants, et l'apprentissage automatique pour identifier les activités inattendues, potentiellement non autorisées et malveillantes au sein de votre environnement AWS. Cela peut inclure des problèmes tels que l'augmentation des privilèges, l'utilisation d'informations d'identification divulguées ou la communication avec des adresses IP ou des domaines malveillants, la présence de logiciels malveillants sur vos instances Amazon EC2 et les charges de travail des conteneurs EKS, ou la découverte d'une activité d'API suspecte. GuardDuty vous informe de l'état de votre environnement AWS en produisant des résultats de sécurité que vous pouvez consulter dans la GuardDuty console ou via Amazon EventBridge. GuardDuty vous permet également d'exporter vos résultats vers un compartiment Amazon Simple Storage Service (S3) et de les intégrer à d'autres services tels que AWS Security Hub et Detective.

Regardez cette conférence technique en ligne AWS intitulée « Détection améliorée des menaces pour Amazon EKS avec Amazon GuardDuty - AWS Online Tech Talks » pour découvrir comment activer ces fonctionnalités de sécurité EKS supplémentaires step-by-step en quelques minutes.

Facultatif : utilisez une solution tierce pour la surveillance du temps d'exécution

La création et la gestion des profils seccomp et Apparmor peuvent s'avérer difficiles si vous n'êtes pas familiarisé avec la sécurité Linux. Si vous n'avez pas le temps de devenir compétent, pensez à utiliser une solution commerciale tierce. Beaucoup d'entre eux ont dépassé les profils statiques tels qu'Apparmor et seccomp et ont commencé à utiliser l'apprentissage automatique pour bloquer ou alerter en cas d'activité suspecte. Quelques-unes de ces solutions se trouvent ci-dessous dans

la section des outils. Des options supplémentaires sont disponibles sur AWS Marketplace for Containers.

Tenez compte des capacités de add/dropping Linux avant d'écrire des politiques seccomp

Les fonctionnalités impliquent diverses vérifications dans les fonctions du noyau accessibles par des appels système. Si la vérification échoue, l'appel système renvoie généralement une erreur. La vérification peut être effectuée soit juste au début d'un appel système spécifique, soit plus profondément dans le noyau, dans des zones pouvant être accessibles via plusieurs appels système différents (comme l'écriture dans un fichier privilégié spécifique). Seccomp, quant à lui, est un filtre syscall qui est appliqué à tous les appels système avant leur exécution. Un processus peut configurer un filtre qui lui permet de révoquer son droit d'exécuter certains appels système ou des arguments spécifiques pour certains appels système.

Avant d'utiliser seccomp, déterminez si les fonctionnalités de adding/removing Linux vous offrent le contrôle dont vous avez besoin. Voir Configuration des capacités pour les conteneurs pour plus d'informations.

Découvrez si vous pouvez atteindre vos objectifs en utilisant les politiques de sécurité du Pod (PSPs)

Les politiques de sécurité des modules offrent de nombreuses manières d'améliorer votre posture de sécurité sans introduire de complexité excessive. Explorez les options disponibles PSPs avant de vous lancer dans la création de profils Seccomp et Apparmor.

> **Warning**
>
> À partir de Kubernetes 1.25, PSPs ils ont été supprimés et remplacés par le contrôleur d'admission Pod Security. Les alternatives tierces qui existent incluent OPA/Gatekeeper Kyverno. Une collection de contraintes et de modèles de contraintes pour la mise en œuvre des politiques que l'on trouve couramment dans le référentiel de la bibliothèque Gatekeeper PSPs peut être extraite du référentiel de la bibliothèque Gatekeeper sur. GitHub Et de nombreuses solutions de remplacement se PSPs trouvent dans la bibliothèque des politiques de Kyverno, y compris la collection complète des normes de sécurité des pods.

Outils et ressources

- 7 choses à savoir avant de commencer
- AppArmorChargeur
- Configuration de nœuds avec des profils
- L'opérateur de profils de sécurité est une amélioration de Kubernetes qui vise à faciliter l'utilisation par les utilisateurs SELinux, seccomp et dans les clusters Kubernetes. AppArmor Il fournit des fonctionnalités permettant à la fois de générer des profils à partir de charges de travail en cours d'exécution et de charger des profils sur des nœuds Kubernetes pour les utiliser dans des pods.
- Inspektor Gadget permet d'inspecter, de suivre et de profiler de nombreux aspects du comportement d'exécution sur Kubernetes, notamment en aidant à la génération de profils seccomp.
- Aqua
- Qualys
- Stackrox
- Sysdig Secure
- Prisme
- NeuVector de SUSE Open Source, plate-forme de sécurité des conteneurs Zero-Trust, fournit des règles de profil de processus et des règles d'accès aux fichiers.

Protection de l'infrastructure (hôtes)

Dans la mesure où il est important de sécuriser les images de vos conteneurs, il est tout aussi important de protéger l'infrastructure qui les gère. Cette section explore différentes manières d'atténuer les risques liés aux attaques lancées directement contre l'hôte. Ces directives doivent être utilisées conjointement avec celles décrites dans la section Runtime Security.

Recommandations

Utiliser un système d'exploitation optimisé pour exécuter des conteneurs

Envisagez d'utiliser Flatcar Linux, Project Atomic, RancherOS et Bottlerocket, un système d'exploitation spécial d'AWS conçu pour exécuter des conteneurs Linux. Cela inclut une surface d'attaque réduite, une image disque vérifiée au démarrage et des limites d'autorisation imposées à l'aide de SELinux.

Vous pouvez également utiliser l'AMI optimisée pour EKS pour vos nœuds de travail Kubernetes. L'AMI optimisée pour EKS est publiée régulièrement et contient un ensemble minimal de packages de système d'exploitation et de fichiers binaires nécessaires pour exécuter vos charges de travail conteneurisées.

Reportez-vous à la spécification de construction RHEL de l'AMI Amazon EKS pour un exemple de script de configuration qui peut être utilisé pour créer une AMI Amazon EKS personnalisée exécutée sur Red Hat Enterprise Linux à l'aide de Hashicorp Packer. Ce script peut être davantage exploité pour créer un EKS personnalisé conforme aux STIG. AMIs

Maintenez le système d'exploitation de votre nœud de travail à jour

Que vous utilisiez un système d'exploitation hôte optimisé pour les conteneurs tel que Bottlerocket ou un système Amazon Machine Image plus grand, mais néanmoins minimaliste, comme l'EKS optimisé AMIs, il est recommandé de maintenir ces images de système d'exploitation hôte à jour avec les derniers correctifs de sécurité.

Pour optimiser l'EKS AMIs, consultez régulièrement le CHANGELOG et/ou le canal des notes de version et automatisez le déploiement des images de nœuds de travail mises à jour dans votre cluster.

Traitez votre infrastructure comme immuable et automatisez le remplacement de vos nœuds de travail

Plutôt que d'effectuer des mises à niveau sur place, remplacez vos employés lorsqu'un nouveau correctif ou une nouvelle mise à jour est disponible. Cela peut être abordé de plusieurs manières. Vous pouvez soit ajouter des instances à un groupe de mise à l'échelle automatique existant à l'aide de la dernière AMI, en bouclant et en vidant les nœuds de manière séquentielle jusqu'à ce que tous les nœuds du groupe soient remplacés par la dernière AMI. Vous pouvez également ajouter des instances à un nouveau groupe de nœuds tout en bouclant et en vidant séquentiellement les nœuds de l'ancien groupe de nœuds jusqu'à ce que tous les nœuds aient été remplacés. Les groupes de nœuds gérés par EKS utilisent la première approche et affichent un message dans la console pour mettre à niveau vos travailleurs lorsqu'une nouvelle AMI sera disponible. `eksctl`dispose également d'un mécanisme permettant de créer des groupes de nœuds avec l'AMI la plus récente et de boucler et de vidanger les pods des groupes de nœuds avant que les instances ne soient mises hors service. Si vous décidez d'utiliser une autre méthode pour remplacer vos nœuds de travail, il est vivement recommandé d'automatiser le processus afin de minimiser la supervision humaine, car vous devrez probablement remplacer les travailleurs régulièrement au fur et à mesure que de nouveaux nœuds seront updates/patches publiés et lorsque le plan de contrôle sera mis à niveau.

Avec EKS Fargate, AWS met automatiquement à jour l'infrastructure sous-jacente au fur et à mesure que des mises à jour sont disponibles. Cela peut souvent se faire sans problème, mais il peut arriver qu'une mise à jour entraîne le report de votre pod. Nous vous recommandons donc de créer des déploiements avec plusieurs répliques lorsque vous exécutez votre application en tant que pod Fargate.

Exécutez régulièrement kube-bench pour vérifier la conformité aux benchmarks CIS pour Kubernetes

kube-bench est un projet open source d'Aqua qui évalue votre cluster par rapport aux benchmarks CIS pour Kubernetes. Le benchmark décrit les meilleures pratiques pour sécuriser les clusters Kubernetes non gérés. Le CIS Kubernetes Benchmark englobe le plan de contrôle et le plan de données. Amazon EKS fournissant un plan de contrôle entièrement géré, toutes les recommandations du CIS Kubernetes Benchmark ne sont pas applicables. Pour s'assurer que cette portée reflète la manière dont Amazon EKS est implémenté, AWS a créé le CIS Amazon EKS Benchmark. Le benchmark EKS hérite de CIS Kubernetes Benchmark avec des contributions supplémentaires de la communauté avec des considérations de configuration spécifiques pour les clusters EKS.

Lorsque vous exécutez kube-bench sur un cluster EKS, suivez les instructions d'Aqua Security. Pour plus d'informations, voir Présentation du benchmark Amazon EKS du CIS.

Minimiser l'accès aux nœuds de travail

Au lieu d'activer l'accès SSH, utilisez le gestionnaire de session SSM lorsque vous devez vous connecter à distance à un hôte. Contrairement aux clés SSH qui peuvent être perdues, copiées ou partagées, le gestionnaire de session vous permet de contrôler l'accès aux instances EC2 à l'aide d'IAM. En outre, il fournit une piste d'audit et un journal des commandes exécutées sur l'instance.

Depuis le 19 août 2020, les groupes de nœuds gérés prennent en charge les AMI personnalisés et les modèles de lancement EC2. Cela vous permet d'intégrer l'agent SSM dans l'AMI ou de l'installer pendant le démarrage du nœud de travail. Si vous préférez ne pas modifier l'AMI optimisée ou le modèle de lancement de l'ASG, vous pouvez installer l'agent SSM avec un DaemonSet as dans cet exemple.

Politique IAM minimale pour l'accès SSH basé sur SSM

La politique gérée par `AmazonSSMManagedInstanceCore` AWS contient un certain nombre d'autorisations qui ne sont pas requises pour le gestionnaire de session SSM/SSM RunCommand si vous souhaitez simplement éviter l'accès SSH. Ce qui est particulièrement préoccupant, c'est la

question des * autorisations `ssm:GetParameter(s)` qui permettraient au rôle d'accéder à tous les paramètres du Parameter Store (y compris SecureStrings lorsque la clé KMS gérée par AWS est configurée).

La politique IAM suivante contient l'ensemble minimal d'autorisations pour permettre l'accès aux nœuds via SSM Systems Manager.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "EnableAccessViaSSMSessionManager",
      "Effect": "Allow",
      "Action": [
        "ssmmessages:OpenDataChannel",
        "ssmmessages:OpenControlChannel",
        "ssmmessages:CreateDataChannel",
        "ssmmessages:CreateControlChannel",
        "ssm:UpdateInstanceInformation"
      ],
      "Resource": "*"
    },
    {
      "Sid": "EnableSSMRunCommand",
      "Effect": "Allow",
      "Action": [
        "ssm:UpdateInstanceInformation",
        "ec2messages:SendReply",
        "ec2messages:GetMessages",
        "ec2messages:GetEndpoint",
        "ec2messages:FailMessage",
        "ec2messages:DeleteMessage",
        "ec2messages:AcknowledgeMessage"
      ],
      "Resource": "*"
    }
  ]
}
```

Une fois cette politique en place et le plug-in Session Manager installé, vous pouvez ensuite exécuter

```
aws ssm start-session --target [INSTANCE_ID_OF_EKS_NODE]
```

pour accéder au nœud.

> **Note**
>
> Vous pouvez également envisager d'ajouter des autorisations pour activer la journalisation du gestionnaire de session.

Déployer des employés sur des sous-réseaux privés

En déployant des employés sur des sous-réseaux privés, vous minimisez leur exposition à Internet, d'où proviennent souvent les attaques. À compter du 22 avril 2020, l'attribution d'adresses IP publiques aux nœuds d'un groupe de nœuds gérés sera contrôlée par le sous-réseau sur lequel ils sont déployés. Auparavant, une adresse IP publique était automatiquement attribuée aux nœuds d'un groupe de nœuds gérés. Si vous choisissez de déployer vos nœuds de travail sur des sous-réseaux publics, mettez en œuvre des règles restrictives relatives aux groupes de sécurité AWS afin de limiter leur exposition.

Exécutez Amazon Inspector pour évaluer l'exposition des hôtes, les vulnérabilités et les écarts par rapport aux meilleures pratiques

Vous pouvez utiliser Amazon Inspector pour vérifier l'absence d'accès réseau involontaire à vos nœuds et les vulnérabilités des instances Amazon EC2 sous-jacentes.

Amazon Inspector peut fournir des données sur les vulnérabilités et les expositions communes (CVE) pour vos instances Amazon EC2 uniquement si l'agent Amazon EC2 Systems Manager (SSM) est installé et activé. Cet agent est préinstallé sur plusieurs Amazon Machine Images (AMIs), y compris Amazon Linux AMIs optimisé pour EKS. Quel que soit le statut de l'agent SSM, toutes vos instances Amazon EC2 sont analysées pour détecter les problèmes d'accessibilité au réseau. Pour plus d'informations sur la configuration des scans pour Amazon EC2, consultez Numérisation des instances Amazon EC2.

> **Important**
>
> Inspector ne peut pas être exécuté sur l'infrastructure utilisée pour exécuter les pods Fargate.

Solutions de rechange

Courir SELinux

> **Note**
>
> Disponible sur Red Hat Enterprise Linux (RHEL), CentOS, Bottlerocket et Amazon Linux 2023

SELinux fournit un niveau de sécurité supplémentaire pour isoler les conteneurs les uns des autres et de l'hôte. SELinux permet aux administrateurs d'appliquer des contrôles d'accès obligatoires (MAC) pour chaque utilisateur, application, processus et fichier. Considérez-le comme un filet de sécurité qui limite les opérations pouvant être effectuées sur des ressources spécifiques en fonction d'un ensemble d'étiquettes. Sur EKS, il SELinux peut être utilisé pour empêcher les conteneurs d'accéder aux ressources des autres.

Les SELinux politiques de conteneur sont définies dans le package container-selinux. Docker CE nécessite ce package (ainsi que ses dépendances) afin que les processus et les fichiers créés par Docker (ou d'autres environnements d'exécution de conteneurs) s'exécutent avec un accès système limité. Les conteneurs tirent parti de l'`container_t`étiquette qui est un alias de`svirt_lxc_net_t`. Ces politiques empêchent efficacement les conteneurs d'accéder à certaines fonctionnalités de l'hôte.

Lorsque vous configurez SELinux pour Docker, Docker étiquette automatiquement les charges de travail `container_t` par type et attribue à chaque conteneur un niveau MCS unique. Cela permettra d'isoler les conteneurs les uns des autres. Si vous avez besoin de restrictions plus souples, vous pouvez créer votre propre profil SElinux qui accorde à un conteneur des autorisations sur des zones spécifiques du système de fichiers. Ceci est similaire PSPs au fait que vous pouvez créer différents profils pour différents conteneurs/pods. Par exemple, vous pouvez avoir un profil pour les charges de travail générales avec un ensemble de contrôles restrictifs et un autre pour les éléments nécessitant un accès privilégié.

SELinux for Containers dispose d'un ensemble d'options qui peuvent être configurées pour modifier les restrictions par défaut. Les SELinux booléens suivants peuvent être activés ou désactivés en fonction de vos besoins :

Booléen	Par défaut	Description
`container_connect_any`	`off`	Autorisez les conteneurs à accéder aux ports privilégiés de l'hôte. Par exemple, si vous avez un conteneur qui doit mapper les ports vers 443 ou 80 sur l'hôte.
`container_manage_cgroup`	`off`	Permettre aux conteneurs de gérer la configuration des cgroups. Par exemple, un conteneur exécutant systemd devra être activé.
`container_use_cephfs`	`off`	Autoriser les conteneurs à utiliser un système de fichiers ceph.

Par défaut, les conteneurs sont autorisés à lire read/execute `/usr` et à lire la majeure partie du contenu`/etc`. Les fichiers se trouvent sous `/var/lib/docker` et `/var/lib/containers` portent l'étiquette`container_var_lib_t`. Pour afficher la liste complète des libellés par défaut, consultez le fichier container.fc.

```
docker container run -it \
  -v /var/lib/docker/image/overlay2/repositories.json:/host/repositories.json \
  centos:7 cat /host/repositories.json
# cat: /host/repositories.json: Permission denied

docker container run -it \
  -v /etc/passwd:/host/etc/passwd \
  centos:7 cat /host/etc/passwd
# cat: /host/etc/passwd: Permission denied
```

Les fichiers étiquetés avec `container_file_t` sont les seuls fichiers accessibles en écriture par les conteneurs. Si vous souhaitez qu'un montage de volume soit inscriptible, vous devez spécifier `:z` ou `:Z` à la fin.

- : zréétiquettera les fichiers afin que le conteneur puisse lire/écrire
- : Zréétiquettera les fichiers afin que seul le conteneur puisse lire/écrire

```
ls -Z /var/lib/misc
# -rw-r--r--. root root system_u:object_r:var_lib_t:s0   postfix.aliasesdb-stamp

docker container run -it \
  -v /var/lib/misc:/host/var/lib/misc:z \
  centos:7 echo "Relabeled!"

ls -Z /var/lib/misc
#-rw-r--r--. root root system_u:object_r:container_file_t:s0 postfix.aliasesdb-stamp
```

```
docker container run -it \
  -v /var/log:/host/var/log:Z \
  fluentbit:latest
```

Dans Kubernetes, le réétiquetage est légèrement différent. Plutôt que de laisser Docker réétiqueter automatiquement les fichiers, vous pouvez spécifier une étiquette MCS personnalisée pour exécuter le pod. Les volumes compatibles avec le réétiquetage seront automatiquement réétiquetés afin d'être accessibles. Les pods dotés d'une étiquette MCS correspondante pourront accéder au volume. Si vous avez besoin d'une isolation stricte, définissez une étiquette MCS différente pour chaque pod.

```
securityContext:
  seLinuxOptions:
    # Provide a unique MCS label per container
    # You can specify user, role, and type also
    # enforcement based on type and level (svert)
    level: s0:c144:c154
```

Dans cet exemple, cela `s0:c144:c154` correspond à une étiquette MCS attribuée à un fichier auquel le conteneur est autorisé à accéder.

Sur EKS, vous pouvez créer des politiques qui permettent l'exécution de conteneurs privilégiés, comme FluentD, et créer SELinux une politique pour lui permettre de lire depuis /var/log sur l'hôte sans avoir à renommer le répertoire hôte. Les pods portant le même label pourront accéder aux mêmes volumes hôtes.

Nous avons implémenté un exemple AMIs pour Amazon EKS SELinux configuré sur CentOS 7 et RHEL 7. Ils AMIs ont été développés pour démontrer des exemples de mises en œuvre répondant aux exigences de clients hautement réglementés.

> **Warning**
>
> SELinux ignorera les conteneurs dont le type n'est pas confiné.

Outils et ressources

- SELinuxRBAC Kubernetes et politiques de sécurité d'expédition pour les applications sur site
- Renforcement itératif de Kubernetes
- Audit 2 Autoriser
- SEAlert
- Générer des SELinux politiques pour les conteneurs avec Udica décrit un outil qui examine les fichiers de spécifications des conteneurs pour les fonctionnalités, les ports et les points de montage de Linux, et génère un ensemble de SELinux règles permettant au conteneur de fonctionner correctement
- Des playbooks de renforcement de l'AMI pour renforcer le système d'exploitation afin de répondre aux différentes exigences réglementaires
- Keiko Upgrade Manager est un projet open source d'Intuit qui orchestre la rotation des nœuds de travail.
- Sysdig Secure
- eksctl

Conformité d’

La conformité est une responsabilité partagée entre AWS et les consommateurs de ses services. D'une manière générale, AWS est responsable de la « sécurité du cloud » alors que ses utilisateurs sont responsables de la « sécurité dans le cloud ». La ligne qui définit les responsabilités d'AWS et de ses utilisateurs varie en fonction du service. Par exemple, avec Fargate, AWS est responsable de la gestion de la sécurité physique de ses centres de données, du matériel, de l'infrastructure virtuelle (Amazon EC2) et de l'environnement d'exécution des conteneurs (Docker). Les utilisateurs de Fargate sont responsables de la sécurisation de l'image du conteneur et de son application.

Savoir qui est responsable de quoi est un élément important à prendre en compte lors de l'exécution de charges de travail qui doivent respecter les normes de conformité.

Le tableau suivant présente les programmes de conformité auxquels se conforment les différents services de conteneurs.

Programme de conformité	Amazon ECS Orchestrator	Amazon EKS Orchestrator	Fargate ECS	Amazon ECR
PCI DSS niveau 1	1	1	1	1
Éligible HIPAA	1	1	1	1
SOC I	1	1	1	1
SOC II	1	1	1	1
SOC III	1	1	1	1
ISO 27001:2013	1	1	1	1
ISO 9001:2015	1	1	1	1
ISO 27017:2015	1	1	1	1
ISO 27018:2019	1	1	1	1
IRAP	1	1	1	1
FedRAMP Modérée (Est/Ouest)	1	1	0	1
FedRAMP Élevé () GovCloud	1	1	0	1
DOD CC SRG	1	Revue DISA () IL5	0	1
HIPAA BAA	1	1	1	1
MTCS	1	1	0	1

Programme de conformité	Amazon ECS Orchestrator	Amazon EKS Orchestrator	Fargate ECS	Amazon ECR
C5	1	1	0	1
K-ISMS	1	1	0	1
ENS High	1	1	0	1
OSPAR	1	1	0	1
HITRUST CSF	1	1	1	1

L'état de conformité évolue au fil du temps. Pour connaître le statut le plus récent, reportez-vous toujours à https://aws.amazon.com/compliance/ services-in-scope /.

Pour plus d'informations sur les modèles d'accréditation du cloud et les meilleures pratiques, consultez le livre blanc d'AWS, Modèles d'accréditation pour une adoption sécurisée du cloud

Déplacement vers la gauche

Le concept du déplacement vers la gauche implique de détecter les violations des politiques et les erreurs plus tôt dans le cycle de vie du développement logiciel. Du point de vue de la sécurité, cela peut être très bénéfique. Un développeur, par exemple, peut résoudre les problèmes liés à sa configuration avant que son application ne soit déployée sur le cluster. En détectant de telles erreurs plus tôt, vous éviterez le déploiement de configurations qui enfreignent vos politiques.

La politique en tant que code

La politique peut être considérée comme un ensemble de règles régissant les comportements, c'est-à-dire les comportements autorisés ou interdits. Par exemple, vous pouvez avoir une politique stipulant que tous les Dockerfiles doivent inclure une directive USER qui permet au conteneur de s'exécuter en tant qu'utilisateur non root. En tant que document, une telle politique peut être difficile à découvrir et à appliquer. Il peut également devenir obsolète à mesure que vos besoins changent. Avec les solutions Policy as Code (PaC), vous pouvez automatiser les contrôles de sécurité, de conformité et de confidentialité qui détectent, préviennent, réduisent et neutralisent les menaces connues et persistantes. En outre, ils vous fournissent un mécanisme pour codifier vos politiques et les gérer comme vous le feriez pour les autres artefacts de code. L'avantage de cette approche est que vous pouvez réutiliser vos GitOps stratégies DevOps et pour gérer et appliquer de manière

cohérente des politiques dans les flottes de clusters Kubernetes. Veuillez consulter Pod Security pour plus d'informations sur les options PaC et le futur de PSPs.

Utiliser policy-as-code des outils dans les pipelines pour détecter les violations avant le déploiement

- OPA est un moteur de politique open source qui fait partie de la CNCF. Il est utilisé pour prendre des décisions politiques et peut être géré de différentes manières, par exemple en tant que bibliothèque de langues ou en tant que service. Les politiques OPA sont rédigées dans un langage spécifique au domaine (DSL) appelé Rego. Bien qu'il soit souvent exécuté dans le cadre d'un contrôleur d'admission dynamique Kubernetes en tant que projet Gatekeeper, l'OPA peut également être intégré à votre pipeline. CI/CD Cela permet aux développeurs d'obtenir des commentaires sur leur configuration plus tôt dans le cycle de publication, ce qui peut les aider à résoudre les problèmes avant de passer à la production. Une collection de politiques OPA communes se trouve dans le GitHub référentiel de ce projet.
- Conftest est basé sur OPA et fournit une expérience axée sur les développeurs pour tester la configuration de Kubernetes.
- Kyverno est un moteur de politiques conçu pour Kubernetes. Avec Kyverno, les politiques sont gérées comme des ressources Kubernetes et aucun nouveau langage n'est nécessaire pour écrire des politiques. Cela permet d'utiliser des outils familiers tels que kubectl, git et kustomize pour gérer les politiques. Les politiques de Kyverno peuvent valider, modifier et générer des ressources Kubernetes, tout en garantissant la sécurité de la chaîne d'approvisionnement en images OCI. La CLI Kyverno peut être utilisée pour tester les politiques et valider les ressources dans le cadre d'un CI/CD pipeline. Toutes les politiques de la communauté Kyverno sont disponibles sur le site Web de Kyverno, et pour des exemples d'utilisation de la CLI Kyverno pour écrire des tests dans des pipelines, consultez le référentiel des politiques.

Outils et ressources

- Atelier d'immersion sur la sécurité Amazon EKS - Conformité réglementaire
- banc Kube
- docker-bench-security
- AWS Inspector
- Examen de la sécurité de Kubernetes Une évaluation de la sécurité de Kubernetes 1.13.4 (2019) réalisée par un tiers

- NeuVector par SUSE, plateforme open source de sécurité des conteneurs Zero-Trust, fournit des rapports de conformité et des contrôles de conformité personnalisés

Réponse aux incidents et criminalistique

Votre capacité à réagir rapidement à un incident peut contribuer à minimiser les dommages causés par une violation. Disposer d'un système d'alerte fiable capable de vous avertir en cas de comportement suspect est la première étape d'un bon plan de réponse aux incidents. En cas d'incident, vous devez rapidement décider de détruire et de remplacer le contenant concerné ou d'isoler et d'inspecter le contenant. Si vous choisissez d'isoler le contenant dans le cadre d'une enquête médico-légale et d'une analyse des causes profondes, les activités suivantes doivent être suivies :

Exemple de plan de réponse aux incidents

Identifiez le pod et le nœud de travail incriminés

Votre premier plan d'action devrait être d'isoler les dégâts. Commencez par identifier l'endroit où la violation s'est produite et isolez ce Pod et son nœud du reste de l'infrastructure.

Identifiez les pods et nœuds de travail incriminés à l'aide du nom de la charge de travail

Si vous connaissez le nom et l'espace de noms du pod incriminé, vous pouvez identifier le nœud de travail exécutant le pod comme suit :

```
kubectl get pods <name> --namespace <namespace> -o=jsonpath='{.spec.nodeName}{"\n"}'
```

Si une ressource de charge de travail telle qu'un déploiement a été compromise, il est probable que tous les pods faisant partie de la ressource de charge de travail soient compromis. Utilisez la commande suivante pour répertorier tous les pods de la ressource de charge de travail et les nœuds sur lesquels ils s'exécutent :

```
selector=$(kubectl get deployments <name> \
 --namespace <namespace> -o json | jq -j \
'.spec.selector.matchLabels | to_entries | .[] | "\(.key)=\(.value)"')

kubectl get pods --namespace <namespace> --selector=$selector \
```

```
-o json | jq -r '.items[] | "\(.metadata.name) \(.spec.nodeName)"'
```

La commande ci-dessus concerne les déploiements. Vous pouvez exécuter la même commande pour d'autres ressources de charge de travail telles que les replicasets, les statefulsets, etc.

Identifiez les pods et nœuds de travail incriminés à l'aide du nom du compte de service

Dans certains cas, vous pouvez identifier qu'un compte de service est compromis. Il est probable que les pods utilisant le compte de service identifié soient compromis. Vous pouvez identifier tous les pods à l'aide du compte de service et des nœuds sur lesquels ils s'exécutent à l'aide de la commande suivante :

```
kubectl get pods -o json --namespace <namespace> | \
    jq -r '.items[] |
    select(.spec.serviceAccount == "<service account name>") |
    "\(.metadata.name) \(.spec.nodeName)"'
```

Identifiez les pods dont les images et les nœuds de travail sont vulnérables ou compromis

Dans certains cas, vous découvrirez peut-être qu'une image de conteneur utilisée dans les modules de votre cluster est malveillante ou compromise. Une image de conteneur est malveillante ou compromise si elle contient un logiciel malveillant, s'il s'agit d'une image défectueuse connue ou si un CVE a été exploité. Vous devez considérer que tous les pods utilisant l'image du conteneur sont compromis. Vous pouvez identifier les modules à l'aide de l'image et des nœuds sur lesquels ils s'exécutent à l'aide de la commande suivante :

```
IMAGE=<Name of the malicious/compromised image>

kubectl get pods -o json --all-namespaces | \
    jq -r --arg image "$IMAGE" '.items[] |
    select(.spec.containers[] | .image == $image) |
    "\(.metadata.name) \(.metadata.namespace) \(.spec.nodeName)"'
```

Isolez le pod en créant une politique réseau qui interdit tout trafic entrant et sortant vers le pod

Une règle interdisant tout trafic peut aider à stopper une attaque déjà en cours en coupant toutes les connexions au module. La politique réseau suivante s'appliquera à un module portant l'étiquette`app=web`.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector:
    matchLabels:
      app: web
  policyTypes:
  - Ingress
  - Egress
```

> **Important**
>
> Une politique réseau peut s'avérer inefficace si un attaquant a accédé à l'hôte sous-jacent. Si vous pensez que cela s'est produit, vous pouvez utiliser les groupes de sécurité AWS pour isoler un hôte compromis des autres hôtes. Lorsque vous modifiez le groupe de sécurité d'un hôte, sachez que cela aura un impact sur tous les conteneurs exécutés sur cet hôte.

Révoquez les informations d'identification de sécurité temporaires attribuées au pod ou au nœud de travail si nécessaire

Si un rôle IAM a été attribué au nœud de travail qui permet aux Pods d'accéder à d'autres ressources AWS, supprimez ces rôles de l'instance pour éviter que l'attaque ne cause de nouveaux dommages. De même, si un rôle IAM a été attribué au pod, déterminez si vous pouvez supprimer les politiques IAM du rôle en toute sécurité sans affecter les autres charges de travail.

Cordon le nœud du travailleur

En bouclant le nœud de travail concerné, vous informez le planificateur afin d'éviter de planifier des pods sur le nœud concerné. Cela vous permettra de supprimer le nœud pour une étude médico-légale sans perturber les autres charges de travail.

> **Note**
>
> Ce guide ne s'applique pas à Fargate, où chaque pod Fargate s'exécute dans son propre environnement sandbox. Au lieu de les boucler, séquestrez les pods Fargate concernés en appliquant une politique réseau qui interdit tout trafic entrant et sortant.

Activer la protection contre le licenciement sur le nœud de travail concerné

Un attaquant peut tenter d'effacer ses méfaits en mettant fin à un nœud affecté. L'activation de la protection contre le licenciement peut empêcher que cela ne se produise. La protection d'extension de l'instance protégera le nœud d'un événement de montée en puissance.

> **Warning**
>
> Vous ne pouvez pas activer la protection contre la résiliation sur une instance Spot.

Étiquetez l'infraction Pod/Node avec une étiquette indiquant qu'elle fait partie d'une enquête en cours

Cela servira d'avertissement aux administrateurs du cluster pour qu'ils ne modifient pas les données concernées Pods/Nodes tant que l'enquête n'est pas terminée.

Capturez les artefacts volatils sur le nœud de travail

- Capturez la mémoire du système d'exploitation. Cela capturera le démon Docker (ou un autre environnement d'exécution de conteneur) et ses sous-processus par conteneur. Cela peut être accompli à l'aide d'outils tels que LiME et Volatility, ou d'outils de niveau supérieur tels que Automated Forensics Orchestrator pour Amazon EC2 qui s'appuient sur ces outils.
- Effectuez un vidage de l'arborescence Netstat des processus en cours d'exécution et des ports ouverts. Cela capturera le démon docker et son sous-processus par conteneur.
- Exécutez des commandes pour enregistrer l'état au niveau du conteneur avant que les preuves ne soient modifiées. Vous pouvez utiliser les fonctionnalités de l'environnement d'exécution des conteneurs pour capturer des informations sur les conteneurs en cours d'exécution. Par exemple, avec Containerd, vous pouvez effectuer les opérations suivantes :
  - `crictl ps`pour les processus en cours d'exécution.
  - `crictl logs CONTAINER`pour les journaux conservés au niveau du démon.

  La même chose pourrait être réalisée avec containerd en utilisant la CLI nerdctl, à la place `docker` de (par exemple). `nerdctl inspect` Certaines commandes supplémentaires sont disponibles en fonction de l'environnement d'exécution du conteneur. Par exemple, Docker doit `docker diff` voir les modifications apportées au système de fichiers du conteneur ou `docker checkpoint` enregistrer tous les états du conteneur, y compris la mémoire volatile (RAM).

Consultez ce billet de blog sur Kubernetes pour découvrir des fonctionnalités similaires avec les environnements d'exécution containerd ou CRI-O.

- Mettez le conteneur en pause pour une capture médico-légale.
- Capturez les volumes EBS de l'instance.

Redéployer une ressource d'espace ou de charge de travail compromise

Une fois que vous avez collecté les données à des fins d'analyse judiciaire, vous pouvez redéployer le pod ou la ressource de charge de travail compromis.

Déployez d'abord le correctif pour corriger la vulnérabilité qui a été compromise et lancez de nouveaux modules de remplacement. Supprimez ensuite les pods vulnérables.

Si les pods vulnérables sont gérés par une ressource de charge de travail Kubernetes de niveau supérieur (par exemple, un déploiement ou DaemonSet), leur suppression en planifiera de nouveaux. Les pods vulnérables seront donc à nouveau lancés. Dans ce cas, vous devez déployer une nouvelle ressource de charge de travail de remplacement après avoir corrigé la vulnérabilité. Vous devez ensuite supprimer la charge de travail vulnérable.

Recommandations

Consultez le livre blanc AWS sur la réponse aux incidents de sécurité

Bien que cette section donne un bref aperçu ainsi que quelques recommandations pour traiter les failles de sécurité présumées, le sujet est traité de manière exhaustive dans le white paper, AWS Security Incident Response.

Journées d'entraînement aux jeux de sécurité

Divisez vos professionnels de la sécurité en deux équipes : rouge et bleue. L'équipe rouge se concentrera sur l'analyse des vulnérabilités des différents systèmes, tandis que l'équipe bleue sera chargée de s'en défendre. Si vous ne disposez pas de suffisamment de professionnels de la sécurité pour créer des équipes distinctes, envisagez de faire appel à une entité externe ayant connaissance des exploits de Kubernetes.

Kubesploit est un framework de test d'intrusion CyberArk que vous pouvez utiliser pour organiser des journées de jeu. Contrairement aux autres outils qui analysent les vulnérabilités de votre cluster, kubesploit simule une attaque réelle. Cela donne à votre équipe bleue l'occasion de mettre en pratique sa réponse à une attaque et d'évaluer son efficacité.

Exécutez des tests de pénétration sur votre cluster

Attaquer régulièrement votre propre cluster peut vous aider à découvrir des vulnérabilités et des erreurs de configuration. Avant de commencer, suivez les directives relatives aux tests d'intrusion avant d'effectuer un test sur votre cluster.

Outils et ressources

- kube-hunter, un outil de test de pénétration pour Kubernetes.
- G705, une boîte à outils d'ingénierie du chaos que vous pouvez utiliser pour simuler des attaques contre vos applications et votre infrastructure.
- Attaquer et défendre les installations Kubernetes
- kubesploit
- NeuVector de SUSE, plateforme open source de sécurité des conteneurs Zero-Trust, fournit des rapports sur les vulnérabilités et les risques ainsi que des notifications d'événements de sécurité
- Menaces persistantes avancées
- Attaque et défense pratiques de Kubernetes
- Compromettre le cluster Kubernetes en exploitant les autorisations RBAC

Sécurité de l'image

Vous devez considérer l'image du conteneur comme la première ligne de défense contre une attaque. Une image peu sécurisée et mal construite peut permettre à un attaquant d'échapper aux limites du conteneur et d'accéder à l'hôte. Une fois sur l'hôte, un attaquant peut accéder à des informations sensibles ou se déplacer latéralement au sein du cluster ou avec votre compte AWS. Les meilleures pratiques suivantes aideront à atténuer le risque que cela se produise.

Recommandations

Créez un minimum d'images

Commencez par retirer tous les fichiers binaires superflus de l'image de conteneur. Si vous utilisez une image inconnue provenant de Dockerhub, inspectez-la à l'aide d'une application telle que Dive qui peut vous montrer le contenu de chacune des couches du conteneur. Supprimez tous les fichiers binaires contenant les bits SETUID et SETGID, car ils peuvent être utilisés pour augmenter les

privilèges, et envisagez de supprimer tous les shells et utilitaires tels que nc et curl qui peuvent être utilisés à des fins malveillantes. Vous pouvez trouver les fichiers contenant les bits SETUID et SETGID à l'aide de la commande suivante :

```
find / -perm /6000 -type f -exec ls -ld {} \;
```

Pour supprimer les autorisations spéciales associées à ces fichiers, ajoutez la directive suivante à votre image de conteneur :

```
RUN find / -xdev -perm /6000 -type f -exec chmod a-s {} \; || true
```

C'est ce que l'on appelle familièrement la modification de votre image.

Utilisez des versions en plusieurs étapes

L'utilisation de builds en plusieurs étapes est un moyen de créer des images minimales. Souvent, des versions en plusieurs étapes sont utilisées pour automatiser certaines parties du cycle d'intégration continue. Par exemple, les versions en plusieurs étapes peuvent être utilisées pour affiner votre code source ou effectuer une analyse de code statique. Cela permet aux développeurs d'obtenir des commentaires quasi immédiats au lieu d'attendre l'exécution d'un pipeline. Les builds en plusieurs étapes sont intéressants du point de vue de la sécurité car ils vous permettent de minimiser la taille de l'image finale envoyée à votre registre de conteneurs. Les images de conteneur dépourvues d'outils de construction et d'autres fichiers binaires superflus améliorent votre niveau de sécurité en réduisant la surface d'attaque de l'image. Pour plus d'informations sur les versions en plusieurs étapes, consultez la documentation des versions en plusieurs étapes de Docker.

Créez une nomenclature logicielle (SBOMs) pour l'image de votre conteneur

Une « nomenclature logicielle » (SBOM) est un inventaire imbriqué des artefacts logiciels qui constituent l'image de votre conteneur. Le SBOM est un élément clé de la sécurité logicielle et de la gestion des risques liés à la chaîne d'approvisionnement logicielle. La génération, le stockage des SBOM dans un référentiel central et l'analyse SBOMs des vulnérabilités permettent de résoudre les problèmes suivants :

- Visibilité : déterminez quels composants constituent l'image de votre conteneur. Le stockage dans un référentiel central permet d'être audité et scanné SBOMs à tout moment, même après le déploiement, afin de détecter les nouvelles vulnérabilités telles que les vulnérabilités de type « jour zéro » et d'y répondre.

- Vérification de la provenance : assurance que les hypothèses existantes concernant l'origine et la provenance d'un artefact sont vraies et que l'artefact ou les métadonnées qui l'accompagnent n'ont pas été falsifiés pendant les processus de construction ou de livraison.
- Fiabilité : assurance que l'on peut faire confiance à un artefact donné et à son contenu pour faire ce qu'il est censé faire, c'est-à-dire qu'ils sont adaptés à un objectif précis. Cela implique de déterminer si le code peut être exécuté en toute sécurité et de prendre des décisions éclairées quant aux risques associés à l'exécution du code. La fiabilité est garantie par la création d'un rapport d'exécution du pipeline attesté ainsi que d'un rapport de scan SBOM attesté et d'un rapport de scan CVE attesté afin de garantir aux consommateurs de l'image que cette image est en fait créée par des moyens sécurisés (pipeline) avec des composants sécurisés.
- Vérification de la confiance dans les dépendances : vérification récursive de l'arbre de dépendances d'un artefact pour vérifier la fiabilité et la provenance des artefacts qu'il utilise. Drift in SBOMs peut aider à détecter les activités malveillantes, notamment les dépendances non autorisées et non fiables et les tentatives d'infiltration.

Les outils suivants peuvent être utilisés pour générer des SBOM :

- Amazon Inspector peut être utilisé pour créer et exporter SBOMs.
- Syft d'Anchore peut également être utilisé pour la génération de SBOM. Pour accélérer les analyses de vulnérabilité, le SBOM généré pour une image de conteneur peut être utilisé comme entrée pour le scan. Le SBOM et le rapport de numérisation sont ensuite attestés et joints à l'image avant de transférer l'image vers un référentiel OCI central tel qu'Amazon ECR à des fins de révision et d'audit.

Pour en savoir plus sur la sécurisation de votre chaîne d'approvisionnement logicielle, consultez le guide des meilleures pratiques de la chaîne d'approvisionnement logicielle de la CNCF.

Scannez régulièrement les images pour détecter les vulnérabilités

Comme leurs homologues de machines virtuelles, les images de conteneur peuvent contenir des fichiers binaires et des bibliothèques d'applications présentant des vulnérabilités ou développer des vulnérabilités au fil du temps. Le meilleur moyen de se prémunir contre les attaques est d'analyser régulièrement vos images à l'aide d'un analyseur d'images. Les images stockées dans Amazon ECR peuvent être numérisées en mode push ou à la demande (une fois par période de 24 heures). L'ECR prend actuellement en charge deux types de numérisation : de base et améliorée. La numérisation de base tire parti de Clair, une solution de numérisation d'images open source gratuite. Le scan

amélioré utilise Amazon Inspector pour fournir des scans continus automatiques moyennant des frais supplémentaires. Une fois qu'une image est numérisée, les résultats sont enregistrés dans EventBridge le flux d'événements pour être intégrés à l'ECR. Vous pouvez également consulter les résultats d'un scan depuis la console ECR. Les images présentant une vulnérabilité ÉLEVÉE ou CRITIQUE doivent être supprimées ou reconstruites. Si une image déployée développe une vulnérabilité, elle doit être remplacée dès que possible.

Il est essentiel de savoir où les images présentant des vulnérabilités ont été déployées pour garantir la sécurité de votre environnement. Bien que vous puissiez créer vous-même une solution de suivi d'images, il existe déjà plusieurs offres commerciales qui offrent cette fonctionnalité et d'autres fonctionnalités avancées prêtes à l'emploi, notamment :

- Grype
- Palo Alto - Prisma Cloud (twistclip)
- Aqua
- Kubei
- Anecdotes
- Snyk

Un webhook de validation Kubernetes pourrait également être utilisé pour vérifier que les images sont exemptes de vulnérabilités critiques. Les webhooks de validation sont invoqués avant l'API Kubernetes. Ils sont généralement utilisés pour rejeter les demandes non conformes aux critères de validation définis dans le webhook. Voici un exemple de webhook sans serveur qui appelle l'API ECR Findings describeImageScan pour déterminer si un pod extrait une image présentant des vulnérabilités critiques. Si des vulnérabilités sont détectées, le pod est rejeté et un message contenant la liste des CVEs est renvoyé sous forme d'événement.

Utiliser des attestations pour valider l'intégrité des artefacts

Une attestation est une « déclaration » signée cryptographiquement qui affirme que quelque chose (un « prédicat », par exemple un pipeline, le SBOM ou le rapport d'analyse des vulnérabilités) est vrai sur un autre point : un « sujet », c'est-à-dire l'image du conteneur.

Les attestations aident les utilisateurs à valider qu'un artefact provient d'une source fiable de la chaîne d'approvisionnement logicielle. Par exemple, nous pouvons utiliser une image de conteneur sans connaître tous les composants logiciels ou toutes les dépendances inclus dans cette image.

Cependant, si nous faisons confiance à ce que dit le producteur de l'image du conteneur à propos du logiciel présent, nous pouvons utiliser l'attestation du producteur pour nous fier à cet artefact. Cela signifie que nous pouvons utiliser l'artefact en toute sécurité dans notre flux de travail au lieu d'avoir effectué l'analyse nous-mêmes.

- Les attestations peuvent être créées à l'aide d'AWS Signer ou de Sigstore cosign.
- Les contrôleurs d'admission Kubernetes tels que Kyverno peuvent être utilisés pour vérifier les attestations.
- Consultez cet atelier pour en savoir plus sur les meilleures pratiques de gestion de la chaîne d'approvisionnement logicielle sur AWS à l'aide d'outils open source, avec des sujets tels que la création et le rattachement d'attestations à une image de conteneur.

Création de politiques IAM pour les référentiels ECR

De nos jours, il n'est pas rare qu'une organisation dispose de plusieurs équipes de développement travaillant indépendamment au sein d'un compte AWS partagé. Si ces équipes n'ont pas besoin de partager des ressources, vous pouvez créer un ensemble de politiques IAM qui limitent l'accès aux référentiels avec lesquels chaque équipe peut interagir. Un bon moyen de l'implémenter est d'utiliser les espaces de noms ECR. Les espaces de noms permettent de regrouper des référentiels similaires. Par exemple, tous les registres de l'équipe A peuvent être préfacés avec le préfixe team-a/ tandis que ceux de l'équipe B peuvent utiliser le préfixe team-b/. La politique visant à restreindre l'accès peut ressembler à ce qui suit :

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowPushPull",
      "Effect": "Allow",
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability",
        "ecr:PutImage",
        "ecr:InitiateLayerUpload",
        "ecr:UploadLayerPart",
        "ecr:CompleteLayerUpload"
      ],
      "Resource": [
```

```
        "arn:aws:ecr:us-east-1:123456789012:repository/team-a/*"
      ]
    }
  ]
}
```

Envisagez d'utiliser des points de terminaison privés ECR

L'API ECR possède un point de terminaison public. Par conséquent, les registres ECR sont accessibles depuis Internet tant que la demande a été authentifiée et autorisée par IAM. Pour ceux qui doivent opérer dans un environnement sandbox où le VPC du cluster ne dispose pas d'une passerelle Internet (IGW), vous pouvez configurer un point de terminaison privé pour l'ECR. La création d'un point de terminaison privé vous permet d'accéder de manière privée à l'API ECR via une adresse IP privée au lieu de router le trafic sur Internet. Pour plus d'informations sur ce sujet, consultez la section Points de terminaison VPC de l'interface Amazon ECR.

Mettre en œuvre des politiques relatives aux terminaux pour l'ECR

La politique de point de terminaison par défaut pour autorise l'accès à tous les référentiels ECR d'une région. Cela peut permettre attacker/insider à an d'exfiltrer des données en les empaquetant sous forme d'image de conteneur et en les transférant vers un registre d'un autre compte AWS. Pour atténuer ce risque, il faut créer une politique de point de terminaison qui limite l'accès des API aux référentiels ECR. Par exemple, la politique suivante permet à tous les principes AWS de votre compte d'effectuer toutes les actions sur vos référentiels ECR et uniquement sur vos référentiels ECR :

```
{
  "Statement": [
    {
      "Sid": "LimitECRAccess",
      "Principal": "*",
      "Action": "*",
      "Effect": "Allow",
      "Resource": "arn:aws:ecr:<region>:<account_id>:repository/*"
    }
  ]
}
```

Vous pouvez encore améliorer cela en définissant une condition qui utilise le nouvel `PrincipalOrgID` attribut qui empêchera la diffusion pushing/pulling d'images selon un principe

IAM ne faisant pas partie de votre organisation AWS. Voir aws : PrincipalOrg ID pour plus de détails. Nous avons recommandé d'appliquer la même politique aux points de terminaison `com.amazonaws.<region>.ecr.dkr` et `com.amazonaws.<region>.ecr.api`. Comme EKS extrait des images pour kube-proxy, coredns et aws-node depuis ECR, vous devez ajouter l'ID de compte du registre, par exemple `602401143452.dkr.ecr.us-west-2.amazonaws.com/` à la liste des ressources dans la politique des terminaux ou modifier la politique pour autoriser les extractions depuis et limiter les push vers votre identifiant de compte. Le tableau ci-dessous indique le mappage entre les comptes AWS à partir desquels les images EKS sont vendues et la région du cluster.

Numéro du compte	Région
602401143452	Toutes les régions commerciales, à l'exception de celles répertoriées ci-dessous
—	—
800184023465	ap-east-1 - Asie-Pacifique (Hong Kong)
558608220178	me-south-1 - Moyen-Orient (Bahreïn)
918309763551	cn-north-1 - Chine (Pékin)
961992271922	cn-northwest-1 - Chine (Ningxia)

Pour plus d'informations sur l'utilisation des politiques de point de terminaison, consultez la section Utilisation des politiques de point de terminaison VPC pour contrôler l'accès à Amazon ECR.

Mettre en œuvre des politiques de cycle de vie pour l'ECR

Le guide de sécurité des conteneurs d'applications du NIST met en garde contre le risque d' « images périmées dans les registres », en notant qu'au fil du temps, out-of-date les anciennes images présentant des vulnérabilités doivent être supprimées pour éviter tout déploiement ou exposition accidentels. Chaque référentiel ECR peut avoir une politique de cycle de vie qui définit des règles relatives à l'expiration des images. La documentation officielle d'AWS décrit comment configurer des règles de test, les évaluer puis les appliquer. Les documents officiels contiennent plusieurs exemples de politique de cycle de vie qui montrent différentes manières de filtrer les images d'un référentiel :

- Filtrage par âge ou nombre d'images

- Filtrage par images balisées ou non étiquetées
- Filtrage par balises d'image, selon plusieurs règles ou une seule règle

? ? ? + avertissement Si l'image d'une application de longue durée est purgée de l'ECR, cela peut provoquer des erreurs d'extraction d'image lorsque l'application est redéployée ou redimensionnée horizontalement. Lorsque vous utilisez des politiques de cycle de vie des images, veillez à mettre en place de bonnes CI/CD pratiques pour maintenir les déploiements et les images auxquelles ils font référence à jour et créez toujours des règles d'expiration [des images] qui tiennent compte de la fréquence à laquelle vous effectuez des publiations/déploiements.

Créez un ensemble d'images sélectionnées

Plutôt que de permettre aux développeurs de créer leurs propres images, envisagez de créer un ensemble d'images validées pour les différentes piles d'applications de votre organisation. Ce faisant, les développeurs peuvent renoncer à apprendre à composer des Dockerfiles et se concentrer sur l'écriture de code. Lorsque les modifications sont fusionnées dans Master, un CI/CD pipeline peut automatiquement compiler l'actif, le stocker dans un référentiel d'artefacts et copier l'artefact dans l'image appropriée avant de le transférer vers un registre Docker tel que ECR. À tout le moins, vous devez créer un ensemble d'images de base à partir desquelles les développeurs pourront créer leurs propres Dockerfiles. Idéalement, vous devez éviter d'extraire des images de Dockerhub car 1/ vous ne savez pas toujours ce qu'elle contient et 2/ environ un cinquième des 1 000 meilleures images présentent des vulnérabilités. Une liste de ces images et de leurs vulnérabilités se trouve ici.

Ajoutez la directive USER à vos Dockerfiles pour l'exécuter en tant qu'utilisateur non root

Comme indiqué dans la section relative à la sécurité du pod, vous devez éviter d'exécuter le conteneur en tant que root. Bien que vous puissiez le configurer dans le cadre du PodSpec, c'est une bonne habitude d'utiliser la `USER` directive dans vos Dockerfiles. La `USER` directive définit l'UID à utiliser lors de l'exécution `RUNENTRYPOINT`, ou `CMD` l'instruction qui apparaît après la directive USER.

Lint vos Dockerfiles

Le linting peut être utilisé pour vérifier que vos Dockerfiles respectent un ensemble de directives prédéfinies, par exemple l'inclusion de la `USER` directive ou l'exigence selon laquelle toutes les images doivent être étiquetées. dockerfile_lint est un projet open source RedHat qui vérifie les meilleures pratiques courantes et inclut un moteur de règles que vous pouvez utiliser pour créer vos

propres règles pour le linting de Dockerfiles. Il peut être intégré à un pipeline CI, dans la mesure où les builds contenant des Dockerfiles qui enfreignent une règle échoueront automatiquement.

Créez des images à partir de zéro

La réduction de la surface d'attaque des images de vos conteneurs doit être l'objectif principal lors de la création d'images. L'idéal est de créer des images minimales dépourvues de fichiers binaires pouvant être utilisés pour exploiter les vulnérabilités. Heureusement, Docker dispose d'un mécanisme permettant de créer des images à partir de `scratch`. Avec des langages tels que Go, vous pouvez créer un binaire lié statique et le référencer dans votre Dockerfile comme dans cet exemple :

```
############################
# STEP 1 build executable binary
############################
FROM golang:alpine AS builder# Install git.
# Git is required for fetching the dependencies.
RUN apk update && apk add --no-cache gitWORKDIR $GOPATH/src/mypackage/myapp/COPY . . #
 Fetch dependencies.
# Using go get.
RUN go get -d -v# Build the binary.
RUN go build -o /go/bin/hello

############################
# STEP 2 build a small image
############################
FROM scratch# Copy our static executable.
COPY --from=builder /go/bin/hello /go/bin/hello# Run the hello binary.
ENTRYPOINT ["/go/bin/hello"]
```

Cela crée une image de conteneur composée de votre application et de rien d'autre, ce qui la rend extrêmement sécurisée.

Utiliser des balises immuables avec ECR

Les balises immuables vous obligent à mettre à jour la balise d'image à chaque transfert vers le référentiel d'images. Cela peut empêcher un attaquant de remplacer une image par une version malveillante sans modifier les balises de l'image. De plus, il vous permet d'identifier facilement et de manière unique une image.

Signez vos images SBOMs, vos cycles de pipeline et vos rapports de vulnérabilité

Lorsque Docker a été introduit pour la première fois, il n'existait aucun modèle cryptographique permettant de vérifier les images des conteneurs. Avec la version v2, Docker a ajouté des résumés au manifeste de l'image. Cela a permis de hacher la configuration d'une image et d'utiliser le hachage pour générer un identifiant pour l'image. Lorsque la signature d'image est activée, le moteur Docker vérifie la signature du manifeste, garantissant ainsi que le contenu a été produit à partir d'une source fiable et qu'aucune altération n'a eu lieu. Après le téléchargement de chaque couche, le moteur vérifie le condensé de la couche, en s'assurant que le contenu correspond au contenu spécifié dans le manifeste. La signature d'image vous permet de créer efficacement une chaîne d'approvisionnement sécurisée, grâce à la vérification des signatures numériques associées à l'image.

Nous pouvons utiliser AWS Signer ou Sigstore Cosign pour signer des images de conteneurs, créer des attestations, des rapports d'analyse des SBOMs vulnérabilités et des rapports d'exécution du pipeline. Ces attestations garantissent la fiabilité et l'intégrité de l'image, qu'elle est effectivement créée par le pipeline sécurisé sans aucune interférence ni altération, et qu'elle ne contient que les composants logiciels documentés (dans le SBOM) vérifiés et approuvés par l'éditeur de l'image. Ces attestations peuvent être jointes à l'image du conteneur et transmises au référentiel.

Dans la section suivante, nous verrons comment utiliser les artefacts attestés pour les audits et les vérifications du contrôleur des admissions.

Vérification de l'intégrité de l'image à l'aide du contrôleur d'admission Kubernetes

Nous pouvons vérifier les signatures d'image et les artefacts attestés de manière automatisée avant de déployer l'image sur le cluster Kubernetes cible à l'aide du contrôleur d'admission dynamique et n'admettre les déploiements que lorsque les métadonnées de sécurité des artefacts sont conformes aux politiques du contrôleur d'admission.

Par exemple, nous pouvons rédiger une politique qui vérifie cryptographiquement la signature d'une image, d'un SBOM attesté, d'un rapport d'exécution du pipeline ou d'un rapport de scan CVE attesté. Nous pouvons écrire des conditions dans la politique pour vérifier les données du rapport, par exemple, un scan CVE ne doit pas comporter de critiqueCVEs. Le déploiement n'est autorisé que pour les images qui répondent à ces conditions et tous les autres déploiements seront rejetés par le contrôleur des admissions.

Voici des exemples de contrôleur d'admission :

- Kyverno

- Contrôleur d'accès OPA
- Portier
- Ratifier
- Kritis
- Tutoriel Grafeas
- Bon cadeau

Mettez à jour les packages dans les images de vos conteneurs

Vous devez inclure RUN `apt-get update && apt-get upgrade` dans vos Dockerfiles pour mettre à jour les packages de vos images. Bien que la mise à niveau nécessite que vous l'exécutiez en tant que root, cela se produit pendant la phase de création de l'image. L'application n'a pas besoin de s'exécuter en tant que root. Vous pouvez installer les mises à jour, puis passer à un autre utilisateur avec la directive USER. Si votre image de base est exécutée en tant qu'utilisateur non root, passez en mode root et inversement ; ne vous fiez pas uniquement aux responsables de l'image de base pour installer les dernières mises à jour de sécurité.

Exécutez `apt-get clean` pour supprimer les fichiers du programme d'installation de`/var/cache/apt/archives/`. Vous pouvez également exécuter `rm -rf /var/lib/apt/lists/*` après l'installation des packages. Cela supprime les fichiers d'index ou les listes de packages disponibles pour l'installation. Sachez que ces commandes peuvent être différentes pour chaque gestionnaire de packages. Par exemple :

```
RUN apt-get update && apt-get install -y \
    curl \
    git \
    libsqlite3-dev \
    && apt-get clean && rm -rf /var/lib/apt/lists/*
```

Outils et ressources

- Atelier d'immersion sur la sécurité Amazon EKS - Image Security
- docker-slim Créez des images minimales sécurisées
- dockle Vérifie que votre Dockerfile est conforme aux meilleures pratiques en matière de création d'images sécurisées

- dockerfile-lint Linter basé sur des règles pour Dockerfiles
- hadolint Un linter de fichiers Docker intelligent
- Gatekeeper et OPA Un contrôleur d'admission basé sur des politiques
- Kyverno Un moteur de politiques natif de Kubernetes
- in-toto Permet à l'utilisateur de vérifier si une étape de la chaîne d'approvisionnement était censée être réalisée et si l'étape a été réalisée par le bon acteur
- Notaire Un projet pour signer des images de conteneurs
- Notaire v2
- Grafeas Une API ouverte de métadonnées d'artefacts pour auditer et gérer votre chaîne d'approvisionnement logicielle
- NeuVector de SUSE Open Source, plate-forme de sécurité des conteneurs Zero-Trust, permet d'analyser les conteneurs, les images et les registres pour détecter les vulnérabilités, les secrets et la conformité.

Stratégie multi-comptes

AWS recommande d'utiliser une stratégie multi-comptes et d'utiliser les organisations AWS pour vous aider à isoler et à gérer vos applications et données professionnelles. L'utilisation d'une stratégie multi-comptes présente de nombreux avantages :

- Augmentation des quotas de service d'API AWS. Des quotas sont appliqués aux comptes AWS, et l'utilisation de plusieurs comptes pour vos charges de travail augmente le quota global disponible pour vos charges de travail.
- Politiques de gestion des identités et des accès (IAM) simplifiées. Le fait d'accorder aux charges de travail et aux opérateurs qui les prennent en charge l'accès uniquement à leurs propres comptes AWS permet de passer moins de temps à élaborer des politiques IAM précises afin de respecter le principe du moindre privilège.
- Isolation améliorée des ressources AWS. De par leur conception, toutes les ressources fournies au sein d'un compte sont logiquement isolées des ressources fournies dans d'autres comptes. Cette limite d'isolation vous permet de limiter les risques liés à un problème lié à une application, à une mauvaise configuration ou à des actions malveillantes. Si un problème survient au sein d'un compte, les répercussions sur les charges de travail contenues dans d'autres comptes peuvent être réduites ou éliminées.
- Autres avantages, comme décrit dans le livre blanc sur la stratégie multi-comptes AWS

Les sections suivantes expliquent comment mettre en œuvre une stratégie multi-comptes pour vos charges de travail EKS en utilisant une approche de cluster EKS centralisée ou décentralisée.

Planification d'une stratégie de comptes à charges de travail multiples pour les clusters à locataires multiples

Dans une stratégie AWS multi-comptes, les ressources appartenant à une charge de travail donnée, telles que les compartiments S3, les ElastiCache clusters et les tables DynamoDB, sont toutes créées dans un compte AWS qui contient toutes les ressources pour cette charge de travail. Ils sont appelés comptes de charge de travail, et le cluster EKS est déployé dans un compte appelé compte de cluster. Les comptes groupés seront explorés dans la section suivante. Le déploiement de ressources dans un compte de charge de travail dédié est similaire au déploiement de ressources Kubernetes dans un espace de noms dédié.

Les comptes de charge de travail peuvent ensuite être ventilés en fonction du cycle de développement du logiciel ou d'autres exigences, le cas échéant. Par exemple, une charge de travail donnée peut avoir un compte de production, un compte de développement ou des comptes permettant d'héberger des instances de cette charge de travail dans une région spécifique. Plus d'informations sont disponibles dans ce livre blanc AWS.

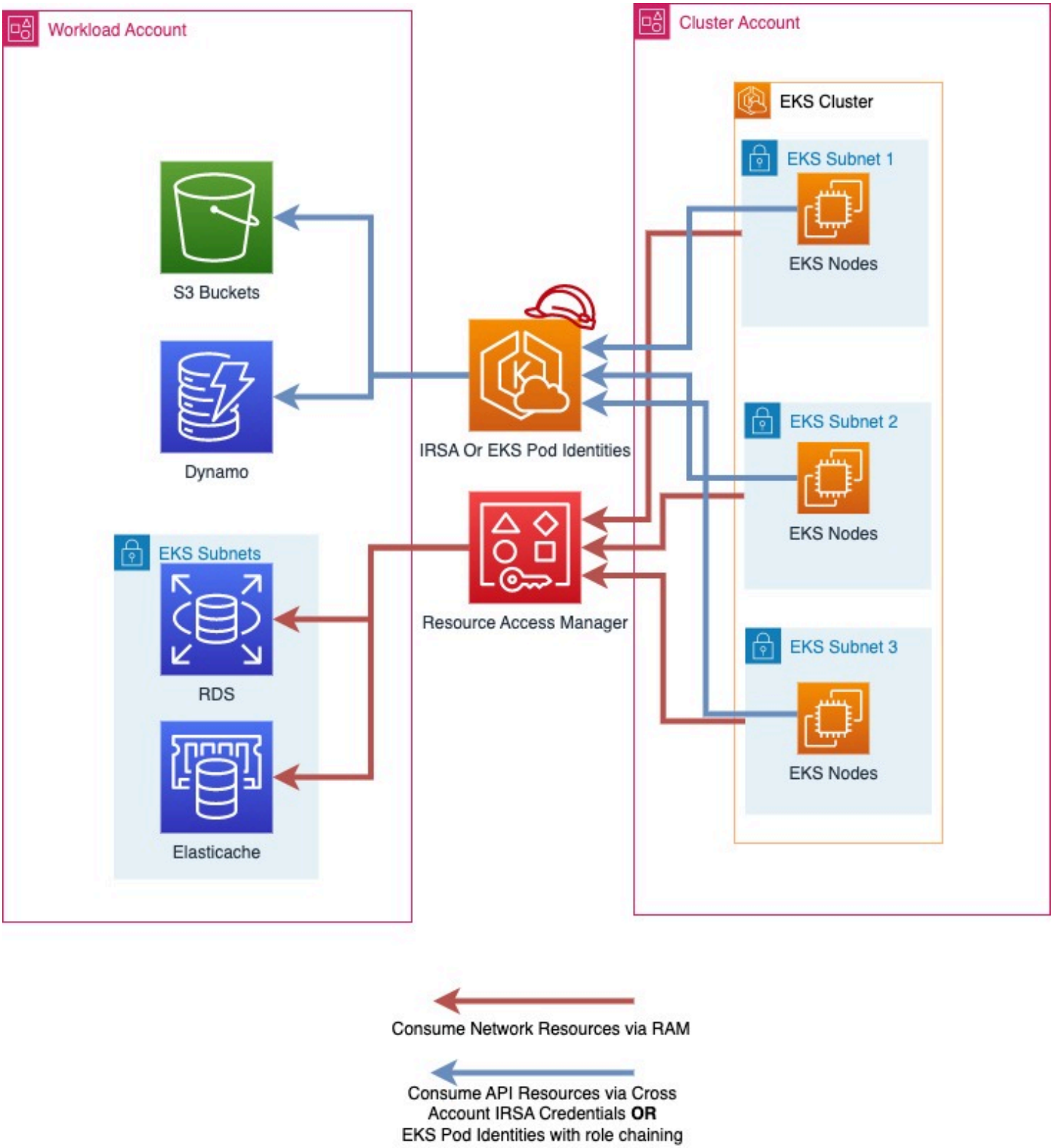
Vous pouvez adopter les approches suivantes lors de la mise en œuvre de la stratégie EKS Multi account :

Cluster EKS centralisé

Dans cette approche, votre cluster EKS sera déployé dans un seul compte AWS appelé`Cluster Account`. En utilisant les rôles IAM pour les comptes de service (IRSA) ou EKS Pod Identities pour fournir des informations d'identification AWS temporaires et AWS Resource Access Manager (RAM) pour simplifier l'accès au réseau, vous pouvez adopter une stratégie multi-comptes pour votre cluster EKS multi-tenant. Le compte de cluster contiendra le VPC, les sous-réseaux, le cluster EKS, les ressources de calcul EC2/FarGate (nœuds de travail) et toutes les configurations réseau supplémentaires nécessaires pour exécuter votre cluster EKS.

Dans une stratégie de comptes à charges de travail multiples pour un cluster à locataires multiples, les comptes AWS s'alignent généralement sur les espaces de noms Kubernetes afin d'isoler les groupes de ressources. Les meilleures pratiques en matière d'isolation des locataires au sein d'un cluster EKS doivent toujours être suivies lors de la mise en œuvre d'une stratégie multi-comptes pour les clusters EKS multi-locataires.

Il est possible d'`Cluster Accounts`en avoir plusieurs au sein de votre organisation AWS, et il est recommandé d'en avoir plusieurs `Cluster Accounts` qui répondent à vos besoins en matière de cycle de développement logiciel. Pour les charges de travail fonctionnant à très grande échelle, vous pouvez en avoir besoin de plusieurs `Cluster Accounts` pour garantir que les quotas de service Kubernetes et AWS sont suffisants pour toutes vos charges de travail.



|Dans le schéma ci-dessus, la RAM AWS est utilisée pour partager des sous-réseaux d'un compte de cluster vers un compte de charge de travail. Les charges de travail exécutées dans des pods EKS

utilisent ensuite les identités IRSA ou EKS Pod et le chaînage des rôles pour assumer un rôle dans leur compte de charge de travail et accéder à leurs ressources AWS.

Mise en œuvre d'une stratégie de comptes multi-charges de travail pour un cluster multi-locataires

Partage de sous-réseaux avec AWS Resource Access Manager

AWS Resource Access Manager (RAM) vous permet de partager des ressources entre des comptes AWS.

Si la RAM est activée pour votre organisation AWS, vous pouvez partager les sous-réseaux VPC entre le compte de cluster et vos comptes de charge de travail. Cela permettra aux ressources AWS détenues par vos comptes de charge de travail, telles que les bases de données Amazon ElastiCache Clusters ou Amazon Relational Database Service (RDS) d'être déployées dans le même VPC que votre cluster EKS, et d'être consommables par les charges de travail exécutées sur votre cluster EKS.

Pour partager une ressource via la RAM, ouvrez la RAM dans la console AWS du compte du cluster et sélectionnez « Resource Shares » et « Create Resource Share ». Nommez votre partage de ressources et sélectionnez les sous-réseaux que vous souhaitez partager. Sélectionnez à nouveau Suivant et entrez le compte à 12 chiffres IDs pour les comptes de charge de travail avec lesquels vous souhaitez partager les sous-réseaux, sélectionnez à nouveau Suivant, puis cliquez sur Créer un partage de ressources pour terminer. Après cette étape, le compte de charge de travail peut déployer des ressources dans ces sous-réseaux.

Les partages de RAM peuvent également être créés par programmation ou avec une infrastructure sous forme de code.

Choisir entre EKS Pod Identities et IRSA

À l'occasion de re:Invent 2023, AWS a lancé EKS Pod Identities, un moyen plus simple de fournir des informations d'identification AWS temporaires à vos pods sur EKS. Les identités IRSA et EKS Pod sont des méthodes valides pour fournir des informations d'identification AWS temporaires à vos pods EKS et elles continueront d'être prises en charge. Vous devez déterminer quelle méthode de livraison répond le mieux à vos besoins.

Lorsque vous travaillez avec un cluster EKS et plusieurs comptes AWS, IRSA peut directement assumer des rôles dans les comptes AWS autres que celui dans lequel le cluster EKS est

directement hébergé, tandis que les identités EKS Pod nécessitent que vous configuriez le chaînage des rôles. Reportez-vous à la documentation EKS pour une comparaison approfondie.

Accès aux ressources de l'API AWS avec des rôles IAM pour les comptes de service

IAM Roles for Service Accounts (IRSA) vous permet de fournir des informations d'identification AWS temporaires à vos charges de travail exécutées sur EKS. L'IRSA peut être utilisé pour obtenir des informations d'identification temporaires pour les rôles IAM dans les comptes de charge de travail à partir du compte de cluster. Cela permet à vos charges de travail exécutées sur vos clusters EKS du compte de cluster de consommer facilement les ressources de l'API AWS, telles que les compartiments S3 hébergés dans le compte de charge de travail, et d'utiliser l'authentification IAM pour des ressources telles que les bases de données Amazon RDS ou Amazon EFS. FileSystems

Les ressources de l'API AWS et les autres ressources qui utilisent l'authentification IAM dans un compte de charge de travail ne sont accessibles qu'à l'aide des informations d'identification des rôles IAM dans ce même compte de charge de travail, sauf lorsque l'accès entre comptes est possible et a été explicitement activé.

Activation de l'IRSA pour l'accès entre comptes

Pour permettre à IRSA pour les charges de travail de votre compte de cluster d'accéder aux ressources de vos comptes de charge de travail, vous devez d'abord créer un fournisseur d'identité IAM OIDC dans votre compte de charge de travail. Cela peut être fait avec la même procédure que pour configurer l'IRSA, sauf que le fournisseur d'identité sera créé dans le compte de charge de travail.

Ensuite, lorsque vous configurez IRSA pour vos charges de travail sur EKS, vous pouvez suivre les mêmes étapes que dans la documentation, mais utiliser l'identifiant de compte à 12 chiffres du compte de charge de travail, comme indiqué dans la section « Exemple de création d'un fournisseur d'identité à partir du cluster d'un autre compte ».

Une fois cette configuration terminée, votre application exécutée dans EKS pourra utiliser directement son compte de service pour assumer un rôle dans le compte de charge de travail et utiliser les ressources qu'il contient.

Accès aux ressources de l'API AWS avec EKS Pod Identities

EKS Pod Identities est une nouvelle façon de fournir des informations d'identification AWS à vos charges de travail exécutées sur EKS. Les identités de pods EKS simplifient la configuration des

ressources AWS, car vous n'avez plus besoin de gérer les configurations OIDC pour fournir des informations d'identification AWS à vos pods sur EKS.

Activation d'EKS Pod Identities pour l'accès entre comptes

Contrairement à IRSA, EKS Pod Identities ne peut être utilisée que pour accorder directement l'accès à un rôle dans le même compte que le cluster EKS. Pour accéder à un rôle dans un autre compte AWS, les pods qui utilisent EKS Pod Identities doivent effectuer un chaînage des rôles.

Le chaînage des rôles peut être configuré dans le profil d'une application avec son fichier de configuration aws à l'aide du fournisseur d'informations d'identification de processus disponible dans divers AWS SDKs. `credential_process`peut être utilisé comme source d'informations d'identification lors de la configuration d'un profil, par exemple :

```
# Content of the AWS Config file
[profile account_b_role]
source_profile = account_a_role
role_arn = arn:aws:iam::444455556666:role/account-b-role

[profile account_a_role]
credential_process = /eks-credential-processrole.sh
```

La source du script appelé par credential_process :

```
#!/bin/bash
# Content of the eks-credential-processrole.sh
# This will retreive the credential from the pod identities agent,
# and return it to the AWS SDK when referenced in a profile
curl -H "Authorization: $(cat $AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE)"
 $AWS_CONTAINER_CREDENTIALS_FULL_URI | jq -c '{AccessKeyId: .AccessKeyId,
 SecretAccessKey: .SecretAccessKey, SessionToken: .Token, Expiration: .Expiration,
 Version: 1}'
```

Vous pouvez créer un fichier de configuration aws comme indiqué ci-dessus avec les rôles de compte A et B et spécifier les variables AWS_CONFIG_FILE et AWS_PROFILE env dans les spécifications de votre pod. Le webhook d'identité EKS Pod ne remplace pas si les variables d'environnement existent déjà dans les spécifications du pod.

```
# Snippet of the PodSpec
containers:
```

```
  - name: container-name
    image: container-image:version
    env:
    - name: AWS_CONFIG_FILE
      value: path-to-customer-provided-aws-config-file
    - name: AWS_PROFILE
      value: account_b_role
```

Lorsque vous configurez des politiques d'approbation des rôles pour le chaînage des rôles avec les identités des pods EKS, vous pouvez référencer les attributs spécifiques d'EKS sous forme de balises de session et utiliser le contrôle d'accès basé sur les attributs (ABAC) pour limiter l'accès à vos rôles IAM à des sessions d'identité EKS Pod spécifiques, telles que le compte de service Kubernetes auquel appartient un pod.

Notez que certains de ces attributs peuvent ne pas être uniques pour tous. Par exemple, deux clusters EKS peuvent avoir des espaces de noms identiques, et un cluster peut avoir des comptes de service portant le même nom dans tous les espaces de noms. Ainsi, lorsque vous accordez l'accès via EKS Pod Identities et ABAC, il est recommandé de toujours prendre en compte l'ARN et l'espace de noms du cluster lors de l'octroi de l'accès à un compte de service.

Identités ABAC et EKS Pod pour un accès entre comptes

Lorsque vous utilisez EKS Pod Identities pour assumer des rôles (chaîne de rôles) dans d'autres comptes dans le cadre d'une stratégie multi-comptes, vous avez la possibilité d'attribuer un rôle IAM unique à chaque compte de service qui doit accéder à un autre compte, ou d'utiliser un rôle IAM commun à plusieurs comptes de service et d'utiliser ABAC pour contrôler les comptes auxquels il peut accéder.

Pour utiliser ABAC afin de contrôler quels comptes de service peuvent assumer un rôle dans un autre compte grâce au chaînage des rôles, vous créez une déclaration de politique de confiance en matière de rôles qui autorise uniquement un rôle à être assumé par une session de rôle lorsque les valeurs attendues sont présentes. La politique de confiance des rôles suivante ne permet à un rôle du compte de cluster EKS (ID de compte 111122223333) d'assumer un rôle que si les `kubernetes-service-account` `kubernetes-namespace` balises `eks-cluster-arn` et ont toutes la valeur attendue.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

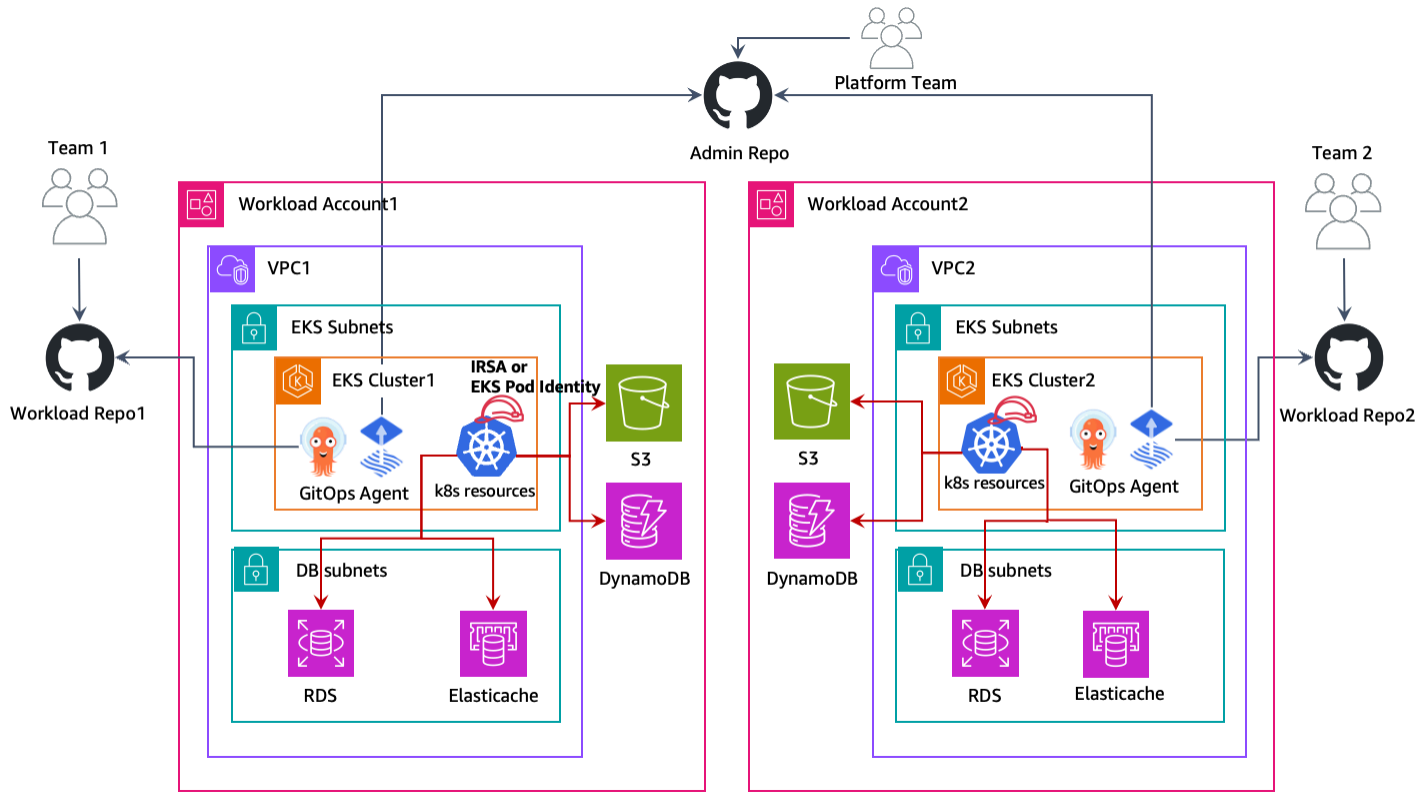
```
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalTag/kubernetes-service-account":
 "PayrollApplication",
                    "aws:PrincipalTag/eks-cluster-arn": "arn:aws:eks:us-
east-1:111122223333:cluster/ProductionCluster",
                    "aws:PrincipalTag/kubernetes-namespace": "PayrollNamespace"
                }
            }
        }
    ]
}
```

Lorsque vous utilisez cette stratégie, il est recommandé de s'assurer que le rôle IAM commun ne dispose que d'`sts:AssumeRole`autorisations et d'aucun autre accès AWS.

Lorsque vous utilisez ABAC, il est important de contrôler qui est autorisé à attribuer des rôles et des utilisateurs IAM uniquement à ceux qui en ont strictement besoin. Une personne capable de baliser un rôle ou un utilisateur IAM serait en mesure de définir des balises roles/users identiques à celles définies par EKS Pod Identities et pourrait être en mesure d'augmenter ses privilèges. Vous pouvez restreindre les personnes autorisées à définir les balises `kubernetes-` et les `eks-` balises sur le rôle IAM et les utilisateurs à l'aide de la politique IAM ou de la politique de contrôle des services (SCP).

Clusters EKS décentralisés

Dans cette approche, les clusters EKS sont déployés sur les comptes AWS de charge de travail respectifs et coexistent avec d'autres ressources AWS telles que les compartiments Amazon S3 VPCs, les tables Amazon DynamoDB, etc. Chaque compte de charge de travail est indépendant, autonome et géré par le cluster Unit/Application teams. This model allows the creation of reusuable blueprints for various cluster capabilities — AI/ML commercial, le traitement par lots, l'usage général, etc., et vendent les clusters en fonction des besoins de l'équipe d'application. Les équipes chargées des applications et des plateformes opèrent à partir de leurs GitOpsréférentiels respectifs pour gérer les déploiements sur les clusters de charge de travail.



Dans le schéma ci-dessus, les clusters Amazon EKS et les autres ressources AWS sont déployés sur les comptes de charge de travail respectifs. Les charges de travail exécutées dans les pods EKS utilisent ensuite les identités IRSA ou EKS Pod pour accéder à leurs ressources AWS.

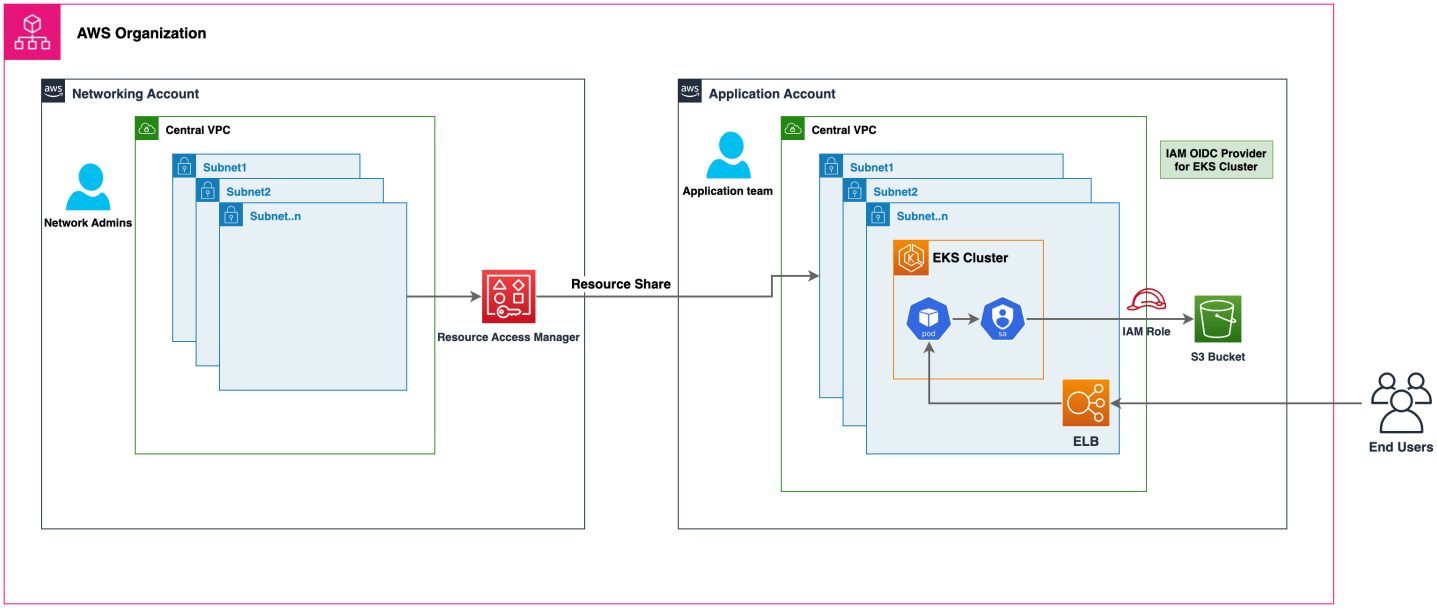
GitOps est un moyen de gérer le déploiement d'applications et d'infrastructures afin que l'ensemble du système soit décrit de manière déclarative dans un référentiel Git. Il s'agit d'un modèle opérationnel qui vous permet de gérer l'état de plusieurs clusters Kubernetes en utilisant les meilleures pratiques en matière de contrôle de version, d'artefacts immuables et d'automatisation. Dans ce modèle multi-clusters, chaque cluster de charge de travail est amorcé avec plusieurs dépôts Git, ce qui permet à chaque équipe (application, plateforme, sécurité, etc.) de déployer ses modifications respectives sur le cluster.

Vous utiliseriez les rôles IAM pour les comptes de service (IRSA) ou les identités EKS Pod dans chaque compte afin de permettre à vos charges de travail EKS d'obtenir des informations d'identification AWS temporaires afin d'accéder en toute sécurité à d'autres ressources AWS. Les rôles IAM sont créés dans les comptes AWS de charge de travail respectifs et les mappent aux comptes de service k8s pour fournir un accès IAM temporaire. Ainsi, aucun accès entre comptes n'est requis dans cette approche. Suivez la documentation sur les rôles IAM pour les comptes de

service pour savoir comment configurer chaque charge de travail pour IRSA, et la documentation EKS Pod Identities sur la façon de configurer les identités des pods EKS dans chaque compte.

Mise en réseau centralisée

Vous pouvez également utiliser la RAM AWS pour partager les sous-réseaux VPC avec des comptes de charge de travail et y lancer des clusters Amazon EKS et d'autres ressources AWS. Cela permet une gestion/administration centralisée du réseau, une connectivité réseau simplifiée et des clusters EKS décentralisés. Consultez ce blog AWS pour une présentation détaillée et des considérations relatives à cette approche.



Dans le schéma ci-dessus, la RAM AWS est utilisée pour partager des sous-réseaux d'un compte réseau central vers un compte de charge de travail. Le cluster EKS et les autres ressources AWS sont ensuite lancés dans ces sous-réseaux dans les comptes de charge de travail respectifs. Les pods EKS utilisent les identités IRSA ou EKS Pod pour accéder à leurs ressources AWS.

Clusters EKS centralisés ou décentralisés

La décision d'utiliser une solution centralisée ou décentralisée dépendra de vos besoins. Ce tableau montre les principales différences entre chaque stratégie.

#	Cluster EKS centralisé	Clusters EKS décentralisés
Gestion des clusters :	Il est plus facile de gérer un seul cluster EKS que	Une automatisation efficace de la gestion des clusters est

#	Cluster EKS centralisé	Clusters EKS décentralisés
	d'administrer plusieurs clusters	nécessaire pour réduire la charge opérationnelle liée à la gestion de plusieurs clusters EKS
Rentabilité :	Permet la réutilisation des ressources du cluster et du réseau EKS, ce qui favorise la rentabilité	Nécessite des configura tions réseau et de clusters par charge de travail, ce qui nécessite des ressources supplémentaires
Résilience :	Plusieurs charges de travail sur le cluster centralisé peuvent être affectées si un cluster est altéré	Si un cluster est endommagé , les dommages sont limités aux seules charges de travail exécutées sur ce cluster. Toutes les autres charges de travail ne sont pas affectées
Isolation et sécurité :	L'isolation/la mutualisation souple est obtenue à l'aide de constructions natives de k8s telles que. `Namespaces` Les charges de travail peuvent partager les ressources sous-jacentes telles que le processeur, la mémoire, etc. Les ressources AWS sont isolées dans leurs propres comptes de charge de travail qui, par défaut, ne sont pas accessibles depuis d'autres comptes AWS.	Isolation renforcée des ressources informatiques, car les charges de travail s'exécutent dans des clusters et des nœuds individuels qui ne partagent aucune ressource. Les ressources AWS sont isolées dans leurs propres comptes de charge de travail qui, par défaut, ne sont pas accessibles depuis d'autres comptes AWS.

#	Cluster EKS centralisé	Clusters EKS décentralisés
Performances et évolutivité :	Lorsque les charges de travail augmentent à très grande échelle, vous pouvez rencontrer des quotas de service Kubernetes et AWS dans le compte du cluster. Vous pouvez déployer des comptes de cluster supplémentaires pour encore plus d'évolutivité	Au fur et à mesure que de nouveaux clusters VPCs sont présents, chaque charge de travail dispose d'un plus grand nombre de k8 disponibles et de quotas de service AWS
Réseautage :	Un seul VPC est utilisé par cluster, ce qui permet de simplifier la connectivité pour les applications de ce cluster	Le routage doit être établi entre le cluster EKS décentral isé VPCs
Gestion des accès Kubernete s :	Nécessité de conserver de nombreux rôles et utilisate urs différents dans le cluster afin de fournir un accès à toutes les équipes chargées de la charge de travail et de garantir que les ressources Kubernetes sont correctement séparées	Gestion des accès simplifiée car chaque cluster est dédié à une charge de travail/à une équipe

#	Cluster EKS centralisé	Clusters EKS décentralisés
Gestion des accès AWS :	Les ressources AWS sont déployées sur leur propre compte, auquel ils ne sont accessibles par défaut qu'avec les rôles IAM dans le compte de charge de travail. Les rôles IAM dans les comptes de charge de travail sont supposés être intercomp tes avec IRSA ou EKS Pod Identities.	Les ressources AWS sont déployées sur leur propre compte, auquel ils ne sont accessibles par défaut qu'avec les rôles IAM dans le compte de charge de travail. Les rôles IAM dans les comptes de charge de travail sont fournis directement aux pods dotés d'identités de pod IRSA ou EKS.

Gestion des accès aux clusters

Une gestion efficace des accès est essentielle pour garantir la sécurité et l'intégrité de vos clusters Amazon EKS. Ce guide explore les différentes options de gestion des accès EKS, en mettant l'accent sur l'utilisation d'AWS IAM Identity Center (anciennement AWS SSO). Nous comparerons différentes approches, discuterons de leurs compromis et soulignerons les limites et considérations connues.

Options de gestion des accès EKS

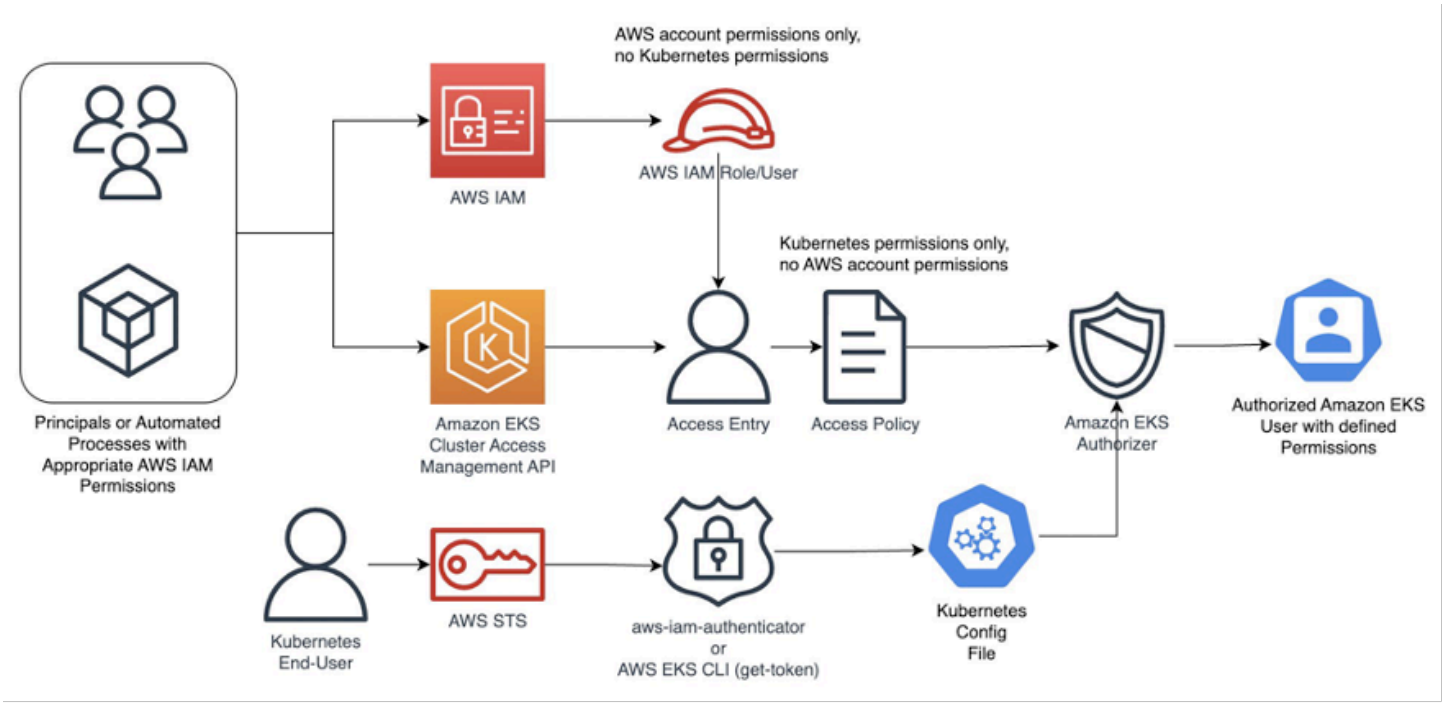
> **Note**
>
> ConfigMapla gestion des accès basée sur le cluster (aws-auth ConfigMap) est obsolète et remplacée par l'API de gestion des accès au cluster (CAM). Pour les nouveaux clusters EKS, implémentez l'API CAM pour gérer l'accès aux clusters. Pour les clusters existants utilisant aws-auth ConfigMap, migrez vers l'API CAM.

Option 1 : centre d'identité AWS IAM avec API de gestion des accès aux clusters (CAM)

- Gestion centralisée des utilisateurs et des autorisations
- Intégration avec les fournisseurs d'identité existants (par exemple Microsoft AD, Okta, PingId etc.)

- L'API CAM utilise des entrées d'accès pour relier les principaux AWS IAM (utilisateurs ou rôles) au cluster EKS. Ces entrées fonctionnent avec les identités gérées par IAM Identity Center, ce qui permet aux administrateurs de contrôler l'accès au cluster pour les utilisateurs et les groupes définis dans Identity Center.

Flux d'authentification du cluster EKS :



1. Les principaux (utilisateurs humains) ou les processus automatisés s'authentifient via AWS IAM en présentant les autorisations de compte AWS appropriées. Au cours de cette étape, ils sont mappés au principal AWS IAM approprié (rôle ou utilisateur).
2. Ensuite, une entrée d'accès EKS mappe ce principal IAM à un principal Kubernetes RBAC (utilisateur ou groupe) en définissant une politique d'accès appropriée, qui contient uniquement les autorisations Kubernetes.
3. Lorsqu'un utilisateur final de Kubernetes essaie d'accéder à un cluster, sa demande d'authentification est traitée par la CLI aws-iam-authenticator AWS EKS et validée par rapport au contexte du cluster dans le fichier kubeconfig.
4. Enfin, l'autorisateur EKS vérifie les autorisations associées à l'entrée d'accès de l'utilisateur authentifié et accorde ou refuse l'accès en conséquence.
   - L'API utilise des politiques d'accès spécifiques à Amazon EKS pour définir le niveau d'autorisation pour chaque entrée d'accès. Ces politiques peuvent être associées aux rôles et

aux autorisations définis dans IAM Identity Center, afin de garantir un contrôle d'accès cohérent entre les services AWS et les clusters EKS.

Avantages par rapport ConfigMap à la gestion des accès basée sur la base :

1. Réduction des risques de mauvaise configuration : la gestion directe basée sur l'API élimine les erreurs courantes associées à l'édition manuelle. ConfigMap Cela permet d'éviter les suppressions accidentelles ou les erreurs de syntaxe susceptibles d'empêcher les utilisateurs d'accéder au cluster.
2. Principe du moindre privilège amélioré : élimine le besoin d'une autorisation d'administrateur de cluster pour l'identité du créateur du cluster et permet une attribution des autorisations plus précise et plus appropriée. Vous pouvez choisir d'ajouter cette autorisation pour les cas d'utilisation de break-glass.
3. Modèle de sécurité amélioré : fournit une validation intégrée des entrées d'accès avant leur application. En outre, offre une intégration plus étroite avec AWS IAM pour l'authentification.
4. Opérations rationalisées : offre un moyen plus intuitif de gérer les autorisations grâce à des outils natifs d'AWS.

Bonnes pratiques :

1. Utilisez AWS Organizations pour gérer plusieurs comptes et appliquer des politiques de contrôle des services (SCPs).
2. Mettez en œuvre le principe du moindre privilège en créant des ensembles d'autorisations spécifiques pour différents rôles EKS (par exemple, administrateur, développeur, lecture seule).
3. Utilisez le contrôle d'accès basé sur les attributs (ABAC) pour attribuer dynamiquement des autorisations aux pods en fonction des attributs de l'utilisateur.
4. Auditez et révisez régulièrement les autorisations d'accès.

Considérations/limites :

1. ARNs Les rôles générés par Identity Center ont des suffixes aléatoires, ce qui les rend difficiles à utiliser dans des configurations statiques.
2. Support limité pour les autorisations détaillées au niveau des ressources Kubernetes. Une configuration supplémentaire est requise pour les rôles Kubernetes RBAC personnalisés.

Outre le RBAC natif de Kubernetes, envisagez d'utiliser Kyverno pour une gestion avancée des autorisations dans les clusters EKS.

Option 2 : AWS IAM Users/Roles mappé aux groupes Kubernetes

Avantages :

1. Contrôle précis des autorisations IAM.
2. Rôle prévisible et statique ARNs

Inconvénients :

1. Frais de gestion accrus pour les comptes utilisateurs
2. Absence de gestion centralisée des identités
3. Potentiel de prolifération des entités IAM

Bonnes pratiques :

1. Utilisez des rôles IAM plutôt que des utilisateurs IAM pour améliorer la sécurité et la gérabilité
2. Implémenter une convention de dénomination pour les rôles afin de garantir la cohérence et la facilité de gestion
3. Utilisez les conditions de la politique IAM pour restreindre l'accès en fonction de balises ou d'autres attributs.
4. Faites régulièrement pivoter les clés d'accès et vérifiez les autorisations.

Considérations/limites :

1. Problèmes d'évolutivité lors de la gestion d'un grand nombre d'utilisateurs ou de rôles
2. Aucune fonctionnalité d'authentification unique intégrée

Option 3 : fournisseurs OIDC

Avantages :

1. Intégration aux systèmes de gestion des identités existants

2. Réduction des frais de gestion pour les comptes utilisateurs

Inconvénients :

1. Complexité de configuration supplémentaire
2. Possibilité d'augmentation de la latence lors de l'authentification
3. Dépendance vis-à-vis d'un fournisseur d'identité externe

Bonnes pratiques :

1. Configurez soigneusement le fournisseur OIDC pour garantir une validation sécurisée des jetons.
2. Utilisez des jetons de courte durée et implémentez des mécanismes d'actualisation des jetons.
3. Auditez et mettez à jour régulièrement les configurations OIDC.

Consultez ce guide pour découvrir une implémentation de référence de l'intégration de fournisseurs d'authentification unique externes à Amazon EKS

Considérations/limites :

1. Intégration native limitée avec les services AWS par rapport à IAM.
2. L'URL de l'émetteur du fournisseur OIDC doit être accessible au public pour qu'EKS puisse découvrir les clés de signature.

AWS EKS Pod Identity et IRSA pour les charges de travail

Amazon EKS propose deux méthodes pour accorder des autorisations AWS IAM aux charges de travail qui s'exécutent dans des clusters Amazon EKS : les rôles IAM pour les comptes de service (IRSA) et les identités EKS Pod.

Bien que les identités IRSA et EKS Pod offrent les avantages de l'accès par le moindre privilège, de l'isolation des informations d'identification et de l'auditabilité, EKS Pod Identity est la méthode recommandée pour accorder des autorisations aux charges de travail.

Pour obtenir des conseils détaillés sur l'identité et les informations d'identification des modules EKS, reportez-vous à la section Identités et informations d'identification des meilleures pratiques en matière de sécurité.

Recommendation

Combinez IAM Identity Center avec l'API CAM

- Gestion simplifiée : en utilisant l'API de gestion des accès aux clusters conjointement avec IAM Identity Center, les administrateurs peuvent gérer l'accès au cluster EKS parallèlement aux autres services AWS, ce qui réduit le besoin de passer d'une interface à l'autre ou de procéder à des modifications ConfigMaps manuelles.
- Utilisez les entrées d'accès pour gérer les autorisations Kubernetes des principaux IAM depuis l'extérieur du cluster. Vous pouvez ajouter et gérer l'accès au cluster à l'aide de l'API EKS, de l'interface de ligne de commande AWS, d'AWS SDKs, d'AWS CloudFormation et de la console de gestion AWS. Cela signifie que vous pouvez gérer les utilisateurs avec les mêmes outils que ceux avec lesquels vous avez créé le cluster.
- Les autorisations Kubernetes granulaires peuvent être appliquées en mappant les utilisateurs ou les groupes Kubernetes aux principaux IAM associés aux identités SSO via des entrées d'accès et des politiques d'accès.
- Pour commencer, suivez Changer le mode d'authentification pour utiliser les entrées d'accès, puis Migration des entrées aws-auth existantes pour accéder aux ConfigMap entrées.

Bonnes pratiques en matière de mise à l'échelle automatique des clusters

Ce guide fournit des conseils sur le scalage automatique des clusters, notamment des conseils pour le mode automatique, Karpenter et Kubernetes Cluster Autoscaler.

Rubriques

- Mode automatique EKS
- Karpenter
- Cluster Autoscaler

Mode automatique EKS

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Le mode automatique Amazon EKS représente une évolution significative de la gestion de l'infrastructure Kubernetes, combinant une infrastructure de cluster sécurisée et évolutive avec des fonctionnalités Kubernetes intégrées gérées par AWS. Le service fournit des opérations de nœuds de travail entièrement gérées, éliminant ainsi la nécessité pour les clients de configurer des groupes ou AutoScaling des groupes de nœuds gérés.

La principale différence architecturale réside dans le fait que le mode automatique d'EKS utilise un système basé sur Karpenter qui provisionne automatiquement les instances EC2 en réponse aux demandes de pods. Ces instances s'exécutent sur Bottlerocket AMIs avec des modules complémentaires préinstallés tels que les pilotes EBS CSI, ce qui permet à l'infrastructure de réellement être gérée par AWS. Contrairement aux méthodes de mise à l'échelle traditionnelles :

- Le Cluster Autoscaler (CAS) traditionnel nécessite une gestion manuelle des groupes de nœuds et ne peut créer que des nœuds avec un seul type d'instance par groupe de nœuds
- Karpenter autogéré offre plus de flexibilité en utilisant l'API EC2 Fleet et peut fournir différents types d'instances, mais nécessite une gestion des clients

- Le mode automatique EKS gère automatiquement toutes les opérations de dimensionnement via NodePools des NodeClasses

Le nouveau système introduit plusieurs améliorations opérationnelles :

- Dimensionnement automatique piloté par pod sans configuration manuelle des groupes de nœuds
- Contrôleurs d'équilibrage de charge gérés intégrés qui créent automatiquement en ALB/NLB fonction des ressources d'entrée
- Fonctionnalités de sécurité intégrées avec identité Pod préconfigurée
- Durée d'exécution maximale du nœud de 21 jours avec remplacement automatique

Du point de vue des coûts, EKS Auto Mode maintient la tarification EC2 standard tout en ajoutant des frais de gestion uniquement pour les nœuds gérés en mode automatique. Il est important de noter que les clients peuvent toujours combiner des nœuds gérés en mode automatique avec des nœuds autogérés dans le même cluster.

AWS gère la plupart des aspects opérationnels, mais les clients restent responsables de la gestion des versions du cluster et peuvent effectuer des mises à niveau contrôlées qui déclenchent des mises à jour continues des nœuds de travail.

Raisons d'utiliser le mode automatique

Le mode automatique s'adresse aux utilisateurs qui souhaitent bénéficier des avantages de Kubernetes et d'EKS tout en minimisant la charge opérationnelle liée à Kubernetes, comme les mises à niveau, et les éléments installation/maintenance critiques de la plate-forme tels que l'auto-scaling, l'équilibrage de charge et le stockage. Le mode automatique permet à EKS de franchir une nouvelle étape dans la minimisation des charges lourdes indifférenciées associées à la maintenance de Kubernetes

FAQ

Quelle est la différence entre le mode automatique d'EKS et le mode Open Source Karpenter ?

Le mode automatique EKS est une vaste suite de fonctionnalités qui simplifient l'exécution de Kubernetes de niveau production. L'une de ces fonctionnalités concerne les avantages de l'auto-scaling de Karpenter, entièrement géré. Du point de vue des opérations, la seule différence

réside dans le mode automatique d'EKS : vous n'avez pas besoin de gérer le déploiement, le dimensionnement et la mise à niveau des modules Karpenter eux-mêmes. Toutes les autres opérations, comme gérées, NodePools fonctionnent NodeClasses de la même manière qu'avec l'open source Karpenter.

Puis-je exécuter des groupes de nœuds gérés parallèlement à des nœuds gérés en mode automatique ?

Oui, vous pouvez exécuter des nœuds statiques via des groupes de nœuds gérés parallèlement à vos nœuds de mise à l'échelle automatique fournis avec le mode automatique

Puis-je migrer un cluster du mode EKS standard vers le mode automatique EKS ?

Oui, les instructions pour activer le mode automatique EKS sur un cluster existant se trouvent dans la documentation AWS officielle

Points à noter : 1. Après avoir activé le mode automatique, vous devez désinstaller tous les composants que vous avez installés et qui sont désormais gérés par le mode automatique, comme Karpenter ou le AWS Load Balancer Controller 2. Vous devez vous assurer que les modules complémentaires que vous avez installés le sont up-to-date. Consultez la documentation.

Comment configurer NodePools en mode automatique EKS ?

Un nouveau cluster sera préconfiguré avec deux NodePools

à usage général

```yaml
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  generation: 1
  labels:
    app.kubernetes.io/managed-by: eks
  name: general-purpose
spec:
  disruption:
    budgets:
      - nodes: 10%
    consolidateAfter: 30s
    consolidationPolicy: WhenEmptyOrUnderutilized
  template:
    spec:
      expireAfter: 336h
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values:
            - on-demand
        - key: eks.amazonaws.com/instance-category
          operator: In
          values:
            - c
            - m
            - r
        - key: eks.amazonaws.com/instance-generation
          operator: Gt
          values:
            - '4'
        - key: kubernetes.io/arch
          operator: In
          values:
            - amd64
        - key: kubernetes.io/os
          operator: In
          values:
            - linux
      terminationGracePeriod: 24h0m0s
```

Cela NodePool indique à Karpenter de lancer des nœuds présentant les caractéristiques suivantes :

1. Type de capacité « à la demande »
2. Types d'instances C, M ou R
3. Génération d'instances de 4
4. Architecture AMD
5. SE Linux

Il définit également la logique de réduction en déclarant que seuls 10 % de tous les nœuds peuvent être perturbés à un moment donné et que la consolidation ne doit avoir lieu que lorsque les nœuds sont vides ou sous-utilisés.

system

```yaml
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  generation: 1
  labels:
    app.kubernetes.io/managed-by: eks
  name: system
spec:
  disruption:
    budgets:
      - nodes: 10%
    consolidateAfter: 30s
    consolidationPolicy: WhenEmptyOrUnderutilized
  template:
    spec:
      expireAfter: 336h
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values:
            - on-demand
        - key: eks.amazonaws.com/instance-category
          operator: In
          values:
            - c
            - m
            - r
        - key: eks.amazonaws.com/instance-generation
          operator: Gt
          values:
            - '4'
        - key: kubernetes.io/arch
          operator: In
          values:
            - amd64
            - arm64
        - key: kubernetes.io/os
          operator: In
          values:
            - linux
      taints:
        - effect: NoSchedule
```

Ceci NodePool est similaire à « usage général », à l'exception des différences suivantes :

1. Il permet d'utiliser des nœuds dotés de l'architecture ARM et de l'architecture AMD.
2. Cela altère ces nœuds avec un NoSchedule sauf s'il existe une tolérance pour « »CriticalAddonsOnly. Ceci est destiné à un usage interne par les modules complémentaires EKS

personnalisé

Vous pouvez créer votre propre personnalisation NodePools en fonction de vos besoins. Pour en savoir plus, NodePools veuillez consulter la documentation de Karpenter.

Puis-je personnaliser l'AMI utilisée par le mode automatique lorsque de nouveaux nœuds sont lancés ?

Non, les seuls appareils pris en charge AMIs sont ceux fournis par Amazon Bottlerocket

Comment puis-je installer des outils ou des agents personnalisés sur mes hôtes Kubernetes ?

La personnalisation des AMI n'étant pas prise en charge, si vous avez besoin d'un logiciel au niveau de l'hôte pour des tâches telles que l'analyse de sécurité, vous devez déployer la charge de travail sous forme de Kubernetes. DaemonSet

Quels composants s'exécutent dans le plan de données de mon cluster lorsque je provisionne un nouveau cluster en mode automatique EKS ?

Si vous créez un cluster avec eksctl ou la console AWS, les seuls pods exécutés dans un cluster en mode automatique EKS sont les pods Kubernetes Metrics Server. Les autres composants du mode automatique d'EKS, tels que Karpenter, le contrôleur AWS Load Balancer et le pilote EBS CSI, sont tous exécutés et gérés hors cluster.

Quels sont les composants gérés exécutés pour prendre en charge mon nouveau cluster EKS Auto Mode ?

Le mode automatique d'EKS automatise complètement le déploiement de la plupart des éléments d'un plan de données nécessaires à la production de Kubernetes. Cela inclut notamment les éléments suivants :

- Karpenter, pour l'auto-scaling du calcul de votre cluster

- AWS Load Balancer Controller pour vous permettre d'exposer facilement les services Kubernetes via l'intégration automatisée d'Elastic Load Balancer
- EBS CSI
- PVC CNI
- Agent d'identité du pod EKS

Comment résoudre les problèmes liés aux composants du mode automatique qui fonctionnaient auparavant en tant que pods dans mon cluster ?

Avec le mode automatique d'EKS, de nombreux composants tels que le AWS Load Balancer Controller et Karpenter sont gérés pour vous en dehors de votre cluster. Vous n'aurez donc pas la même visibilité sur les journaux que celle à laquelle vous êtes habitué lorsque vous vous gérez vous-même. Si vous êtes dans une situation où vous devez résoudre des problèmes liés au fonctionnement d'une fonctionnalité du mode automatique, créez un ticket de support AWS.

Karpenter

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Karpenter est un projet open source conçu pour améliorer la gestion du cycle de vie des nœuds au sein des clusters Kubernetes. Il automatise le provisionnement et le déprovisionnement des nœuds en fonction des besoins de planification spécifiques des pods, ce qui permet une mise à l'échelle et une optimisation des coûts efficaces. Ses principales fonctions sont les suivantes :

- Surveillez les pods que le planificateur Kubernetes ne peut pas planifier en raison de contraintes de ressources.
- Évaluez les exigences de planification (demandes de ressources, sélecteurs de nœuds, affinités, tolérances, etc.) des pods non planifiables.
- Provisionnez de nouveaux nœuds qui répondent aux exigences de ces modules.
- Supprimez les nœuds lorsqu'ils ne sont plus nécessaires.

Avec Karpenter, vous pouvez définir NodePools des contraintes relatives au provisionnement des nœuds, telles que les taches, les étiquettes, les exigences (types d'instances, zones, etc.) et les limites du total des ressources provisionnées. Lorsque vous déployez des charges de travail, vous pouvez spécifier diverses contraintes de planification dans les spécifications du module, telles que les requests/limits, node selectors, node/pod affinités de ressources, les tolérances et les contraintes de dispersion topologique. Karpenter fournira ensuite des nœuds de la bonne taille en fonction de ces spécifications.

Pourquoi utiliser Karpenter

Avant le lancement de Karpenter, les utilisateurs de Kubernetes s'appuyaient principalement sur les groupes Amazon EC2 Auto Scaling et le Kubernetes Cluster Autoscaler (CAS) pour ajuster dynamiquement la capacité de calcul de leurs clusters. Avec Karpenter, vous n'avez pas besoin de créer des dizaines de groupes de nœuds pour bénéficier de la flexibilité et de la diversité que vous offre Karpenter. Contrairement à CAS, Karpenter n'est pas aussi étroitement lié aux versions de Kubernetes et ne vous oblige pas à passer d'AWS à Kubernetes. APIs

Karpenter consolide les responsabilités d'orchestration des instances au sein d'un système unique, plus simple, plus stable et adapté aux clusters. Karpenter a été conçu pour surmonter certains des défis présentés par Cluster Autoscaler en proposant des méthodes simplifiées pour :

- Provisionnez des nœuds en fonction des exigences de charge de travail.
- Créez diverses configurations de nœuds par type d'instance, à l'aide d'NodePool options flexibles. Au lieu de gérer de nombreux groupes de nœuds personnalisés spécifiques, Karpenter pourrait vous permettre de gérer diverses capacités de charge de travail avec un seul nœud flexible NodePool.
- Améliorez la planification des pods à grande échelle en lançant rapidement des nœuds et en planifiant des pods.

Pour obtenir des informations et de la documentation sur l'utilisation de Karpenter, rendez-vous sur le site karpenter.sh.

Recommandations

Les meilleures pratiques sont divisées en sections sur Karpenter lui-même et sur la planification des modules. NodePools

Les meilleures pratiques de Karpenter

Les meilleures pratiques suivantes couvrent des sujets liés à Karpenter lui-même.

Confinement AMIs dans les clusters de production

Nous vous recommandons vivement d'épingler les célèbres Amazon Machine Images (AMIs) utilisées par Karpenter pour les clusters de production. L'utilisation `amiSelector` d'un alias défini sur`@latest`, ou l'utilisation d'une autre méthode qui entraîne un déploiement non testé AMIs au moment de leur publication, présente le risque de défaillances de charge de travail et d'interruptions de service dans vos clusters de production. Par conséquent, nous vous recommandons vivement d'épingler les versions fonctionnelles testées de AMIs pour vos clusters de production pendant que vous testez les nouvelles versions dans des clusters hors production. Par exemple, vous pouvez définir un alias dans votre fichier NodeClass comme suit :

```
amiSelectorTerms
  - alias: al2023@v20240807
```

Pour plus d'informations sur la gestion et le repérage AMIs dans Karpenter, consultez la section Gestion AMIs dans la documentation de Karpenter.

Utilisez Karpenter pour les charges de travail dont les besoins en capacité évoluent

Karpenter rapproche la gestion de la mise à l'échelle de la version native de Kubernetes APIs plutôt que les groupes Autoscaling (ASGs) et les groupes de nœuds gérés (MNG). Les ASG et les MNG sont des abstractions natives d'AWS dans lesquelles le dimensionnement est déclenché en fonction de métriques au niveau d'AWS, telles que la charge du processeur EC2. Cluster Autoscaler fait le lien entre les abstractions Kubernetes et les abstractions AWS, mais perd une certaine flexibilité à cause de cela, comme la planification pour une zone de disponibilité spécifique.

Karpenter supprime une couche d'abstraction AWS pour apporter une partie de la flexibilité directement à Kubernetes. Karpenter est idéal pour les clusters dont les charges de travail sont soumises à des périodes de forte demande ou à des exigences informatiques diverses. MNGs et ASGs conviennent aux clusters exécutant des charges de travail qui ont tendance à être plus statiques et cohérentes. Vous pouvez utiliser une combinaison de nœuds gérés dynamiquement et statiquement, en fonction de vos besoins.

Envisagez d'autres projets de mise à l'échelle automatique lorsque...

Vous avez besoin de fonctionnalités qui sont toujours en cours de développement dans Karpenter. Karpenter étant un projet relativement récent, envisagez d'autres projets de mise à l'échelle automatique pour le moment si vous avez besoin de fonctionnalités qui ne font pas encore partie de Karpenter.

Exécutez le contrôleur Karpenter sur EKS Fargate ou sur un nœud de travail appartenant à un groupe de nœuds

Karpenter est installé à l'aide d'un graphique Helm. Le graphique Helm installe le contrôleur Karpenter et un module Webhook en tant que déploiement qui doit être exécuté avant que le contrôleur ne puisse être utilisé pour dimensionner votre cluster. Nous recommandons un minimum d'un petit groupe de nœuds avec au moins un nœud de travail. Vous pouvez également exécuter ces modules sur EKS Fargate en créant un profil Fargate pour l'espace de noms. `karpenter` Cela entraînera l'exécution de tous les pods déployés dans cet espace de noms sur EKS Fargate. N'exécutez pas Karpenter sur un nœud géré par Karpenter.

Aucun modèle de lancement personnalisé n'est pris en charge avec Karpenter

Aucun modèle de lancement personnalisé n'est pris en charge avec la version 1 APIs. Vous pouvez utiliser des données utilisateur personnalisées en spécifiant and/or directement la AMIs personnalisation dans le EC2NodeClass. De plus amples informations sur la manière de procéder sont disponibles sur NodeClasses.

Exclure les types d'instances qui ne correspondent pas à votre charge de travail

Envisagez d'exclure des types d'instances spécifiques à l'aide de la `node.kubernetes.io/instance-type` clé s'ils ne sont pas requis par les charges de travail exécutées dans votre cluster.

L'exemple suivant montre comment éviter de provisionner de grandes instances Graviton.

```
- key: node.kubernetes.io/instance-type
  operator: NotIn
  values:
  - m6g.16xlarge
  - m6gd.16xlarge
  - r6g.16xlarge
  - r6gd.16xlarge
```

```
  - c6g.16xlarge
```

Activer la gestion des interruptions lors de l'utilisation de Spot

Karpenter prend en charge la gestion native des interruptions et peut gérer les événements d'interruption involontaires tels que les interruptions liées aux instances Spot, les événements de maintenance planifiée, les événements de termination/d'arrêt d'instance susceptibles de perturber vos charges de travail. Lorsque Karpenter détecte de tels événements pour les nœuds, il entache, vide et arrête automatiquement les nœuds concernés à l'avance afin de commencer à nettoyer en douceur les charges de travail avant toute interruption. En cas d'interruption ponctuelle avec un préavis de 2 minutes, Karpenter démarre rapidement un nouveau nœud afin que les pods puissent être déplacés avant que l'instance ne soit récupérée. Pour activer la gestion des interruptions, vous configurez l'argument `--interruption-queue` CLI avec le nom de la file d'attente SQS configurée à cette fin. Il n'est pas conseillé d'utiliser la gestion des interruptions Karpenter avec Node Termination Handler, comme expliqué ici.

Les pods qui nécessitent des points de contrôle ou d'autres formes de vidange gracieuse, nécessitant 2 minutes avant l'arrêt, devraient permettre à Karpenter de gérer les interruptions dans leurs clusters.

Cluster privé Amazon EKS sans accès Internet sortant

Lorsque vous approvisionnez un cluster EKS dans un VPC sans accès à Internet, vous devez vous assurer que vous avez configuré votre environnement conformément aux exigences du cluster privé décrites dans la documentation EKS. En outre, vous devez vous assurer que vous avez créé un point de terminaison régional STS VPC dans votre VPC. Si ce n'est pas le cas, vous verrez des erreurs similaires à celles qui apparaissent ci-dessous.

```
{"level":"FATAL","time":"2024-02-29T14:28:34.392Z","logger":"controller","message":"Checking
 EC2 API connectivity, WebIdentityErr: failed to retrieve credentials
\ncaused by: RequestError: send request failed\ncaused by: Post
 \"https://sts.<region>.amazonaws.com/\": dial tcp 54.239.32.126:443: i/o
 timeout","commit":"596ea97"}
```

Ces modifications sont nécessaires dans un cluster privé car le Karpenter Controller utilise les rôles IAM pour les comptes de service (IRSA). Les pods configurés avec IRSA obtiennent des informations d'identification en appelant l'API AWS Security Token Service (AWS STS). S'il n'y a pas d'accès Internet sortant, vous devez créer et utiliser un point de terminaison VPC AWS STS dans votre VPC.

Les clusters privés nécessitent également que vous créiez un point de terminaison VPC pour SSM. Lorsque Karpenter essaie de provisionner un nouveau nœud, il interroge les configurations du

modèle de lancement et un paramètre SSM. Si vous n'avez pas de point de terminaison VPC SSM dans votre VPC, cela provoquera l'erreur suivante :

```
{"level":"ERROR","time":"2024-02-29T14:28:12.889Z","logger":"controller","message":"Unable
 to hydrate the AWS launch template cache, RequestCanceled: request context canceled
\ncaused by: context canceled","commit":"596ea97","tag-key":"karpenter.k8s.aws/
cluster","tag-value":"eks-workshop"}
...
{"level":"ERROR","time":"2024-02-29T15:08:58.869Z","logger":"controller.nodeclass","message":"d
 amis from ssm, getting ssm parameter \"/aws/service/eks/optimized-ami/1.27/amazon-
linux-2/recommended/image_id\", RequestError: send request failed\ncaused by:
 Post \"https://ssm.<region>.amazonaws.com/\": dial tcp 67.220.228.252:443: i/o
 timeout","commit":"596ea97","ec2nodeclass":"default","query":"/aws/service/eks/
optimized-ami/1.27/amazon-linux-2/recommended/image_id"}
```

Il n'existe aucun point de terminaison VPC pour l'API Price List Query. Par conséquent, les données sur les prix deviendront obsolètes au fil du temps. Karpenter contourne ce problème en incluant des données de tarification à la demande dans son binaire, mais ne met à jour ces données que lorsque Karpenter est mis à niveau. L'échec des demandes de données tarifaires entraînera les messages d'erreur suivants :

```
{"level":"ERROR","time":"2024-02-29T15:08:58.522Z","logger":"controller.pricing","message":"ret
 on-demand pricing data, RequestError: send request failed\ncaused by: Post
 \"https://api.pricing.<region>.amazonaws.com/\": dial tcp 18.196.224.8:443:
 i/o timeout; RequestError: send request failed\ncaused by: Post \"https://
api.pricing.<region>.amazonaws.com/\": dial tcp 18.185.143.117:443: i/o
 timeout","commit":"596ea97"}
```

Reportez-vous à cette documentation pour utiliser Karpenter dans un cluster EKS entièrement privé et pour savoir quels points de terminaison VPC doivent être créés.

Création NodePools

Les meilleures pratiques suivantes couvrent des sujets liés à la création NodePools.

Créez plusieurs NodePools lorsque...

Lorsque différentes équipes partagent un cluster et doivent exécuter leurs charges de travail sur différents nœuds de travail, ou ont des exigences différentes en matière de système d'exploitation ou de type d'instance, créez-en plusieurs NodePools. Par exemple, une équipe peut vouloir utiliser

Bottlerocket, tandis qu'une autre peut vouloir utiliser Amazon Linux. De même, une équipe peut avoir accès à du matériel GPU coûteux dont une autre n'aurait pas besoin. L'utilisation de plusieurs NodePools permet de s'assurer que les ressources les plus appropriées sont disponibles pour chaque équipe.

Créez des NodePools créations qui s'excluent mutuellement ou qui sont pondérées

Il est recommandé de créer des modèles NodePools qui s'excluent mutuellement ou qui sont pondérés afin de garantir un comportement de planification cohérent. Si ce n'est pas le cas et que plusieurs NodePools correspondent, Karpenter choisira au hasard laquelle utiliser, ce qui provoquera des résultats inattendus. Voici des exemples utiles pour créer plusieurs NodePools :

Création d'un NodePool avec GPU et autorisation uniquement de l'exécution de charges de travail spéciales sur ces nœuds (coûteux) :

```
# NodePool for GPU Instances with Taints
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu
spec:
  disruption:
    consolidateAfter: 1m
    consolidationPolicy: WhenEmptyOrUnderutilized
  template:
    metadata: {}
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
      expireAfter: Never
      requirements:
      - key: node.kubernetes.io/instance-type
        operator: In
        values:
        - p3.8xlarge
        - p3.16xlarge
      - key: kubernetes.io/os
        operator: In
        values:
        - linux
```

```
      - key: kubernetes.io/arch
        operator: In
        values:
        - amd64
      - key: karpenter.sh/capacity-type
        operator: In
        values:
        - on-demand
      taints:
      - effect: NoSchedule
        key: nvidia.com/gpu
        value: "true"
```

Déploiement avec tolérance aux salissures :

```
# Deployment of GPU Workload will have tolerations defined
apiVersion: apps/v1
kind: Deployment
metadata:
  name: inflate-gpu
spec:
    spec:
      tolerations:
      - key: "nvidia.com/gpu"
        operator: "Exists"
        effect: "NoSchedule"
```

Pour un déploiement général pour une autre équipe, la NodePool spécification peut inclure NodeAffinity. Un déploiement pourrait alors être utilisé nodeSelectorTerms pour correspondre`billing-team`.

```
# NodePool for regular EC2 instances
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: generalcompute
spec:
  template:
    metadata:
      labels:
        billing-team: my-team
    spec:
```

```
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
      expireAfter: Never
      requirements:
      - key: node.kubernetes.io/instance-type
        operator: In
        values:
        - m5.large
        - m5.xlarge
        - m5.2xlarge
        - c5.large
        - c5.xlarge
        - c5a.large
        - c5a.xlarge
        - r5.large
        - r5.xlarge
      - key: kubernetes.io/os
        operator: In
        values:
        - linux
      - key: kubernetes.io/arch
        operator: In
        values:
        - amd64
      - key: karpenter.sh/capacity-type
        operator: In
        values:
        - on-demand
```

Déploiement à l'aide de NodeAffinity :

```
# Deployment will have spec.affinity.nodeAffinity defined
kind: Deployment
metadata:
  name: workload-my-team
spec:
  replicas: 200
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
```

```
            nodeSelectorTerms:
              - matchExpressions:
                - key: "billing-team"
                  operator: "In"
                  values: ["my-team"]
```

Utilisez des minuteries (TTL) pour supprimer automatiquement les nœuds du cluster

Vous pouvez utiliser des minuteries sur les nœuds provisionnés pour définir à quel moment supprimer les nœuds dépourvus de modules de charge de travail ou ayant atteint une date d'expiration. L'expiration des nœuds peut être utilisée comme moyen de mise à niveau, afin que les nœuds soient retirés et remplacés par des versions mises à jour. Voir Expiration dans la documentation de Karpenter pour plus d'informations sur l'utilisation `spec.template.spec` pour configurer l'expiration des nœuds.

Évitez de trop restreindre les types d'instances que Karpenter peut fournir, en particulier lors de l'utilisation de Spot

Lors de l'utilisation de Spot, Karpenter utilise la stratégie d'allocation Price Capacity Optimized pour approvisionner les instances EC2. Cette stratégie demande à EC2 de provisionner les instances issues des pools les plus profonds en fonction du nombre d'instances que vous lancez et de réduire au maximum le risque d'interruption. EC2 Fleet demande ensuite des instances Spot auprès du pool le moins cher. Plus vous autorisez Karpenter à utiliser de types d'instances, plus EC2 peut optimiser le temps d'exécution de votre instance ponctuelle. Par défaut, Karpenter utilisera tous les types d'instances proposés par EC2 dans la région et les zones de disponibilité dans lesquelles votre cluster est déployé. Karpenter choisit intelligemment parmi tous les types d'instances en fonction des pods en attente pour s'assurer que vos pods sont planifiés sur des instances de taille et d'équipement appropriés. Par exemple, si votre pod ne nécessite pas de GPU, Karpenter ne planifiera pas votre pod sur un type d'instance EC2 prenant en charge un GPU. Si vous n'êtes pas sûr des types d'instances à utiliser, vous pouvez exécuter le sélecteur d'instance Amazon ec2-instance-selector pour générer une liste des types d'instances correspondant à vos besoins de calcul. Par exemple, la CLI prend la mémoire vCPU, l'architecture et la région comme paramètres d'entrée et vous fournit une liste d'instances EC2 qui répondent à ces contraintes.

```
$ ec2-instance-selector --memory 4 --vcpus 2 --cpu-architecture x86_64 -r ap-
southeast-1
c5.large
c5a.large
c5ad.large
```

```
c5d.large
c6i.large
t2.medium
t3.medium
t3a.medium
```

Vous ne devez pas imposer trop de contraintes à Karpenter lorsque vous utilisez des instances Spot, car cela peut affecter la disponibilité de vos applications. Supposons, par exemple, que toutes les instances d'un type particulier soient récupérées et qu'il n'existe aucune alternative appropriée pour les remplacer. Vos pods resteront en attente jusqu'à ce que la capacité ponctuelle pour les types d'instances configurés soit réapprovisionnée. Vous pouvez réduire le risque d'erreurs liées à une capacité insuffisante en répartissant vos instances entre différentes zones de disponibilité, car les pools d'instances sont différents d'un endroit à l'autre AZs. Cela dit, la meilleure pratique générale consiste à autoriser Karpenter à utiliser un ensemble diversifié de types d'instances lors de l'utilisation de Spot.

Pods de planification

Les meilleures pratiques suivantes concernent le déploiement de pods dans un cluster à l'aide de Karpenter pour le provisionnement des nœuds.

Suivez les meilleures pratiques d'EKS en matière de haute disponibilité

Si vous devez exécuter des applications à haute disponibilité, suivez les recommandations générales des meilleures pratiques d'EKS. Consultez la documentation Topology Spread in Karpenter pour plus de détails sur la façon de répartir les pods sur les nœuds et les zones. Utilisez les budgets d'interruption pour définir le nombre minimum de pods disponibles qui doivent être entretenus, en cas de tentative d'expulsion ou de suppression de pods.

Utilisez des contraintes en couches pour limiter les fonctionnalités de calcul disponibles auprès de votre fournisseur de cloud

Le modèle de contraintes en couches de Karpenter vous permet de créer un ensemble complexe de contraintes de déploiement NodePool des modules afin d'obtenir les meilleures correspondances possibles pour la planification des modules. Voici des exemples de contraintes qu'une spécification de pod peut demander :

- Nécessité de fonctionner dans des zones de disponibilité où seules des applications spécifiques sont disponibles. Supposons, par exemple, que vous ayez un pod qui doit communiquer avec une autre application qui s'exécute sur une instance EC2 résidant dans une zone de disponibilité

particulière. Si votre objectif est de réduire le trafic inter-AZ dans votre VPC, vous souhaiterez peut-être co-localiser les pods dans l'AZ où se trouve l'instance EC2. Ce type de ciblage est souvent réalisé à l'aide de sélecteurs de nœuds. Pour plus d'informations sur les sélecteurs de nœuds, consultez la documentation de Kubernetes.

- Nécessitant certains types de processeurs ou d'autres matériels. Consultez la section Accélérateurs de la documentation de Karpenter pour un exemple de spécification de pod qui nécessite que le pod fonctionne sur un GPU.

Créez des alarmes de facturation pour surveiller vos dépenses informatiques

Lorsque vous configurez votre cluster pour qu'il évolue automatiquement, vous devez créer des alarmes de facturation pour vous avertir lorsque vos dépenses dépassent un seuil et ajouter des limites de ressources à votre configuration Karpenter. La définition des limites de ressources avec Karpenter est similaire à la définition de la capacité maximale d'un groupe AWS Autoscaling dans la mesure où elle représente la quantité maximale de ressources de calcul pouvant être instanciées par un Karpenter. NodePool

> **Note**
>
> Il n'est pas possible de définir une limite globale pour l'ensemble du cluster. Les limites s'appliquent à des domaines spécifiques NodePools.

L'extrait ci-dessous indique à Karpenter de ne fournir qu'un maximum de 1 000 cœurs de processeur et 1 000 Go de mémoire. Karpenter arrêtera d'ajouter de la capacité uniquement lorsque la limite sera atteinte ou dépassée. Lorsqu'une limite est dépassée, le contrôleur Karpenter écrit un message `memory resource usage of 1001 exceeds limit of 1000` ou un message similaire dans les journaux du contrôleur. Si vous acheminez les journaux de vos conteneurs vers CloudWatch des journaux, vous pouvez créer un filtre de métriques pour rechercher des modèles ou des termes spécifiques dans vos journaux, puis créer une CloudWatchalarme pour vous avertir lorsque le seuil de métriques que vous avez configuré est dépassé.

Pour plus d'informations sur l'utilisation des limites avec Karpenter, consultez la section Définition des limites de ressources dans la documentation de Karpenter.

```
spec:
  limits:
    cpu: 1000
```

```
    memory: 1000Gi
```

Si vous n'utilisez pas de limites ou ne limitez pas les types d'instances que Karpenter peut provisionner, Karpenter continuera d'ajouter de la capacité de calcul à votre cluster selon les besoins. Si cette configuration de Karpenter permet à votre cluster de s'adapter librement, cela peut également avoir des implications financières importantes. C'est pour cette raison que nous recommandons de configurer les alarmes de facturation. Les alarmes de facturation vous permettent d'être alerté et averti de manière proactive lorsque les frais estimés calculés sur votre ou vos comptes dépassent un seuil défini. Consultez Configuration d'une alarme CloudWatch de facturation Amazon pour surveiller de manière proactive les frais estimés pour plus d'informations.

Vous pouvez également activer la détection des anomalies de coûts, une fonctionnalité de gestion des coûts d'AWS qui utilise l'apprentissage automatique pour surveiller en permanence vos coûts et votre utilisation afin de détecter les dépenses inhabituelles. Vous trouverez de plus amples informations dans le guide de démarrage d'AWS Cost Anomaly Detection. Si vous êtes allé jusqu'à créer un budget dans AWS Budgets, vous pouvez également configurer une action pour vous avertir lorsqu'un seuil spécifique a été dépassé. Avec les actions budgétaires, vous pouvez envoyer un e-mail, publier un message sur un sujet SNS ou envoyer un message à un chatbot tel que Slack. Pour plus d'informations, consultez Configuration des actions AWS Budgets.

Utilisez l'do-not-disrupt annotation karpenter.sh/ pour empêcher Karpenter de déprovisionner un nœud

Si vous exécutez une application critique sur un nœud approvisionné par Karpenter, comme un traitement par lots de longue durée ou une application dynamique, et que le TTL du nœud a expiré, l'application sera interrompue lorsque l'instance sera arrêtée. En ajoutant une `karpenter.sh/do-not-disrupt` annotation au pod, vous demandez à Karpenter de conserver le nœud jusqu'à ce que le pod soit fermé ou que l'`karpenter.sh/do-not-disrupt`annotation soit supprimée. Consultez la documentation de Distruption pour plus d'informations.

Si les seuls pods qui restent sur un nœud ne relevant pas du daemonset sont ceux associés aux tâches, Karpenter est en mesure de cibler et de supprimer ces nœuds à condition que le statut de la tâche soit « réussi » ou « échec ».

Configurer requests=limits pour toutes les ressources autres que le processeur lors de l'utilisation de la consolidation

La consolidation et la planification fonctionnent en général en comparant les demandes de ressources du pod à la quantité de ressources allouables sur un nœud. Les limites de ressources

ne sont pas prises en compte. Par exemple, les pods dont la limite de mémoire est supérieure à la demande de mémoire peuvent dépasser la demande. Si plusieurs modules d'un même nœud éclatent en même temps, cela peut entraîner la fermeture de certains modules en raison d'un manque de mémoire (OOM). La consolidation peut augmenter les chances que cela se produise, car il est possible de regrouper les pods sur les nœuds uniquement en tenant compte de leurs demandes.

LimitRanges À utiliser pour configurer les valeurs par défaut pour les demandes de ressources et les limites

Kubernetes ne définissant pas de requêtes ou de limites par défaut, la consommation de ressources d'un conteneur provenant de l'hôte, du processeur et de la mémoire sous-jacents n'est pas limitée. Le planificateur Kubernetes examine le nombre total de demandes d'un pod (le plus élevé entre le nombre total de demandes provenant des conteneurs du pod ou le total des ressources provenant des conteneurs d'initialisation du pod) afin de déterminer sur quel nœud de travail planifier le pod. De même, Karpenter prend en compte les demandes d'un module pour déterminer le type d'instance qu'il fournit. Vous pouvez utiliser une plage de limites pour appliquer une valeur par défaut raisonnable à un espace de noms, au cas où les demandes de ressources ne seraient pas spécifiées par certains pods.

Voir Configurer les demandes de mémoire par défaut et les limites pour un espace de noms

Appliquez des demandes de ressources précises à toutes les charges de travail

Karpenter est en mesure de lancer les nœuds les mieux adaptés à vos charges de travail lorsque ses informations concernant vos exigences en matière de charges de travail sont exactes. Cela est particulièrement important si vous utilisez la fonction de consolidation de Karpenter.

Voir Configurer et dimensionner les ressources Requests/Limits pour toutes les charges de travail

Recommandations du CoreDNS

Mettre à jour la configuration de CoreDNS pour maintenir la fiabilité

Lors du déploiement de pods CoreDNS sur des nœuds gérés par Karpenter, étant donné la nature dynamique de Karpenter dans les nouveaux nœuds terminating/creating rapidement adaptés à la demande, il est conseillé de respecter les meilleures pratiques suivantes :

Durée du lameduck CoreDNS

Sonde de préparation CoreDNS

Cela garantira que les requêtes DNS ne sont pas dirigées vers un module CoreDNS qui n'est pas encore prêt ou qui a été arrêté.

Plans de charpentier

Karpenter adopte une approche axée sur les applications pour fournir de la capacité de calcul au plan de données Kubernetes. Dans certains scénarios de charge de travail courants, vous vous demandez peut-être comment les configurer correctement. Karpenter Blueprints est un référentiel qui inclut une liste de scénarios de charge de travail courants conformes aux meilleures pratiques décrites ici. Vous disposerez de toutes les ressources nécessaires pour même créer un cluster EKS avec Karpenter configuré et tester chacun des plans inclus dans le référentiel. Vous pouvez combiner différents plans pour créer enfin celui dont vous avez besoin pour votre ou vos charges de travail.

Ressources supplémentaires

- Atelier d'immersion d'une journée chez Karpenter
- Atelier d'optimisation des coûts de Karpenter
- Atelier EKS - Charpentier
- Karpenter ou Cluster Autoscaler
- Séance Karpenter à re:Invent 2023
- Tutoriel : Exécutez des clusters Kubernetes à moindre coût avec Amazon EC2 Spot et Karpenter

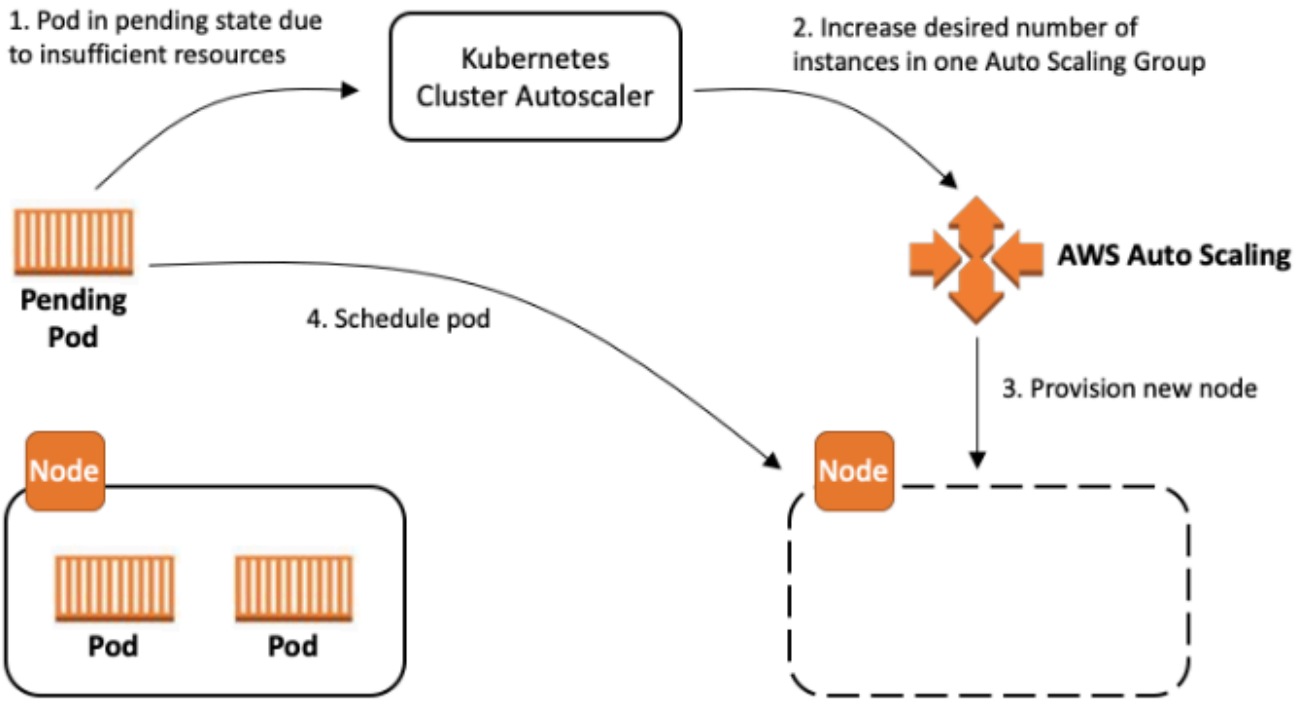
Cluster Autoscaler

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Présentation de

Le Kubernetes Cluster Autoscaler est une solution populaire de mise à l'échelle automatique de cluster gérée par SIG Autoscaling. Il est chargé de s'assurer que votre cluster dispose de suffisamment de nœuds pour planifier vos pods sans gaspiller de ressources. Il surveille les pods qui ne sont pas planifiés et les nœuds sous-utilisés. Il simule ensuite l'ajout ou la suppression de nœuds

avant d'appliquer la modification à votre cluster. L'implémentation d'AWS Cloud Provider dans Cluster Autoscaler contrôle le `.DesiredReplicas` champ de vos groupes EC2 Auto Scaling.



Ce guide fournit un modèle mental pour configurer le Cluster Autoscaler et choisir le meilleur ensemble de compromis pour répondre aux exigences de votre organisation. Bien qu'il n'existe pas de meilleure configuration, il existe un ensemble d'options de configuration qui vous permettent de trouver un compromis entre performances, évolutivité, coût et disponibilité. En outre, ce guide fournit des conseils et des bonnes pratiques pour optimiser votre configuration pour AWS.

Glossaire

La terminologie suivante sera fréquemment utilisée dans ce document. Ces termes peuvent avoir un sens large, mais sont limités aux définitions ci-dessous aux fins du présent document.

L'évolutivité fait référence aux performances du Cluster Autoscaler lorsque le nombre de pods et de nœuds de votre cluster Kubernetes augmente. Lorsque les limites d'évolutivité sont atteintes, les performances et les fonctionnalités du Cluster Autoscaler se dégradent. Lorsque le Cluster Autoscaler dépasse ses limites d'évolutivité, il ne peut plus ajouter ou supprimer de nœuds dans votre cluster.

Les performances font référence à la rapidité avec laquelle le Cluster Autoscaler est capable de prendre et d'exécuter des décisions de dimensionnement. Un Cluster Autoscaler parfaitement

performant prendrait instantanément une décision et déclencherait une action de mise à l'échelle en réponse à des stimuli, tels qu'un module devenant imprévisible.

La disponibilité signifie que les pods peuvent être programmés rapidement et sans interruption. Cela inclut les cas où les pods nouvellement créés doivent être planifiés et lorsqu'un nœud réduit met fin à tous les pods restants planifiés pour celui-ci.

Le coût est déterminé par la décision qui sous-tend la mise à l'échelle et à l'échelle des événements. Les ressources sont gaspillées si un nœud existant est sous-utilisé ou si un nouveau nœud trop grand est ajouté pour les pods entrants. Selon le cas d'utilisation, l'arrêt prématuré des pods peut entraîner des coûts en raison d'une décision agressive de réduction de la taille.

Les groupes de nœuds sont un concept Kubernetes abstrait désignant un groupe de nœuds au sein d'un cluster. Il ne s'agit pas d'une véritable ressource Kubernetes, mais elle existe sous forme d'abstraction dans le Cluster Autoscaler, l'API Cluster et d'autres composants. Les nœuds d'un groupe de nœuds partagent des propriétés telles que des étiquettes et des taches, mais peuvent être constitués de plusieurs zones de disponibilité ou types d'instances.

Les groupes EC2 Auto Scaling peuvent être utilisés comme implémentation de groupes de nœuds sur EC2. Les groupes EC2 Auto Scaling sont configurés pour lancer des instances qui rejoignent automatiquement leurs clusters Kubernetes et appliquent des étiquettes et des modifications à la ressource de nœud correspondante dans l'API Kubernetes.

Les groupes de nœuds gérés EC2 sont une autre implémentation des groupes de nœuds sur EC2. Ils simplifient la configuration manuelle des groupes de scalage automatique EC2 et fournissent des fonctionnalités de gestion supplémentaires, telles que la mise à niveau de la version des nœuds et la terminaison progressive des nœuds.

Utilisation du Cluster Autoscaler

Le Cluster Autoscaler est généralement installé en tant que Déploiement dans votre cluster. Il utilise l'élection des leaders pour garantir une haute disponibilité, mais le travail est effectué par une seule réplique à la fois. Il n'est pas évolutif horizontalement. Pour les configurations de base, la configuration par défaut devrait fonctionner immédiatement en suivant les instructions d'installation fournies, mais il y a quelques points à garder à l'esprit.

Assurez-vous que :

- La version du Cluster Autoscaler correspond à la version du Cluster. La compatibilité entre versions n'est ni testée ni prise en charge.

- La découverte automatique est activée, sauf si vous avez des cas d'utilisation avancés spécifiques qui empêchent l'utilisation de ce mode.

Utiliser l'accès le moins privilégié au rôle IAM

Lorsque l'Auto Discovery est utilisé, nous vous recommandons vivement d'utiliser l'accès avec le moindre privilège en limitant les actions `autoscaling:SetDesiredCapacity` et `autoscaling:TerminateInstanceInAutoScalingGroup` aux groupes Auto Scaling dont le périmètre est limité au cluster actuel.

Cela empêchera un cluster Autoscaler exécuté dans un cluster de modifier les groupes de nœuds d'un autre cluster même si l'`--node-group-auto-discovery`argument n'était pas limité aux groupes de nœuds du cluster à l'aide de balises (par exemple). `k8s.io/cluster-autoscaler/<cluster-name>`

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "autoscaling:SetDesiredCapacity",
                "autoscaling:TerminateInstanceInAutoScalingGroup"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/k8s.io/cluster-autoscaler/enabled": "true",
                    "aws:ResourceTag/k8s.io/cluster-autoscaler/my-cluster": "owned"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeAutoScalingInstances",
                "autoscaling:DescribeLaunchConfigurations",
                "autoscaling:DescribeScalingActivities",
                "autoscaling:DescribeTags",
                "ec2:DescribeImages",
```

```
                "ec2:DescribeInstanceTypes",
                "ec2:DescribeLaunchTemplateVersions",
                "ec2:GetInstanceTypesFromInstanceRequirements",
                "eks:DescribeNodegroup"
            ],
            "Resource": "*"
        }
    ]
}
```

Configuration de vos groupes de nœuds

Une mise à l'échelle automatique efficace commence par la configuration correcte d'un ensemble de groupes de nœuds pour votre cluster. Il est essentiel de sélectionner le bon ensemble de groupes de nœuds pour optimiser la disponibilité et réduire les coûts liés à vos charges de travail. AWS implémente des groupes de nœuds à l'aide des groupes EC2 Auto Scaling, qui sont flexibles pour un grand nombre de cas d'utilisation. Cependant, le Cluster Autoscaler émet certaines hypothèses concernant vos groupes de nœuds. En maintenant la cohérence des configurations de votre groupe EC2 Auto Scaling avec ces hypothèses, vous minimiserez les comportements indésirables.

Assurez-vous que :

- Chaque nœud d'un groupe de nœuds possède des propriétés de planification identiques, telles que les étiquettes, les taches et les ressources.
  - En MixedInstancePolicies effet, les types d'instance doivent avoir la même forme pour le processeur, la mémoire et le processeur graphique
  - Le premier type d'instance spécifié dans la politique sera utilisé pour simuler la planification.
  - Si votre politique prévoit des types d'instances supplémentaires dotés de plus de ressources, les ressources risquent d'être gaspillées en cas de mise à l'échelle.
  - Si votre politique prévoit des types d'instances supplémentaires avec moins de ressources, les pods risquent de ne pas être planifiés sur les instances.
- Les groupes de nœuds avec de nombreux nœuds sont préférés à de nombreux groupes de nœuds avec moins de nœuds. Cela aura le plus grand impact sur l'évolutivité.
- Dans la mesure du possible, privilégiez les fonctionnalités EC2 lorsque les deux systèmes fournissent un support (par exemple, Régions, MixedInstancePolicy)

> **Note**
>
> Nous vous recommandons d'utiliser les groupes de nœuds gérés par EKS. Les groupes de nœuds gérés sont dotés de puissantes fonctionnalités de gestion, notamment des fonctionnalités pour Cluster Autoscaler, telles que la découverte automatique des groupes EC2 Auto Scaling et la terminaison progressive des nœuds.

Optimisation des performances et de l'évolutivité

Comprendre la complexité d'exécution de l'algorithme de mise à l'échelle automatique vous aidera à régler le Cluster Autoscaler pour qu'il continue à fonctionner correctement dans les grands clusters de plus de 1 000 nœuds.

Les principaux boutons permettant de régler l'évolutivité du Cluster Autoscaler sont les ressources fournies au processus, l'intervalle d'analyse de l'algorithme et le nombre de groupes de nœuds dans le cluster. D'autres facteurs interviennent dans la véritable complexité d'exécution de cet algorithme, tels que la complexité des plugins de planification et le nombre de pods. Ces paramètres sont considérés comme non configurables car ils sont naturels à la charge de travail du cluster et ne peuvent pas être facilement ajustés.

Le Cluster Autoscaler charge l'état complet du cluster en mémoire, y compris les pods, les nœuds et les groupes de nœuds. À chaque intervalle d'analyse, l'algorithme identifie les pods non planifiables et simule la planification pour chaque groupe de nœuds. Le réglage de ces facteurs s'accompagne de différents compromis qui doivent être soigneusement pris en compte pour votre cas d'utilisation.

Mise à l'échelle automatique verticale du cluster Autoscaler

Le moyen le plus simple de faire évoluer le Cluster Autoscaler vers des clusters plus importants consiste à augmenter le nombre de demandes de ressources pour son déploiement. La mémoire et le processeur doivent être augmentés pour les clusters de grande taille, bien que cela varie considérablement en fonction de la taille du cluster. L'algorithme de mise à l'échelle automatique stocke tous les pods et nœuds en mémoire, ce qui peut entraîner une empreinte mémoire supérieure à un gigaoctet dans certains cas. L'augmentation des ressources se fait généralement manuellement. Si vous constatez que le réglage constant des ressources crée une charge opérationnelle, pensez à utiliser l'Addon Resizer ou le Vertical Pod Autoscaler.

Réduction du nombre de groupes de nœuds

Minimiser le nombre de groupes de nœuds est un moyen de garantir que le Cluster Autoscaler continuera à fonctionner correctement sur les clusters de grande taille. Cela peut s'avérer difficile pour certaines organisations qui structurent leurs groupes de nœuds par équipe ou par application. Bien que cela soit entièrement pris en charge par l'API Kubernetes, cela est considéré comme un anti-modèle Cluster Autoscaler ayant des répercussions sur l'évolutivité. Il existe de nombreuses raisons d'utiliser plusieurs groupes de nœuds (par exemple, Spot ou GPUs), mais dans de nombreux cas, il existe d'autres conceptions qui permettent d'obtenir le même effet tout en utilisant un petit nombre de groupes.

Assurez-vous que :

- L'isolation des pods est effectuée à l'aide d'espaces de noms plutôt que de groupes de nœuds.
  - Cela peut ne pas être possible dans les clusters multi-locataires à faible niveau de confiance.
  - Pod ResourceRequests et ResourceLimits sont correctement réglés pour éviter les problèmes de ressources.
  - Des types d'instances plus importants permettront d'optimiser l'emballage des bacs et de réduire la surcharge du module système.
- NodeTaints ou NodeSelectors sont utilisés pour planifier les pods comme exception, et non comme règle.
- Les ressources régionales sont définies comme un seul groupe EC2 Auto Scaling avec plusieurs zones de disponibilité.

Réduction de l'intervalle de numérisation

Un faible intervalle de numérisation (10 secondes, par exemple) permettra au Cluster Autoscaler de répondre le plus rapidement possible lorsque les pods ne sont pas programmables. Cependant, chaque analyse entraîne de nombreux appels d'API à l'API Kubernetes et au groupe EC2 Auto Scaling ou au groupe de nœuds gérés EKS. APIs Ces appels d'API peuvent entraîner une limitation du débit ou même une indisponibilité du service pour votre plan de contrôle Kubernetes.

L'intervalle d'analyse par défaut est de 10 secondes, mais sur AWS, le lancement d'un nœud prend beaucoup plus de temps pour lancer une nouvelle instance. Cela signifie qu'il est possible d'augmenter l'intervalle sans augmenter significativement le temps de mise à l'échelle globale. Par exemple, si le lancement d'un nœud prend 2 minutes, la modification de l'intervalle à 1 minute permettra de réduire de 6 fois le nombre d'appels d'API contre des mises à l'échelle 38 % plus lentes.

Partage entre groupes de nœuds

Le Cluster Autoscaler peut être configuré pour fonctionner sur un ensemble spécifique de groupes de nœuds. Grâce à cette fonctionnalité, il est possible de déployer plusieurs instances du Cluster Autoscaler, chacune étant configurée pour fonctionner sur un ensemble différent de groupes de nœuds. Cette stratégie vous permet d'utiliser un nombre arbitrairement important de groupes de nœuds, en échangeant les coûts contre l'évolutivité. Nous recommandons de ne l'utiliser qu'en dernier recours pour améliorer les performances.

Le Cluster Autoscaler n'a pas été conçu à l'origine pour cette configuration, il présente donc certains effets secondaires. Comme les partitions ne communiquent pas, il est possible que plusieurs autoscalers tentent de planifier un pod non planifiable. Cela peut entraîner une extension inutile de plusieurs groupes de nœuds. Ces nœuds supplémentaires seront réduits après le`scale-down-delay`.

```
metadata:
  name: cluster-autoscaler
  namespace: cluster-autoscaler-1

...

--nodes=1:10:k8s-worker-asg-1
--nodes=1:10:k8s-worker-asg-2

---

metadata:
  name: cluster-autoscaler
  namespace: cluster-autoscaler-2

...

--nodes=1:10:k8s-worker-asg-3
--nodes=1:10:k8s-worker-asg-4
```

Assurez-vous que :

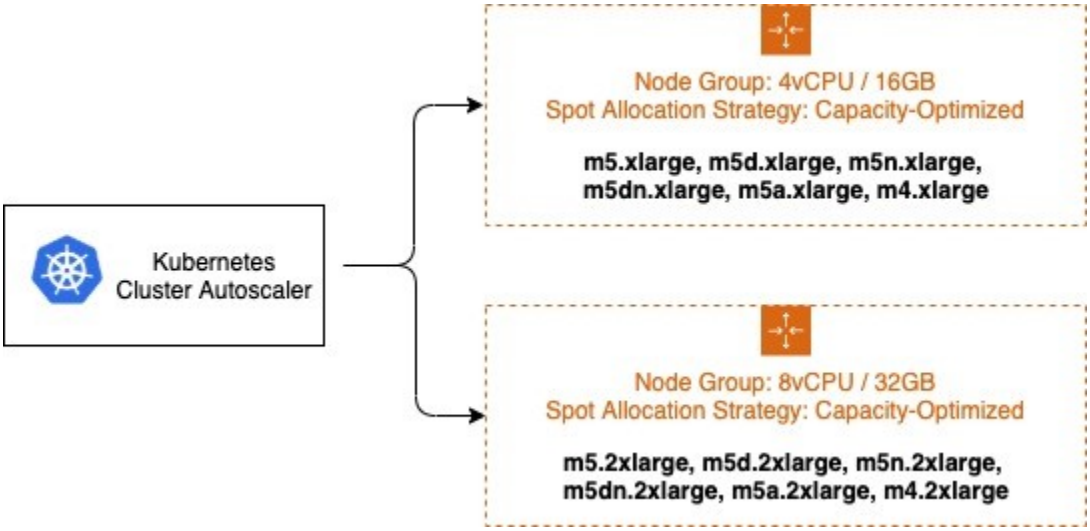
- Chaque partition est configurée pour pointer vers un ensemble unique de groupes EC2 Auto Scaling
- Chaque partition est déployée dans un espace de noms distinct afin d'éviter les conflits liés à l'élection du leader

Optimisation des coûts et de la disponibilité

Instances Spot

Vous pouvez utiliser des instances Spot dans vos groupes de nœuds et économiser jusqu'à 90 % sur le prix à la demande. En contrepartie, les instances Spot peuvent être interrompues à tout moment lorsqu'EC2 a besoin de récupérer sa capacité. Des erreurs de capacité insuffisante se produiront lorsque votre groupe EC2 Auto Scaling ne pourra pas augmenter en raison d'un manque de capacité disponible. En optimisant la diversité en sélectionnant de nombreuses familles d'instances, vous pouvez augmenter vos chances d'atteindre l'échelle souhaitée en exploitant de nombreux pools de capacité Spot et en réduisant l'impact des interruptions des instances ponctuelles sur la disponibilité de votre cluster. Les politiques d'instance mixte avec les instances Spot sont un excellent moyen d'augmenter la diversité sans augmenter le nombre de groupes de nœuds. N'oubliez pas que si vous avez besoin de ressources garanties, utilisez des instances à la demande plutôt que des instances ponctuelles.

Il est essentiel que tous les types d'instances disposent d'une capacité de ressources similaire lors de la configuration des politiques d'instance mixtes. Le simulateur de planification de l'autoscaler utilise le premier InstanceType du. MixedInstancePolicy Si les types d'instances suivants sont plus importants, les ressources risquent d'être gaspillées après une mise à l'échelle. S'ils sont plus petits, vos pods risquent de ne pas être programmés pour les nouvelles instances en raison d'une capacité insuffisante. Par exemple, les instances M4, M5, M5a et M5n ont toutes des quantités similaires de processeur et de mémoire et sont d'excellentes candidates pour un. MixedInstancePolicy L'outil EC2 Instance Selector peut vous aider à identifier des types d'instances similaires.



Il est recommandé d'isoler les capacités On-Demand et Spot dans des groupes EC2 Auto Scaling distincts. Cela est préférable à l'utilisation d'une stratégie de capacité de base car les propriétés de planification sont fondamentalement différentes. Comme les instances Spot peuvent être interrompues à tout moment (lorsqu'EC2 a besoin de récupérer de la capacité), les utilisateurs altèrent souvent leurs nœuds préemptables, ce qui nécessite une tolérance explicite des pods quant au comportement de préemption. Ces altérations se traduisent par des propriétés de planification différentes pour les nœuds. Ils doivent donc être séparés en plusieurs groupes EC2 Auto Scaling.

Le Cluster Autoscaler utilise un concept d'extenseurs, qui propose différentes stratégies pour sélectionner le groupe de nœuds à dimensionner. La stratégie `--expander=least-waste` est une bonne stratégie générale par défaut, et si vous comptez utiliser plusieurs groupes de nœuds pour diversifier les instances Spot (comme décrit dans l'image ci-dessus), cela pourrait contribuer à optimiser davantage les coûts des groupes de nœuds en redimensionnant le groupe qui serait le mieux utilisé après l'activité de dimensionnement.

Prioriser un groupe de nœuds/ASG

Vous pouvez également configurer la mise à l'échelle automatique basée sur les priorités à l'aide de l'extenseur Priority. `--expander=priority`permet à votre cluster de donner la priorité à un groupe de nœuds /ASG, et s'il n'est pas en mesure d'évoluer pour une raison quelconque, il choisira le groupe de nœuds suivant dans la liste des priorités. Cela est utile dans les situations où, par exemple, vous souhaitez utiliser des types d'instances P3 car leur GPU fournit des performances optimales pour votre charge de travail, mais comme deuxième option, vous pouvez également utiliser des types d'instances P2.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-autoscaler-priority-expander
  namespace: kube-system
data:
  priorities: |-
    10:
      - .*p2-node-group.*
    50:
      - .*p3-node-group.*
```

Cluster Autoscaler essaiera d'augmenter le groupe EC2 Auto Scaling correspondant au nom p3-node-group. Si cette opération échoue dans le délai imparti`--max-node-provision-time`, elle tentera de redimensionner un groupe EC2 Auto Scaling portant le nom p2-node-group. Cette valeur

par défaut est de 15 minutes et peut être réduite pour une sélection plus réactive des groupes de nœuds. Toutefois, si la valeur est trop faible, cela peut entraîner des redimensionnements inutiles.

Surapprovisionnement

Le Cluster Autoscaler minimise les coûts en garantissant que les nœuds ne sont ajoutés au cluster que lorsque cela est nécessaire et qu'ils sont supprimés lorsqu'ils ne sont pas utilisés. Cela a un impact significatif sur la latence du déploiement, car de nombreux pods seront obligés d'attendre qu'un nœud augmente avant de pouvoir être planifiés. Les nœuds peuvent prendre plusieurs minutes pour devenir disponibles, ce qui peut augmenter la latence de planification des pods d'un certain ordre de grandeur.

Cela peut être atténué par un surapprovisionnement, qui négocie le coût de la latence de planification. Le surprovisionnement est mis en œuvre à l'aide de pods temporaires à priorité négative, qui occupent de l'espace dans le cluster. Lorsque les modules nouvellement créés ne sont pas planifiables et ont une priorité plus élevée, les modules temporaires sont préemptés pour faire de la place. Les pods temporaires deviennent alors non planifiables, ce qui incite le Cluster Autoscaler à étendre les nouveaux nœuds surapprovisionnés.

Le surprovisionnement présente d'autres avantages moins évidents. Sans surprovisionnement, l'un des effets secondaires d'un cluster très utilisé est que les pods prendront des décisions de planification moins optimales en utilisant la `preferredDuringSchedulingIgnoredDuringExecution` règle de l'affinité des pods ou des nœuds. Un cas d'utilisation courant consiste à séparer les pods d'une application hautement disponible entre les zones de disponibilité à l'aide de AntiAffinity. Le surprovisionnement peut augmenter de manière significative les chances qu'un nœud de la bonne zone soit disponible.

La quantité de capacité surprovisionnée est une décision commerciale prudente pour votre entreprise. Il s'agit essentiellement d'un compromis entre performance et coût. L'une des façons de prendre cette décision consiste à déterminer votre fréquence de mise à l'échelle moyenne et à la diviser par le temps nécessaire pour augmenter la capacité d'un nouveau nœud. Par exemple, si, en moyenne, vous avez besoin d'un nouveau nœud toutes les 30 secondes et qu'EC2 met 30 secondes à approvisionner un nouveau nœud, un seul nœud de surprovisionnement garantira la disponibilité d'un nœud supplémentaire, réduisant ainsi le temps de latence de planification de 30 secondes au prix d'une seule instance EC2 supplémentaire. Pour améliorer les décisions de planification zonale, surapprovisionnez un nombre de nœuds égal au nombre de zones de disponibilité de votre groupe EC2 Auto Scaling afin que le planificateur puisse sélectionner la meilleure zone pour les pods entrants.

Empêcher la réduction des expulsions

L'expulsion de certaines applications est coûteuse. L'analyse des mégadonnées, les tâches d'apprentissage automatique et les tests finiront par être terminés, mais devront être redémarrés en cas d'interruption. Le Cluster Autoscaler tentera de réduire la taille de tout nœud situé sous le scale-down-utilization-threshold, ce qui interrompra tous les pods restants sur le nœud. Cela peut être évité en veillant à ce que les pods dont l'expulsion coûte cher soient protégés par une étiquette reconnue par le Cluster Autoscaler.

Assurez-vous que :

- Coûteux à expulser, les pods ont l'annotation `cluster-autoscaler.kubernetes.io/safe-to-evict=false`

Cas d'utilisation avancés

Volumes EBS

Le stockage permanent est essentiel pour créer des applications dynamiques, telles que des bases de données ou des caches distribués. Les volumes EBS permettent ce cas d'utilisation sur Kubernetes, mais sont limités à une zone spécifique. Ces applications peuvent être hautement disponibles si elles sont réparties sur plusieurs zones de zone de disponibilité à l'aide d'un volume EBS distinct pour chaque zone de zone de disponibilité. Le Cluster Autoscaler peut ensuite équilibrer la mise à l'échelle des groupes EC2 Autoscaling.

Assurez-vous que :

- L'équilibrage des groupes de nœuds est activé en définissant `balance-similar-node-groups=true`.
- Les groupes de nœuds sont configurés avec des paramètres identiques, à l'exception des différentes zones de disponibilité et des volumes EBS.

Co-planification

Les tâches d'entraînement distribué de machine learning bénéficient de manière significative de la latence réduite des configurations de nœuds de même zone. Ces charges de travail déploient plusieurs pods dans une zone spécifique. Cela peut être réalisé en définissant l'affinité des pods pour tous les pods co-planifiés ou en utilisant l'affinité des nœuds à l'aide `topologyKey: failure-`

`domain.beta.kubernetes.io/zone` de. Le Cluster Autoscaler va ensuite redimensionner une zone spécifique pour répondre aux demandes. Vous souhaiterez peut-être allouer plusieurs groupes EC2 Auto Scaling, un par zone de disponibilité, afin de permettre le basculement sur incident pour l'ensemble de la charge de travail co-planifiée.

Assurez-vous que :

- L'équilibrage des groupes de nœuds est activé en configurant `balance-similar-node-groups=false`
- L'affinité des nœuds et/ou la préemption des pods sont utilisées lorsque les clusters incluent à la fois des groupes de nœuds régionaux et zonaux.
  - Utilisez l'affinité des nœuds pour forcer ou encourager les groupes régionaux à éviter les groupes de nœuds zonaux, et vice versa.
  - Si les modules zonaux sont programmés sur des groupes de nœuds régionaux, cela entraînera un déséquilibre de capacité pour vos modules régionaux.
  - Si vos charges de travail zonales peuvent tolérer des perturbations et des relocalisations, configurez Pod Preemption pour permettre aux pods répartis à l'échelle régionale de forcer la préemption et la replanification sur une zone moins disputée.

Accélérateurs

Certains clusters tirent parti d'accélérateurs matériels spécialisés tels que le GPU. Lors de la mise à l'échelle, le plug-in du dispositif accélérateur peut prendre plusieurs minutes pour annoncer la ressource au cluster. Le Cluster Autoscaler a simulé que ce nœud serait doté de l'accélérateur, mais tant que l'accélérateur ne sera pas prêt et ne mettra pas à jour les ressources disponibles du nœud, les pods en attente ne peuvent pas être planifiés sur le nœud. Cela peut entraîner une Répétition inutile de la montée en puissance.

En outre, les nœuds dotés d'accélérateurs et d'une utilisation élevée du processeur ou de la mémoire ne seront pas pris en compte pour la réduction, même si l'accélérateur n'est pas utilisé. Ce comportement peut être coûteux en raison du coût relatif des accélérateurs. Au lieu de cela, le Cluster Autoscaler peut appliquer des règles spéciales pour considérer les nœuds à réduire s'ils ont des accélérateurs inoccupés.

Pour garantir le comportement correct dans ces cas, vous pouvez configurer le kubelet sur vos nœuds d'accélérateur pour étiqueter le nœud avant qu'il ne rejoigne le cluster. Le Cluster Autoscaler utilisera ce sélecteur d'étiquette pour déclencher le comportement optimisé de l'accélérateur.

Assurez-vous que :

- Le Kubelet pour les nœuds GPU est configuré avec `--node-labels k8s.amazonaws.com/accelerator=$ACCELERATOR_TYPE`
- Les nœuds dotés d'accélérateurs respectent la même règle de propriétés de planification mentionnée ci-dessus.

Échelle à partir de 0

Cluster Autoscaler est capable de redimensionner les groupes de nœuds jusqu'à zéro, ce qui peut permettre de réaliser d'importantes économies. Il détecte les ressources du processeur, de la mémoire et du processeur graphique d'un Auto Scaling Group en inspectant les informations InstanceType spécifiées dans son LaunchConfiguration ou LaunchTemplate. Certains pods nécessitent des ressources supplémentaires, telles que `WindowsENI` `PrivateIPv4Address` NodeSelectors ou des souillures spécifiques, qui ne peuvent pas être découvertes à partir du LaunchConfiguration. Le Cluster Autoscaler peut prendre en compte ces facteurs en les découvrant à partir des balises du groupe EC2 Auto Scaling. Par exemple :

```
Key: k8s.io/cluster-autoscaler/node-template/resources/$RESOURCE_NAME
Value: 5
Key: k8s.io/cluster-autoscaler/node-template/label/$LABEL_KEY
Value: $LABEL_VALUE
Key: k8s.io/cluster-autoscaler/node-template/taint/$TAINT_KEY
Value: NoSchedule
```

> **Note**
>
> N'oubliez pas que lorsque vous passez à zéro, votre capacité est renvoyée à EC2 et peut être indisponible à l'avenir.

Paramètres supplémentaires

Il existe de nombreuses options de configuration qui peuvent être utilisées pour régler le comportement et les performances du Cluster Autoscaler. La liste complète des paramètres est disponible sur GitHub.

Paramètre	Description	Par défaut
intervalle de numérisation	À quelle fréquence le cluster est-il réévalué en vue d'une augmentation ou d'une réduction	10 secondes
max-empty-bulk-delete	Nombre maximum de nœuds vides pouvant être supprimés simultanément.	10
scale-down-delay-after-ajouter	Combien de temps après l'augmentation de cette réduction reprendra l'évaluat ion ?	10 minutes
scale-down-delay-after-supp rimer	Combien de temps après la suppression du nœud, l'évaluation à l'échelle réduite reprend, valeur par défaut : scan-interval	intervalle de numérisation
scale-down-delay-after-échec	Combien de temps après l'échec de la réduction, cette évaluation reprendra ?	3 minutes
scale-down-unneeded-time	Pendant combien de temps un nœud devrait-il être inutile avant de pouvoir être réduit	10 minutes
scale-down-unready-time	Pendant combien de temps un nœud non prêt devrait-il être inutile avant de pouvoir être réduit	20 minutes
scale-down-utilization-threshold	Niveau d'utilisation du nœud, défini comme la somme des ressources demandées	0.5

Paramètre	Description	Par défaut
	divisée par la capacité, en dessous duquel un nœud peut être envisagé pour une réduction	
scale-down-non-empty-décompte des candidats	Nombre maximum de nœuds non vides considérés au cours d'une itération comme candidats à une réduction avec drain. Une valeur inférieure signifie une meilleure réactivité de l'autorité de certification, mais une réduction plus lente de la latence est possible. Une valeur plus élevée peut affecter les performances de l'autorité de certification avec de grands clusters (des centaines de nœuds). Réglez cette valeur sur une valeur non positive pour désactiver cette heuristique. CA ne limitera pas le nombre de nœuds pris en compte. »	30

Paramètre	Description	Par défaut
scale-down-candidates-pool-ratio	Un ratio de nœuds considérés comme des candidats non vides supplémentaires à réduire lorsque certains candidats de l'itération précédente ne sont plus valides. Une valeur inférieure signifie une meilleure réactivité de l'autorité de certification, mais une réduction plus lente de la latence est possible. Une valeur plus élevée peut affecter les performances de l'autorité de certification avec de grands clusters (des centaines de nœuds). Réglez cette valeur sur 1.0 pour désactiver cette heuristique. CA considérera tous les nœuds comme candidats supplémentaires.	0.1

Paramètre	Description	Par défaut
scale-down-candidates-pool-nombre minimum	Nombre minimal de nœuds considérés comme des candidats non vides supplémentaires à réduire lorsque certains candidats de l'itération précédente ne sont plus valides. Lors du calcul de la taille du pool pour les candidats supplémentaires, nous prenons `max(#nodes * scale-down-candidates-pool-ratio, scale-down-candidates-pool-min-count)`	50

Ressources supplémentaires

Cette page contient une liste de présentations et de démonstrations de Cluster Autoscaler. Si vous souhaitez ajouter une présentation ou une démonstration ici, veuillez envoyer une pull request.

Présentation/Démo	Présentateurs
Autoscaling et optimisation des coûts sur Kubernetes : de 0 à 100	Guy Templeton, Skyscanner et Jiaxin Shan, Amazon
Approfondissement de la mise à l'échelle automatique du SIG	Maciek Pytel et Marcin Wielgus

Références

- https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md
- https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/aws/README.md
- https://github.com/aws/sélecteur d'instance amazon-ec2

- https://github.com/aws/aws-node-termination-handler

Meilleures pratiques en matière de fiabilité

Cette section fournit des conseils pour rendre les charges de travail exécutées sur EKS résilientes et hautement disponibles.

Comment utiliser ce guide

Ce guide est destiné aux développeurs et aux architectes qui souhaitent développer et exploiter des services hautement disponibles et tolérants aux pannes dans EKS. Le guide est organisé en différents domaines thématiques pour en faciliter la consommation. Chaque rubrique commence par un bref aperçu, suivi d'une liste de recommandations et de bonnes pratiques pour la fiabilité de vos clusters EKS.

Introduction

Les meilleures pratiques en matière de fiabilité pour EKS ont été regroupées sous les rubriques suivantes :

- Applications
- Plan de contrôle
- Plan de données

Qu'est-ce qui rend un système fiable ? Si un système peut fonctionner de manière cohérente et répondre aux demandes malgré les changements survenus dans son environnement au fil du temps, il peut être qualifié de fiable. Pour ce faire, le système doit détecter les défaillances, se réparer automatiquement et être capable d'évoluer en fonction de la demande.

Les clients peuvent utiliser Kubernetes comme base pour exploiter de manière fiable des applications et des services critiques. Mais outre l'intégration des principes de conception d'applications basés sur des conteneurs, l'exécution fiable des charges de travail nécessite également une infrastructure fiable. Dans Kubernetes, l'infrastructure comprend le plan de contrôle et le plan de données.

EKS fournit un plan de contrôle Kubernetes de niveau production conçu pour être hautement disponible et tolérant aux pannes.

Dans EKS, AWS est responsable de la fiabilité du plan de contrôle Kubernetes. EKS exécute le plan de contrôle Kubernetes dans trois zones de disponibilité d'une région AWS. Il gère automatiquement la disponibilité et l'évolutivité des serveurs d'API Kubernetes et du cluster etcd.

La responsabilité de la fiabilité du plan de données est partagée entre vous, le client et AWS. EKS propose quatre options de nœuds de travail pour déployer le plan de données Kubernetes.

Le mode automatique EKS, qui est l'option la plus gérée, gère le provisionnement, le dimensionnement et les mises à jour du plan de données, tout en fournissant des fonctionnalités de calcul, de mise en réseau et de stockage gérées. Le mode AMIs automatique est fréquemment publié et les clusters sont automatiquement mis à jour vers la dernière AMI pour déployer les correctifs CVE et les correctifs de sécurité. Vous pouvez contrôler le moment où cela se produit en configurant les commandes d'interruption sur votre mode automatique NodePools.

Fargate gère le provisionnement et le dimensionnement du plan de données en exécutant un pod par nœud. La troisième option, les groupes de nœuds gérés, gère le provisionnement et les mises à jour du plan de données. Enfin, les nœuds autogérés constituent l'option la moins gérée pour le plan de données. Plus vous utilisez de plans de données gérés par AWS, moins vous êtes responsable.

Les groupes de nœuds gérés automatisent le provisionnement et la gestion du cycle de vie des nœuds EC2. Vous pouvez utiliser l'API EKS (à l'aide de la console EKS, de l'API AWS, de la CLI AWSCloudFormation, de Terraform ou`eksctl`) pour créer, dimensionner et mettre à niveau des nœuds gérés. Les nœuds gérés exécutent des instances Amazon Linux 2 EC2 optimisées pour EKS sur votre compte, et vous pouvez installer des packages logiciels personnalisés en activant l'accès SSH. Lorsque vous provisionnez des nœuds gérés, ils s'exécutent dans le cadre d'un groupe Auto Scaling géré par EKS qui peut couvrir plusieurs zones de disponibilité ; vous contrôlez cela via les sous-réseaux que vous fournissez lors de la création de nœuds gérés. EKS étiquette également automatiquement les nœuds gérés afin qu'ils puissent être utilisés avec Cluster Autoscaler.

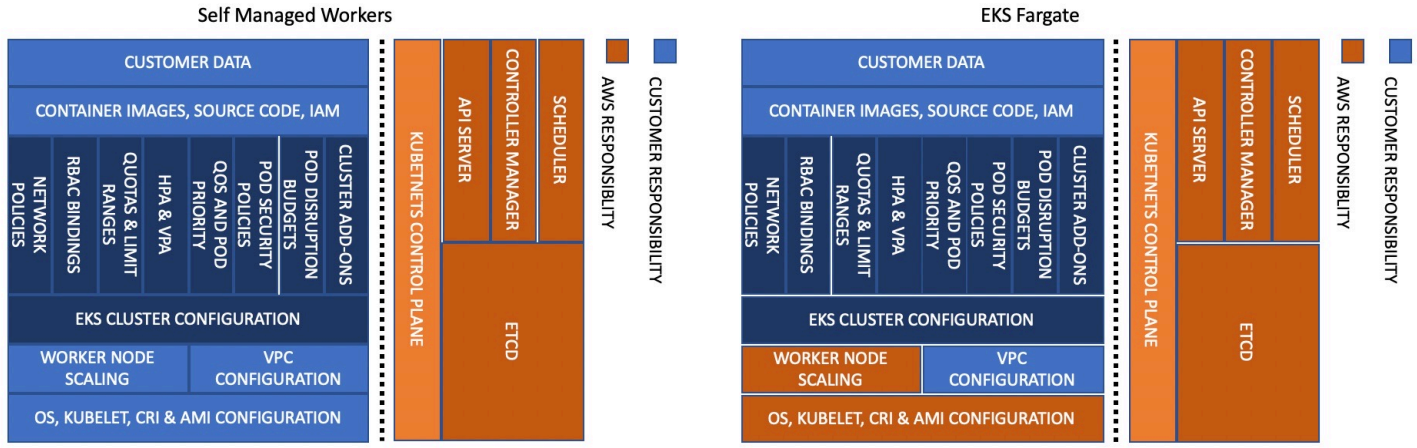
Amazon EKS suit le modèle de responsabilité partagée CVEs et les correctifs de sécurité sur les groupes de nœuds gérés. Dans la mesure où les nœuds gérés exécutent le système optimisé pour Amazon EKS, AMIs Amazon EKS est chargé de créer des versions corrigées de ceux-ci AMIs lors de la correction de bogues. Toutefois, vous êtes responsable du déploiement de ces versions d'AMI corrigées vers vos groupes de nœuds gérés.

EKS gère également la mise à jour des nœuds, bien que vous deviez lancer le processus de mise à jour. Le processus de mise à jour du nœud géré est expliqué dans la documentation EKS.

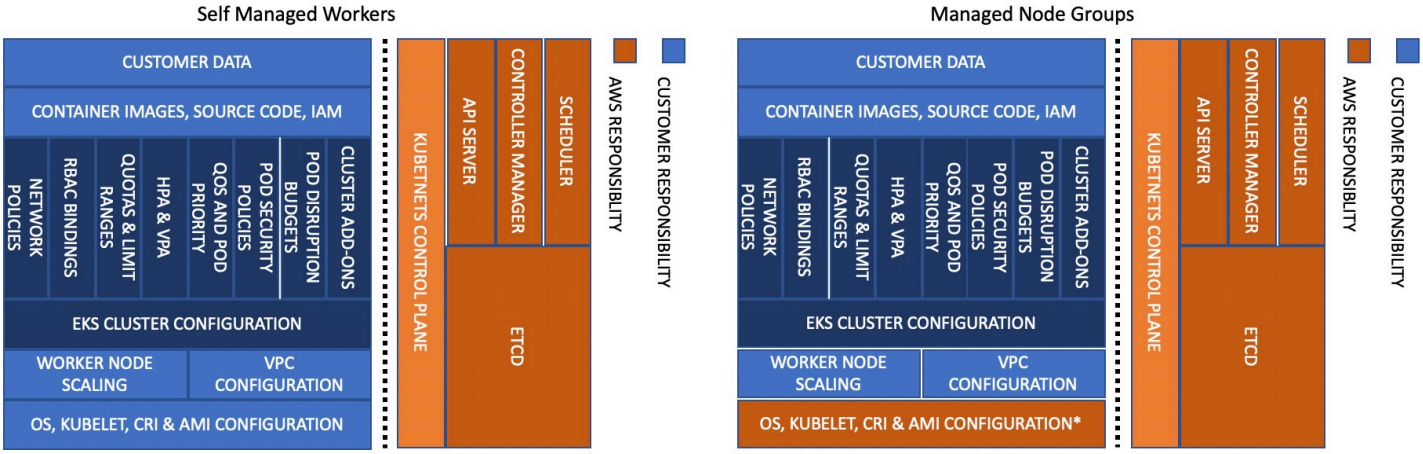
Si vous exécutez des nœuds autogérés, vous pouvez utiliser l'AMI Linux optimisée pour Amazon EKS pour créer des nœuds de travail. Vous êtes responsable de l'application des correctifs et de la

mise à niveau de l'AMI et des nœuds. Il est recommandé d'utiliser ou d'utiliser l'`eksctl`infrastructure en tant qu'outils de code pour provisionner des nœuds autogérés, car cela vous permettra de mettre à niveau facilement les nœuds autogérés. CloudFormation Envisagez de migrer vers de nouveaux nœuds lorsque vous mettez à jour des nœuds de travail, car le processus de migration altère l'ancien groupe de nœuds `NoSchedule` et épuise les nœuds une fois qu'une nouvelle pile est prête à accepter la charge de travail du pod existant. Toutefois, vous pouvez également effectuer une mise à niveau sur place des nœuds autogérés.

Modèle de responsabilité partagée - Fargate



Modèle de responsabilité partagée - MNG



Ce guide inclut un ensemble de recommandations que vous pouvez utiliser pour améliorer la fiabilité de votre plan de données EKS, des composants principaux de Kubernetes et de vos applications.

Commentaires

Ce guide est publié GitHub pour recueillir les commentaires et suggestions directs de l'ensemble de la EKS/Kubernetes communauté. Si vous avez une bonne pratique que vous pensez que nous devrions inclure dans le guide, veuillez signaler un problème ou soumettre un PR dans le GitHub référentiel. Nous avons l'intention de mettre à jour le guide régulièrement à mesure que de nouvelles fonctionnalités sont ajoutées au service ou lorsqu'une nouvelle bonne pratique évolue.

Exécution d'applications à haute disponibilité

Vos clients s'attendent à ce que votre application soit toujours disponible, y compris lorsque vous apportez des modifications, en particulier lors de pics de trafic. Une architecture évolutive et résiliente permet à vos applications et services de fonctionner sans interruption, ce qui garantit la satisfaction de vos utilisateurs. Une infrastructure évolutive croît et se réduit en fonction des besoins de l'entreprise. L'élimination des points de défaillance uniques est une étape essentielle pour améliorer la disponibilité d'une application et la rendre résiliente.

Avec Kubernetes, vous pouvez exploiter vos applications et les exécuter de manière hautement disponible et résiliente. Sa gestion déclarative garantit qu'une fois l'application configurée, Kubernetes essaiera en permanence de faire correspondre l'état actuel à l'état souhaité.

Recommandations

Configurer les budgets d'interruption de service des pods

Les budgets d'interruption des modules sont utilisés pour limiter le nombre d'interruptions simultanées auxquelles une application sera confrontée. Ils doivent être configurés pour les charges de travail s'il est important de toujours avoir une partie de cette charge de travail disponible. EKS Auto Mode, Karpenter et Cluster Autoscaler connaissent et respectent les budgets d'interruption des modules configurés lors de la réduction des effectifs. EKS Auto Mode, Karpenter et Managed Node Groups respectent également les budgets d'interruption des pods lors de la mise à jour des nœuds

Évitez d'utiliser des Singleton Pods

Si l'ensemble de votre application s'exécute dans un seul Pod, votre application ne sera pas disponible en cas de fermeture de ce Pod. Au lieu de déployer des applications à l'aide de pods individuels, créez des déploiements. Si un pod créé par un déploiement échoue ou est arrêté, le

contrôleur de déploiement démarrera un nouveau pod pour garantir que le nombre spécifié de répliques de pods fonctionne toujours.

Exécutez plusieurs répliques

L'exécution de plusieurs répliques (pods) d'une application à l'aide d'un déploiement permet à celle-ci de fonctionner de manière hautement disponible. Si l'une des répliques tombe en panne, les répliques restantes fonctionneront toujours, mais à capacité réduite, jusqu'à ce que Kubernetes crée un autre pod pour compenser la perte. En outre, vous pouvez utiliser le Horizontal Pod Autoscaler pour dimensionner automatiquement les répliques en fonction de la charge de travail.

Planifier des répliques sur les nœuds

L'exécution de plusieurs répliques ne sera pas très utile si toutes les répliques s'exécutent sur le même nœud et que le nœud devient indisponible. Envisagez d'utiliser des contraintes d'anti-affinité ou de propagation de la topologie des pods pour répartir les répliques d'un déploiement sur plusieurs nœuds de travail.

Vous pouvez encore améliorer la fiabilité d'une application classique en l'exécutant sur plusieurs applications AZs.

Utilisation des règles anti-affinité du Pod

Le manifeste ci-dessous indique au planificateur Kubernetes de préférer placer les pods sur des nœuds distincts et. AZs Il ne nécessite pas de nœuds ou d'AZ distincts, car si c'était le cas, Kubernetes ne sera pas en mesure de planifier des pods une fois qu'un pod sera exécuté dans chaque AZ. Si votre application ne nécessite que trois répliques, vous pouvez utiliser `requiredDuringSchedulingIgnoredDuringExecution` fortopologyKey: `topology.kubernetes.io/zone`, et le planificateur Kubernetes ne planifiera pas deux pods dans la même zone de zones.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spread-host-az
  labels:
    app: web-server
spec:
  replicas: 4
  selector:
    matchLabels:
```

```
      app: web-server
  template:
    metadata:
      labels:
        app: web-server
    spec:
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - web-server
              topologyKey: topology.kubernetes.io/zone
            weight: 100
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - web-server
              topologyKey: kubernetes.io/hostname
            weight: 99
      containers:
      - name: web-app
        image: nginx:1.16-alpine
```

Utilisation des contraintes de propagation de la topologie Pod

À l'instar des règles anti-affinité des pods, les contraintes de propagation de la topologie des pods vous permettent de rendre votre application disponible sur différents domaines de défaillance (ou topologie) tels que les hôtes ou. AZs Cette approche fonctionne très bien lorsque vous essayez de garantir la tolérance aux pannes ainsi que la disponibilité en disposant de plusieurs répliques dans chacun des différents domaines topologiques. Les règles d'anti-affinité des pods, en revanche, peuvent facilement produire un résultat lorsque vous n'avez qu'une seule réplique dans un domaine topologique, car les pods présentant une anti-affinité les uns envers les autres ont un effet répulsif. Dans de tels cas, une seule réplique sur un nœud dédié n'est pas idéale en termes de tolérance aux pannes et ne constitue pas une bonne utilisation des ressources. Les contraintes d'étalement de

topologie vous permettent de mieux contrôler l'étalement ou la distribution que le planificateur doit essayer d'appliquer dans les domaines topologiques. Voici quelques propriétés importantes à utiliser dans cette approche :

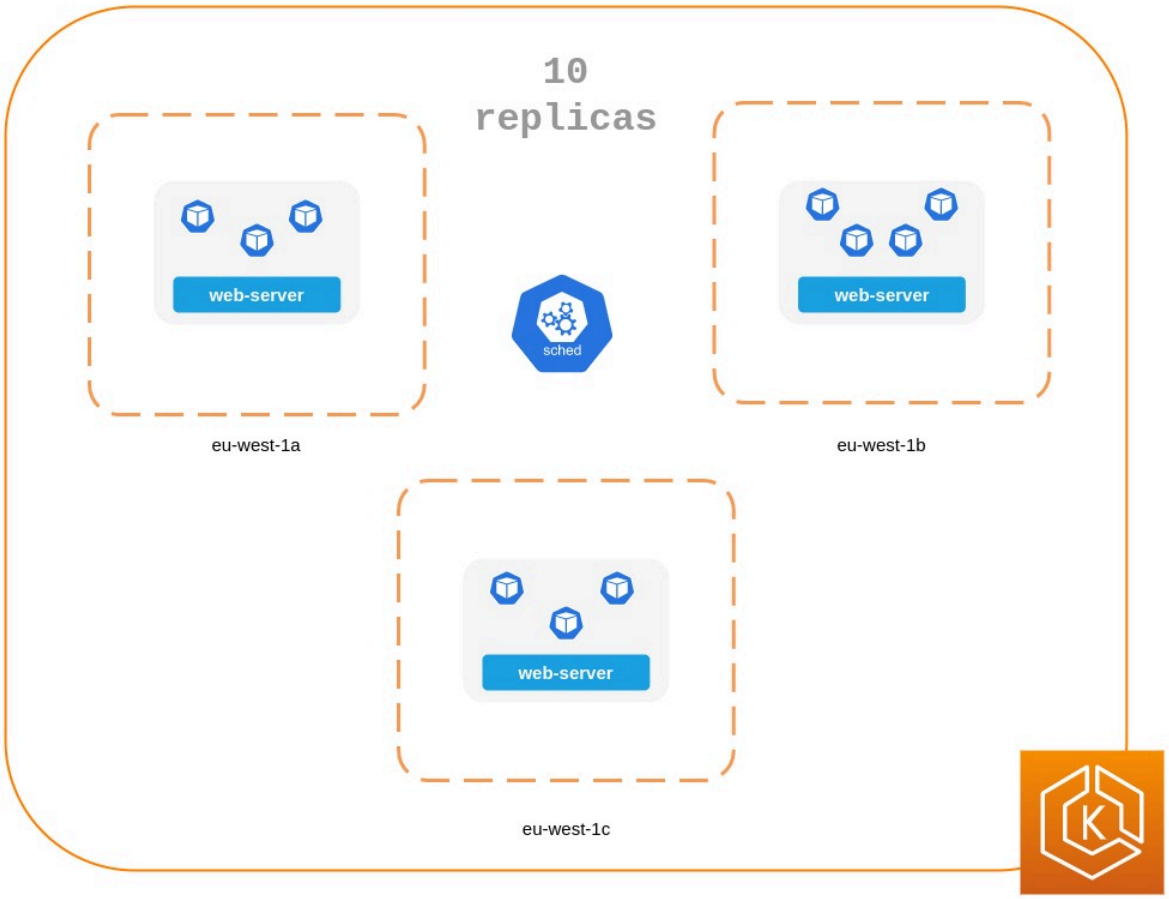
1. Le `maxSkew` est utilisé pour contrôler ou déterminer le point maximal auquel les choses peuvent être inégales entre les domaines topologiques. Par exemple, si une application possède 10 répliques et est déployée sur 3 AZs, vous ne pouvez pas obtenir une répartition uniforme, mais vous pouvez influencer le degré d'inégalité de la distribution. Dans ce cas, il `maxSkew` peut être compris entre 1 et 10. Une valeur de 1 signifie que vous pouvez vous retrouver avec un spread similaire `4,3,3` `3,4,3` ou `3,3,4` supérieur à 3 AZs. En revanche, une valeur de 10 signifie que vous pouvez vous retrouver avec un spread similaire `10,0,0` `0,10,0` ou supérieur `0,0,10` à 3 AZs.
2. `topologyKey`Il s'agit d'une clé pour l'une des étiquettes de nœuds et définit le type de domaine topologique à utiliser pour la distribution des pods. Par exemple, un spread zonal comporterait la paire clé-valeur suivante :

   ```
   topologyKey: "topology.kubernetes.io/zone"
   ```

3. La `whenUnsatisfiable` propriété est utilisée pour déterminer comment vous souhaitez que le planificateur réagisse si les contraintes souhaitées ne peuvent pas être satisfaites.
4. Le `labelSelector` est utilisé pour trouver les pods correspondants afin que le planificateur puisse en prendre connaissance lorsqu'il décide où placer les pods conformément aux contraintes que vous spécifiez.

En plus de ce qui précède, il existe d'autres champs que vous pouvez consulter plus en détail dans la documentation de Kubernetes.

La topologie du pod répartit les contraintes sur 3 AZs




```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spread-host-az
  labels:
    app: web-server
spec:
  replicas: 10
  selector:
    matchLabels:
      app: web-server
  template:
    metadata:
      labels:
        app: web-server
    spec:
      topologySpreadConstraints:
```

```
      - maxSkew: 1
        topologyKey: "topology.kubernetes.io/zone"
        whenUnsatisfiable: ScheduleAnyway
        labelSelector:
          matchLabels:
            app: express-test
      containers:
      - name: web-app
        image: nginx:1.16-alpine
```

Exécuter le serveur Kubernetes Metrics

Installez le serveur de métriques Kubernetes pour vous aider à faire évoluer vos applications. Les modules complémentaires Kubernetes Autoscaler tels que HPA et VPA doivent suivre les métriques des applications pour les dimensionner. Le serveur de mesures collecte des métriques de ressources qui peuvent être utilisées pour prendre des décisions de dimensionnement. Les métriques sont collectées à partir de kubelets et diffusées au format Metrics API.

Le serveur de métriques ne conserve aucune donnée et ne constitue pas une solution de surveillance. Son objectif est d'exposer les métriques d'utilisation du processeur et de la mémoire à d'autres systèmes. Si vous souhaitez suivre l'état de votre application au fil du temps, vous avez besoin d'un outil de surveillance tel que Prometheus ou Amazon. CloudWatch

Suivez la documentation EKS pour installer le serveur de mesures dans votre cluster EKS.

Autoscaler à nacelle horizontale (HPA)

HPA peut adapter automatiquement votre application en fonction de la demande et vous aider à éviter d'avoir un impact sur vos clients pendant les pics de trafic. Il est implémenté sous la forme d'une boucle de contrôle dans Kubernetes qui interroge périodiquement les métriques APIs qui fournissent des métriques de ressources.

HPA peut récupérer des métriques à partir des éléments suivants APIs : 1. `metrics.k8s.io`également connue sous le nom d'API Resource Metrics — Fournit l'utilisation du processeur et de la mémoire pour les pods 2. `custom.metrics.k8s.io` — Fournit des métriques provenant d'autres collecteurs de métriques tels que Prometheus ; ces métriques sont internes à votre cluster Kubernetes. 3. `external.metrics.k8s.io` — Fournit des métriques externes à votre cluster Kubernetes (par exemple, profondeur de file d'attente SQS, latence ELB).

Vous devez utiliser l'un de ces trois indicateurs APIs pour fournir la métrique permettant de dimensionner votre application.

Mise à l'échelle des applications en fonction de métriques personnalisées ou externes

Vous pouvez utiliser des métriques personnalisées ou externes pour adapter votre application à des métriques autres que l'utilisation du processeur ou de la mémoire. Les serveurs d'API Custom Metrics fournissent l'`custom-metrics.k8s.io`API que HPA peut utiliser pour dimensionner automatiquement les applications.

Vous pouvez utiliser l'adaptateur Prometheus pour Kubernetes Metrics pour collecter des métriques provenant de Prometheus et les APIs utiliser avec le HPA. Dans ce cas, l'adaptateur Prometheus exposera les métriques Prometheus au format de l'API Metrics.

Une fois que vous avez déployé l'adaptateur Prometheus, vous pouvez demander des métriques personnalisées à l'aide de kubectl. `kubectl get –raw /apis/custom.metrics.k8s.io/v1beta1/`

Comme son nom l'indique, les métriques externes permettent à Horizontal Pod Autoscaler de dimensionner les déploiements à l'aide de métriques externes au cluster Kubernetes. Par exemple, dans le cadre du traitement des charges de travail par lots, il est courant d'ajuster le nombre de répliques en fonction du nombre de tâches en cours dans une file d'attente SQS.

Pour dimensionner automatiquement les charges de travail Kubernetes, vous pouvez utiliser KEDA (Kubernetes Event-driven Autoscaling), un projet open source capable de piloter le dimensionnement des conteneurs en fonction d'un certain nombre d'événements personnalisés. Ce blog AWS explique comment utiliser Amazon Managed Service for Prometheus pour l'auto-scaling de la charge de travail Kubernetes.

Autoscaler à nacelle verticale (VPA)

Le VPA ajuste automatiquement la réserve de processeur et de mémoire pour vos pods afin de vous aider à « dimensionner correctement » vos applications. Pour les applications qui doivent être mises à l'échelle verticalement, ce qui est fait en augmentant l'allocation des ressources, vous pouvez utiliser le VPA pour redimensionner automatiquement les répliques de Pod ou fournir des recommandations de dimensionnement.

Votre application peut devenir temporairement indisponible si le VPA doit la redimensionner, car l'implémentation actuelle du VPA n'effectue aucun ajustement sur place des pods ; elle recréera plutôt le pod qui doit être redimensionné.

La documentation EKS inclut une procédure pas à pas pour configurer le VPA.

Le projet Fairwinds Goldilocks fournit un tableau de bord permettant de visualiser les recommandations VPA pour les demandes et les limites du processeur et de la mémoire. Son mode de mise à jour VPA vous permet de dimensionner automatiquement les pods en fonction des recommandations de la VPA.

Mise à jour des applications

Les applications modernes nécessitent une innovation rapide associée à un haut degré de stabilité et de disponibilité. Kubernetes vous fournit les outils nécessaires pour mettre à jour vos applications en permanence sans perturber vos clients.

Examinons certaines des meilleures pratiques qui permettent de déployer rapidement des modifications sans sacrifier la disponibilité.

Disposer d'un mécanisme pour effectuer des annulations

Le fait d'avoir un bouton d'annulation peut éviter les catastrophes. Il est recommandé de tester les déploiements dans un environnement inférieur distinct (environnement de test ou de développement) avant de mettre à jour le cluster de production. L'utilisation d'un pipeline CI/CD peut vous aider à automatiser et à tester les déploiements. Avec un pipeline de déploiement continu, vous pouvez rapidement revenir à l'ancienne version si la mise à niveau s'avère défectueuse.

Vous pouvez utiliser les déploiements pour mettre à jour une application en cours d'exécution. Cela se fait généralement en mettant à jour l'image du conteneur. Vous pouvez l'utiliser `kubectl` pour mettre à jour un déploiement comme suit :

```
kubectl --record deployment.apps/nginx-deployment set image nginx-deployment
 nginx=nginx:1.16.1
```

L'`--record`argument enregistre les modifications apportées au déploiement et vous aide si vous devez effectuer une restauration. `kubectl rollout history deployment`affiche les modifications enregistrées apportées aux déploiements dans votre cluster. Vous pouvez annuler une modification en utilisant`kubectl rollout undo deployment <DEPLOYMENT_NAME>`.

Par défaut, lorsque vous mettez à jour un déploiement qui nécessite une recréation des modules, le déploiement effectue une mise à jour continue. En d'autres termes, Kubernetes ne mettra à jour qu'une partie des pods en cours d'exécution dans un déploiement et non tous les pods en même temps. Vous pouvez contrôler la manière dont Kubernetes effectue les mises à jour progressives via les propriétés. `RollingUpdateStrategy`

Lorsque vous effectuez une mise à jour continue d'un déploiement, vous pouvez utiliser la `Max Unavailable`propriété pour spécifier le nombre maximum de pods qui peuvent être indisponibles pendant la mise à jour. La `Max Surge` propriété Deployment vous permet de définir le nombre maximum de pods pouvant être créés par rapport au nombre souhaité de pods.

Envisagez `max unavailable` de procéder à des ajustements pour vous assurer qu'un déploiement ne perturbe pas vos clients. Par exemple, Kubernetes définit 25 % `max unavailable` par défaut, ce qui signifie que si vous avez 100 pods, il se peut que seuls 75 pods fonctionnent activement pendant un déploiement. Si votre application a besoin d'un minimum de 80 pods, ce déploiement peut être perturbateur. Au lieu de cela, vous pouvez `max unavailable` régler sur 20 % pour garantir qu'il y ait au moins 80 pods fonctionnels tout au long du déploiement.

Utiliser les blue/green déploiements

Les changements sont intrinsèquement risqués, mais ceux qui ne peuvent pas être annulés peuvent être potentiellement catastrophiques. Les procédures de modification qui vous permettent de remonter le temps de manière efficace par le biais d'un rollback rendent les améliorations et les expérimentations plus sûres. Blue/green les déploiements vous offrent une méthode pour retirer rapidement les modifications en cas de problème. Dans cette stratégie de déploiement, vous créez un environnement pour la nouvelle version. Cet environnement est identique à la version actuelle de l'application mise à jour. Une fois le nouvel environnement configuré, le trafic est acheminé vers le nouvel environnement. Si la nouvelle version produit les résultats souhaités sans générer d'erreurs, l'ancien environnement est supprimé. Dans le cas contraire, le trafic est rétabli à l'ancienne version.

Vous pouvez effectuer blue/green des déploiements dans Kubernetes en créant un nouveau déploiement identique au déploiement de la version existante. Une fois que vous avez vérifié que les pods du nouveau déploiement fonctionnent sans erreur, vous pouvez commencer à envoyer du trafic vers le nouveau déploiement en modifiant les `selector` spécifications du service qui achemine le trafic vers les pods de votre application.

De nombreux outils d'intégration continue tels que Flux, Jenkins et Spinnaker vous permettent d'automatiser les déploiements. blue/green Le blog AWS Containers inclut une présentation détaillée de l'utilisation d'AWS Load Balancer Controller : utilisation d'AWS Load Balancer Controller blue/green pour le déploiement, le déploiement Canary et les tests A/B

Utiliser les déploiements Canary

Les déploiements Canary sont une variante des blue/green déploiements qui peuvent réduire de manière significative les risques liés aux modifications. Dans cette stratégie de déploiement, vous

créez un nouveau déploiement avec moins de pods en plus de votre ancien déploiement, et vous redirigez un faible pourcentage du trafic vers le nouveau déploiement. Si les indicateurs indiquent que la nouvelle version fonctionne aussi bien ou mieux que la version existante, vous augmentez progressivement le trafic vers le nouveau déploiement tout en le dimensionnant jusqu'à ce que tout le trafic soit redirigé vers le nouveau déploiement. En cas de problème, vous pouvez acheminer tout le trafic vers l'ancien déploiement et arrêter d'envoyer du trafic vers le nouveau déploiement.

Bien que Kubernetes ne propose aucune méthode native pour effectuer des déploiements Canary, vous pouvez utiliser des outils tels que Flagger avec Istio.

Bilans de santé et auto-guérison

Aucun logiciel n'est exempt de bogues, mais Kubernetes peut vous aider à minimiser l'impact des défaillances logicielles. Dans le passé, si une application tombait en panne, quelqu'un devait remédier à la situation en redémarrant l'application manuellement. Kubernetes vous permet de détecter les défaillances logicielles de vos pods et de les remplacer automatiquement par de nouvelles répliques. Kubernetes vous permet de surveiller l'état de santé de vos applications et de remplacer automatiquement les instances défectueuses.

Kubernetes prend en charge trois types de tests de santé :

1. Sonde Liveness
2. Sonde de démarrage (prise en charge dans Kubernetes version 1.16+)
3. Sonde de préparation

Kubelet, l'agent Kubernetes, est chargé d'exécuter toutes les vérifications mentionnées ci-dessus. Kubelet peut vérifier l'état de santé d'un pod de trois manières : kubelet peut soit exécuter une commande shell dans le conteneur d'un pod, soit envoyer une requête HTTP GET à son conteneur, soit ouvrir un socket TCP sur un port spécifié.

Si vous choisissez une sonde `exec` basée, qui exécute un script shell dans un conteneur, assurez-vous que la commande shell se termine avant que la `timeoutSeconds` valeur n'expire. Dans le cas contraire, votre nœud sera soumis à des `<defunct>` processus, ce qui entraînera une défaillance du nœud.

Recommandations

Utilisez Liveness Probe pour éliminer les gousses malsaines

La sonde Liveness peut détecter les situations de blocage dans lesquelles le processus continue de s'exécuter, mais où l'application ne répond plus. Par exemple, si vous exécutez un service Web qui écoute sur le port 80, vous pouvez configurer une sonde Liveness pour envoyer une requête HTTP GET sur le port 80 du Pod. Kubelet envoie périodiquement une requête GET au Pod et attend une réponse ; si le Pod répond entre 200 et 399, le kubelet considère que le Pod est en bonne santé ; sinon, le Pod sera marqué comme non sain. Si un Pod échoue continuellement aux contrôles de santé, le kubelet l'arrêtera.

Vous pouvez l'utiliser `initialDelaySeconds` pour retarder la première sonde.

Lorsque vous utilisez la Liveness Probe, assurez-vous que votre application ne se retrouve pas dans une situation dans laquelle tous les Pods échouent simultanément à la Liveness Probe, car Kubernetes essaiera de remplacer tous vos Pods, ce qui mettra votre application hors ligne. En outre, Kubernetes continuera à créer de nouveaux pods qui échoueront également aux Liveness Probes, ce qui alourdira inutilement le plan de contrôle. Évitez de configurer la Liveness Probe pour qu'elle dépende d'un facteur externe à votre Pod, par exemple une base de données externe. En d'autres termes, une external-to-your-Pod base de données non réactive ne devrait pas faire échouer vos Pods à leurs Liveness Probes.

Dans son article LIVENESS PROBES ARE DANGEROUS, Sandor Szücs décrit les problèmes qui peuvent être causés par des sondes mal configurées.

Utilisez Startup Probe pour les applications dont le démarrage prend plus de temps

Lorsque votre application a besoin de plus de temps pour démarrer, vous pouvez utiliser la sonde de démarrage pour retarder la sonde de réactivité et de disponibilité. Par exemple, une application Java qui a besoin d'hydrater le cache d'une base de données peut avoir besoin de deux minutes pour être pleinement fonctionnelle. Toute sonde de vivacité ou de préparation jusqu'à ce qu'elle soit complètement fonctionnelle peut échouer. La configuration d'une sonde de démarrage permettra à l'application Java de retrouver son intégrité avant l'exécution de Liveness ou Readiness Probe.

Jusqu'à ce que la sonde de démarrage réussisse, toutes les autres sondes sont désactivées. Vous pouvez définir la durée maximale pendant laquelle Kubernetes doit attendre le démarrage de l'application. Si, après la durée maximale configurée, le pod échoue toujours aux sondes de démarrage, il sera arrêté et un nouveau pod sera créé.

La Startup Probe est similaire à la Liveness Probe : en cas d'échec, le Pod est recréé. Comme Ricardo A. l'explique dans son article Fantastic Probes And How To Configure Them, les sondes de démarrage doivent être utilisées lorsque l'heure de démarrage d'une application est imprévisible. Si vous savez que votre application a besoin de dix secondes pour démarrer, vous devez plutôt utiliser Liveness/Readiness Probe. `initialDelaySeconds`

Utiliser la sonde Readiness Probe pour détecter une indisponibilité partielle

Alors que la sonde Liveness détecte les défaillances d'une application qui sont résolues en arrêtant le Pod (d'où le redémarrage de l'application), Readiness Probe détecte les situations dans lesquelles l'application peut être temporairement indisponible. Dans ces situations, l'application peut ne plus répondre temporairement ; toutefois, elle devrait être rétablie une fois cette opération terminée.

Par exemple, lors d' I/O opérations intensives sur le disque, les applications peuvent être temporairement indisponibles pour traiter les demandes. Dans ce cas, la résiliation du Pod de l'application n'est pas une solution ; dans le même temps, les demandes supplémentaires envoyées au Pod peuvent échouer.

Vous pouvez utiliser la sonde Readiness Probe pour détecter une indisponibilité temporaire de votre application et arrêter d'envoyer des demandes à son Pod jusqu'à ce qu'elle redevienne fonctionnelle. Contrairement à Liveness Probe, où une défaillance entraînerait une recréation de Pod, un échec de Readiness Probe signifierait que Pod ne recevra aucun trafic en provenance du service Kubernetes. Lorsque la sonde de disponibilité aboutit, Pod recommence à recevoir du trafic en provenance du service.

Tout comme pour la Liveness Probe, évitez de configurer des Readiness Probes qui dépendent d'une ressource externe au Pod (telle qu'une base de données). Voici un scénario dans lequel un Readiness mal configuré peut rendre l'application non fonctionnelle : si la sonde de disponibilité d'un pod échoue alors que la base de données de l'application est inaccessible, les autres répliques du pod échoueront également simultanément car elles partagent les mêmes critères de contrôle de santé. En configurant la sonde de cette manière, chaque fois que la base de données n'est pas disponible, les sondes de préparation du pod échoueront et Kubernetes cessera d'envoyer du trafic à tous les pods.

L'un des effets secondaires de l'utilisation des sondes de préparation est qu'elles peuvent augmenter le temps nécessaire à la mise à jour des déploiements. Les nouvelles répliques ne recevront pas de trafic tant que les sondes de préparation ne seront pas concluantes ; d'ici là, les anciennes répliques continueront à recevoir du trafic.

Gérer les perturbations

Les pods ont une durée de vie limitée. Même si vous avez des pods qui fonctionnent depuis longtemps, il est prudent de vous assurer que les pods se terminent correctement le moment venu. Selon votre stratégie de mise à niveau, les mises à niveau du cluster Kubernetes peuvent nécessiter la création de nouveaux nœuds de travail, ce qui nécessite que tous les pods soient recréés sur des nœuds plus récents. Une gestion appropriée des résiliations et des budgets d'interruption de service peuvent vous aider à éviter les interruptions de service, car les pods sont retirés des anciens nœuds et recréés sur les nouveaux nœuds.

La méthode préférée pour mettre à niveau les nœuds de travail consiste à créer de nouveaux nœuds de travail et à mettre fin aux anciens. Avant de mettre fin aux nœuds de travail, vous devez le `drain` faire. Lorsqu'un nœud de travail est vidé, tous ses modules sont expulsés en toute sécurité. La sécurité est un mot clé ici ; lorsque les modules d'un travailleur sont expulsés, ils ne reçoivent pas simplement un `SIGKILL` signal. Au lieu de cela, un `SIGTERM` signal est envoyé au processus principal (PID 1) de chaque conteneur des Pods en cours d'expulsion. Une fois le `SIGTERM` signal envoyé, Kubernetes accorde au processus un certain temps (période de grâce) avant qu'un `SIGKILL` signal ne soit envoyé. Ce délai de grâce est de 30 secondes par défaut ; vous pouvez remplacer le délai par défaut en utilisant `grace-period` flag dans kubectl ou en le déclarant `terminationGracePeriodSeconds` dans votre Podspec.

```
kubectl delete pod <pod name> —grace-period=<seconds>
```

Il est courant d'avoir des conteneurs dans lesquels le processus principal n'a pas de PID 1. Considérez ce conteneur d'échantillons basé sur Python :

```
$ kubectl exec python-app -it ps
 PID USER TIME COMMAND
 1   root 0:00 {script.sh} /bin/sh ./script.sh
 5   root 0:00 python app.py
```

Dans cet exemple, le script shell reçoit`SIGTERM`, le processus principal, qui se trouve être une application Python dans cet exemple, ne reçoit aucun `SIGTERM` signal. Lorsque le Pod sera arrêté, l'application Python sera arrêtée brusquement. Cela peut être résolu en modifiant le `ENTRYPOINT`conteneur pour lancer l'application Python. Vous pouvez également utiliser un outil tel que dumb-init pour vous assurer que votre application peut gérer les signaux.

Vous pouvez également utiliser les hooks de conteneur pour exécuter un script ou une requête HTTP au démarrage ou à l'arrêt du conteneur. L'action `PreStop` hook s'exécute avant que le

conteneur ne reçoive un `SIGTERM` signal et doit être terminée avant que ce signal ne soit envoyé. La `terminationGracePeriodSeconds` valeur s'applique à partir du moment où l'action `PreStop` hook commence à s'exécuter, et non à partir du moment où le `SIGTERM` signal est envoyé.

Recommandations

Protégez les charges de travail critiques avec Pod Disruption Budgets

Pod Disruption Budget ou PDB peuvent interrompre temporairement le processus d'expulsion si le nombre de répliques d'une application tombe en dessous du seuil déclaré. Le processus d'expulsion se poursuivra une fois que le nombre de répliques disponibles aura dépassé le seuil. Vous pouvez utiliser PDB pour déclarer le `maxUnavailable` nombre `minAvailable` et le nombre de répliques. Par exemple, si vous souhaitez qu'au moins trois copies de votre application soient disponibles, vous pouvez créer un PDB.

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: my-svc-pdb
spec:
  minAvailable: 3
  selector:
    matchLabels:
      app: my-svc
```

La politique PDB ci-dessus indique à Kubernetes d'arrêter le processus d'expulsion jusqu'à ce que trois répliques ou plus soient disponibles. Le drainage des nœuds respecte`PodDisruptionBudgets`. Lors d'une mise à niveau d'un groupe de nœuds géré par EKS, les nœuds sont vidangés avec un délai de 15 minutes. Au bout de quinze minutes, si la mise à jour n'est pas forcée (l'option s'appelle Mise à jour continue dans la console EKS), la mise à jour échoue. Si la mise à jour est forcée, les modules sont supprimés.

Pour les nœuds autogérés, vous pouvez également utiliser des outils tels qu'AWS Node Termination Handler, qui garantit que le plan de contrôle Kubernetes répond de manière appropriée aux événements susceptibles de rendre votre instance EC2 indisponible, tels que les événements de maintenance EC2 et les interruptions ponctuelles d'EC2. Il utilise l'API Kubernetes pour boucler le nœud afin de garantir qu'aucun nouveau pod n'est planifié, puis le vide, mettant fin à tous les pods en cours d'exécution.

Vous pouvez utiliser l'anti-affinité des pods pour planifier les pods d'un déploiement sur différents nœuds et éviter les retards liés au PDB lors des mises à niveau des nœuds.

Pratiquez l'ingénierie du chaos

L'ingénierie du chaos est la discipline qui consiste à expérimenter sur un système distribué afin de renforcer la confiance dans la capacité du système à résister à des conditions de production turbulentes.

Dans son blog, Dominik Tornow explique que Kubernetes est un système déclaratif où « l'utilisateur fournit une représentation de l'état souhaité du système au système. Le système prend ensuite en compte l'état actuel et l'état souhaité pour déterminer la séquence de commandes permettant de passer de l'état actuel à l'état souhaité. « Cela signifie que Kubernetes stocke toujours l'état souhaité et que si le système s'écarte, Kubernetes prendra des mesures pour rétablir l'état. Par exemple, si un nœud de travail devient indisponible, Kubernetes replanifiera les pods sur un autre nœud de travail. De même, en cas de `replica` panne, le contrôleur de déploiement en créera un nouveau. `replica` De cette façon, les contrôleurs Kubernetes corrigent automatiquement les défaillances.

Les outils d'ingénierie du chaos tels que Gremlin vous aident à tester la résilience de votre cluster Kubernetes et à identifier les points de défaillance uniques. Les outils qui introduisent un chaos artificiel dans votre cluster (et au-delà) peuvent révéler des faiblesses systémiques, permettre d'identifier les goulots d'étranglement et les erreurs de configuration, et corriger les problèmes dans un environnement contrôlé. La philosophie de Chaos Engineering préconise de casser les objets volontairement et de tester l'infrastructure sous contrainte afin de minimiser les temps d'arrêt imprévus.

Utiliser un Service Mesh

Vous pouvez utiliser un maillage de services pour améliorer la résilience de votre application. Les maillages de services permettent service-to-service la communication et augmentent l'observabilité de votre réseau de microservices. La plupart des produits Service Mesh fonctionnent en faisant fonctionner un petit proxy réseau à côté de chaque service qui intercepte et inspecte le trafic réseau de l'application. Vous pouvez placer votre application dans un maillage sans modifier votre application. À l'aide des fonctionnalités intégrées du proxy de service, vous pouvez lui demander de générer des statistiques réseau, de créer des journaux d'accès et d'ajouter des en-têtes HTTP aux demandes sortantes de suivi distribué.

Un maillage de services peut vous aider à rendre vos microservices plus résilients grâce à des fonctionnalités telles que les nouvelles tentatives automatiques de demande, les délais d'expiration, les disjonctions et la limitation de débit.

Si vous gérez plusieurs clusters, vous pouvez utiliser un maillage de services pour permettre la service-to-service communication entre clusters.

Maillages de service

- Istio
- LinkerD
- Consul

Observabilité

L'observabilité est un terme générique qui inclut la surveillance, l'enregistrement et le traçage. Les applications basées sur des microservices sont distribuées par nature. Contrairement aux applications monolithiques où la surveillance d'un seul système est suffisante, dans une architecture d'application distribuée, vous devez surveiller les performances de chaque composant. Vous pouvez utiliser des systèmes de surveillance, de journalisation et de suivi distribué au niveau du cluster pour identifier les problèmes dans votre cluster avant qu'ils ne perturbent vos clients.

Les outils intégrés de Kubernetes pour le dépannage et la surveillance sont limités. Le serveur de mesures collecte les métriques relatives aux ressources et les stocke en mémoire, mais ne les conserve pas. Vous pouvez consulter les journaux d'un Pod à l'aide de kubectl, mais Kubernetes ne conserve pas automatiquement les journaux. Et la mise en œuvre du traçage distribué se fait soit au niveau du code de l'application, soit à l'aide de maillages de services.

L'extensibilité de Kubernetes brille ici. Kubernetes vous permet d'apporter votre solution centralisée préférée de surveillance, de journalisation et de suivi.

Recommandations

Surveillez vos applications

Le nombre de mesures que vous devez surveiller dans les applications modernes ne cesse de croître. Il est utile de disposer d'un moyen automatisé de suivre vos applications afin de pouvoir vous

concentrer sur la résolution des problèmes de vos clients. Les outils de surveillance à l'échelle du cluster tels que Prometheus ou CloudWatchContainer Insights peuvent surveiller votre cluster et votre charge de travail et vous indiquer quand, ou de préférence, avant que les choses ne se passent mal.

Les outils de surveillance vous permettent de créer des alertes auxquelles votre équipe des opérations peut s'abonner. Envisagez des règles pour activer les alarmes en cas d'événements susceptibles, lorsqu'ils sont exacerbés, d'entraîner une panne ou d'avoir un impact sur les performances des applications.

Si vous ne savez pas quels indicateurs vous devez surveiller, vous pouvez vous inspirer des méthodes suivantes :

- Méthode RED. Représente les demandes, les erreurs et la durée.
- Méthode USE. Désigne l'utilisation, la saturation et les erreurs.

Le billet de Sysdig Best practices for alerting on Kubernetes inclut une liste complète des composants susceptibles d'avoir un impact sur la disponibilité de vos applications.

Utiliser la bibliothèque cliente Prometheus pour exposer les métriques de l'application

En plus de surveiller l'état de l'application et d'agréger des métriques standard, vous pouvez également utiliser la bibliothèque cliente Prometheus pour exposer des métriques personnalisées spécifiques à l'application afin d'améliorer l'observabilité de l'application.

Utilisez des outils de journalisation centralisés pour collecter et conserver les journaux

La journalisation dans EKS se divise en deux catégories : les journaux du plan de contrôle et les journaux des applications. La journalisation du plan de contrôle EKS fournit des journaux d'audit et de diagnostic directement depuis le plan de contrôle vers CloudWatch les journaux de votre compte. Les journaux d'application sont des journaux produits par des pods exécutés au sein de votre cluster. Les journaux d'applications incluent les journaux produits par les pods qui exécutent les applications de logique métier et les composants du système Kubernetes tels que CoreDNS, Cluster Autoscaler, Prometheus, etc.

EKS fournit cinq types de journaux du plan de contrôle :

1. Journaux des composants du serveur d'API Kubernetes
2. Audit
3. Authentificateur

4. Responsable du contrôleur
5. Planificateur

Le gestionnaire du contrôleur et les journaux du planificateur peuvent aider à diagnostiquer les problèmes liés au plan de contrôle, tels que les goulots d'étranglement et les erreurs. Par défaut, les journaux du plan de contrôle EKS ne sont pas envoyés à CloudWatch Logs. Vous pouvez activer la journalisation du plan de contrôle et sélectionner les types de journaux du plan de contrôle EKS que vous souhaitez capturer pour chaque cluster de votre compte

La collecte des journaux d'applications nécessite l'installation d'un outil d'agrégation de journaux tel que Fluent Bit, Fluentd ou CloudWatchContainer Insights dans votre cluster.

Les outils d'agrégation de journaux Kubernetes s'exécutent au fur DaemonSets et à mesure que les journaux des conteneurs sont extraits des nœuds. Les journaux des applications sont ensuite envoyés vers une destination centralisée pour y être stockés. Par exemple, CloudWatch Container Insights peut utiliser Fluent Bit ou Fluentd pour collecter des journaux et les envoyer à Logs à CloudWatch des fins de stockage. Fluent Bit et Fluentd prennent en charge de nombreux systèmes d'analyse de journaux populaires tels qu'Elasticsearch et InfluxDB, ce qui vous permet de modifier le backend de stockage de vos journaux en modifiant Fluent bit ou la configuration des journaux de Fluentd.

Utiliser un système de suivi distribué pour identifier les goulots d'étranglement

Une application moderne typique possède des composants répartis sur le réseau, et sa fiabilité dépend du bon fonctionnement de chacun des composants composant l'application. Vous pouvez utiliser une solution de suivi distribué pour comprendre le flux des demandes et la manière dont les systèmes communiquent. Les traces peuvent vous indiquer où se trouvent les goulots d'étranglement dans votre réseau d'applications et prévenir les problèmes susceptibles de provoquer des défaillances en cascade.

Deux options s'offrent à vous pour implémenter le suivi dans vos applications : vous pouvez soit implémenter le suivi distribué au niveau du code à l'aide de bibliothèques partagées, soit utiliser un maillage de services.

La mise en œuvre du traçage au niveau du code peut s'avérer désavantageuse. Dans cette méthode, vous devez apporter des modifications à votre code. Cela est encore plus compliqué si vous avez des applications polyglottes. Vous êtes également responsable de la gestion d'une autre bibliothèque, dans l'ensemble de vos services.

Les maillages de service tels que LinkerD et Istio peuvent être utilisés pour implémenter le suivi distribué dans votre application avec un minimum de modifications du code de l'application. Vous pouvez utiliser le maillage de service pour standardiser la génération, la journalisation et le suivi des métriques.

Les outils de traçage tels qu'AWS X-Ray et Jaeger prennent en charge les implémentations de bibliothèques partagées et de maillage de services.

Envisagez d'utiliser un outil de suivi tel qu'AWS X-Ray ou Jaeger qui prend en charge les deux implémentations (bibliothèque partagée et maillage de services) afin de ne pas avoir à changer d'outil si vous adoptez ultérieurement le maillage de services.

Plan de contrôle EKS

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Amazon Elastic Kubernetes Service (EKS) est un service Kubernetes géré qui vous permet d'exécuter facilement Kubernetes sur AWS sans avoir à installer, exploiter et gérer votre propre plan de contrôle Kubernetes ou vos propres nœuds de travail. Il fonctionne en amont avec Kubernetes et est certifié conforme à Kubernetes. Cette conformité garantit qu'EKS prend en charge les API Kubernetes, tout comme la version communautaire open source que vous pouvez installer sur EC2 ou sur site. Les applications existantes exécutées sur Kubernetes en amont sont compatibles avec Amazon EKS.

EKS gère automatiquement la disponibilité et l'évolutivité des nœuds du plan de contrôle Kubernetes et remplace automatiquement les nœuds du plan de contrôle défectueux.

Architecture d'EKS

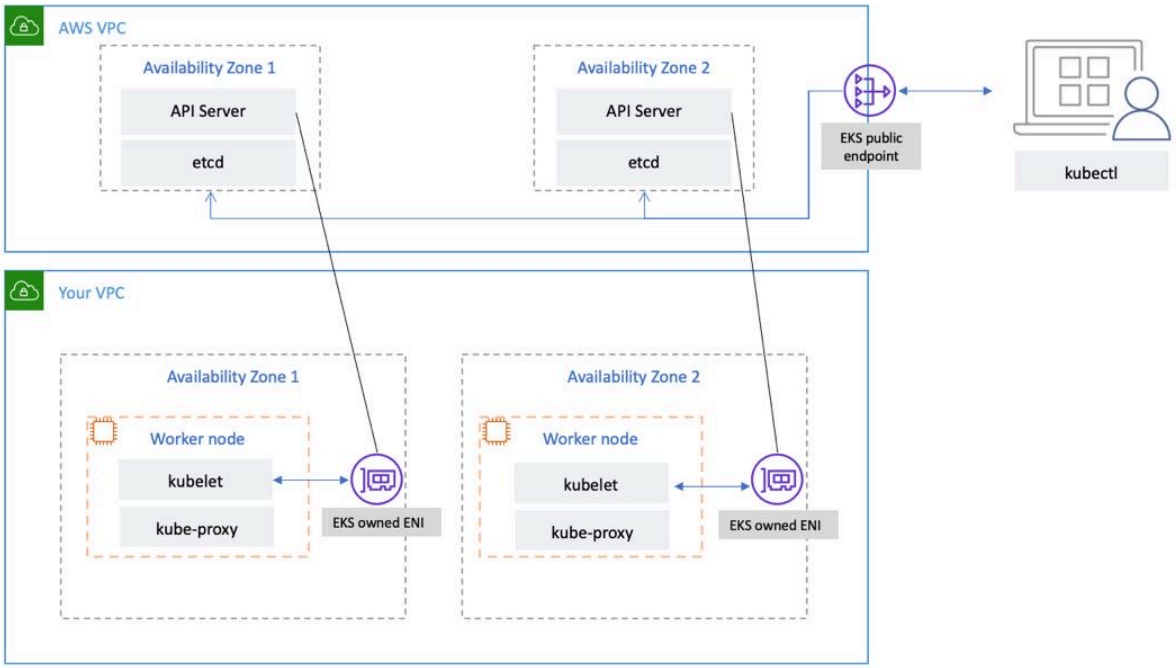
L'architecture EKS est conçue pour éliminer tous les points de défaillance susceptibles de compromettre la disponibilité et la durabilité du plan de contrôle Kubernetes.

Le plan de contrôle Kubernetes géré par EKS s'exécute dans un VPC géré par EKS. Le plan de contrôle EKS comprend les nœuds du serveur d'API Kubernetes, le cluster, etc. Des nœuds de serveur d'API Kubernetes qui exécutent des composants tels que le serveur d'API, le planificateur et s'exécutent `kube-controller-manager` dans un groupe d'auto-scaling. EKS exécute au moins

deux nœuds de serveur d'API dans des zones de disponibilité distinctes (AZs) au sein de la région AWS. De même, pour des raisons de durabilité, les nœuds du serveur etcd s'exécutent également dans un groupe d'auto-scaling composé de trois nœuds. AZs EKS exécute une passerelle NAT dans chaque AZ, et les serveurs API et les serveurs etcd s'exécutent dans un sous-réseau privé. Cette architecture garantit qu'un événement dans une seule zone de disponibilité n'affecte pas la disponibilité du cluster EKS.

Lorsque vous créez un nouveau cluster, Amazon EKS crée un point de terminaison hautement disponible pour le serveur d'API Kubernetes géré que vous utilisez pour communiquer avec votre cluster (à l'aide d'outils tels que). `kubectl` Le point de terminaison géré utilise le NLB pour équilibrer la charge des serveurs d'API Kubernetes. EKS fournit également deux ENI différentes AZs pour faciliter la communication avec vos nœuds de travail.

Connectivité réseau EKS Data Plane



Vous pouvez configurer si le serveur d'API de votre cluster Kubernetes est accessible depuis l'Internet public (via le point de terminaison public) ou via votre VPC (en utilisant le serveur géré par EKS) ou les deux. ENIs

Que les utilisateurs et les nœuds de travail se connectent au serveur API via le point de terminaison public ou l'ENI géré par EKS, il existe des chemins de connexion redondants.

Recommandations

Passez en revue les recommandations suivantes.

Surveiller les métriques du plan de contrôle

La surveillance des métriques de l'API Kubernetes peut vous donner un aperçu des performances du plan de contrôle et identifier les problèmes. Un plan de contrôle défaillant peut compromettre la disponibilité des charges de travail exécutées au sein du cluster. Par exemple, des contrôleurs mal écrits peuvent surcharger les serveurs d'API, affectant ainsi la disponibilité de votre application.

Kubernetes expose les métriques du plan de contrôle au point de terminaison. `/metrics`

Vous pouvez consulter les métriques exposées à l'aide de `kubectl` :

```
kubectl get --raw /metrics
```

Ces statistiques sont représentées dans un format texte Prometheus.

Vous pouvez utiliser Prometheus pour collecter et stocker ces statistiques. En mai 2020, CloudWatch ajout de la prise en charge de la surveillance des métriques CloudWatch de Prometheus dans Container Insights. Vous pouvez donc également utiliser Amazon CloudWatch pour surveiller le plan de contrôle EKS. Vous pouvez utiliser le didacticiel pour ajouter une nouvelle cible Prometheus Scrape : Prometheus KPI Server Metrics pour collecter des métriques CloudWatch et créer un tableau de bord pour surveiller le plan de contrôle de votre cluster.

Vous trouverez les statistiques du serveur d'API Kubernetes ici. Par exemple, `apiserver_request_duration_seconds` peut indiquer le temps d'exécution des demandes d'API.

Envisagez de surveiller les métriques du plan de contrôle suivantes :

Serveur d'API

Métrique	Description
`apiserver_request_total`	Le compteur de requêtes apiserver est ventilé pour chaque verbe, valeur d'exécution à sec, groupe, version, ressource, étendue, composant et code de réponse HTTP.

Métrique	Description
`apiserver_request_duration_seconds*`	Histogramme de latence de réponse en secondes pour chaque verbe, valeur d'essai, groupe, version, ressource, sous-ressource, portée et composant.
`apiserver_admission_controller_admission_duration_seconds*`	Histogramme de latence du contrôleur d'admission en secondes, identifié par son nom et ventilé pour chaque opération, ressource d'API et type (validation ou admission).
`apiserver_admission_webhook_rejection_count`	Nombre de refus de webbooks d'admission. Identifié par nom, opération, code de rejet, type (validation ou admission), type d'erreur (calling_webhook_error, apiserver_internal_error, no_error)
`rest_client_request_duration_seconds*`	Histogramme de latence des demandes en secondes. Décomposé par verbe et par URL.
`rest_client_requests_total`	Nombre de requêtes HTTP, partitionnées par code d'état, méthode et hôte.

- Les métriques de l'histogramme incluent les suffixes _bucket, _sum et _count.

etcd

Métrique	Descri on
`etcd_request_duration_seconds*`	Histog me de latenc des demar

Métrique	Descri on
	Etcd en secon pour chaqu opérat et type d'objet
`apiserver_storage_db_total_size_in_bytes` ou `apiserver_storage_size_bytes` (à partir de EKS v1.28)	Taille de la base de donné Etcd.

- Les métriques de l'histogramme incluent les suffixes _bucket, _sum et _count.

Envisagez d'utiliser le tableau de bord de vue d'ensemble de la surveillance de Kubernetes pour visualiser et surveiller les demandes du serveur d'API Kubernetes, ainsi que les mesures de latence, etc.

> **Important**
>
> Lorsque la limite de taille de base de données est dépassée, etcd émet une alarme d'absence d'espace et arrête de prendre d'autres demandes d'écriture. En d'autres termes, le cluster passe en lecture seule et toutes les demandes de mutation d'objets, telles que la création de nouveaux pods, le dimensionnement des déploiements, etc., seront rejetées par le serveur API du cluster.

Authentification par cluster

EKS prend actuellement en charge deux types d'authentification : les jetons de compte porteur/de service et l'authentification IAM qui utilise l'authentification par jeton Webhook. Lorsque les utilisateurs appellent l'API Kubernetes, un webhook transmet à IAM un jeton d'authentification inclus dans la demande. Le jeton, une URL signée en base 64, est généré par l'interface de ligne de commande AWS (AWS CLI).

L'utilisateur ou le rôle IAM qui crée le cluster EKS obtient automatiquement un accès complet au cluster. Vous pouvez gérer l'accès au cluster EKS en modifiant le fichier de configuration aws-auth.

Si vous configurez mal le `aws-auth` configmap et perdez l'accès au cluster, vous pouvez toujours utiliser l'utilisateur ou le rôle du créateur du cluster pour accéder à votre cluster EKS.

Dans le cas peu probable où vous ne pourriez pas utiliser le service IAM dans la région AWS, vous pouvez également utiliser le jeton porteur du compte de service Kubernetes pour gérer le cluster.

Créez un `super-admin` compte autorisé à effectuer toutes les actions du cluster :

```
kubectl -n kube-system create serviceaccount super-admin
```

Créez une liaison de rôle qui donne le rôle super-admin cluster-admin :

```
kubectl create clusterrolebinding super-admin-rb --clusterrole=cluster-admin --
serviceaccount=kube-system:super-admin
```

Obtenez le secret du compte de service :

```
SECRET_NAME=`kubectl -n kube-system get serviceaccount/super-admin -o
 jsonpath='{.secrets[0].name}'`
```

Obtenez le jeton associé au secret :

```
TOKEN=`kubectl -n kube-system get secret $SECRET_NAME -o jsonpath='{.data.token}'|
 base64 --decode`
```

Ajoutez un compte de service et un jeton à `kubeconfig` :

```
kubectl config set-credentials super-admin --token=$TOKEN
```

Définissez le contexte actuel `kubeconfig` pour utiliser le compte super-administrateur :

```
kubectl config set-context --current --user=super-admin
```

La finale `kubeconfig` devrait ressembler à ceci :

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data:<REDACTED>
    server: https://<CLUSTER>.gr7.us-west-2.eks.amazonaws.com
  name: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
contexts:
- context:
    cluster: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
    user: super-admin
  name: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
current-context: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
kind: Config
preferences: {}
users:
#- name: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
#  user:
#    exec:
#      apiVersion: client.authentication.k8s.io/v1beta1
#      args:
#      - --region
#      - us-west-2
#      - eks
#      - get-token
#      - --cluster-name
#      - <<cluster name>>
#      command: aws
#      env: null
- name: super-admin
  user:
    token: <<super-admin sa's secret>>
```

Webhooks d'admission

Kubernetes propose deux types de webhooks d'admission : les webhooks d'admission validants et les webhooks d'admission mutants. Ils permettent à un utilisateur d'étendre l'API Kubernetes et de

valider ou de muter des objets avant qu'ils ne soient acceptés par l'API. Les mauvaises configurations de ces webhooks peuvent déstabiliser le plan de contrôle EKS en bloquant les opérations critiques du cluster.

Pour éviter d'avoir un impact sur les opérations critiques du cluster, évitez de définir des webhooks « fourre-tout » comme suit :

```
- name: "pod-policy.example.com"
  rules:
  - apiGroups:   ["*"]
    apiVersions: ["*"]
    operations:  ["*"]
    resources:   ["*"]
    scope: "*"
```

Ou assurez-vous que le webhook dispose d'une politique d'ouverture en cas d'échec avec un délai d'expiration inférieur à 30 secondes afin de garantir que si votre webhook n'est pas disponible, cela n'affectera pas les charges de travail critiques du cluster.

Bloquez les pods en cas de danger **`sysctls`**

`Sysctl`est un utilitaire Linux qui permet aux utilisateurs de modifier les paramètres du noyau pendant l'exécution. Ces paramètres du noyau contrôlent différents aspects du comportement du système d'exploitation, tels que le réseau, le système de fichiers, la mémoire virtuelle et la gestion des processus.

Kubernetes permet d'attribuer `sysctl` des profils aux pods. Kubernetes est classé `systcls` comme sûr et dangereux. Les espaces de noms sécurisés `sysctls` sont placés dans le conteneur ou le pod, et leur configuration n'a aucun impact sur les autres pods du nœud ou sur le nœud lui-même. En revanche, les sysctls non sécurisés sont désactivés par défaut car ils peuvent potentiellement perturber les autres pods ou rendre le nœud instable.

Comme `sysctls` les éléments non sécurisés sont désactivés par défaut, le kubelet ne créera pas de Pod avec un profil dangereux`sysctl`. Si vous créez un tel pod, le planificateur attribuera à plusieurs reprises de tels pods aux nœuds, alors que le nœud ne le lance pas. Cette boucle infinie finit par mettre à rude épreuve le plan de contrôle du cluster, le rendant instable.

Envisagez d'utiliser OPA Gatekeeper ou Kyverno pour rejeter les pods présentant des risques. `sysctls`

Gestion des mises à niveau de clusters

Depuis avril 2021, le cycle de publication de Kubernetes est passé de quatre versions par an (une fois par trimestre) à trois versions par an. Une nouvelle version mineure (comme 1. 21 ou 1. 22) est publié environ toutes les quinze semaines. À partir de Kubernetes 1.19, chaque version mineure est prise en charge pendant environ douze mois après sa première publication. Avec l'arrivée de Kubernetes v1.28, l'écart de compatibilité entre le plan de contrôle et les nœuds de travail est passé des versions mineures n-2 à n-3. Pour en savoir plus, consultez la section Meilleures pratiques pour les mises à niveau de clusters.

Connectivité des terminaux du cluster

Lorsque vous travaillez avec Amazon EKS (Elastic Kubernetes Service), vous pouvez rencontrer des interruptions de connexion ou des erreurs lors d'événements tels que le dimensionnement ou l'application de correctifs au plan de contrôle Kubernetes. Ces événements peuvent entraîner le remplacement des instances de kube-apiserver, ce qui peut entraîner le renvoi d'adresses IP différentes lors de la résolution du FQDN. Ce document décrit les meilleures pratiques permettant aux utilisateurs d'API Kubernetes de maintenir une connectivité fiable.

> **Note**
>
> La mise en œuvre de ces meilleures pratiques peut nécessiter des mises à jour des configurations client ou des scripts afin de gérer efficacement les nouvelles stratégies de résolution DNS et de réessayer.

Le principal problème provient de la mise en cache côté client du DNS et du risque d'adresses IP périmées du point de terminaison EKS (NLB public pour point de terminaison public ou X-ENI pour point de terminaison privé). Lorsque les instances de kube-apiserver sont remplacées, le nom de domaine complet (FQDN) peut être converti en de nouvelles adresses IP. Cependant, en raison des paramètres TTL (DNS Time to Live), qui sont définis sur 60 secondes dans la zone Route 53 gérée par AWS, les clients peuvent continuer à utiliser des adresses IP obsolètes pendant une courte période.

Pour atténuer ces problèmes, les utilisateurs d'API Kubernetes (tels que kubectl, les CI/CD pipelines et les applications personnalisées) doivent mettre en œuvre les meilleures pratiques suivantes :

- Implémenter la rerésolution du DNS

- Implémentez les nouvelles tentatives avec Backoff et Jitter. Par exemple, consultez cet article intitulé Failures Happen
- Implémentez les délais d'attente des clients. Définissez des délais d'expiration appropriés pour éviter que les demandes de longue durée ne bloquent votre application. Sachez que certaines bibliothèques clientes Kubernetes, en particulier celles générées par les générateurs OpenAPI, peuvent ne pas permettre de définir facilement des délais d'expiration personnalisés.
  - Exemple 1 avec kubectl :

```
kubectl get pods --request-timeout 10s # default: no timeout
```

  - Exemple 2 avec Python : le client Kubernetes fournit un paramètre _request_timeout

En mettant en œuvre ces meilleures pratiques, vous pouvez améliorer de manière significative la fiabilité et la résilience de vos applications lorsque vous interagissez avec l'API Kubernetes. N'oubliez pas de tester minutieusement ces implémentations, en particulier dans des conditions de défaillance simulées, afin de vous assurer qu'elles se comportent comme prévu lors des événements de dimensionnement ou d'application de correctifs.

Gestion de grands clusters

EKS surveille activement la charge sur les instances du plan de contrôle et les adapte automatiquement pour garantir des performances élevées. Toutefois, vous devez tenir compte des problèmes et des limites de performances potentiels au sein de Kubernetes ainsi que des quotas dans les services AWS lorsque vous exécutez de grands clusters.

- Les clusters comportant plus de 1 000 services peuvent subir une latence réseau lors de l'utilisation `kube-proxy` en `iptables` mode in, selon les tests effectués par l' ProjectCalico équipe. La solution consiste à passer `kube-proxy`en `ipvs` mode exécution.
- Vous pouvez également rencontrer une limitation des demandes d'API EC2 si le CNI doit demander des adresses IP pour les pods ou si vous devez créer fréquemment de nouvelles instances EC2. Vous pouvez réduire les appels à l'API EC2 en configurant le CNI pour mettre en cache les adresses IP. Vous pouvez utiliser des types d'instances EC2 plus grands pour réduire les événements de dimensionnement EC2.

Ressources supplémentaires :

- Démystifier le réseau de clusters pour les nœuds de travail Amazon EKS
- Contrôle d'accès aux points de terminaison du cluster Amazon EKS
- AWS re:Invent 2019 : Amazon EKS sous le capot (-R1) CON421

Plan de données EKS

Pour exploiter des applications résilientes et à haute disponibilité, vous avez besoin d'un plan de données hautement disponible et résilient. Un plan de données élastique permet à Kubernetes de dimensionner et de réparer automatiquement vos applications. Un plan de données résilient se compose de deux nœuds de travail ou plus, peut augmenter ou diminuer en fonction de la charge de travail et se rétablir automatiquement en cas de panne.

Plusieurs options s'offrent à vous pour les nœuds de travail dotés d'EKS : nœuds gérés en mode automatique EKS, instances EC2 et Fargate.

Le mode automatique EKS offre le chemin le plus simple vers un plan de données résilient. Le mode automatique étend la gestion des clusters Kubernetes par AWS au-delà du cluster lui-même, afin de permettre à AWS de configurer et de gérer également l'infrastructure qui permet le bon fonctionnement de vos charges de travail. Le mode automatique redimensionne automatiquement le plan de données à la hausse ou à la baisse au fur et à mesure que Kubernetes adapte les pods et veille à ce que les nœuds de votre cluster soient dimensionnés de manière appropriée et rentable pour les charges de travail en cours d'exécution.

Si vous choisissez des instances EC2, vous pouvez gérer vous-même les nœuds de travail ou utiliser des groupes de nœuds gérés par EKS. Vous pouvez avoir un cluster combinant le mode automatique, des nœuds de travail gérés et autogérés et Fargate.

Fargate exécute chaque Pod dans un environnement informatique isolé. Chaque Pod exécuté sur Fargate possède son propre nœud de travail. Fargate redimensionne automatiquement le plan de données au fur et à mesure que Kubernetes redimensionne les pods. Vous pouvez redimensionner à la fois le plan de données et votre charge de travail à l'aide de l'autoscaler à modules horizontaux.

La méthode préférée pour dimensionner les nœuds de travail EC2 (si vous n'utilisez pas le mode automatique EKS où cela est effectué automatiquement par AWS) consiste à utiliser les groupes Karpenter, Kubernetes Cluster Autoscaler ou EC2 Auto Scaling.

Recommandations

Répartissez les nœuds de travail et les charges de travail sur plusieurs AZs

Vous pouvez protéger vos charges de travail contre les défaillances dans une zone de disponibilité individuelle en exécutant des nœuds de travail et des pods en plusieurs AZs. Vous pouvez contrôler l'AZ dans laquelle les nœuds de travail sont créés à l'aide des sous-réseaux dans lesquels vous créez les nœuds.

La méthode recommandée pour répartir les pods AZs est d'utiliser les contraintes de propagation topologique pour les pods. Les fonctionnalités de mise à l'échelle automatique telles que le mode automatique d'EKS et Karpenter sont conscientes des contraintes de dispersion topologique et lancent automatiquement les nœuds correctement AZs pour permettre de respecter vos contraintes.

Le déploiement ci-dessous répartit les pods AZs si possible, en les laissant fonctionner de toute façon dans le cas contraire :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-server
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web-server
  template:
    metadata:
      labels:
        app: web-server
    spec:
      topologySpreadConstraints:
        - maxSkew: 1
          whenUnsatisfiable: ScheduleAnyway
          topologyKey: topology.kubernetes.io/zone
          labelSelector:
            matchLabels:
              app: web-server
      containers:
      - name: web-app
        image: nginx
        resources:
```

```
          requests:
            cpu: 1
```

> **Note**
>
> `kube-schedulerne` connaît les domaines topologiques que via les nœuds qui existent avec ces étiquettes. Si le déploiement ci-dessus est déployé sur un cluster avec des nœuds dans une seule zone, tous les pods seront planifiés sur ces `kube-scheduler` nœuds, sans connaître les autres zones. Pour que cette répartition topologique fonctionne comme prévu avec le planificateur, des nœuds doivent déjà exister dans toutes les zones. La `minDomains` propriété des contraintes de dispersion topologique est utilisée pour informer le planificateur du nombre de domaines éligibles, même si un nœud y est exécuté pour éviter ce problème.

> **Warning**
>
> Si `DoNotSchedule` vous définissez `whenUnsatisfiable` cette valeur sur, les pods ne seront pas planifiables si la contrainte de propagation de la topologie ne peut pas être respectée. Il ne doit être défini que s'il est préférable que les pods ne s'exécutent pas au lieu de violer la contrainte de propagation de la topologie.

Sur les anciennes versions de Kubernetes, vous pouvez utiliser les règles anti-affinité des pods pour planifier des pods sur plusieurs. AZs Le manifeste ci-dessous indique au planificateur Kubernetes de préférer planifier les pods de manière distincte. AZs

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-server
  labels:
    app: web-server
spec:
  replicas: 4
  selector:
    matchLabels:
      app: web-server
  template:
    metadata:
```

```
      labels:
        app: web-server
    spec:
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - web-server
              topologyKey: failure-domain.beta.kubernetes.io/zone
            weight: 100
      containers:
      - name: web-app
        image: nginx
```

> **Warning**
>
> N'exigez pas que les pods soient planifiés de AZs manière distincte, sinon le nombre de pods dans un déploiement ne dépassera jamais le nombre de AZs.

Garantir la capacité de lancer des nœuds dans chaque AZ lors de l'utilisation de volumes EBS

Si vous utilisez Amazon EBS pour fournir des volumes persistants, vous devez vous assurer que les pods et le volume EBS associé se trouvent dans le même AZ. Un pod ne peut pas accéder aux volumes persistants sauvegardés par EBS situés dans une autre zone de disponibilité. Le planificateur Kubernetes sait dans quelle zone se trouve un nœud de travail à partir des étiquettes figurant sur le nœud et planifie toujours un pod nécessitant un volume EBS dans la même zone que le volume. Toutefois, si aucun nœud de travail n'est disponible dans l'AZ où se trouve le volume, le Pod ne peut pas être planifié.

Si vous utilisez le mode automatique d'EKS ou Karpenter, vous devez vous assurer que vous avez NodeClass sélectionné des sous-réseaux dans chaque AZ. Si vous utilisez des groupes de nœuds gérés, vous devez vous assurer que vous disposez d'un groupe de nœuds dans chaque zone de disponibilité.

Une capacité de stockage EBS est intégrée au mode automatique d'EKS, mais si vous utilisez Karpenter ou des groupes de nœuds gérés, l'EBS CSI devra également être installé.

Utiliser le mode automatique EKS pour gérer les nœuds de travail

Le mode automatique d'EKS rationalise la gestion d'EKS en fournissant des clusters prêts pour la production avec une charge opérationnelle minimale. Le mode automatique est chargé d'augmenter ou de diminuer le nombre de nœuds en fonction des pods exécutés dans le cluster. Les nœuds sont automatiquement mis à jour avec les correctifs logiciels et les mises à jour, les mises à jour étant effectuées conformément aux paramètres d'NodePoolinterruption configurés et aux budgets d'interruption des modules.

Exécutez l'agent de surveillance des nœuds

L'agent de surveillance des nœuds surveille les problèmes de santé des nœuds et y réagit en publiant des événements Kubernetes et en mettant à jour l'état des nœuds. L'agent de surveillance des nœuds est inclus dans les nœuds en mode automatique d'EKS et peut être installé en tant qu'extension EKS pour les nœuds qui ne sont pas gérés par le mode automatique.

Le mode automatique EKS, les groupes de nœuds gérés et Karpenter ont tous la capacité de détecter les conditions fatales signalées par l'agent de surveillance des nœuds et de réparer ces nœuds automatiquement lorsque ces conditions se produisent.

Implémenter la QoS

Pour les applications critiques, pensez à définir `requests` = `limits` pour le conteneur du Pod. Cela garantira que le conteneur ne sera pas détruit si un autre pod demande des ressources.

Il est recommandé d'implémenter des limites de processeur et de mémoire pour tous les conteneurs, car cela évite qu'un conteneur ne consomme par inadvertance des ressources système et n'ait un impact sur la disponibilité d'autres processus colocalisés.

Configurer et dimensionner les ressources Requests/Limits pour toutes les charges de travail

Certaines directives générales peuvent être appliquées au dimensionnement des demandes de ressources et aux limites des charges de travail :

- Ne spécifiez pas de limites de ressources sur le processeur. En l'absence de limites, la demande agit comme un poids sur le temps processeur relatif dont disposent les conteneurs. Cela permet à vos charges de travail d'utiliser l'intégralité du processeur sans limite artificielle ni privation.

- Pour les ressources autres que le processeur, la configuration `requests` = `limits` fournit le comportement le plus prévisible. Si `requests` ! `=limits`, la qualité de service du conteneur est également réduite, passant de garantie à burstable, ce qui le rend plus susceptible d'être expulsé en cas de pression sur les nœuds.
- Pour les ressources autres que le processeur, ne spécifiez pas de limite beaucoup plus grande que la demande. Plus la taille de la configuration est grande`requests`, plus les nœuds `limits` sont susceptibles d'être surchargés, ce qui augmente les risques d'interruption de la charge de travail.
- Les requêtes correctement dimensionnées sont particulièrement importantes lors de l'utilisation d'une solution d'auto-scaling des nœuds telle que Karpenter ou Cluster. AutoScaler Ces outils examinent vos demandes de charge de travail afin de déterminer le nombre et la taille des nœuds à provisionner. Si vos demandes sont trop petites et que les limites sont plus élevées, vous risquez de voir vos charges de travail expulsées ou d'éliminer OOM si elles ont été regroupées sur un nœud.

Il peut être difficile de déterminer les demandes de ressources, mais des outils tels que le Vertical Pod Autoscaler peuvent vous aider à « dimensionner correctement » les demandes en observant l'utilisation des ressources du conteneur lors de l'exécution. Les autres outils qui peuvent être utiles pour déterminer la taille des demandes sont les suivants :

- Boucles d'or
- Parca
- Profileur
- rsg

Configurer les quotas de ressources pour les espaces de noms

Les espaces de noms sont destinés à être utilisés dans des environnements avec de nombreux utilisateurs répartis sur plusieurs équipes ou projets. Ils fournissent un champ d'application pour les noms et permettent de répartir les ressources du cluster entre plusieurs équipes, projets et charges de travail. Vous pouvez limiter la consommation globale de ressources dans un espace de noms. L'`ResourceQuota`objet peut limiter la quantité d'objets pouvant être créés dans un espace de noms par type, ainsi que la quantité totale de ressources de calcul pouvant être consommées par les ressources de ce projet. Vous pouvez limiter la somme totale des ressources de and/or calcul de stockage (processeur et mémoire) qui peuvent être demandées dans un espace de noms donné.

Si le quota de ressources est activé pour un espace de noms pour les ressources de calcul telles que le processeur et la mémoire, les utilisateurs doivent spécifier les demandes ou les limites pour chaque conteneur de cet espace de noms.

Envisagez de configurer des quotas pour chaque espace de noms. Envisagez `LimitRanges` de l'utiliser pour appliquer automatiquement des limites préconfigurées aux conteneurs au sein d'un espace de noms.

Limiter l'utilisation des ressources du conteneur dans un espace de noms

Les quotas de ressources permettent de limiter la quantité de ressources qu'un espace de noms peut utiliser. L'`LimitRangeobjet` peut vous aider à implémenter les ressources minimales et maximales qu'un conteneur peut demander. `LimitRange`Vous pouvez ainsi définir une demande par défaut et des limites pour les conteneurs, ce qui est utile si la définition de limites de ressources de calcul n'est pas une pratique courante dans votre organisation. Comme son nom l'indique, `LimitRange` peut imposer une utilisation minimale et maximale des ressources de calcul par pod ou conteneur dans un espace de noms. De plus, appliquez le minimum et le maximum de demandes de stockage par PersistentVolumeClaim espace de noms.

Envisagez `LimitRange` de l'utiliser conjointement avec `ResourceQuota` pour appliquer des limites au niveau du conteneur et de l'espace de noms. La définition de ces limites garantit qu'un conteneur ou un espace de noms n'empiète pas sur les ressources utilisées par les autres locataires du cluster.

Utiliser NodeLocal DNSCache

Vous pouvez améliorer les performances du DNS du cluster en exécutant NodeLocalDNSCache. Cette fonctionnalité exécute un agent de mise en cache DNS sur les nœuds du cluster en tant DaemonSet que. Tous les pods utilisent l'agent de mise en cache DNS exécuté sur le nœud pour la résolution des noms au lieu d'utiliser le `kube-dns` service. Cette fonctionnalité est automatiquement incluse dans le mode automatique d'EKS.

Configurer l'auto-scaling CoreDNS

Une autre méthode pour améliorer les performances du DNS du cluster consiste à activer l'auto-scaling intégré des pods CoreDNS.

Cette fonctionnalité surveille en permanence l'état du cluster, notamment le nombre de nœuds et de cœurs de processeur. En fonction de ces informations, le contrôleur ajuste dynamiquement le nombre de réplicas du déploiement CoreDNS dans le cluster EKS.

Bonnes pratiques en matière de mise en réseau

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Il est essentiel de comprendre le réseau Kubernetes pour exploiter efficacement votre cluster et vos applications. Le réseau de pods, également appelé réseau de clusters, est au cœur du réseau Kubernetes. Kubernetes prend en charge les plug-ins CNI (Container Network Interface) pour la mise en réseau des clusters.

Amazon EKS prend officiellement en charge le plugin Amazon Virtual Private Cloud (VPC) CNI pour implémenter le réseau Kubernetes Pod. Le VPC CNI fournit une intégration native avec AWS VPC et fonctionne en mode sous-couche. En mode sous-couche, les pods et les hôtes sont situés sur la même couche réseau et partagent l'espace de noms du réseau. L'adresse IP du Pod est cohérente du point de vue du cluster et du VPC.

Ce guide présente l'interface réseau de conteneurs Amazon VPC (VPC CNI) dans le contexte de la mise en réseau de clusters Kubernetes. Le VPC CNI est le plugin réseau par défaut pris en charge par EKS et constitue donc l'objet du guide. Le VPC CNI est hautement configurable pour prendre en charge différents cas d'utilisation. Ce guide comprend également des sections dédiées sur les différents cas d'utilisation du VPC CNI, les modes de fonctionnement, les sous-composants, suivies des recommandations.

Amazon EKS exécute Kubernetes en amont et est certifié conforme à Kubernetes. Bien que vous puissiez utiliser d'autres plug-ins CNI, ce guide ne fournit pas de recommandations pour gérer les plug-ins alternatifs CNIs. Consultez la documentation EKS Alternate CNI pour obtenir une liste des partenaires et des ressources permettant de gérer CNIs efficacement les alternatives.

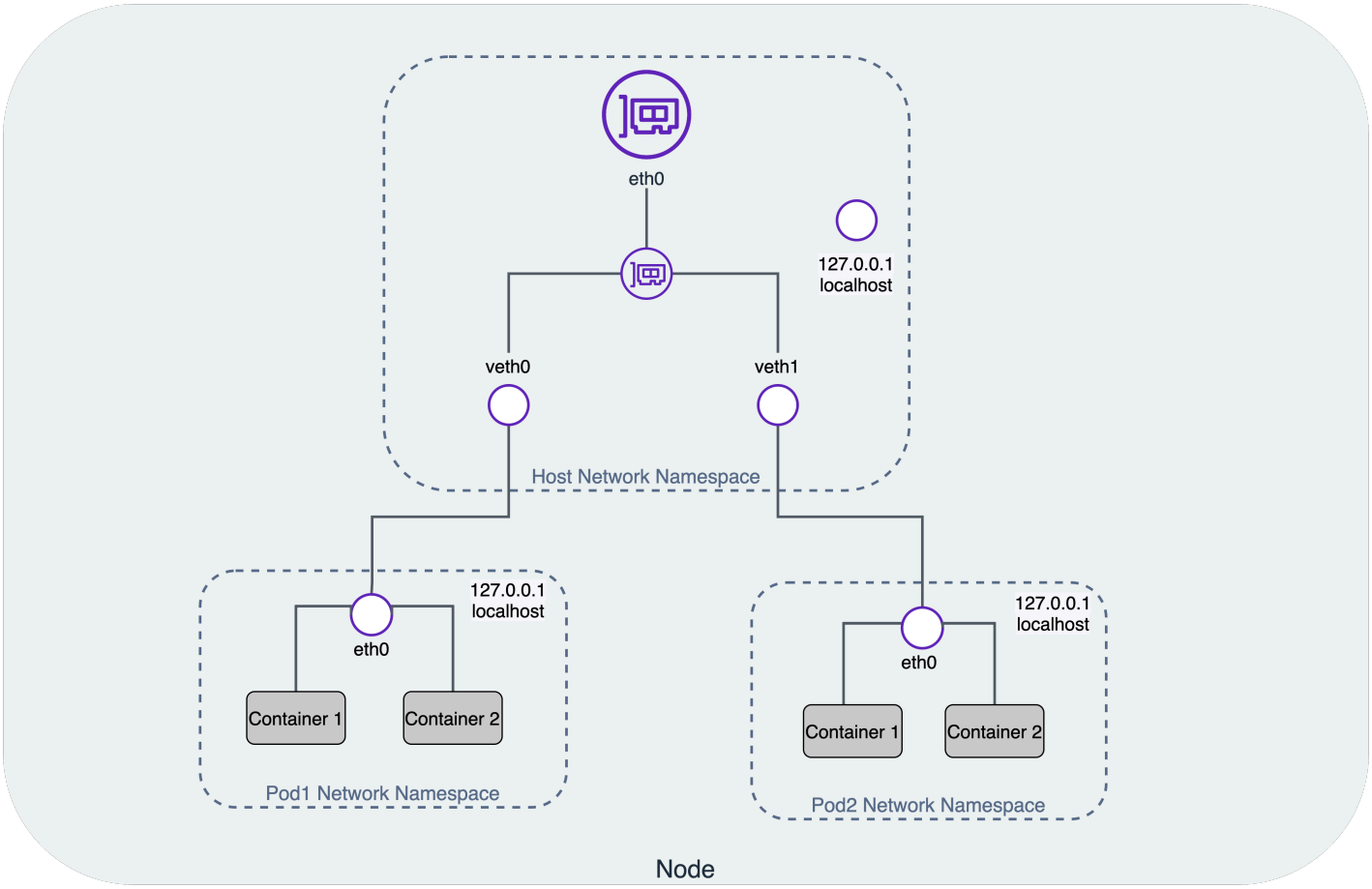
Modèle de réseau Kubernetes

Kubernetes définit les exigences suivantes en matière de mise en réseau de clusters :

- Les pods planifiés sur le même nœud doivent pouvoir communiquer avec d'autres pods sans utiliser le NAT (Network Address Translation).
- Tous les démons du système (processus d'arrière-plan, par exemple, kubelet) exécutés sur un nœud particulier peuvent communiquer avec les pods exécutés sur le même nœud.

- Les pods qui utilisent le réseau hôte doivent pouvoir contacter tous les autres pods sur tous les autres nœuds sans utiliser le NAT.

Consultez le modèle de réseau Kubernetes pour plus de détails sur ce que Kubernetes attend des implémentations réseau compatibles. La figure suivante illustre la relation entre les espaces de noms du réseau Pod et l'espace de noms du réseau hôte.



Interface réseau de conteneurs (CNI)

Kubernetes prend en charge les spécifications CNI et les plugins pour implémenter le modèle de réseau Kubernetes. Un CNI se compose d'une spécification (version actuelle 1.0.0) et de bibliothèques permettant d'écrire des plugins pour configurer les interfaces réseau dans des conteneurs, ainsi que d'un certain nombre de plugins pris en charge. Le CNI s'occupe uniquement de la connectivité réseau des conteneurs et de la suppression des ressources allouées lorsque le conteneur est supprimé.

Le plugin CNI est activé en passant à kubelet l'option de ligne de `--network-plugin=cni` commande. Kubelet lit un fichier depuis `--cni-conf-dir` (par défaut/ etc/cni/net .d) et utilise la configuration CNI de ce fichier pour configurer le réseau de chaque Pod. Le fichier de configuration CNI doit correspondre à la spécification CNI (version minimale 0.4.0) et tous les plugins CNI requis référencés par la configuration doivent être présents dans le `--cni-bin-dir` répertoire (par défaut/). opt/cni/bin S'il existe plusieurs fichiers de configuration CNI dans le répertoire, le kubelet utilise le fichier de configuration qui vient en premier par nom dans l'ordre lexicographique.

Amazon Virtual Private Cloud (VPC) CNI

Le VPC CNI fourni par AWS est le module complémentaire réseau par défaut pour les clusters EKS. Le module complémentaire VPC CNI est installé par défaut lorsque vous provisionnez des clusters EKS. Le VPC CNI s'exécute sur les nœuds de travail Kubernetes. Le module complémentaire VPC CNI comprend le binaire CNI et le plug-in de gestion des adresses IP (ipamd). Le CNI attribue une adresse IP du réseau VPC à un Pod. L'ipamd gère les interfaces réseau AWS Elastic (ENIs) vers chaque nœud Kubernetes et gère le pool de chaleur de. IPs Le VPC CNI fournit des options de configuration pour la pré-allocation ENIs et des adresses IP pour accélérer le démarrage des Pod. Consultez Amazon VPC CNI pour connaître les meilleures pratiques recommandées en matière de gestion des plug-ins.

Amazon EKS vous recommande de spécifier des sous-réseaux dans au moins deux zones de disponibilité lorsque vous créez un cluster. Amazon VPC CNI alloue des adresses IP aux pods à partir des sous-réseaux des nœuds. Nous vous recommandons vivement de vérifier les adresses IP disponibles dans les sous-réseaux. Veuillez prendre en compte les recommandations relatives aux VPC et aux sous-réseaux avant de déployer des clusters EKS.

Amazon VPC CNI alloue un pool chaud d'adresses IP secondaires à partir du ENIs sous-réseau attaché à l'ENI principal du nœud. Ce mode de VPC CNI est appelé mode IP secondaire. Le nombre d'adresses IP et donc le nombre de pods (densité de pods) est défini par le nombre ENIs et l'adresse IP par ENI (limites), tels que définis par le type d'instance. Le mode secondaire est le mode par défaut et fonctionne bien pour les petits clusters dotés de types d'instances plus petits. Veuillez envisager d'utiliser le mode préfixe si vous rencontrez des problèmes de densité de capsules. Vous pouvez également augmenter le nombre d'adresses IP disponibles sur le nœud pour les pods en attribuant des préfixes à. ENIs

Amazon VPC CNI s'intègre nativement à AWS VPC et permet aux utilisateurs d'appliquer les meilleures pratiques existantes en matière de mise en réseau et de sécurité des VPC AWS pour créer des clusters Kubernetes. Cela inclut la possibilité d'utiliser les journaux de flux VPC, les

politiques de routage VPC et les groupes de sécurité pour isoler le trafic réseau. Par défaut, le CNI Amazon VPC applique le groupe de sécurité associé à l'ENI principal sur le nœud aux pods. Envisagez d'activer les groupes de sécurité pour les pods lorsque vous souhaitez attribuer des règles réseau différentes à un pod.

Par défaut, le VPC CNI attribue des adresses IP aux pods à partir du sous-réseau attribué à l'ENI principal d'un nœud. Il est courant de rencontrer une pénurie d' IPv4 adresses lors de l'exécution de grands clusters avec des milliers de charges de travail. AWS VPC vous permet d'étendre la disponibilité IPs en affectant un secondaire CIDRs pour contourner l'épuisement des IPv4 blocs CIDR. AWS VPC CNI vous permet d'utiliser une plage d'adresses CIDR de sous-réseau différente pour les pods. Cette fonctionnalité du VPC CNI est appelée mise en réseau personnalisée. Vous pouvez envisager d'utiliser un réseau personnalisé pour utiliser `100.64.0.0/10` et `198.19.0.0/16` CIDRs (CG-NAT) avec EKS. Cela vous permet de créer un environnement dans lequel les pods ne consomment plus aucune RFC1918 adresse IP provenant de votre VPC.

La mise en réseau personnalisée est l'une des options permettant de IPv4 résoudre le problème de l'épuisement des adresses, mais elle implique une surcharge opérationnelle. Pour résoudre ce problème, nous recommandons d'utiliser des IPv6 clusters plutôt que des réseaux personnalisés. Plus précisément, nous vous recommandons de migrer vers des IPv6 clusters si vous avez complètement épuisé tout l'espace d' IPv4 adressage disponible pour votre VPC. Évaluez les plans de soutien IPv6 de votre organisation et déterminez si l'investissement dans ce domaine IPv6 peut avoir une plus grande valeur à long terme.

L'assistance d'EKS IPv6 est axée sur la résolution du problème d'épuisement des adresses IP causé par un espace d' IPv4 adressage limité. En réponse aux problèmes d' IPv4 épuisement des clients, EKS a privilégié les pods réservés IPv6 uniquement aux pods à double pile. En d'autres termes, les pods peuvent accéder aux IPv4 ressources, mais aucune adresse ne leur est attribuée dans la plage d' IPv4 adresses CIDR VPC. Le VPC CNI attribue des IPv6 adresses aux pods à partir du bloc d'adresse CIDR VPC géré par AWS. IPv6

Calculateur de sous-réseau

Ce projet inclut un document Excel de calcul de sous-réseau. Ce document de calcul simule la consommation d'adresses IP d'une charge de travail spécifiée selon différentes options de configuration ENI, telles que `WARM_IP_TARGET` et`WARM_ENI_TARGET`. Le document comprend deux feuilles, une première pour le mode Warm ENI et une seconde pour le mode Warm IP. Consultez le guide VPC CNI pour plus d'informations sur ces modes.

Entrées :

- Taille CIDR du sous-réseau
- Cible ENI chaude ou cible IP chaude
- Liste des instances
  - type, nombre et nombre de pods de charge de travail planifiés par instance

Sorties :

- Nombre total de pods hébergés
- Nombre de sous-réseaux consommés IPs
- Nombre de sous-réseaux restants IPs
- Détails au niveau de l'instance
  - Nombre de Warm IPs/ENIs par instance
  - Nombre d'actifs IPs/ENIs par instance

Considérations relatives au VPC et aux sous-réseaux

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

L'exploitation d'un cluster EKS nécessite une connaissance du réseau VPC AWS, en plus du réseau Kubernetes.

Nous vous recommandons de comprendre les mécanismes de communication du plan de contrôle EKS avant de commencer à concevoir votre VPC ou à déployer des clusters dans des clusters existants. VPCs

Reportez-vous aux considérations relatives aux VPC de cluster et aux considérations relatives aux groupes de sécurité Amazon EKS lorsque vous concevez l'architecture d'un VPC et de sous-réseaux à utiliser avec EKS.

Présentation de

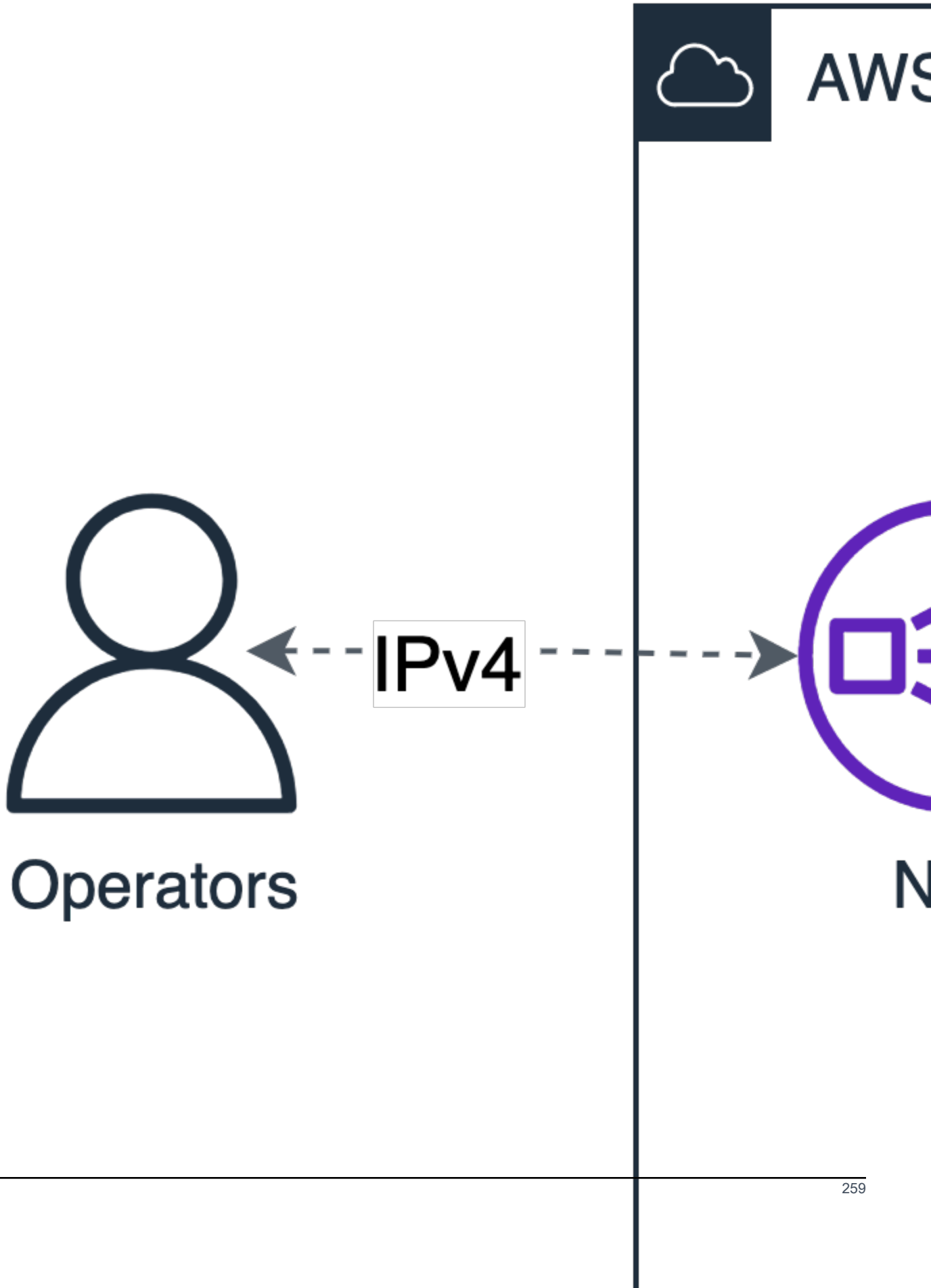
Architecture du cluster EKS

Un cluster EKS est composé de deux VPCs :

- VPC géré par AWS qui héberge le plan de contrôle Kubernetes. Ce VPC n'apparaît pas dans le compte client.
- VPC géré par le client qui héberge les nœuds Kubernetes. C'est là que s'exécutent les conteneurs, ainsi que d'autres infrastructures AWS gérées par le client, telles que les équilibreurs de charge utilisés par le cluster. Ce VPC apparaît dans le compte client. Vous devez créer un VPC géré par le client avant de créer un cluster. L'eksctl crée un VPC si vous n'en fournissez pas.

Les nœuds du VPC client doivent pouvoir se connecter au point de terminaison du serveur d'API géré dans le VPC AWS. Cela permet aux nœuds de s'enregistrer auprès du plan de contrôle Kubernetes et de recevoir des demandes pour exécuter des pods d'applications.

Les nœuds se connectent au plan de contrôle EKS via (a) un point de terminaison public EKS ou (b) une interface réseau élastique entre comptes (X-ENI) gérée par EKS. Lorsqu'un cluster est créé, vous devez spécifier au moins deux sous-réseaux VPC. EKS place un X-ENI dans chaque sous-réseau spécifié lors de la création du cluster (également appelé sous-réseaux du cluster). Le serveur d'API Kubernetes utilise ces comptes croisés ENIs pour communiquer avec les nœuds déployés sur les sous-réseaux VPC du cluster gérés par le client.


AWS
IPv4
Operators

Au démarrage du nœud, le script de démarrage EKS est exécuté et les fichiers de configuration du nœud Kubernetes sont installés. Dans le cadre du processus de démarrage de chaque instance, les agents d'exécution du conteneur, Kubelet et les agents de nœud Kubernetes sont lancés.

Pour enregistrer un nœud, Kubelet contacte le point de terminaison du cluster Kubernetes. Il établit une connexion avec le point de terminaison public extérieur au VPC ou avec le point de terminaison privé au sein du VPC. Kubelet reçoit des instructions d'API et fournit régulièrement des mises à jour de statut et des pulsations cardiaques au terminal.

Communication par plan de contrôle EKS

EKS dispose de deux méthodes pour contrôler l'accès au point de terminaison du cluster. Le contrôle d'accès aux points de terminaison vous permet de choisir si le point de terminaison est accessible depuis l'Internet public ou uniquement via votre VPC. Vous pouvez activer le point de terminaison public (qui est le point de terminaison par défaut), le point de terminaison privé ou les deux à la fois.

La configuration du point de terminaison de l'API du cluster détermine le chemin emprunté par les nœuds pour communiquer avec le plan de contrôle. Notez que ces paramètres de point de terminaison peuvent être modifiés à tout moment via la console ou l'API EKS.

Point de terminaison public

Il s'agit du comportement par défaut pour les nouveaux clusters Amazon EKS. Lorsque seul le point de terminaison public du cluster est activé, les demandes d'API Kubernetes provenant du VPC de votre cluster (telles que la communication entre le nœud de travail et le plan de contrôle) quittent le VPC, mais pas le réseau d'Amazon. Pour que les nœuds puissent se connecter au plan de contrôle, ils doivent disposer d'une adresse IP publique et d'une route vers une passerelle Internet ou d'une route vers une passerelle NAT où ils peuvent utiliser l'adresse IP publique de la passerelle NAT.

Point de terminaison public et privé

Lorsque les points de terminaison publics et privés sont activés, les demandes d'API Kubernetes provenant du VPC communiquent avec le plan de contrôle via le X au sein de votre VPC. ENIs Le serveur d'API de votre cluster est accessible depuis internet.

Point de terminaison privé

Il n'y a pas d'accès public à votre serveur API depuis Internet lorsque seul un point de terminaison privé est activé. Tout le trafic vers le serveur d’API du cluster doit provenir de votre VPC ou d’un réseau connecté. Les nœuds communiquent avec le serveur API via X- ENIs au sein de votre VPC. Notez que les outils de gestion de cluster doivent avoir accès au point de terminaison privé.

Découvrez comment vous connecter à un point de terminaison de cluster Amazon EKS privé depuis l'extérieur d'Amazon VPC.

Notez que le point de terminaison du serveur API du cluster est résolu par les serveurs DNS publics en une adresse IP privée provenant du VPC. Dans le passé, le point de terminaison ne pouvait être résolu qu'à partir du VPC.

Configurations VPC

Amazon VPC prend en charge IPv4 et adresse. IPv6 Amazon EKS est compatible IPv4 par défaut. Un bloc IPv4 CIDR doit être associé à un VPC. Vous pouvez éventuellement associer plusieurs blocs de routage interdomaines IPv4 sans classe (CIDR) et plusieurs blocs IPv6 CIDR à votre VPC. Lorsque vous créez un VPC, vous devez spécifier un bloc IPv4 CIDR pour le VPC à partir des plages d' IPv4 adresses privées spécifiées dans la RFC 1918. La taille de bloc autorisée se situe entre un `/16` préfixe (65 536 adresses IP) et un `/28` préfixe (16 adresses IP).

Lorsque vous créez un nouveau VPC, vous pouvez joindre un seul bloc IPv6 CIDR, et jusqu'à cinq lorsque vous modifiez un VPC existant. La longueur du préfixe de la taille du bloc IPv6 CIDR peut être comprise entre /44 et /60 et pour les IPv6 sous-réseaux, elle peut être comprise entre /44/ et /64. Vous pouvez demander un bloc IPv6 CIDR à partir du pool d' IPv6 adresses géré par Amazon. Reportez-vous à la section Blocs d'adresse CIDR VPC du guide de l'utilisateur VPC pour plus d'informations.

Les clusters Amazon EKS prennent en charge à la fois IPv4 et IPv6. Par défaut, les clusters EKS utilisent l' IPv4 adresse IP. La spécification IPv6 au moment de la création du cluster permettra d'utiliser les IPv6 clusters. IPv6 les clusters nécessitent une double pile VPCs et des sous-réseaux.

Amazon EKS vous recommande d'utiliser au moins deux sous-réseaux situés dans des zones de disponibilité différentes lors de la création du cluster. Les sous-réseaux que vous transmettez lors de la création du cluster sont appelés sous-réseaux du cluster. Lorsque vous créez un cluster, Amazon EKS crée jusqu'à 4 comptes croisés (comptes x ou x-ENIs) ENIs dans les sous-réseaux que vous spécifiez. Les x- ENIs sont toujours déployés et sont utilisés pour le trafic d'administration du cluster tel que la livraison des journaux, l'exécution et le proxy. Reportez-vous au guide de l'utilisateur EKS pour obtenir des informations complètes sur les exigences relatives aux VPC et aux sous-réseaux.

Les nœuds de travail Kubernetes peuvent s'exécuter dans les sous-réseaux du cluster, mais cela n'est pas recommandé. Lors des mises à niveau du cluster, Amazon EKS fournit des ENIs ressources supplémentaires dans les sous-réseaux du cluster. Lorsque votre cluster prend de l'ampleur, les nœuds de travail et les pods peuvent consommer le contenu disponible IPs dans le

sous-réseau du cluster. Par conséquent, afin de vous assurer qu'il y en a suffisamment, IPs vous pouvez envisager d'utiliser des sous-réseaux de cluster dédiés avec le masque de réseau /28.

Les nœuds de travail Kubernetes peuvent s'exécuter dans un sous-réseau public ou privé. Le caractère public ou privé d'un sous-réseau indique si le trafic au sein du sous-réseau est acheminé via une passerelle Internet. Les sous-réseaux publics disposent d'une entrée de table de routage vers Internet via la passerelle Internet, mais pas les sous-réseaux privés.

Le trafic qui provient d'un autre endroit et atteint vos nœuds est appelé « entrée ». Le trafic provenant des nœuds et quittant le réseau est appelé « sortie ». Les nœuds dotés d'adresses IP publiques ou élastiques (EIPs) au sein d'un sous-réseau configuré avec une passerelle Internet autorisent l'entrée depuis l'extérieur du VPC. Les sous-réseaux privés ont généralement un routage vers une passerelle NAT, qui n'autorise pas le trafic entrant vers les nœuds des sous-réseaux depuis l'extérieur du VPC tout en permettant au trafic provenant des nœuds de quitter le VPC (sortie).

Dans le IPv6 monde, chaque adresse est routable sur Internet. Les IPv6 adresses associées aux nœuds et aux pods sont publiques. Les sous-réseaux privés sont pris en charge par la mise en œuvre de passerelles Internet de sortie uniquement (EIGW) dans un VPC, autorisant le trafic sortant tout en bloquant tout le trafic entrant. Les meilleures pratiques pour implémenter IPv6 des sous-réseaux sont disponibles dans le guide de l'utilisateur du VPC.

Vous pouvez configurer le VPC et les sous-réseaux de trois manières différentes :

Utiliser uniquement des sous-réseaux publics

Dans les mêmes sous-réseaux publics, les nœuds et les ressources d'entrée (telles que les équilibreurs de charge) sont créés. Ajoutez une balise au sous-réseau public `kubernetes.io/role/elb`pour créer des équilibreurs de charge adaptés à Internet. Dans cette configuration, le point de terminaison du cluster peut être configuré pour être public, privé ou les deux (public et privé).

Utilisation de sous-réseaux privés et publics

Les nœuds sont créés sur des sous-réseaux privés, tandis que les ressources d'entrée sont instanciées dans des sous-réseaux publics. Vous pouvez activer l'accès public, privé ou les deux (public et privé) au point de terminaison du cluster. Selon la configuration du point de terminaison du cluster, le trafic du nœud entrera via la passerelle NAT ou l'ENI.

Utiliser uniquement des sous-réseaux privés

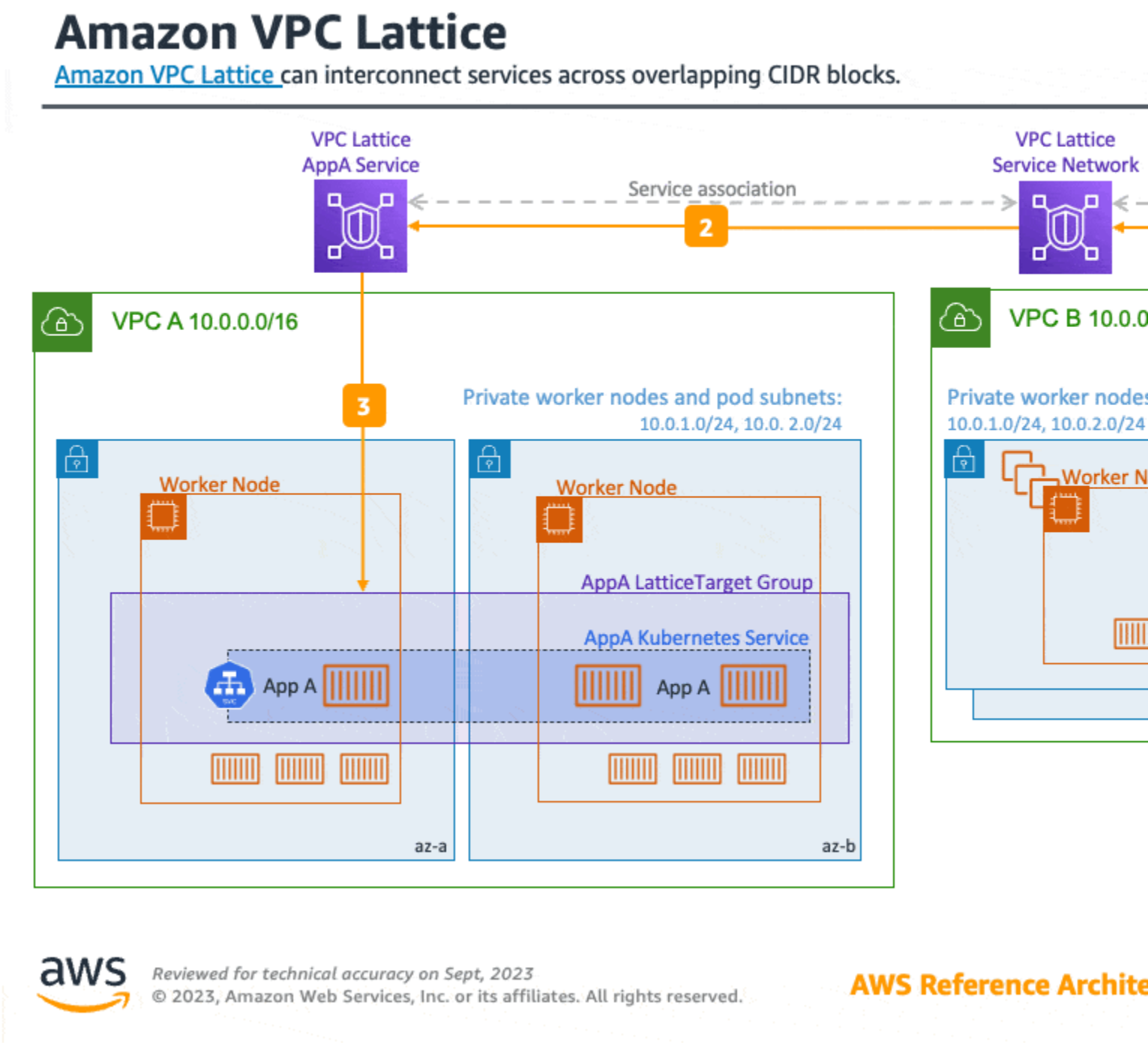
Les nœuds et les entrées sont créés dans des sous-réseaux privés. Utilisation de la balise de `kubernetes.io/role/internal-elb`sous-réseau pour créer des équilibreurs de charge

internes. L'accès au point de terminaison de votre cluster nécessitera une connexion VPN. Vous devez activer AWS PrivateLink pour EC2 et tous les référentiels Amazon ECR et S3. Seul le point de terminaison privé du cluster doit être activé. Nous vous suggérons de passer en revue les exigences relatives aux clusters privés EKS avant de les approvisionner.

Communication à travers VPCs

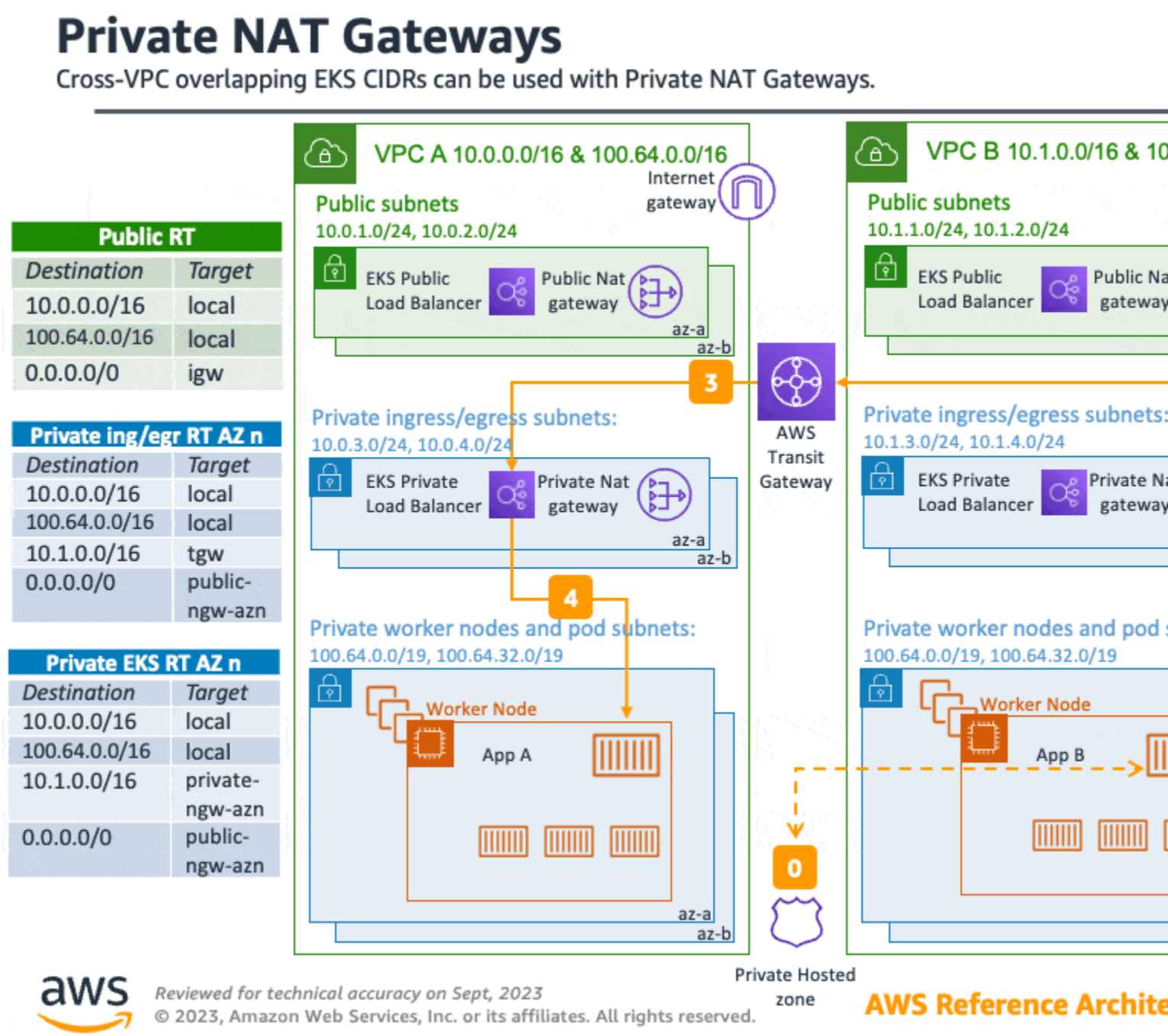
Il existe de nombreux scénarios dans lesquels vous avez besoin VPCs de déployer plusieurs clusters EKS distincts sur ces clusters VPCs.

Vous pouvez utiliser Amazon VPC Lattice pour connecter des services de manière cohérente et sécurisée entre plusieurs VPCs comptes (sans avoir à fournir de connectivité supplémentaire par des services tels que le peering VPC, AWS ou AWS PrivateLink Transit Gateway). Pour en savoir plus, cliquez ici.



Amazon VPC Lattice fonctionne dans l'espace d'adressage lien-local dans IPv4 et IPv6, fournissant une connectivité entre les services dont les adresses peuvent se chevaucher. IPv4 Pour des raisons d'efficacité opérationnelle, nous recommandons vivement de déployer des clusters et des nœuds EKS sur des plages d'adresses IP qui ne se chevauchent pas. Si votre infrastructure comprend des plages VPCs d'adresses IP qui se chevauchent, vous devez concevoir votre réseau en conséquence.

Nous suggérons une passerelle NAT privée, ou VPC CNI en mode réseau personnalisé, associée à une passerelle de transit pour intégrer les charges de travail sur EKS afin de résoudre les problèmes CIDR qui se chevauchent tout en préservant les adresses IP routables. RFC1918

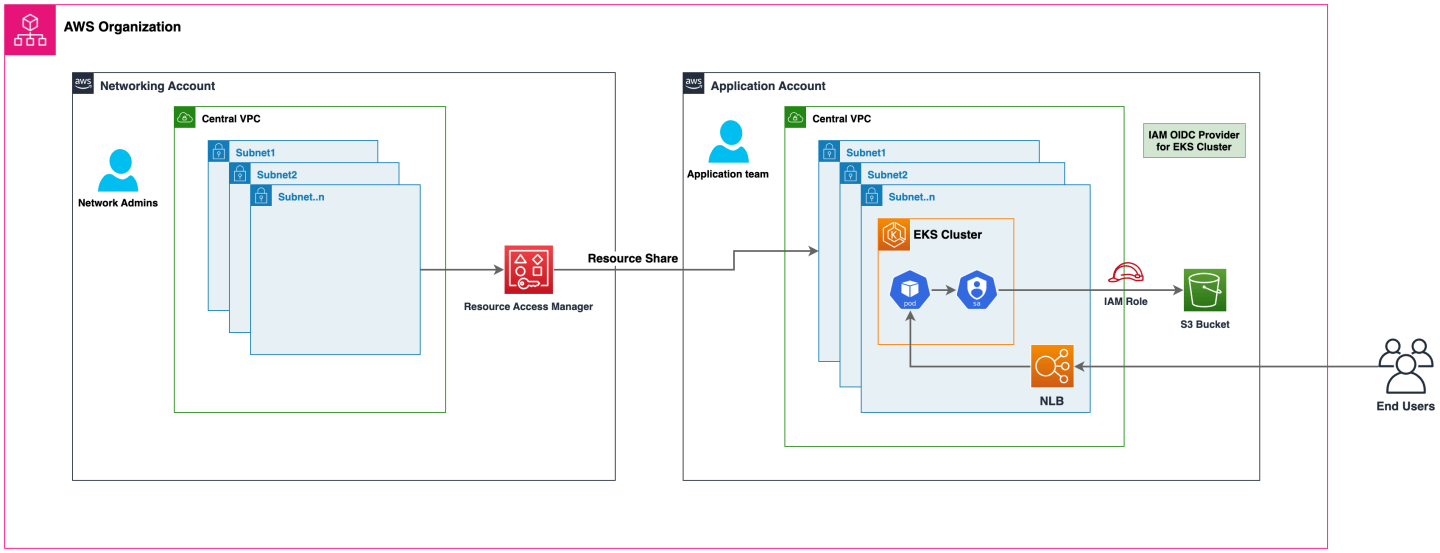


Envisagez d'utiliser AWS PrivateLink, également connu sous le nom de service de point de terminaison, si vous êtes le fournisseur de services et souhaitez partager votre service Kubernetes et votre entrée (ALB ou NLB) avec le VPC de votre client sur des comptes séparés.

Partage d'un VPC entre plusieurs comptes

De nombreuses entreprises ont adopté Amazon partagé VPCs afin de rationaliser l'administration du réseau, de réduire les coûts et d'améliorer la sécurité de plusieurs comptes AWS au sein d'une organisation AWS. Ils utilisent AWS Resource Access Manager (RAM) pour partager en toute sécurité les ressources AWS prises en charge avec des comptes AWS individuels, des unités organisationnelles (OUs) ou l'ensemble de l'organisation AWS.

Vous pouvez déployer des clusters Amazon EKS, des groupes de nœuds gérés et d'autres ressources AWS de support (telles que des groupes de sécurité LoadBalancers, des points de terminaison, etc.) dans des sous-réseaux VPC partagés à partir d'un autre compte AWS utilisant la RAM AWS. La figure ci-dessous illustre un exemple d'architecture de haut niveau. Cela permet aux équipes réseau centrales de contrôler les structures réseau telles que les sous-réseaux VPCs, etc., tout en permettant aux équipes chargées des applications ou des plateformes de déployer des clusters Amazon EKS dans leurs comptes AWS respectifs. Une présentation complète de ce scénario est disponible dans ce dépôt github.



Considérations relatives à l'utilisation de sous-réseaux partagés

- Les clusters et nœuds de travail Amazon EKS peuvent être créés au sein de sous-réseaux partagés faisant tous partie du même VPC. Amazon EKS ne prend pas en charge la création de clusters sur plusieurs clusters VPCs.

- Amazon EKS utilise les groupes de sécurité VPC AWS (SGs) pour contrôler le trafic entre le plan de contrôle Kubernetes et les nœuds de travail du cluster. Les groupes de sécurité sont également utilisés pour contrôler le trafic entre les nœuds de travail, les autres ressources VPC et les adresses IP externes. Vous devez créer ces groupes de sécurité dans le application/participant compte. Assurez-vous que les groupes de sécurité que vous avez l'intention d'utiliser pour vos modules se trouvent également dans le compte du participant. Vous pouvez configurer les règles entrantes et sortantes au sein de vos groupes de sécurité pour autoriser le trafic nécessaire à destination et en provenance des groupes de sécurité situés dans le compte Central VPC.
- Créez des rôles IAM et des politiques associées dans le compte de participant sur lequel réside votre cluster Amazon EKS. Ces rôles et politiques IAM sont essentiels pour accorder les autorisations nécessaires aux clusters Kubernetes gérés par Amazon EKS, ainsi qu'aux nœuds et aux pods exécutés sur Fargate. Les autorisations permettent à Amazon EKS de passer des appels vers d'autres services AWS en votre nom.
- Vous pouvez suivre les approches suivantes pour autoriser l'accès entre comptes aux ressources AWS telles que les compartiments Amazon S3, les tables Dynamodb, etc., à partir des pods k8s :
  - Approche politique basée sur les ressources : si le service AWS prend en charge les politiques relatives aux ressources, vous pouvez ajouter une politique basée sur les ressources appropriée pour autoriser l'accès entre comptes aux rôles IAM attribués aux pods Kubernetes. Dans ce scénario, le fournisseur OIDC, les rôles IAM et les politiques d'autorisation existent dans le compte de l'application. Pour trouver les services AWS qui prennent en charge les politiques basées sur les ressources, consultez les services AWS qui fonctionnent avec IAM et recherchez les services dont la valeur est « Oui » dans la colonne « Basé sur les ressources ».
  - Approche du fournisseur OIDC : les ressources IAM telles que le fournisseur OIDC, les rôles IAM, les politiques d'autorisation et de confiance seront créées sur le compte AWS des autres participants où les ressources existent. Ces rôles seront attribués aux pods Kubernetes dans le compte de l'application, afin qu'ils puissent accéder aux ressources entre comptes. Consultez le blog sur les rôles IAM entre comptes pour les comptes de service Kubernetes pour une présentation complète de cette approche.
- Vous pouvez déployer les ressources Amazon Elastic Loadbalancer (ELB) (ALB ou NLB) pour acheminer le trafic vers les pods k8s via des applications ou des comptes réseau centraux. Reportez-vous à la procédure pas à pas d'Expose Amazon EKS Pods Through Cross-Account Load Balancer pour obtenir des instructions détaillées sur le déploiement des ressources ELB sur un compte réseau central. Cette option offre une flexibilité accrue, car elle permet au compte Central Networking de contrôler totalement la configuration de sécurité des ressources Load Balancer.

- Lorsque vous utilisez `custom networking feature` Amazon VPC CNI, vous devez utiliser les mappages d'ID de zone de disponibilité (AZ) répertoriés dans le compte réseau central pour les créer. `ENIConfig` Cela est dû au mappage aléatoire des noms physiques AZs avec les noms AZ dans chaque compte AWS.

Groupes de sécurité

Un groupe de sécurité contrôle le trafic autorisé à atteindre et à quitter les ressources auxquelles il est associé. Amazon EKS utilise des groupes de sécurité pour gérer la communication entre le plan de contrôle et les nœuds. Lorsque vous créez un cluster, Amazon EKS crée un groupe de sécurité nommé `eks-cluster-sg-my-cluster-uniqueID`. EKS associe ces groupes de sécurité aux nœuds gérés ENIs et aux nœuds. Les règles par défaut permettent à tout le trafic de circuler librement entre votre cluster et vos nœuds, et autorise tout le trafic sortant vers n'importe quelle destination.

Lorsque vous créez un cluster, vous pouvez définir vos propres groupes de sécurité. Consultez les recommandations relatives aux groupes de sécurité lorsque vous spécifiez vos propres groupes de sécurité.

Recommandations

Envisagez un déploiement multi-AZ

Les régions AWS fournissent plusieurs zones de disponibilité (AZ) physiquement séparées et isolées, connectées par un réseau à faible latence, à haut débit et hautement redondant. Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications qui basculent automatiquement entre les zones de disponibilité sans interruption. Amazon EKS recommande vivement de déployer des clusters EKS dans plusieurs zones de disponibilité. Pensez à spécifier des sous-réseaux dans au moins deux zones de disponibilité lorsque vous créez le cluster.

Kubelet exécuté sur les nœuds ajoute automatiquement des étiquettes à l'objet du nœud, telles que. `topology.kubernetes.io/region=us-west-2` Nous recommandons d'utiliser des étiquettes de nœuds en conjonction avec les contraintes de propagation de la topologie des pods pour contrôler la manière dont les pods sont répartis entre les zones. Ces conseils permettent au planificateur Kubernetes de placer des pods pour une meilleure disponibilité attendue, réduisant ainsi le risque qu'une panne corrélée affecte l'ensemble de votre charge de travail. Reportez-vous à la section Affectation de nœuds aux pods pour voir des exemples de restrictions relatives au sélecteur de nœuds et à la répartition AZ.

Vous pouvez définir les sous-réseaux ou les zones de disponibilité lorsque vous créez des nœuds. Les nœuds sont placés dans des sous-réseaux de clusters si aucun sous-réseau n'est configuré. La prise en charge par EKS des groupes de nœuds gérés répartit automatiquement les nœuds sur plusieurs zones de disponibilité en fonction de la capacité disponible. Karpenter respectera le placement du spread AZ en dimensionnant les nœuds pour indiquer AZs si les charges de travail définissent des limites de propagation topologique.

Les AWS Elastic Load Balancers sont gérés par le AWS Load Balancer Controller pour un cluster Kubernetes. Il fournit un Application Load Balancer (ALB) pour les ressources d'entrée de Kubernetes et un Network Load Balancer (NLB) pour les services Kubernetes de type Loadbalancer. Le contrôleur Elastic Load Balancer utilise des balises pour découvrir les sous-réseaux. Le contrôleur ELB nécessite au moins deux zones de disponibilité (AZs) pour approvisionner correctement les ressources d'entrée. Envisagez de configurer au moins deux sous-réseaux AZs pour tirer parti de la sécurité et de la fiabilité de la redondance géographique.

Déployer des nœuds sur des sous-réseaux privés

Un VPC comprenant à la fois des sous-réseaux privés et publics est la méthode idéale pour déployer des charges de travail Kubernetes sur EKS. Envisagez de définir au moins deux sous-réseaux publics et deux sous-réseaux privés dans deux zones de disponibilité distinctes. La table de routage associée d'un sous-réseau public contient une route vers une passerelle Internet. Les pods peuvent interagir avec Internet via une passerelle NAT. Les sous-réseaux privés sont pris en charge par les passerelles Internet de sortie uniquement dans l'environnement (EIGW). IPv6

L'instanciation de nœuds dans des sous-réseaux privés offre un contrôle maximal du trafic vers les nœuds et est efficace pour la grande majorité des applications Kubernetes. Les ressources d'entrée (comme les équilibreurs de charge) sont instanciées dans des sous-réseaux publics et acheminent le trafic vers des pods fonctionnant sur des sous-réseaux privés.

Envisagez le mode privé uniquement si vous exigez une sécurité et une isolation réseau strictes. Dans cette configuration, trois sous-réseaux privés sont déployés dans des zones de disponibilité distinctes au sein du VPC de la région AWS. Les ressources déployées sur les sous-réseaux ne peuvent pas accéder à Internet, pas plus qu'Internet ne peut accéder aux ressources des sous-réseaux. Pour que votre application Kubernetes puisse accéder à d'autres services AWS, vous devez configurer les points de terminaison de passerelle PrivateLink des interfaces and/or . Vous pouvez configurer des équilibreurs de charge internes pour rediriger le trafic vers les pods à l'aide d'AWS Load Balancer Controller. Les sous-réseaux privés doivent être marqués (`kubernetes.io/role/internal-elb: 1`) pour que le contrôleur puisse configurer les équilibreurs de charge. Pour que

les nœuds s'enregistrent auprès du cluster, le point de terminaison du cluster doit être défini sur le mode privé. Veuillez consulter le guide du cluster privé pour connaître toutes les exigences et considérations.

Envisagez le mode public et privé pour le point de terminaison du cluster

Amazon EKS propose des modes de point de public-and-private terminaison de cluster uniquement publics et privés uniquement. Le mode par défaut est public uniquement, mais nous vous recommandons de configurer le point de terminaison du cluster en mode public et privé. Cette option permet aux appels d'API Kubernetes au sein du VPC de votre cluster (tels que les node-to-control-plane communications) d'utiliser le point de terminaison et le trafic privés du VPC pour rester dans le VPC de votre cluster. Le serveur API de votre cluster, quant à lui, est accessible depuis Internet. Cependant, nous recommandons vivement de limiter les blocs CIDR qui peuvent utiliser le point de terminaison public. Découvrez comment configurer l'accès public et privé aux terminaux, notamment en limitant les blocs CIDR.

Nous vous suggérons d'utiliser un point de terminaison privé uniquement lorsque vous avez besoin de sécurité et d'isolation du réseau. Nous vous recommandons d'utiliser l'une des options répertoriées dans le guide de l'utilisateur d'EKS pour vous connecter à un serveur d'API en privé.

Configurez les groupes de sécurité avec soin

Amazon EKS prend en charge l'utilisation de groupes de sécurité personnalisés. Tous les groupes de sécurité personnalisés doivent autoriser la communication entre les nœuds et le plan de contrôle Kubernetes. Vérifiez les exigences en matière de ports et configurez les règles manuellement lorsque votre organisation n'autorise pas les communications ouvertes.

EKS applique les groupes de sécurité personnalisés que vous fournissez lors de la création du cluster aux interfaces gérées (X-ENIs). Cependant, il ne les associe pas immédiatement aux nœuds. Lors de la création de groupes de nœuds, il est vivement recommandé d'associer manuellement des groupes de sécurité personnalisés. Envisagez d'activer securityGroupSelectorles conditions pour permettre à Karpenter de découvrir les groupes de sécurité personnalisés par les modèles de nœuds lors du dimensionnement automatique des nœuds.

Nous vous recommandons vivement de créer un groupe de sécurité pour autoriser tout le trafic de communication entre nœuds. Pendant le processus d'amorçage, les nœuds ont besoin d'une connectivité Internet sortante pour accéder au point de terminaison du cluster. Évaluez les exigences en matière d'accès sortant, telles que la connexion sur site et l'accès au registre des conteneurs, et définissez des règles appropriées. Avant de mettre les modifications en production, nous vous

conseillons vivement de vérifier soigneusement les connexions dans votre environnement de développement.

Déployez des passerelles NAT dans chaque zone de disponibilité

Si vous déployez des nœuds dans des sous-réseaux privés (IPv4 et IPv6), envisagez de créer une passerelle NAT dans chaque zone de disponibilité (AZ) afin de garantir une architecture indépendante des zones et de réduire les dépenses entre les zones. Chaque passerelle NAT d'une AZ est mise en œuvre de manière redondante.

CNI Amazon VPC

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Amazon EKS implémente la mise en réseau de clusters via le plug-in Amazon VPC Container Network Interface, également connu sous le nom de VPC CNI. Le plugin CNI permet aux pods Kubernetes d'avoir la même adresse IP que sur le réseau VPC. Plus précisément, tous les

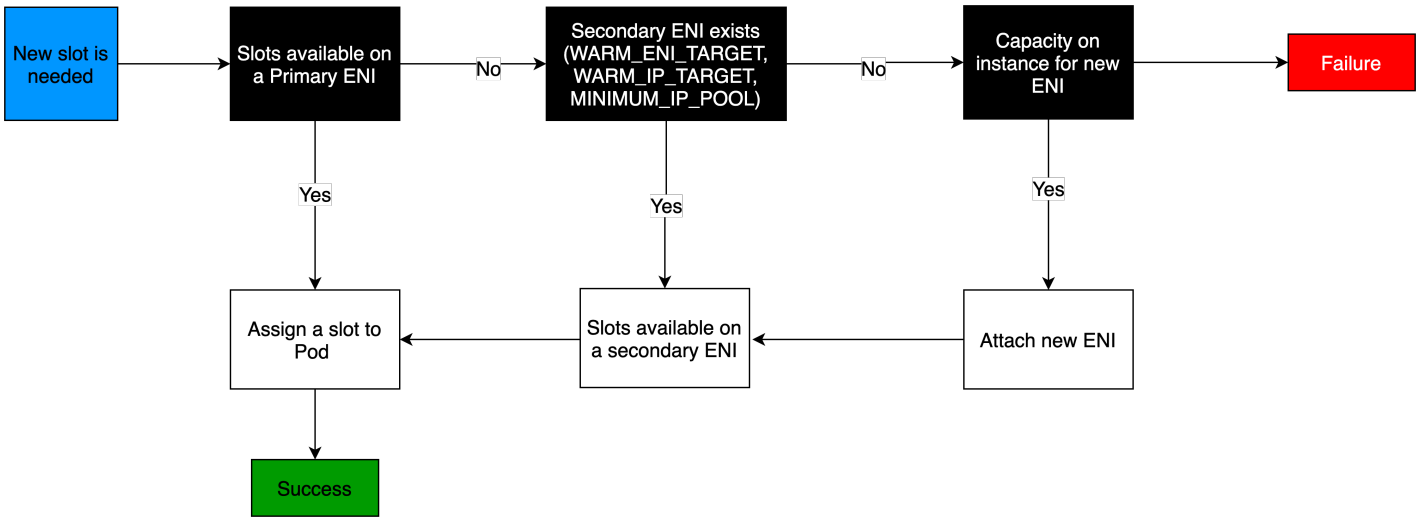
conteneurs du Pod partagent un espace de noms réseau et peuvent communiquer entre eux via des ports locaux.

Amazon VPC CNI comporte deux composants :

- CNI Binary, qui configurera le réseau Pod pour permettre Pod-to-Pod la communication. Le binaire CNI s'exécute sur le système de fichiers racine d'un nœud et est invoqué par le kubelet lorsqu'un nouveau pod est ajouté ou qu'un pod existant est supprimé du nœud.
- ipamd, un démon de gestion des adresses IP (IPAM) local aux nœuds (IPAM) de longue date, chargé de :
  - gestion ENIs sur un nœud, et
  - gestion d'un pool d'adresses IP ou de préfixes disponibles

Lorsqu'une instance est créée, EC2 crée et attache une ENI principale associée à un sous-réseau principal. Le sous-réseau principal peut être public ou privé. Les pods qui s'exécutent en mode HostNetwork utilisent l'adresse IP principale attribuée au nœud ENI principal et partagent le même espace de noms réseau que l'hôte.

Le plugin CNI gère les interfaces réseau Elastic (ENI) sur le nœud. Lorsqu'un nœud est provisionné, le plug-in CNI alloue automatiquement un pool de slots (IPs ou de préfixes) du sous-réseau du nœud à l'ENI principal. Ce pool est connu sous le nom de pool chaud et sa taille est déterminée par le type d'instance du nœud. Selon les paramètres CNI, un emplacement peut être une adresse IP ou un préfixe. Lorsqu'un emplacement a été attribué à un ENI, le CNI peut en attacher des emplacements supplémentaires aux nœuds ENIs avec un pool chaud. Ces éléments supplémentaires ENIs sont appelés secondaires ENIs. Chaque ENI ne peut prendre en charge qu'un certain nombre de slots, en fonction du type d'instance. Le CNI attache davantage ENIs aux instances en fonction du nombre de slots nécessaires, qui correspond généralement au nombre de pods. Ce processus se poursuit jusqu'à ce que le nœud ne puisse plus prendre en charge d'ENI supplémentaire. Le CNI préalloue également du « warm » ENIs et des emplacements pour accélérer le démarrage du Pod. Notez que chaque type d'instance possède un nombre maximum de ENIs pièces pouvant être attachées. Il s'agit d'une contrainte sur la densité des pods (nombre de pods par nœud), en plus des ressources de calcul.



Le nombre maximal d'interfaces réseau et le nombre maximal de slots que vous pouvez utiliser varient selon le type d'instance EC2. Étant donné que chaque pod consomme une adresse IP sur un slot, le nombre de pods que vous pouvez exécuter sur une instance EC2 particulière dépend du nombre d'emplacements ENIs pouvant y être attachés et du nombre d'emplacements pris en charge par chaque ENI. Nous vous suggérons de définir le nombre maximum de pods par EKS dans le guide de l'utilisateur afin d'éviter d'épuiser les ressources du processeur et de la mémoire de l'instance. Les pods `hostNetwork` utilisés sont exclus de ce calcul. Vous pouvez envisager d'utiliser un script appelé max-pod-calculator.sh pour calculer le nombre maximal de pods recommandé par EKS pour un type d'instance donné.

Présentation de

Le mode IP secondaire est le mode par défaut pour le VPC CNI. Ce guide fournit un aperçu générique du comportement du VPC CNI lorsque le mode IP secondaire est activé. La fonctionnalité d'ipamd (allocation d'adresses IP) peut varier en fonction des paramètres de configuration du VPC CNI, the section called “Mode préfixe pour Linux” tels quethe section called “Groupes de sécurité par pod”, et. the section called “Mise en réseau personnalisée”

L'Amazon VPC CNI est déployé en tant que daemonset Kubernetes nommé aws-node sur les nœuds de travail. Lorsqu'un nœud de travail est provisionné, une ENI par défaut, appelée ENI principale, lui est attachée. Le CNI alloue un pool chaud d'adresses IP secondaires à partir du sous-réseau attaché à l'ENI principal du nœud. ENIs Par défaut, ipamd tente d'attribuer une ENI supplémentaire au nœud. L'IPAMD alloue des ENI supplémentaires lorsqu'un seul pod est planifié et qu'une adresse IP secondaire est attribuée par l'ENI principal. Cet ENI « chaud » permet une mise en réseau plus

rapide des Pod. Lorsque le pool d'adresses IP secondaires s'épuise, le CNI ajoute une autre ENI pour en attribuer d'autres.

Le nombre ENIs et les adresses IP d'un pool sont configurés via des variables d'environnement appelées WARM_ENI_TARGET, WARM_IP_TARGET, MINIMUM_IP_TARGET. Le `aws-node` Daemonset vérifiera périodiquement qu'un nombre suffisant de fichiers ENIs sont connectés. Un nombre suffisant de ENIs sont joints lorsque toutes `WARM_IP_TARGET` les `MINIMUM_IP_TARGET` conditions sont remplies. `WARM_ENI_TARGET` Si le nombre d'ENI attachés est insuffisant, le CNI appellera l'API EC2 pour en attacher d'autres jusqu'à ce que la `MAX_ENI` limite soit atteinte.

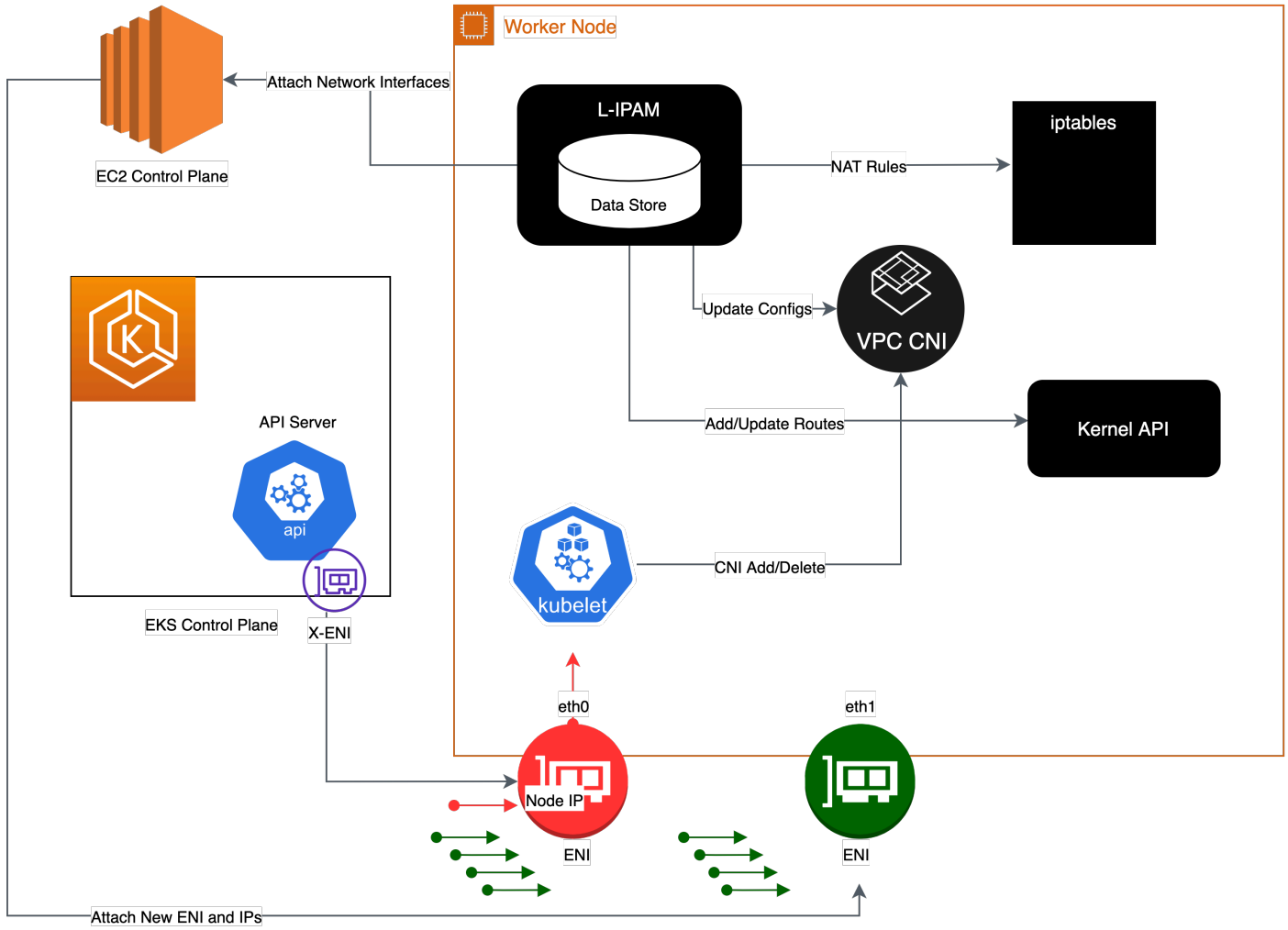
- `WARM_ENI_TARGET`- Entier, les valeurs supérieures à 0 indiquent que l'exigence est activée
  - Le nombre de Warm ENIs à maintenir. Une ENI est « chaude » lorsqu'elle est attachée en tant qu'ENI secondaire à un nœud, mais elle n'est utilisée par aucun Pod. Plus précisément, aucune adresse IP de l'ENI n'a été associée à un Pod.
  - Exemple : considérez une instance avec 2 ENIs, chaque ENI prenant en charge 5 adresses IP. WARM_ENI_TARGET est défini sur 1. Si exactement 5 adresses IP sont associées à l'instance, le CNI en conserve 2 ENIs attachées à l'instance. La première ENI est en cours d'utilisation, et les 5 adresses IP possibles de cette ENI sont utilisées. Le deuxième ENI est « chaud » avec les 5 adresses IP dans le pool. Si un autre Pod est lancé sur l'instance, une 6ème adresse IP sera nécessaire. Le CNI attribuera à ce 6e pod une adresse IP provenant du deuxième ENI et 5 IPs du pool. Le deuxième ENI est maintenant utilisé et n'est plus dans un état « chaud ». Le CNI attribuera une 3e ENI pour maintenir au moins une ENI chaude.

> **Note**
>
> Le warm consomme ENIs toujours les adresses IP du CIDR de votre VPC. Les adresses IP sont « inutilisées » ou « chaudes » jusqu'à ce qu'elles soient associées à une charge de travail, telle qu'un pod.

- `WARM_IP_TARGET`, Entier, les valeurs supérieures à 0 indiquent que l'exigence est activée
  - Le nombre d'adresses IP chaudes à conserver. Une adresse IP chaude est disponible sur un ENI connecté activement, mais elle n'a pas été attribuée à un pod. En d'autres termes, le nombre de Warm IPs disponibles est le nombre IPs qui peut être attribué à un Pod sans nécessiter d'ENI supplémentaire.

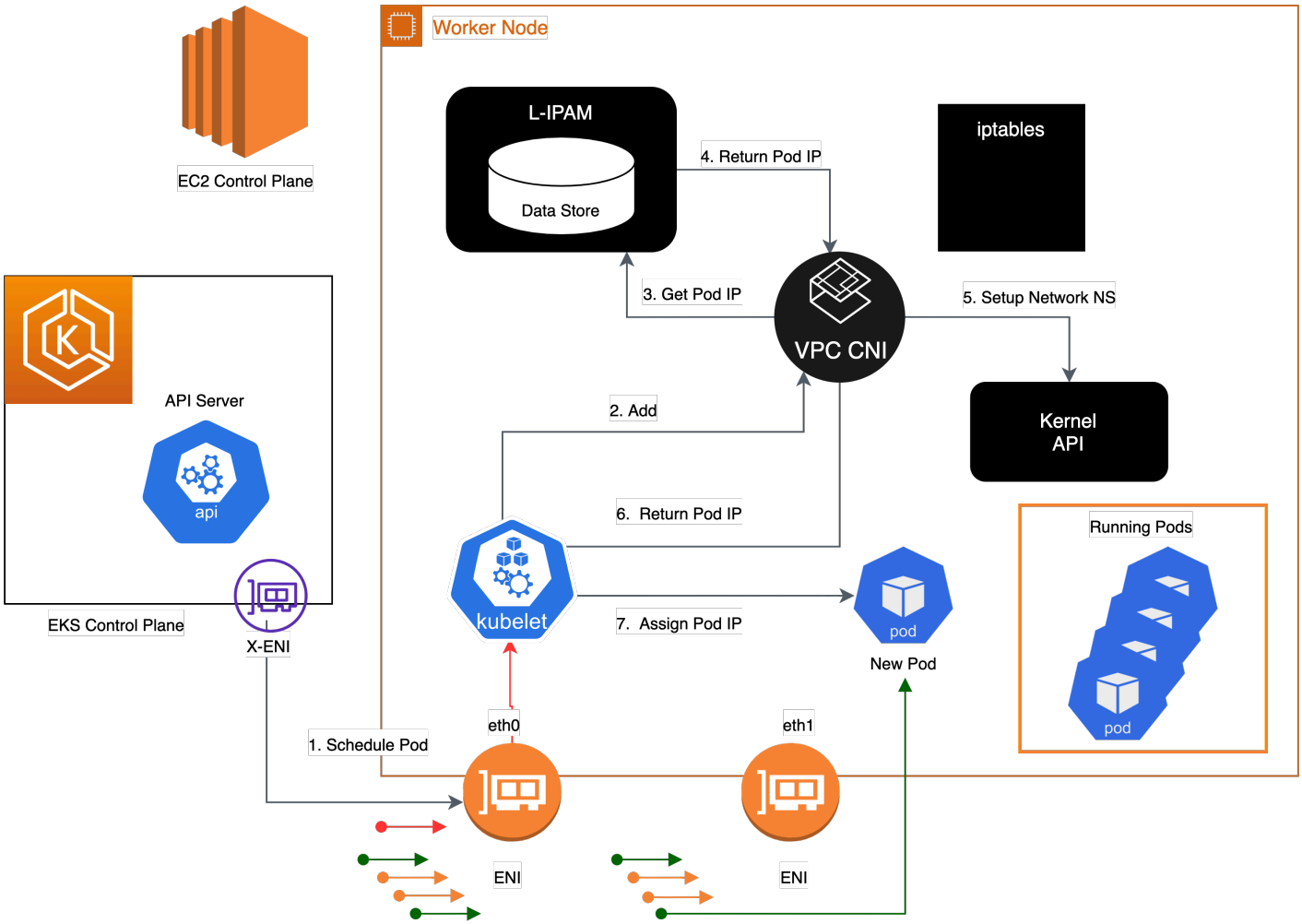
- Exemple : considérez une instance avec 1 ENI, chaque ENI prenant en charge 20 adresses IP. WARM_IP_TARGET est défini sur 5. WARM_ENI_TARGET est défini sur 0. Une seule ENI sera attachée jusqu'à ce qu'une 16e adresse IP soit nécessaire. Ensuite, le CNI attachera un deuxième ENI, consommant 20 adresses possibles du sous-réseau CIDR.
- `MINIMUM_IP_TARGET`, Entier, les valeurs supérieures à 0 indiquent que l'exigence est activée
  - Le nombre minimum d'adresses IP à attribuer à tout moment. Ceci est couramment utilisé pour précharger l'attribution de plusieurs ENIs lors du lancement de l'instance.
  - Exemple : considérez une instance récemment lancée. Il possède 1 ENI et chaque ENI prend en charge 10 adresses IP. MINIMUM_IP_TARGET est défini sur 100. L'ENI joint immédiatement 9 adresses supplémentaires ENIs pour un total de 100 adresses. Cela se produit indépendamment des valeurs WARM_IP_TARGET ou WARM_ENI_TARGET.

Ce projet inclut un document Excel de calcul de sous-réseau. Ce document de calcul simule la consommation d'adresses IP d'une charge de travail spécifiée selon différentes options de configuration ENI, telles que `WARM_IP_TARGET` et`WARM_ENI_TARGET`.

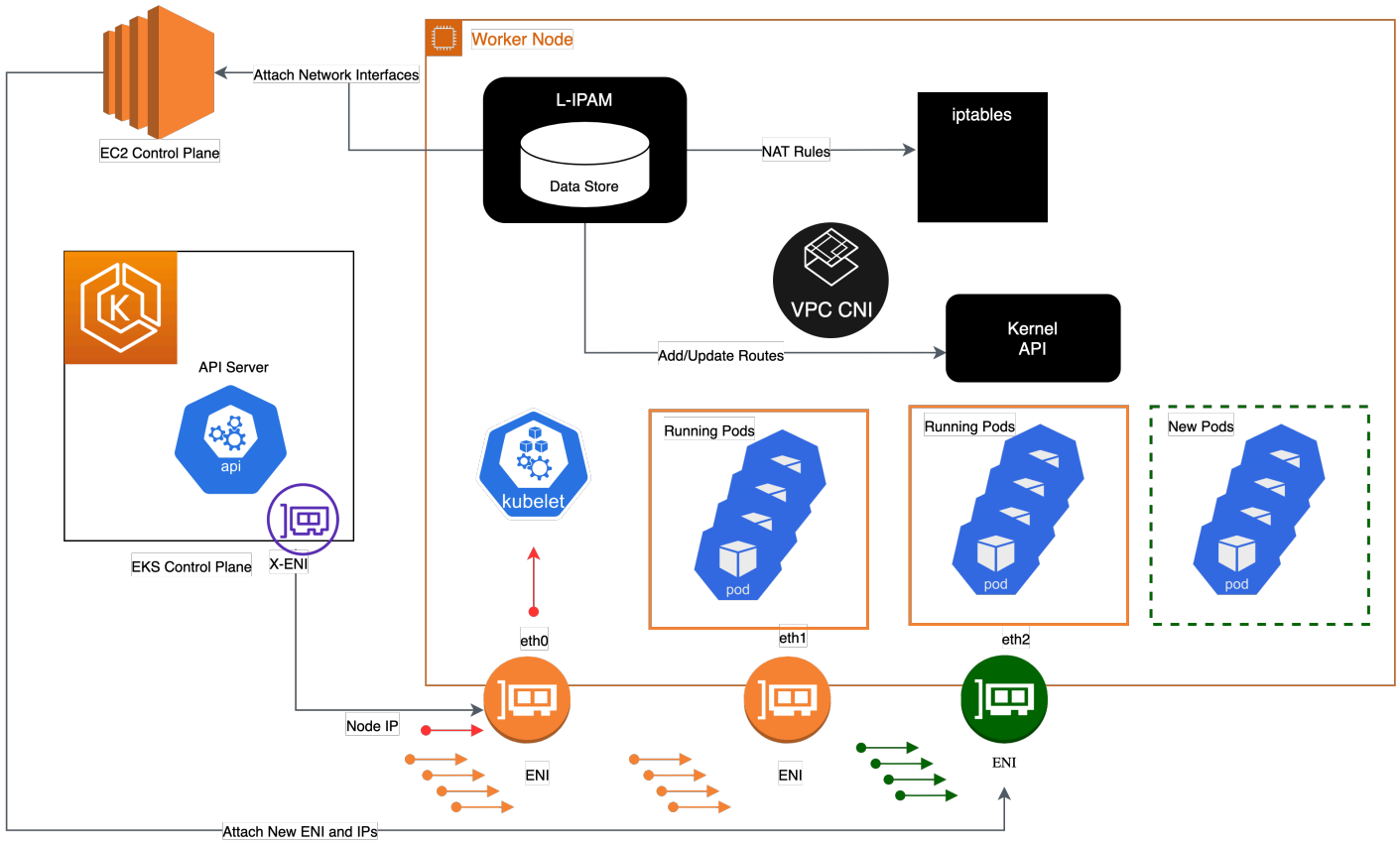


Lorsque Kubelet reçoit une demande d'ajout de Pod, le binaire CNI demande à ipamd une adresse IP disponible, qu'ipamd fournit ensuite au Pod. Le binaire CNI connecte le réseau hôte et le réseau Pod.

Les pods déployés sur un nœud sont, par défaut, affectés aux mêmes groupes de sécurité que l'ENI principal. Les pods peuvent également être configurés avec différents groupes de sécurité.



Lorsque le groupe d'adresses IP est épuisé, le plugin attache automatiquement une autre interface réseau Elastic à l'instance et alloue un autre ensemble d'adresses IP secondaires à cette interface. Ce processus se poursuit jusqu'à ce que le nœud ne puisse plus prendre en charge des interfaces réseau Elastic supplémentaires.



Lorsqu'un pod est supprimé, le VPC CNI place l'adresse IP du pod dans un cache de refroidissement de 30 secondes. Les IPs fichiers placés dans un cache de refroidissement ne sont pas attribués aux nouveaux pods. Lorsque la période de refroidissement est terminée, le VPC CNI déplace le Pod IP vers le pool chaud. La période de refroidissement empêche le recyclage prématuré des adresses IP des pods et permet à kube-proxy de terminer la mise à jour des règles iptables sur tous les nœuds du cluster. Lorsque le nombre de paramètres du pool de chauffage est ENIs supérieur IPs ou égal à ce nombre, le plugin ipamd revient IPs dans le ENIs VPC.

Comme décrit ci-dessus dans le mode IP secondaire, chaque pod reçoit une adresse IP privée secondaire de l'un des ENI attachés à une instance. Étant donné que chaque pod utilise une adresse IP, le nombre de pods que vous pouvez exécuter sur une instance EC2 particulière dépend du nombre de pods ENIs pouvant y être attachés et du nombre d'adresses IP qu'elle prend en charge. Le VPC CNI vérifie le fichier de limites pour savoir combien ENIs d'adresses IP sont autorisées pour chaque type d'instance.

Vous pouvez utiliser la formule suivante pour déterminer le nombre maximum de pods que vous pouvez déployer sur un nœud.

```
(Number of network interfaces for the instance type * (the number of IP addresses per
 network interface - 1)) + 2
```

Le +2 indique les pods qui nécessitent un réseau hôte, tel que kube-proxy et VPC CNI. Amazon EKS nécessite que kube-proxy et VPC CNI fonctionnent sur chaque nœud, et ces exigences sont prises en compte dans la valeur max-pods. Si vous souhaitez exécuter des pods de mise en réseau hôte supplémentaires, pensez à mettre à jour la valeur max-pods. Vous pouvez les spécifier `--kubelet-extra-args "–max-pods=110"` en tant que données utilisateur dans le modèle de lancement.

Par exemple, sur un cluster comportant 3 nœuds c5.large (3 ENIs et 10 au maximum IPs par ENI), lorsque le cluster démarre et dispose de 2 pods CoreDNS, le CNI consomme 50 adresses IP et en conserve 43 dans le warm pool. IPs Le pool de chaleur permet de lancer le Pod plus rapidement lorsque l'application est déployée.

Nœud 1 (avec module CoreDNS) : ENIs 2, 20 assignés IPs

Nœud 2 (avec module CoreDNS) : ENIs 2, 20 assignés IPs

Nœud 3 (pas de Pod) : 1 ENI. 10 IPs assignés.

Pour le nœud 1 et le nœud 2 (configuration identique) :

- 2 ENIs × 10 IPs par ENI = 20 IPs au total
- Soustraire 2 primaires IPs (1 par ENI) = 18 IPs
- Soustraire 1 IP pour le pod CoreDNS = 17 disponibles IPs
- Donc, chacun de ces nœuds en a 17 IPs dans un pool chaud

Pour le nœud 3 :

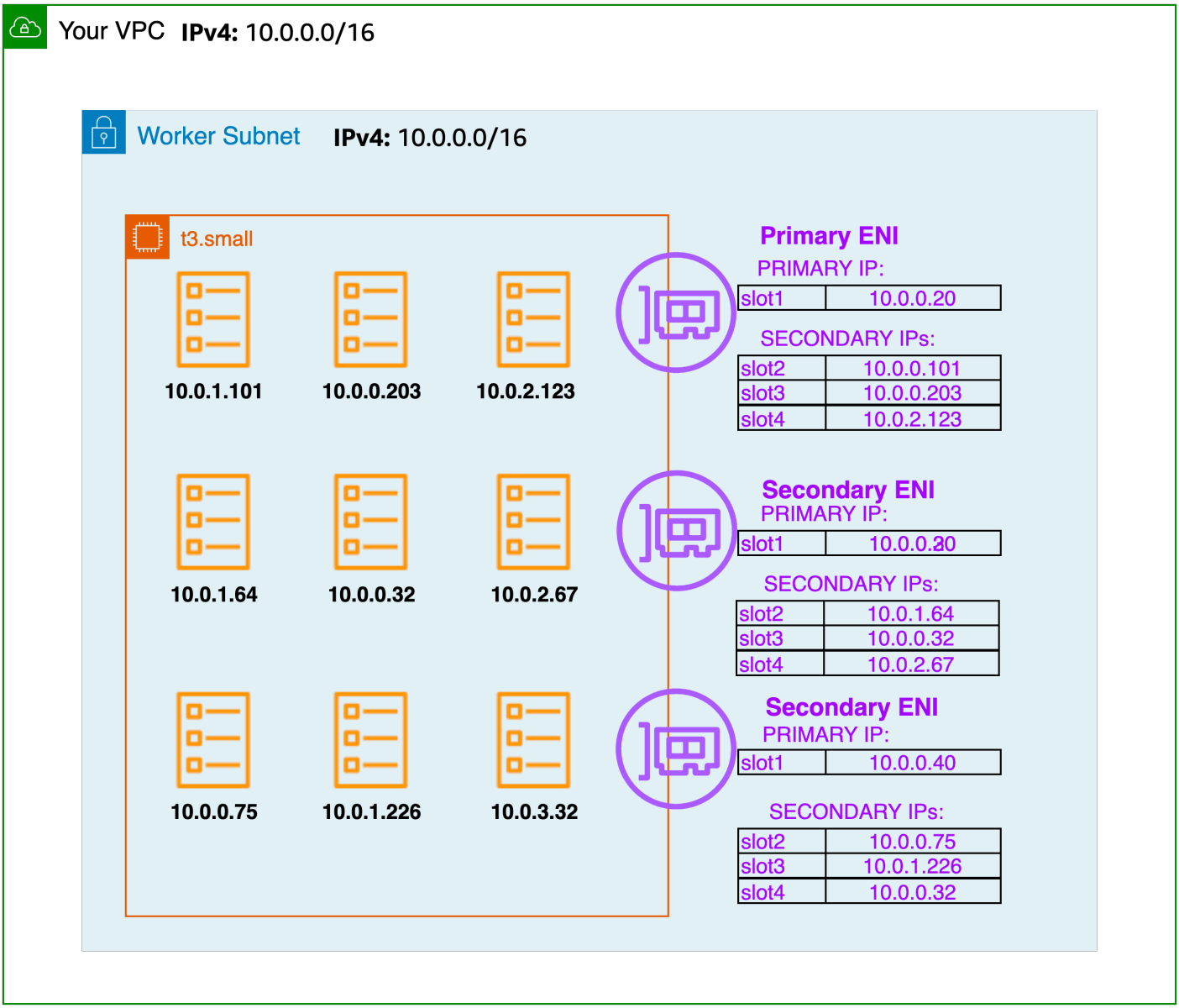
- 1 ENI × 10 IPs = 10 IPs au total
- Soustraire 1 IP principale = 9 IPs disponible dans une piscine chaude

Calcul de la piscine chaude totale : - 17 (nœud 1) + 17 (nœud 2) + 9 (nœud 3) = 43 IPs

N'oubliez pas que les modules d'infrastructure, qui fonctionnent souvent comme des ensembles de démons, contribuent chacun au nombre maximal de modules. Il peut s'agir notamment des éléments suivants :

- CoreDNS
- Amazon Elastic LoadBalancer
- Pods opérationnels pour serveur de métriques

Nous vous suggérons de planifier votre infrastructure en combinant les capacités de ces Pods. Pour obtenir la liste du nombre maximal de pods pris en charge par chaque type d'instance, consultez eni-max-pods.txt on GitHub.



Recommandations

Déployer le cluster EKS avec le mode automatique

Lorsque vous utilisez le mode automatique EKS pour créer un cluster, AWS gère la configuration de l'interface réseau de conteneurs VPC (CNI) de votre cluster. Avec le mode automatique Amazon EKS, il n’est pas nécessaire d’installer ou de mettre à niveau de modules complémentaires de mise en réseau. Assurez-vous toutefois que vos charges de travail sont compatibles avec la configuration CNI du VPC géré.

Déployer le module complémentaire géré par VPC CNI

Lorsque vous provisionnez un cluster, Amazon EKS installe automatiquement le VPC CNI. Amazon EKS prend néanmoins en charge les modules complémentaires gérés qui permettent au cluster d'interagir avec les ressources AWS sous-jacentes telles que le calcul, le stockage et le réseau. Nous vous recommandons vivement de déployer des clusters avec des modules complémentaires gérés, notamment le VPC CNI.

Le module complémentaire géré Amazon EKS permet l'installation et la gestion VPC CNI pour les clusters Amazon EKS. Les modules complémentaires Amazon EKS incluent les derniers correctifs de sécurité et corrections de bogues et sont validés par AWS pour fonctionner avec Amazon EKS. Le module complémentaire VPC CNI vous permet de garantir en permanence la sécurité et la stabilité de vos clusters Amazon EKS et de réduire les efforts nécessaires à l'installation, à la configuration et à la mise à jour des modules complémentaires. En outre, un module complémentaire géré peut être ajouté, mis à jour ou supprimé via l'API Amazon EKS, l'AWS Management Console, l'AWS CLI et eksctl.

Vous pouvez trouver les champs gérés du VPC CNI en utilisant le `--show-managed-fields` drapeau associé à la commande. `kubectl get`

```
kubectl get daemonset aws-node --show-managed-fields -n kube-system -o yaml
```

Les modules complémentaires gérés empêchent la dérive des configurations en les remplaçant automatiquement toutes les 15 minutes. Cela signifie que toutes les modifications apportées aux modules complémentaires gérés, effectuées via l'API Kubernetes après leur création, seront annulées par le processus automatique de prévention de la dérive et seront également définies par défaut lors du processus de mise à jour des modules complémentaires.

Les champs gérés par EKS sont répertoriés sous ManagedFields avec le gestionnaire EKS. Les champs gérés par EKS incluent le compte de service, l'image, l'URL de l'image, la sonde de vivacité, la sonde de préparation, les étiquettes, les volumes et les montages de volume.

> **Note**
>
> Les champs les plus fréquemment utilisés tels que WARM_ENI_TARGET, WARM_IP_TARGET et MINIMUM_IP_TARGET ne sont pas gérés et ne seront pas réconciliés. Les modifications apportées à ces champs seront conservées lors de la mise à jour du module complémentaire.

Nous vous suggérons de tester le comportement des modules complémentaires dans vos clusters hors production pour une configuration spécifique avant de mettre à jour les clusters de production. Suivez également les étapes du guide de l'utilisateur EKS pour les configurations complémentaires.

Migrer vers un module complémentaire géré

Vous allez gérer la compatibilité des versions et mettre à jour les correctifs de sécurité du VPC CNI autogéré. Pour mettre à jour un module complémentaire autogéré, vous devez utiliser Kubernetes APIs et les instructions décrites dans le guide de l'utilisateur d'EKS. Nous vous recommandons de migrer vers un module complémentaire géré pour les clusters EKS existants et vous recommandons vivement de créer une sauvegarde de vos paramètres CNI actuels avant la migration. Pour configurer les modules complémentaires gérés, vous pouvez utiliser l'API Amazon EKS, la console de gestion AWS ou l'interface de ligne de commande AWS.

```
kubectl apply view-last-applied daemonset aws-node -n kube-system  aws-k8s-cni-old.yaml
```

Amazon EKS remplacera les paramètres de configuration CNI si le champ est répertorié comme géré avec les paramètres par défaut. Nous vous déconseillons de modifier les champs gérés. Le module complémentaire ne réconcilie pas les champs de configuration tels que les variables d'environnement chaud et les modes CNI. Les pods et les applications continueront de fonctionner pendant que vous migrez vers un CNI géré.

Backup des paramètres CNI avant la mise à jour

Le VPC CNI s'exécute sur le plan de données client (nœuds), c'est pourquoi Amazon EKS ne met pas automatiquement à jour le module complémentaire (géré et autogéré) lorsque de nouvelles

versions sont publiées ou après la mise à jour de votre cluster vers une nouvelle version mineure de Kubernetes. Pour mettre à jour le module complémentaire pour un cluster existant, vous devez déclencher une mise à jour via l'API update-addon ou en cliquant sur le lien Mettre à jour maintenant dans la console EKS pour les modules complémentaires. Si vous avez déployé un module complémentaire autogéré, suivez les étapes décrites dans la section Mise à jour du module complémentaire VPC CNI autogéré.

Nous vous recommandons vivement de mettre à jour une version mineure à la fois. Par exemple, si `1.9` est votre version mineure actuelle que vous souhaitez la mettre à jour vers `1.11`, vous devez d'abord mettre à jour vers le dernier correctif et la dernière version de build de `1.10`, puis vers le dernier correctif et la dernière version de build de `1.11`.

Effectuez une inspection du daemonset aws-node avant de mettre à jour Amazon VPC CNI. Effectuez une sauvegarde des paramètres existants. Si vous utilisez un module complémentaire géré, vérifiez que vous n'avez mis à jour aucun paramètre susceptible d'être remplacé par Amazon EKS. Nous vous recommandons d'intégrer une étape de post-mise à jour dans votre flux de travail d'automatisation ou une étape d'application manuelle après une mise à jour d'un module complémentaire.

```
kubectl apply view-last-applied daemonset aws-node -n kube-system  aws-k8s-cni-old.yaml
```

Dans le cas d'un module complémentaire autogéré, comparez la sauvegarde GitHub à la version `releases` activée pour voir les versions disponibles et vous familiariser avec les modifications apportées à la version vers laquelle vous souhaitez effectuer la mise à jour. Nous vous recommandons d'utiliser Helm pour gérer les modules complémentaires autogérés et utiliser les fichiers de valeurs pour appliquer les paramètres. Toute opération de mise à jour impliquant la suppression du Daemonset entraînera l'arrêt de l'application et doit être évitée.

Comprendre le contexte de sécurité

Nous vous conseillons vivement de comprendre les contextes de sécurité configurés pour gérer efficacement le VPC CNI. Amazon VPC CNI comporte deux composants : CNI binary et ipamd (aws-node) Daemonset. Le CNI s'exécute en tant que binaire sur un nœud et a accès au système de fichiers racine du nœud. Il dispose également d'un accès privilégié car il traite les iptables au niveau du nœud. Le binaire CNI est invoqué par le kubelet lorsque Pods est ajouté ou supprimé.

Le daemonset aws-node est un processus de longue haleine responsable de la gestion des adresses IP au niveau du nœud. L'aws-node s'exécute en `hostNetwork` mode et autorise l'accès au

périphérique de bouclage et à l'activité réseau des autres pods sur le même nœud. Le conteneur d'initialisation aws-node s'exécute en mode privilégié et monte le socket CRI, ce qui permet au Daemonset de surveiller l'utilisation des adresses IP par les pods exécutés sur le nœud. Amazon EKS s'efforce de supprimer l'exigence privilégiée du conteneur d'initialisation aws-node. De plus, l'aws-node doit mettre à jour les entrées NAT et charger les modules iptables. Il fonctionne donc avec les privilèges NET_ADMIN.

Amazon EKS recommande de déployer les politiques de sécurité définies par le manifeste aws-node pour la gestion des adresses IP des Pods et les paramètres réseau. Pensez à passer à la dernière version de VPC CNI. En outre, pensez à ouvrir un GitHub numéro si vous avez des exigences de sécurité spécifiques.

Utiliser un rôle IAM distinct pour CNI

L'AWS VPC CNI nécessite des autorisations AWS Identity and Access Management (IAM). La politique CNI doit être configurée avant que le rôle IAM puisse être utilisé. Vous pouvez utiliser `AmazonEKS_CNI_Policy`une politique gérée par AWS pour les IPv4 clusters. La politique gérée par Amazoneks CNI ne dispose d'autorisations que pour les clusters. IPv4 Vous devez créer une politique IAM distincte pour les IPv6 clusters avec les autorisations répertoriées ici.

Par défaut, le VPC CNI hérite du rôle IAM du nœud Amazon EKS (groupes de nœuds gérés et autogérés).

Il est fortement recommandé de configurer un rôle IAM distinct avec les politiques pertinentes pour Amazon VPC CNI. Dans le cas contraire, les pods d'Amazon VPC CNI obtiennent l'autorisation attribuée au rôle IAM du nœud et ont accès au profil d'instance attribué au nœud.

Le plugin VPC CNI crée et configure un compte de service appelé aws-node. Par défaut, le compte de service est lié au rôle IAM du nœud Amazon EKS auquel est attachée la politique CNI d'Amazon EKS. Pour utiliser le rôle IAM distinct, nous vous recommandons de créer un nouveau compte de service avec la politique CNI Amazon EKS attachée. Pour utiliser un nouveau compte de service, vous devez redéployer les pods CNI. Envisagez de spécifier un module complémentaire géré par CNI `--service-account-role-arn` pour VPC lors de la création de nouveaux clusters. Assurez-vous de supprimer la politique CNI d'Amazon EKS pour les deux rôles IPv4 et pour le rôle IPv6 de nœud Amazon EKS.

Il est conseillé de bloquer l'accès aux métadonnées des instances afin de minimiser le rayon d'impact d'une faille de sécurité.

Gérer les défaillances des Liveness/Readiness sondes

Nous vous conseillons d'augmenter les valeurs de délai de réponse et de disponibilité de la sonde (par défaut`timeoutSeconds: 10`) pour les clusters EKS 1.20 et versions ultérieures afin d'éviter que des défaillances de sonde n'entraînent le blocage du pod de votre application dans un état ContainerCreating. Ce problème a été observé dans les clusters gourmands en données et dans les clusters de traitement par lots. Une utilisation élevée du processeur entraîne des défaillances de santé de la sonde aws-node, ce qui entraîne des demandes de processeur Pod non satisfaites. Outre la modification du délai d'expiration de la sonde, assurez-vous que les demandes de ressources du processeur (par défaut`CPU: 25m`) pour aws-node sont correctement configurées. Nous vous déconseillons de mettre à jour les paramètres, sauf si votre nœud rencontre des problèmes.

Nous vous encourageons vivement à exécuter sudo `bash /opt/cni/bin/aws-cni-support.sh` sur un nœud pendant que vous contactez le support Amazon EKS. Le script aidera à évaluer les journaux Kubelet et l'utilisation de la mémoire sur le nœud. Pensez à installer l'agent SSM sur les nœuds de travail Amazon EKS pour exécuter le script.

Configuration de la IPTables politique de transfert sur les instances AMI non optimisées pour EKS

Si vous utilisez une AMI personnalisée, assurez-vous de définir la politique de transfert d'iptables sur ACCEPT sous kubelet.service. De nombreux systèmes définissent la politique de transfert d'iptables sur DROP. Vous pouvez créer une AMI personnalisée à l'aide de HashiCorp Packer et d'une spécification de construction avec des ressources et des scripts de configuration issus du référentiel d'AMI Amazon EKS sur AWS GitHub. Vous pouvez mettre à jour le kubelet.service et suivre les instructions spécifiées ici pour créer une AMI personnalisée.

Mettez régulièrement à niveau la version CNI

Le VPC CNI est rétrocompatible. La dernière version fonctionne avec toutes les versions de Kubernetes prises en charge par Amazon EKS. En outre, le VPC CNI est proposé en tant que module complémentaire EKS (voir « Déployer le module complémentaire géré par VPC CNI » ci-dessus). Bien que les modules complémentaires EKS orchestrent les mises à niveau des modules complémentaires, ils ne mettent pas automatiquement à niveau les modules complémentaires tels que le CNI, car ils s'exécutent sur le plan de données. Vous êtes responsable de la mise à niveau du module complémentaire VPC CNI après les mises à niveau des nœuds de travail gérés et autogérés.

Optimisation de l'utilisation des adresses IP

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Les environnements conteneurisés prennent de l'ampleur à un rythme rapide, grâce à la modernisation des applications. Cela signifie que de plus en plus de nœuds de travail et de pods sont déployés.

Le plug-in Amazon VPC CNI attribue à chaque pod une adresse IP issue du ou des CIDR du VPC. Cette approche fournit une visibilité complète des adresses des Pod grâce à des outils tels que les journaux de flux VPC et d'autres solutions de surveillance. En fonction de votre type de charge de travail, cela peut entraîner la consommation d'un nombre important d'adresses IP par les pods.

Lors de la conception de votre architecture réseau AWS, il est important d'optimiser la consommation d'adresses IP Amazon EKS au niveau du VPC et du nœud. Cela vous aidera à atténuer les problèmes d'épuisement des adresses IP et à augmenter la densité des pods par nœud.

Dans cette section, nous aborderons les techniques qui peuvent vous aider à atteindre ces objectifs.

Optimisation de la consommation IP au niveau des nœuds

La délégation de préfixes est une fonctionnalité d'Amazon Virtual Private Cloud (Amazon VPC) qui vous permet d'attribuer des préfixes IPv4 ou IPv6 à vos instances Amazon Elastic Compute Cloud (Amazon EC2). Il augmente le nombre d'adresses IP par interface réseau (ENI), ce qui augmente la densité de modules par nœud et améliore l'efficacité de votre calcul. La délégation de préfixes est également prise en charge par le Custom Networking.

Pour des informations détaillées, consultez les sections Délégation de préfixes avec les nœuds Linux et Délégation de préfixes avec les nœuds Windows.

Atténuer l'épuisement des IP

Pour éviter que vos clusters n'utilisent toutes les adresses IP disponibles, nous vous recommandons vivement de dimensionner votre réseau VPCs et vos sous-réseaux en tenant compte de la croissance.

L'adoption IPv6est un excellent moyen d'éviter ces problèmes dès le début. Toutefois, pour les entreprises dont les besoins en évolutivité dépassent le planning initial et ne peuvent pas les adopter IPv6, il est recommandé d'améliorer la conception des VPC pour pallier l'épuisement des adresses IP. La technique la plus couramment utilisée par les clients Amazon EKS consiste à ajouter un élément secondaire non routable CIDRs au VPC et à configurer le CNI du VPC pour utiliser cet espace IP supplémentaire lors de l'attribution d'adresses IP aux pods. C'est ce que l'on appelle communément le réseau personnalisé.

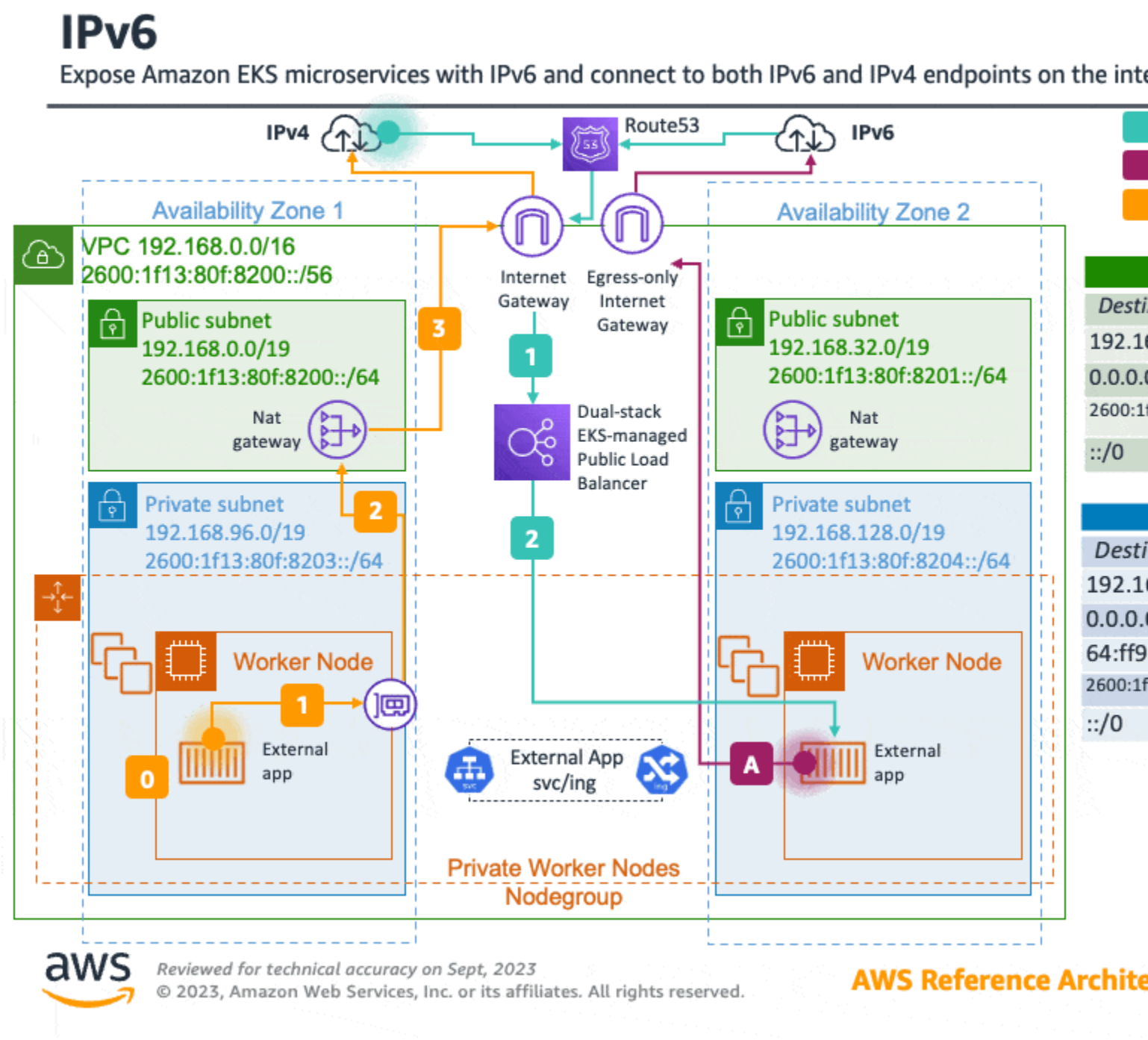
Nous aborderons les variables du CNI Amazon VPC que vous pouvez utiliser pour optimiser le pool de chaleur IPs attribué à vos nœuds. Nous clôturerons cette section avec d'autres modèles architecturaux qui ne sont pas intrinsèques à Amazon EKS mais qui peuvent contribuer à atténuer l'épuisement des adresses IP.

Utilisation IPv6 (recommandée)

L'adoption IPv6 est le moyen le plus simple de contourner les RFC1918 limites ; nous vous recommandons vivement d'envisager l'adoption IPv6 comme première option lorsque vous choisissez une architecture réseau. IPv6 fournit un espace d'adresses IP total nettement plus important, et les administrateurs de clusters peuvent se concentrer sur la migration et le dimensionnement des applications sans consacrer d'efforts à contourner IPv4 les limites.

Les clusters Amazon EKS prennent en charge à la fois IPv4 et IPv6. Par défaut, les clusters EKS utilisent l'espace d' IPv4 adressage. La spécification d'un espace d'adressage IPv6 basé au moment de la création du cluster permettra l'utilisation de IPv6. Dans un cluster IPv6 EKS, les pods et les services reçoivent des IPv6 adresses tout en conservant la capacité des IPv4 terminaux existants à se connecter aux services exécutés sur des IPv6 clusters et vice versa. Toutes les pod-to-pod communications au sein d'un cluster ont toujours lieu IPv6. Dans un VPC (/56), la taille de bloc IPv6 CIDR pour les IPv6 sous-réseaux est fixée à /64. Cela fournit 2^64 (environ 18 quintillions) IPv6 adresses permettant d'étendre vos déploiements sur EKS.

Pour des informations détaillées, consultez la section Running IPv6 EKS Clusters et pour une expérience pratique, consultez la section Comprendre IPv6 Amazon EKS de l' IPv6 atelier Get hands with.



Optimisation de la consommation IP dans les IPv4 clusters

Cette section est dédiée aux clients qui exécutent des applications héritées, mais qui ne and/or sont pas prêts à effectuer la migration IPv6. Bien que nous encouragions toutes les organisations à migrer vers cette solution le plus rapidement IPv6 possible, nous sommes conscients que certaines devront peut-être encore envisager d'autres approches pour augmenter leurs charges de travail liées aux

IPv4 conteneurs. C'est pourquoi nous vous expliquerons également les modèles architecturaux permettant d'optimiser IPv4 (RFC1918) la consommation d'espace avec les clusters Amazon EKS.

Plan de croissance

Comme première ligne de défense contre l'épuisement des adresses IP, nous vous recommandons vivement de dimensionner vos sous-réseaux IPv4 VPCs et vos sous-réseaux en tenant compte de la croissance, afin d'éviter que vos clusters n'utilisent toutes les adresses IP disponibles. Vous ne pourrez pas créer de nouveaux pods ou nœuds si les sous-réseaux ne disposent pas d'un nombre suffisant d'adresses IP disponibles.

Avant de créer un VPC et des sous-réseaux, il est conseillé de revenir en arrière à partir de l'échelle de charge de travail requise. Par exemple, lorsque les clusters sont créés à l'aide d'eksctl (un outil CLI simple pour créer et gérer des clusters sur EKS), 19 sous-réseaux sont créés par défaut. Un masque réseau de /19 convient à la majorité des types de charge de travail permettant d'allouer plus de 8 000 adresses.

> **Important**
>
> Lorsque vous VPCs dimensionnez des sous-réseaux, certains éléments (autres que les pods et les nœuds) peuvent consommer des adresses IP, par exemple les équilibreurs de charge, les bases de données RDS et d'autres services intégrés au VPC.

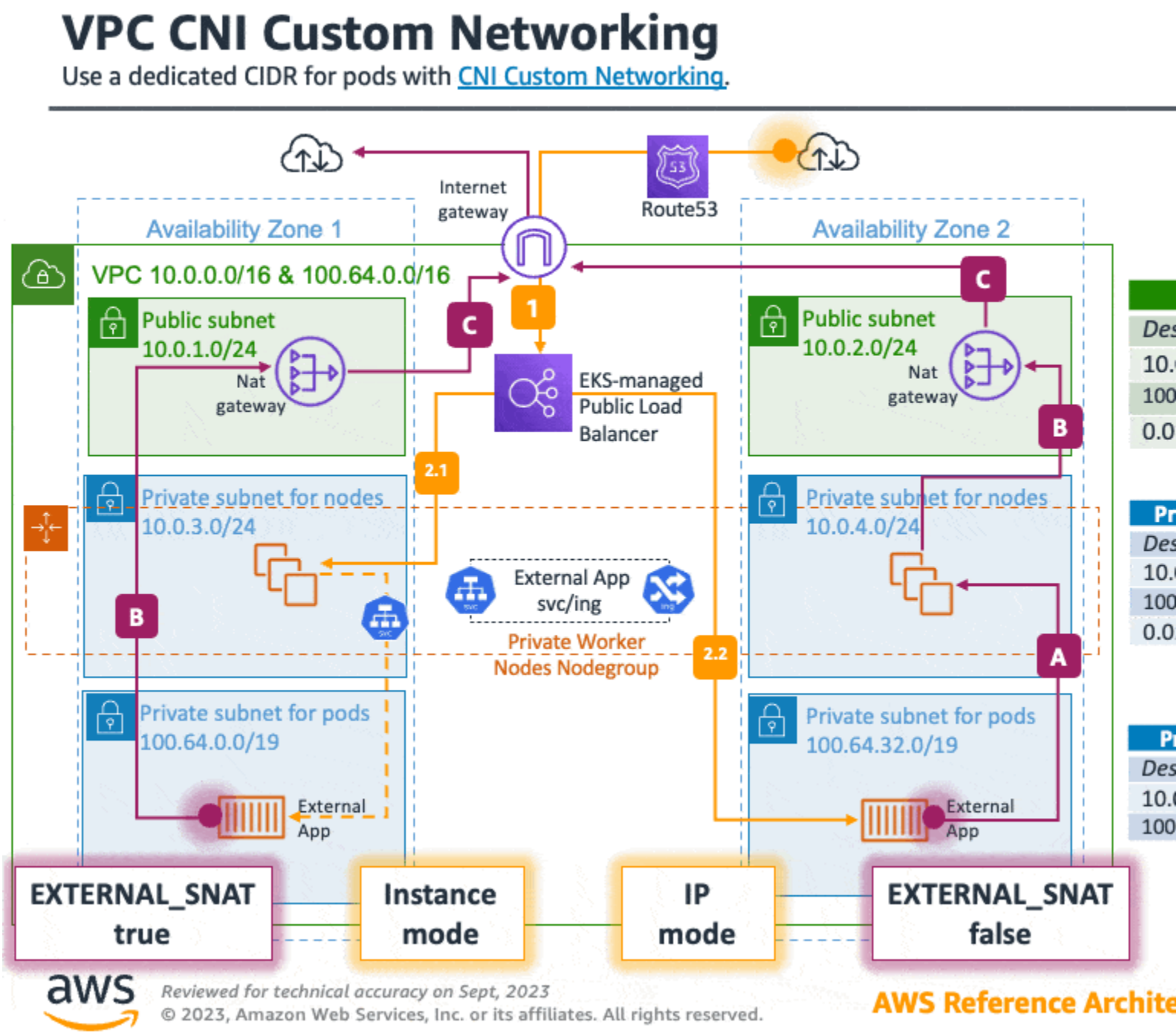
En outre, Amazon EKS peut créer jusqu'à 4 interfaces réseau élastiques (X-ENI) nécessaires pour permettre la communication avec le plan de contrôle (plus d'informations ici). Lors des mises à niveau de clusters, Amazon EKS crée de nouveaux X ENIs et supprime les anciens lorsque la mise à niveau est réussie. C'est pourquoi nous recommandons un masque réseau d'au moins /28 (16 adresses IP) pour les sous-réseaux associés à un cluster EKS.

Vous pouvez utiliser l'exemple de feuille de calcul EKS Subnet Calculator pour planifier votre réseau. La feuille de calcul calcule l'utilisation des adresses IP en fonction des charges de travail et de la configuration VPC ENI. L'utilisation de l'adresse IP est comparée à celle d'un IPv4 sous-réseau afin de déterminer si la configuration et la taille du sous-réseau sont suffisantes pour votre charge de travail. N'oubliez pas que, si les sous-réseaux de votre VPC n'ont plus d'adresses IP disponibles, nous vous suggérons de créer un nouveau sous-réseau en utilisant les blocs d'adresse CIDR d'origine du VPC. Notez qu'Amazon EKS permet désormais de modifier les sous-réseaux et les groupes de sécurité du cluster.

Mise en réseau personnalisée

Si vous êtes sur le point d'épuiser l'espace RFC1918 IP, vous pouvez utiliser le modèle de réseau personnalisé pour conserver le routabilité IPs en programmant des pods dans des sous-réseaux supplémentaires dédiés. Bien que le réseau personnalisé accepte une plage VPC valide pour la plage CIDR secondaire, nous vous recommandons d'utiliser CIDRs (/16) à partir de l'espace CG-NAT, c'est-à-dire `198.19.0.0/16` car ces plages sont moins susceptibles `100.64.0.0/10` d'être utilisées dans un environnement d'entreprise que les plages. RFC1918

Pour des informations détaillées, veuillez consulter la section dédiée à la mise en réseau personnalisée.



Découverte améliorée des sous-réseaux

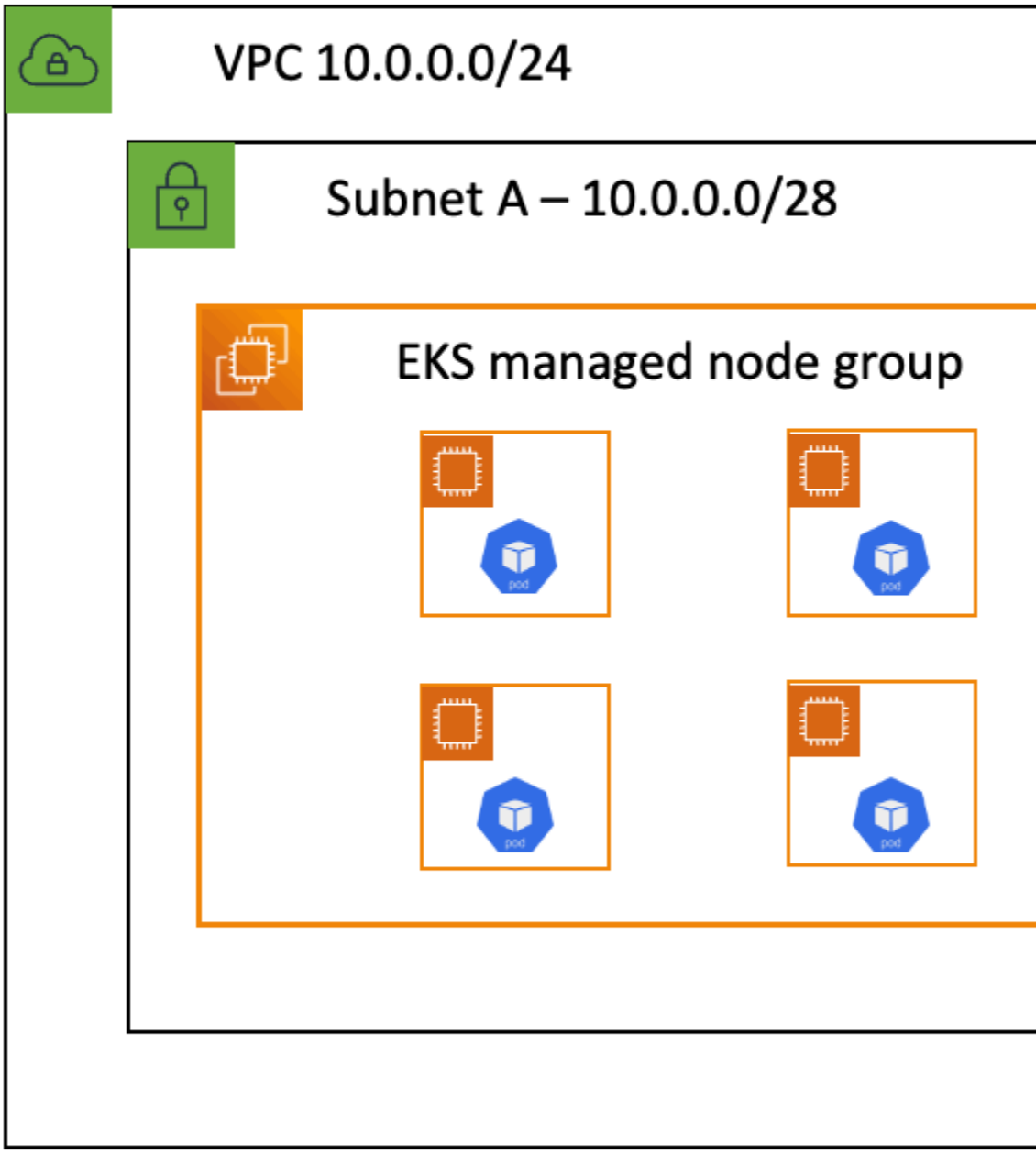
Enhanced Subnet Discovery fournit une alternative de configuration réseau rationalisée pour l'épuisement des adresses IP, en balisant les nouveaux sous-réseaux afin qu'ils soient détectables par le CNI Amazon VPC. Grâce à Enhanced Subnet Discovery, les charges de travail actuelles peuvent continuer à s'exécuter sur les mêmes sous-réseaux et Amazon Elastic Kubernetes Service

(Amazon EKS) peut désormais planifier des pods supplémentaires sur les nouveaux « sous-réseaux utilisables ».

Si les sous-réseaux actuels de votre cluster sont à court d'adresses IP, vous pouvez simplement ajouter des sous-réseaux supplémentaires à votre cluster Amazon EKS comme suit :

1. Associez un nouveau bloc CIDR à votre VPC.
2. Créez un nouveau sous-réseau dans le nouveau bloc CIDR et étiquetez-le avec « kubernetes ». io/role/cni» = « 1 ».
3. Activez la configuration ENABLE_SUBNET_DISCOVERY du module complémentaire Amazon VPC CNI sur « true » (valeur par défaut depuis la version 1.18.0).

Une fois la découverte améliorée des sous-réseaux activée sur vos clusters VPC et Amazon EKS, de nouvelles interfaces réseau élastiques ENIs () seront associées à vos nœuds Amazon EKS, comme décrit dans le schéma suivant :


VPC 10.0.0.0/24
Subnet A – 10.0.0.0/28
EKS managed node group
pod
pod
pod
pod

Pour plus d'informations, consultez Amazon VPC CNI présente la découverte améliorée des sous-réseaux sur le blog consacré aux conteneurs AWS.

Optimisez la piscine IPs chaude

Avec la configuration par défaut, le VPC CNI conserve une ENI complète (et associée IPs) dans le pool de chaleur. Cela peut en consommer un grand nombre IPs, en particulier pour les types d'instances plus importants.

Si le sous-réseau de votre cluster dispose d'un nombre limité d'adresses IP disponibles, examinez ces variables d'environnement de configuration VPC CNI :

- `WARM_IP_TARGET`
- `MINIMUM_IP_TARGET`
- `WARM_ENI_TARGET`

Vous pouvez configurer la valeur de `MINIMUM_IP_TARGET` pour qu'elle corresponde étroitement au nombre de pods que vous comptez exécuter sur vos nœuds. Cela garantira qu'au fur et à mesure de la création des pods, le CNI pourra attribuer des adresses IP à partir du warm pool sans appeler l'API EC2.

N'oubliez pas que définir une valeur `WARM_IP_TARGET` trop faible entraînera des appels supplémentaires vers l'API EC2, ce qui pourrait entraîner une limitation des demandes. Pour les grands clusters, utilisez along `MINIMUM_IP_TARGET` pour éviter de limiter le nombre de demandes.

Pour configurer ces options, vous pouvez télécharger le `aws-k8s-cni.yaml` manifeste et définir les variables d'environnement. Au moment de la rédaction de cet article, la dernière version se trouve ici. Vérifiez que la version de la valeur de configuration correspond à la version VPC CNI installée.

> **Warning**
>
> Ces paramètres seront réinitialisés aux valeurs par défaut lorsque vous mettrez à jour le CNI. Veuillez effectuer une sauvegarde du CNI avant de le mettre à jour. Passez en revue les paramètres de configuration pour déterminer s'il est nécessaire de les réappliquer une fois la mise à jour réussie.

Vous pouvez ajuster les paramètres CNI à la volée sans interruption pour vos applications existantes, mais vous devez choisir des valeurs adaptées à vos besoins d'évolutivité. Par exemple, si vous

travaillez avec des charges de travail par lots, nous vous recommandons de mettre à jour la valeur par défaut `WARM_ENI_TARGET` pour répondre aux besoins de la balance Pod. Le réglage `WARM_ENI_TARGET` sur une valeur élevée permet toujours de maintenir le pool d'adresses IP chaudes requis pour exécuter des charges de travail par lots importantes et d'éviter ainsi les retards de traitement des données.

> **Warning**
>
> L'amélioration de la conception de votre VPC est la réponse recommandée à l'épuisement des adresses IP. Envisagez des solutions telles que IPv6 et secondaire CIDRs. Le réglage de ces valeurs pour minimiser le nombre de Warm IPs devrait être une solution temporaire une fois les autres options exclues. Une mauvaise configuration de ces valeurs peut interférer avec le fonctionnement du cluster. Avant d'apporter des modifications à un système de production, assurez-vous de prendre connaissance des considérations figurant sur cette page.

Surveiller l'inventaire des adresses IP

Outre les solutions décrites ci-dessus, il est également important d'avoir une visibilité sur l'utilisation des adresses IP. Vous pouvez surveiller l'inventaire des adresses IP des sous-réseaux à l'aide de CNI Metrics Helper. Certaines des mesures disponibles sont les suivantes :

- nombre maximum de personnes que ENIs le cluster peut prendre en charge
- nombre de personnes ENIs déjà allouées
- nombre d'adresses IP actuellement attribuées aux Pods
- nombre total et maximum d'adresses IP disponibles

Vous pouvez également configurer des CloudWatch alarmes pour être averti si un sous-réseau manque d'adresses IP.

> **Warning**
>
> Assurez-vous que la `DISABLE_METRICS` variable pour VPC CNI est définie sur false.

Autres considérations

Il existe d'autres modèles architecturaux qui ne sont pas intrinsèques à Amazon EKS et qui peuvent contribuer à l'épuisement des adresses IP. Par exemple, vous pouvez optimiser la communication entre plusieurs comptes VPCs ou partager un VPC entre plusieurs comptes afin de limiter l'allocation d' IPv4 adresses.

Pour en savoir plus sur ces modèles, cliquez ici :

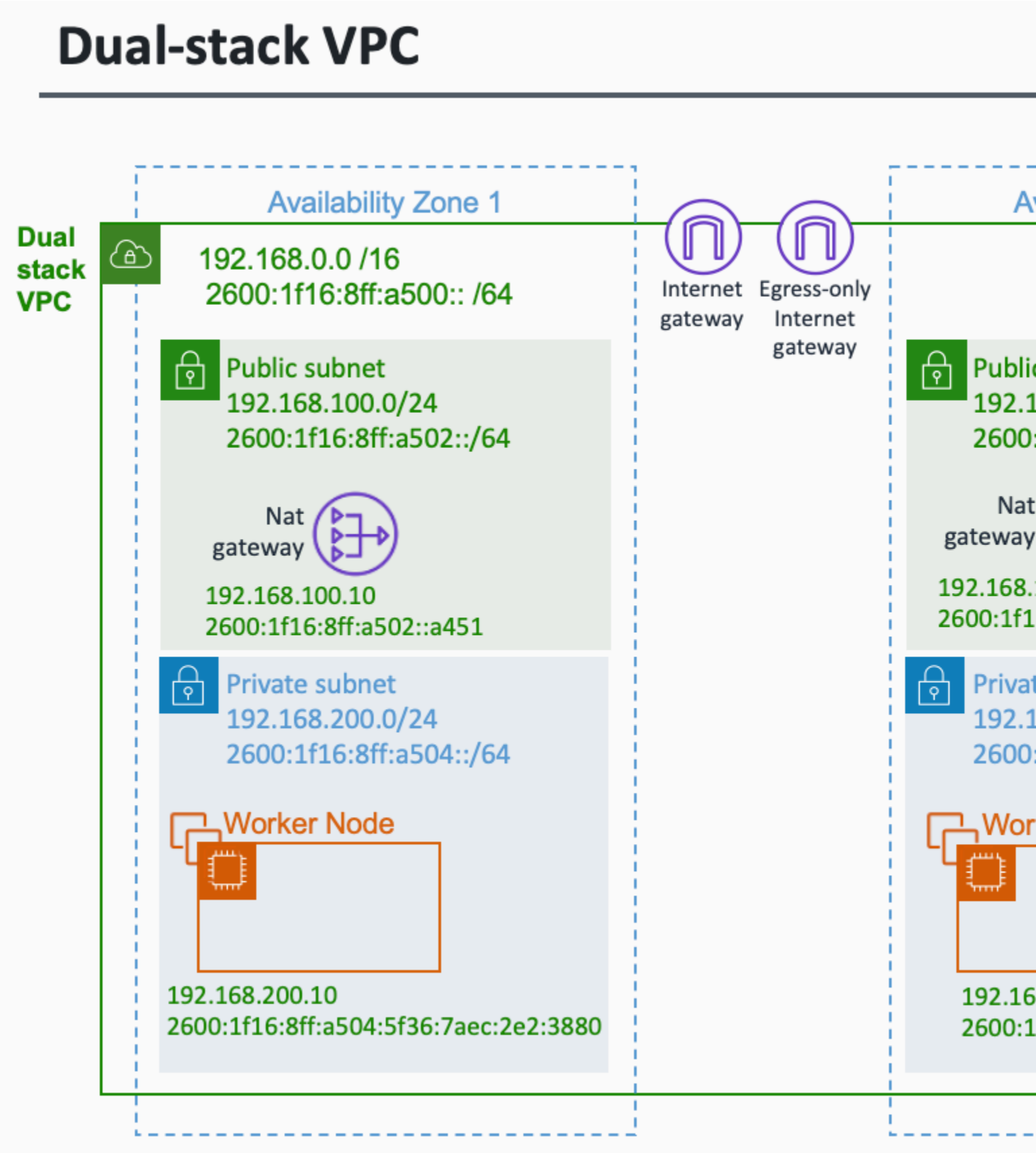
- Conception de réseaux Amazon VPC à très grande échelle,
- Développez une connectivité multi-comptes multi-VPC sécurisée avec Amazon VPC Lattice.

Exécution de clusters IPv6 EKS

Le mode EKS en IPv6 mode résout le problème IPv4 d'épuisement qui se manifeste souvent dans les clusters EKS à grande échelle. L'assistance d'EKS IPv6 est axée sur la résolution du problème d' IPv4 épuisement, qui découle de la taille limitée de l'espace d' IPv4 adressage. Il s'agit d'une préoccupation importante soulevée par un certain nombre de nos clients et elle est distincte de la fonctionnalité IPv4IPv6 Kubernetes/Dual-Stack. EKS/ IPv6 offrira également la flexibilité nécessaire pour interconnecter les limites du réseau en minimisant IPv6 CIDRs ainsi les risques de chevauchement des CIDR, résolvant ainsi un double problème (en cluster, en cluster). Lors du déploiement de clusters EKS en IPv6 mode (--ip-family ipv6), l'action n'est pas réversible. En termes simples, le IPv6 support EKS est activé pendant toute la durée de vie de votre cluster.

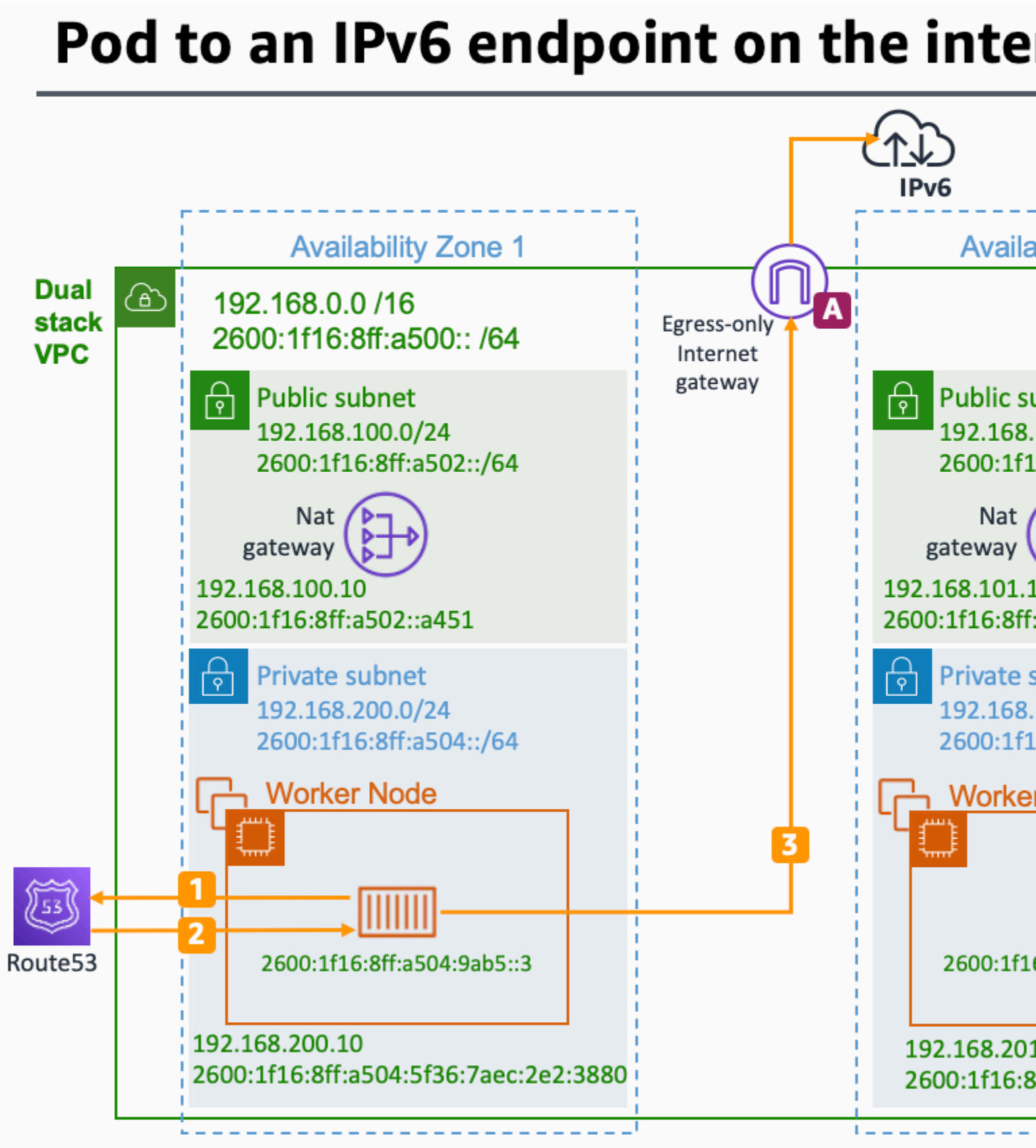
Dans un cluster IPv6 EKS, les Pods and Services recevront des IPv6 adresses tout en maintenant la compatibilité avec les IPv4 Endpoints existants. Cela inclut la possibilité pour les IPv4 points de terminaison externes d'accéder aux services du cluster et les pods d'accéder aux points de terminaison externes IPv4 .

Le IPv6 support Amazon EKS tire parti des fonctionnalités IPv6 VPC natives. Chaque VPC est doté d'un préfixe d' IPv4 adresse (la taille du bloc CIDR peut être comprise entre /16 et /28) et d'un préfixe d'adresse /56 unique (fixe) provenant du GUA (IPv6 adresse globale unicast) d'Amazon ; vous pouvez attribuer un préfixe d'adresse /64 à chaque sous-réseau de votre VPC. IPv4 les fonctionnalités, telles que les tables de routage, les listes de contrôle d'accès réseau, le peering et la résolution DNS, fonctionnent de la même manière dans un VPC IPv6 activé. Le VPC est alors appelé VPC à double pile, suivant les sous-réseaux à double pile. Le schéma suivant décrit le modèle de base du VPC qui prend en charge les clusters basés IPV4 IPv6 sur EKS/ : IPv6


Dual-stack VPC
Dual stack VPC
Availability Zone 1
192.168.0.0 /16
2600:1f16:8ff:a500:: /64
Internet gateway
Egress-only Internet gateway
Public subnet
192.168.100.0/24
2600:1f16:8ff:a502::/64
Nat gateway
192.168.100.10
2600:1f16:8ff:a502::a451
Private subnet
192.168.200.0/24
2600:1f16:8ff:a504::/64
Worker Node
192.168.200.10
2600:1f16:8ff:a504:5f36:7aec:2e2:3880

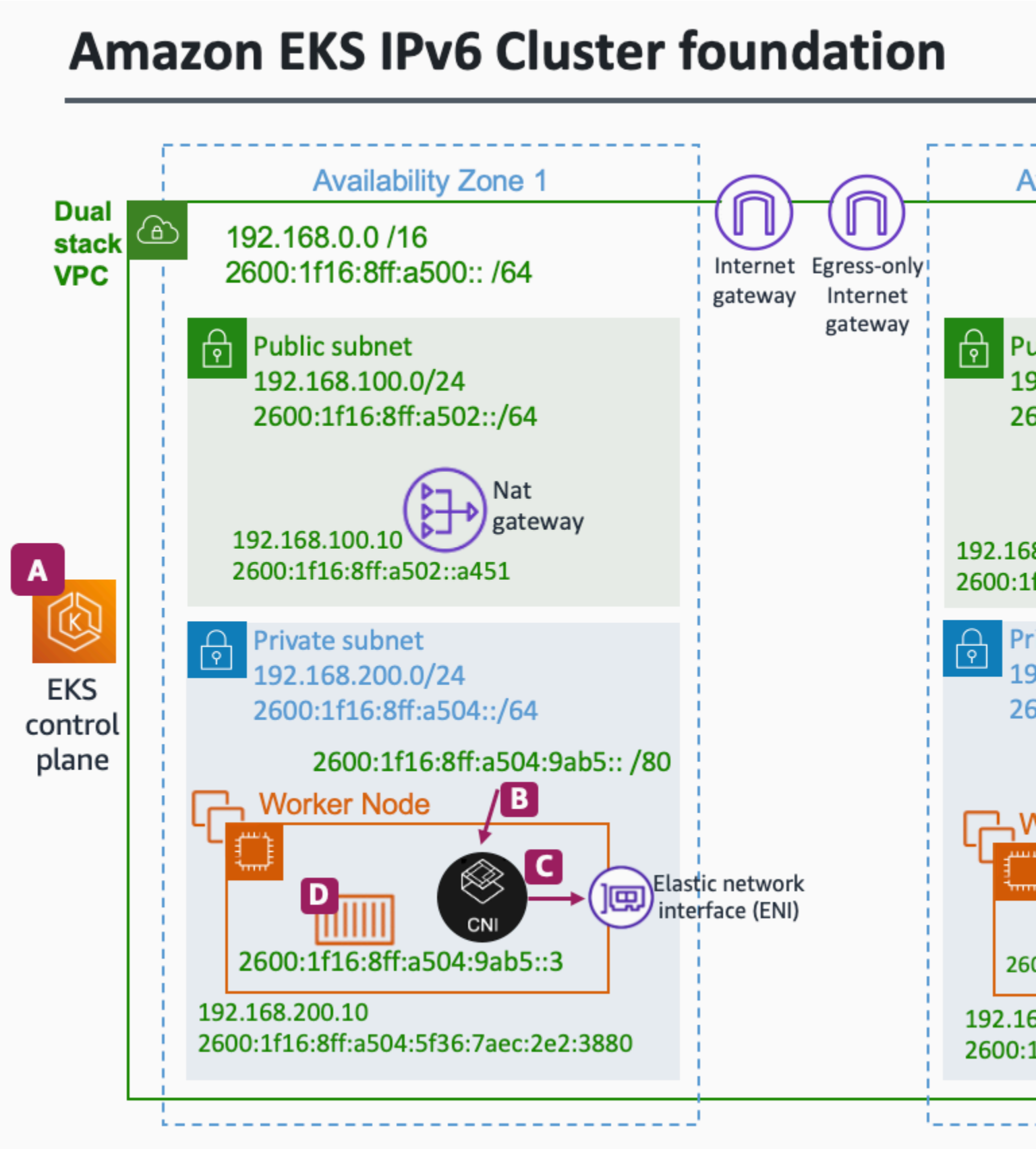
Dans le IPv6 monde, chaque adresse est routable sur Internet. Par défaut, le VPC alloue le IPv6 CIDR à partir de la plage GUA publique. Cependant, depuis août 2024, vous pouvez également utiliser l' IPv6 adressage privé pour VPCs et les sous-réseaux avec Amazon VPC IP Address Manager (IPAM). Consultez ce billet de blog AWS Networking et la documentation VPC pour plus d'informations.

Le schéma suivant illustre un flux de sortie IPv6 Internet Pod au sein d'un cluster IPv6 EKS/ :


Pod to an IPv6 endpoint on the inte
IPv6
Availability Zone 1
Dual stack VPC
192.168.0.0 /16
2600:1f16:8ff:a500:: /64
Egress-only Internet gateway
A
Public subnet
192.168.100.0/24
2600:1f16:8ff:a502::/64
Nat gateway
192.168.100.10
2600:1f16:8ff:a502::a451
Private subnet
192.168.200.0/24
2600:1f16:8ff:a504::/64
Worker Node
3
1
2
Route53
2600:1f16:8ff:a504:9ab5::3
192.168.200.10
2600:1f16:8ff:a504:5f36:7aec:2e2:3880

Les meilleures pratiques pour implémenter IPv6 des sous-réseaux sont disponibles dans le guide de l'utilisateur du VPC.

Dans un cluster IPv6 EKS, les nœuds et les pods reçoivent des IPv6 adresses publiques. EKS attribue des IPv6 adresses aux services sur la base d'adresses locales uniques de IPv6 monodiffusion (ULA). Le CIDR du service ULA pour un IPv6 cluster est automatiquement attribué lors de la phase de création du cluster et ne peut pas être spécifié, contrairement à IPv4. Le schéma suivant illustre un modèle de base de plan de données basé sur un plan de contrôle de cluster IPv6 basé sur EKS/ :


Amazon EKS IPv6 Cluster foundation
Availability Zone 1
Dual stack VPC
192.168.0.0 /16
2600:1f16:8ff:a500:: /64
Internet gateway
Egress-only Internet gateway
Public subnet
192.168.100.0/24
2600:1f16:8ff:a502::/64
Nat gateway
192.168.100.10
2600:1f16:8ff:a502::a451
A
EKS control plane
Private subnet
192.168.200.0/24
2600:1f16:8ff:a504::/64
2600:1f16:8ff:a504:9ab5:: /80
Worker Node
B
C
D
CNI
Elastic network interface (ENI)
2600:1f16:8ff:a504:9ab5::3
192.168.200.10
2600:1f16:8ff:a504:5f36:7aec:2e2:3880

Présentation de

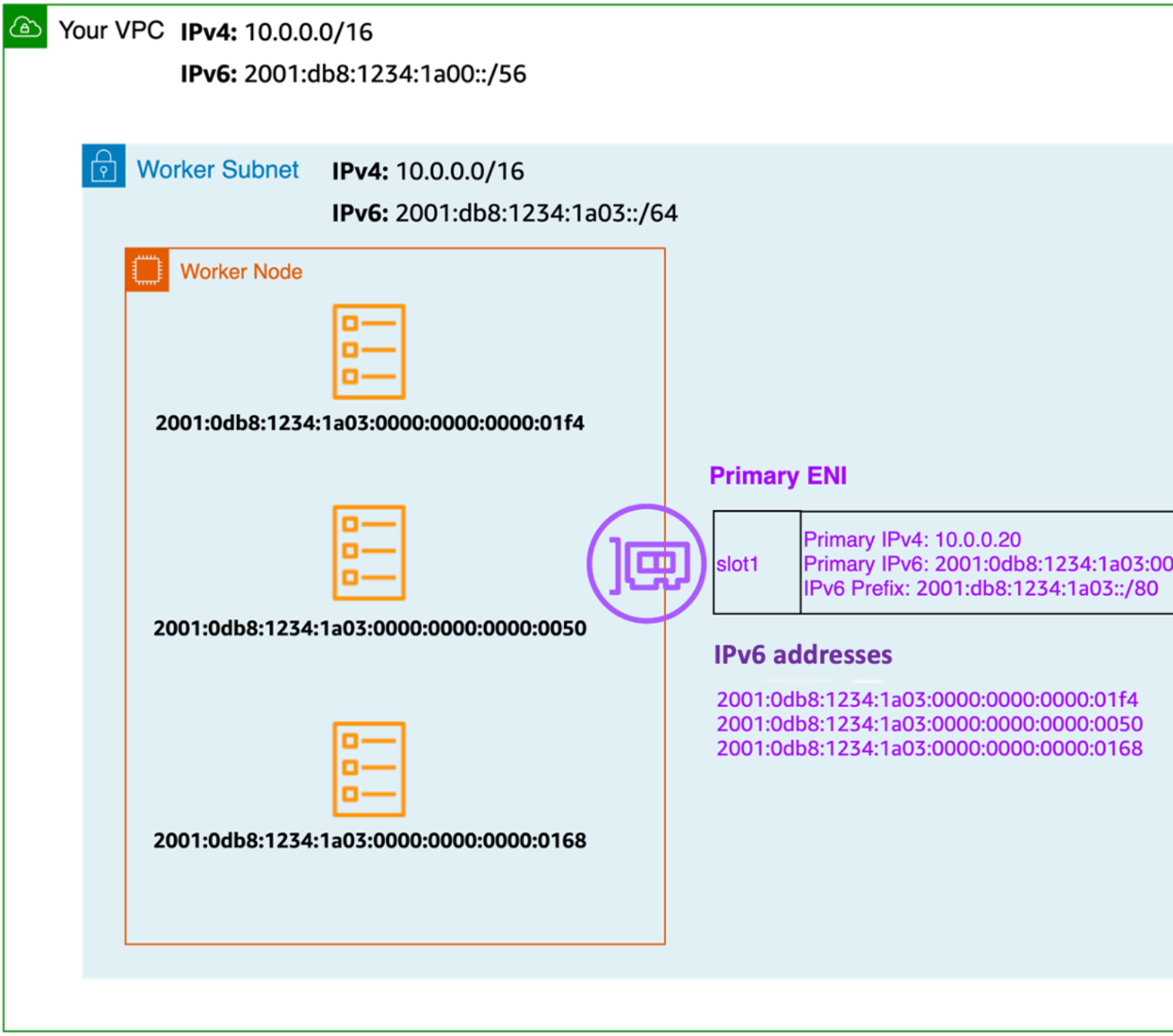
EKS/ n'IPv6 est pris en charge qu'en mode préfixe (mode d'attribution IP ENI du plug-in VPC-CNI). En savoir plus sur le mode préfixe.

L'attribution de préfixes ne fonctionne que sur les instances EC2 basées sur Nitro, c'est pourquoi EKS/IPv6 n'est pris en charge que lorsque le plan de données du cluster utilise des instances basées sur EC2 Nitro.

En termes simples, un IPv6 préfixe de /80 (par nœud de travail) produira environ 10 ^ 14 IPv6 adresses, le facteur limitant IPs ne sera plus celui de la densité du pod (en termes de ressources).

IPv6 l'attribution du préfixe ne se produit qu'au moment du démarrage du nœud de travail EKS. Ce comportement est connu pour atténuer les scénarios dans lesquels les IPv4 clusters EKS/ à taux de désabonnement élevé sont souvent retardés dans la planification des pods en raison d'appels d'API limités générés par le plug-in VPC CNI (ipamd) visant à allouer des adresses privées en temps opportun. IPv4 Il est également connu que les boutons avancés du plug-in VPC-CNI ajustent inutilement WARM_IP/ENI, MINIMUM_IP.

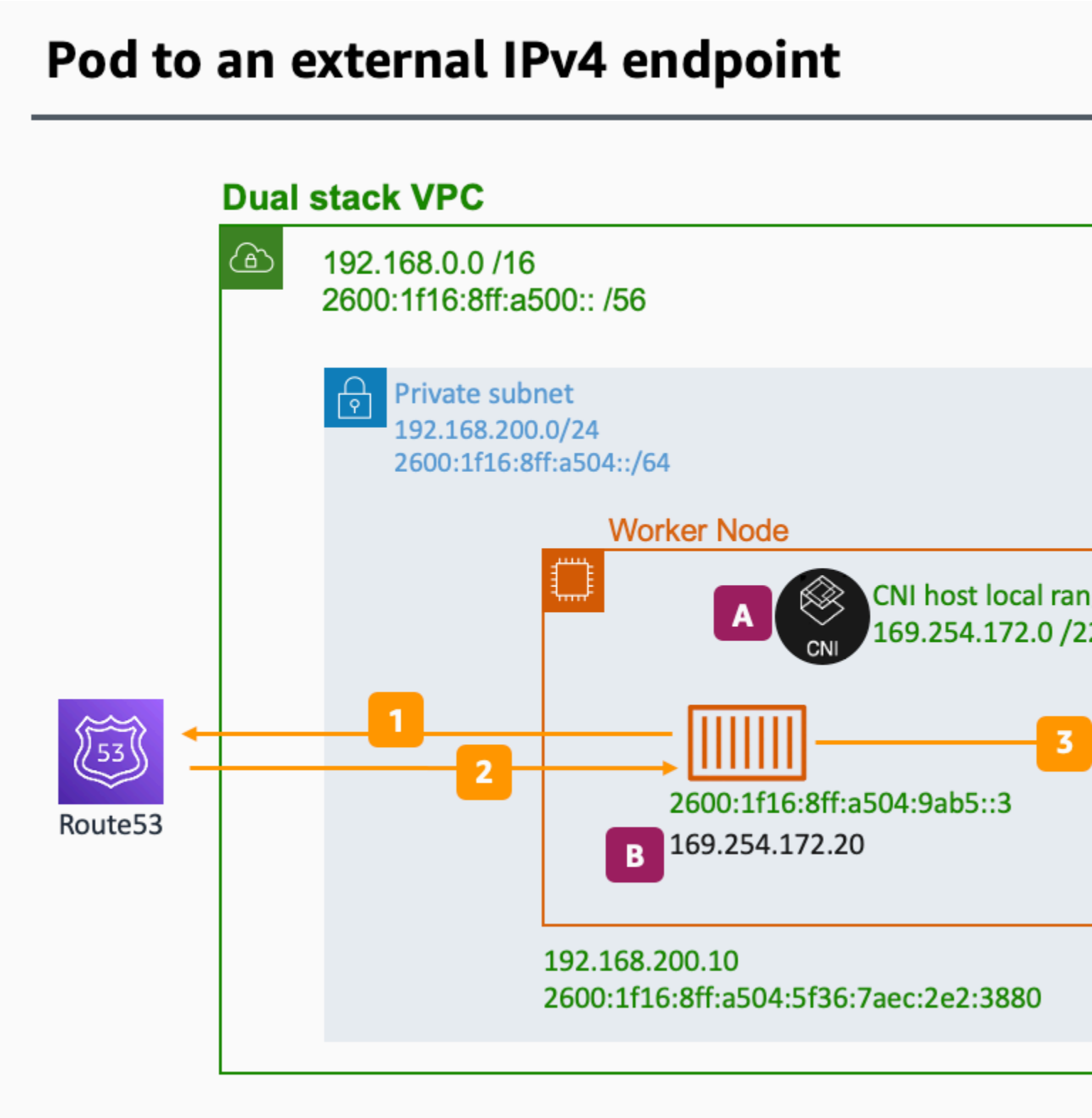
Le schéma suivant zoome sur une interface réseau élastique (IPv6 ENI) d'un nœud de travail :



Chaque nœud de travail EKS se voit attribuer des IPv6 adresses IPv4 et des entrées DNS correspondantes. Pour un nœud de travail donné, une seule IPv4 adresse du sous-réseau à double pile est consommée. Le support EKS vous IPv6 permet de communiquer avec les IPv4 points de terminaison (AWS, sur site, Internet) par le biais d'un modèle de sortie uniquement basé sur les opinions de sortie. IPv4 EKS implémente un plug-in CNI local à l'hôte, secondaire au plug-in VPC CNI, qui alloue et configure une adresse pour un Pod. IPv4 Le plugin CNI configure une IPv4 adresse non routable spécifique à l'hôte pour un Pod de la plage 169.254.172.0/22. L' IPv4 adresse

attribuée au Pod est unique au nœud de travail et n'est pas annoncée au-delà du nœud de travail. 169.254.172.0/22 fournit jusqu'à 1 024 adresses uniques pouvant prendre en charge des types d'instances de grande taille. IPv4

Le schéma suivant illustre le flux d'un IPv6 Pod se connectant à un IPv4 point de terminaison situé en dehors des limites du cluster (hors Internet) :

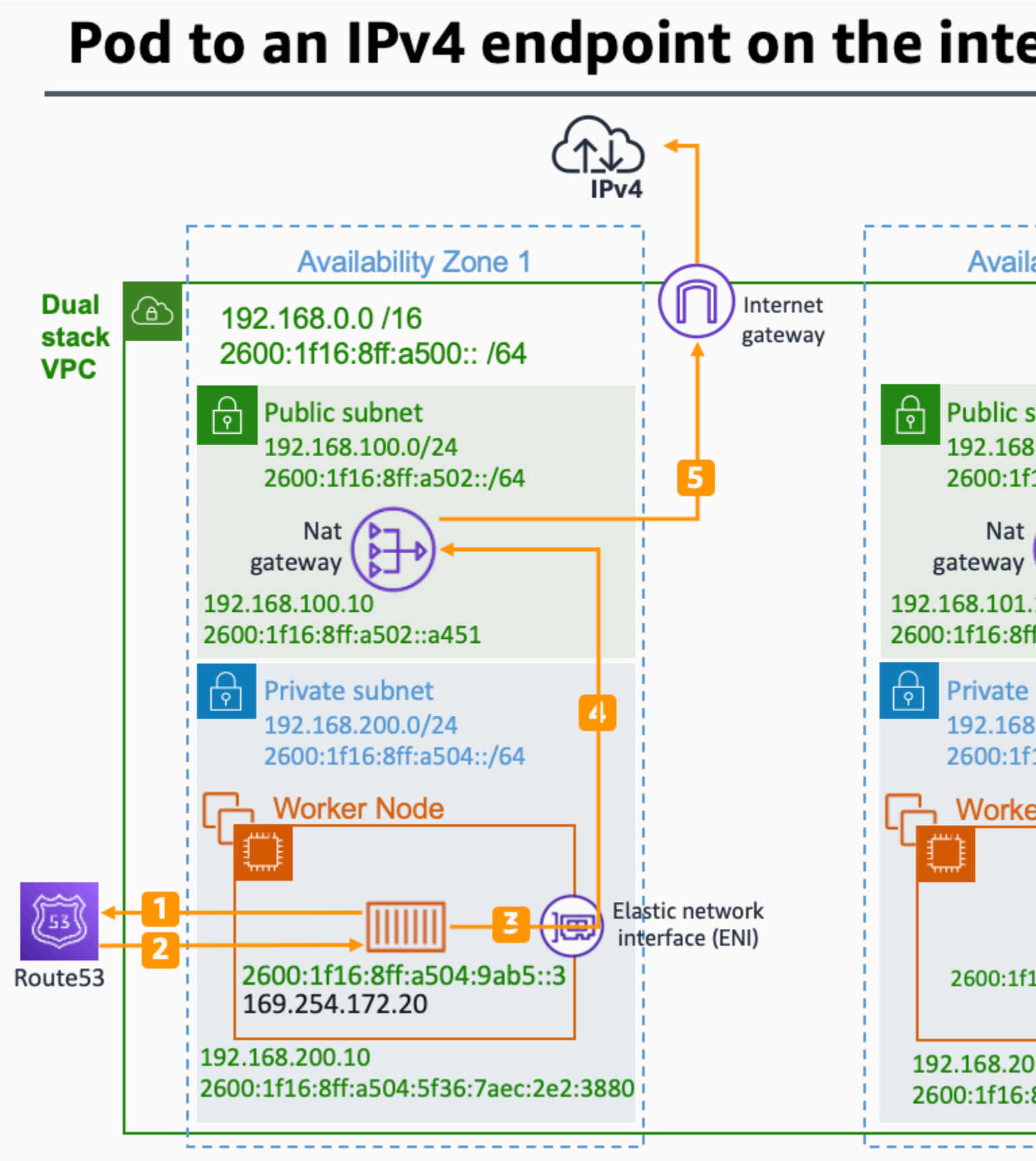


Dans le schéma ci-dessus, Pods effectuera une recherche DNS pour le point de terminaison et, après réception d'une réponse IPv4 « A », l'adresse IPv4 unique du nœud du Pod est traduite par

traduction d'adresse réseau source (SNAT) en adresse IPv4 privée (VPC) de l'interface réseau principale attachée au nœud de travail EC2.

> **Note**
>
> Le modèle ci-dessus nécessite DNS64 d'être désactivé sur les sous-réseaux sur lesquels les EKS/ IPv6 Pods sont exécutés. Lorsqu'il DNS64 est activé, le résolveur DNS renvoie une IPv6 adresse synthétisée pour les points de IPv4 terminaison uniquement, ainsi qu'une adresse. IPv4 Par conséquent, le trafic passe par la NAT64 fonctionnalité de la passerelle NAT (si elle est incluse dans l'architecture) au lieu de rester dans le VPC, comme indiqué dans le schéma ci-dessus. Cela peut entraîner une utilisation inattendue de la passerelle NAT et des coûts associés.

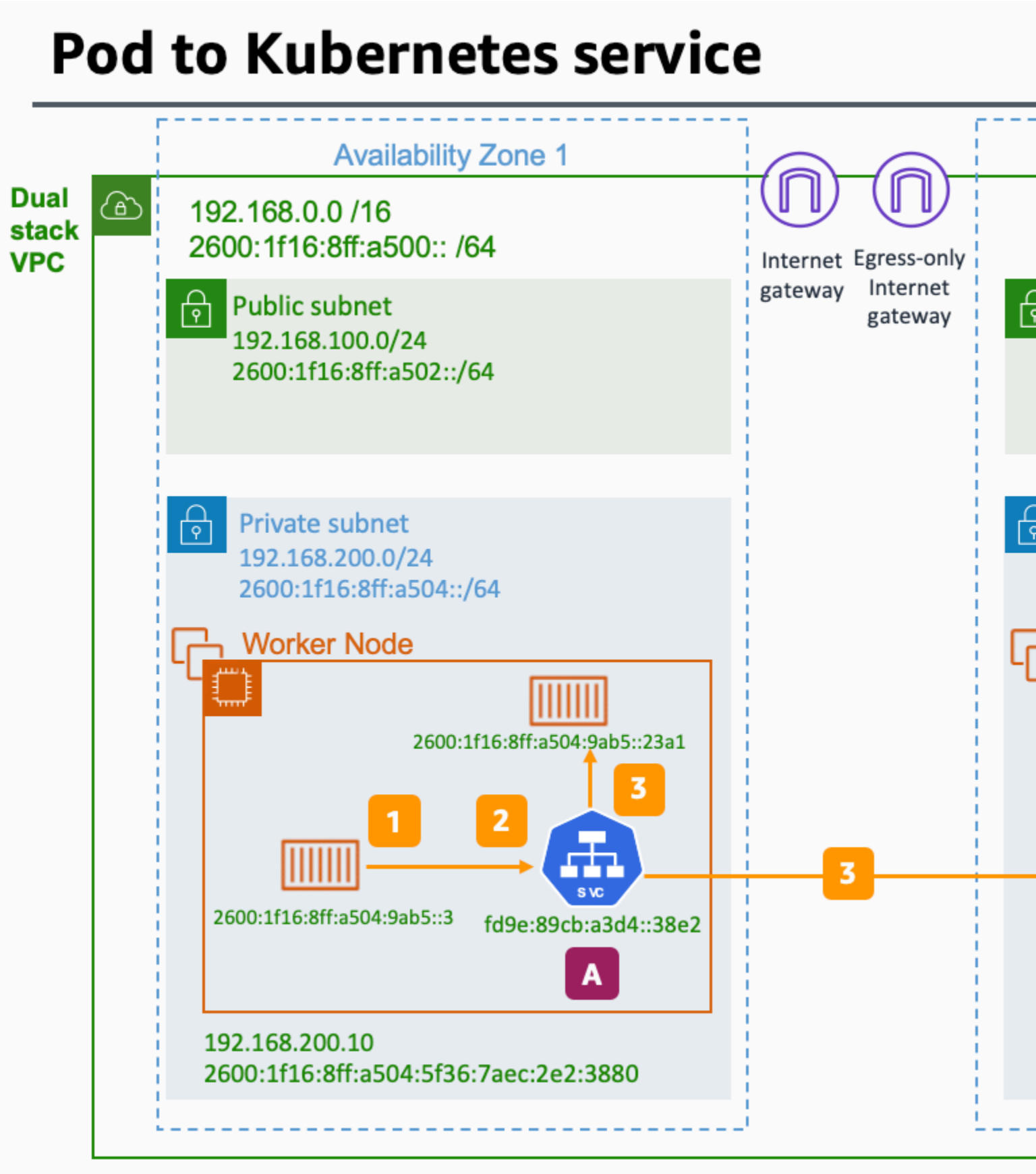
Les EKS/ IPv6 Pods devront également se connecter aux IPv4 points de terminaison via Internet à l'aide d' IPv4 adresses publiques, pour qu'un flux similaire existe. Le schéma suivant illustre le flux d'un IPv6 pod se connectant à un IPv4 point de terminaison situé en dehors des limites du cluster (routable par Internet) :


Pod to an IPv4 endpoint on the inte
IPv4
Availability Zone 1
Dual stack VPC
192.168.0.0 /16
2600:1f16:8ff:a500:: /64
Internet gateway
Public subnet
192.168.100.0/24
2600:1f16:8ff:a502::/64
5
Nat gateway
192.168.100.10
2600:1f16:8ff:a502::a451
Private subnet
192.168.200.0/24
2600:1f16:8ff:a504::/64
4
Worker Node
1
2
3
Elastic network interface (ENI)
Route53
2600:1f16:8ff:a504:9ab5::3
169.254.172.20
192.168.200.10
2600:1f16:8ff:a504:5f36:7aec:2e2:3880

Dans le schéma ci-dessus, Pods effectuera une recherche DNS pour le point de terminaison et, après réception d'une réponse IPv4 « A », l'adresse IPv4 unique du nœud du Pod est traduite par traduction d'adresse réseau source (SNAT) en adresse IPv4 privée (VPC) de l'interface réseau principale attachée au nœud de travail EC2. L'adresse IPv4 du pod (IPv4 source : adresse IP principale EC2) est ensuite routée vers la passerelle NAT IPv4 où l'adresse IP principale EC2 est traduite (SNAT) en une adresse IP publique routable sur Internet valide (adresse IP IPv4 publique assignée à la passerelle NAT).

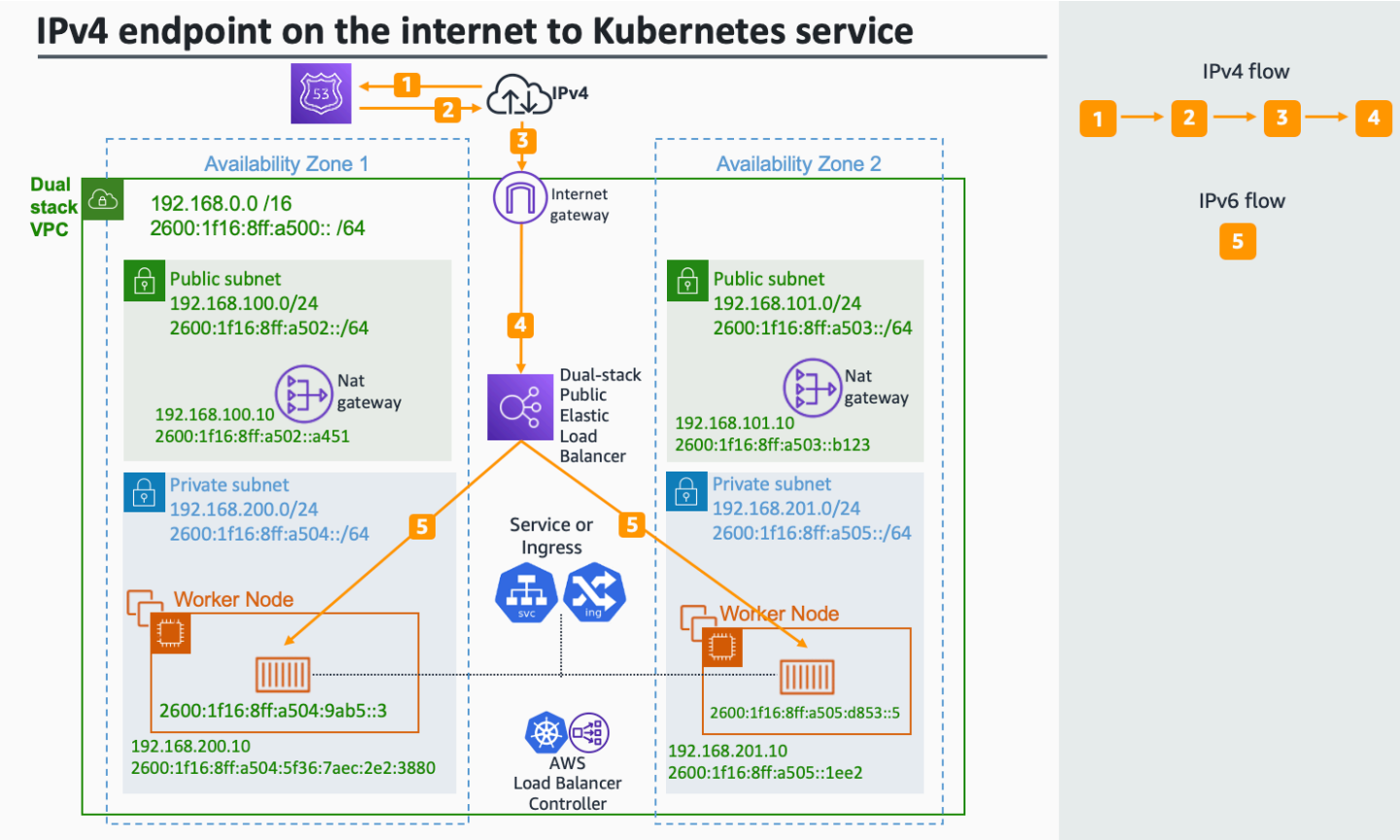
Toute Pod-to-Pod communication entre les nœuds utilise toujours une IPv6 adresse. Le VPC CNI configure iptables pour qu'il le gère IPv6 tout en bloquant les connexions. IPv4

Les services Kubernetes ne recevront que des adresses IPv6 (ClusterIP) provenant d'adresses locales uniques (ULA). IPv6 Le CIDR du service ULA pour un IPv6 cluster est automatiquement attribué lors de la phase de création du cluster EKS et ne peut pas être modifié. Le schéma suivant illustre le flux de service Pod to Kubernetes :


Pod to Kubernetes service
Availability Zone 1
Dual stack VPC
192.168.0.0 /16
2600:1f16:8ff:a500:: /64
Internet gateway
Egress-only Internet gateway
Public subnet
192.168.100.0/24
2600:1f16:8ff:a502::/64
Private subnet
192.168.200.0/24
2600:1f16:8ff:a504::/64
Worker Node
2600:1f16:8ff:a504:9ab5::23a1
1
2
3
3
SVC
2600:1f16:8ff:a504:9ab5::3
fd9e:89cb:a3d4::38e2
A
192.168.200.10
2600:1f16:8ff:a504:5f36:7aec:2e2:3880

Les services sont accessibles à Internet à l'aide d'un équilibreur de charge AWS. L'équilibreur de charge reçoit le public IPv4 et les IPv6 adresses, c'est-à-dire un équilibreur de charge à double pile. Pour les IPv4 clients accédant aux services Kubernetes du IPv6 cluster, l'équilibreur de charge s'occupe de la traduction. IPv4 IPv6

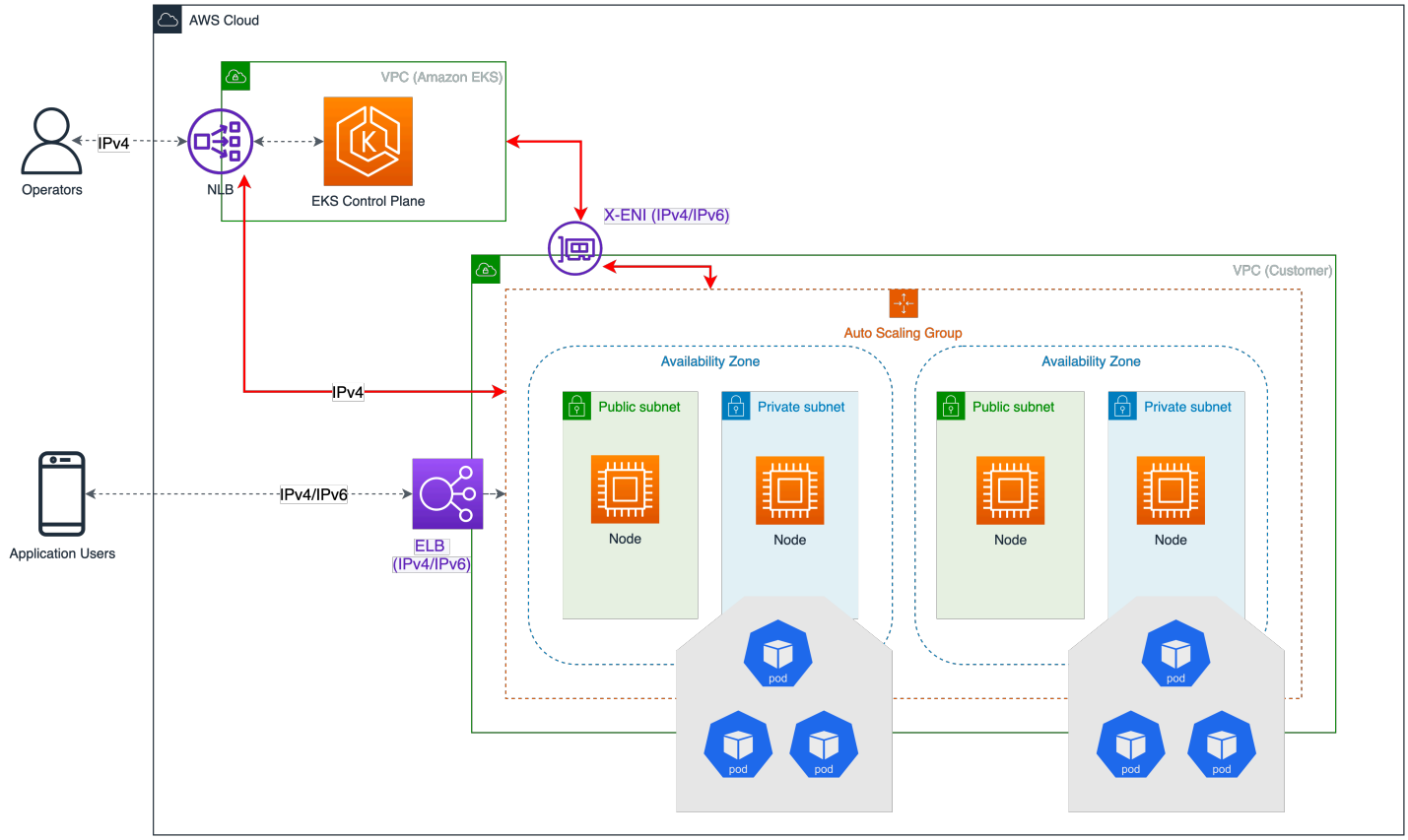
Amazon EKS recommande d'exécuter des nœuds de travail et des pods dans des sous-réseaux privés. Vous pouvez créer des équilibreurs de charge publics dans les sous-réseaux publics qui équilibrent la charge du trafic vers les pods exécutés sur des nœuds situés dans des sous-réseaux privés. Le schéma suivant montre un IPv4 internaute accédant à un service basé sur EKS/ IPv6 Ingress :



> **Note**
>
> Le modèle ci-dessus nécessite de déployer la version la plus récente du contrôleur d'équilibrage de charge AWS

Communication avec le plan de données EKS Control Plane

EKS fournira Cross-Account ENIs (X-ENIs) en mode double pile (IPv4/IPv6). Les composants des nœuds Kubernetes tels que kubelet et kube-proxy sont configurés pour prendre en charge le dual stack. Kubelet et kube-proxy s'exécutent en mode HostNetwork et se lient à la fois IPv4 aux IPv6 adresses associées à l'interface réseau principale d'un nœud. Le serveur API Kubernetes communique avec les pods et les composants des nœuds via le X- is based. ENIs IPv6 Les pods communiquent avec les serveurs API via le XENIs, et la communication entre les pods et les serveurs API utilise toujours le mode. IPv6



Recommandations

Planification basée sur les ressources informatiques

Un seul IPv6 préfixe suffit pour exécuter de nombreux pods sur un seul nœud. Cela supprime également efficacement les limites ENI et IP sur le nombre maximum de pods sur un nœud. Bien que IPv6 la dépendance directe à l'égard des Max-pods soit supprimée, lorsque vous utilisez des pièces jointes de préfixes avec des types d'instance plus petits tels que le m5.large, vous risquez d'épuiser les ressources du processeur et de la mémoire de l'instance bien avant d'épuiser ses adresses IP.

Vous devez définir manuellement la valeur maximale de pod recommandée par EKS si vous utilisez des groupes de nœuds autogérés ou un groupe de nœuds gérés avec un ID AMI personnalisé.

Vous pouvez utiliser la formule suivante pour déterminer le nombre maximum de pods que vous pouvez déployer sur un nœud pour un cluster IPv6 EKS.

```
((Number of network interfaces for instance type (number of prefixes per network
 interface-1)* 16) + 2
```

```
((3 ENIs)_((10 secondary IPs per ENI-1)_ 16)) + 2 = 460 (real)
```

Les groupes de nœuds gérés calculent automatiquement le nombre maximum de pods pour vous. Évitez de modifier la valeur recommandée par EKS pour le nombre maximum de pods afin d'éviter les échecs de planification des pods en raison de limitations de ressources.

Évaluer l'objectif du réseau personnalisé existant

Si la mise en réseau personnalisée est actuellement activée, Amazon EKS recommande de réévaluer vos besoins en la matière avec IPv6. Si vous avez choisi d'utiliser un réseau personnalisé pour résoudre le problème d' IPv4 épuisement, cela n'est plus nécessaire avec IPv6. Si vous utilisez un réseau personnalisé pour répondre à une exigence de sécurité, telle qu'un réseau distinct pour les nœuds et les pods, nous vous encourageons à soumettre une demande de feuille de route EKS.

Capsules Fargate dans un cluster EKS/ IPv6

EKS prend en IPv6 charge les pods fonctionnant sur Fargate. Les pods exécutés sur Fargate IPv6 consommeront des IPv4 adresses privées routables VPC découpées à partir des plages d'adresses CIDR VPC (). IPv4 IPv6 En termes simples, la densité de votre cluster de EKS/Fargate Pods sera limitée aux IPv6 adresses IPv4 et aux adresses disponibles. Il est recommandé de dimensionner votre double pile en fonction subnets/VPC CIDRs de sa croissance future. Vous ne pourrez pas planifier de nouveaux Fargate Pods si le sous-réseau sous-jacent ne contient pas d'adresse disponible, quelles que soient les adresses IPv4 disponibles. IPv6

Déployez le contrôleur AWS Load Balancer (LBC)

Le contrôleur de service Kubernetes intégré à l'arbre en amont n'est pas pris en charge. IPv6 Nous vous recommandons d'utiliser la version la plus récente du module complémentaire AWS Load Balancer Controller. Le LBC ne déploiera un NLB à double pile ou un ALB à double pile qu'après avoir consommé la définition de service/ingress Kubernetes correspondante

annotée avec : et `"alb.ingress.kubernetes.io/ip-address-type: dualstack"` `"alb.ingress.kubernetes.io/target-type: ip"`

Mise en réseau personnalisée

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Par défaut, Amazon VPC CNI attribuera aux Pods une adresse IP sélectionnée dans le sous-réseau principal. Le sous-réseau principal est le CIDR du sous-réseau auquel l'ENI principal est attaché, généralement le sous-réseau du nœud ou de l'hôte.

Si le CIDR du sous-réseau est trop petit, le CNI risque de ne pas être en mesure d'acquérir suffisamment d'adresses IP secondaires à attribuer à vos pods. Il s'agit d'un défi courant pour les IPv4 clusters EKS.

La mise en réseau personnalisée est l'une des solutions à ce problème.

La mise en réseau personnalisée résout le problème d'épuisement des adresses IP en attribuant le nœud et le pod IPs à partir des espaces d'adressage VPC secondaires (CIDR). Le support réseau personnalisé prend en charge les ressources ENIConfig personnalisées. ENIConfig Cela inclut une plage d'adresses CIDR de sous-réseau alternative (découpée à partir d'un CIDR VPC secondaire), ainsi que le ou les groupes de sécurité auxquels appartiendront les pods. Lorsque le réseau personnalisé est activé, le VPC CNI crée un réseau secondaire ENIs dans le sous-réseau défini ci-dessous. ENIConfig Le CNI attribue aux Pods des adresses IP à partir d'une plage CIDR définie dans un CRD. ENIConfig

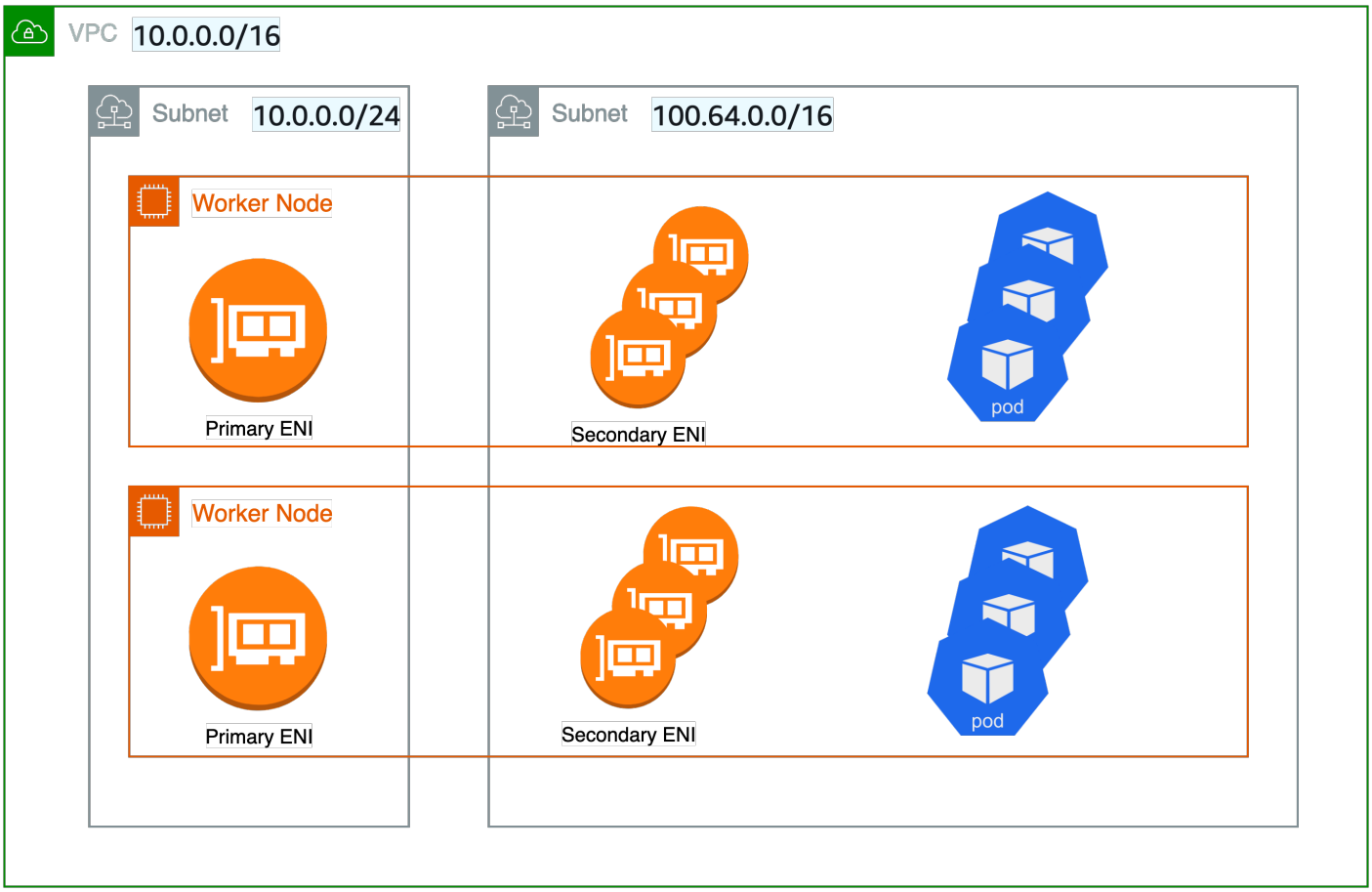
Comme l'ENI principal n'est pas utilisé par les réseaux personnalisés, le nombre maximum de pods que vous pouvez exécuter sur un nœud est inférieur. Les pods du réseau hôte continuent d'utiliser l'adresse IP attribuée à l'ENI principal. En outre, l'ENI principal est utilisé pour gérer la traduction du réseau source et acheminer le trafic des Pods en dehors du nœud.

Exemple de configuration

Bien que le réseau personnalisé accepte une plage VPC valide pour la plage CIDR secondaire, nous vous recommandons d'utiliser CIDRs (/16) depuis l'espace CG-NAT, c'est-à-dire 100.64.0.0/10

ou 198.19.0.0/16, car ces plages sont moins susceptibles d'être utilisées dans un environnement d'entreprise que les autres plages. RFC1918 Pour plus d'informations sur les associations de blocs CIDR autorisées et restreintes que vous pouvez utiliser avec votre VPC, IPv4 consultez la section Restrictions relatives aux associations de blocs CIDR dans la section VPC et dimensionnement des sous-réseaux de la documentation VPC.

Comme le montre le schéma ci-dessous, l'interface réseau élastique (ENI) principale du nœud de travail utilise toujours la plage d'adresses CIDR VPC principale (dans ce cas 10.0.0.0/16) mais la plage secondaire utilise ENIs la plage d'adresses CIDR VPC secondaire (dans ce cas 100.64.0.0/16). Maintenant, pour que les Pods utilisent la plage CIDR 100.64.0.0/16, vous devez configurer le plug-in CNI pour utiliser un réseau personnalisé. Vous pouvez suivre les étapes décrites ici.



Si vous souhaitez que le CNI utilise un réseau personnalisé, définissez la variable d'`AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG``true`environnement sur.

```
kubectl set env daemonset aws-node -n kube-system
 AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

Quand`AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true`, le CNI attribuera l'adresse IP du pod à partir d'un sous-réseau défini dans. `ENIConfig` La ressource `ENIConfig` personnalisée est utilisée pour définir le sous-réseau dans lequel les pods seront planifiés.

```
apiVersion : crd.k8s.amazonaws.com/v1alpha1
kind : ENIConfig
metadata:
  name: us-west-2a
spec:
  securityGroups:
    - sg-0dff111a1d11c1c11
  subnet: subnet-011b111c1f11fdf11
```

Lors de la création des ressources `ENIconfig` personnalisées, vous devrez créer de nouveaux nœuds de travail et vider les nœuds existants. Les nœuds de travail et les pods existants ne seront pas affectés.

Recommandations

Utilisez un réseau personnalisé lorsque

Nous vous recommandons d'envisager un réseau personnalisé si vous êtes IPv4 épuisé et que vous ne pouvez pas IPv6 encore l'utiliser. La prise en charge de l'RFC6598espace par Amazon EKS vous permet de faire évoluer les pods au-delà de la résolution des RFC1918problèmes d'épuisement. Envisagez d'utiliser la délégation de préfixes avec un réseau personnalisé pour augmenter la densité des pods sur un nœud.

Vous pouvez envisager une mise en réseau personnalisée si vous avez des exigences de sécurité pour exécuter des Pods sur un réseau différent avec des exigences de groupe de sécurité différentes. Lorsque la mise en réseau personnalisée est activée, les pods utilisent des sous-réseaux ou des groupes de sécurité différents de ENIConfig ceux définis dans l'interface réseau principale du nœud.

La mise en réseau personnalisée est en effet une option idéale pour déployer plusieurs clusters et applications EKS afin de connecter les services de centre de données sur site. Vous pouvez augmenter le nombre d'adresses privées (RFC1918) accessibles à EKS dans votre VPC pour des services tels qu'Amazon Elastic Load Balancing et NAT-GW, tout en utilisant un espace CG-NAT non routable pour vos pods sur plusieurs clusters. La mise en réseau personnalisée avec la passerelle de transit et un VPC à services partagés (y compris des passerelles NAT entre plusieurs zones de disponibilité pour une haute disponibilité) vous permet de fournir des flux de trafic évolutifs et

prévisibles. Ce billet de blog décrit un modèle architectural qui constitue l'une des méthodes les plus recommandées pour connecter les EKS Pods à un réseau de centre de données à l'aide d'un réseau personnalisé.

Évitez les réseaux personnalisés lorsque

Prêt à être mis en œuvre IPv6

La mise en réseau personnalisée peut atténuer les problèmes d'épuisement des adresses IP, mais elle nécessite des frais opérationnels supplémentaires. Si vous déployez actuellement un VPC à double pile (IPv4/IPv6) ou si votre plan IPv6 inclut un support, nous vous recommandons d' IPv6 implémenter des clusters à la place. Vous pouvez configurer des clusters IPv6 EKS et migrer vos applications. Dans un cluster IPv6 EKS, Kubernetes et Pods obtiennent une IPv6 adresse et peuvent communiquer entre eux et avec les points de terminaison. IPv4 IPv6 Consultez les meilleures pratiques relatives à l'exécution de clusters IPv6 EKS.

Espace CG-NAT épuisé

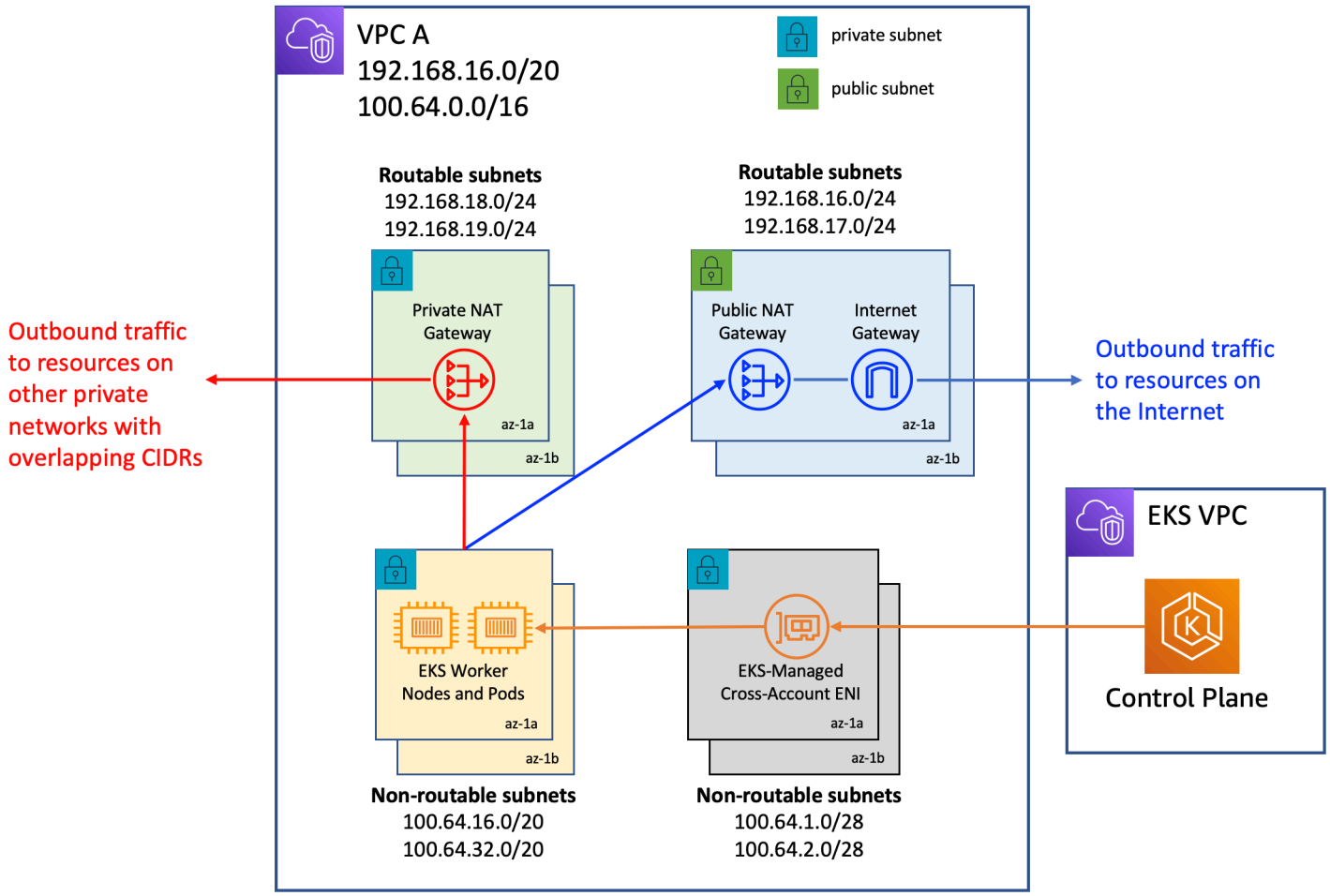
En outre, si vous utilisez CIDRs actuellement l'espace CG-NAT ou si vous ne parvenez pas à lier un CIDR secondaire à votre VPC de cluster, vous devrez peut-être explorer d'autres options, telles que l'utilisation d'un CNI alternatif. Nous vous recommandons vivement d'obtenir un support commercial ou de posséder les connaissances internes nécessaires pour déboguer et soumettre des correctifs au projet de plugin open source CNI. Consultez le guide de l'utilisateur d'Alternate CNI Plugins pour plus de détails.

Utiliser une passerelle NAT privée

Amazon VPC propose désormais des fonctionnalités de passerelle NAT privée. La passerelle NAT privée d'Amazon permet aux instances situées dans des sous-réseaux privés de se connecter à d'autres réseaux VPCs et à des réseaux locaux qui se chevauchent. CIDRs Envisagez d'utiliser la méthode décrite dans ce billet de blog pour utiliser une passerelle NAT privée afin de résoudre les problèmes de communication liés aux charges de travail EKS causés par le chevauchement CIDRs, une plainte importante exprimée par nos clients. La mise en réseau personnalisée ne peut pas résoudre à elle seule les difficultés liées au chevauchement des CIDR, ce qui aggrave les problèmes de configuration.

L'architecture réseau utilisée dans la mise en œuvre de ce billet de blog suit les recommandations de la section Activer la communication entre les réseaux qui se chevauchent dans la documentation Amazon VPC. Comme démontré dans ce billet de blog, vous pouvez étendre l'utilisation de la passerelle NAT privée en conjonction avec les RFC6598 adresses pour gérer les problèmes

d'épuisement des adresses IP privées des clients. Les clusters EKS et les nœuds de travail sont déployés dans la plage d'adresses CIDR secondaire VPC non routable 100.64.0.0/16, tandis que la passerelle NAT privée et la passerelle NAT sont déployées dans les plages d'adresses CIDR routables. RFC1918 Le blog explique comment une passerelle de transit est utilisée pour se connecter VPCs afin de faciliter la communication entre des plages d'adresses CIDR VPCs non routables qui se chevauchent. Dans les cas d'utilisation dans lesquels les ressources EKS situées dans la plage d'adresses non routable d'un VPC doivent communiquer avec d'autres ressources dont les plages d'adresses VPCs ne se chevauchent pas, les clients ont la possibilité d'utiliser le peering VPC pour les interconnecter. VPCs Cette méthode pourrait permettre de réaliser des économies, car tous les transferts de données au sein d'une zone de disponibilité via une connexion d'appairage VPC sont désormais gratuits.

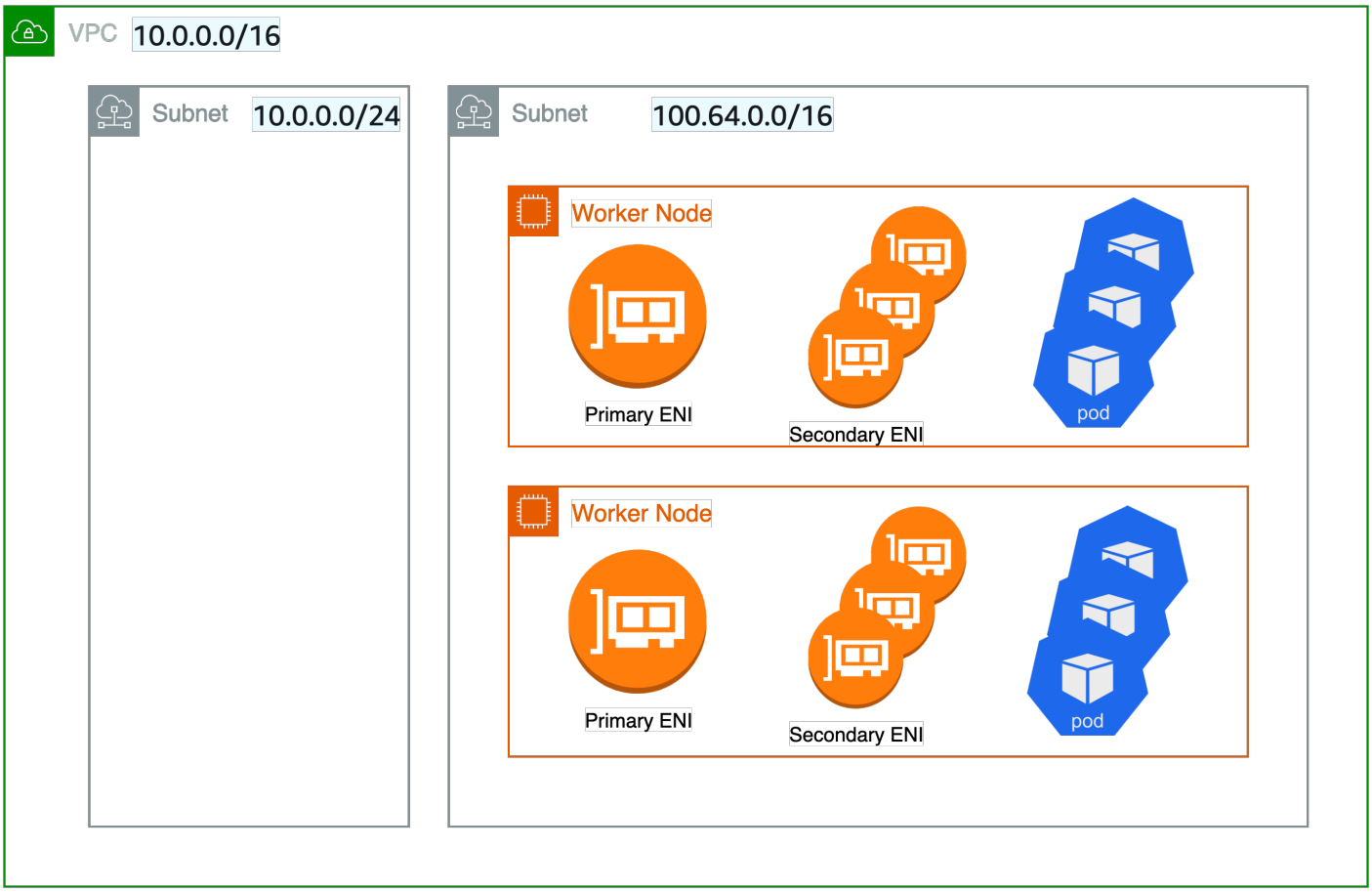


Réseau unique pour les nœuds et les pods

Si vous devez isoler vos nœuds et vos pods sur un réseau spécifique pour des raisons de sécurité, nous vous recommandons de déployer des nœuds et des pods sur un sous-réseau à partir d'un bloc d'adresse CIDR secondaire plus important (par exemple 100.64.0.0/8). Après l'installation

du nouveau CIDR dans votre VPC, vous pouvez déployer un autre groupe de nœuds à l'aide du CIDR secondaire et vider les nœuds d'origine pour redéployer automatiquement les pods vers les nouveaux nœuds de travail. Pour plus d'informations sur la façon de l'implémenter, consultez ce billet de blog.

La mise en réseau personnalisée n'est pas utilisée dans la configuration représentée dans le schéma ci-dessous. Les nœuds de travail Kubernetes sont plutôt déployés sur des sous-réseaux de la plage d'adresses CIDR VPC secondaire de votre VPC, telle que 100.64.0.0/10. Vous pouvez continuer à exécuter le cluster EKS (le plan de contrôle restera sur le plan d'origine)subnet/s), but the nodes and Pods will be moved to a secondary subnet/s. Il s'agit d'une autre technique, bien que peu conventionnelle, pour atténuer le risque d'épuisement de la propriété intellectuelle dans un VPC. Nous proposons de vider les anciens nœuds avant de redéployer les pods vers les nouveaux nœuds de travail.



Automatisez la configuration avec des étiquettes de zone de disponibilité

Vous pouvez permettre à Kubernetes d'appliquer automatiquement la zone de disponibilité (AZ) correspondante ENIConfig pour le nœud de travail.

Kubernetes ajoute automatiquement la balise `topology.kubernetes.io/zone`à vos nœuds de travail. Amazon EKS recommande d'utiliser la zone de disponibilité comme nom de configuration ENI lorsque vous ne disposez que d'un seul sous-réseau secondaire (CIDR alternatif) par AZ. Vous pouvez ensuite définir l'étiquette utilisée pour découvrir le nom de configuration ENI sur`topology.kubernetes.io/zone`. Notez que la balise `failure-domain.beta.kubernetes.io/zone` est obsolète et remplacée par la balise. `topology.kubernetes.io/zone`

1. Définissez `name` le champ sur la zone de disponibilité de votre VPC.
2. Activez la configuration automatique via la commande suivante
3. Définissez l'étiquette de configuration à l'aide de la commande suivante

```
kubectl set env daemonset aws-node -n kube-system
 "AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true"
kubectl set env daemonset aws-node -n kube-system
 "ENI_CONFIG_LABEL_DEF=topology.kubernetes.io/zone"
```

Si vous avez plusieurs sous-réseaux secondaires par zone de disponibilité, vous devez en créer un spécifique`ENI_CONFIG_LABEL_DEF`. Vous pouvez envisager de configurer des nœuds `ENI_CONFIG_LABEL_DEF` en tant que `k8s.amazonaws.com/eniConfig`et d'étiqueter des nœuds avec des noms ENIConfig personnalisés, tels que `k8s.amazonaws.com/eniConfig=us-west-2a-subnet-1`et. `k8s.amazonaws.com/eniConfig=us-west-2a-subnet-2`

Remplacez les pods lors de la configuration du réseau secondaire

L'activation du réseau personnalisé ne modifie pas les nœuds existants. La mise en réseau personnalisée est une action perturbatrice. Plutôt que de remplacer progressivement tous les nœuds de travail de votre cluster après avoir activé la mise en réseau personnalisée, nous vous suggérons de mettre à jour le CloudFormation modèle AWS du guide de démarrage EKS avec une ressource personnalisée qui appelle une fonction Lambda pour mettre à jour le `aws-node` Daemonset avec la variable d'environnement afin de permettre une mise en réseau personnalisée avant le provisionnement des nœuds de travail.

Si des nœuds de votre cluster étaient équipés de pods en cours d'exécution avant de passer à la fonctionnalité réseau CNI personnalisée, vous devez boucler et vider les nœuds pour arrêter correctement les pods, puis mettre fin aux nœuds. Seuls les nouveaux nœuds correspondant à l' ENIConfig étiquette ou aux annotations utilisent un réseau personnalisé. Par conséquent, les

pods planifiés sur ces nouveaux nœuds peuvent se voir attribuer une adresse IP à partir du CIDR secondaire.

Calculer le nombre maximum de pods par nœud

Étant donné que l'ENI principal du nœud n'est plus utilisé pour attribuer les adresses IP des pods, le nombre de pods que vous pouvez exécuter sur un type d'instance EC2 donné diminue. Pour contourner cette limitation, vous pouvez utiliser l'attribution de préfixes avec un réseau personnalisé. Avec l'attribution d'un préfixe, chaque adresse IP secondaire est remplacée par un préfixe /28 sur l'adresse secondaire. ENIs

Tenez compte du nombre maximum de pods pour une instance m5.large dotée d'un réseau personnalisé.

Le nombre maximum de pods que vous pouvez exécuter sans attribution de préfixe est de 29

- `3 ENIs - 1) * (10 secondary IPs per ENI - 1 + 2 = 20`

L'activation des pièces jointes par préfixe augmente le nombre de pods à 290.

- `(3 ENIs - 1) * ((10 secondary IPs per ENI - 1) * 16 + 2 = 290`

Cependant, nous vous suggérons de définir max-pods sur 110 plutôt que sur 290 car l'instance ne contient qu'un nombre assez restreint de virtuels. CPUs Pour les instances de plus grande taille, EKS recommande une valeur maximale de 250 pods. Lorsque vous utilisez des pièces jointes de préfixes avec des types d'instance plus petits (par exemple m5.large), il est possible que vous épuisiez les ressources du processeur et de la mémoire de l'instance bien avant ses adresses IP.

> **Note**
>
> Lorsque le préfixe CNI attribue un préfixe /28 à une ENI, il doit s'agir d'un bloc contigu d'adresses IP. Si le sous-réseau à partir duquel le préfixe est généré est très fragmenté, l'attachement du préfixe peut échouer. Vous pouvez éviter que cela ne se produise en créant un nouveau VPC dédié pour le cluster ou en réservant au sous-réseau un ensemble de CIDR exclusivement pour les pièces jointes de préfixes. Consultez la section Subnet CIDR reservations pour plus d'informations à ce sujet.

Identifier l'utilisation existante de l'espace CG-NAT

La mise en réseau personnalisée vous permet d'atténuer le problème d'épuisement des adresses IP, mais elle ne peut pas résoudre tous les problèmes. Si vous utilisez déjà l'espace CG-NAT pour votre cluster, ou si vous n'êtes tout simplement pas en mesure d'associer un CIDR secondaire à votre VPC de cluster, nous vous suggérons d'explorer d'autres options, comme l'utilisation d'un CNI alternatif ou le passage à des clusters. IPv6

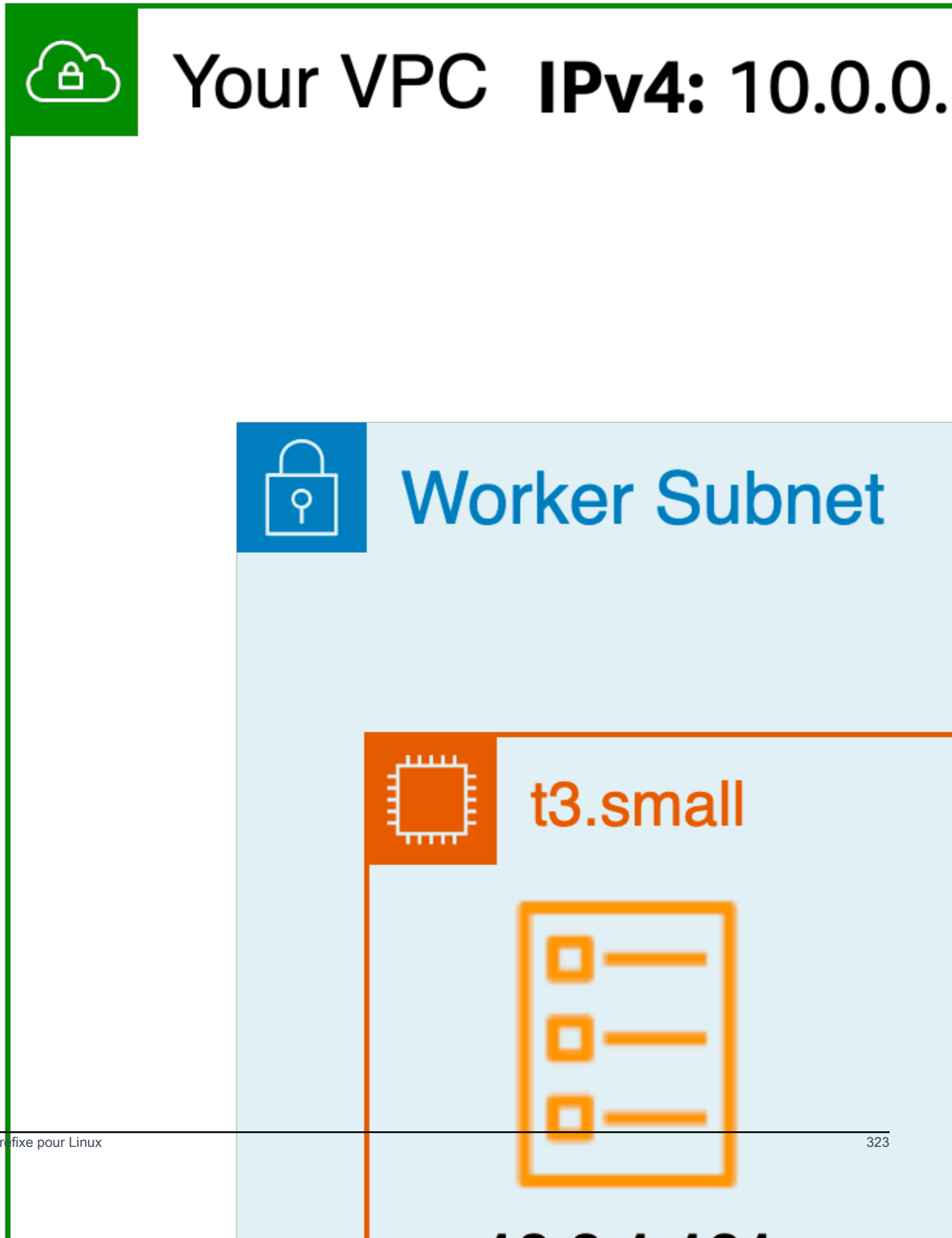
Mode préfixe pour Linux

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Amazon VPC CNI attribue des préfixes réseau aux interfaces réseau Amazon EC2 afin d'augmenter le nombre d'adresses IP disponibles pour les nœuds et d'augmenter la densité de pods par nœud. Vous pouvez configurer la version 1.9.0 ou ultérieure du module complémentaire Amazon VPC CNI pour IPv4 attribuer IPv6 CIDRs et au lieu d'attribuer des adresses IP secondaires individuelles aux interfaces réseau.

Le mode préfixe est activé par défaut sur les IPv6 clusters et est la seule option prise en charge. Le VPC CNI attribue un IPv6 préfixe /80 à un emplacement sur un ENI. Reportez-vous à la IPv6 section de ce guide pour plus d'informations.

Avec le mode d'attribution de préfixes, le nombre maximum d'interfaces réseau élastiques par type d'instance reste le même, mais vous pouvez désormais configurer Amazon VPC CNI pour attribuer des préfixes d'adresse /28 (16 adresses IP IPv4), au lieu d'attribuer des adresses IPv4 individuelles aux emplacements des interfaces réseau. Lorsqu'il `ENABLE_PREFIX_DELEGATION` est défini sur true VPC, le CNI alloue une adresse IP à un Pod à partir du préfixe attribué à un ENI. Veuillez suivre les instructions mentionnées dans le guide de l'utilisateur d'EKS pour activer le mode Prefix IP.

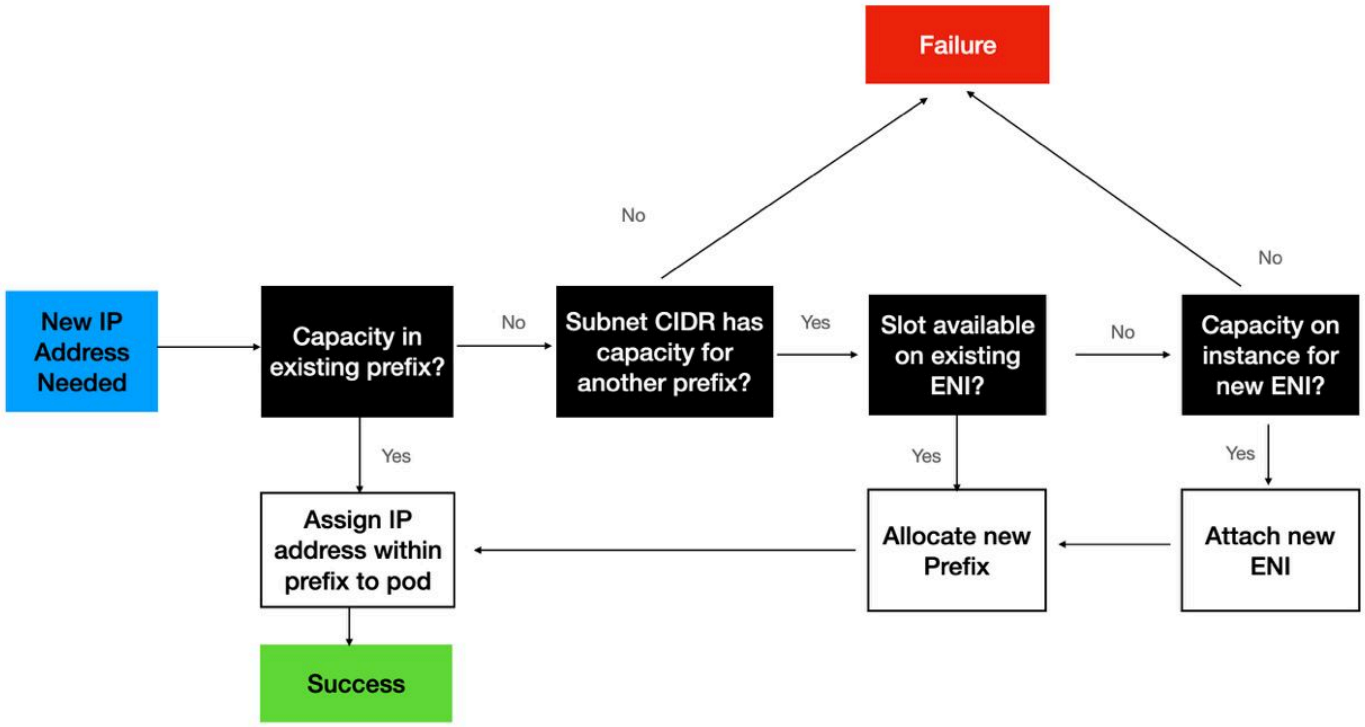

Your VPC
IPv4: 10.0.0.
Worker Subnet
t3.small

Le nombre maximal d'adresses IP que vous pouvez attribuer à une interface réseau dépend du type d'instance. Chaque préfixe que vous attribuez à une interface réseau est considéré comme une adresse IP unique. Par exemple, une instance `c5.large` a une limite de `10` adresses IPv4 par interface réseau. Chaque interface réseau de cette instance possède une adresse IPv4 principale. Si une interface réseau ne possède pas d'adresses IPv4 secondaires, vous pouvez attribuer jusqu'à 9 préfixes à l'interface réseau. Pour chaque adresse IPv4 supplémentaire que vous attribuez à une interface réseau, vous pouvez attribuer un préfixe de moins à l'interface réseau. Consultez la documentation AWS EC2 sur les adresses IP par interface réseau par type d'instance et sur l'attribution de préfixes aux interfaces réseau.

Lors de l'initialisation du nœud de travail, le VPC CNI attribue un ou plusieurs préfixes à l'ENI principal. Le CNI préalloue un préfixe pour accélérer le démarrage du pod en maintenant un pool chaud. Le nombre de préfixes à conserver dans une piscine chaude peut être contrôlé en définissant des variables d'environnement.

- `WARM_PREFIX_TARGET`, le nombre de préfixes à allouer en sus des besoins actuels.
- `WARM_IP_TARGET`, le nombre d'adresses IP à attribuer en sus des besoins actuels.
- `MINIMUM_IP_TARGET`, le nombre minimum d'adresses IP pouvant être disponibles à tout moment.
- `WARM_IP_TARGET`et `MINIMUM_IP_TARGET` si défini, il `WARM_PREFIX_TARGET` remplacera.

Au fur et à mesure que de plus en plus de Pods sont prévus, des préfixes supplémentaires seront demandés pour l'ENI existant. Tout d'abord, le VPC CNI tente d'attribuer un nouveau préfixe à une ENI existante. Si l'ENI est à pleine capacité, le VPC CNI tente d'attribuer une nouvelle ENI au nœud. Le nouveau ENIs sera attaché jusqu'à ce que la limite maximale d'ENI (définie par le type d'instance) soit atteinte. Lorsqu'une nouvelle ENI est attachée, ipamd alloue un ou plusieurs préfixes nécessaires pour maintenir le paramètre `WARM_PREFIX_TARGETWARM_IP_TARGET`, et`MINIMUM_IP_TARGET`.



Recommandations

Utilisez le mode préfixe lorsque

Utilisez le mode préfixe si vous rencontrez un problème de densité de pods sur les nœuds de travail. Pour éviter les erreurs VPC CNI, nous vous recommandons d'examiner les sous-réseaux à la recherche de blocs d'adresses contigus pour le préfixe /28 avant de passer en mode préfixe. Reportez-vous à la section « Utiliser les réservations de sous-réseaux pour éviter la fragmentation des sous-réseaux (IPv4) » pour plus de détails sur les réservations de sous-réseaux.

Pour des raisons de rétrocompatibilité, la limite maximale de pods est définie pour prendre en charge le mode IP secondaire. Pour augmenter la densité des pods, veuillez spécifier la `max-pods` valeur à Kubelet et `--use-max-pods=false` en tant que donnée utilisateur pour les nœuds. Vous pouvez envisager d'utiliser le script max-pod-calculator.sh pour calculer le nombre maximum de pods recommandé par EKS pour un type d'instance donné. Reportez-vous au guide de l'utilisateur EKS pour obtenir des informations sur les utilisateurs, par exemple.

```
./max-pods-calculator.sh --instance-type m5.large --cni-version ``1.9``.0 --cni-prefix-delegation-enabled
```

Le mode d'attribution de préfixes est particulièrement pertinent pour les utilisateurs du réseau personnalisé CNI où l'ENI principal n'est pas utilisé pour les pods. Grâce à l'attribution de préfixes, vous pouvez toujours en attacher davantage IPs sur presque tous les types d'instances Nitro, même sans l'ENI principal utilisé pour les pods.

Évitez le mode préfixe lorsque

Si votre sous-réseau est très fragmenté et ne possède pas suffisamment d'adresses IP disponibles pour créer des préfixes /28, évitez d'utiliser le mode préfixe. L'attachement du préfixe peut échouer si le sous-réseau à partir duquel le préfixe est produit est fragmenté (sous-réseau très utilisé avec des adresses IP secondaires dispersées). Ce problème peut être évité en créant un nouveau sous-réseau et en réservant un préfixe.

En mode préfixe, le groupe de sécurité attribué aux nœuds de travail est partagé par les pods. Envisagez d'utiliser des groupes de sécurité pour les pods si vous avez des exigences de sécurité pour garantir la conformité en exécutant des applications ayant des exigences de sécurité réseau variables sur des ressources informatiques partagées.

Utiliser des types d'instance similaires dans le même groupe de nœuds

Votre groupe de nœuds peut contenir de nombreux types d'instances. Si le nombre maximal de pods d'une instance est faible, cette valeur est appliquée à tous les nœuds du groupe de nœuds. Envisagez d'utiliser des types d'instances similaires dans un groupe de nœuds pour optimiser l'utilisation des nœuds. Nous vous recommandons de configurer node.kubernetes.io/instance-type dans la partie relative aux exigences de l'API du fournisseur si vous utilisez Karpenter pour le dimensionnement automatique des nœuds.

> **Warning**
>
> Le nombre maximal d'espaces pour tous les nœuds d'un groupe de nœuds donné est défini par le nombre maximal d'espaces le plus faible de tous les types d'instances du groupe de nœuds.

Configurer `WARM_PREFIX_TARGET` pour conserver les IPv4 adresses

La valeur par défaut du manifeste d'installation pour `WARM_PREFIX_TARGET` est 1. Dans la plupart des cas, la valeur recommandée de 1 pour `WARM_PREFIX_TARGET` permet de combiner des temps de lancement rapides du pod tout en minimisant les adresses IP inutilisées attribuées à l'instance.

Si vous avez besoin de conserver davantage les IPv4 adresses par nœud, utilisez les `MINIMUM_IP_TARGET` paramètres `WARM_IP_TARGET` et utilisez ces paramètres, qui prévalent `WARM_PREFIX_TARGET` lorsqu'ils sont configurés. En définissant une valeur inférieure `WARM_IP_TARGET` à 16, vous pouvez empêcher le CNI de conserver un préfixe excédentaire entier attaché.

Préférez attribuer de nouveaux préfixes plutôt que de joindre une nouvelle ENI

L'attribution d'un préfixe supplémentaire à une ENI existante est une opération d'API EC2 plus rapide que la création et l'attachement d'une nouvelle ENI à l'instance. L'utilisation de préfixes améliore les performances tout en étant économe en termes d'allocation d' IPv4 adresses. L'attachement d'un préfixe prend généralement moins d'une seconde, tandis que l'attachement d'un nouvel ENI peut prendre jusqu'à 10 secondes. Dans la plupart des cas d'utilisation, le CNI n'aura besoin que d'un seul ENI par nœud de travail lorsqu'il est exécuté en mode préfixe. Si vous pouvez vous permettre (dans le pire des cas) d'utiliser jusqu'à 15 points non utilisés IPs par nœud, nous vous recommandons vivement d'utiliser le nouveau mode réseau d'attribution de préfixes et de tirer parti des gains de performances et d'efficacité qui en découlent.

Utiliser les réservations de sous-réseaux pour éviter la fragmentation des sous-réseaux () IPv4

Lorsqu'EC2 attribue un IPv4 préfixe /28 à une ENI, il doit s'agir d'un bloc contigu d'adresses IP provenant de votre sous-réseau. Si le sous-réseau à partir duquel le préfixe est généré est fragmenté (sous-réseau très utilisé avec des adresses IP secondaires dispersées), l'attachement du préfixe peut échouer et le message d'erreur suivant s'affichera dans les journaux VPC CNI :

```
failed to allocate a private IP/Prefix address: InsufficientCidrBlocks: There are not
 enough free cidr blocks in the specified subnet to satisfy the request.
```

Pour éviter la fragmentation et disposer d'un espace contigu suffisant pour créer des préfixes, vous pouvez utiliser les réservations CIDR du sous-réseau VPC pour réserver de l'espace IP au sein d'un sous-réseau à l'usage exclusif des préfixes. Une fois que vous avez créé une réservation, le plugin VPC CNI appelle EC2 APIs pour attribuer des préfixes qui sont automatiquement alloués à partir de l'espace réservé.

Il est recommandé de créer un nouveau sous-réseau, de réserver de l'espace pour les préfixes et d'activer l'attribution de préfixes avec le VPC CNI pour les nœuds de travail exécutés dans ce sous-réseau. Si le nouveau sous-réseau est dédié uniquement aux pods exécutés dans votre cluster EKS avec l'attribution de préfixe VPC CNI activée, vous pouvez ignorer l'étape de réservation du préfixe.

Évitez de rétrograder le VPC CNI

Le mode préfixe fonctionne avec les versions VPC CNI 1.9.0 et ultérieures. La rétrogradation du module complémentaire Amazon VPC CNI vers une version inférieure à la version 1.9.0 doit être évitée une fois que le mode préfixe est activé et que des préfixes ont été attribués. ENIs Vous devez supprimer et recréer des nœuds si vous décidez de rétrograder le VPC CNI.

Remplacez tous les nœuds lors de la transition vers la délégation de préfixes

Il est vivement recommandé de créer de nouveaux groupes de nœuds pour augmenter le nombre d'adresses IP disponibles plutôt que de remplacer progressivement les nœuds de travail existants. Bouclez et videz tous les nœuds existants pour expulser en toute sécurité tous vos pods existants. Pour éviter les interruptions de service, nous vous suggérons de mettre en place des budgets d'interruption de service sur vos clusters de production pour les charges de travail critiques. Les pods des nouveaux nœuds se verront attribuer une adresse IP à partir d'un préfixe attribué à un ENI. Après avoir confirmé que les pods sont en cours d'exécution, vous pouvez supprimer les anciens nœuds et groupes de nœuds. Si vous utilisez des groupes de nœuds gérés, veuillez suivre les étapes mentionnées ici pour supprimer un groupe de nœuds en toute sécurité.

Mode préfixe pour Windows

Dans Amazon EKS, chaque pod qui s'exécute sur un hôte Windows se voit attribuer une adresse IP secondaire par défaut par le contrôleur de ressources VPC. Cette adresse IP est une adresse routable par VPC allouée depuis le sous-réseau de l'hôte. Sous Linux, chaque ENI attachée à l'instance possède plusieurs emplacements qui peuvent être remplis par une adresse IP secondaire ou un /28 CIDR (un préfixe). Les hôtes Windows ne prennent toutefois en charge qu'un seul ENI et ses emplacements disponibles. L'utilisation uniquement d'adresses IP secondaires peut limiter artificiellement le nombre de modules que vous pouvez exécuter sur un hôte Windows, même lorsque de nombreuses adresses IP peuvent être attribuées.

Afin d'augmenter la densité des modules sur les hôtes Windows, en particulier lorsque vous utilisez des types d'instance plus petits, vous pouvez activer la délégation de préfixes pour les nœuds Windows. Lorsque la délégation de préfixes est activée, les IPv4 préfixes /28 sont attribués aux emplacements ENI plutôt qu'aux adresses IP secondaires. La délégation de préfixes peut être activée en ajoutant l'`enable-windows-prefix-delegation: "true"`entrée à la carte de `amazon-vpc-cni` configuration. Il s'agit de la même carte de configuration où vous devez définir une `enable-windows-ipam: "true"` entrée pour activer le support Windows.

Suivez les instructions mentionnées dans le guide de l'utilisateur d'EKS pour activer le mode de délégation de préfixes pour les nœuds Windows.

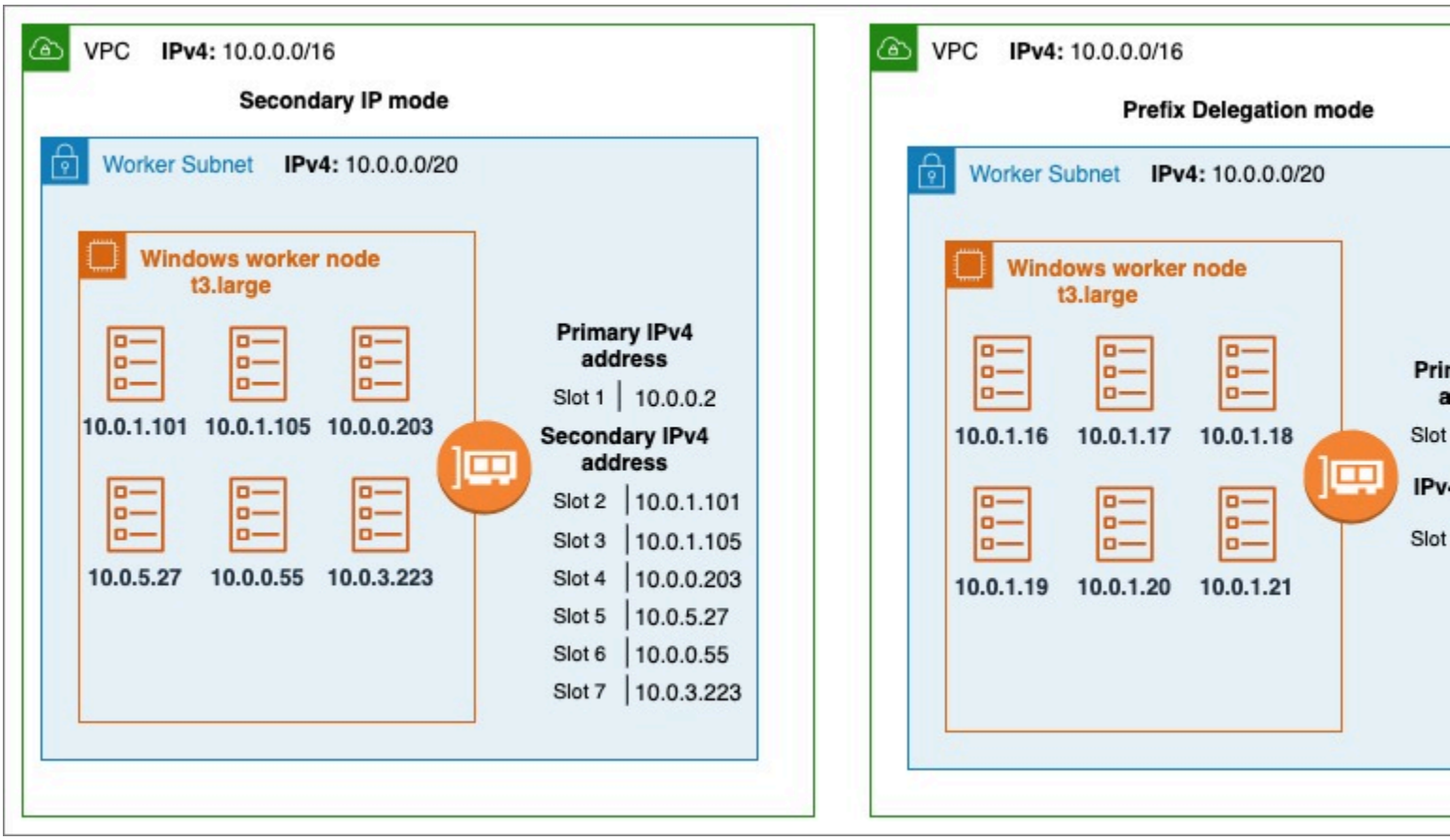


Figure : Comparaison du mode IP secondaire avec le mode de délégation de préfixes

Le nombre maximum d'adresses IP que vous pouvez attribuer à une interface réseau dépend du type d'instance et de sa taille. Chaque préfixe attribué à une interface réseau consomme un emplacement disponible. Par exemple, une `c5.large` instance a une limite de `10` slots par interface réseau. Le premier emplacement d'une interface réseau est toujours utilisé par l'adresse IP principale de l'interface, ce qui vous laisse 9 emplacements pour les préfixes et les adresses IP and/or secondaires. Si des préfixes sont attribués à ces emplacements, le nœud peut prendre en charge 144 adresses IP (9* 16) alors que si des adresses IP secondaires leur sont attribuées, il ne peut prendre en charge que 9 adresses IP. Consultez la documentation sur les adresses IP par interface réseau et par type d'instance et sur l'attribution de préfixes aux interfaces réseau pour plus d'informations.

Lors de l'initialisation du nœud de travail, le contrôleur de ressources VPC attribue un ou plusieurs préfixes à l'ENI principal pour accélérer le démarrage du pod en maintenant un pool chaud

d'adresses IP. Le nombre de préfixes à conserver dans le warm pool peut être contrôlé en définissant les paramètres de configuration suivants dans la carte de `amazon-vpc-cni` configuration.

- `warm-prefix-target`, le nombre de préfixes à allouer en sus des besoins actuels.
- `warm-ip-target`, le nombre d'adresses IP à attribuer en sus des besoins actuels.
- `minimum-ip-target`, le nombre minimum d'adresses IP pouvant être disponibles à tout moment.
- `warm-ip-target` and/or `minimum-ip-target`si défini, il `warm-prefix-target` remplacera.

Au fur et à mesure que de nouveaux pods sont prévus sur le nœud, des préfixes supplémentaires seront demandés pour l'ENI existant. Lorsqu'un pod est planifié sur le nœud, le contrôleur de ressources VPC essaie d'abord d'attribuer une IPv4 adresse à partir des préfixes existants sur le nœud. Si cela n'est pas possible, un nouveau IPv4 préfixe sera demandé tant que le sous-réseau dispose de la capacité requise.

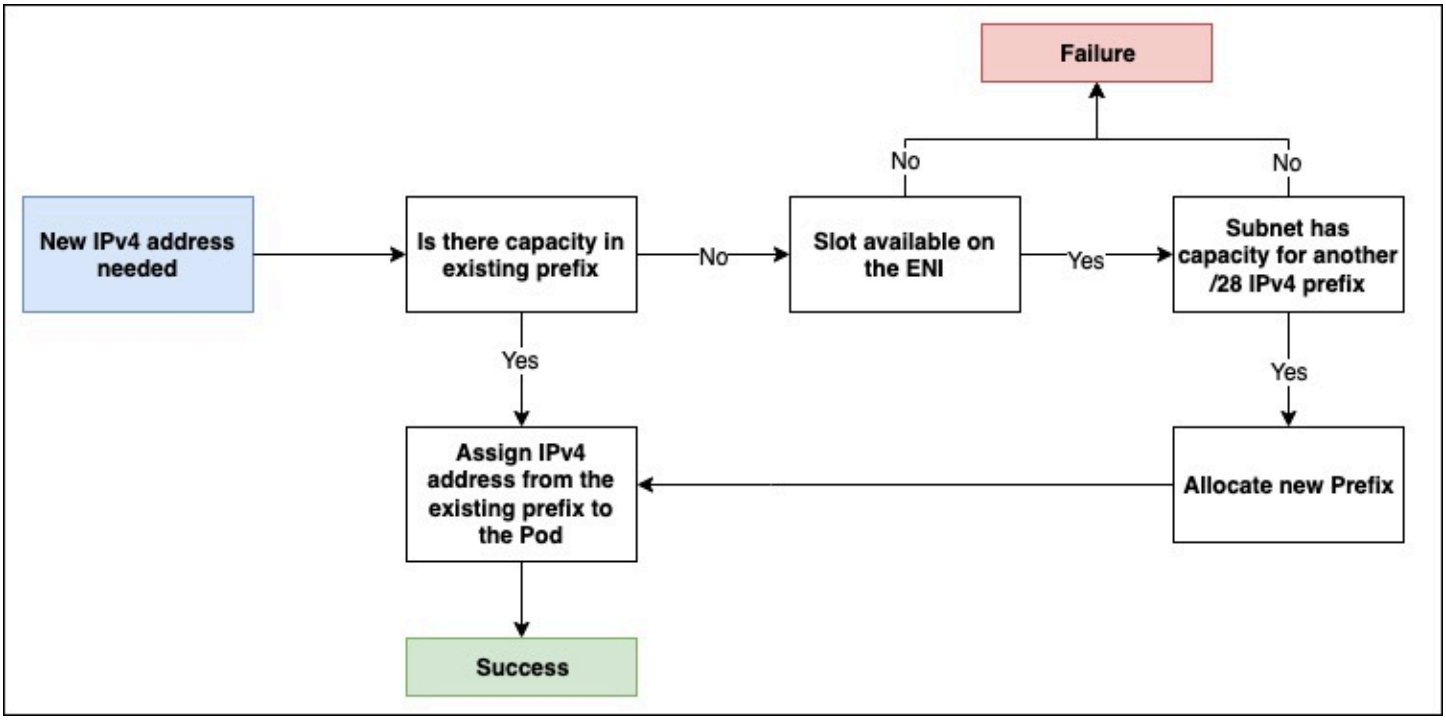


Figure : Flux de travail lors de l'attribution de l' IPv4 adresse au Pod

Recommandations

Utilisez la délégation de préfixes lorsque

Utilisez la délégation de préfixes si vous rencontrez des problèmes de densité de pods sur les nœuds de travail. Pour éviter les erreurs, nous vous recommandons d'examiner les sous-réseaux pour

détecter les blocs d'adresses contigus pour le préfixe /28 avant de passer en mode préfixe. Reportez-vous à la section « Utiliser les réservations de sous-réseaux pour éviter la fragmentation des sous-réseaux (IPv4) » pour plus de détails sur les réservations de sous-réseaux.

Par défaut, `max-pods` les nœuds sous Windows sont définis sur`110`. Pour la grande majorité des types d'instances, cela devrait être suffisant. Si vous souhaitez augmenter ou diminuer cette limite, ajoutez ce qui suit à la commande bootstrap dans vos données utilisateur :

```
-KubeletExtraArgs '--max-pods=example-value'
```

Pour plus de détails sur les paramètres de configuration du bootstrap pour les nœuds Windows, consultez la documentation ici.

Évitez la délégation de préfixes lorsque

Si votre sous-réseau est très fragmenté et ne possède pas suffisamment d'adresses IP disponibles pour créer des préfixes /28, évitez d'utiliser le mode préfixe. L'attachement du préfixe peut échouer si le sous-réseau à partir duquel le préfixe est produit est fragmenté (sous-réseau très utilisé avec des adresses IP secondaires dispersées). Ce problème peut être évité en créant un nouveau sous-réseau et en réservant un préfixe.

Configurer les paramètres de délégation de préfixes afin de conserver les adresses IPv4

`warm-prefix-targetwarm-ip-target`, et `minimum-ip-target` peut être utilisé pour affiner le comportement de la pré-mise à l'échelle et de la mise à l'échelle dynamique à l'aide de préfixes. Par défaut, les valeurs suivantes sont utilisées :

```
warm-ip-target: "1"
minimum-ip-target: "3"
```

En affinant ces paramètres de configuration, vous pouvez atteindre un équilibre optimal entre la conservation des adresses IP et la réduction de la latence du pod due à l'attribution de l'adresse IP. Pour plus d'informations sur ces paramètres de configuration, consultez la documentation ici.

Utiliser les réservations de sous-réseaux pour éviter la fragmentation des sous-réseaux () IPv4

Lorsqu'EC2 attribue un IPv4 préfixe /28 à une ENI, il doit s'agir d'un bloc contigu d'adresses IP provenant de votre sous-réseau. Si le sous-réseau à partir duquel le préfixe est généré est fragmenté

(sous-réseau très utilisé avec des adresses IP secondaires dispersées), l'attachement du préfixe peut échouer et vous verrez l'événement de nœud suivant :

```
InsufficientCidrBlocks: The specified subnet does not have enough free cidr blocks to
 satisfy the request
```

Pour éviter la fragmentation et disposer d'un espace contigu suffisant pour créer des préfixes, utilisez les réservations CIDR du sous-réseau VPC pour réserver de l'espace IP au sein d'un sous-réseau à l'usage exclusif des préfixes. Une fois que vous avez créé une réservation, les adresses IP des blocs réservés ne seront pas attribuées à d'autres ressources. De cette façon, le contrôleur de ressources VPC pourra obtenir les préfixes disponibles lors de l'appel d'assignation au nœud ENI.

Il est recommandé de créer un nouveau sous-réseau, de réserver de l'espace pour les préfixes et d'activer l'attribution de préfixes pour les nœuds de travail exécutés dans ce sous-réseau. Si le nouveau sous-réseau est dédié uniquement aux pods exécutés dans votre cluster EKS avec la délégation de préfixes activée, vous pouvez ignorer l'étape de réservation du préfixe.

Remplacez tous les nœuds lors de la migration du mode IP secondaire vers le mode de délégation de préfixes ou vice versa

Il est vivement recommandé de créer de nouveaux groupes de nœuds pour augmenter le nombre d'adresses IP disponibles plutôt que de remplacer progressivement les nœuds de travail existants.

Lorsque vous utilisez des groupes de nœuds autogérés, les étapes de transition sont les suivantes :

- Augmentez la capacité de votre cluster afin que les nouveaux nœuds soient en mesure d'accueillir vos charges de travail
- Activer/désactiver la fonctionnalité de délégation de préfixes pour Windows
- Bouclez et videz tous les nœuds existants pour expulser en toute sécurité tous vos pods existants. Pour éviter les interruptions de service, nous vous suggérons de mettre en place des budgets d'interruption de service sur vos clusters de production pour les charges de travail critiques.
- Après avoir confirmé que les pods sont en cours d'exécution, vous pouvez supprimer les anciens nœuds et groupes de nœuds. Les pods des nouveaux nœuds se verront attribuer une IPv4 adresse à partir d'un préfixe attribué au nœud ENI.

Lorsque vous utilisez des groupes de nœuds gérés, les étapes de transition sont les suivantes :

- Activer/désactiver la fonctionnalité de délégation de préfixes pour Windows

- Mettez à jour le groupe de nœuds en suivant les étapes décrites ici. Cela exécute des étapes similaires à celles décrites ci-dessus, mais elles sont gérées par EKS.

> **Warning**
>
> Exécutez tous les Pods sur un nœud dans le même mode

Pour Windows, nous vous recommandons d'éviter d'exécuter des pods à la fois en mode IP secondaire et en mode délégation de préfixes. Une telle situation peut se produire lorsque vous passez du mode IP secondaire au mode délégation de préfixes ou vice versa lorsque vous exécutez des charges de travail Windows.

Bien que cela n'ait aucun impact sur vos pods en cours d'exécution, il peut y avoir des incohérences en ce qui concerne la capacité d'adresse IP du nœud. Par exemple, considérez un nœud t3.xlarge qui possède 14 emplacements pour les adresses secondaires. IPv4 Si vous utilisez 10 pods, 10 emplacements de l'ENI seront utilisés par des adresses IP secondaires. Une fois que vous avez activé la délégation de préfixes, la capacité annoncée au serveur kube-api serait (14 emplacements x 16 adresses IP par préfixe) 244, mais la capacité réelle à ce moment-là serait de (4 emplacements restants * 16 adresses par préfixe) 64. Cette incohérence entre la capacité annoncée et la capacité réelle (emplacements restants) peut entraîner des problèmes si vous utilisez plus de pods qu'il n'y a d'adresses IP disponibles pour l'attribution.

Cela étant dit, vous pouvez utiliser la stratégie de migration décrite ci-dessus pour faire passer en toute sécurité vos pods d'une adresse IP secondaire à des adresses obtenues à partir de préfixes. Lorsque vous passez d'un mode à l'autre, les Pods continueront de fonctionner normalement et :

- Lorsque vous passez du mode IP secondaire au mode délégation de préfixes, les adresses IP secondaires attribuées aux pods en cours d'exécution ne seront pas publiées. Des préfixes seront attribués aux machines à sous gratuites. Une fois qu'un pod est fermé, l'adresse IP secondaire et le slot qu'il utilisait sont libérés.
- Lorsque vous passez du mode de délégation de préfixes au mode IP secondaire, un préfixe est libéré lorsque tous les éléments à IPs portée ne sont plus alloués aux pods. Si une adresse IP provenant du préfixe est attribuée à un pod, ce préfixe sera conservé jusqu'à ce que les pods soient résiliés.

Problèmes de débogage liés à la délégation de préfixes

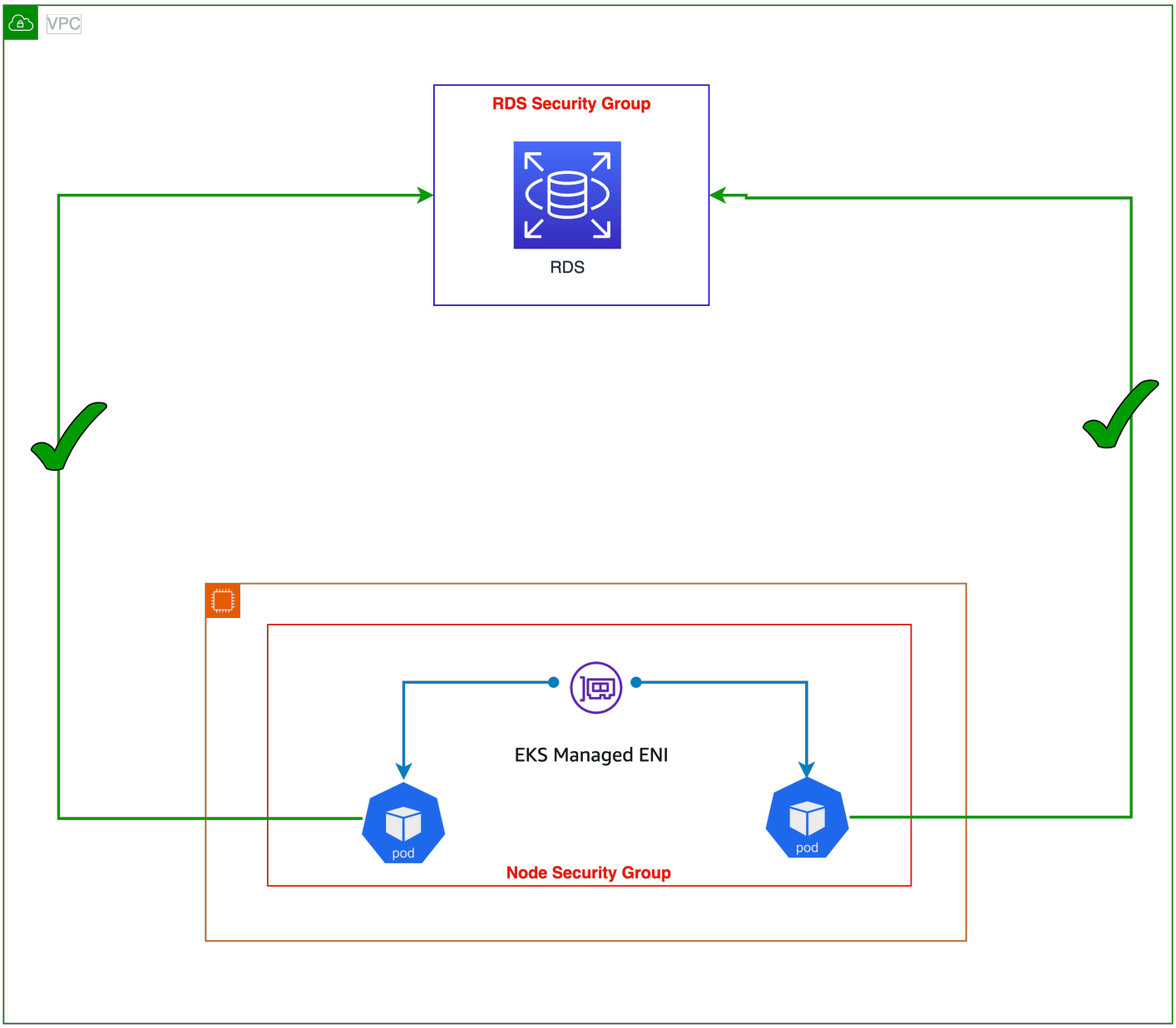
Vous pouvez utiliser notre guide de débogage ici pour approfondir le problème que vous rencontrez avec la délégation de préfixes sous Windows.

Groupes de sécurité par pod

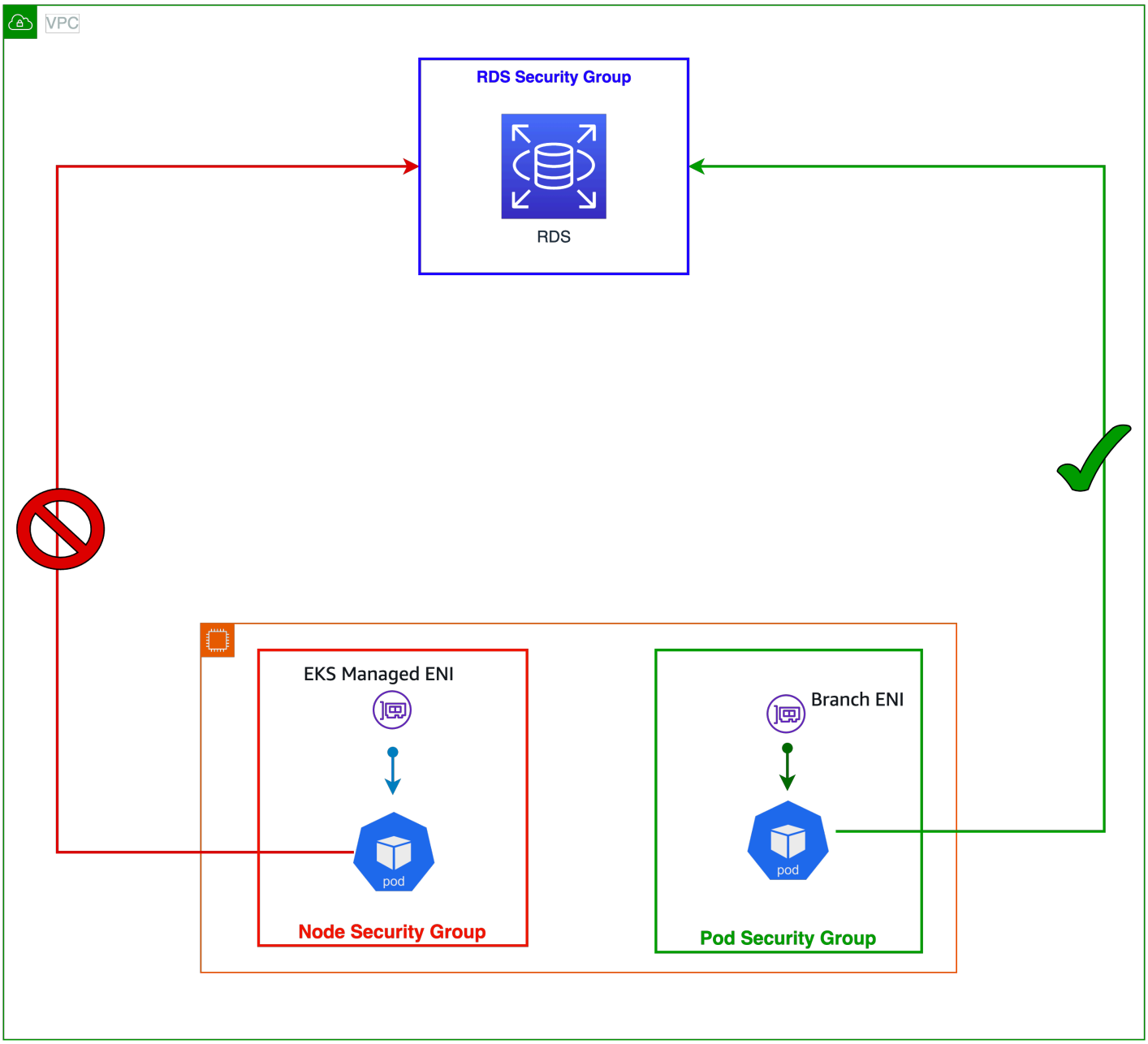
> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Un groupe de sécurité AWS agit comme un pare-feu virtuel pour les instances EC2 afin de contrôler le trafic entrant et sortant. Par défaut, le CNI Amazon VPC utilisera des groupes de sécurité associés à l'ENI principal sur le nœud. Plus précisément, chaque ENI associée à l'instance aura les mêmes groupes de sécurité EC2. Ainsi, chaque pod d'un nœud partage les mêmes groupes de sécurité que le nœud sur lequel il s'exécute.

Comme le montre l'image ci-dessous, tous les pods d'applications fonctionnant sur des nœuds de travail auront accès au service de base de données RDS (si l'on considère que le RDS entrant autorise le groupe de sécurité des nœuds). Les groupes de sécurité sont trop grossiers car ils s'appliquent à tous les pods exécutés sur un nœud. Les groupes de sécurité pour Pods permettent de segmenter le réseau pour les charges de travail, élément essentiel d'une bonne stratégie de défense en profondeur.



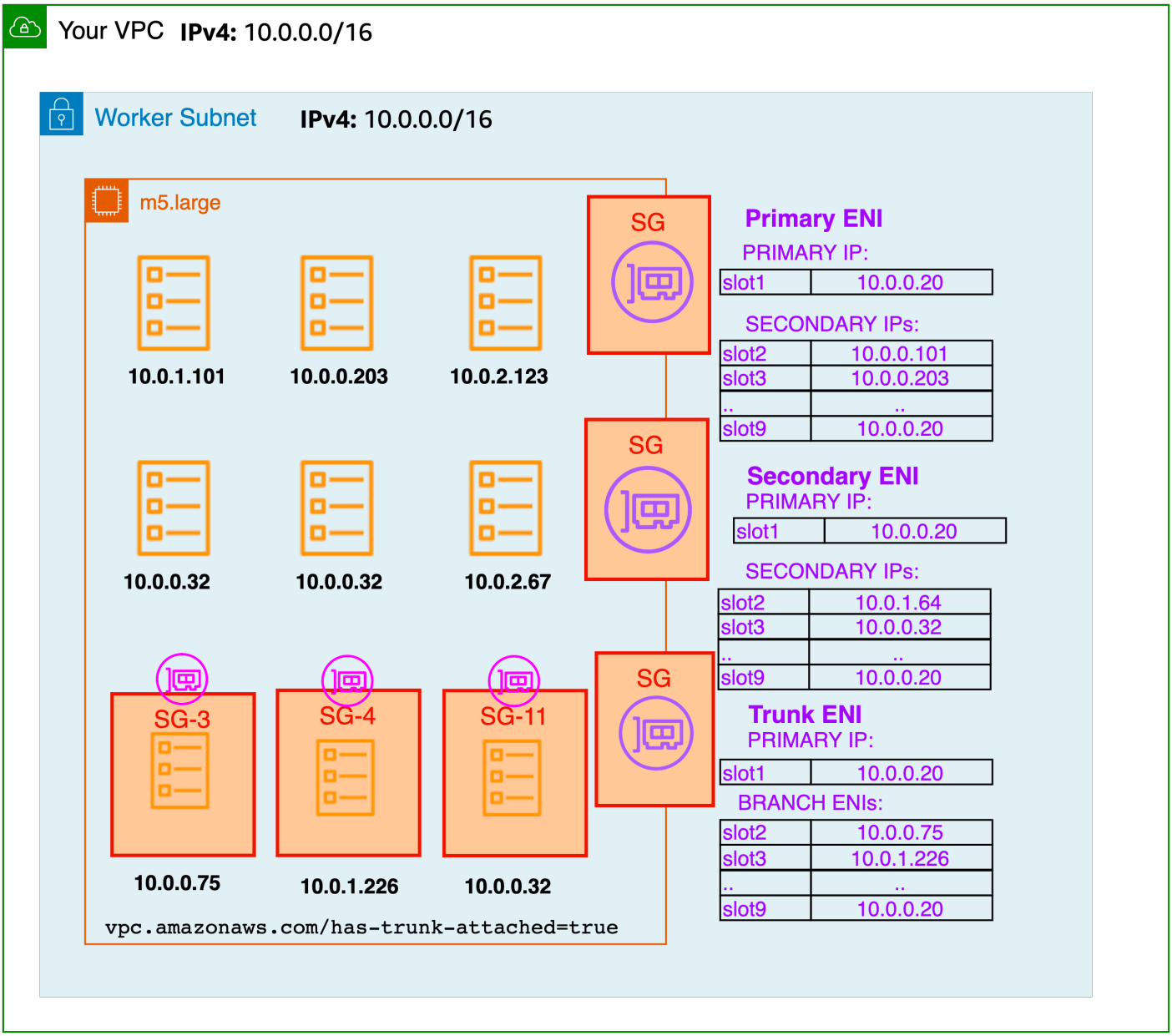
Avec les groupes de sécurité pour Pods, vous pouvez améliorer l'efficacité du calcul en exécutant des applications répondant à des exigences de sécurité réseau variables sur des ressources informatiques partagées. Plusieurs types de règles de sécurité, tels que Pod-to-Pod les services Pod-to-External AWS, peuvent être définis en un seul endroit avec les groupes de sécurité EC2 et appliqués aux charges de travail avec Kubernetes native. APIs L'image ci-dessous montre les groupes de sécurité appliqués au niveau du Pod et montre comment ils simplifient le déploiement de vos applications et l'architecture de vos nœuds. Le Pod peut désormais accéder à la base de données Amazon RDS.



Vous pouvez activer les groupes de sécurité pour les pods en configurant `ENABLE_POD_ENI=true` le VPC CNI. Une fois activé, le contrôleur de ressources VPC exécuté sur le plan de contrôle (géré par EKS) crée et attache une interface de liaison appelée « aws-k8 s-trunk-eni » au nœud. L'interface de jonction agit comme une interface réseau standard attachée à l'instance. Pour gérer les interfaces de liaison, vous devez ajouter la politique `AmazonEKSVPCResourceController` gérée au rôle de cluster associé à votre cluster Amazon EKS.

Le contrôleur crée également des interfaces de branche nommées « aws-k8 s-branch-eni » et les associe à l'interface de jonction. Un groupe de sécurité est attribué aux pods à l'aide de la ressource

SecurityGroupPolicypersonnalisée et sont associés à une interface de branche. Les groupes de sécurité étant spécifiés à l'aide d'interfaces réseau, nous sommes désormais en mesure de planifier des pods nécessitant des groupes de sécurité spécifiques sur ces interfaces réseau supplémentaires. Consultez la section du guide de l'utilisateur EKS sur les groupes de sécurité pour les pods, y compris les conditions préalables au déploiement.



La capacité de l'interface des succursales s'ajoute aux limites de types d'instances existantes pour les adresses IP secondaires. Les pods qui utilisent des groupes de sécurité ne sont pas pris en compte dans la formule max-pods et lorsque vous utilisez un groupe de sécurité pour des pods, vous

devez envisager d'augmenter la valeur max-pods ou accepter d'exécuter moins de pods que le nœud ne peut réellement en supporter.

Un m5.large peut avoir jusqu'à 9 interfaces réseau de succursales et jusqu'à 27 adresses IP secondaires attribuées à ses interfaces réseau standard. Comme le montre l'exemple ci-dessous, le nombre maximum de pods par défaut pour un m5.large est de 29, et EKS compte les pods qui utilisent des groupes de sécurité dans le calcul du nombre maximum de pods. Consultez le guide de l'utilisateur d'EKS pour savoir comment modifier les max-pods pour les nœuds.

Lorsque des groupes de sécurité pour Pods sont utilisés en combinaison avec un réseau personnalisé, le groupe de sécurité défini dans les groupes de sécurité pour Pods est utilisé plutôt que le groupe de sécurité spécifié dans le ENIConfig. Par conséquent, lorsque la mise en réseau personnalisée est activée, évaluez soigneusement l'ordre des groupes de sécurité lors de l'utilisation de groupes de sécurité par pod.

Recommandations

Désactiver le démultiplexage précoce TCP pour Liveness Probe

Si vous utilisez des sondes Liveness ou Readiness, vous devez également désactiver le démultiplexage précoce TCP, afin que le kubelet puisse se connecter aux pods sur les interfaces réseau des succursales via TCP. Cela n'est requis qu'en mode strict. Pour ce faire, exécutez la commande suivante :

```
kubectl edit daemonset aws-node -n kube-system
```

Dans la `initContainer` section, modifiez la valeur `DISABLE_TCP_EARLY_DEMUX` de `true`.

Utilisez Security Group For Pods pour tirer parti des investissements existants dans la configuration AWS.

Les groupes de sécurité facilitent la restriction de l'accès réseau aux ressources VPC, telles que les bases de données RDS ou les instances EC2. L'un des avantages évidents des groupes de sécurité par pod est la possibilité de réutiliser les ressources des groupes de sécurité AWS existantes. Si vous utilisez des groupes de sécurité comme pare-feu réseau pour limiter l'accès à vos services AWS, nous vous proposons d'appliquer des groupes de sécurité aux pods à l'aide des ENI de branche. Envisagez d'utiliser des groupes de sécurité pour les pods si vous transférez des applications d'instances EC2 vers EKS et si vous limitez l'accès à d'autres services AWS dotés de groupes de sécurité.

Configurer le mode d'application du groupe de sécurité Pod

La version 1.11 du plugin Amazon VPC CNI a ajouté un nouveau paramètre nommé `POD_SECURITY_GROUP_ENFORCING_MODE` (« mode d'application »). Le mode d'application contrôle à la fois quels groupes de sécurité s'appliquent au pod et si le NAT source est activé. Vous pouvez définir le mode d'application comme strict ou standard. Strict est la valeur par défaut, reflétant le comportement précédent du CNI VPC défini sur. `ENABLE_POD_ENI` `true`

En mode strict, seuls les groupes de sécurité ENI des succursales sont appliqués. Le NAT source est également désactivé.

En mode standard, les groupes de sécurité associés à la fois à l'ENI principal et à l'ENI de branche (associé au pod) sont appliqués. Le trafic réseau doit respecter les deux groupes de sécurité.

> **Warning**
>
> Tout changement de mode n'aura d'impact que sur les pods récemment lancés. Les pods existants utiliseront le mode configuré lors de la création du pod. Les clients devront recycler les pods existants avec des groupes de sécurité s'ils souhaitent modifier le comportement du trafic.

Mode d'application : utilisez le mode strict pour isoler le trafic des pods et des nœuds :

Par défaut, les groupes de sécurité pour Pods sont définis sur le « mode strict ». Utilisez ce paramètre si vous devez complètement séparer le trafic du pod du reste du trafic du nœud. En mode strict, le NAT source est désactivé afin que les groupes de sécurité sortants ENI de la branche puissent être utilisés.

> **Warning**
>
> Lorsque le mode strict est activé, tout le trafic sortant d'un pod quitte le nœud et entre dans le réseau VPC. Le trafic entre les pods d'un même nœud passera par le VPC. Cela augmente le trafic VPC et limite les fonctionnalités basées sur les nœuds. Le n' NodeLocal DNSCache est pas pris en charge avec le mode strict.

Mode d'application : utilisez le mode standard dans les situations suivantes

IP source du client visible par les conteneurs du Pod

Si vous devez que l'adresse IP source du client reste visible pour les conteneurs du Pod, pensez `POD_SECURITY_GROUP_ENFORCING_MODE` à la définir sur`standard`. Les services Kubernetes prennent en charge externalTrafficPolicy =local pour permettre la préservation de l'adresse IP source du client (cluster de type par défaut). Vous pouvez désormais exécuter des services Kubernetes de type NodePort et LoadBalancer utiliser des cibles d'instance externalTrafficPolicy définies sur Local en mode standard. `Local`préserve l'adresse IP source du client et évite un second saut pour LoadBalancer et NodePort type Services.

Déploiement NodeLocal DNSCache

Lorsque vous utilisez des groupes de sécurité pour les pods, configurez le mode standard pour prendre en charge les pods qui les utilisent NodeLocal DNSCache. NodeLocal DNSCache améliore les performances du DNS du cluster en exécutant un agent de mise en cache DNS sur les nœuds du cluster en tant DaemonSet que. Cela aidera les pods ayant les exigences les plus élevées en matière de QPS DNS à interroger les Kube-DNS/CoreDNS locaux ayant un cache local, ce qui améliorera la latence.

NodeLocal DNSCache n'est pas pris en charge en mode strict car tout le trafic réseau, même à destination du nœud, entre dans le VPC.

Soutenir la politique réseau Kubernetes

Nous recommandons d'utiliser le mode d'application standard lorsque vous utilisez la politique réseau avec des pods auxquels des groupes de sécurité sont associés.

Nous vous recommandons vivement d'utiliser des groupes de sécurité pour les pods afin de limiter l'accès au niveau du réseau aux services AWS qui ne font pas partie d'un cluster. Envisagez des politiques réseau pour restreindre le trafic réseau entre les pods au sein d'un cluster, souvent appelé East/West trafic.

Identifier les incompatibilités avec les groupes de sécurité par pod

Les instances basées sur Windows et autres que Nitro ne prennent pas en charge les groupes de sécurité pour les pods. Pour utiliser des groupes de sécurité avec des pods, les instances doivent être étiquetées avec isTrunkingEnabled. Utilisez des politiques réseau pour gérer l'accès entre les

pods plutôt que les groupes de sécurité si vos pods ne dépendent d'aucun service AWS au sein ou en dehors de votre VPC.

Utilisez des groupes de sécurité par pod pour contrôler efficacement le trafic vers les services AWS

Si une application exécutée dans le cluster EKS doit communiquer avec une autre ressource du VPC, par exemple une base de données RDS, envisagez d'utiliser SGs for pods. Bien que certains moteurs de politiques vous permettent de spécifier un CIDR ou un nom DNS, ils constituent un choix moins optimal lorsque vous communiquez avec des services AWS dont les points de terminaison se trouvent au sein d'un VPC.

En revanche, les politiques réseau de Kubernetes fournissent un mécanisme permettant de contrôler le trafic entrant et sortant à la fois à l'intérieur et à l'extérieur du cluster. Les politiques réseau Kubernetes doivent être prises en compte si les dépendances de votre application vis-à-vis d'autres services AWS sont limitées. Vous pouvez configurer des politiques réseau qui spécifient des règles de sortie basées sur des plages d'adresses CIDR afin de limiter l'accès aux services AWS, par opposition à la sémantique native d'AWS telle que. SGs Vous pouvez utiliser les politiques réseau de Kubernetes pour contrôler le trafic réseau entre les pods (souvent appelé East/West trafic) et entre les pods et les services externes. Les politiques réseau Kubernetes sont mises en œuvre aux niveaux OSI 3 et 4.

Amazon EKS vous permet d'utiliser des moteurs de politique réseau tels que Calico et Cilium. Par défaut, les moteurs de politique réseau ne sont pas installés. Veuillez consulter les guides d'installation respectifs pour obtenir des instructions de configuration. Pour plus d'informations sur l'utilisation de la politique réseau, consultez les meilleures pratiques de sécurité d'EKS. La fonctionnalité des noms d'hôte DNS est disponible dans les versions d'entreprise des moteurs de politique réseau, ce qui peut être utile pour contrôler le trafic entre Kubernetes Services/Pods et les ressources exécutées en dehors d'AWS. Vous pouvez également envisager de prendre en charge les noms d'hôte DNS pour les services AWS qui ne prennent pas en charge les groupes de sécurité par défaut.

Marquez un seul groupe de sécurité pour utiliser AWS Loadbalancer Controller

Lorsque de nombreux groupes de sécurité sont alloués à un pod, Amazon EKS recommande de baliser un seul groupe de sécurité avec la mention « `kubernetes.io/cluster/$name`partagé » ou « possédé ». La balise permet au contrôleur AWS Loadbalancer de mettre à jour les règles des groupes de sécurité afin d'acheminer le trafic vers les Pods. Si un seul groupe de sécurité est attribué

à un Pod, l'attribution d'une balise est facultative. Les autorisations définies dans un groupe de sécurité sont additives. Il suffit donc de baliser un seul groupe de sécurité pour que le contrôleur d'équilibrage de charge localise et réconcilie les règles. Cela permet également de respecter les quotas par défaut définis par les groupes de sécurité.

Configurer le NAT pour le trafic sortant

Le NAT source est désactivé pour le trafic sortant provenant des pods auxquels des groupes de sécurité ont été affectés. Pour les pods utilisant des groupes de sécurité qui nécessitent un accès à Internet, lancez des nœuds de travail sur des sous-réseaux privés configurés avec une passerelle ou une instance NAT et activez le SNAT externe dans le CNI.

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

Déployer des pods avec des groupes de sécurité sur des sous-réseaux privés

Les pods auxquels des groupes de sécurité sont affectés doivent être exécutés sur des nœuds déployés sur des sous-réseaux privés. Notez que les pods dotés de groupes de sécurité assignés déployés sur des sous-réseaux publics ne pourront pas accéder à Internet.

Vérifier les terminationGracePeriodsecondes dans le fichier de spécifications du pod

Assurez-vous que cette valeur `terminationGracePeriodSeconds` est différente de zéro dans le fichier de spécifications de votre Pod (30 secondes par défaut). Cela est essentiel pour qu'Amazon VPC CNI puisse supprimer le réseau Pod du nœud de travail. Lorsqu'il est réglé sur zéro, le plug-in CNI ne supprime pas le réseau Pod de l'hôte et la branche ENI n'est pas nettoyée efficacement.

Utilisation de groupes de sécurité pour les pods avec Fargate

Les groupes de sécurité pour les pods qui s'exécutent sur Fargate fonctionnent de manière très similaire aux pods qui s'exécutent sur des nœuds de travail EC2. Par exemple, vous devez créer le groupe de sécurité avant de le référencer dans celui que SecurityGroupPolicy vous associez à votre Fargate Pod. Par défaut, le groupe de sécurité du cluster est attribué à tous les Fargate Pods lorsque vous n'en attribuez pas explicitement un SecurityGroupPolicy à un Fargate Pod. Pour des raisons de simplicité, vous pouvez ajouter le groupe de sécurité du cluster à celui d'un Fagate Pod SecurityGroupPolicy , sinon vous devrez ajouter les règles de groupe de sécurité minimales à votre groupe de sécurité. Vous pouvez trouver le groupe de sécurité du cluster à l'aide de l'API describe-cluster.

```
 aws eks describe-cluster --name CLUSTER_NAME --query
 'cluster.resourcesVpcConfig.clusterSecurityGroupId'
```

```
cat >my-fargate-sg-policy.yaml <<EOF
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
  name: my-fargate-sg-policy
  namespace: my-fargate-namespace
spec:
  podSelector:
    matchLabels:
      role: my-fargate-role
  securityGroups:
    groupIds:
      - cluster_security_group_id
      - my_fargate_pod_security_group_id
EOF
```

Les règles minimales relatives aux groupes de sécurité sont répertoriées ici. Ces règles permettent aux Fargate Pods de communiquer avec des services intégrés au cluster tels que kube-apiserver, kubelet et CoreDNS. Vous devez également ajouter des règles pour autoriser les connexions entrantes et sortantes à destination et en provenance de votre Fargate Pod. Cela permettra à votre pod de communiquer avec d'autres pods ou ressources de votre VPC. En outre, vous devez inclure des règles permettant à Fargate d'extraire les images de conteneurs d'Amazon ECR ou d'autres registres de conteneurs tels que. DockerHub Pour plus d'informations, consultez les plages d'adresses IP AWS dans le manuel de référence général AWS.

Vous pouvez utiliser les commandes ci-dessous pour trouver les groupes de sécurité appliqués à un Fargate Pod.

```
kubectl get pod FARGATE_POD -o jsonpath='{.metadata.annotations.fargate\.amazonaws
\.com/pod-sg}{"\n"}'
```

Notez la commande eNIid ci-dessus.

```
aws ec2 describe-network-interfaces --network-interface-ids ENI_ID --query
 'NetworkInterfaces[*].Groups[*]'
```

Les pods Fargate existants doivent être supprimés et recréés pour que de nouveaux groupes de sécurité puissent être appliqués. Par exemple, la commande suivante lance le déploiement de l'application d'exemple. Pour mettre à jour des pods spécifiques, vous pouvez modifier l'espace de noms et le nom du déploiement dans la commande ci-dessous.

```
kubectl rollout restart -n example-ns deployment example-pod
```

Équilibrage de charge

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Les équilibreurs de charge reçoivent le trafic entrant et le distribuent entre les cibles de l'application prévue hébergée dans un cluster EKS. Cela améliore la résilience de l'application. Lorsqu'il est déployé dans un cluster EKS, le contrôleur AWS Load Balancer crée et gère les AWS Elastic Load Balancers pour ce cluster. Lorsqu'un service de type Kubernetes LoadBalancer est créé, le contrôleur AWS Load Balancer crée un Network Load Balancer (NLB) qui équilibre la charge du trafic reçu au niveau de la couche 4 du modèle OSI. Lors de la création d'un objet Kubernetes Ingress, le AWS Load Balancer Controller crée un Application Load Balancer (ALB) qui équilibre la charge du trafic au niveau de la couche 7 du modèle OSI.

Choix du type de Load Balancer

Le portefeuille AWS Elastic Load Balancing (ELB) prend en charge les équilibreurs de charge suivants : les équilibreurs de charge d'application (ALB), les équilibreurs de charge réseau (NLB), les équilibreurs de charge de passerelle (GWLB) et les équilibreurs de charge classiques (CLB). Cette section sur les meilleures pratiques se concentrera sur l'ALB et le NLB, qui sont les deux plus pertinents pour les clusters EKS.

La principale considération lors du choix du type d'équilibreur de charge est la charge de travail requise.

Pour des informations plus détaillées et à titre de référence pour tous les équilibreurs de charge AWS, consultez Comparaisons de produits

Choisissez l'Application Load Balancer (ALB) si votre charge de travail est HTTP/HTTPS

Si une charge de travail nécessite un équilibrage de charge au niveau de la couche 7 du modèle OSI, le AWS Load Balancer Controller peut être utilisé pour provisionner un ALB ; nous aborderons le provisionnement dans la section suivante. L'ALB est contrôlé et configuré par la ressource Ingress mentionnée précédemment et achemine le trafic HTTP ou HTTPS vers différents pods au sein du cluster. L'ALB offre aux clients la flexibilité de modifier l'algorithme de routage du trafic de l'application ; l'algorithme de routage par défaut est circulaire, l'algorithme de routage des demandes les moins en suspens constituant également une alternative.

Choisissez le Network Load Balancer (NLB) si votre charge de travail est TCP ou si votre charge de travail nécessite la préservation de l'adresse IP source des clients

Un Network Load Balancer fonctionne au niveau de la quatrième couche (Transport) du modèle d'interconnexion des systèmes ouverts (OSI). Il est adapté aux charges de travail basées sur TCP et UDP. Network Load Balancer préserve également par défaut l'adresse IP source des clients lors de la présentation du trafic au pod.

Choisissez le Network Load Balancer (NLB) si votre charge de travail ne peut pas utiliser le DNS

Une autre raison essentielle d'utiliser le NLB est si vos clients ne peuvent pas utiliser le DNS. Dans ce cas, le NLB est peut-être mieux adapté à votre charge de travail, car les IPs composants d'un Network Load Balancer sont statiques. Bien qu'il soit recommandé aux clients d'utiliser le DNS pour résoudre des noms de domaine en adresses IP lorsqu'ils se connectent à des équilibreurs de charge, si l'application d'un client ne prend pas en charge la résolution DNS et n'accepte que le codage en dur IPs , un NLB est mieux adapté car il est statique et reste le IPs même pendant toute la durée de vie du NLB.

Provisionnement d'équilibreurs de charge

Après avoir déterminé le Load Balancer le mieux adapté à vos charges de travail, les clients disposent d'un certain nombre d'options pour le provisionner.

Provisionnez des équilibreurs de charge en déployant le contrôleur AWS Load Balancer

Il existe deux méthodes principales pour provisionner des équilibreurs de charge au sein d'un cluster EKS.

- Tirer parti du contrôleur de service dans le fournisseur de cloud AWS (ancien)
- Tirer parti du contrôleur AWS Load Balancer (recommandé)

Par défaut, le Kubernetes Service Controller, également connu sous le nom de contrôleur intégré à l'arborescence, réconcilie le type de ressource Kubernetes Service. LoadBalancer Ce contrôleur est intégré au composant AWS Cloud Provider qui fonctionne en tant que Kubernetes Cloud Controller Manager.

La configuration de l'Elastic Load Balancer provisionné est contrôlée par des annotations qui doivent être ajoutées au manifeste du service Kubernetes. Les annotations utilisées par le Service Controller et le AWS Load Balancer Controller sont différentes.

Le Service Controller est obsolète et ne reçoit actuellement que des corrections de bogues critiques. Lorsque vous créez un service Kubernetes de type Kubernetes LoadBalancer, le Service Controller crée un AWS CLB par défaut, mais peut également créer AWS NLB si vous utilisez la bonne annotation. Il convient de noter que Service Controller ne prend pas en charge les ressources Kubernetes Ingress et qu'il ne le prend pas non plus en charge. IPv6

Nous vous recommandons d'utiliser le contrôleur AWS Load Balancer dans vos clusters EKS pour réconcilier le service Kubernetes et les ressources d'entrée. Vous devez utiliser les bonnes annotations dans votre service Kubernetes ou dans votre manifeste d'entrée afin qu'AWS Load Balancer Controller soit responsable du processus de réconciliation. (au lieu du contrôleur de service)

Si vous utilisez le mode automatique EKS, le contrôleur AWS Load Balancer vous est fourni automatiquement ; aucune installation n'est nécessaire.

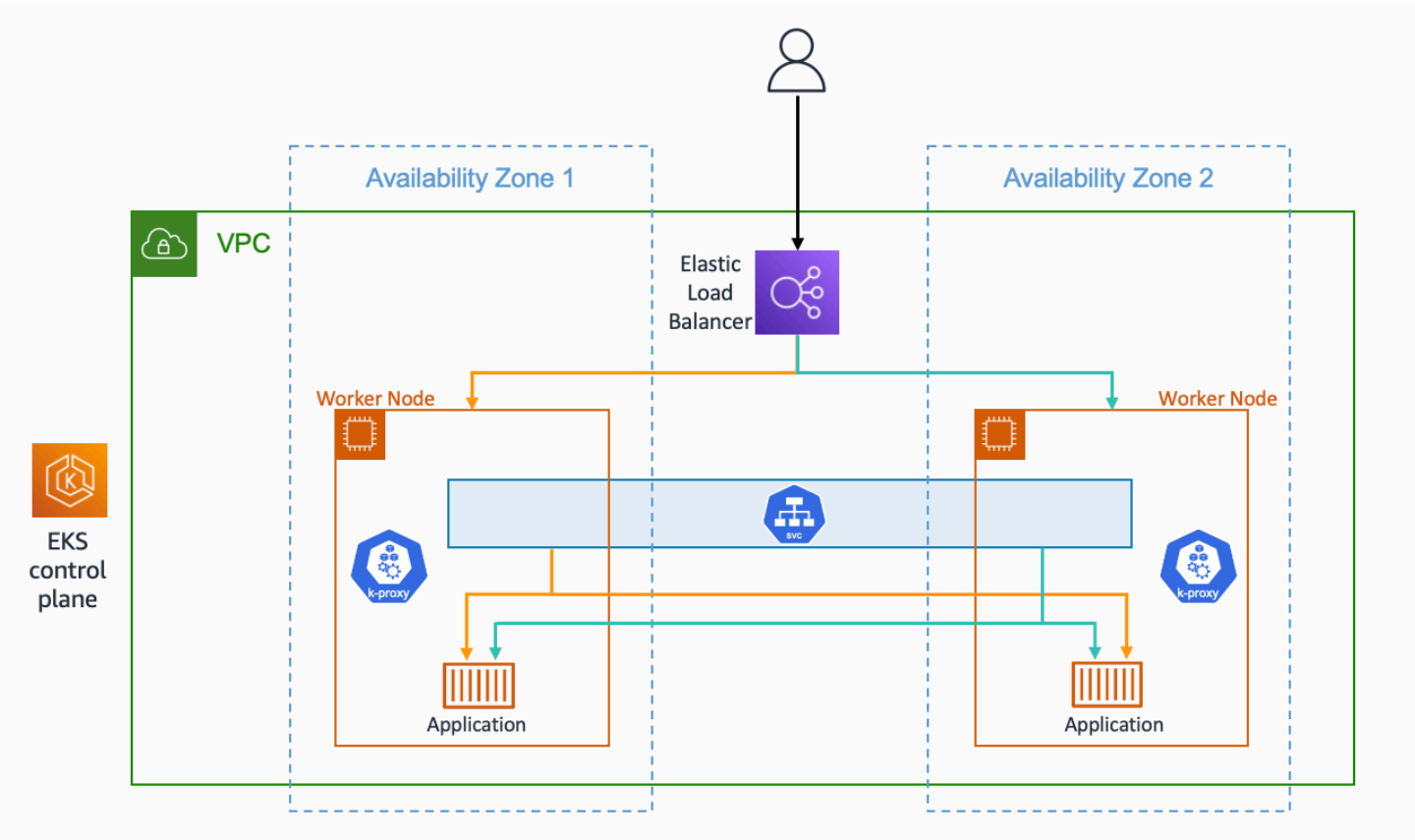
Choisir le type de cible du Load Balancer

Enregistrez les pods en tant que cibles à l'aide du type de cible IP

Un AWS Elastic Load Balancer : Network & Application envoie le trafic reçu aux cibles enregistrées dans un groupe cible. Pour un cluster EKS, il existe deux types de cibles que vous pouvez enregistrer

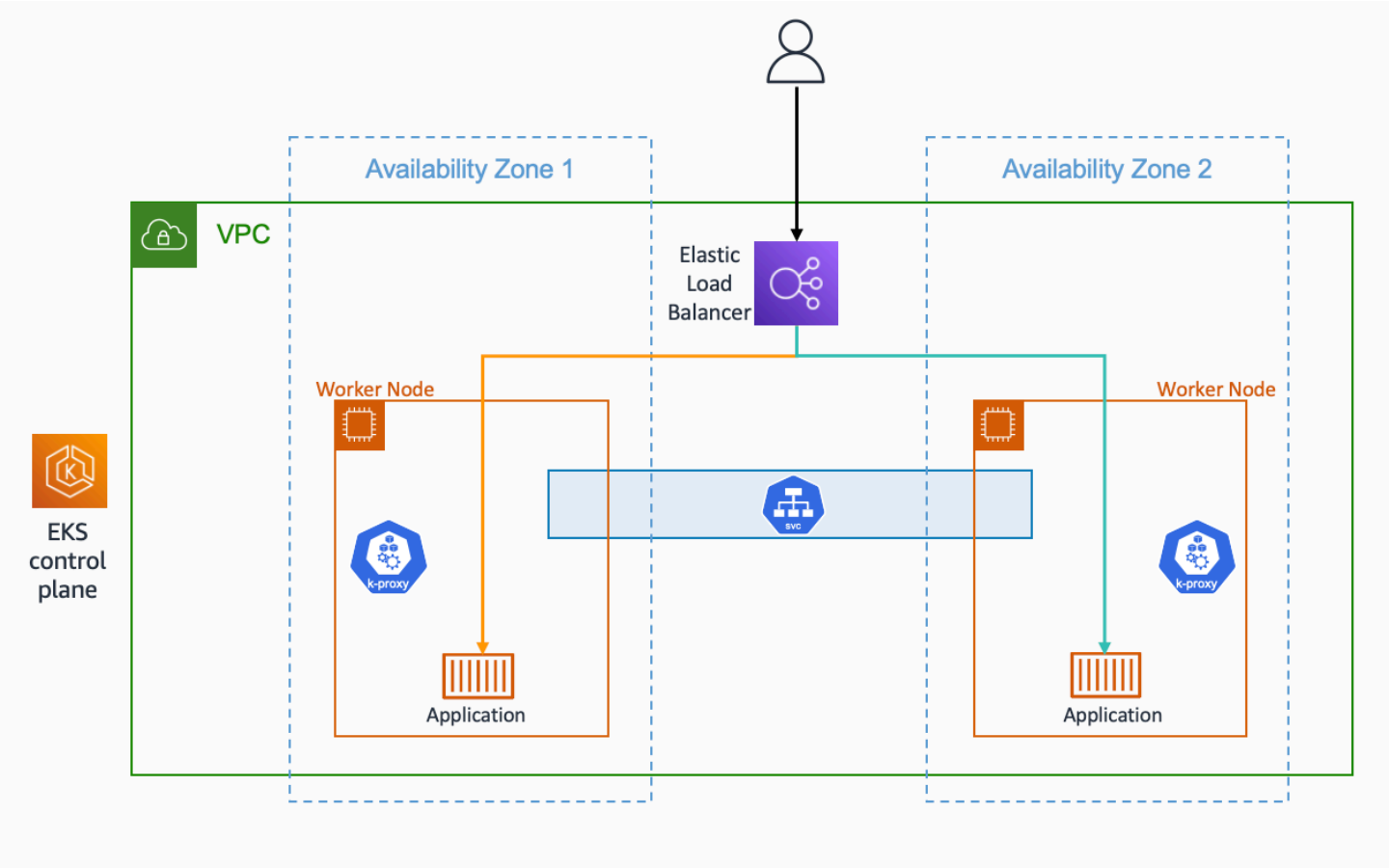
dans le groupe cible : Instance et IP. Le type de cible utilisé a une incidence sur les éléments enregistrés et sur la manière dont le trafic est acheminé du Load Balancer vers le pod. Par défaut, le contrôleur AWS Load Balancer enregistrera les cibles en utilisant le type « Instance ». Cette cible sera l'adresse IP du nœud de travailNodePort, ce qui implique notamment :

- Le trafic provenant du Load Balancer sera transféré vers le Worker Node sur le NodePort, il est traité selon les règles iptables (configurées par kube-proxy exécuté sur le nœud), puis transféré au service sur son ClusterIP (toujours sur le nœud). Enfin, le service sélectionne au hasard un pod enregistré et lui transmet le trafic. Ce flux implique plusieurs sauts et une latence supplémentaire peut être encourue, en particulier parce que le service sélectionne parfois un pod exécuté sur un autre nœud de travail qui peut également se trouver dans une autre AZ.
- Étant donné que le Load Balancer enregistre le nœud de travail comme cible, cela signifie que son bilan de santé envoyé à la cible ne sera pas reçu directement par le pod mais par le nœud de travail sur son nœud de travail NodePort et que le trafic de contrôle de santé suivra le même chemin que celui décrit ci-dessus.
- La surveillance et le dépannage sont plus complexes car le trafic transféré par le Load Balancer n'est pas directement envoyé aux pods. Vous devez donc soigneusement corréler le paquet reçu sur le Worker Node avec le Service ClusterIP et, éventuellement, avec le pod pour avoir une end-to-end visibilité complète sur le chemin du paquet afin de résoudre correctement les problèmes.



En revanche, si vous configurez le type de cible comme « IP » comme nous le recommandons, les implications seront les suivantes :

- Le trafic provenant du Load Balancer sera transféré directement vers le pod, ce qui simplifie le chemin réseau car il contourne les sauts supplémentaires précédents des nœuds de travail et de l'adresse IP du cluster de services, réduit la latence qui aurait autrement été encourue si le service avait transféré le trafic vers un pod situé dans une autre zone AZ et enfin, cela supprime le traitement de la surcharge de traitement par les règles iptables sur les nœuds de travail.
- Le bilan de santé du Load Balancer est reçu et traité directement par le pod, ce qui signifie que l'état cible « sain » ou « malsain » est une représentation directe de l'état de santé du pod.
- La surveillance et le dépannage sont simplifiés et tout outil utilisé pour capturer l'adresse IP du paquet révélera directement le trafic bidirectionnel entre le Load Balancer et le pod dans ses champs source et destination.



Pour créer un AWS Elastic Load Balancing qui utilise des cibles IP, vous devez ajouter :

- `alb.ingress.kubernetes.io/target-type: ip`annotation au manifeste de votre entrée lors de la configuration de votre entrée Kubernetes (Application Load Balancer)
- `service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip`annotation au manifeste de votre service lors de la configuration de votre service Kubernetes de type LoadBalancer (Network Load Balancer).

Configuration des vérifications de l'état de santé du Load Balancer

Bien que Kubernetes fournisse ses propres mécanismes de vérification de l'état (détaillés dans la section suivante), nous recommandons de mettre en œuvre les contrôles de santé ELB en tant que protection complémentaire qui fonctionne en dehors du plan de contrôle de Kubernetes. Cette couche indépendante continue de surveiller votre application même pendant :

- Interruptions du plan de contrôle Kubernetes
- Retards d'exécution des sondes

- Partitions réseau entre kubelet et pod

Pour les charges de travail critiques nécessitant une disponibilité maximale et une reprise accélérée dans les scénarios mentionnés ci-dessus, les bilans de santé ELB fournissent un filet de sécurité essentiel qui fonctionne parallèlement aux mécanismes natifs de Kubernetes, et non à leur place.

Afin de configurer et d'affiner les contrôles de santé de votre ELB, vous devez utiliser des annotations dans votre service Kubernetes ou dans votre manifeste d'entrée qui seront rapprochées par Service Controller ou AWS Load Balancer Controller.

Disponibilité et cycle de vie du pod

Lors de la mise à niveau d'une application, vous devez vous assurer que celle-ci est toujours disponible pour traiter les demandes afin que les utilisateurs ne subissent aucune interruption de service. L'un des défis courants de ce scénario consiste à synchroniser l'état de disponibilité de vos charges de travail entre la couche Kubernetes et l'infrastructure, par exemple les équilibreurs de charge externes. Les sections suivantes mettent en évidence les meilleures pratiques pour faire face à de tels scénarios.

> **Note**
>
> Les explications ci-dessous sont basées sur EndpointSlicesle remplacement recommandé des endpoints dans Kubernetes. Les différences entre les deux sont négligeables dans le contexte des scénarios décrits ci-dessous. AWS Load Balancer Controller consomme par défaut des points de terminaison, que vous pouvez activer EndpointSlices en les activant enable-endpoint-sliceflagsur le contrôleur.

Utiliser des bilans de santé

Kubernetes exécute par défaut le contrôle de l'état du processus où le processus kubelet sur le nœud vérifie si le processus principal du conteneur est en cours d'exécution ou non. Sinon, par défaut, il redémarre ce conteneur. Cependant, vous pouvez également configurer les sondes Kubernetes pour identifier lorsqu'un processus de conteneur est en cours d'exécution mais dans un état d'impasse, ou si une application a démarré avec succès ou non. Les sondes peuvent être basées sur les mécanismes exec, grpc, HttpGet et TCPSocket. En fonction du type et du résultat de la sonde, le conteneur peut être redémarré.

Veuillez consulter la section Création du pod dans l'annexe ci-dessous pour revoir la séquence des événements du processus de création du pod.

Utiliser des sondes de préparation

Par défaut, lorsque tous les conteneurs d'un pod fonctionnent, l'état du pod est considéré comme « prêt ». Cependant, il se peut que l'application ne soit toujours pas en mesure de traiter les demandes des clients. Par exemple, l'application peut avoir besoin d'extraire des données ou de la configuration d'une ressource externe pour pouvoir traiter les demandes. Dans un tel état, vous ne voudriez ni supprimer l'application ni lui transmettre de requêtes. La sonde de disponibilité vous permet de vous assurer que le pod n'est pas considéré comme « prêt », c'est-à-dire qu'il ne sera pas ajouté à l' EndpointSliceobjet tant que le résultat de la sonde ne le sera pas`success`. D'autre part, si la sonde tombe en panne plus loin sur la ligne, le Pod est retiré de l' EndpointSlice objet. Vous pouvez configurer une sonde de disponibilité dans le manifeste du Pod pour chaque conteneur. `kubelet`un processus sur chaque nœud exécute la sonde de préparation sur les conteneurs de ce nœud.

Utilisez les barrières de disponibilité des Pod

L'un des aspects de la sonde de préparation est le fait qu'elle ne contient aucun mécanisme de rétroaction/influence externe. Le processus kubelet sur le nœud exécute la sonde et définit l'état de la sonde. Cela n'a aucun impact sur les requêtes entre les microservices eux-mêmes dans la couche Kubernetes (trafic est-ouest) puisque le EndpointSlice Controller tient la liste des points de terminaison (Pods) toujours à jour. Pourquoi et quand auriez-vous besoin d'un mécanisme externe alors ?

Lorsque vous exposez vos applications en utilisant le type de service Kubernetes Load Balancer ou Kubernetes Ingress (pour le trafic nord-sud), la liste des Pod IPs pour le service Kubernetes correspondant doit être propagée vers l'équilibreur de charge de l'infrastructure externe afin que celui-ci dispose également d'une liste de cibles à jour. AWS Load Balancer Controller comble cette lacune. Lorsque vous utilisez AWS Load Balancer Controller et que vous en tirez part`target group: IP`, tout comme `kube-proxy` AWS Load Balancer Controller reçoit également une mise à jour (`watch`via), puis il communique avec l'API ELB pour configurer et commencer à enregistrer l'adresse IP du pod en tant que cible sur l'ELB.

Lorsque vous effectuez une mise à jour continue d'un déploiement, de nouveaux pods sont créés, et dès que l'état d'un nouveau pod est « prêt », un old/existing pod est résilié. Au cours de ce processus, l' EndpointSliceobjet Kubernetes est mis à jour plus rapidement que le temps nécessaire

à l'ELB pour enregistrer les nouveaux pods en tant que cibles, voir enregistrement des cibles. Pendant une courte période, il se peut que l'état de la couche Kubernetes ne corresponde pas à celui de l'infrastructure dans lequel les demandes des clients peuvent être abandonnées. Au cours de cette période, au sein de la couche Kubernetes, les nouveaux pods seraient prêts à traiter les demandes, mais du point de vue de l'ELB, ils ne le sont pas.

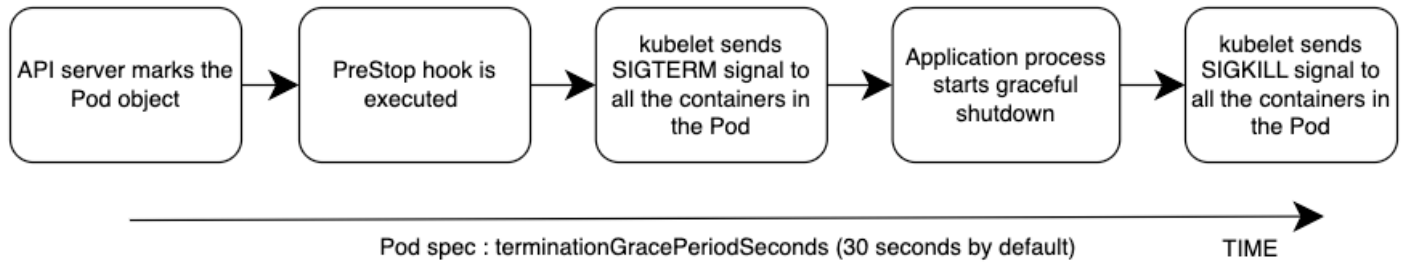
Pod Readiness Gates vous permet de définir des exigences supplémentaires qui doivent être satisfaites avant que l'état du pod soit considéré comme « prêt ». Dans le cas d'AWS ELB, le AWS Load Balancer Controller surveille l'état de la cible (le Pod) sur l'AWS ELB et une fois que l'enregistrement de la cible est terminé et que son statut passe à « Healthy », le contrôleur met à jour l'état du pod sur « Ready ». Cette approche vous permet d'influencer l'état du pod en fonction de l'état du réseau externe, qui est l'état cible sur l'AWS ELB. Les Pod Readiness Gates jouent un rôle crucial dans les scénarios de mise à jour continue, car ils vous permettent d'éviter que la mise à jour continue d'un déploiement ne mette fin aux anciens pods jusqu'à ce que le statut cible des nouveaux pods devienne « sain » sur AWS ELB.

Arrêtez les applications en douceur

Votre application doit répondre à un signal SIGTERM en démarrant son arrêt progressif afin que les clients ne subissent aucune interruption de service. Cela signifie que votre application doit exécuter des procédures de nettoyage telles que la sauvegarde des données, la fermeture des descripteurs de fichiers, la fermeture des connexions à la base de données, le traitement des demandes en cours de vol avec élégance et la sortie en temps opportun pour répondre à la demande de résiliation du Pod. Vous devez définir le délai de grâce suffisamment long pour que le nettoyage puisse être terminé. Pour savoir comment réagir au signal SIGTERM, vous pouvez consulter les ressources du langage de programmation correspondant que vous utilisez pour votre application.

Si votre application ne parvient pas à s'arrêter correctement à la réception d'un signal SIGTERM ou si elle ignore ou ne reçoit pas le signal, vous pouvez utiliser le PreStophook pour initier un arrêt progressif de l'application. Le hook Presttop est exécuté immédiatement avant l'envoi du signal SIGTERM et il peut effectuer des opérations arbitraires sans avoir à implémenter ces opérations dans le code de l'application lui-même.

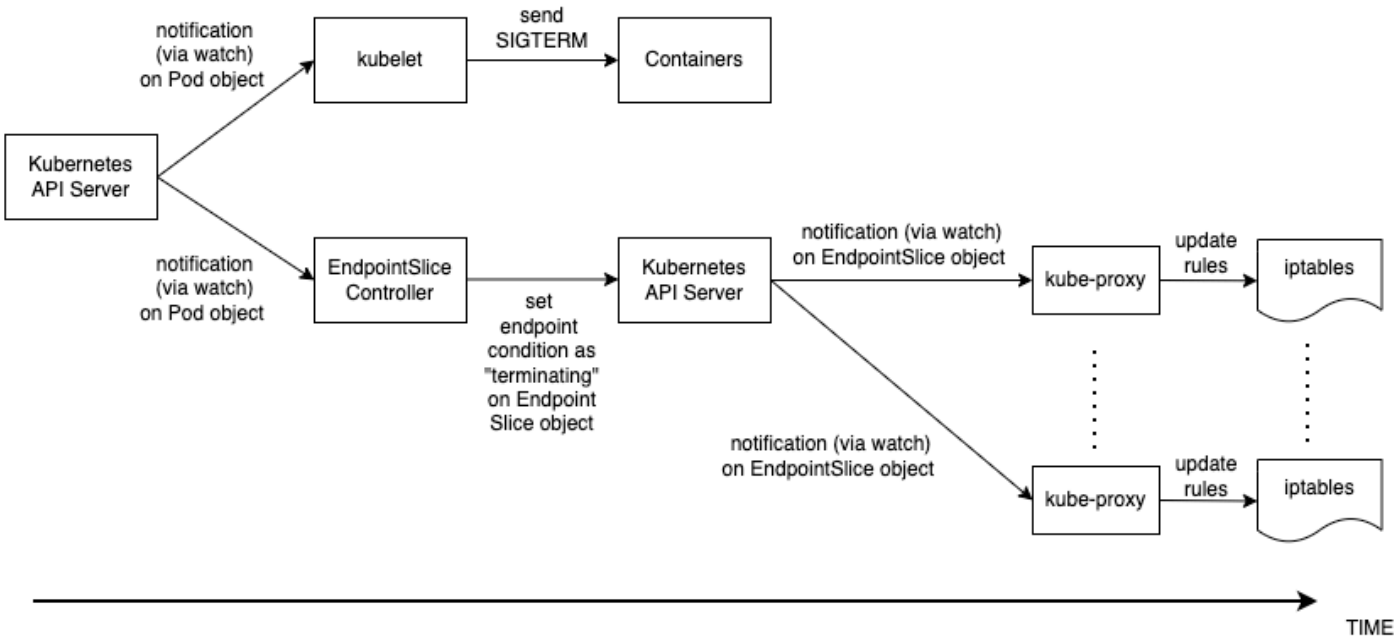
La séquence globale des événements est illustrée dans le schéma ci-dessous. Remarque : quel que soit le résultat de la procédure d'arrêt progressif de l'application ou le résultat du PreStop hook, les conteneurs d'applications sont finalement fermés à la fin de la période de grâce via SIGKILL.



Veuillez consulter la section Suppression du pod dans l'annexe ci-dessous pour revoir la séquence des événements du processus de suppression du pod.

Gérez avec élégance les demandes des clients

La séquence des événements de la suppression du pod est différente de celle de la création du pod. Lorsqu'un pod est créé, l'adresse IP du pod est `kubelet` mise à jour dans l'API Kubernetes et ce n'est qu'alors que l' EndpointSlice objet est mis à jour. D'autre part, lorsqu'un Pod est arrêté, l'API Kubernetes avertit à la fois le kubelet et EndpointSlice le contrôleur. Inspectez soigneusement le schéma suivant qui montre la séquence des événements.



La façon dont l'état se propage depuis le serveur API jusqu'aux règles iptables sur les nœuds expliquées ci-dessus crée une condition de course intéressante. Parce qu'il y a de fortes chances que le conteneur reçoive le signal SIGKILL bien avant que le kube-proxy sur chaque nœud ne mette à jour les règles iptables locales. Dans un tel cas, deux scénarios méritent d'être mentionnés :

- Si votre application abandonne immédiatement et brutalement les demandes et les connexions en cours de vol dès réception de SIGTERM, les clients seront confrontés à 50 fois plus d'erreurs partout.
- Même si votre application garantit que toutes les demandes et connexions en cours de vol sont traitées dans leur intégralité à la réception de SIGTERM, les nouvelles demandes des clients seront toujours envoyées au conteneur de l'application pendant la période de grâce, car les règles iptables ne sont peut-être pas encore mises à jour. Jusqu'à ce que la procédure de nettoyage ferme le socket du serveur sur le conteneur, ces nouvelles demandes entraîneront de nouvelles connexions. Lorsque le délai de grâce prend fin, les connexions établies après le SIGTERM sont alors abandonnées sans condition puisque SIGKILL est envoyé.

La définition d'une période de grâce suffisamment longue dans la spécification Pod peut résoudre ce problème, mais en fonction du délai de propagation et du nombre réel de demandes clients, il est difficile de prévoir le temps qu'il faudra à l'application pour fermer les connexions correctement. Par conséquent, l'approche la moins parfaite mais la plus faisable consiste à utiliser un PreStop hook pour retarder le signal SIGTERM jusqu'à ce que les règles iptables soient mises à jour afin de s'assurer qu'aucune nouvelle demande client n'est envoyée à l'application. Au lieu de cela, seules les connexions existantes sont maintenues. PreStop hook peut être un simple gestionnaire Exec tel que. `sleep 10`

Le comportement et la recommandation mentionnés ci-dessus s'appliquent également lorsque vous exposez vos applications à l'aide de Load Balancer, de type Kubernetes Service, ou de Kubernetes Ingress (pour le trafic nord-sud) à l'aide d'AWS Load Balancer Controller et de l'effet de levier. `target group: IP` Parce que tout comme `kube-proxy` le AWS Load Balancer Controller reçoit également une mise à jour (via watch) sur l' EndpointSlice objet, puis il communique avec l'API ELB pour commencer à désenregistrer l'adresse IP du pod auprès de l'ELB. Cependant, en fonction de la charge de l'API Kubernetes ou de l'API ELB, cela peut également prendre du temps et le SIGTERM a peut-être déjà été envoyé à l'application il y a longtemps. Une fois que l'ELB commence à désenregistrer la cible, il arrête d'envoyer des demandes à cette cible afin que l'application ne reçoive aucune nouvelle demande et l'ELB entame également un délai de désenregistrement de 300 secondes par défaut. Pendant le processus de désinscription, l'ELB `draining` attend essentiellement que les requests/existing connexions en vol vers cette cible soient épuisées. Une fois le délai de désenregistrement expiré, la cible n'est plus utilisée et toutes les demandes en vol adressées à cette cible sont supprimées de force.

Utiliser le budget d'interruption du Pod

Configurez un budget d'interruption de service (PDB) pour vos applications. PDBlimits le nombre de pods d'une application répliquée qui sont simultanément indisponibles en raison d'interruptions volontaires. Cela garantit qu'un nombre ou un pourcentage minimum de pods restent disponibles dans un déploiement StatefulSet ou. Par exemple, une application basée sur un quorum doit veiller à ce que le nombre de répliques en cours d'exécution ne soit jamais inférieur au nombre requis pour un quorum. Une interface Web peut également garantir que le nombre de répliques servant la charge ne tombe jamais en dessous d'un certain pourcentage du total. PDB protégera l'application contre des actions telles que le drainage des nœuds ou le déploiement de nouvelles versions de déploiements. N'oubliez pas que les PDB ne protégeront pas l'application contre les perturbations involontaires telles qu'une défaillance du système d'exploitation du nœud ou une perte de connectivité réseau. Pour plus d'informations, reportez-vous à la documentation relative à la spécification d'un budget d'interruption pour votre application dans Kubernetes.

Références

- KubeCon Session Europe 2019 - Prêt ? Une étude approfondie de Pod Readiness Gates for Service Health
- Livre - Kubernetes en action
- Blog AWS - Comment faire évoluer rapidement votre application avec ALB sur EKS (sans perte de trafic)

Annexe

Création de pods

Il est impératif de comprendre quelle est la séquence des événements dans un scénario dans lequel un Pod est déployé, puis il doit healthy/ready recevoir et traiter les demandes des clients. Parlons de la séquence des événements.

1. Un pod est créé sur le plan de contrôle Kubernetes (c'est-à-dire par une commande kubectl, une mise à jour de déploiement ou une action de dimensionnement).
2. `kube-scheduler`assigne le Pod à un nœud du cluster.
3. Le processus kubelet exécuté sur le nœud attribué reçoit la mise à jour (via`watch`) et communique avec le moteur d'exécution du conteneur pour démarrer les conteneurs définis dans la spécification Pod.

4. Lorsque les conteneurs commencent à fonctionner, le kubelet met à jour la condition du Pod comme `Ready` dans l'objet Pod de l'API Kubernetes.
5. Le EndpointSliceController reçoit la mise à jour de l'état du Pod (via`watch`) et ajoute le Pod IP/Port en tant que nouveau point de terminaison à l'EndpointSliceobjet (liste des Pod IPs) du service Kubernetes correspondant.
6. Le processus kube-proxy sur chaque nœud reçoit la mise à jour (via`watch`) sur l'EndpointSlice objet, puis met à jour les règles iptables sur chaque nœud, avec le nouveau port IP/IP du pod.

Suppression du pod

Tout comme lors de la création d'un pod, il est impératif de comprendre quelle est la séquence des événements lors de la suppression du pod. Parlons de la séquence des événements.

1. Une demande de suppression de Pod est envoyée au serveur d'API Kubernetes (c'est-à-dire par une `kubectl` commande, une mise à jour du déploiement ou une action de dimensionnement).
2. Le serveur d'API Kubernetes lance une période de grâce, qui est de 30 secondes par défaut, en définissant le champ DeletionTimestamp dans l'objet Pod. (La période de grâce peut être configurée dans les spécifications du Pod jusqu'à`terminationGracePeriodSeconds`)
3. Le `kubelet` processus exécuté sur le nœud reçoit la mise à jour (via une montre) sur l'objet Pod et envoie un signal SIGTERM à l'identifiant de processus 1 (PID 1) à l'intérieur de chaque conteneur de ce Pod. Il regarde ensuite le`terminationGracePeriodSeconds`.
4. Le EndpointSliceController reçoit également la mise à jour (via`watch`) de l'étape 2 et définit la condition du point de terminaison sur « terminaison » dans l'EndpointSliceobjet (liste des Pod IPs) du service Kubernetes correspondant.
5. Le processus kube-proxy sur chaque nœud reçoit la mise à jour (via`watch`) sur l'EndpointSlice objet, puis les règles iptables sur chaque nœud sont mises à jour par le kube-proxy pour arrêter de transférer les demandes des clients au Pod.
6. À l'`terminationGracePeriodSeconds`expiration, le signal SIGKILL est `kubelet` envoyé au processus parent de chaque conteneur du Pod et y met fin de force.
7. TheEndpointSliceLe contrôleur supprime le point de terminaison de l'EndpointSliceobjet.
8. Le serveur API supprime l'objet Pod.

Surveillance des charges de travail EKS pour détecter les problèmes de performance du réseau

Surveillance du trafic CoreDNS pour détecter les problèmes de limitation du DNS

L'exécution de charges de travail intensives en DNS peut parfois entraîner des défaillances CoreDNS intermittentes en raison de la limitation du DNS, ce qui peut avoir un impact sur les applications et entraîner des erreurs occasionnelles. UnknownHostException

Le déploiement de CoreDNS comporte une politique anti-affinité qui demande au planificateur Kubernetes d'exécuter des instances de CoreDNS sur des nœuds de travail distincts du cluster, c'est-à-dire d'éviter de colocaliser des répliques sur le même nœud de travail. Cela réduit efficacement le nombre de requêtes DNS par interface réseau, car le trafic provenant de chaque réplique est acheminé via un ENI différent. Si vous remarquez que les requêtes DNS sont limitées en raison de la limite de 1 024 paquets par seconde, vous pouvez 1) essayer d'augmenter le nombre de répliques CoreDNS ou 2) implémenter. NodeLocal DNSCache Voir Surveiller les métriques CoreDNS pour plus d'informations.

Défi

- La perte de paquets se produit en quelques secondes et il peut être difficile pour nous de surveiller correctement ces modèles afin de déterminer si la limitation du DNS se produit réellement.
- Les requêtes DNS sont limitées au niveau de l'interface elastic network. Ainsi, les requêtes limitées n'apparaissent pas dans la journalisation des requêtes.
- Les journaux de flux ne capturent pas tout le trafic IP. P. ex. Le trafic généré par des instances lorsqu'elles contactent le serveur DNS Amazon. Si vous utilisez votre propre serveur DNS, tout le trafic vers ce serveur DNS est enregistré

Solution

Un moyen simple d'identifier les problèmes de limitation du DNS dans les nœuds de travail consiste à capturer `linklocal_allowance_exceeded` des métriques. Le linklocal_allowance_exceeded est le nombre de paquets abandonnés parce que le PPS du trafic vers les services proxy locaux a dépassé le maximum pour l'interface réseau. Cela affecte le trafic vers le service DNS, le service des métadonnées d’instance et le service Amazon Time Sync. Au lieu de suivre cet événement en temps

réel, nous pouvons également diffuser cette métrique sur Amazon Managed Service for Prometheus et les faire visualiser dans Amazon Managed Grafana

Surveillance des délais de requêtes DNS à l'aide des métriques Conntrack

Une autre métrique qui peut aider à surveiller la limitation du CoreDNS et le délai de requête est et. `conntrack_allowance_available` `conntrack_allowance_exceeded` Les défaillances de connectivité causées par le dépassement des limites de connexion suivies peuvent avoir un impact plus important que celles résultant du dépassement des autres tolérances. Lorsque vous utilisez le protocole TCP pour transférer des données, les paquets mis en file d'attente ou abandonnés en raison du dépassement des limites du réseau d'instance EC2, telles que la bande passante, le PPS, etc., sont généralement gérés correctement grâce aux capacités de contrôle de congestion du TCP. Les flux concernés seront ralentis et les paquets perdus seront retransmis. Toutefois, lorsqu'une instance dépasse son quota de connexions suivies, aucune nouvelle connexion ne peut être établie tant que certaines des connexions existantes ne sont pas fermées pour faire de la place à de nouvelles connexions.

`conntrack_allowance_available`et `conntrack_allowance_exceeded` aide les clients à surveiller le nombre de connexions suivies, qui varie selon les cas. Ces indicateurs de performance réseau donnent aux clients une visibilité sur le nombre de paquets mis en file d'attente ou abandonnés lorsque les limites réseau d'une instance, telles que la bande passante réseau Packets-Per-Second (PPS), le suivi des connexions et l'accès aux services locaux liés (Amazon DNS, Instance Meta Data Service, Amazon Time Sync) sont dépassées

`conntrack_allowance_available`est le nombre de connexions suivies qui peuvent être établies par l'instance avant d'atteindre le seuil de connexions suivies autorisé pour ce type d'instance (pris en charge uniquement pour les instances basées sur Nitro). `conntrack_allowance_exceeded`est le nombre de paquets abandonnés car le suivi des connexions a dépassé le maximum pour l'instance et aucune nouvelle connexion n'a pu être établie.

Autres indicateurs de performance réseau importants

Parmi les autres indicateurs de performance réseau importants, citons :

`bw_in_allowance_exceeded`(la valeur idéale de la métrique doit être zéro) est le nombre de paquets mis en file d'attente and/or abandonnés parce que la bande passante agrégée entrante a dépassé le maximum pour l'instance

`bw_out_allowance_exceeded`(la valeur idéale de la métrique doit être zéro) est le nombre de paquets mis en file d'attente and/or abandonnés parce que la bande passante agrégée sortante a dépassé le maximum pour l'instance

`pps_allowance_exceeded`(la valeur idéale de la métrique doit être zéro) est le nombre de paquets mis en file d'attente and/or abandonnés parce que le PPS bidirectionnel a dépassé le maximum pour l'instance

Capture des métriques pour surveiller les charges de travail et détecter les problèmes de performance du réseau

Le pilote Elastic Network Adapter (ENA) publie les mesures de performance réseau décrites ci-dessus à partir des instances où elles sont activées. Toutes les mesures de performance du réseau peuvent être publiées à CloudWatch l'aide de l' CloudWatch agent. Veuillez consulter le blog pour plus d'informations.

Capturons maintenant les statistiques décrites ci-dessus, stockons-les dans Amazon Managed Service for Prometheus et visualisons-les à l'aide d'Amazon Managed Grafana

Conditions préalables

- ethtool - Assurez-vous que ethtool est installé sur les nœuds de travail
- Un espace de travail AMP configuré dans votre compte AWS. Pour obtenir des instructions, voir Création d'un espace de travail dans le guide de l'utilisateur AMP.
- Espace de travail géré par Amazon à Grafana

Déploiement de Prometheus Node Exporter

Afin de collecter les métriques ethtool pour les nœuds de travail, nous devons déployer un collecteur qui rassemble les métriques et les expose au format Prometheus. Les commandes suivantes ajouteront le référentiel `prometheus-community` Helm et installeront le `prometheus-node-exporter` graphique avec le `ethtool` collecteur activé.

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm upgrade -i prometheus-node-exporter prometheus-community/prometheus-node-exporter
 \
  --set extraArgs[0]="--collector.ethtool" \
```

```
  --set extraArgs[1]="--collector.ethtool.device-include=(eth|em|eno|ens|enp)[0-9s]+" \
  --set extraArgs[2]="--collector.ethtool.metrics-include=.*" \
  --set podAnnotations."prometheus\.io/scrape='true'" \
  --set podAnnotations."prometheus\.io/port='9100'"
```

Déployez le collecteur ADOT pour récupérer les métriques ethtool et les stocker dans l'espace de travail Amazon Managed Service for Prometheus

Chaque cluster pour lequel vous installez AWS Distro OpenTelemetry (ADOT) doit avoir ce rôle pour autoriser votre compte de service AWS à stocker des métriques dans Amazon Managed Service for Prometheus. Suivez ces étapes pour créer et associer votre rôle IAM à votre compte de service Amazon EKS à l'aide de IRSA :

```
eksctl create iamserviceaccount --name adot-collector --namespace default
 --cluster <CLUSTER_NAME> --attach-policy-arn arn:aws:iam::aws:policy/
AmazonPrometheusRemoteWriteAccess --attach-policy-arn arn:aws:iam::aws:policy/
AWSXrayWriteOnlyAccess --attach-policy-arn arn:aws:iam::aws:policy/
CloudWatchAgentServerPolicy --region <REGION> --approve  --override-existing-
serviceaccounts
```

Déployons le collecteur ADOT pour extraire les métriques de l'exportateur Prometheus ethtool et les stocker dans Amazon Managed Service for Prometheus

La procédure suivante utilise un exemple de fichier YAML avec le déploiement comme valeur de mode. Il s'agit du mode par défaut, qui déploie le collecteur ADOT de la même manière qu'une application autonome. Cette configuration reçoit les métriques OTLP de l'exemple d'application et les métriques Amazon Managed Service for Prometheus extraites des pods du cluster.

```
curl -o collector-config-amp.yaml https://raw.githubusercontent.com/aws-observability/
aws-otel-community/master/sample-configs/operator/collector-config-amp.yaml
```

Dans le collector-config-amp fichier .yaml, remplacez les valeurs suivantes par vos propres valeurs :

- mode : déploiement
- Compte de service : adot-collector
- point de terminaison : <YOUR_REMOTE_WRITE_ENDPOINT>
- région : <VOTRE_ >AWS_REGION
- nom : adot-collector

```
kubectl apply -f collector-config-amp.yaml
```

Une fois le collecteur adot déployé, les métriques seront stockées avec succès dans Amazon Prometheus

Configurer le gestionnaire d'alertes dans Amazon Managed Service pour que Prometheus envoie des notifications

Vous pouvez utiliser le gestionnaire d'alertes d'Amazon Managed Service for Prometheus pour configurer des règles d'alerte pour les alertes critiques, puis vous pouvez envoyer des notifications à un sujet Amazon SNS. Configurons les règles d'enregistrement et les règles d'alerte pour vérifier les indicateurs discutés jusqu'à présent.

Nous utiliserons le contrôleur ACK pour Amazon Managed Service for Prometheus pour définir les règles d'alerte et d'enregistrement.

Déployons le contrôleur ACL pour le service Amazon Managed Service for Prometheus :

```
export SERVICE=prometheusservice
export RELEASE_VERSION=`curl -sL https://api.github.com/repos/aws-controllers-k8s/
$SERVICE-controller/releases/latest | grep '"tag_name":' | cut -d'"' -f4`
export ACK_SYSTEM_NAMESPACE=ack-system
export AWS_REGION=us-east-1
aws ecr-public get-login-password --region us-east-1 | helm registry login --username
 AWS --password-stdin public.ecr.aws
helm install --create-namespace -n $ACK_SYSTEM_NAMESPACE ack-$SERVICE-controller \
oci://public.ecr.aws/aws-controllers-k8s/$SERVICE-chart --version=$RELEASE_VERSION --
set=aws.region=$AWS_REGION
```

Exécutez la commande et après quelques instants, vous devriez voir le message suivant :

```
You are now able to create Amazon Managed Service for Prometheus (AMP) resources!

The controller is running in "cluster" mode.

The controller is configured to manage AWS resources in region: "us-east-1"

The ACK controller has been successfully installed and ACK can now be used to provision
 an Amazon Managed Service for Prometheus workspace.
```

Créons maintenant un fichier yaml pour configurer la définition du gestionnaire d'alertes et les groupes de règles. Enregistrez le fichier ci-dessous sous `rulegroup.yaml`

```
apiVersion: prometheusservice.services.k8s.aws/v1alpha1
kind: RuleGroupsNamespace
metadata:
   name: default-rule
spec:
   workspaceID: <Your WORKSPACE-ID>
   name: default-rule
   configuration: |
     groups:
     - name: ppsallowance
       rules:
       - record: metric:pps_allowance_exceeded
         expr: rate(node_net_ethtool{device="eth0",type="pps_allowance_exceeded"}[30s])
       - alert: PPSAllowanceExceeded
         expr: rate(node_net_ethtool{device="eth0",type="pps_allowance_exceeded"}
 [30s]) > 0
         labels:
           severity: critical

         annotations:
           summary: Connections dropped due to total allowance exceeding for the
 (instance {{ $labels.instance }})
           description: "PPSAllowanceExceeded is greater than 0"
     - name: bw_in
       rules:
       - record: metric:bw_in_allowance_exceeded
         expr: rate(node_net_ethtool{device="eth0",type="bw_in_allowance_exceeded"}
[30s])
       - alert: BWINAllowanceExceeded
         expr: rate(node_net_ethtool{device="eth0",type="bw_in_allowance_exceeded"}
 [30s]) > 0
         labels:
           severity: critical

         annotations:
           summary: Connections dropped due to total allowance exceeding for the
 (instance {{ $labels.instance }})
           description: "BWInAllowanceExceeded is greater than 0"
     - name: bw_out
       rules:
```

```
      - record: metric:bw_out_allowance_exceeded
        expr: rate(node_net_ethtool{device="eth0",type="bw_out_allowance_exceeded"}
[30s])
      - alert: BWOutAllowanceExceeded
        expr: rate(node_net_ethtool{device="eth0",type="bw_out_allowance_exceeded"}
 [30s]) > 0
        labels:
          severity: critical

        annotations:
          summary: Connections dropped due to total allowance exceeding for the
 (instance {{ $labels.instance }})
          description: "BWoutAllowanceExceeded is greater than 0"
    - name: conntrack
      rules:
      - record: metric:conntrack_allowance_exceeded
        expr: rate(node_net_ethtool{device="eth0",type="conntrack_allowance_exceeded"}
[30s])
      - alert: ConntrackAllowanceExceeded
        expr: rate(node_net_ethtool{device="eth0",type="conntrack_allowance_exceeded"}
 [30s]) > 0
        labels:
          severity: critical

        annotations:
          summary: Connections dropped due to total allowance exceeding for the
 (instance {{ $labels.instance }})
          description: "ConnTrackAllowanceExceeded is greater than 0"
    - name: linklocal
      rules:
      - record: metric:linklocal_allowance_exceeded
        expr: rate(node_net_ethtool{device="eth0",type="linklocal_allowance_exceeded"}
[30s])
      - alert: LinkLocalAllowanceExceeded
        expr: rate(node_net_ethtool{device="eth0",type="linklocal_allowance_exceeded"}
 [30s]) > 0
        labels:
          severity: critical

        annotations:
          summary: Packets dropped due to PPS rate allowance exceeded for local
 services  (instance {{ $labels.instance }})
          description: "LinkLocalAllowanceExceeded is greater than 0"
```

Remplacez votre WORKSPACE-ID par l'identifiant de l'espace de travail que vous utilisez.

Configurons maintenant la définition du gestionnaire d'alertes. Enregistrez le fichier ci-dessous sous `alertmanager.yaml`

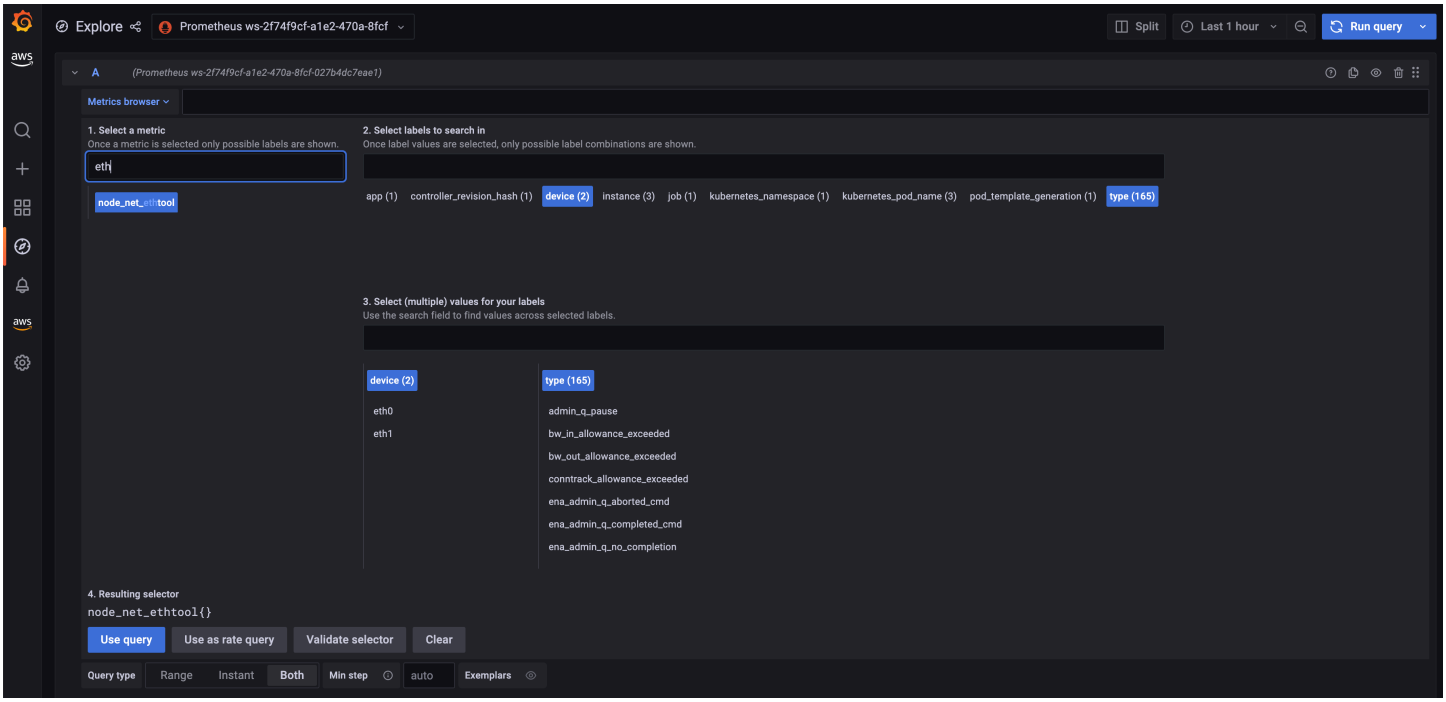
```
apiVersion: prometheusservice.services.k8s.aws/v1alpha1
kind: AlertManagerDefinition
metadata:
  name: alert-manager
spec:
  workspaceID: <Your WORKSPACE-ID >
  configuration: |
    alertmanager_config: |
      route:
         receiver: default_receiver
       receivers:
       - name: default_receiver
          sns_configs:
          - topic_arn: TOPIC-ARN
            sigv4:
              region: REGION
            message: |
              alert_type: {{ .CommonLabels.alertname }}
              event_type: {{ .CommonLabels.event_type }}
```

Remplacez votre WORKSPACE-ID par l'ID d'espace de travail du nouvel espace de travail, TOPIC-ARN par l'ARN d'une rubrique Amazon Simple Notification Service à laquelle vous souhaitez envoyer les alertes, et REGION par la région actuelle de la charge de travail. Assurez-vous que votre espace de travail est autorisé à envoyer des messages à Amazon SNS.

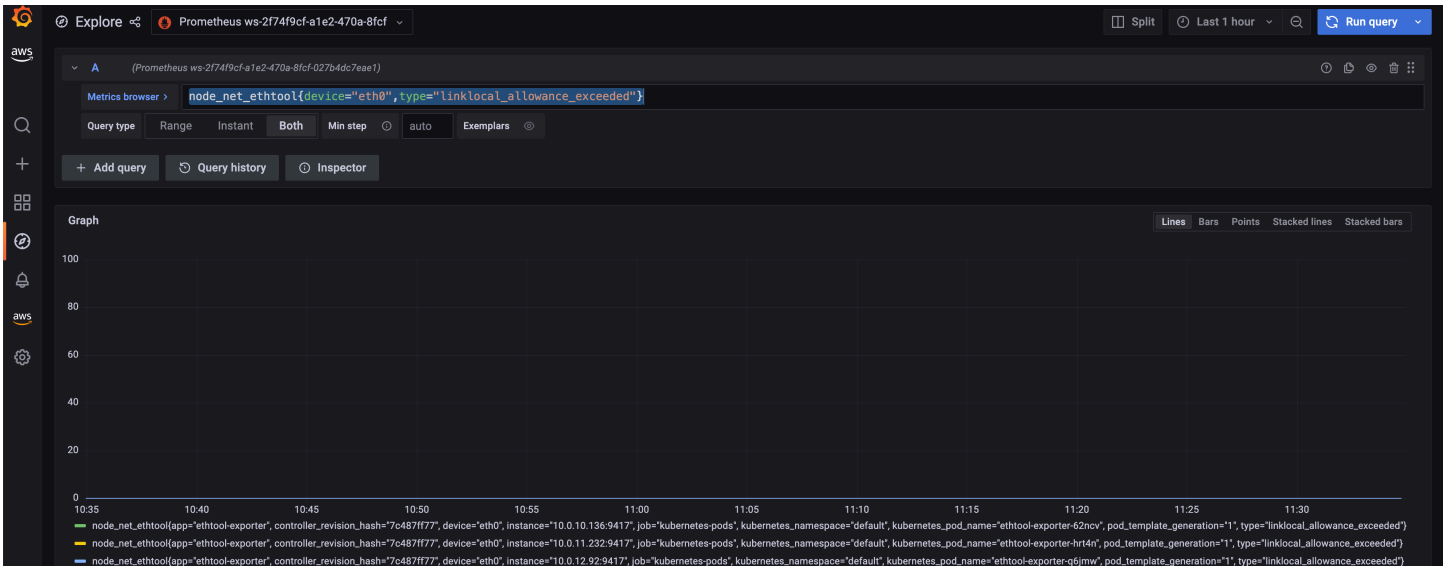
Visualisez les métriques d'ethtool dans Amazon Managed Grafana

Visualisons les statistiques dans Amazon Managed Grafana et créons un tableau de bord. Configurez le service géré Amazon pour Prometheus en tant que source de données dans la console Amazon Managed Grafana. Pour obtenir des instructions, voir Ajouter Amazon Prometheus en tant que source de données

Explorons maintenant les statistiques dans Amazon Managed Grafana : cliquez sur le bouton Explorer et recherchez ethtool :



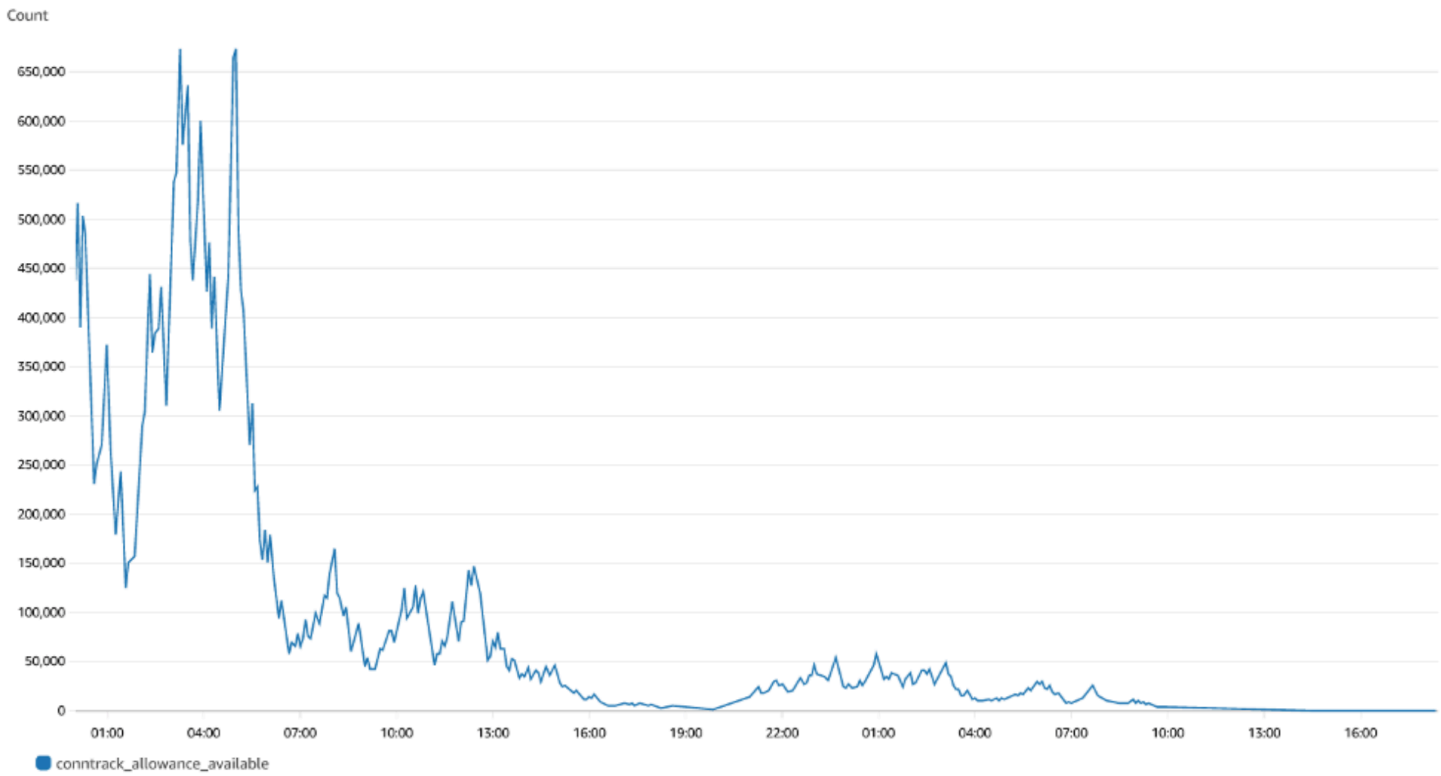
Créons un tableau de bord pour la métrique linklocal_allowance_exceeded à l'aide de la requête. `rate(node_net_ethtool{device="eth0",type="linklocal_allowance_exceeded"}[30s])` Il en résultera le tableau de bord ci-dessous.



Nous pouvons clairement voir qu'aucun paquet n'a été déposé car la valeur est nulle.

Créons un tableau de bord pour la métrique conntrack_allowance_exceeded à l'aide de la requête. `rate(node_net_ethtool{device="eth0",type="conntrack_allowance_exceeded"}[30s])` Il en résultera le tableau de bord ci-dessous.

La métrique `conntrack_allowance_exceeded` peut être visualisée dans CloudWatch, à condition que vous exécutiez un agent cloudwatch comme décrit ici. Le tableau de bord qui en CloudWatch résultera ressemblera à ce qui suit :



Nous pouvons clairement voir qu'aucun paquet n'a été déposé car la valeur est nulle. Si vous utilisez des instances basées sur Nitro, vous pouvez créer un tableau de bord similaire `conntrack_allowance_available` et surveiller de manière proactive les connexions dans votre

instance EC2. Vous pouvez étendre cela en configurant des alertes dans Amazon Managed Grafana pour envoyer des notifications à Slack, SNS, Pagerduty, etc.

Exécution de kube-proxy en mode IPVS

EKS en mode IP Virtual Server (IPVS) résout le problème de latence réseau souvent rencontré lors de l'exécution de grands clusters avec plus de 1 000 services avec kube-proxy exécuté en mode iptables traditionnel. Ce problème de performance est le résultat du traitement séquentiel des règles de filtrage de paquets iptables pour chaque paquet. Ce problème de latence a été résolu dans nftables, le successeur d'iptables. Cependant, au moment de la rédaction de cet article, kube-proxy est toujours en cours de développement pour utiliser nftables. Pour contourner ce problème, vous pouvez configurer votre cluster pour qu'il s'exécute `kube-proxy` en mode IPVS.

Présentation de

IPVS, qui est devenu GA depuis la version 1.11 de Kubernetes, utilise des tables de hachage plutôt que la recherche linéaire pour traiter les paquets, ce qui améliore l'efficacité des clusters comportant des milliers de nœuds et de services. IPVS a été conçu pour l'équilibrage de charge, ce qui en fait une solution adaptée aux problèmes de performances réseau de Kubernetes.

IPVS propose plusieurs options pour répartir le trafic vers les modules principaux. Vous trouverez des informations détaillées sur chaque option dans la documentation officielle de Kubernetes, mais une liste simple est présentée ci-dessous. Round Robin et Least Connections font partie des options d'équilibrage de charge IPVS les plus populaires dans Kubernetes.

```
- rr (Round Robin)
- wrr (Weighted Round Robin)
- lc (Least Connections)
- wlc (Weighted Least Connections)
- lblc (Locality Based Least Connections)
- lblcr (Locality Based Least Connections with Replication)
- sh (Source Hashing)
- dh (Destination Hashing)
- sed (Shortest Expected Delay)
- nq (Never Queue)
```

Mise en œuvre

Seules quelques étapes sont nécessaires pour activer IPVS dans votre cluster EKS. La première chose à faire est de vous assurer que le `ipvsadm` package d'administration du serveur virtuel Linux

est installé sur les images de vos nœuds de travail EKS. Pour installer ce package sur une image basée sur Fedora, telle qu'Amazon Linux 2023, vous pouvez exécuter la commande suivante sur l'instance du nœud de travail.

```
sudo dnf install -y ipvsadm
```

Sur une image basée sur Debian, telle qu'Ubuntu, la commande d'installation ressemblerait à ceci.

```
sudo apt-get install ipvsadm
```

Ensuite, vous devez charger les modules du noyau pour les options de configuration IPVS répertoriées ci-dessus. Nous recommandons d'écrire ces modules dans un fichier situé à l'intérieur du `/etc/modules-load.d/` répertoire afin qu'ils puissent survivre au redémarrage.

```
sudo sh -c 'cat << EOF > /etc/modules-load.d/ipvs.conf
ip_vs
ip_vs_rr
ip_vs_wrr
ip_vs_lc
ip_vs_wlc
ip_vs_lblc
ip_vs_lblcr
ip_vs_sh
ip_vs_dh
ip_vs_sed
ip_vs_nq
nf_conntrack
EOF'
```

Vous pouvez exécuter la commande suivante pour charger ces modules sur une machine déjà en cours d'exécution.

```
sudo modprobe ip_vs
sudo modprobe ip_vs_rr
sudo modprobe ip_vs_wrr
sudo modprobe ip_vs_lc
sudo modprobe ip_vs_wlc
sudo modprobe ip_vs_lblc
sudo modprobe ip_vs_lblcr
sudo modprobe ip_vs_sh
```

```
sudo modprobe ip_vs_dh
sudo modprobe ip_vs_sed
sudo modprobe ip_vs_nq
sudo modprobe nf_conntrack
```

> **Note**
>
> Il est vivement recommandé d'exécuter ces étapes du nœud de travail dans le cadre du processus d'amorçage de votre nœud de travail via un script de données utilisateur ou dans tout script de génération exécuté pour créer une AMI de nœud de travail personnalisée.

Ensuite, vous allez configurer votre cluster `kube-proxy` DaemonSet pour qu'il s'exécute en mode IPVS. Cela se fait en réglant le `kube-proxy` `mode` to `ipvs` et le `ipvs scheduler` to sur l'une des options d'équilibrage de charge répertoriées ci-dessus, par exemple : `rr` pour Round Robin.

> **Warning**
>
> Il s'agit d'un changement perturbateur qui doit être effectué en dehors des heures de bureau. Nous vous recommandons d'apporter ces modifications lors de la création initiale du cluster EKS afin de minimiser les impacts.

Vous pouvez émettre une commande AWS CLI pour activer IPVS en mettant à jour le module complémentaire `kube-proxy` EKS.

```
aws eks update-addon --cluster-name $CLUSTER_NAME --addon-name kube-proxy \
  --configuration-values '{"ipvs": {"scheduler": "rr"}, "mode": "ipvs"}' \
  --resolve-conflicts OVERWRITE
```

Vous pouvez également le faire en modifiant le `kube-proxy-config` ConfigMap dans votre cluster.

```
kubectl -n kube-system edit cm kube-proxy-config
```

Recherchez le `scheduler` paramètre ci-dessous `ipvs` et définissez la valeur sur l'une des options d'équilibrage de charge ipvs répertoriées ci-dessus, par exemple : `rr` pour Round Robin. Trouvez le `mode` paramètre, qui est par défaut`iptables`, et remplacez la valeur par. `ipvs` Le résultat de l'une ou l'autre option doit ressembler à la configuration ci-dessous.

```
  iptables:
    masqueradeAll: false
    masqueradeBit: 14
    minSyncPeriod: 0s
    syncPeriod: 30s
  ipvs:
    excludeCIDRs: null
    minSyncPeriod: 0s
    scheduler: "rr"
    syncPeriod: 30s
  kind: KubeProxyConfiguration
  metricsBindAddress: 0.0.0.0:10249
  mode: "ipvs"
  nodePortAddresses: null
  oomScoreAdj: -998
  portRange: ""
  udpIdleTimeout: 250ms
```

Si vos nœuds de travail ont été joints à votre cluster avant d'apporter ces modifications, vous devrez redémarrer le kube-proxy DaemonSet.

```
kubectl -n kube-system rollout restart ds kube-proxy
```

Validation

Vous pouvez vérifier que votre cluster et vos nœuds de travail s'exécutent en mode IPVS en exécutant la commande suivante sur l'un de vos nœuds de travail.

```
sudo ipvsadm -L
```

Au minimum, vous devriez voir un résultat similaire à celui ci-dessous, affichant les entrées pour le service Kubernetes API Server à et le service CoreDNS à `10.100.0.1` l'adresse. `10.100.0.10`

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  ip-10-100-0-1.us-east-1. rr
  -> ip-192-168-113-81.us-eas Masq        1      0          0
  -> ip-192-168-162-166.us-ea Masq        1      1          0
TCP  ip-10-100-0-10.us-east-1 rr
  -> ip-192-168-104-215.us-ea Masq        1      0          0
```

```
  -> ip-192-168-123-227.us-ea Masq        1      0          0
UDP  ip-10-100-0-10.us-east-1 rr
  -> ip-192-168-104-215.us-ea Masq        1      0          0
  -> ip-192-168-123-227.us-ea Masq        1      0          0
```

> **Note**
>
> Cet exemple de sortie provient d'un cluster EKS dont la plage d'adresses IP de service est de`10.100.0.0/16`.

Bonnes pratiques d'évolutivité d'EKS

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Ce guide fournit des conseils pour la mise à l'échelle des clusters EKS. L'objectif de la mise à l'échelle d'un cluster EKS est de maximiser la quantité de travail qu'un seul cluster peut effectuer. L'utilisation d'un seul cluster EKS de grande taille peut réduire la charge opérationnelle par rapport à l'utilisation de plusieurs clusters, mais elle comporte des inconvénients pour des éléments tels que les déploiements multirégionaux, l'isolation des locataires et les mises à niveau de clusters. Dans ce document, nous allons nous concentrer sur la manière d'atteindre une évolutivité maximale avec un seul cluster.

Comment utiliser ce guide

Ce guide est destiné aux développeurs et aux administrateurs chargés de créer et de gérer des clusters EKS dans AWS. Il se concentre sur certaines pratiques génériques de dimensionnement de Kubernetes, mais il n'est pas spécifique aux clusters Kubernetes autogérés ou aux clusters exécutés en dehors d'une région AWS avec EKS Anywhere.

Chaque rubrique comporte un bref aperçu, suivi de recommandations et de bonnes pratiques pour l'exploitation de clusters EKS à grande échelle. Les rubriques n'ont pas besoin d'être lues dans un ordre particulier et les recommandations ne doivent pas être appliquées sans avoir testé et vérifié qu'elles fonctionnent dans vos clusters.

Comprendre les dimensions d'échelle

L'évolutivité est différente des performances et de la fiabilité, et ces trois éléments doivent être pris en compte lors de la planification de vos besoins en matière de cluster et de charge de travail. À mesure que les clusters évoluent, ils doivent être surveillés, mais ce guide ne traitera pas des meilleures pratiques en matière de surveillance. EKS peut s'adapter à de grandes tailles, mais vous devez planifier la manière dont vous allez faire évoluer un cluster au-delà de 300 nœuds ou 5 000 pods. Il ne s'agit pas de chiffres absolus, mais ils proviennent de la collaboration de ce guide avec plusieurs utilisateurs, ingénieurs et professionnels du support.

La mise à l'échelle dans Kubernetes est multidimensionnelle et il n'existe pas de paramètres ou de recommandations spécifiques adaptés à toutes les situations. Les principaux domaines dans lesquels nous pouvons fournir des conseils en matière de mise à l'échelle sont les suivants :

Le plan de contrôle Kubernetes dans un cluster EKS inclut tous les services qu'AWS exécute et fait évoluer automatiquement pour vous (par exemple, le serveur d'API Kubernetes). La mise à l'échelle du plan de contrôle relève de la responsabilité d'AWS, mais l'utilisation responsable du plan de contrôle relève de votre responsabilité.

Le dimensionnement du plan de données Kubernetes gère les ressources AWS requises pour votre cluster et vos charges de travail, mais elles ne font pas partie du plan de contrôle EKS. Les ressources, notamment les instances EC2, le kubelet et le stockage, doivent toutes être mises à l'échelle en fonction de l'évolution de votre cluster.

Les services de cluster sont des contrôleurs et des applications Kubernetes qui s'exécutent au sein du cluster et fournissent des fonctionnalités à votre cluster et à vos charges de travail. Il peut s'agir de modules complémentaires EKS ainsi que d'autres services ou de cartes Helm que vous installez à des fins de conformité et d'intégration. Ces services dépendent souvent des charges de travail et, à mesure que vos charges de travail augmentent, vos services de cluster devront évoluer avec elles.

Les charges de travail sont la raison pour laquelle vous disposez d'un cluster et doivent évoluer horizontalement avec le cluster. Certaines intégrations et certains paramètres des charges de travail dans Kubernetes peuvent aider le cluster à évoluer. Les abstractions Kubernetes, telles que les espaces de noms et les services, présentent également des considérations architecturales.

Très grande mise à l'échelle

Si vous souhaitez étendre un seul cluster au-delà de 1 000 nœuds ou 50 000 pods, nous serions ravis de discuter avec vous. Nous vous recommandons de contacter votre équipe d'assistance ou votre responsable de compte technique pour entrer en contact avec des spécialistes qui peuvent vous aider à planifier et à évoluer au-delà des informations fournies dans ce guide. Amazon EKS peut prendre en charge jusqu'à 100 000 nœuds dans un seul cluster si vous êtes sélectionné pour l'intégration.

Plan de contrôle Kubernetes

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Le plan de contrôle Kubernetes comprend le serveur d'API Kubernetes, le Kubernetes Controller Manager, le planificateur et d'autres composants nécessaires au fonctionnement de Kubernetes. Les limites d'évolutivité de ces composants varient en fonction de ce que vous exécutez dans le cluster, mais les domaines ayant le plus d'impact sur le dimensionnement sont la version de Kubernetes, l'utilisation et le dimensionnement des nœuds individuels.

Limitez la charge de travail et l'éclatement des nœuds

> **Important**
>
> Pour éviter d'atteindre les limites d'API sur le plan de contrôle, vous devez limiter les pics de dimensionnement qui augmentent la taille du cluster de pourcentages à deux chiffres à la fois (par exemple, de 1 000 nœuds à 1 100 nœuds ou de 4 000 à 4 500 pods à la fois).

Le plan de contrôle EKS s'adapte automatiquement à la croissance de votre cluster, mais sa rapidité est limitée. Lorsque vous créez un cluster EKS pour la première fois, le plan de contrôle ne pourra pas immédiatement s'adapter à des centaines de nœuds ou à des milliers de pods. Pour en savoir plus sur les améliorations apportées par EKS à la mise à l'échelle, consultez ce billet de blog.

La mise à l'échelle d'applications de grande envergure nécessite l'adaptation de l'infrastructure pour être totalement prête (par exemple, des équilibreurs de charge de chauffage). Pour contrôler la vitesse de mise à l'échelle, assurez-vous que vous effectuez la mise à l'échelle en fonction des indicateurs adaptés à votre application. La mise à l'échelle du processeur et de la mémoire peut ne pas prédire avec précision les contraintes de votre application et l'utilisation de métriques personnalisées (par exemple, les demandes par seconde) dans Kubernetes Horizontal Pod Autoscaler (HPA) peut être une meilleure option de dimensionnement.

Pour utiliser une métrique personnalisée, consultez les exemples de la documentation Kubernetes. Si vous avez des besoins de dimensionnement plus avancés ou si vous devez effectuer une mise

à l'échelle en fonction de sources externes (par exemple, la file d'attente AWS SQS), utilisez KEDA pour le dimensionnement de la charge de travail basé sur les événements.

Diminuez les nœuds et les pods en toute sécurité

Remplacez les instances de longue durée

Le remplacement régulier des nœuds permet de préserver la santé de votre cluster en évitant les dérives de configuration et les problèmes qui ne surviennent qu'après une disponibilité prolongée (par exemple, des fuites de mémoire lentes). Le remplacement automatique vous fournira de bons processus et de bonnes pratiques pour les mises à niveau des nœuds et les correctifs de sécurité. Si chaque nœud de votre cluster est remplacé régulièrement, il est moins difficile de maintenir des processus distincts pour une maintenance continue.

Utilisez les paramètres TTL (time to live) de Karpenter pour remplacer les instances après leur exécution pendant un certain temps. Les groupes de nœuds autogérés peuvent utiliser ce `max-instance-lifetime` paramètre pour faire tourner les nœuds automatiquement. Les groupes de nœuds gérés ne disposent pas actuellement de cette fonctionnalité, mais vous pouvez suivre la demande ici GitHub.

Supprimer les nœuds sous-utilisés

Vous pouvez supprimer des nœuds lorsqu'aucune charge de travail n'est en cours d'exécution en utilisant le seuil de réduction dans le Kubernetes Cluster Autoscaler avec le paramètre `--scale-down-utilization-threshold`ou dans Karpenter, vous pouvez utiliser le paramètre provisionneur. `ttlSecondsAfterEmpty`

Utilisez les budgets d'interruption de service et la fermeture sécurisée des nœuds

La suppression des pods et des nœuds d'un cluster Kubernetes nécessite que les contrôleurs mettent à jour plusieurs ressources (par exemple). EndpointSlices Si vous le faites fréquemment ou trop rapidement, vous risquez de ralentir le serveur d'API et d'interrompre les applications lorsque les modifications se propagent aux contrôleurs. Les budgets d'interruption des modules constituent une bonne pratique pour ralentir le taux de désabonnement afin de protéger la disponibilité de la charge de travail lorsque des nœuds sont supprimés ou replanifiés dans un cluster.

Utiliser le cache côté client lors de l'exécution de Kubectl

L'utilisation inefficace de la commande kubectl peut ajouter une charge supplémentaire au serveur d'API Kubernetes. Vous devez éviter d'exécuter des scripts ou des automatismes qui utilisent kubectl

de manière répétée (par exemple dans une boucle for) ou d'exécuter des commandes sans cache local.

`kubectl`possède un cache côté client qui met en cache les informations de découverte provenant du cluster afin de réduire le nombre d'appels d'API requis. Le cache est activé par défaut et est actualisé toutes les 10 minutes.

Si vous exécutez kubectl à partir d'un conteneur ou sans cache côté client, vous risquez de rencontrer des problèmes de limitation de l'API. Il est recommandé de conserver le cache de votre cluster en le montant afin `--cache-dir` d'éviter d'effectuer des appels d'API inutiles.

Désactiver la compression kubectl

La désactivation de la compression kubectl dans votre fichier kubeconfig peut réduire l'utilisation de l'API et du processeur client. Par défaut, le serveur compresse les données envoyées au client afin d'optimiser la bande passante du réseau. Cela augmente la charge du processeur sur le client et le serveur pour chaque demande et la désactivation de la compression peut réduire le surdébit et la latence si vous disposez d'une bande passante suffisante. Pour désactiver la compression, vous pouvez utiliser le `--disable-compression=true` drapeau ou le définir `disable-compression: true` dans votre fichier kubeconfig.

```
apiVersion: v1
clusters:
- cluster:
    server: serverURL
    disable-compression: true
  name: cluster
```

Autoscaler Shard Cluster

Le Kubernetes Cluster Autoscaler a été testé pour atteindre 1 000 nœuds. Sur un cluster de plus de 1 000 nœuds, il est recommandé d'exécuter plusieurs instances du Cluster Autoscaler en mode partition. Chaque instance de Cluster Autoscaler est configurée pour redimensionner un ensemble de groupes de nœuds. L'exemple suivant montre 2 configurations de mise à l'échelle automatique de cluster configurées pour chacune d'entre elles redimensionner 4 groupes de nœuds.

ClusterAutoscaler-1

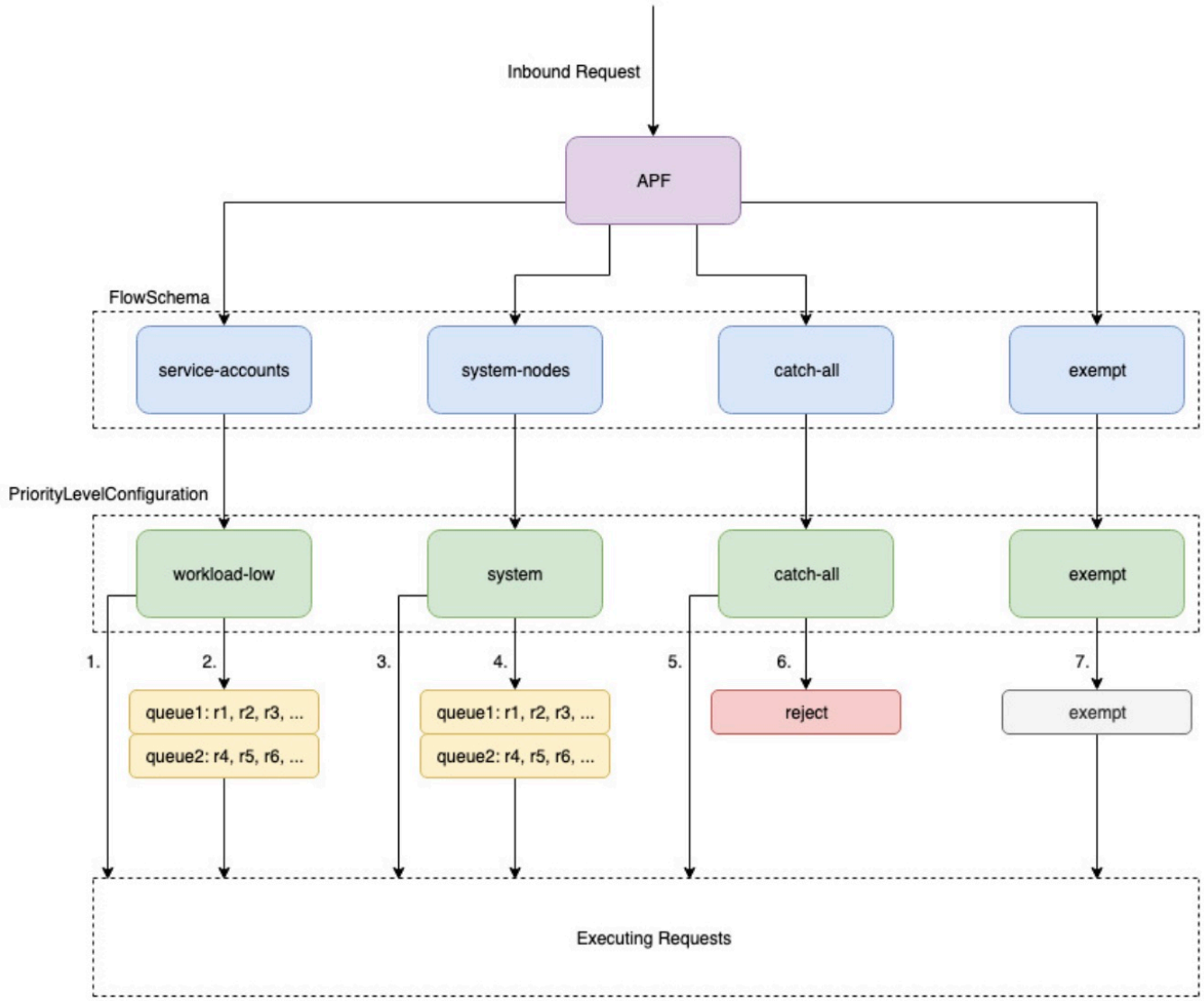
```
autoscalingGroups:
```

```
- name: eks-core-node-grp-20220823190924690000000011-80c1660e-030d-476d-cb0d-
d04d585a8fcb
  maxSize: 50
  minSize: 2
- name: eks-data_m1-20220824130553925600000011-5ec167fa-ca93-8ca4-53a5-003e1ed8d306
  maxSize: 450
  minSize: 2
- name: eks-data_m2-20220824130733258600000015-aac167fb-8bf7-429d-d032-e195af4e25f5
  maxSize: 450
  minSize: 2
- name: eks-data_m3-20220824130553914900000003-18c167fa-ca7f-23c9-0fea-f9edefbda002
  maxSize: 450
  minSize: 2
```

ClusterAutoscaler-2

```
autoscalingGroups:
- name: eks-data_m4-20220824130553925500000000f-5ec167fa-ca86-6b83-ae9d-1e07ade3e7c4
  maxSize: 450
  minSize: 2
- name: eks-data_m5-20220824130744542100000017-02c167fb-a1f7-3d9e-a583-43b4975c050c
  maxSize: 450
  minSize: 2
- name: eks-data_m6-20220824130553924300000000d-9cc167fa-ca94-132a-04ad-e43166cef41f
  maxSize: 450
  minSize: 2
- name: eks-data_m7-20220824130553921000000009-96c167fa-ca91-d767-0427-91c879ddf5af
  maxSize: 450
  minSize: 2
```

Priorité et équité des API



1. If workload-low has not reached its concurrency limit, the request will be dispatched immediately.
2. If workload-low has reached its concurrency limit, the request will be queued prior to dispatch.
3. If system has not reached its concurrency limit, the request will be dispatched immediately.
4. If system has reached its concurrency limit, the request will be queued prior to dispatch.
5. If catch-all has not reached its concurrency limit, the request will be dispatched immediately.
6. If catch-all has reached its concurrency limit, the request will be dropped.
7. Requests mapping to exempt will always be dispatched.

Présentation de

Pour éviter d'être surchargé pendant les périodes d'augmentation des demandes, le serveur API limite le nombre de demandes en cours de vol qu'il peut avoir en suspens à un moment donné. Une

fois cette limite dépassée, le serveur API commence à rejeter les demandes et renvoie aux clients un code de réponse HTTP 429 pour « trop de demandes ». Il est préférable que le serveur abandonne les demandes et demande aux clients de réessayer ultérieurement plutôt que de n'imposer aucune limite côté serveur quant au nombre de demandes et de surcharger le plan de contrôle, ce qui pourrait entraîner une dégradation des performances ou une indisponibilité.

Le mécanisme utilisé par Kubernetes pour configurer la façon dont ces demandes en vol sont réparties entre différents types de demandes est appelé API Priority and Fairness. Le serveur API configure le nombre total de demandes en vol qu'il peut accepter en additionnant les valeurs spécifiées par les `--max-requests-inflight` indicateurs et. `--max-mutating-requests-inflight` EKS utilise les valeurs par défaut de 400 et 200 demandes pour ces indicateurs, ce qui permet d'envoyer un total de 600 demandes à un moment donné. Cependant, à mesure qu'il agrandit le plan de contrôle en fonction de l'augmentation de l'utilisation et de la perte de charge de travail, il augmente en conséquence le quota de demandes en vol jusqu'en 2000 (sous réserve de modifications). L'APF indique comment ces quotas de demandes en vol sont ensuite subdivisés entre les différents types de demandes. Notez que les plans de contrôle EKS sont hautement disponibles avec au moins 2 serveurs API enregistrés pour chaque cluster. Cela signifie que le nombre total de demandes en vol que votre cluster peut traiter est le double (ou supérieur s'il est davantage redimensionné horizontalement) du quota en vol défini par kube-apiserver. Cela représente plusieurs milliers de requests/second clusters EKS parmi les plus importants.

Deux types d'objets Kubernetes, appelés PriorityLevelConfigurations et FlowSchemas, configurent la façon dont le nombre total de demandes est réparti entre les différents types de demandes. Ces objets sont gérés automatiquement par le serveur API et EKS utilise la configuration par défaut de ces objets pour la version mineure de Kubernetes donnée. PriorityLevelConfigurations représentent une fraction du nombre total de demandes autorisées. Par exemple, la charge de travail la plus élevée PriorityLevelConfiguration est allouée à 98 demandes sur un total de 600. La somme des demandes allouées à toutes PriorityLevelConfigurations sera égale à 600 (ou légèrement supérieure à 600 car le serveur d'API arrondira le chiffre si un niveau donné est accordé à une fraction d'une demande). Pour vérifier le PriorityLevelConfigurations contenu de votre cluster et le nombre de requêtes allouées à chacun, vous pouvez exécuter la commande suivante. Voici les valeurs par défaut d'EKS 1.32 :

```
$ kubectl get --raw /metrics | grep apiserver_flowcontrol_nominal_limit_seats
apiserver_flowcontrol_nominal_limit_seats{priority_level="catch-all"} 13
apiserver_flowcontrol_nominal_limit_seats{priority_level="exempt"} 0
apiserver_flowcontrol_nominal_limit_seats{priority_level="global-default"} 49
apiserver_flowcontrol_nominal_limit_seats{priority_level="leader-election"} 25
```

```
apiserver_flowcontrol_nominal_limit_seats{priority_level="node-high"} 98
apiserver_flowcontrol_nominal_limit_seats{priority_level="system"} 74
apiserver_flowcontrol_nominal_limit_seats{priority_level="workload-high"} 98
apiserver_flowcontrol_nominal_limit_seats{priority_level="workload-low"} 245
```

Le deuxième type d'objet est FlowSchemas. Les requêtes du serveur d'API avec un ensemble de propriétés donné sont classées dans la même catégorie FlowSchema. Ces propriétés incluent soit l'utilisateur authentifié, soit les attributs de la demande, tels que le groupe d'API, l'espace de noms ou la ressource. A indique FlowSchema également à quel type de demande PriorityLevelConfiguration ce type de demande doit être mappé. Ensemble, les deux objets indiquent : « Je veux que ce type de demande soit pris en compte dans cette part des demandes en vol. » Lorsqu'une requête atteint le serveur API, celui-ci vérifie chacune de ses requêtes FlowSchemas jusqu'à ce qu'il en trouve une qui correspond à toutes les propriétés requises. Si plusieurs FlowSchemas correspondent à une requête, le serveur API choisira celle qui a la plus petite priorité correspondante spécifiée en tant que propriété dans l'objet. FlowSchema

Le mappage de FlowSchemas vers PriorityLevelConfigurations peut être visualisé à l'aide de cette commande :

```
$ kubectl get flowschemas
NAME                          PRIORITYLEVEL     MATCHINGPRECEDENCE
 DISTINGUISHERMETHOD   AGE     MISSINGPL
exempt                        exempt            1                   <none>
     7h19m   False
eks-exempt                    exempt            2                   <none>
     7h19m   False
probes                        exempt            2                   <none>
     7h19m   False
system-leader-election        leader-election   100                 ByUser
     7h19m   False
endpoint-controller           workload-high     150                 ByUser
     7h19m   False
workload-leader-election      leader-election   200                 ByUser
     7h19m   False
system-node-high              node-high         400                 ByUser
     7h19m   False
system-nodes                  system            500                 ByUser
     7h19m   False
kube-controller-manager       workload-high     800                 ByNamespace
     7h19m   False
kube-scheduler                workload-high     800                 ByNamespace
     7h19m   False
```

```
kube-system-service-accounts   workload-high     900                 ByNamespace
     7h19m   False
eks-workload-high              workload-high     1000                ByUser
     7h14m   False
service-accounts               workload-low      9000                ByUser
     7h19m   False
global-default                 global-default    9900                ByUser
     7h19m   False
catch-all                      catch-all         10000               ByUser
     7h19m   False
```

PriorityLevelConfigurations peut avoir un type de file d'attente, de rejet ou d'exemption. Pour les types Queue et Reject, une limite est imposée au nombre maximum de demandes en vol pour ce niveau de priorité, mais le comportement diffère lorsque cette limite est atteinte. Par exemple, le workload-high PriorityLevelConfiguration utilise le type Queue et dispose de 98 requêtes disponibles pour être utilisées par le controller-manager, le endpoint-controller, le planificateur, les contrôleurs associés à eks et depuis des pods exécutés dans l'espace de noms kube-system. Comme le type Queue est utilisé, le serveur API essaiera de conserver les demandes en mémoire en espérant que le nombre de demandes en vol tombe en dessous de 98 avant l'expiration de ces demandes. Si le délai d'attente d'une demande est dépassé ou si trop de demandes sont déjà en attente, le serveur API n'a pas d'autre choix que de supprimer la demande et de renvoyer au client un 429. Notez que la mise en file d'attente peut empêcher une demande de recevoir un 429, mais cela implique un compromis en termes de end-to-end latence accrue sur la demande.

Considérons maintenant le fourre-tout FlowSchema qui correspond au fourre-tout avec le type Reject PriorityLevelConfiguration . Si les clients atteignent la limite de 13 demandes en vol, le serveur API n'effectuera pas de mise en file d'attente et supprimera les demandes instantanément avec un code de réponse 429. Enfin, les demandes correspondant PriorityLevelConfiguration à un type Exempt ne recevront jamais de 429 et seront toujours expédiées immédiatement. Ceci est utilisé pour les demandes hautement prioritaires telles que les requêtes healthz ou les demandes provenant du groupe system:masters.

Surveillance des demandes APF et abandonnées

Pour confirmer si des demandes sont abandonnées à cause de l'APF, les métriques du serveur d'API `apiserver_flowcontrol_rejected_requests_total` peuvent être surveillées afin de vérifier les informations concernées FlowSchemas et PriorityLevelConfigurations. Par exemple, cet indicateur montre que 100 demandes provenant des comptes de service FlowSchema ont été abandonnées en raison de l'expiration des délais dans les files d'attente où la charge de travail est faible :

```
% kubectl get --raw /metrics | grep apiserver_flowcontrol_rejected_requests_total
apiserver_flowcontrol_rejected_requests_total{flow_schema="service-
accounts",priority_level="workload-low",reason="time-out"} 100
```

Pour vérifier dans quelle mesure une donnée PriorityLevelConfiguration est proche de la réception de 429 secondes ou de l'augmentation de la latence due à une file d'attente, vous pouvez comparer la différence entre la limite de simultanéité et la simultanéité utilisée. Dans cet exemple, nous avons une mémoire tampon de 100 demandes.

```
% kubectl get --raw /metrics | grep
 'apiserver_flowcontrol_nominal_limit_seats.*workload-low'
apiserver_flowcontrol_nominal_limit_seats{priority_level="workload-low"} 245

% kubectl get --raw /metrics | grep
 'apiserver_flowcontrol_current_executing_seats.*workload-low'
apiserver_flowcontrol_current_executing_seats{flow_schema="service-
accounts",priority_level="workload-low"} 145
```

Pour vérifier si une demande PriorityLevelConfiguration est mise en file d'attente mais pas nécessairement abandonnée, la métrique de `apiserver_flowcontrol_current_inqueue_requests` peut être référencée :

```
% kubectl get --raw /metrics | grep
 'apiserver_flowcontrol_current_inqueue_requests.*workload-low'
apiserver_flowcontrol_current_inqueue_requests{flow_schema="service-
accounts",priority_level="workload-low"} 10
```

Parmi les autres indicateurs utiles de Prometheus, citons :

- apiserver_flowcontrol_dispatched_requests_total
- apiserver_flowcontrol_request_execution_seconds
- apiserver_flowcontrol_request_wait_duration_seconds

Consultez la documentation en amont pour obtenir la liste complète des métriques APF.

Empêcher les demandes abandonnées

Évitez les 429 en modifiant votre charge de travail

Lorsque l'APF abandonne des demandes en raison d'un PriorityLevelConfiguration dépassement du nombre maximum de demandes autorisées en vol, les clients concernés FlowSchemas peuvent réduire le nombre de demandes exécutées à un moment donné. Cela peut être accompli en réduisant le nombre total de demandes effectuées au cours de la période au cours de laquelle 429 demandes se produisent. Notez que les demandes de longue durée, telles que les appels de liste coûteux, sont particulièrement problématiques car elles sont considérées comme des demandes en vol pendant toute la durée de leur exécution. La réduction du nombre de ces demandes coûteuses ou l'optimisation de la latence de ces appels de liste (par exemple, en réduisant le nombre d'objets récupérés par demande ou en passant à une demande de surveillance) peut contribuer à réduire la simultanéité totale requise par la charge de travail donnée.

Évitez les 429 en modifiant vos paramètres APF

> **Warning**
>
> Ne modifiez les paramètres APF par défaut que si vous savez ce que vous faites. Des paramètres APF mal configurés peuvent entraîner l'abandon des demandes du serveur d'API et des perturbations importantes de la charge de travail.

Une autre approche pour empêcher les demandes abandonnées consiste à modifier la valeur par défaut FlowSchemas ou à l' PriorityLevelConfigurations installer sur les clusters EKS. EKS installe les paramètres par défaut en amont pour FlowSchemas et PriorityLevelConfigurations pour la version mineure de Kubernetes donnée. Le serveur API rétablit automatiquement ces objets avec leurs valeurs par défaut en cas de modification, sauf si l'annotation suivante sur les objets est définie sur false :

```
metadata:
  annotations:
    apf.kubernetes.io/autoupdate-spec: "false"
```

À un niveau élevé, les paramètres APF peuvent être modifiés pour :

- Allouez davantage de capacité en vol aux demandes qui vous intéressent.

- Isolez les demandes non essentielles ou coûteuses susceptibles de réduire la capacité pour d'autres types de demandes.

Cela peut être accompli soit en modifiant la valeur par défaut FlowSchemas , PriorityLevelConfigurations soit en créant de nouveaux objets de ce type. Les opérateurs peuvent augmenter les valeurs assuredConcurrencyShares des PriorityLevelConfigurations objets concernés afin d'augmenter le pourcentage de demandes en vol qui leur sont allouées. En outre, le nombre de demandes pouvant être mises en file d'attente à un moment donné peut également être augmenté si l'application peut gérer la latence supplémentaire causée par le fait que les demandes sont mises en file d'attente avant leur expédition.

Il est également possible FlowSchema de créer de nouveaux PriorityLevelConfigurations objets spécifiques à la charge de travail du client. Sachez qu'en allouant davantage assuredConcurrencyShares à des applications existantes PriorityLevelConfigurations ou nouvelles PriorityLevelConfigurations , le nombre de demandes pouvant être traitées par d'autres compartiments sera réduit, car la limite globale restera fixée à 600 en vol par serveur d'API.

Lorsque vous modifiez les paramètres par défaut de l'APF, ces métriques doivent être surveillées sur un cluster hors production afin de garantir que la modification des paramètres ne provoque pas de 429 indésirables :

1. La métrique de `apiserver_flowcontrol_rejected_requests_total` doit être surveillée pour tous FlowSchemas afin de s'assurer qu'aucun compartiment ne commence à supprimer des demandes.
2. Les valeurs pour `apiserver_flowcontrol_nominal_limit_seats` et `apiserver_flowcontrol_current_executing_seats` doivent être comparées afin de garantir que la simultanéité d'utilisation ne risque pas de dépasser la limite pour ce niveau de priorité.

Un cas d'utilisation courant pour définir une nouvelle FlowSchema et PriorityLevelConfiguration est l'isolation. Supposons que nous voulions isoler les appels d'événements de liste de longue durée provenant des pods pour leur propre part de demandes. Cela évitera que les demandes importantes émanant de pods utilisant les comptes de service existants ne FlowSchema reçoivent 429 demandes et ne soient privées de capacité de demande. Rappelez-vous que le nombre total de demandes en vol est limité. Toutefois, cet exemple montre que les paramètres APF peuvent être modifiés afin de mieux répartir la capacité des demandes pour une charge de travail donnée :

Exemple FlowSchema d'objet pour isoler les demandes d'événements de liste :

```
apiVersion: flowcontrol.apiserver.k8s.io/v1
kind: FlowSchema
metadata:
  name: list-events-default-service-accounts
spec:
  distinguisherMethod:
    type: ByUser
  matchingPrecedence: 8000
  priorityLevelConfiguration:
    name: catch-all
  rules:
  - resourceRules:
    - apiGroups:
      - '*'
      namespaces:
      - default
      resources:
      - events
      verbs:
      - list
    subjects:
    - kind: ServiceAccount
      serviceAccount:
        name: default
        namespace: default
```

- Cela FlowSchema capture tous les appels d'événements de liste effectués par les comptes de service dans l'espace de noms par défaut.
- La priorité correspondante de 8000 est inférieure à la valeur de 9000 utilisée par les comptes de service existants, de FlowSchema sorte que ces appels d'événements de liste correspondront à list-events-default-service -accounts plutôt qu'à des comptes de service.
- Nous utilisons le fourre-tout PriorityLevelConfiguration pour isoler ces demandes. Ce compartiment autorise uniquement l'utilisation de 13 demandes en vol par ces appels d'événements de longue durée. Les pods commenceront à recevoir 429 dès qu'ils essaieront d'émettre plus de 13 de ces demandes simultanément.

Récupération de ressources dans le serveur d'API

L'obtention d'informations depuis le serveur d'API est un comportement attendu pour les clusters de toutes tailles. À mesure que vous augmentez le nombre de ressources dans le cluster, la fréquence des demandes et le volume de données peuvent rapidement devenir un obstacle pour le plan de contrôle et entraîner une latence et un ralentissement des API. En fonction de la gravité de la latence, cela entraîne des temps d'arrêt inattendus si vous ne faites pas attention.

Savoir ce que vous demandez et à quelle fréquence sont les premières étapes pour éviter ce type de problèmes. Voici des conseils pour limiter le volume de requêtes en fonction des meilleures pratiques de dimensionnement. Les suggestions de cette section sont fournies dans l'ordre, en commençant par les options connues pour être les plus évolutives.

Utiliser des informateurs partagés

Lorsque vous créez des contrôleurs et des systèmes d'automatisation qui s'intègrent à l'API Kubernetes, vous devez souvent obtenir des informations à partir des ressources Kubernetes. Si vous interrogez régulièrement ces ressources, cela peut entraîner une charge importante sur le serveur d'API.

L'utilisation d'un informateur de la bibliothèque client-go vous permettra de suivre les modifications apportées aux ressources en fonction des événements au lieu de les interroger. Les informateurs réduisent encore la charge en utilisant un cache partagé pour les événements et les modifications, de sorte que plusieurs contrôleurs surveillant les mêmes ressources n'ajoutent pas de charge supplémentaire.

Les contrôleurs doivent éviter d'interroger les ressources à l'échelle du cluster sans étiquettes ni sélecteurs de champs, en particulier dans les grands clusters. Chaque sondage non filtré nécessite l'envoi d'un grand nombre de données inutiles depuis etcd via le serveur API pour être filtrées par le client. En filtrant en fonction des libellés et des espaces de noms, vous pouvez réduire la quantité de travail que le serveur d'API doit effectuer pour répondre aux demandes et aux données envoyées au client.

Optimisation de l'utilisation de l'API Kubernetes

Lorsque vous appelez l'API Kubernetes avec des contrôleurs personnalisés ou une automatisation, il est important de limiter les appels aux seules ressources dont vous avez besoin. Sans limites, vous pouvez entraîner une charge inutile sur le serveur d'API, etc.

Il est recommandé d'utiliser l'argument watch dans la mesure du possible. En l'absence d'arguments, le comportement par défaut consiste à répertorier les objets. Pour utiliser watch au lieu de list, vous pouvez l'ajouter `?watch=true` à la fin de votre demande d'API. Par exemple, pour placer tous les pods dans l'espace de noms par défaut avec une montre, utilisez :

```
/api/v1/namespaces/default/pods?watch=true
```

Si vous mettez en vente des objets, vous devez limiter la portée de ce que vous mettez en vente et la quantité de données renvoyées. Vous pouvez limiter les données renvoyées en ajoutant des `limit=500` arguments aux demandes. L'`fieldSelector`argument et le `/namespace/` chemin peuvent être utiles pour s'assurer que vos listes ont une portée aussi étroite que nécessaire. Par exemple, pour répertorier uniquement les pods en cours d'exécution dans l'espace de noms par défaut, utilisez le chemin d'API et les arguments suivants.

```
/api/v1/namespaces/default/pods?fieldSelector=status.phase=Running&limit=500
```

Ou listez tous les pods qui fonctionnent avec :

```
/api/v1/pods?fieldSelector=status.phase=Running&limit=500
```

Une autre option pour limiter les appels de surveillance ou les objets listés consiste à utiliser une option `resourceVersionsque vous pouvez consulter dans la documentation de Kubernetes`. Sans `resourceVersion` argument, vous recevrez la version la plus récente disponible qui nécessite une lecture par quorum etcd, qui est la lecture la plus coûteuse et la plus lente de la base de données. La ResourceVersion dépend des ressources que vous essayez d'interroger et qui se trouvent dans le `metadata.resourseVersion` champ. Ceci est également recommandé en cas d'utilisation de Watch Calls et pas seulement de listes d'appels

Il existe une `resourceVersion=0` option spéciale qui renverra les résultats depuis le cache du serveur API. Cela peut réduire la charge de l'etcd mais ne supporte pas la pagination.

```
/api/v1/namespaces/default/pods?resourceVersion=0
```

Il est recommandé d'utiliser watch avec une ResourceVersion définie comme étant la valeur connue la plus récente reçue de la liste ou de la surveillance précédente. Ceci est géré automatiquement dans client-go. Mais il est suggéré de le vérifier si vous utilisez un client k8s dans d'autres langues.

```
/api/v1/namespaces/default/pods?watch=true&resourceVersion=362812295
```

Si vous appelez l'API sans aucun argument, cela consommera le plus de ressources pour le serveur d'API et etc. Cet appel permettra d'obtenir tous les pods dans tous les espaces de noms sans pagination ni limitation de portée et nécessitera une lecture du quorum depuis etcd.

```
/api/v1/pods
```

Empêchez le DaemonSet tonnerre des troupeaux

A DaemonSet garantit que tous (ou certains) nœuds exécutent une copie d'un pod. Lorsque des nœuds rejoignent le cluster, le daemonset-controller crée des pods pour ces nœuds. Lorsque les nœuds quittent le cluster, ces pods sont collectés à la poubelle. La suppression d'un DaemonSet va nettoyer les pods qu'il a créés.

Voici quelques utilisations typiques de DaemonSet a :

- Exécution d'un démon de stockage en cluster sur chaque nœud
- Exécution d'un démon de collecte de logs sur chaque nœud
- Exécution d'un démon de surveillance des nœuds sur chaque nœud

Sur les clusters comportant des milliers de nœuds, la création d'un nouveau DaemonSet nœud DaemonSet, sa mise à jour ou l'augmentation du nombre de nœuds peuvent entraîner une charge élevée sur le plan de contrôle. Si DaemonSet les pods émettent des requêtes coûteuses au serveur d'API au démarrage du pod, ils peuvent entraîner une utilisation importante des ressources sur le plan de contrôle en raison d'un grand nombre de demandes simultanées.

En fonctionnement normal, vous pouvez utiliser un `RollingUpdate` pour assurer le déploiement progressif des nouveaux DaemonSet modules. Dans le cadre d'une stratégie de `RollingUpdate` mise à jour, une fois que vous avez mis à jour un DaemonSet modèle, le contrôleur supprime les anciens DaemonSet DaemonSet modules et en crée de nouveaux automatiquement de manière contrôlée. Au maximum, un pod DaemonSet sera exécuté sur chaque nœud pendant tout le processus de mise à jour. Vous pouvez effectuer un déploiement progressif en `maxUnavailable` réglant sur 1, `maxSurge` 0 et `minReadySeconds` 60. Si vous ne spécifiez pas de stratégie de mise à jour, Kubernetes créera par défaut un `RollingUpdate` avec `maxUnavailable` 1, `maxSurge` 0 et `minReadySeconds` 0.

```
minReadySeconds: 60
strategy:
```

```
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 0
    maxUnavailable: 1
```

A `RollingUpdate` garantit le déploiement progressif de nouveaux DaemonSet pods s'ils ont déjà été créés et s'ils contiennent le nombre attendu de `Ready` pods sur tous les nœuds. DaemonSet Des problèmes graves liés aux troupeaux peuvent survenir dans certaines conditions qui ne sont pas couvertes par les `RollingUpdate` stratégies.

Empêchez le tonnerre des troupeaux lors de la création DaemonSet

Par défaut, quelle que soit la `RollingUpdate` configuration, le daemonset-controller du kube-controller-manager créera des pods pour tous les nœuds correspondants simultanément lorsque vous en créerez un nouveau. DaemonSet Pour forcer le déploiement progressif des modules après avoir créé un DaemonSet, vous pouvez utiliser un `NodeSelector` ou`NodeAffinity`. Cela créera un DaemonSet nœud correspondant à zéro nœud, puis vous pourrez progressivement mettre à jour les nœuds pour les rendre éligibles à l'exécution d'un pod à partir du DaemonSet à un rythme contrôlé. Vous pouvez suivre cette approche :

- Ajoutez une étiquette à tous les nœuds pour`run-daemonset=false`.

```
kubectl label nodes --all run-daemonset=false
```

- Créez votre DaemonSet avec un `NodeAffinity` paramètre correspondant à n'importe quel nœud sans `run-daemonset=false` étiquette. Dans un premier temps, cela se traduira par DaemonSet l'absence de pods correspondants.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: run-daemonset
          operator: NotIn
          values:
          - "false"
```

- Retirez l'`run-daemonset=false`étiquette de vos nœuds à un rythme contrôlé. Vous pouvez utiliser ce script bash comme exemple :

```
#!/bin/bash

nodes=$(kubectl get --raw "/api/v1/nodes" | jq -r '.items | .[].metadata.name')

for node in ${nodes[@]}; do
   echo "Removing run-daemonset label from node $node"
   kubectl label nodes $node run-daemonset-
   sleep 5
done
```

- Vous pouvez éventuellement supprimer le `NodeAffinity` paramètre de votre DaemonSet objet. Notez que cela déclenchera également un `RollingUpdate` et remplacera progressivement tous les DaemonSet pods existants car le DaemonSet modèle a changé.

Empêchez le tonnerre des troupeaux lors de la mise à l'échelle des nœuds

À l'instar de DaemonSet la création, la création rapide de nouveaux nœuds peut entraîner le démarrage simultané d'un grand nombre de DaemonSet pods. Vous devez créer de nouveaux nœuds à un rythme contrôlé afin que le contrôleur crée DaemonSet des pods au même rythme. Si cela n'est pas possible, vous pouvez rendre les nouveaux nœuds initialement inéligibles à l'existant DaemonSet en utilisant`NodeAffinity`. Ensuite, vous pouvez ajouter progressivement une étiquette aux nouveaux nœuds afin que le daemonset-controller crée des pods à un rythme contrôlé. Vous pouvez suivre cette approche :

- Ajoutez une étiquette à tous les nœuds existants pour `run-daemonset=true`

```
kubectl label nodes --all run-daemonset=true
```

- Mettez à jour votre DaemonSet compte avec un `NodeAffinity` paramètre correspondant à n'importe quel nœud doté d'une `run-daemonset=true` étiquette. Notez que cela déclenchera également un `RollingUpdate` et remplacera progressivement tous les DaemonSet pods existants car le DaemonSet modèle a changé. Vous devez attendre la fin de `RollingUpdate` l'opération avant de passer à l'étape suivante.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: run-daemonset
          operator: In
          values:
          - "true"
```

- Créez de nouveaux nœuds dans votre cluster. Notez que ces nœuds n'auront pas d'`run-daemonset=true`étiquette et ne DaemonSet correspondront donc pas à ces nœuds.
- Ajoutez l'`run-daemonset=true`étiquette à vos nouveaux nœuds (qui n'en ont pas actuellement) à un rythme contrôlé. `run-daemonset` Vous pouvez utiliser ce script bash comme exemple :

```
#!/bin/bash

nodes=$(kubectl get --raw "/api/v1/nodes?labelSelector=%21run-daemonset" | jq -r
 '.items | .[].metadata.name')

for node in ${nodes[@]}; do
   echo "Adding run-daemonset=true label to node $node"
   kubectl label nodes $node run-daemonset=true
   sleep 5
done
```

- Vous pouvez éventuellement supprimer le `NodeAffinity` paramètre de votre DaemonSet objet et supprimer l'`run-daemonset`étiquette de tous les nœuds.

Évitez les foules lors des mises à jour DaemonSet

Une `RollingUpdate` politique ne respectera le `maxUnavailable` paramètre que pour les DaemonSet pods qui le sont`Ready`. Si DaemonSet a uniquement `NotReady` des pods ou un pourcentage élevé de `NotReady` pods et que vous mettez à jour son modèle, le daemonset-controller créera de nouveaux pods simultanément pour tous les pods. `NotReady` Cela peut entraîner de graves problèmes de troupeau s'il y a un nombre important de `NotReady` capsules, par exemple si les capsules tournent continuellement en boucle ou ne parviennent pas à extraire des images.

Pour forcer le déploiement progressif des modules lorsque vous mettez à jour un module DaemonSet et qu'il existe `NotReady` des modules, vous pouvez modifier temporairement la stratégie de mise à jour DaemonSet du formulaire `RollingUpdate` au`OnDelete`. Après avoir mis à jour un DaemonSet modèle, le contrôleur crée de nouveaux modules après avoir supprimé manuellement les anciens afin que vous puissiez contrôler le déploiement des nouveaux modules. `OnDelete` Vous pouvez suivre cette approche :

- Vérifiez si vous avez `NotReady` des capsules dans votre DaemonSet.
- Si ce n'est pas le cas, vous pouvez mettre à jour le DaemonSet modèle en toute sécurité et la `RollingUpdate` stratégie garantira un déploiement progressif.
- Si c'est le cas, vous devez d'abord mettre à jour votre stratégie DaemonSet pour pouvoir utiliser `OnDelete` cette stratégie.

```
updateStrategy:
  type: OnDelete
```

- Ensuite, mettez à jour votre DaemonSet modèle en y apportant les modifications nécessaires.
- Après cette mise à jour, vous pouvez supprimer les anciens DaemonSet modules en émettant des demandes de suppression de modules à un rythme contrôlé. Vous pouvez utiliser ce script bash comme exemple où le DaemonSet nom est fluentd-elasticsearch dans l'espace de noms kube-system :

```
#!/bin/bash

daemonset_pods=$(kubectl get --raw "/api/v1/namespaces/kube-system/pods?
labelSelector=name%3Dfluentd-elasticsearch" | jq -r '.items | .[].metadata.name')

for pod in ${daemonset_pods[@]}; do
   echo "Deleting pod $pod"
   kubectl delete pod $pod -n kube-system
   sleep 5
done
```

- Enfin, vous pouvez DaemonSet revenir à la `RollingUpdate` stratégie précédente.

Plan de données Kubernetes

La sélection des types d'instances EC2 est probablement l'une des décisions les plus difficiles auxquelles les clients sont confrontés, car il s'agit de clusters comportant de multiples charges de travail. Il n'y a pas one-size-fits de solution complète. Voici quelques conseils qui vous aideront à éviter les écueils courants liés à la mise à l'échelle du calcul.

Mise à l'échelle automatique des nœuds

Nous vous recommandons d'utiliser la mise à l'échelle automatique des nœuds qui réduit la charge de travail et s'intègre parfaitement à Kubernetes. Les groupes de nœuds gérés et Karpenter sont recommandés pour les clusters à grande échelle.

Les groupes de nœuds gérés vous offrent la flexibilité des groupes Amazon EC2 Auto Scaling avec des avantages supplémentaires pour les mises à niveau et la configuration gérées. Il peut être mis à l'échelle à l'aide du Kubernetes Cluster Autoscaler et constitue une option courante pour les clusters ayant des besoins informatiques variés.

Karpenter est un autoscaler de nœuds open source, natif de la charge de travail, créé par AWS. Il fait évoluer les nœuds d'un cluster en fonction des exigences de charge de travail en matière de ressources (par exemple, GPU) et des contraintes et des tolérances (par exemple, la répartition des zones) sans gérer les groupes de nœuds. Les nœuds sont créés directement à partir d'EC2, ce qui permet d'éviter les quotas de groupes de nœuds par défaut (450 nœuds par groupe) et d'offrir une plus grande flexibilité de sélection d'instances tout en réduisant les frais opérationnels. Nous recommandons aux clients d'utiliser Karpenter dans la mesure du possible.

Utilisez de nombreux types d'instances EC2

Chaque région AWS dispose d'un nombre limité d'instances disponibles par type d'instance. Si vous créez un cluster qui n'utilise qu'un seul type d'instance et que vous augmentez le nombre de nœuds au-delà de la capacité de la région, vous recevrez un message d'erreur indiquant qu'aucune instance n'est disponible. Pour éviter ce problème, vous ne devez pas limiter arbitrairement le type d'instances pouvant être utilisées dans votre cluster.

Karpenter utilisera par défaut un large éventail de types d'instances compatibles et choisira une instance au moment du provisionnement en fonction des exigences en matière de charge de travail, de disponibilité et de coût. Vous pouvez élargir la liste des types d'instances utilisés dans la `karpenter.k8s.aws/instance-category` clé de NodePools.

Le Kubernetes Cluster Autoscaler nécessite que les groupes de nœuds soient de taille similaire afin qu'ils puissent être mis à l'échelle de manière cohérente. Vous devez créer plusieurs groupes en fonction de la taille du processeur et de la mémoire et les dimensionner indépendamment. Utilisez le sélecteur d'instance ec2 pour identifier les instances dont la taille est similaire à celle de vos groupes de nœuds.

```
ec2-instance-selector --service eks --vcpus-min 8 --memory-min 16
a1.2xlarge
a1.4xlarge
a1.metal
c4.4xlarge
c4.8xlarge
c5.12xlarge
c5.18xlarge
c5.24xlarge
c5.2xlarge
c5.4xlarge
c5.9xlarge
c5.metal
```

Préférez des nœuds plus grands pour réduire la charge du serveur d'API

Lorsque vous décidez des types d'instances à utiliser, le nombre réduit de nœuds volumineux alourdira le plan de contrôle Kubernetes, car il y aura moins de kubelets et moins de nœuds en cours d'exécution. DaemonSets Cependant, les nœuds de grande taille peuvent ne pas être pleinement utilisés comme les nœuds plus petits. La taille des nœuds doit être évaluée en fonction de la disponibilité de votre charge de travail et de vos exigences d'échelle.

Un cluster avec trois instances u-24tb1.metal (24 To de mémoire et 448 cœurs) possède 3 kubelets et serait limité à 110 pods par nœud par défaut. Si vos pods utilisent 4 cœurs chacun, cela peut être attendu (4 cœurs x 110 = 440 cœurs/nœud). Avec un cluster à 3 nœuds, votre capacité à gérer un incident d'instance serait faible, car une panne d'instance pourrait avoir un impact sur un tiers du cluster. Vous devez spécifier les exigences relatives aux nœuds et la répartition des pods dans vos charges de travail afin que le planificateur Kubernetes puisse placer correctement les charges de travail.

Les charges de travail doivent définir les ressources dont elles ont besoin et la disponibilité requise en fonction des contraintes, des tolérances et. PodTopologySpread Ils doivent préférer les nœuds les plus grands qui peuvent être pleinement utilisés et atteindre les objectifs de disponibilité afin de réduire la charge du plan de contrôle, de réduire les opérations et de réduire les coûts.

Le planificateur Kubernetes essaiera automatiquement de répartir les charges de travail entre les zones de disponibilité et les hôtes si des ressources sont disponibles. Si aucune capacité n'est disponible, le Kubernetes Cluster Autoscaler tentera d'ajouter des nœuds de manière uniforme dans chaque zone de disponibilité. Karpenter essaiera d'ajouter des nœuds le plus rapidement et le moins cher possible, sauf si la charge de travail spécifie d'autres exigences.

Pour forcer les charges de travail à se répartir avec le planificateur et à créer de nouveaux nœuds entre les zones de disponibilité, vous devez utiliser : topologySpreadConstraints

```
spec:
  topologySpreadConstraints:
    - maxSkew: 3
      topologyKey: "topology.kubernetes.io/zone"
      whenUnsatisfiable: ScheduleAnyway
      labelSelector:
        matchLabels:
          dev: my-deployment
    - maxSkew: 2
      topologyKey: "kubernetes.io/hostname"
      whenUnsatisfiable: ScheduleAnyway
      labelSelector:
        matchLabels:
          dev: my-deployment
```

Utilisez des tailles de nœuds similaires pour des performances de charge de travail cohérentes

Les charges de travail doivent définir la taille des nœuds sur lesquels elles doivent être exécutées afin de garantir des performances constantes et une mise à l'échelle prévisible. Une charge de travail nécessitant 500 millions de processeurs fonctionnera différemment sur une instance à 4 cœurs par rapport à une instance à 16 cœurs. Évitez les types d'instance qui utilisent le burstable, CPUs comme les instances de la série T.

Pour garantir des performances constantes à vos charges de travail, une charge de travail peut utiliser les étiquettes Karpenter compatibles pour cibler des tailles d'instances spécifiques.

```
kind: deployment
...
spec:
  template:
```

```
    spec:
    containers:
    nodeSelector:
      karpenter.k8s.aws/instance-size: 8xlarge
```

Les charges de travail planifiées dans un cluster à l'aide du Kubernetes Cluster Autoscaler doivent associer un sélecteur de nœuds à des groupes de nœuds en fonction de la correspondance des étiquettes.

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/nodegroup
            operator: In
            values:
            - 8-core-node-group    # match your node group name
```

Utiliser efficacement les ressources informatiques

Les ressources de calcul incluent les instances EC2 et les zones de disponibilité. L'utilisation efficace des ressources informatiques augmentera votre évolutivité, votre disponibilité, vos performances et réduira votre coût total. L'utilisation efficace des ressources est extrêmement difficile à prévoir dans un environnement de mise à l'échelle automatique comportant plusieurs applications. Karpenter a été créé pour fournir des instances à la demande en fonction des besoins en matière de charge de travail afin de maximiser l'utilisation et la flexibilité.

Karpenter permet aux charges de travail de déclarer le type de ressources informatiques dont elles ont besoin sans créer au préalable des groupes de nœuds ou configurer des étiquettes altérées pour des nœuds spécifiques. Consultez les meilleures pratiques de Karpenter pour plus d'informations. Envisagez d'activer la consolidation dans votre fournisseur Karpenter pour remplacer les nœuds sous-utilisés.

Automatisez les mises à jour d'Amazon Machine Image (AMI)

La mise à jour des composants du nœud de travail vous permettra de disposer des derniers correctifs de sécurité et des fonctionnalités compatibles avec l'API Kubernetes. La mise à jour du kubelet

est le composant le plus important pour les fonctionnalités de Kubernetes, mais l'automatisation du système d'exploitation, du noyau et des correctifs d'applications installés localement réduira la maintenance à mesure que vous évoluerez.

Il est recommandé d'utiliser la dernière AMI Bottlerocket optimisée pour Amazon EKS ou Amazon EKS optimisée pour Amazon EKS pour l'image de votre nœud. Karpenter utilisera automatiquement la dernière AMI disponible pour approvisionner de nouveaux nœuds dans le cluster. Les groupes de nœuds gérés mettront à jour l'AMI lors d'une mise à jour de groupe de nœuds, mais ne mettront pas à jour l'ID de l'AMI au moment du provisionnement du nœud.

Pour les groupes de nœuds gérés, vous devez mettre à jour le modèle de lancement d'Auto Scaling Group (ASG) avec de nouvelles AMI IDs lorsqu'elles seront disponibles pour les versions de correctifs. Les versions mineures de l'AMI (par exemple 1.23.5 à 1.24.3) seront disponibles dans la console EKS et dans l'API sous forme de mises à niveau pour le groupe de nœuds. Les versions de patch (par exemple 1.23.5 à 1.23.6) ne seront pas présentées comme des mises à niveau pour les groupes de nœuds. Si vous souhaitez maintenir votre groupe de nœuds à jour avec les versions de correctifs de l'AMI, vous devez créer une nouvelle version du modèle de lancement et laisser le groupe de nœuds remplacer les instances par la nouvelle version de l'AMI.

Vous pouvez trouver la dernière AMI disponible sur cette page ou utiliser l'interface de ligne de commande AWS.

```
aws ssm get-parameter \
  --name /aws/service/eks/optimized-ami/1.24/amazon-linux-2/recommended/image_id \
  --query "Parameter.Value" \
  --output text
```

Utiliser plusieurs volumes EBS pour les conteneurs

Les volumes EBS ont un quota input/output (E/S) basé sur le type de volume (par exemple gp3) et la taille du disque. Si vos applications partagent un seul volume racine EBS avec l'hôte, cela peut épuiser le quota de disque pour l'ensemble de l'hôte et obliger les autres applications à attendre la capacité disponible. Les applications écrivent sur le disque si elles écrivent des fichiers sur leur partition superposée, montent un volume local depuis l'hôte, et également lorsqu'elles se déconnectent en sortie standard (STDOUT), selon l'agent de journalisation utilisé.

Pour éviter I/O l'épuisement du disque, vous devez monter un deuxième volume dans le dossier d'état du conteneur (par exemple /run/containerd), utiliser des volumes EBS distincts pour le stockage de la charge de travail et désactiver la journalisation locale inutile.

Pour monter un deuxième volume sur vos instances EC2 à l'aide d'eksctl, vous pouvez utiliser un groupe de nœuds avec cette configuration :

```
managedNodeGroups:
  - name: al2-workers
    amiFamily: AmazonLinux2
    desiredCapacity: 2
    volumeSize: 80
    additionalVolumes:
      - volumeName: '/dev/sdz'
        volumeSize: 100
    preBootstrapCommands:
    - |
      "systemctl stop containerd"
      "mkfs -t ext4 /dev/nvme1n1"
      "rm -rf /var/lib/containerd/*"
      "mount /dev/nvme1n1 /var/lib/containerd/"
      "systemctl start containerd"
```

Si vous utilisez Terraform pour provisionner vos groupes de nœuds, veuillez consulter des exemples dans EKS Blueprints for Terraform. Si vous utilisez Karpenter pour approvisionner des nœuds, vous pouvez utiliser les données utilisateur `blockDeviceMappings`des nœuds pour ajouter des volumes supplémentaires.

Pour monter un volume EBS directement sur votre pod, vous devez utiliser le pilote AWS EBS CSI et utiliser un volume avec une classe de stockage.

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ebs-claim
spec:
  accessModes:
    - ReadWriteOnce
```

```
  storageClassName: ebs-sc
  resources:
    requests:
      storage: 4Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: public.ecr.aws/docker/library/nginx
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: ebs-claim
```

Évitez les instances avec de faibles limites d'attachement EBS si les charges de travail utilisent des volumes EBS

L'EBS est l'un des moyens les plus simples pour les charges de travail de disposer d'un stockage persistant, mais il comporte également des limites d'évolutivité. Chaque type d'instance possède un nombre maximum de volumes EBS pouvant être attachés. Les charges de travail doivent déclarer les types d'instances sur lesquels elles doivent s'exécuter et limiter le nombre de répliques sur une seule instance présentant des caractéristiques de Kubernetes.

Désactiver la journalisation inutile sur le disque

Évitez la journalisation locale inutile en n'exécutant pas vos applications grâce à la journalisation de débogage en production et en désactivant la journalisation qui lit et écrit fréquemment sur le disque. Journald est le service de journalisation local qui conserve une mémoire tampon de journal et la vide régulièrement sur le disque. Journald est préféré à Syslog qui enregistre chaque ligne immédiatement sur le disque. La désactivation de Syslog réduit également la quantité totale de stockage dont vous avez besoin et évite d'avoir à appliquer des règles complexes de rotation des journaux. Pour désactiver Syslog, vous pouvez ajouter l'extrait suivant à votre configuration cloud-init :

```
runcmd:
  - [ systemctl, disable, --now, syslog.service ]
```

Instances de correctifs en place lorsque la vitesse de mise à jour du système d'exploitation est nécessaire

> **Important**
>
> L'application de correctifs aux instances en place ne doit être effectuée que lorsque cela est nécessaire. Amazon recommande de traiter l'infrastructure comme immuable et de tester de manière approfondie les mises à jour proposées dans des environnements inférieurs, de la même manière que les applications. Cette section s'applique lorsque cela n'est pas possible.

L'installation d'un package sur un hôte Linux existant prend quelques secondes sans perturber les charges de travail conteneurisées. Le package peut être installé et validé sans boucler, vider ou remplacer l'instance.

Pour remplacer une instance, vous devez d'abord en créer, en valider et en distribuer une nouvelle AMIs. Une instance de remplacement doit être créée, et l'ancienne instance doit être bouclée et vidée. Les charges de travail doivent ensuite être créées sur la nouvelle instance, vérifiées et répétées pour toutes les instances qui doivent être corrigées. Il faut des heures, des jours ou des semaines pour remplacer les instances en toute sécurité sans perturber les charges de travail.

Amazon recommande d'utiliser une infrastructure immuable créée, testée et promue à partir d'un système déclaratif automatisé, mais si vous devez appliquer des correctifs à des systèmes rapidement, vous devrez appliquer des correctifs aux systèmes en place et les remplacer au fur et à mesure que de nouveaux systèmes seront AMIs disponibles. En raison du décalage horaire important entre l'application de correctifs et le remplacement des systèmes, nous recommandons d'utiliser AWS Systems Manager Patch Manager pour automatiser l'application de correctifs aux nœuds lorsque cela est nécessaire.

L'application de correctifs aux nœuds vous permettra de déployer rapidement les mises à jour de sécurité et de remplacer les instances selon un calendrier régulier après la mise à jour de votre AMI. Si vous utilisez un système d'exploitation doté d'un système de fichiers racine en lecture seule, tel que Flatcar Container Linux ou Bottlerocket OS, nous vous recommandons d'utiliser les opérateurs de mise à jour compatibles avec ces systèmes d'exploitation. L'opérateur de mise à jour Flatcar Linux

et l'opérateur de mise à jour Bottlerocket redémarreront les instances pour maintenir les nœuds à jour automatiquement.

Services de cluster

Les services de cluster s'exécutent au sein d'un cluster EKS, mais ils ne constituent pas des charges de travail utilisateur. Si vous avez un serveur Linux, vous devez souvent exécuter des services tels que NTP, Syslog et un environnement d'exécution de conteneur pour prendre en charge vos charges de travail. Les services de cluster sont similaires et prennent en charge des services qui vous aident à automatiser et à exploiter votre cluster. Dans Kubernetes, ils sont généralement exécutés dans l'espace de noms du système kube et certains sont exécutés en tant que. DaemonSets

Les services de cluster sont censés avoir un temps de disponibilité élevé et sont souvent essentiels en cas de panne et pour le dépannage. Si un service de cluster principal n'est pas disponible, vous risquez de perdre l'accès aux données qui peuvent aider à récupérer ou à prévenir une panne (par exemple, utilisation élevée du disque). Ils doivent s'exécuter sur des instances de calcul dédiées, telles qu'un groupe de nœuds distinct ou AWS Fargate. Cela garantira que les services du cluster ne sont pas affectés sur les instances partagées par des charges de travail susceptibles d'augmenter ou d'utiliser davantage de ressources.

CoreDNS à l'échelle

La mise à l'échelle de CoreDNS repose sur deux mécanismes principaux. Réduction du nombre d'appels au service CoreDNS et augmentation du nombre de répliques.

Réduisez les requêtes externes en diminuant le nombre de points

Le paramètre ndots indique le nombre de points (également appelés « points ») d'un nom de domaine considérés comme suffisants pour éviter d'interroger le DNS. Si votre application possède un paramètre ndots de 5 (par défaut) et que vous demandez des ressources à un domaine externe tel que api.example.com (2 points), CoreDNS sera interrogé pour chaque domaine de recherche défini dans /etc/resolv.conf pour un domaine plus spécifique. Par défaut, les domaines suivants seront recherchés avant de faire une demande externe.

```
api.example.<namespace>.svc.cluster.local
api.example.svc.cluster.local
api.example.cluster.local
api.example.<region>.compute.internal
```

Les `region` valeurs `namespace` et seront remplacées par l'espace de noms de vos charges de travail et votre région de calcul. Il se peut que vous disposiez de domaines de recherche supplémentaires en fonction des paramètres de votre cluster.

Vous pouvez réduire le nombre de requêtes adressées à CoreDNS en diminuant l'option ndots de votre charge de travail ou en qualifiant complètement vos demandes de domaine en incluant un suivi. (par exemple`api.example.com.`). Si votre charge de travail se connecte à des services externes via le DNS, nous vous recommandons de définir ndots sur 2 afin que les charges de travail ne rendent pas inutiles les requêtes DNS du cluster au sein du cluster. Vous pouvez définir un serveur DNS et un domaine de recherche différents si la charge de travail ne nécessite pas l'accès aux services du cluster.

```
spec:
  dnsPolicy: "None"
  dnsConfig:
    options:
      - name: ndots
        value: "2"
      - name: edns0
```

Si vous réduisez ndots à une valeur trop faible ou si les domaines auxquels vous vous connectez ne sont pas suffisamment spécifiques (y compris les derniers points), il est possible que les recherches DNS échouent. Assurez-vous de tester l'impact de ce paramètre sur vos charges de travail.

Redimensionner CoreDNS horizontalement

Les instances CoreDNS peuvent évoluer en ajoutant des répliques supplémentaires au déploiement. Il est recommandé d'utiliser le NodeLocal DNS ou l'autoscaler proportionnel au cluster pour dimensionner CoreDNS.

NodeLocal Le DNS nécessitera l'exécution d'une instance par nœud, en tant que nœud, DaemonSet ce qui nécessite davantage de ressources de calcul dans le cluster, mais cela évitera l'échec des requêtes DNS et réduira le temps de réponse des requêtes DNS dans le cluster. L'autoscaler proportionnel du cluster adaptera CoreDNS en fonction du nombre de nœuds ou de cœurs du cluster. Il ne s'agit pas d'une corrélation directe avec les requêtes, mais cela peut être utile en fonction de vos charges de travail et de la taille de votre cluster. L'échelle proportionnelle par défaut consiste à ajouter une réplique supplémentaire pour 256 cœurs ou 16 nœuds du cluster, selon la première éventualité.

Si vous utilisez le module complémentaire CoreDNS EKS, pensez à activer l'option de mise à l'échelle automatique. L'autoscaler CoreDNS ajuste dynamiquement le nombre de répliques CoreDNS en surveillant le nombre de nœuds et le nombre de cœurs du processeur, en utilisant une formule qui prend le maximum de (nœuds ÷ 16) ou (cœurs de processeur ÷ 256), en augmentant immédiatement si nécessaire et en diminuant progressivement pour maintenir la stabilité.

Faites dimensionner le serveur de métriques Kubernetes à la verticale

Le serveur Kubernetes Metrics prend en charge le dimensionnement horizontal et vertical. En dimensionnant horizontalement le serveur de métriques, il sera hautement disponible, mais il ne sera pas redimensionné horizontalement pour gérer un plus grand nombre de métriques de clusters. Vous devrez dimensionner verticalement le serveur de métriques en fonction de ses recommandations au fur et à mesure que les nœuds et les métriques collectées sont ajoutés au cluster.

Le serveur de métriques conserve en mémoire les données qu'il collecte, agrège et sert. À mesure qu'un cluster s'agrandit, la quantité de données que le serveur de métriques stocke augmente. Dans les grands clusters, le serveur Metrics aura besoin de plus de ressources de calcul que la quantité de mémoire et de CPU réservée dans l'installation par défaut. Vous pouvez utiliser le Vertical Pod Autoscaler (VPA) ou Addon Resizer pour redimensionner le serveur de métriques. L'Addon Resizer évolue verticalement proportionnellement aux nœuds de travail et le VPA évolue en fonction de l'utilisation du processeur et de la mémoire.

Durée du lameduck CoreDNS

Les pods utilisent le `kube-dns` service pour la résolution des noms. Kubernetes utilise le NAT de destination (DNAT) pour rediriger le `kube-dns` trafic des nœuds vers les pods principaux CoreDNS. Au fur et à mesure que vous adaptez le déploiement CoreDNS`kube-proxy`, les règles iptables et les chaînes sont mises à jour sur les nœuds afin de rediriger le trafic DNS vers les pods CoreDNS. La propagation de nouveaux points de terminaison lorsque vous augmentez et la suppression de règles lorsque vous réduisez CoreDNS peuvent prendre entre 1 et 10 secondes selon la taille du cluster.

Ce délai de propagation peut entraîner des échecs de recherche DNS lorsqu'un pod CoreDNS est arrêté alors que les règles iptables du nœud n'ont pas été mises à jour. Dans ce scénario, le nœud peut continuer à envoyer des requêtes DNS à un pod CoreDNS arrêté.

Vous pouvez réduire les échecs de recherche DNS en définissant une durée de lameduck dans vos pods CoreDNS. En mode lameduck, CoreDNS continuera de répondre aux demandes en vol. La définition d'une durée de lameduck retardera le processus d'arrêt de CoreDNS, laissant aux nœuds le temps dont ils ont besoin pour mettre à jour leurs règles et chaînes iptables.

Nous vous recommandons de définir la durée de CoreDNS lameduck sur 30 secondes.

Sonde de préparation CoreDNS

Nous vous recommandons d'utiliser `/ready` plutôt que `/health` pour la sonde de préparation de CoreDNS.

Conformément à la recommandation précédente de fixer la durée du lameduck à 30 secondes, afin de laisser suffisamment de temps pour que les règles iptables du nœud soient mises à jour avant la fin du pod, l'utilisation `/ready` plutôt que de `/health` la sonde de préparation CoreDNS garantit que le pod CoreDNS est parfaitement préparé au démarrage pour répondre rapidement aux requêtes DNS.

```
readinessProbe:
  httpGet:
    path: /ready
    port: 8181
    scheme: HTTP
```

Pour plus d'informations sur le plugin CoreDNS Ready, veuillez consulter https://coredns. io/plugins/ready/

Agents de journalisation et de surveillance

Les agents de journalisation et de surveillance peuvent alourdir considérablement le plan de contrôle de votre cluster, car ils interrogent le serveur d'API pour enrichir les journaux et les métriques avec des métadonnées de charge de travail. L'agent d'un nœud n'a accès qu'aux ressources du nœud local pour voir des informations telles que le conteneur et le nom du processus. En interrogeant le serveur d'API, il peut ajouter des détails supplémentaires tels que le nom et les étiquettes du déploiement de Kubernetes. Cela peut être extrêmement utile pour le dépannage, mais cela nuit à la mise à l'échelle.

Comme il existe de nombreuses options de journalisation et de surveillance, nous ne pouvons pas donner d'exemples pour chaque fournisseur. Avec fluentbit, nous recommandons d'activer Use_Kubelet pour récupérer les métadonnées du kubelet local plutôt que du serveur d'API Kubernetes et de définir un nombre qui réduit les appels répétés lorsque les données peuvent `Kube_Meta_Cache_TTL` être mises en cache (par exemple 60).

Le dimensionnement de la surveillance et de la journalisation comporte deux options générales :

- Désactiver les intégrations
- Échantillonnage et filtrage

La désactivation des intégrations n'est souvent pas une option car vous perdez les métadonnées des journaux. Cela élimine le problème de dimensionnement de l'API, mais cela introduira d'autres problèmes en ne disposant pas des métadonnées requises en cas de besoin.

L'échantillonnage et le filtrage réduisent le nombre de mesures et de journaux collectés. Cela réduira le nombre de requêtes adressées à l'API Kubernetes, ainsi que la quantité de stockage nécessaire pour les métriques et les journaux collectés. La réduction des coûts de stockage réduira le coût de l'ensemble du système.

La possibilité de configurer l'échantillonnage dépend du logiciel de l'agent et peut être mise en œuvre à différents points d'ingestion. Il est important d'ajouter un échantillonnage le plus près possible de l'agent, car c'est probablement là que les appels du serveur d'API ont lieu. Contactez votre fournisseur pour en savoir plus sur l'assistance en matière d'échantillonnage.

Si vous utilisez CloudWatch and CloudWatch Logs, vous pouvez ajouter un filtrage par agent à l'aide des modèles décrits dans la documentation.

Pour éviter de perdre des journaux et des métriques, vous devez envoyer vos données vers un système capable de les mettre en mémoire tampon en cas de panne du terminal récepteur. Avec Fluentbit, vous pouvez utiliser Amazon Kinesis Data Firehose pour conserver temporairement des données, ce qui peut réduire le risque de surcharge de votre emplacement de stockage final.

Charges de travail

Les charges de travail ont un impact sur la taille que peut atteindre votre cluster. Les charges de travail qui utilisent APIs fortement Kubernetes limitent le nombre total de charges de travail que vous pouvez avoir dans un seul cluster, mais vous pouvez modifier certaines valeurs par défaut pour réduire la charge.

Les charges de travail d'un cluster Kubernetes ont accès à des fonctionnalités intégrées à l'API Kubernetes (par exemple, Secrets et ServiceAccounts), mais ces fonctionnalités ne sont pas toujours obligatoires et doivent être désactivées si elles ne sont pas utilisées. La limitation de l'accès aux charges de travail et de la dépendance à l'égard du plan de contrôle Kubernetes augmentera le nombre de charges de travail que vous pouvez exécuter dans le cluster et améliorera la sécurité

de vos clusters en supprimant les accès inutiles aux charges de travail et en mettant en œuvre les pratiques du moindre privilège. Consultez les meilleures pratiques en matière de sécurité pour plus d'informations.

Utilisation IPv6 pour la mise en réseau des pods

Vous ne pouvez pas passer d'un VPC IPv4 à un VPC. IPv6 Il est donc important de l'activer IPv6 avant de provisionner un cluster. Si vous l'activez IPv6 dans un VPC, cela ne signifie pas que vous devez l'utiliser et si vos pods et services l'utilisent, IPv6 vous pouvez toujours acheminer le trafic vers et depuis IPv4 les adresses. Consultez les meilleures pratiques de mise en réseau EKS pour plus d'informations.

L'utilisation IPv6 dans votre cluster permet d'éviter certaines des limites de dimensionnement des clusters et de la charge de travail les plus courantes. IPv6 évite l'épuisement des adresses IP lorsque les pods et les nœuds ne peuvent pas être créés car aucune adresse IP n'est disponible. Il présente également des améliorations des performances par nœud, car les pods reçoivent les adresses IP plus rapidement en réduisant le nombre de pièces jointes ENI par nœud. Vous pouvez obtenir des performances de nœud similaires en utilisant le mode IPv4 préfixe dans le VPC CNI, mais vous devez tout de même vous assurer que vous disposez de suffisamment d'adresses IP disponibles dans le VPC.

Limiter le nombre de services par espace de noms

Le nombre maximum de services dans un espace de noms est de 5 000 et le nombre maximum de services dans un cluster est de 10 000. Pour aider à organiser les charges de travail et les services, à améliorer les performances et à éviter un impact en cascade sur les ressources limitées aux espaces de noms, nous recommandons de limiter le nombre de services par espace de noms à 500.

Le nombre de règles de tables IP créées par nœud avec kube-proxy augmente avec le nombre total de services dans le cluster. La génération de milliers de règles de tables IP et le routage de paquets via ces règles ont un impact sur les performances des nœuds et augmentent la latence du réseau.

Créez des espaces de noms Kubernetes qui englobent un seul environnement d'application tant que le nombre de services par espace de noms est inférieur à 500. Cela permettra de réduire la taille de la découverte de services pour éviter les limites de découverte de services et peut également vous aider à éviter les collisions de noms de services. Les environnements d'applications (par exemple dev, test, prod) doivent utiliser des clusters EKS distincts au lieu d'espaces de noms.

Comprendre les quotas d'Elastic Load Balancer

Lors de la création de vos services, réfléchissez au type d'équilibrage de charge que vous utiliserez (par exemple, Network Load Balancer (NLB) ou Application Load Balancer (ALB)). Chaque type d'équilibreur de charge fournit des fonctionnalités différentes et possède des quotas différents. Certains quotas par défaut peuvent être ajustés, mais certains quotas maximaux ne peuvent pas être modifiés. Pour consulter les quotas et l'utilisation de votre compte, consultez le tableau de bord des Quotas de Service dans la console AWS.

Par exemple, les cibles ALB par défaut sont 1000. Si votre service compte plus de 1 000 points de terminaison, vous devrez augmenter le quota, diviser le service en plusieurs ALBs ou utiliser Kubernetes Ingress. Les cibles NLB par défaut sont de 3 000, mais elles sont limitées à 500 cibles par AZ. Si votre cluster gère plus de 500 pods pour un service NLB, vous devrez en utiliser plusieurs AZs ou demander une augmentation de la limite de quota.

Une alternative à l'utilisation d'un équilibreur de charge couplé à un service consiste à utiliser un contrôleur d'entrée. Le contrôleur AWS Load Balancer peut créer des ressources ALBs d'entrée, mais vous pouvez envisager d'exécuter un contrôleur dédié dans votre cluster. Un contrôleur d'entrée intégré au cluster vous permet d'exposer plusieurs services Kubernetes à partir d'un seul équilibreur de charge en exécutant un proxy inverse au sein de votre cluster. Les contrôleurs possèdent différentes fonctionnalités, telles que la prise en charge de l'API Gateway, qui peuvent présenter des avantages en fonction du nombre et de l'ampleur de vos charges de travail.

Utilisez Route 53, Global Accelerator ou CloudFront

Pour qu'un service utilisant plusieurs équilibreurs de charge soit disponible en tant que point de terminaison unique, vous devez utiliser Amazon CloudFront, AWS Global Accelerator ou Amazon Route 53 pour exposer tous les équilibreurs de charge en tant que point de terminaison unique destiné au client. Chaque option présente des avantages différents et peut être utilisée séparément ou conjointement selon vos besoins.

La Route 53 peut exposer plusieurs équilibreurs de charge sous un nom commun et peut envoyer du trafic vers chacun d'entre eux en fonction du poids attribué. Pour en savoir plus sur les poids DNS, consultez la documentation et découvrez comment les implémenter avec le contrôleur DNS externe Kubernetes dans la documentation d'AWS Load Balancer Controller.

Global Accelerator peut acheminer les charges de travail vers la région la plus proche en fonction de l'adresse IP de la demande. Cela peut être utile pour les charges de travail déployées dans plusieurs régions, mais cela n'améliore pas le routage vers un seul cluster dans une seule région. L'utilisation

de Route 53 en combinaison avec le Global Accelerator présente des avantages supplémentaires tels que la vérification de l'état de santé et le basculement automatique en cas d'absence d'AZ. Vous pouvez voir un exemple d'utilisation de Global Accelerator avec Route 53 dans ce billet de blog.

CloudFront peut être utilisé avec Route 53 et Global Accelerator ou seul pour acheminer le trafic vers plusieurs destinations. CloudFront met en cache les actifs servis à partir des sources d'origine, ce qui peut réduire les besoins en bande passante en fonction de ce que vous servez.

À utiliser à la EndpointSlices place des points de terminaison

Lorsque vous découvrez des pods correspondant à une étiquette de service, vous devez les utiliser à la EndpointSlicesplace des endpoints. Les points de terminaison étaient un moyen simple d'exposer des services à petite échelle, mais les services volumineux qui évoluent automatiquement ou qui sont mis à jour génèrent un trafic important sur le plan de contrôle Kubernetes. EndpointSlices ont un regroupement automatique qui active des éléments tels que des indications tenant compte de la topologie.

Toutes les manettes ne sont pas utilisées EndpointSlices par défaut. Vous devez vérifier les paramètres de votre manette et l'activer si nécessaire. Pour le contrôleur AWS Load Balancer, vous devez activer l'indicateur `--enable-endpoint-slices` facultatif à utiliser. EndpointSlices

Utilisez des secrets immuables et externes si possible

Le kubelet conserve un cache des clés et des valeurs actuelles pour les secrets utilisés dans les volumes destinés aux pods de ce nœud. Le kubelet surveille les secrets pour détecter les changements. À mesure que le cluster évolue, le nombre croissant de montres peut avoir un impact négatif sur les performances du serveur d'API.

Il existe deux stratégies pour réduire le nombre de montres sur Secrets :

- Pour les applications qui n'ont pas besoin d'accéder aux ressources Kubernetes, vous pouvez désactiver le montage automatique des secrets des comptes de service en définissant Token : false automountServiceAccount
- Si les secrets de votre application sont statiques et ne seront pas modifiés à l'avenir, marquez-les comme immuables. Le kubelet ne surveille pas l'API pour détecter les secrets immuables.

Pour désactiver le montage automatique d'un compte de service sur les pods, vous pouvez utiliser le paramètre suivant dans votre charge de travail. Vous pouvez annuler ces paramètres si des charges de travail spécifiques nécessitent un compte de service.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app
automountServiceAccountToken: true
```

Surveillez le nombre de secrets du cluster avant qu'il ne dépasse la limite de 10 000. Vous pouvez voir le nombre total de secrets dans un cluster à l'aide de la commande suivante. Vous devez surveiller cette limite à l'aide de vos outils de surveillance du cluster.

```
kubectl get secrets -A | wc -l
```

Vous devez configurer la surveillance pour alerter un administrateur du cluster avant que cette limite ne soit atteinte. Envisagez d'utiliser des options de gestion des secrets externes telles qu'AWS Key Management Service (AWS KMS) ou Hashicorp Vault avec le pilote CSI Secrets Store.

Historique des déploiements limités

Les pods peuvent être lents lors de la création, de la mise à jour ou de la suppression, car les anciens objets sont toujours suivis dans le cluster. Vous pouvez réduire le nombre `revisionHistoryLimit` de déploiements pour nettoyer les anciens, ReplicaSets ce qui réduira le nombre total d'objets suivis par le Kubernetes Controller Manager. La limite d'historique par défaut pour les déploiements en 10.

Si votre cluster crée de nombreux objets de travail par le biais de mécanismes CronJobs ou d'autres mécanismes, vous devez utiliser le `ttlSecondsAfterFinished`paramètre pour nettoyer automatiquement les anciens modules du cluster. Cela supprimera les tâches exécutées avec succès de l'historique des tâches après un certain temps.

Désactiver enableServiceLinks par défaut

Lorsqu'un pod s'exécute sur un nœud, le kubelet ajoute un ensemble de variables d'environnement pour chaque service actif. Les processus Linux ont une taille maximale pour leur environnement, qui peut être atteinte si vous avez trop de services dans votre espace de noms. Le nombre de services par espace de noms ne doit pas dépasser 5 000. Ensuite, le nombre de variables d'environnement de service dépasse les limites du shell, ce qui entraîne le blocage des Pods au démarrage.

Il existe d'autres raisons pour lesquelles les pods ne devraient pas utiliser de variables d'environnement de service pour la découverte de services. Les conflits de noms de variables

d'environnement, les noms de service divulgués et la taille totale de l'environnement en sont quelques exemples. Vous devez utiliser CoreDNS pour découvrir les points de terminaison de service.

Limiter les webhooks d'admission dynamique par ressource

Les webhooks d'admission dynamiques incluent les webhooks d'admission et les webhooks mutants. Il s'agit de points de terminaison d'API ne faisant pas partie du plan de contrôle Kubernetes qui sont appelés en séquence lorsqu'une ressource est envoyée à l'API Kubernetes. Chaque webhook a un délai d'expiration par défaut de 10 secondes et peut augmenter la durée d'une demande d'API si vous avez plusieurs webhooks ou si l'un d'entre eux expire.

Assurez-vous que vos webhooks sont hautement disponibles, en particulier lors d'un incident AZ, et que la FailurePolicy est correctement définie pour rejeter la ressource ou ignorer l'échec. N'appelez pas de webhook lorsque vous n'en avez pas besoin en autorisant les commandes kubectl --dry-run à contourner le webhook.

```
apiVersion: admission.k8s.io/v1
kind: AdmissionReview
request:
  dryRun: False
```

Les webhooks mutants peuvent modifier les ressources en succession fréquente. Si vous avez 5 webhooks mutants et que vous déployez 50 ressources, etcd stockera toutes les versions de chaque ressource jusqu'à ce que le compactage s'exécute (toutes les 5 minutes) afin de supprimer les anciennes versions des ressources modifiées. Dans ce scénario, lorsqu'etcd supprime les ressources remplacées, 200 versions de ressources seront supprimées d'etcd et, en fonction de la taille des ressources, cela peut utiliser un espace considérable sur l'hôte etcd jusqu'à ce que la défragmentation s'exécute toutes les 15 minutes.

Cette défragmentation peut provoquer des pauses dans etcd, ce qui pourrait avoir d'autres effets sur l'API et les contrôleurs Kubernetes. Vous devez éviter de modifier fréquemment de grandes ressources ou de modifier des centaines de ressources en succession rapide.

Comparez les charges de travail entre plusieurs clusters

Si vous avez deux clusters qui devraient avoir des performances similaires, mais ce n'est pas le cas, essayez de comparer les indicateurs pour en déterminer la raison.

Par exemple, la comparaison de la latence d'un cluster est un problème courant. Cela est généralement dû à une différence dans le volume des demandes d'API. Vous pouvez exécuter la CloudWatch LogInsight requête suivante pour comprendre la différence.

```
filter @logStream like "kube-apiserver-audit"
| stats count(*) as cnt by objectRef.apiGroup, objectRef.apiVersion,
 objectRef.resource, userAgent, verb, responseStatus.code
| sort cnt desc
| limit 1000
```

Vous pouvez ajouter des filtres supplémentaires pour l'affiner, par exemple en vous concentrant sur toutes les demandes de liste provenant de`foo`.

```
filter @logStream like "kube-apiserver-audit"
| filter verb = "list"
| filter user.username like "foo"
| stats count(*) as cnt by objectRef.apiGroup, objectRef.apiVersion,
 objectRef.resource, responseStatus.code
| sort cnt desc
| limit 1000
```

Théorie de la mise à l'échelle de Kubernetes

Nœuds par rapport au taux de désabonnement

Lorsque nous discutons de l'évolutivité de Kubernetes, nous le faisons souvent en fonction du nombre de nœuds présents dans un seul cluster. Il est intéressant de noter que c'est rarement la métrique la plus utile pour comprendre l'évolutivité. Par exemple, un cluster de 5 000 nœuds avec un nombre important mais fixe de pods n'exercerait pas beaucoup de stress sur le plan de contrôle après la configuration initiale. Cependant, si nous prenions un cluster de 1 000 nœuds et essayions de créer 10 000 emplois de courte durée en moins d'une minute, cela exercerait une forte pression soutenue sur le plan de contrôle.

Le simple fait d'utiliser le nombre de nœuds pour comprendre le dimensionnement peut être trompeur. Il est préférable de penser en termes de taux de changement qui se produit au cours d'une période donnée (utilisons un intervalle de 5 minutes pour cette discussion, car c'est ce que les requêtes Prometheus utilisent généralement par défaut). Voyons pourquoi le fait de définir le problème en termes de taux de variation peut nous donner une meilleure idée de ce qu'il faut ajuster pour atteindre l'échelle souhaitée.

Réflexion en requêtes par seconde

Kubernetes dispose d'un certain nombre de mécanismes de protection pour chaque composant (Kubelet, Scheduler, Kube Controller Manager et serveur d'API) afin d'éviter de submerger le maillon suivant de la chaîne Kubernetes. Par exemple, le Kubelet dispose d'un indicateur pour limiter les appels au serveur d'API à un certain rythme. Ces mécanismes de protection sont généralement, mais pas toujours, exprimés en termes de requêtes autorisées par seconde ou de QPS.

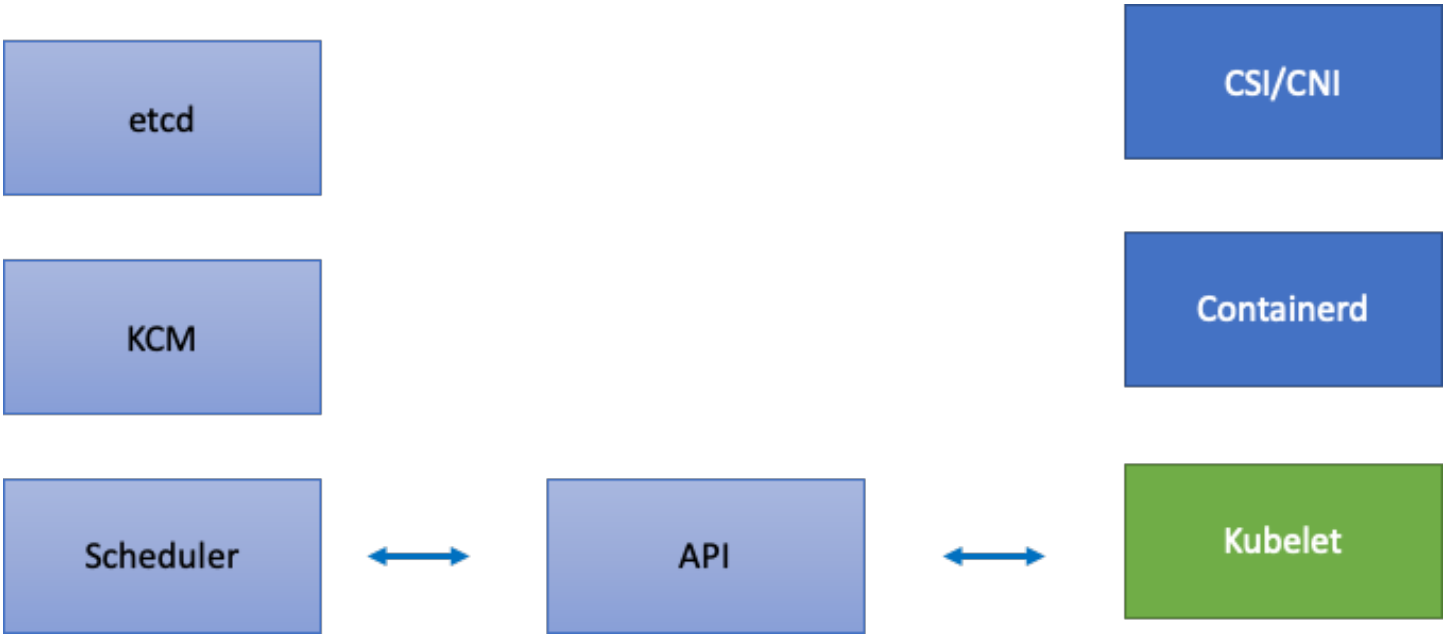
Il faut faire très attention lors de la modification de ces paramètres QPS. La suppression d'un goulot d'étranglement, tel que le nombre de requêtes par seconde sur un Kubelet, aura un impact sur les autres composants en aval. Cela peut surcharger le système au-delà d'un certain rythme. Il est donc essentiel de comprendre et de surveiller chaque partie de la chaîne de services pour réussir à dimensionner les charges de travail sur Kubernetes.

> **Note**
>
> Le serveur d'API dispose d'un système plus complexe avec l'introduction de la priorité et de l'équité des API, dont nous parlerons séparément.

> **Note**
>
> Attention, certains indicateurs semblent convenir, mais ils mesurent en fait autre chose. Par exemple, cela `kubelet_http_inflight_requests` concerne uniquement le serveur de métriques dans Kubelet, et non le nombre de requêtes de Kubelet aux requêtes apiserver. Cela pourrait nous amener à mal configurer le drapeau QPS sur le Kubelet. Une requête sur les journaux d'audit d'un Kubelet en particulier serait un moyen plus fiable de vérifier les métriques.

Mise à l'échelle des composants distribués

EKS étant un service géré, divisons les composants Kubernetes en deux catégories : les composants gérés par AWS, qui incluent etcd, Kube Controller Manager et le planificateur (sur la partie gauche du schéma), et les composants configurables par le client tels que Kubelet, Container Runtime et les différents opérateurs qui appellent AWS, APIs tels que les pilotes réseau et de stockage (sur la partie droite du schéma). Nous laissons le serveur d'API au centre même s'il est géré par AWS, car les paramètres de priorité et d'équité des API peuvent être configurés par les clients.



Goulets d'étranglement en amont et en aval

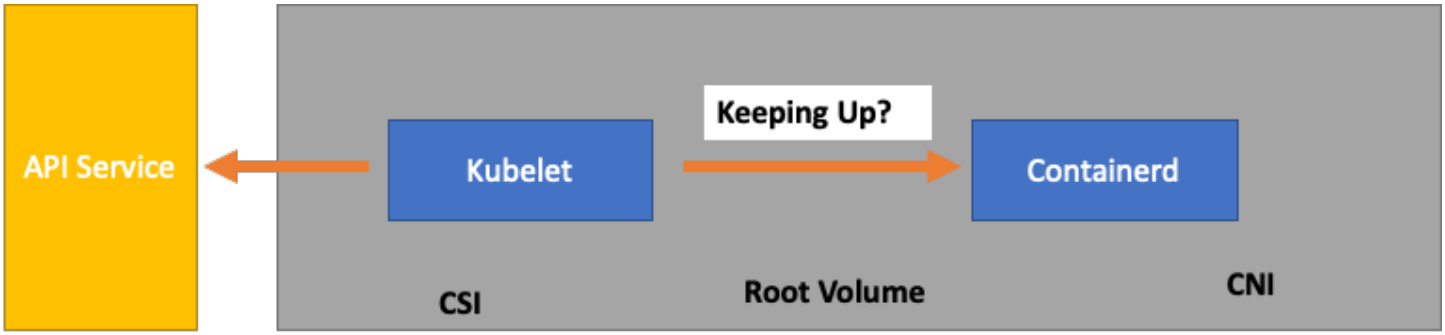
Lorsque nous surveillons chaque service, il est important d'examiner les indicateurs dans les deux sens afin de détecter les goulots d'étranglement. Apprenons comment procéder en utilisant Kubelet comme exemple. Kubelet parle à la fois au serveur d'API et au runtime du conteneur ; comment et que devons-nous surveiller pour détecter si l'un des composants rencontre un problème ?

Combien de pods par nœud

Lorsque nous examinons les chiffres d'échelle, tels que le nombre de pods pouvant fonctionner sur un nœud, nous pouvons prendre les 110 pods par nœud pris en charge en amont à leur valeur nominale.

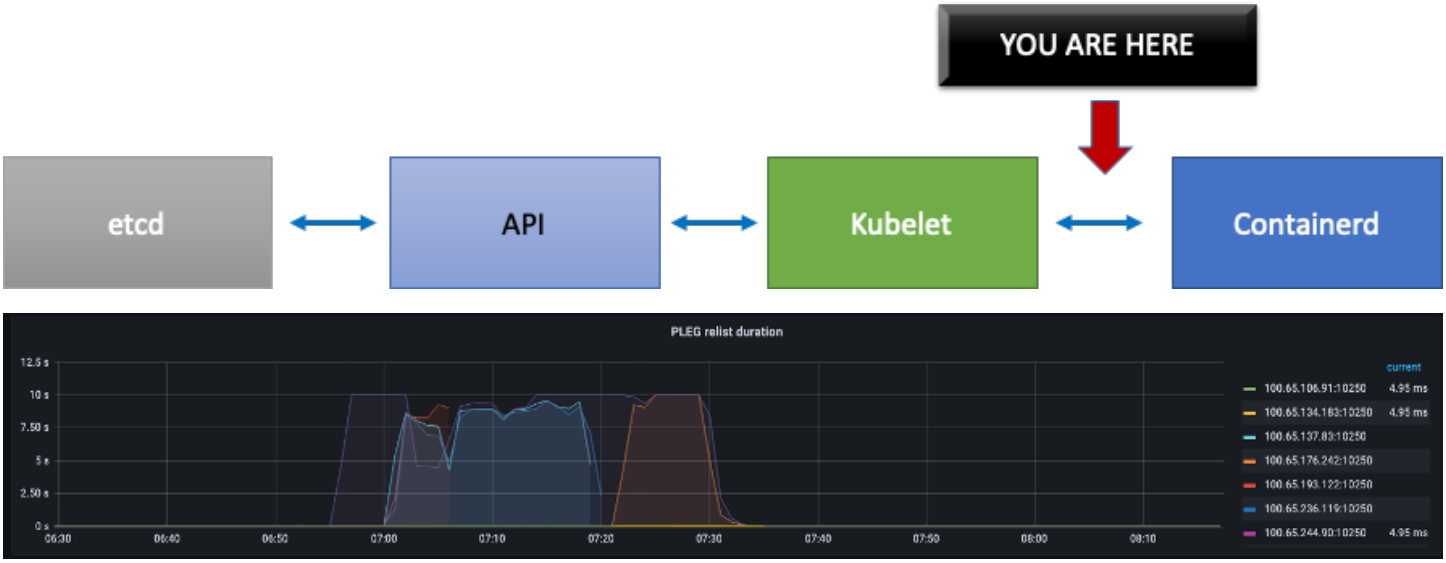
> **Note**
>
> https://kubernetes. io/docs/setup/best-practices/cluster-grand/

Cependant, votre charge de travail est probablement plus complexe que celle testée lors d'un test d'évolutivité dans Upstream. Pour nous assurer que nous pouvons gérer le nombre de pods que nous voulons exécuter en production, veillons à ce que le Kubelet « suive » le rythme du runtime de Containerd.



Pour simplifier à l'excès, le Kubelet obtient l'état des pods à partir du runtime du conteneur (dans notre cas, Containerd). Et si un trop grand nombre de pods changeaient de statut trop rapidement ? Si le taux de modification est trop élevé, les demandes [adressées au moteur d'exécution du conteneur] peuvent expirer.

> **Note**
>
> Kubernetes est en constante évolution, ce sous-système est actuellement en cours de modification. https://github.com/kubernetes/améliorations/problèmes/3386



Dans le graphique ci-dessus, nous voyons une ligne plate indiquant que nous venons d'atteindre la valeur du délai d'expiration pour la métrique de durée de génération des événements du cycle de vie du pod. Si vous souhaitez voir cela dans votre propre cluster, vous pouvez utiliser la syntaxe ProMQL suivante.

```
increase(kubelet_pleg_relist_duration_seconds_bucket{instance="$instance"}
[$__rate_interval])
```

Si nous sommes témoins de ce comportement de temporisation, nous savons que nous avons poussé le nœud au-delà de la limite dont il était capable. Nous devons corriger la cause du délai d'attente avant de poursuivre. Cela peut être réalisé en réduisant le nombre de pods par nœud ou en recherchant les erreurs susceptibles de provoquer un volume élevé de tentatives (affectant ainsi le taux de désabonnement). L'important à retenir est que les métriques sont le meilleur moyen de comprendre si un nœud est capable de gérer le taux de désabonnement des pods assignés par rapport à l'utilisation d'un nombre fixe.

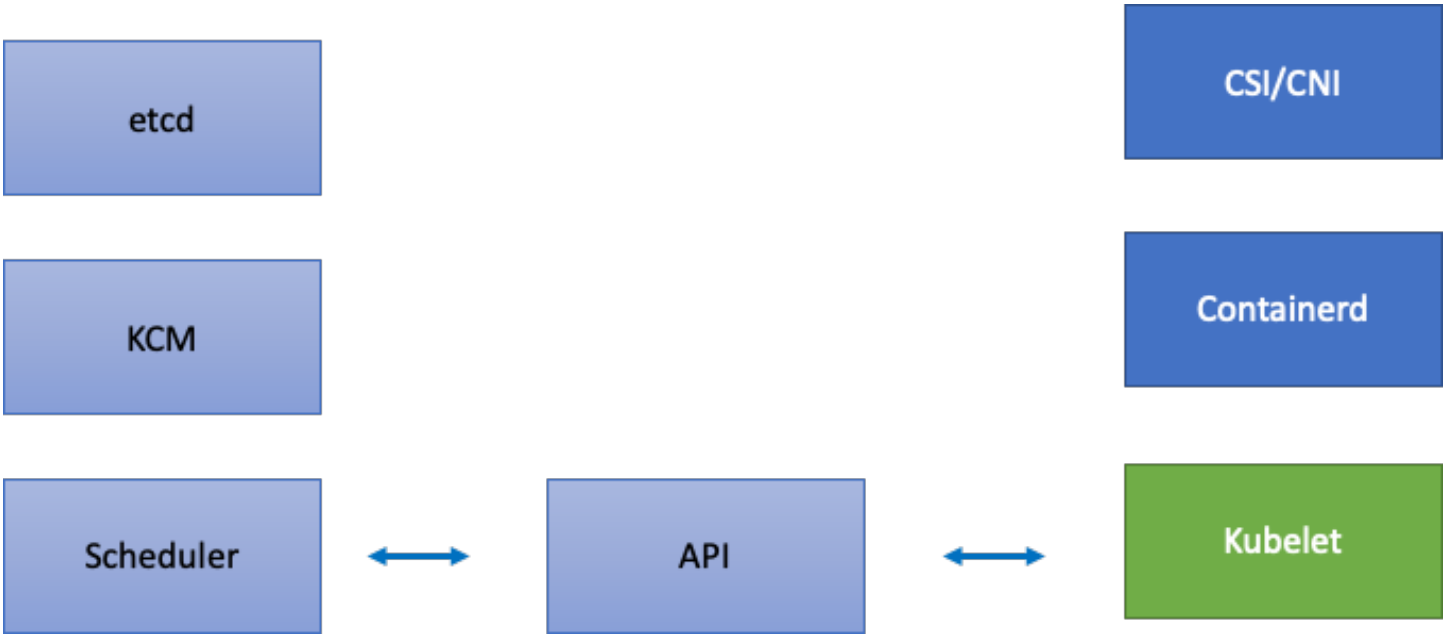
Échelle par métriques

Bien que le concept d'utilisation de métriques pour optimiser les systèmes soit ancien, il est souvent négligé lorsque les utilisateurs commencent leur parcours vers Kubernetes. Au lieu de nous concentrer sur des chiffres spécifiques (110 pods par nœud), nous concentrons nos efforts sur la recherche des indicateurs qui nous aident à identifier les goulots d'étranglement dans notre système. Comprendre les bons seuils pour ces indicateurs peut nous donner un degré élevé de confiance dans la configuration optimale de notre système.

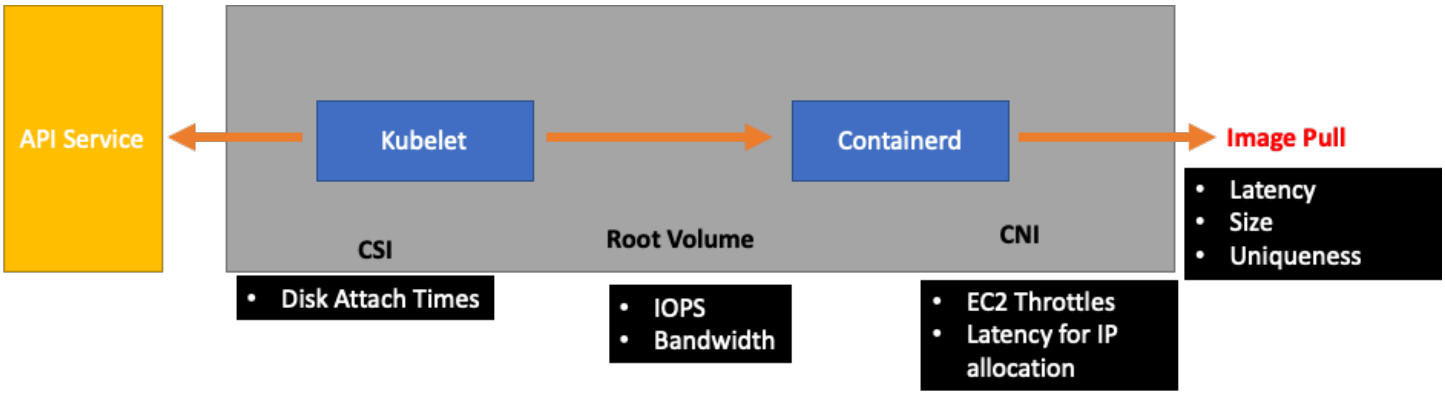
L'impact des changements

Une tendance courante qui peut nous causer des problèmes consiste à nous concentrer sur la première erreur de métrique ou de journal qui semble suspecte. Lorsque nous avons constaté que le Kubelet avait expiré plus tôt, nous avons pu essayer des choses aléatoires, comme augmenter le taux par seconde que le Kubelet est autorisé à envoyer, etc. Cependant, il est sage de regarder la situation dans son ensemble en aval de l'erreur détectée en premier. Apportez chaque modification dans un but précis et en vous appuyant sur des données.

En aval du Kubelet se trouve le runtime Containerd (erreurs du pod), DaemonSets tel que le pilote de stockage (CSI) et le pilote réseau (CNI) qui communiquent avec l'API EC2, etc.



Continuons notre exemple précédent où le Kubelet ne suit pas le rythme du temps d'exécution. Il existe un certain nombre de points où nous pouvons regrouper un nœud si densément qu'il déclenche des erreurs.



Lorsque nous concevons la bonne taille de nœud pour nos charges de travail, ces easy-to-overlook signaux peuvent exercer une pression inutile sur le système, limitant ainsi à la fois notre échelle et nos performances.

Le coût des erreurs inutiles

Les contrôleurs Kubernetes excellent lorsqu'il s'agit de réessayer en cas d'erreur, mais cela a un coût. Ces nouvelles tentatives peuvent augmenter la pression sur des composants tels que le Kube Controller Manager. La surveillance de telles erreurs est un élément important des tests à grande échelle.

Lorsque moins d'erreurs se produisent, il est plus facile de détecter les problèmes dans le système. En veillant régulièrement à ce que nos clusters soient exempts d'erreurs avant les opérations majeures (telles que les mises à niveau), nous pouvons simplifier les journaux de dépannage en cas d'événements imprévus.

Élargir notre point de vue

Dans les clusters à grande échelle comportant des milliers de nœuds, nous ne voulons pas rechercher les goulots d'étranglement individuellement. Dans ProMQL, nous pouvons trouver les valeurs les plus élevées d'un ensemble de données à l'aide d'une fonction appelée topk ; K étant une variable, nous plaçons le nombre d'éléments souhaités. Ici, nous utilisons trois nœuds pour savoir si tous les Kubelets du cluster sont saturés. Nous avons étudié la latence jusqu'à présent. Voyons maintenant si le Kubelet supprime les événements.

```
topk(3, increase(kubelet_pleg_discard_events{}[$__rate_interval]))
```

Décomposer cette déclaration.

- Nous utilisons la variable Grafana pour nous `$__rate_interval` assurer qu'elle obtient les quatre échantillons dont elle a besoin. Cela permet de contourner un sujet complexe en matière de surveillance grâce à une variable simple.
- `topk`ne nous donnez que les meilleurs résultats et le chiffre 3 limite ces résultats à trois. Il s'agit d'une fonction utile pour les métriques à l'échelle du cluster.
- `{}`dites-nous qu'il n'y a pas de filtres. Normalement, vous devez saisir le nom de la tâche quelle que soit la règle de scraping, mais comme ces noms varient, nous le laisserons vide.

Diviser le problème en deux

Pour remédier à un goulot d'étranglement dans le système, nous adopterons une approche qui consiste à trouver un indicateur indiquant l'existence d'un problème en amont ou en aval, car cela nous permettra de le diviser en deux. Ce sera également un principe fondamental de la façon dont nous affichons nos données métriques.

Le serveur d'API est un bon point de départ pour ce processus, car il nous permet de voir s'il y a un problème avec une application cliente ou avec le plan de contrôle.

Surveillance du plan de contrôle

Serveur d'API

Lorsque vous examinez notre serveur d'API, il est important de se rappeler que l'une de ses fonctions est de limiter les demandes entrantes afin d'éviter de surcharger le plan de contrôle. Ce qui peut sembler être un goulot d'étranglement au niveau du serveur d'API pourrait en fait le protéger contre des problèmes plus graves. Nous devons prendre en compte les avantages et les inconvénients de l'augmentation du volume de demandes transitant par le système. Pour déterminer si les valeurs du serveur d'API doivent être augmentées, voici un petit échantillon des points auxquels nous devons faire attention :

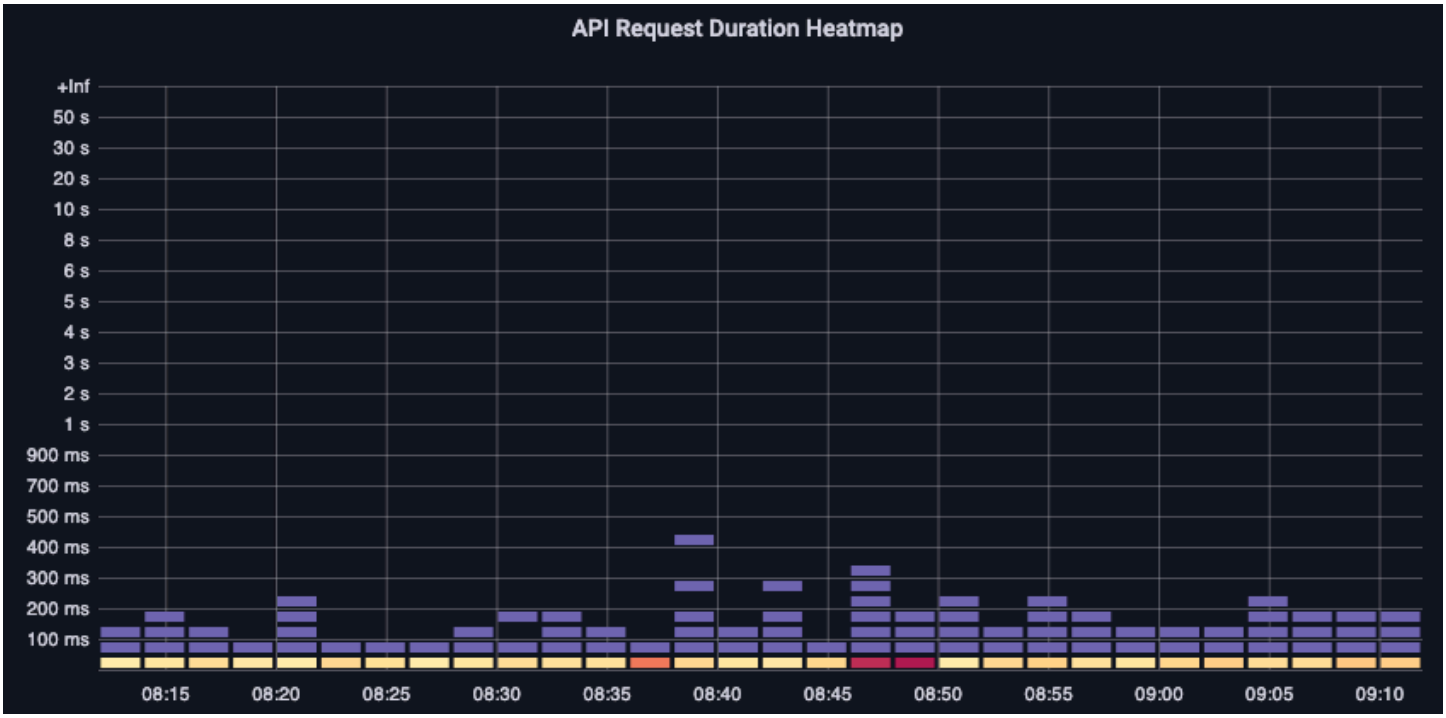
1. Quelle est la latence des demandes transitant par le système ?
2. Cette latence est-elle le serveur d'API lui-même, ou quelque chose « en aval » comme etcd ?
3. La profondeur de la file d'attente du serveur d'API contribue-t-elle à cette latence ?
4. Les files d'attente APF (API Priority and Fairness) sont-elles correctement configurées pour les modèles d'appel d'API que nous souhaitons ?

Où est le problème ?

Pour commencer, nous pouvons utiliser la métrique de latence des API pour nous donner un aperçu du temps nécessaire au serveur d'API pour traiter les demandes. Utilisons les cartes thermiques ProMQL et Grafana ci-dessous pour afficher ces données.

```
max(increase(apiserver_request_duration_seconds_bucket{subresource!
="status",subresource!="token",subresource!="scale",subresource!="/
healthz",subresource!="binding",subresource!="proxy",verb!="WATCH"}[$__rate_interval]))
 by (le)
```

> **Note**
>
> Pour une description détaillée de la façon de surveiller le serveur d'API à l'aide du tableau de bord d'API utilisé dans cet article, veuillez consulter le blog suivant



Ces demandes sont toutes traitées en moins d'une seconde, ce qui indique que le plan de contrôle traite les demandes en temps opportun. Et si ce n'était pas le cas ?

Le format que nous utilisons dans la durée de la demande d'API ci-dessus est une carte thermique. L'avantage du format heatmap, c'est qu'il nous indique la valeur du délai d'expiration de l'API par défaut (60 secondes). Cependant, ce que nous devons vraiment savoir, c'est à partir de quel seuil cette valeur devrait être préoccupante avant d'atteindre le seuil de temporisation. Pour une indication approximative des seuils acceptables, nous pouvons utiliser le SLO Kubernetes en amont, que vous pouvez trouver ici

> **Note**
>
> Vous remarquez la fonction max dans cette déclaration ? Lorsque vous utilisez des métriques qui regroupent plusieurs serveurs (par défaut, deux serveurs d'API sur EKS), il est important de ne pas faire la moyenne de ces serveurs ensemble.

Schémas de trafic asymétriques

Et si un serveur d'API [pod] était légèrement chargé et l'autre très chargé ? Si nous faisions la moyenne de ces deux chiffres ensemble, nous risquerions de mal interpréter ce qui se passait. Par exemple, nous avons ici trois serveurs d'API, mais toute la charge se trouve sur l'un de ces serveurs

d'API. En règle générale, tout ce qui possède plusieurs serveurs, tels que les serveurs etcd et API, doit être séparé en cas d'investissement pour des problèmes d'échelle et de performance.

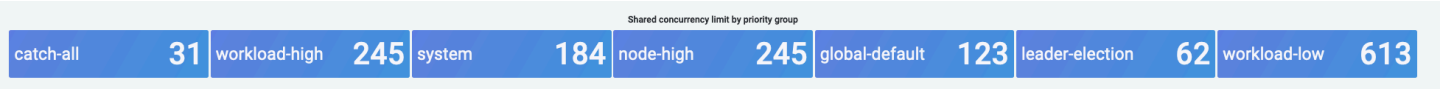
Avec le passage à la priorité et à l'équité des API, le nombre total de demandes sur le système n'est qu'un des facteurs à vérifier pour voir si le serveur d'API est surabonné. Étant donné que le système fonctionne désormais sur une série de files d'attente, nous devons vérifier si l'une de ces files d'attente est pleine et si le trafic correspondant à cette file est en train de diminuer.

Examinons ces files d'attente avec la requête suivante :

```
max without(instance)(apiserver_flowcontrol_nominal_limit_seats{})
```

> **Note**
>
> Pour plus d'informations sur le fonctionnement de l'API A&F, veuillez consulter le guide des meilleures pratiques suivant

Nous voyons ici les sept groupes de priorités différents qui apparaissent par défaut sur le cluster



Ensuite, nous voulons voir quel pourcentage de ce groupe de priorités est utilisé, afin de savoir si un certain niveau de priorité est saturé. Il peut être souhaitable de limiter les demandes au niveau le plus bas de la charge de travail, mais une baisse du niveau d'élection d'un leader ne le serait pas.

Le système API Priority and Fairness (APF) comporte un certain nombre d'options complexes, dont certaines peuvent avoir des conséquences imprévues. Un problème courant que

nous constatons sur le terrain consiste à augmenter la profondeur de la file d'attente au point d'ajouter une latence inutile. Nous pouvons surveiller ce problème en utilisant la `apiserver_flowcontrol_current_inqueue_request` métrique. Nous pouvons vérifier les baisses à l'aide du`apiserver_flowcontrol_rejected_requests_total`. Ces métriques seront une valeur différente de zéro si un bucket dépasse sa simultanéité.

L'augmentation de la profondeur de la file d'attente peut faire du serveur API une source importante de latence et doit être effectuée avec précaution. Nous vous recommandons de faire preuve de prudence en ce qui concerne le nombre de files d'attente créées. Par exemple, le nombre de partages sur un système EKS est de 600. Si nous créons trop de files d'attente, cela peut réduire le nombre de partages dans les files d'attente importantes nécessitant un débit élevé, telles que la file d'attente pour l'élection des leaders ou la file d'attente système. La création d'un trop grand nombre de files d'attente supplémentaires peut rendre plus difficile le dimensionnement correct de ces files d'attente.

Pour nous concentrer sur un simple changement impactant que vous pouvez apporter à APF, nous prenons simplement des actions provenant de compartiments sous-utilisés et nous augmentons la taille des compartiments dont l'utilisation est maximale. En redistribuant intelligemment les actions entre ces compartiments, vous pouvez réduire le risque de baisse.

Pour plus d'informations, consultez les paramètres de priorité et d'équité des API dans le guide des meilleures pratiques d'EKS.

Latence entre l'API et la latence etcd

Comment pouvons-nous utiliser le metrics/logs serveur API pour déterminer s'il y a un problème avec le serveur API, ou un problème lié au serveur API, ou une combinaison des deux ? upstream/downstream Pour mieux comprendre cela, voyons comment le serveur API et etcd peuvent être liés, et comment il peut être facile de dépanner le mauvais système.

Dans le graphique ci-dessous, nous voyons la latence du serveur API, mais nous constatons également qu'une grande partie de cette latence est corrélée au serveur etcd en raison des barres du

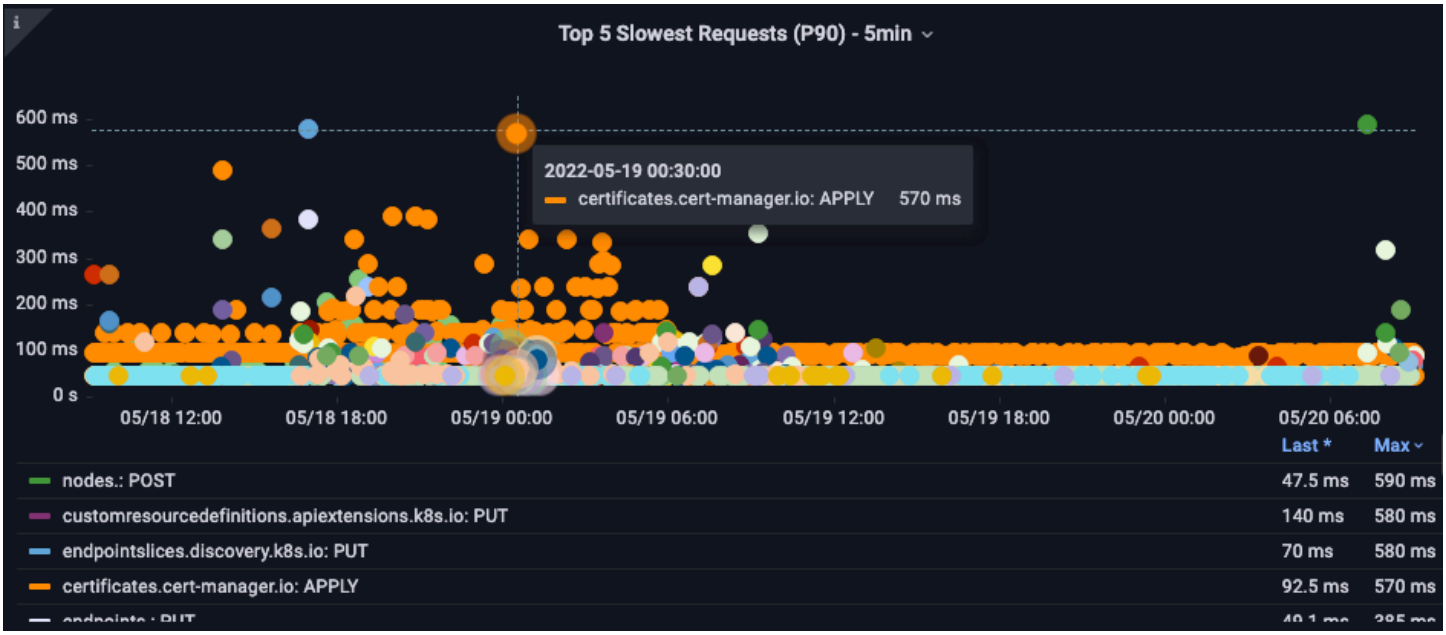
graphique indiquant la majeure partie de la latence au niveau etcd. S'il y a 15 secondes de latence etcd en même temps que 20 secondes de latence du serveur API, alors la majeure partie de la latence se situe en fait au niveau etcd.

En examinant l'ensemble du flux, nous constatons qu'il est sage de ne pas se concentrer uniquement sur le serveur d'API, mais de rechercher également les signaux indiquant qu'etcd est soumis à des contraintes (c'est-à-dire que les compteurs d'application lente augmentent). Pouvoir accéder rapidement à la bonne zone problématique d'un simple coup d'œil est ce qui rend un tableau de bord puissant.

> **Note**
>
> Le tableau de bord présenté dans la section Troubleshooter.json se trouve à l'adresse https://github.com/RiskyAdventure/ Dashboards/blob/main/api Troubleshooter.json

Problèmes liés au plan de contrôle et problèmes liés au client

Dans ce graphique, nous recherchons les appels d'API dont l'exécution a pris le plus de temps au cours de cette période. Dans ce cas, nous voyons qu'une ressource personnalisée (CRD) appelle une fonction APPLY qui est l'appel le plus latent pendant la période 05:40.



Grâce à ces données, nous pouvons utiliser une requête ProMQL ad hoc ou CloudWatch Insights pour extraire les requêtes LIST du journal d'audit pendant cette période afin de voir de quelle application il s'agit.

Trouver la source avec CloudWatch

Il est préférable d'utiliser les métriques pour identifier le problème que nous voulons examiner et pour affiner à la fois le délai et les paramètres de recherche du problème. Une fois que nous aurons ces données, nous souhaitons passer aux journaux pour obtenir des informations plus détaillées sur les heures et les erreurs. Pour ce faire, nous transformerons nos journaux en indicateurs à l'aide de CloudWatch Logs Insights.

Par exemple, pour étudier le problème ci-dessus, nous utiliserons la requête CloudWatch Logs Insights suivante pour extraire UserAgent et RequestURI afin de déterminer quelle application est à l'origine de cette latence.

> **Note**
>
> Un nombre approprié doit être utilisé pour ne pas entraîner un List/Resync comportement normal sur une montre.

```
fields @timestamp, @message
```

```
| filter @logStream like "kube-apiserver-audit"
| filter ispresent(requestURI)
| filter verb = "list"
| parse requestReceivedTimestamp /\d+-\d+-(?<StartDay>\d+)T(?<StartHour>\d+):(?
<StartMinute>\d+):(?<StartSec>\d+).(?<StartMsec>\d+)Z/
| parse stageTimestamp /\d+-\d+-(?<EndDay>\d+)T(?<EndHour>\d+):(?<EndMinute>\d+):(?
<EndSec>\d+).(?<EndMsec>\d+)Z/
| fields (StartHour * 3600 + StartMinute * 60 + StartSec + StartMsec / 1000000) as
 StartTime, (EndHour * 3600 + EndMinute * 60 + EndSec + EndMsec / 1000000) as EndTime,
 (EndTime - StartTime) as DeltaTime
| stats avg(DeltaTime) as AverageDeltaTime, count(*) as CountTime by requestURI,
 userAgent
| filter CountTime >=50
| sort AverageDeltaTime desc
```

À l'aide de cette requête, nous avons trouvé deux agents différents exécutant un grand nombre d'opérations de liste à latence élevée. Splunk et CloudWatch agent. Forts de ces données, nous pouvons prendre la décision de supprimer, de mettre à jour ou de remplacer ce contrôleur par un autre projet.

requestURI	AverageDeltaTime	AverageD	CountTime
/api/v1/pods	rest-client/2.1.0 (linux-gnu x86_64) ruby/2.5.5p157	9.1395	87
/api/v1/pods?limit=500&resourceVersion=0	amazon-cloudwatch-agent/v0.0.0 (linux/amd64) kubernetes/$Format	5.3976	929

> **Note**
>
> Pour plus de détails à ce sujet, veuillez consulter le blog suivant

Planificateur

Étant donné que les instances du plan de contrôle EKS sont exécutées sur un compte AWS distinct, nous ne serons pas en mesure de supprimer ces composants pour les métriques (le serveur d'API étant l'exception). Cependant, étant donné que nous avons accès aux journaux d'audit de ces composants, nous pouvons transformer ces journaux en métriques pour déterminer si l'un des sous-systèmes est à l'origine d'un goulot d'étranglement en matière de mise à l'échelle. Utilisons CloudWatch Logs Insights pour voir combien de pods non planifiés se trouvent dans la file d'attente du planificateur.

Pods non planifiés dans le journal du planificateur

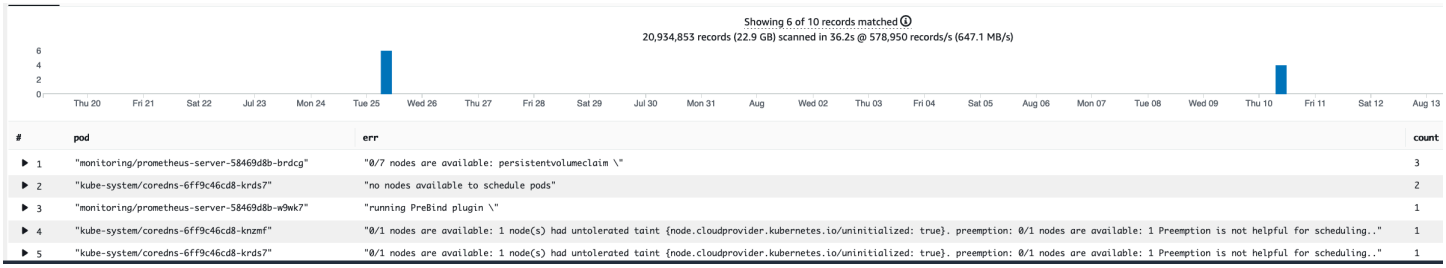
Si nous avions accès à l'extraction des métriques du planificateur directement sur un Kubernetes autogéré (tel que Kops), nous utiliserions le ProMQL suivant pour comprendre le backlog du planificateur.

```
max without(instance)(scheduler_pending_pods)
```

Comme nous n'avons pas accès à la métrique ci-dessus dans EKS, nous utiliserons la requête CloudWatch Logs Insights ci-dessous pour voir le backlog en vérifiant le nombre de pods qui n'ont pas pu être déplanifiés au cours d'une période donnée. Nous pourrions ensuite approfondir les messages aux heures de pointe afin de comprendre la nature du goulot d'étranglement. Par exemple, les nœuds ne tournent pas assez vite ou le limiteur de débit dans le planificateur lui-même.

```
fields timestamp, pod, err, @message
| filter @logStream like "scheduler"
| filter @message like "Unable to schedule pod"
| parse @message  /^.(?<date>\d{4})\s+(?<timestamp>\d+:\d+:\d+\.\d+)\s+\S*\s+\S+\]\s
\"(.*?)\"\s+pod=(?<pod>\"(.*?)\")\s+err=(?<err>\"(.*?)\")/
| count(*) as count by pod, err
| sort count desc
```

Nous voyons ici les erreurs du planificateur indiquant que le pod n'a pas été déployé car le PVC de stockage n'était pas disponible.



#	pod	err	count
▶ 1	"monitoring/prometheus-server-58469d8b-brdcg"	"0/7 nodes are available: persistentvolumeclaim \"	3
▶ 2	"kube-system/coredns-6ff9c46cd8-krds7"	"no nodes available to schedule pods"	2
▶ 3	"monitoring/prometheus-server-58469d8b-w9wk7"	"running PreBind plugin \"	1
▶ 4	"kube-system/coredns-6ff9c46cd8-knzmf"	"0/1 nodes are available: 1 node(s) had untolerated taint {node.cloudprovider.kubernetes.io/uninitialized: true}. preemption: 0/1 nodes are available: 1 Preemption is not helpful for scheduling.."	1
▶ 5	"kube-system/coredns-6ff9c46cd8-krds7"	"0/1 nodes are available: 1 node(s) had untolerated taint {node.cloudprovider.kubernetes.io/uninitialized: true}. preemption: 0/1 nodes are available: 1 Preemption is not helpful for scheduling.."	1

> **Note**
>
> La journalisation des audits doit être activée sur le plan de contrôle pour activer cette fonction. Il est également recommandé de limiter la conservation des journaux afin de ne pas augmenter inutilement les coûts au fil du temps. Voici un exemple d'activation de toutes les fonctions de journalisation à l'aide de l'outil EKSCTL.

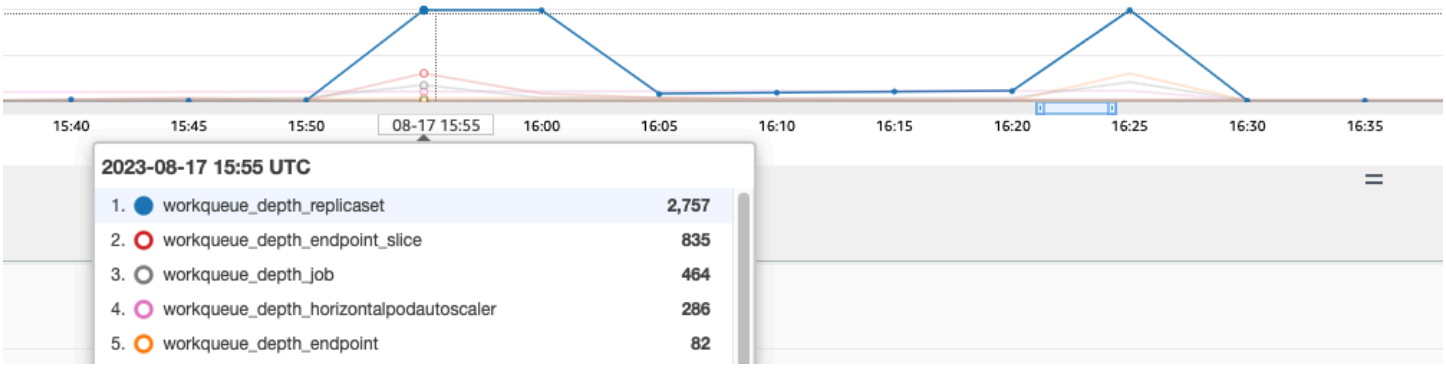
```
cloudWatch:
  clusterLogging:
    enableTypes: ["*"]
    logRetentionInDays: 10
```

Gestionnaire de contrôleurs Kube

Kube Controller Manager, comme tous les autres contrôleurs, impose des limites quant au nombre d'opérations qu'il peut effectuer simultanément. Passons en revue certains de ces indicateurs en examinant une configuration KOPS dans laquelle nous pouvons définir ces paramètres.

```
  kubeControllerManager:
    concurrentEndpointSyncs: 5
    concurrentReplicasetSyncs: 5
    concurrentNamespaceSyncs: 10
    concurrentServiceaccountTokenSyncs: 5
    concurrentServiceSyncs: 5
    concurrentResourceQuotaSyncs: 5
    concurrentGcSyncs: 20
    kubeAPIBurst: 30
    kubeAPIQPS: "20"
```

Ces contrôleurs ont des files d'attente qui se remplissent lorsque le taux de désabonnement d'un cluster est élevé. Dans ce cas, nous constatons que le contrôleur Replicaset Set a un important backlog dans sa file d'attente.



Nous avons deux manières différentes de faire face à une telle situation. Si nous utilisions l'autogestion, nous pourrions simplement augmenter le nombre de goroutines simultanées, mais cela aurait un impact sur etcd en traitant davantage de données dans le KCM. L'autre option serait de

réduire le nombre d'objets replicaset utilisés lors du déploiement afin de réduire le nombre d'objets replicaset que nous pouvons annuler, réduisant ainsi la pression `.spec.revisionHistoryLimit` sur ce contrôleur.

```
spec:
  revisionHistoryLimit: 2
```

Les autres fonctionnalités de Kubernetes peuvent être ajustées ou désactivées pour réduire la pression dans les systèmes à taux de désabonnement élevé. Par exemple, si l'application de nos pods n'a pas besoin de parler directement à l'API k8s, la désactivation du secret projeté dans ces pods diminuera la charge. ServiceaccountTokenSyncs C'est le moyen le plus souhaitable de régler ces problèmes si possible.

```
kind: Pod
spec:
  automountServiceAccountToken: false
```

Dans les systèmes où nous ne pouvons pas accéder aux métriques, nous pouvons à nouveau consulter les journaux pour détecter les conflits. Si nous voulions connaître le nombre de demandes traitées par contrôleur ou au niveau agrégé, nous utiliserions la requête CloudWatch Logs Insights suivante.

Volume total traité par le KCM

```
# Query to count API qps coming from kube-controller-manager, split by controller type.
# If you're seeing values close to 20/sec for any particular controller, it's most
 likely seeing client-side API throttling.
fields @timestamp, @logStream, @message
| filter @logStream like /kube-apiserver-audit/
| filter userAgent like /kube-controller-manager/
# Exclude lease-related calls (not counted under kcm qps)
| filter requestURI not like "apis/coordination.k8s.io/v1/namespaces/kube-system/
leases/kube-controller-manager"
# Exclude API discovery calls (not counted under kcm qps)
| filter requestURI not like "?timeout=32s"
# Exclude watch calls (not counted under kcm qps)
| filter verb != "watch"
# If you want to get counts of API calls coming from a specific controller, uncomment
 the appropriate line below:
# | filter user.username like "system:serviceaccount:kube-system:job-controller"
# | filter user.username like "system:serviceaccount:kube-system:cronjob-controller"
```

```
# | filter user.username like "system:serviceaccount:kube-system:deployment-controller"
# | filter user.username like "system:serviceaccount:kube-system:replicaset-controller"
# | filter user.username like "system:serviceaccount:kube-system:horizontal-pod-
autoscaler"
# | filter user.username like "system:serviceaccount:kube-system:persistent-volume-
binder"
# | filter user.username like "system:serviceaccount:kube-system:endpointslice-
controller"
# | filter user.username like "system:serviceaccount:kube-system:endpoint-controller"
# | filter user.username like "system:serviceaccount:kube-system:generic-garbage-
controller"
| stats count(*) as count by user.username
| sort count desc
```

Lorsque l'on examine les problèmes d'évolutivité, il est essentiel d'examiner chaque étape du processus (API, planificateur, KCM, etc.) avant de passer à la phase de dépannage détaillée. En production, vous constaterez souvent qu'il faut ajuster plusieurs parties de Kubernetes pour que le système fonctionne de manière optimale. Il est facile de résoudre par inadvertance ce qui n'est qu'un symptôme (tel qu'un délai d'expiration d'un nœud) dû à un goulot d'étranglement beaucoup plus important.

ETCD

etcd utilise un fichier mappé en mémoire pour stocker efficacement les paires clé-valeur. Il existe un mécanisme de protection permettant de définir la taille de cet espace mémoire disponible, généralement défini aux limites de 2, 4 et 8 Go. La diminution du nombre d'objets dans la base de données signifie moins de nettoyage à effectuer par Etcd lorsque les objets sont mis à jour et que les anciennes versions doivent être nettoyées. Ce processus de nettoyage des anciennes versions d'un objet est appelé compactage. Après un certain nombre d'opérations de compactage, un processus ultérieur permet de récupérer de l'espace utilisable, appelé défragmentation, qui se produit au-dessus d'un certain seuil ou selon un calendrier fixe.

Nous pouvons prendre quelques mesures relatives aux utilisateurs pour limiter le nombre d'objets dans Kubernetes et ainsi réduire l'impact du processus de compactage et de défragmentation. Par exemple, Helm garde un high`revisionHistoryLimit`. Cela permet de conserver les anciens objets, tels que ReplicaSets ceux du système, pour pouvoir effectuer des annulations. En fixant les limites d'historique à 2, nous pouvons réduire le nombre d'objets (par exemple ReplicaSets) de dix à deux, ce qui allégerait la charge du système.

```
apiVersion: apps/v1
```

```
kind: Deployment
spec:
  revisionHistoryLimit: 2
```

Du point de vue de la surveillance, si les pics de latence du système se produisent selon un schéma défini, séparés par des heures, il peut être utile de vérifier si ce processus de défragmentation en est la source. Nous pouvons le constater en utilisant CloudWatch Logs.

Si vous souhaitez connaître les start/end heures de défragmentation, utilisez la requête suivante :

```
fields @timestamp, @message
| filter @logStream like /etcd-manager/
| filter @message like /defraging|defraged/
| sort @timestamp asc
```

@message	@logStream
Oct 6 19:19:52 ip-172-16-176-9 etcdnanny: {"level":"info","ts":"2022-10-06T19:19:52.826Z","caller":"defrag/defrag.go:312","msg":"defraging","…	etcd-manager-i-0101a20d88e74fa5b
Oct 6 19:20:28 ip-172-16-176-9 etcdnanny: {"level":"info","ts":"2022-10-06T19:20:28.836Z","caller":"defrag/defrag.go:322","msg":"defraged"}	etcd-manager-i-0101a20d88e74fa5b
Oct 6 19:25:57 ip-172-16-101-191 etcdnanny: {"level":"info","ts":"2022-10-06T19:25:57.650Z","caller":"defrag/defrag.go:312","msg":"defraging"…	etcd-manager-i-0e3aa5c2ae801af44
Oct 6 19:26:34 ip-172-16-101-191 etcdnanny: {"level":"info","ts":"2022-10-06T19:26:34.457Z","caller":"defrag/defrag.go:322","msg":"defraged"}	etcd-manager-i-0e3aa5c2ae801af44
Oct 6 19:32:26 ip-172-16-50-136 etcdnanny: {"level":"info","ts":"2022-10-06T19:32:26.592Z","caller":"defrag/defrag.go:312","msg":"defraging",…	etcd-manager-i-025ebb6b463ee0f21
Oct 6 19:33:09 ip-172-16-50-136 etcdnanny: {"level":"info","ts":"2022-10-06T19:33:09.575Z","caller":"defrag/defrag.go:322","msg":"defraged"}	etcd-manager-i-025ebb6b463ee0f21

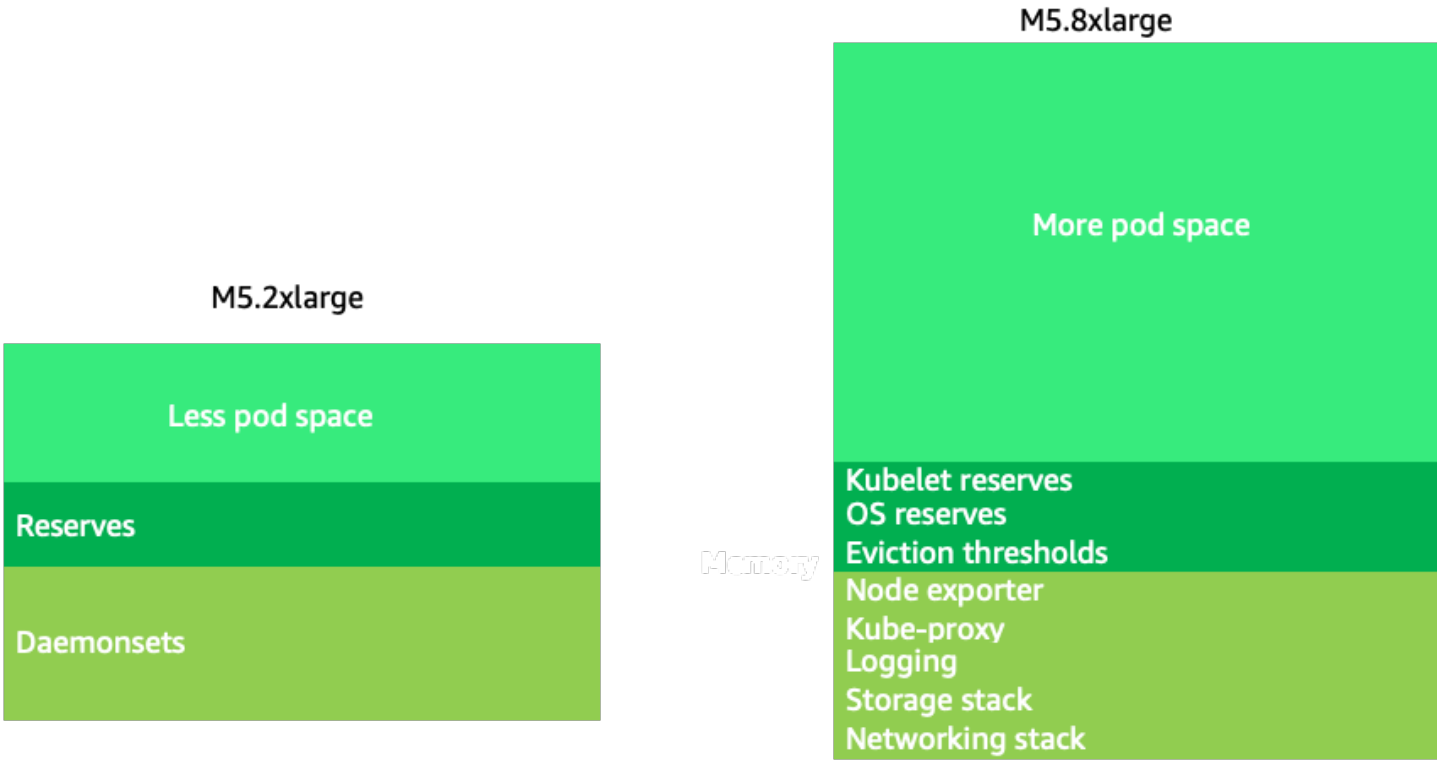
Efficacité des nœuds et des charges de travail

L'efficacité de nos charges de travail et de nos nœuds permet de réduire complexity/cost tout en augmentant les performances et l'évolutivité. De nombreux facteurs doivent être pris en compte lors de la planification de cette efficacité, et il est plus facile de penser en termes de compromis par rapport à un paramètre de meilleures pratiques pour chaque fonctionnalité. Examinons ces compromis en profondeur dans la section suivante.

Sélection du nœud

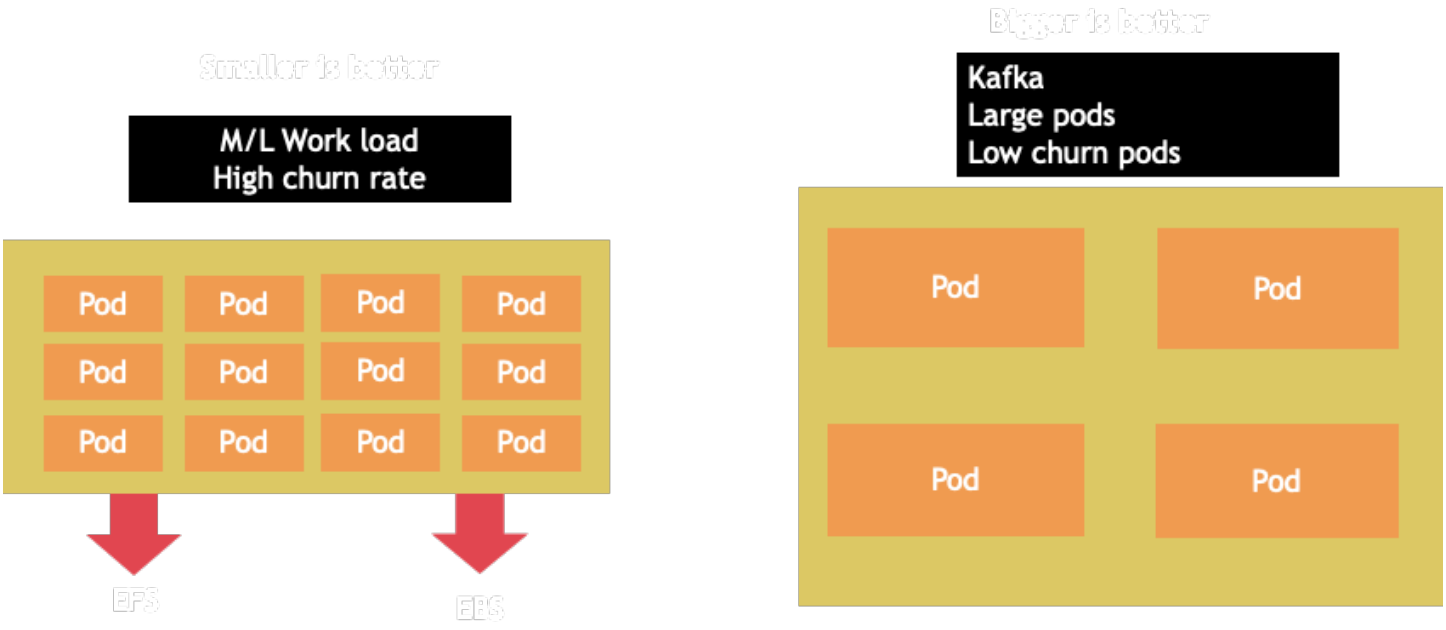
L'utilisation de nœuds légèrement plus grands (4 à 12 fois plus grands) augmente l'espace disponible pour faire fonctionner les pods, car cela réduit le pourcentage du nœud utilisé pour les « frais généraux », tels que les réserves pour DaemonSetsles composants du système. Dans le schéma ci-dessous, nous voyons la différence entre l'espace utilisable sur un système 2xlarge par rapport à un système 8xlarge avec juste un nombre modéré de. DaemonSets

> **Note**
>
> Étant donné que k8s évolue horizontalement en règle générale, pour la plupart des applications, il n'est pas logique de prendre en compte l'impact sur les performances des nœuds de taille NUMA, d'où la recommandation d'une plage inférieure à cette taille de nœud.



Les nœuds de grande taille nous permettent d'avoir un pourcentage plus élevé d'espace utilisable par nœud. Cependant, ce modèle peut être poussé à l'extrême en emballant le nœud avec un tel nombre de pods qu'il provoque des erreurs ou sature le nœud. La surveillance de la saturation des nœuds est essentielle pour utiliser avec succès des nœuds de plus grande taille.

La sélection de nœuds est rarement une one-size-fits-all proposition. Il est souvent préférable de répartir les charges de travail présentant des taux de désabonnement radicalement différents entre différents groupes de nœuds. Les petites charges de travail par lots présentant un taux de désabonnement élevé seraient mieux prises en charge par la famille d'instances 4xlarge, tandis qu'une application à grande échelle telle que Kafka, qui utilise 8 vCPU et présente un faible taux de désabonnement, serait mieux servie par la famille 12xlarge.



> **Note**
>
> Un autre facteur à prendre en compte dans le cas de nœuds de très grande taille est que les CGROUPS ne cachent pas le nombre total de vCPU à l'application conteneurisée. Les environnements d'exécution dynamiques peuvent souvent générer un nombre involontaire de threads du système d'exploitation, ce qui crée une latence difficile à résoudre. Pour ces applications, l'épinglage du processeur est recommandé. Pour une exploration plus approfondie du sujet, veuillez consulter la vidéo https://www.youtube.com suivante/watch ? v = _ NqtfDy KAqg

Emballage Node Bin

Règles entre Kubernetes et Linux

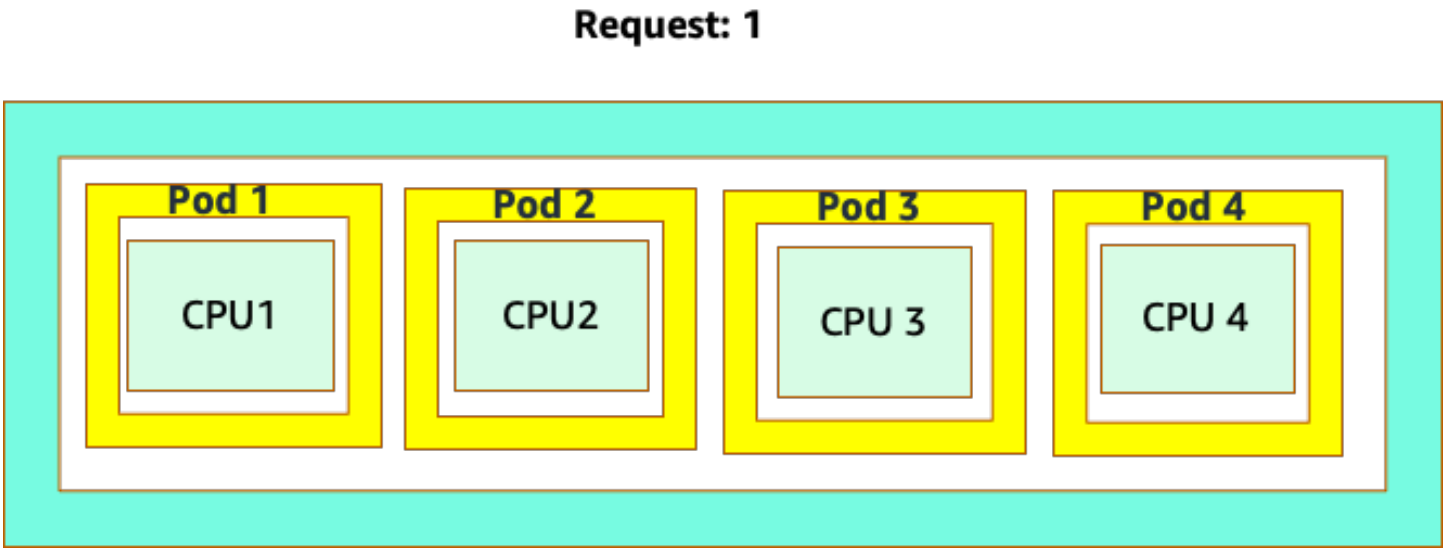
Nous devons tenir compte de deux ensembles de règles lorsque nous traitons des charges de travail sur Kubernetes. Les règles du planificateur Kubernetes, qui utilise la valeur de la requête pour planifier les pods sur un nœud, puis ce qui se passe une fois le pod planifié, relèvent du domaine de Linux, et non de Kubernetes.

Une fois le planificateur Kubernetes terminé, un nouvel ensemble de règles prend le relais, le Linux Completely Fair Scheduler (CFS). Le principal point à retenir est que Linux CFS n'a pas le concept

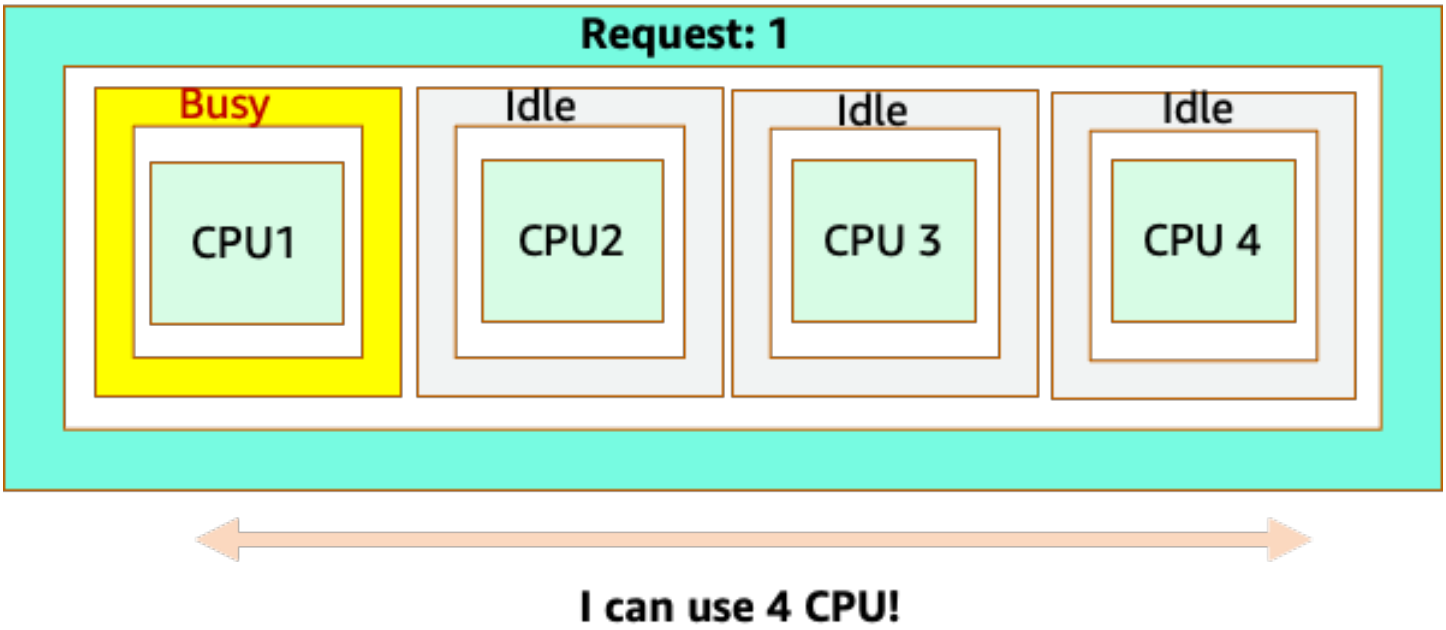
de noyau. Nous expliquerons pourquoi une approche centrée sur les cœurs peut entraîner des problèmes majeurs liés à l'optimisation des charges de travail en vue de leur mise à l'échelle.

Penser en fonction des cœurs

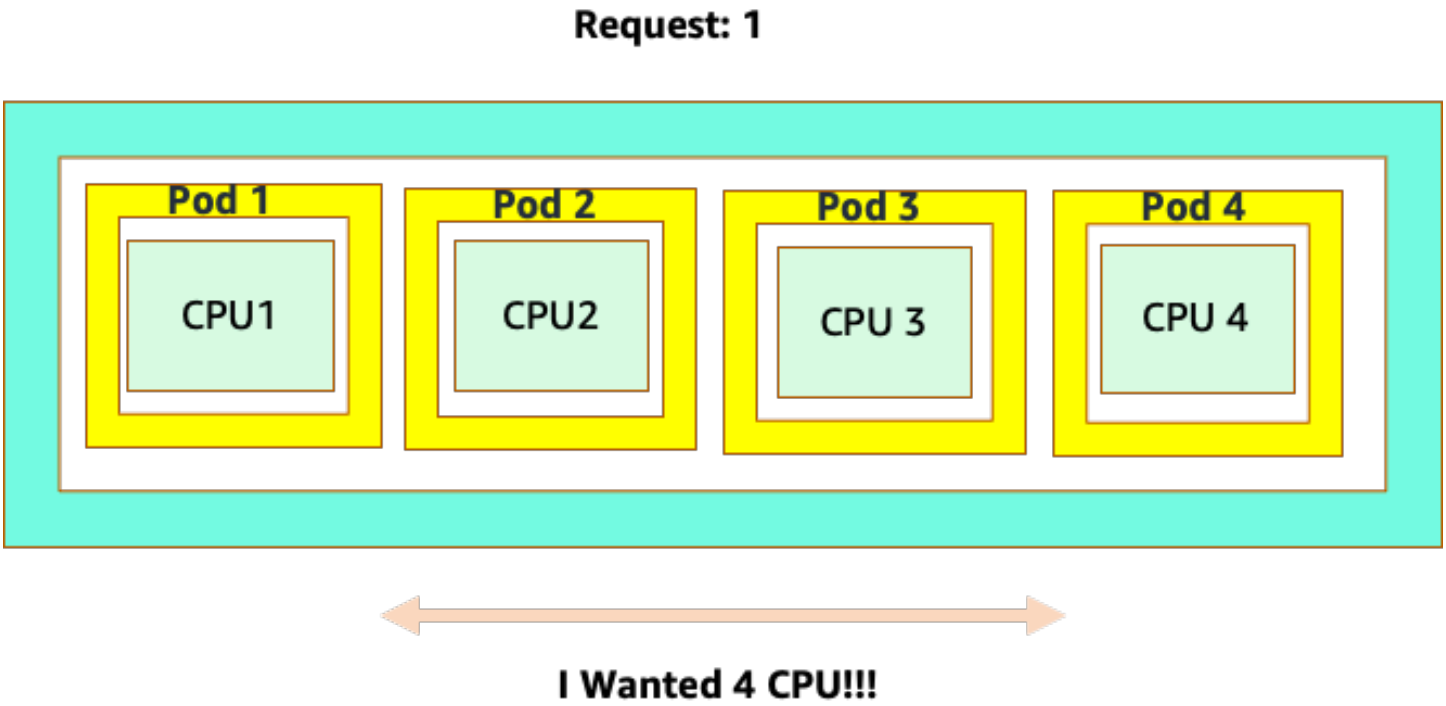
La confusion commence parce que le planificateur Kubernetes utilise le concept de cœurs. Du point de vue du planificateur Kubernetes, si nous examinions un nœud avec 4 pods NGINX, chacun avec une demande d'un ensemble de cœurs, le nœud ressemblerait à ceci.



Cependant, faisons un essai de réflexion pour voir en quoi cela est différent du point de vue de Linux CFS. La chose la plus importante à retenir lorsque vous utilisez le système Linux CFS est que les conteneurs occupés (CGROUPS) sont les seuls conteneurs pris en compte dans le système de partage. Dans ce cas, seul le premier conteneur est occupé, il est donc autorisé à utiliser les 4 cœurs du nœud.



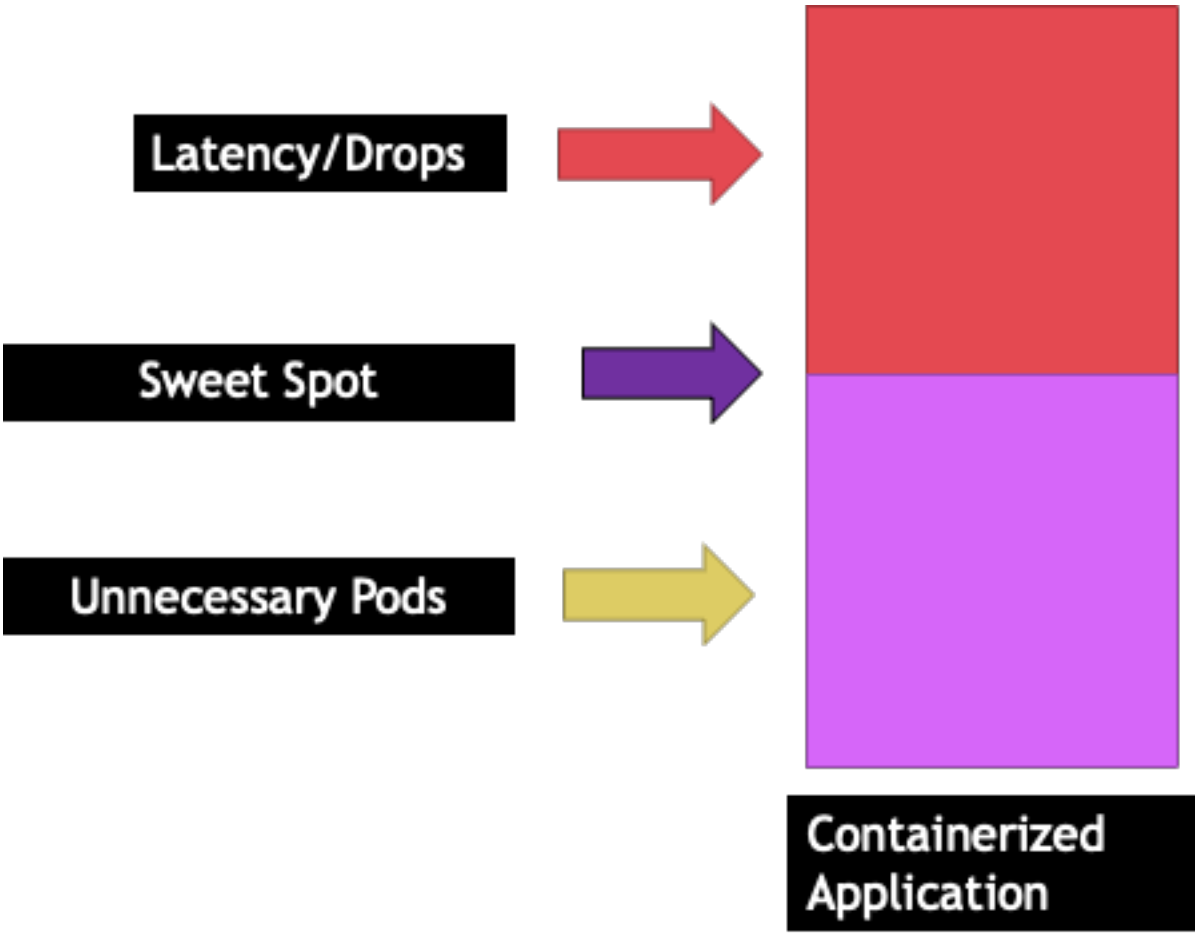
Pourquoi est-ce important ? Supposons que nous ayons effectué nos tests de performance dans un cluster de développement où une application NGINX était le seul conteneur occupé sur ce nœud. Lorsque nous passons l'application en production, voici ce qui se produit : l'application NGINX a besoin de 4 vCPU de ressources, mais comme tous les autres pods du nœud sont occupés, les performances de notre application sont limitées.



Cette situation nous conduirait à ajouter de nouveaux conteneurs inutilement, car nous ne permettions pas à nos applications d'atteindre leur « point idéal ». Explorons ce concept important d'un peu plus `"sweet spot"` en détail.

Taille adaptée à l'application

Chaque application a un certain point où elle ne peut plus supporter de trafic. Le dépassement de ce point peut augmenter les délais de traitement et même réduire le trafic lorsqu'il est poussé bien au-delà de ce point. C'est ce que l'on appelle le point de saturation de l'application. Pour éviter les problèmes de mise à l'échelle, nous devons essayer de redimensionner l'application avant qu'elle n'atteigne son point de saturation. Appelons ce point le point idéal.



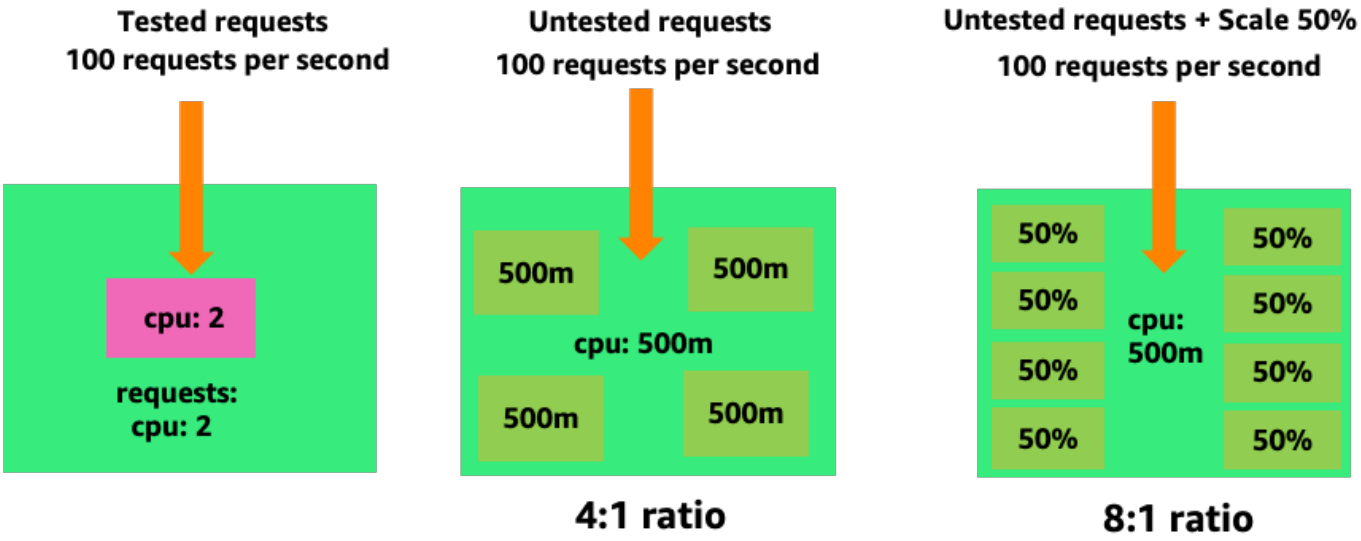
Nous devons tester chacune de nos applications pour comprendre leur point idéal. Il n'y aura pas de directives universelles ici car chaque application est différente. Au cours de ces tests, nous essayons de comprendre la meilleure métrique qui indique le point de saturation de nos applications. Souvent, les métriques d'utilisation sont utilisées pour indiquer qu'une application est saturée, mais cela peut rapidement entraîner des problèmes de dimensionnement (nous aborderons cette rubrique en détail

dans une section ultérieure). Une fois que nous aurons atteint ce « point idéal », nous pourrons l'utiliser pour augmenter efficacement nos charges de travail.

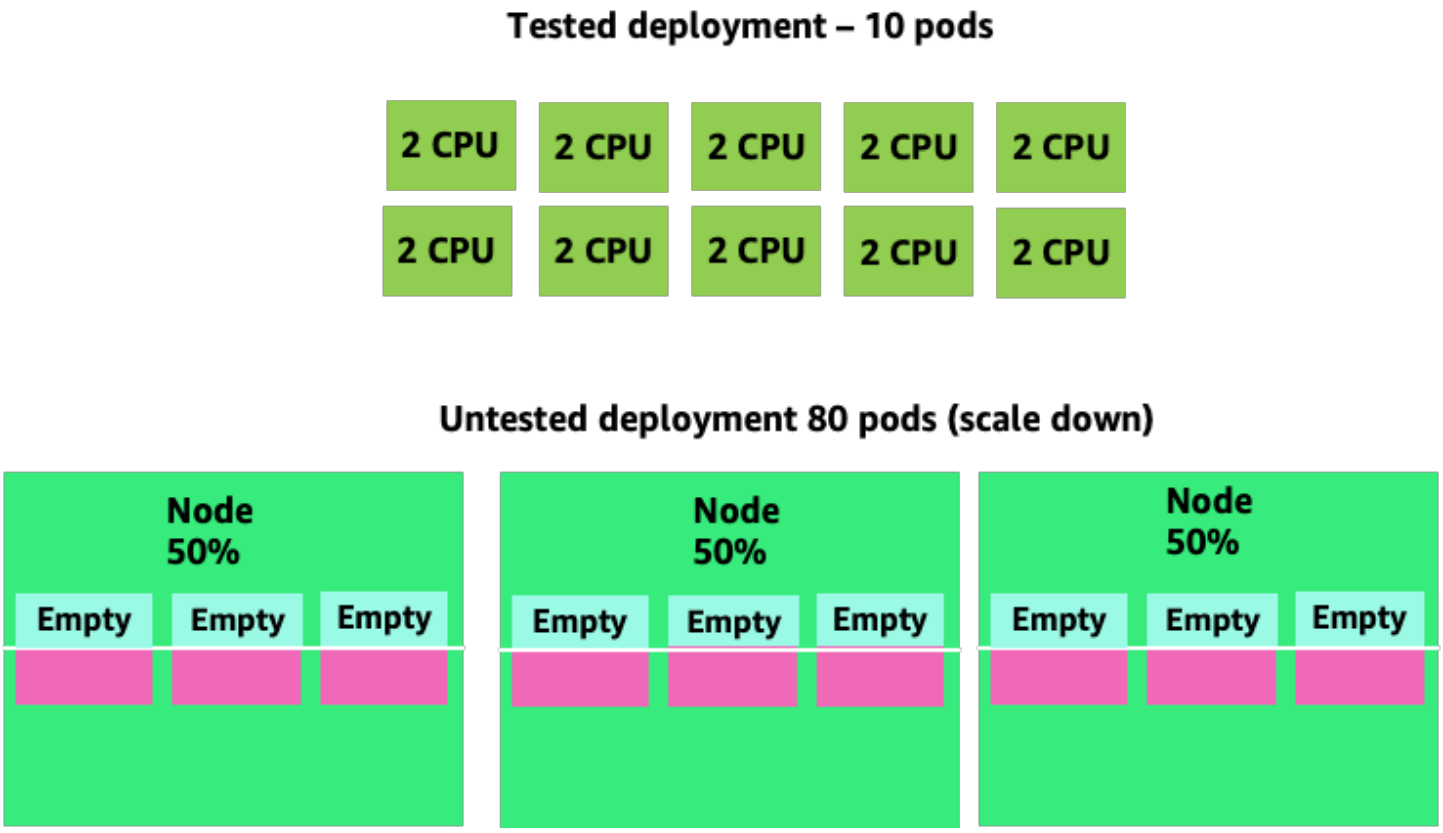
À l'inverse, que se passerait-il si nous agissions bien avant le point idéal et que nous créions des modules inutiles ? Découvrons-le dans la section suivante.

Étalement de la nacelle

Pour voir comment la création de modules inutiles peut rapidement devenir incontrôlable, examinons le premier exemple sur la gauche. L'échelle verticale correcte de ce conteneur nécessite environ deux volts CPUs d'utilisation pour traiter 100 demandes par seconde. Cependant, si nous sous-provisionnions la valeur des demandes en définissant les demandes sur un demi-cœur, nous aurions désormais besoin de 4 modules pour chaque module dont nous avons réellement besoin. Ce problème est encore aggravé par le fait que si notre HPA était défini par défaut sur 50 % du processeur, ces pods deviendraient à moitié vides, ce qui créerait un ratio de 8:1.

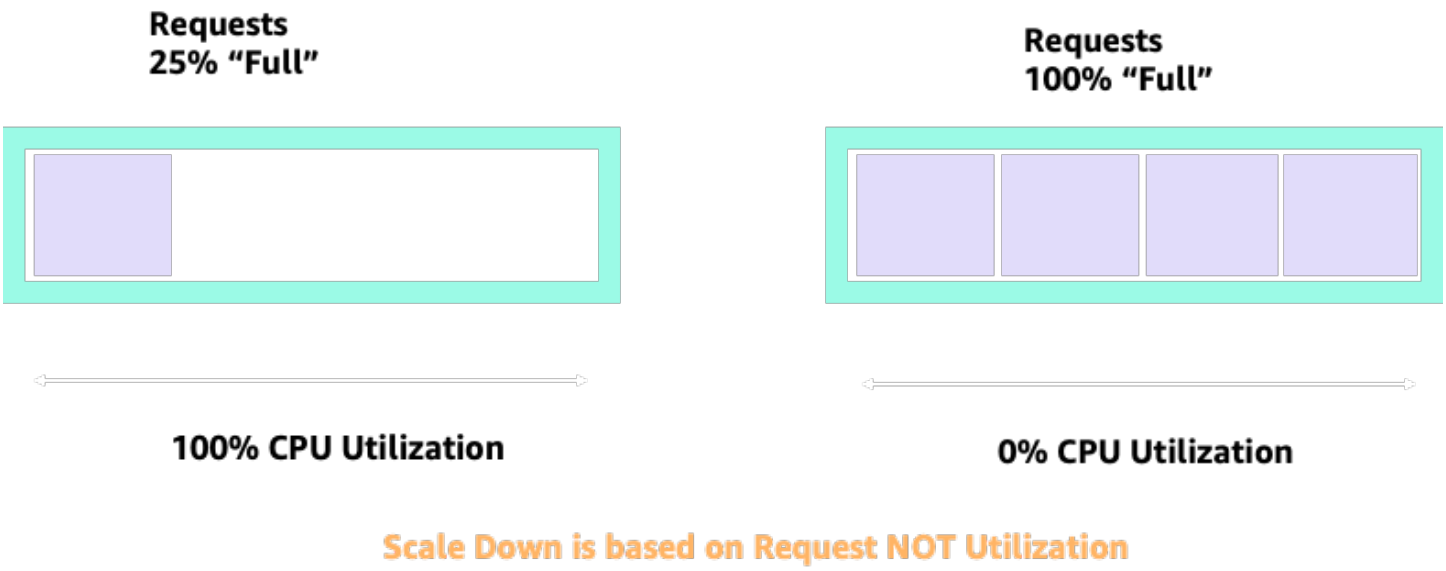


En aggravant ce problème, nous pouvons rapidement voir comment cela peut devenir incontrôlable. Un déploiement de dix modules dont le point idéal a été mal défini pourrait rapidement atteindre 80 modules et l'infrastructure supplémentaire nécessaire pour les faire fonctionner.



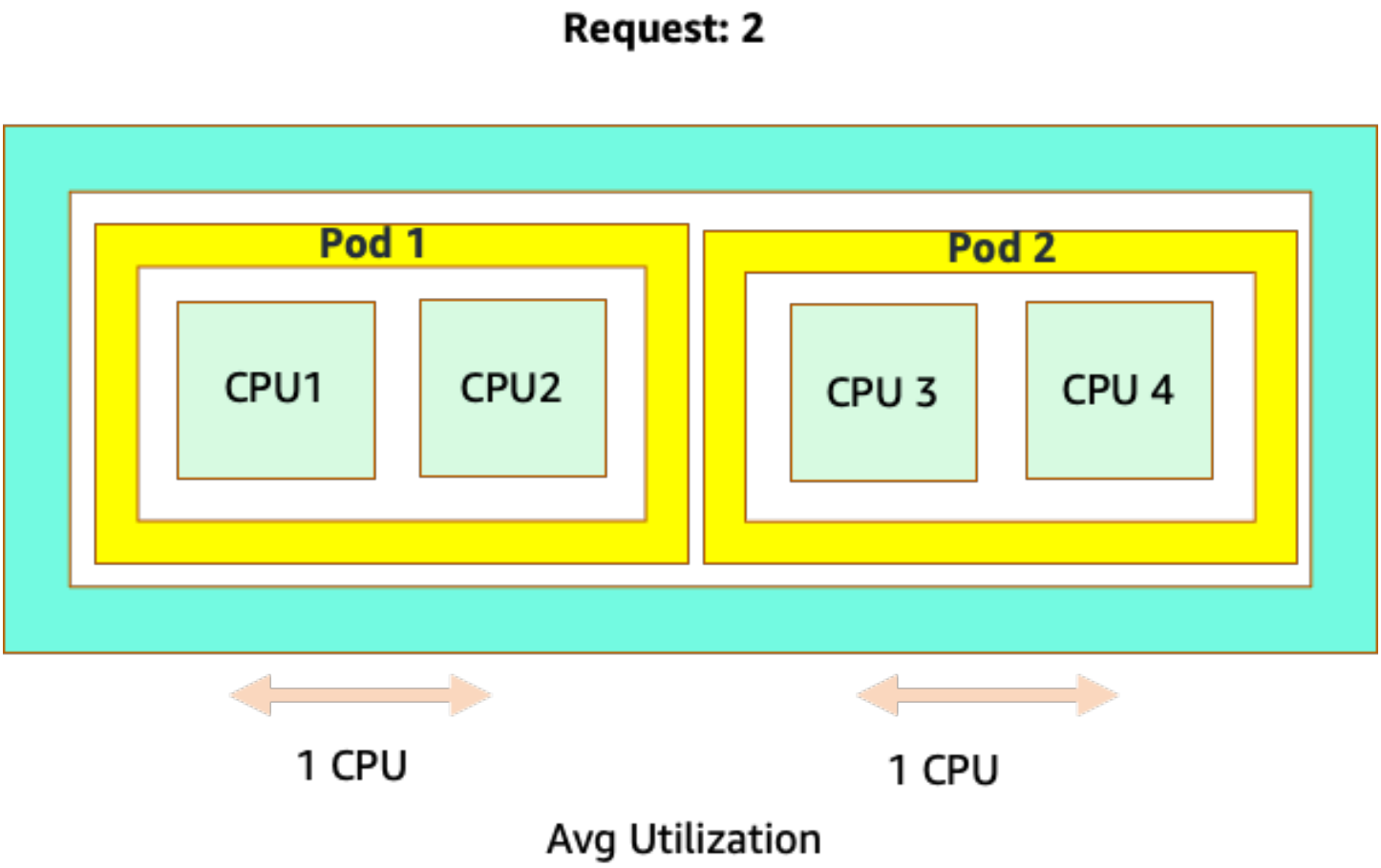
Maintenant que nous comprenons l'impact de ne pas autoriser les applications à fonctionner dans leur zone idéale, revenons au niveau des nœuds et demandons-nous pourquoi cette différence entre le planificateur Kubernetes et Linux CFS est si importante.

Lorsque nous augmentons ou diminuons avec HPA, nous pouvons avoir un scénario dans lequel nous disposons de beaucoup d'espace pour allouer plus de pods. Ce serait une mauvaise décision car le nœud représenté sur la gauche utilise déjà 100 % du processeur. Dans un scénario irréaliste mais théoriquement possible, nous pourrions avoir l'autre extrême où notre nœud est complètement plein, alors que l'utilisation de notre processeur est nulle.

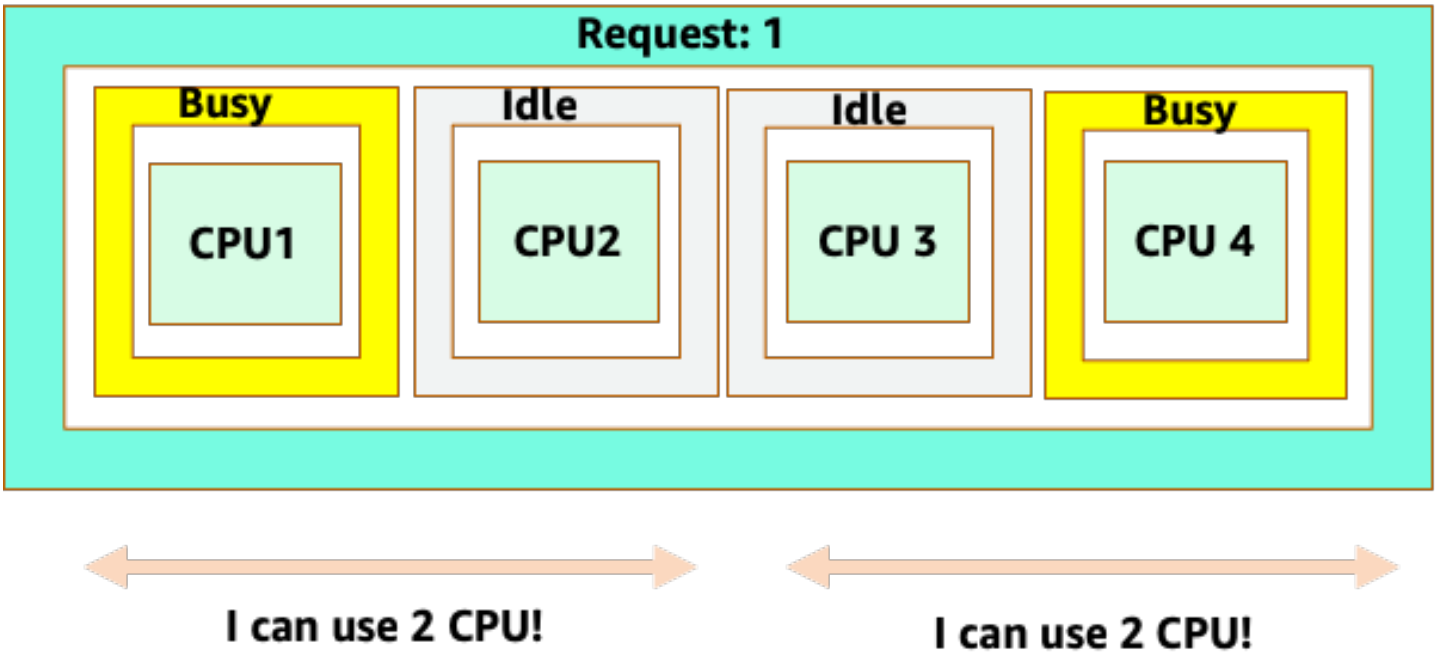


Configuration des demandes

Il serait tentant de définir la demande à la valeur « idéale » pour cette application, mais cela entraînerait des inefficiences, comme le montre le schéma ci-dessous. Ici, nous avons défini la valeur de la requête sur 2 vCPU, mais l'utilisation moyenne de ces pods ne fait fonctionner qu'un seul processeur la plupart du temps. Ce paramètre nous ferait perdre 50 % de nos cycles de processeur, ce qui serait inacceptable.



Cela nous amène à la réponse complexe au problème. L'utilisation des conteneurs ne peut être envisagée en vase clos ; il faut tenir compte des autres applications exécutées sur le nœud. Dans l'exemple suivant, les conteneurs surchargés sont mélangés à deux conteneurs à faible utilisation du processeur qui peuvent être limités en mémoire. De cette façon, nous permettons aux conteneurs d'atteindre leur point idéal sans surcharger le nœud.



Le concept important à retenir de tout cela est que l'utilisation du concept de cœurs du planificateur Kubernetes pour comprendre les performances des conteneurs Linux peut entraîner de mauvaises décisions, car ils ne sont pas liés.

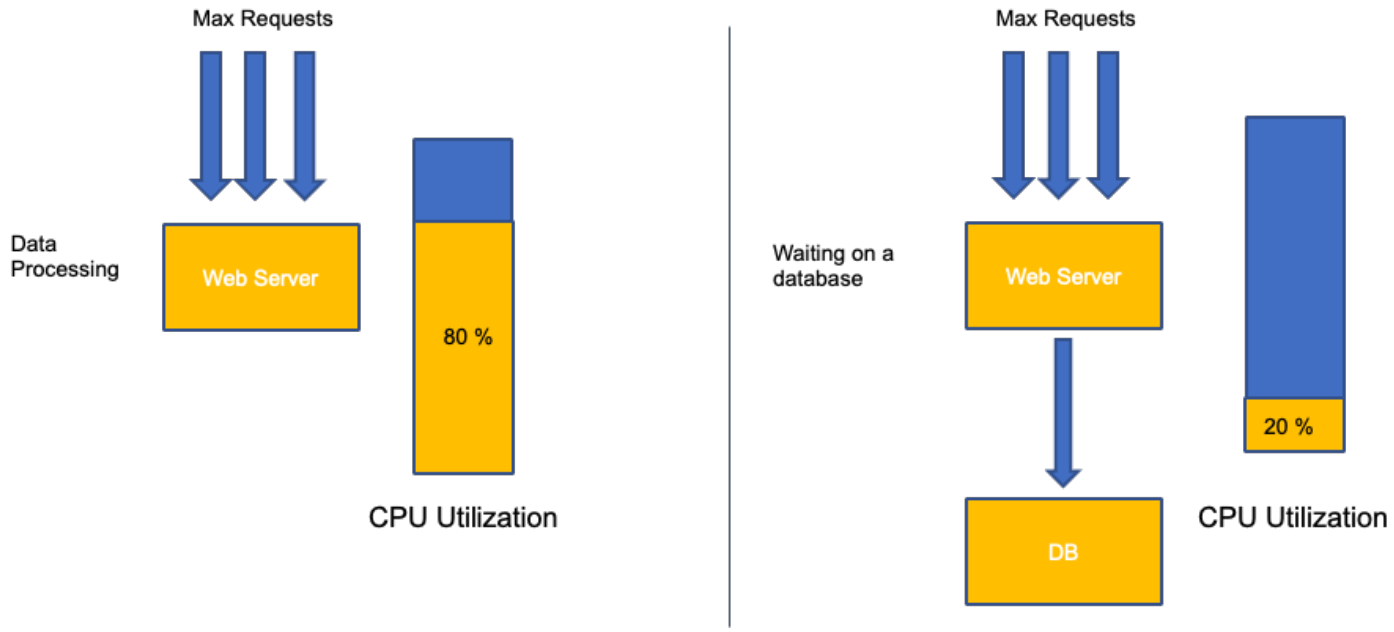
> **Note**
>
> Linux CFS a ses points forts. Cela est particulièrement vrai pour les charges de travail I/O basées. Toutefois, si votre application utilise des cœurs complets sans sidecars et qu'elle n'est pas I/O requise, l'épinglage du processeur peut considérablement compliquer ce processus. C'est pourquoi nous vous encourageons à le faire avec ces mises en garde.

Utilisation ou saturation

Une erreur courante lors de la mise à l'échelle des applications consiste à utiliser uniquement l'utilisation du processeur pour votre métrique de dimensionnement. Dans les applications complexes, c'est presque toujours un mauvais indicateur qu'une application est réellement saturée de demandes. Dans l'exemple de gauche, nous voyons que toutes nos requêtes arrivent en fait sur le serveur Web. L'utilisation du processeur suit donc bien le rythme de saturation.

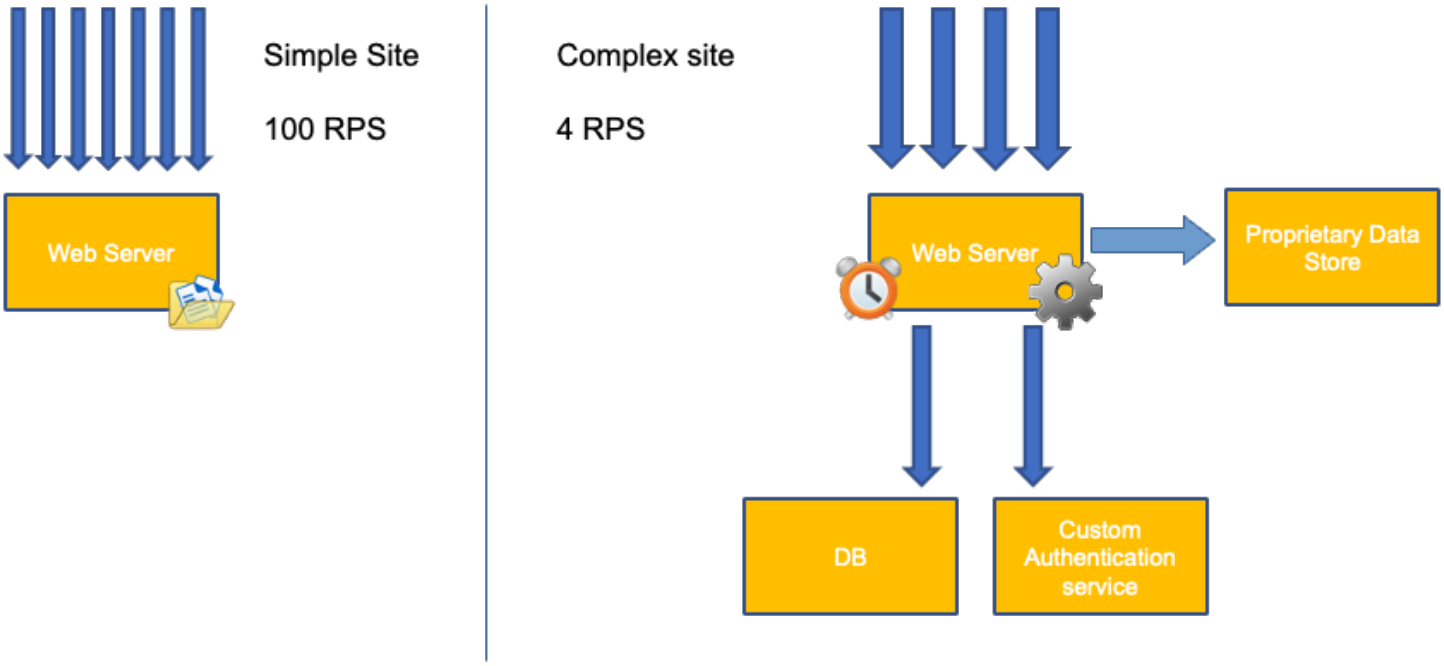
Dans les applications du monde réel, il est probable que certaines de ces demandes soient traitées par une couche de base de données ou une couche d'authentification, etc. Dans ce cas plus courant,

notez que le processeur ne suit pas la saturation car la demande est traitée par d'autres entités. Dans ce cas, le processeur est un très mauvais indicateur de saturation.



L'utilisation d'une métrique incorrecte dans les performances des applications est la principale raison du dimensionnement inutile et imprévisible de Kubernetes. Il faut choisir avec soin la bonne métrique de saturation pour le type d'application que vous utilisez. Il est important de noter qu'il n'existe pas de recommandation universelle qui puisse être donnée. En fonction de la langue utilisée et du type d'application en question, il existe un ensemble varié de mesures de saturation.

Nous pouvons penser que ce problème ne concerne que l'utilisation du processeur, mais d'autres métriques courantes, telles que le nombre de demandes par seconde, peuvent également être confrontées exactement au même problème que celui décrit ci-dessus. Notez que la demande peut également être envoyée aux couches de base de données, aux couches d'authentification, sans être directement traitée par notre serveur Web. Il s'agit donc d'une mauvaise métrique pour une véritable saturation du serveur Web lui-même.



Malheureusement, il n'existe pas de réponse facile lorsqu'il s'agit de choisir la bonne métrique de saturation. Voici quelques directives à prendre en compte :

- Comprenez votre langage d'exécution : les langages dotés de plusieurs threads de système d'exploitation réagiront différemment des applications à thread unique, ce qui aura un impact différent sur le nœud.
- Comprenez l'échelle verticale correcte : quelle quantité de mémoire tampon souhaitez-vous pour que vos applications mettent à l'échelle verticale avant de redimensionner un nouveau pod ?
- Quelles mesures reflètent réellement la saturation de votre application ? La métrique de saturation d'un Kafka Producer serait très différente de celle d'une application Web complexe.
- Comment toutes les autres applications du nœud s'influencent-elles ? Les performances des applications ne se font pas en vase clos, les autres charges de travail du nœud ont un impact majeur.

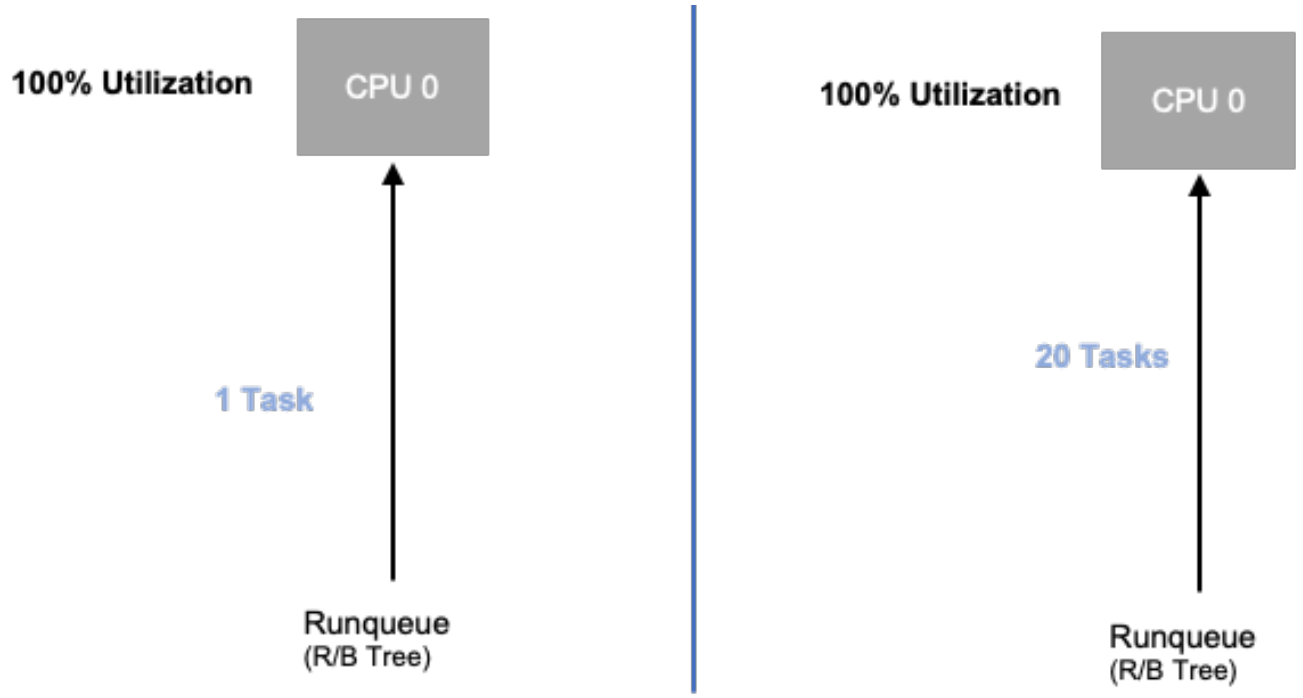
Pour clore cette section, il serait facile de rejeter ce qui précède comme étant trop complexe et inutile. Il arrive souvent que nous rencontrions un problème, mais nous n'en connaissons pas la véritable nature car nous examinons les mauvais indicateurs. Dans la section suivante, nous verrons comment cela pourrait se produire.

Saturation des nœuds

Maintenant que nous avons exploré la saturation des applications, examinons ce même concept du point de vue des nœuds. Prenons deux CPUs modèles utilisés à 100 % pour voir la différence entre l'utilisation et la saturation.

Le vCPU de gauche est utilisé à 100 %, mais aucune autre tâche n'attend d'être exécutée sur ce vCPU, donc d'un point de vue purement théorique, c'est très efficace. Entre-temps, dans le deuxième exemple, 20 applications à thread unique attendent d'être traitées par un vCPU. Les 20 applications seront désormais confrontées à un certain type de latence en attendant leur tour pour être traitées par le vCPU. En d'autres termes, le vCPU de droite est saturé.

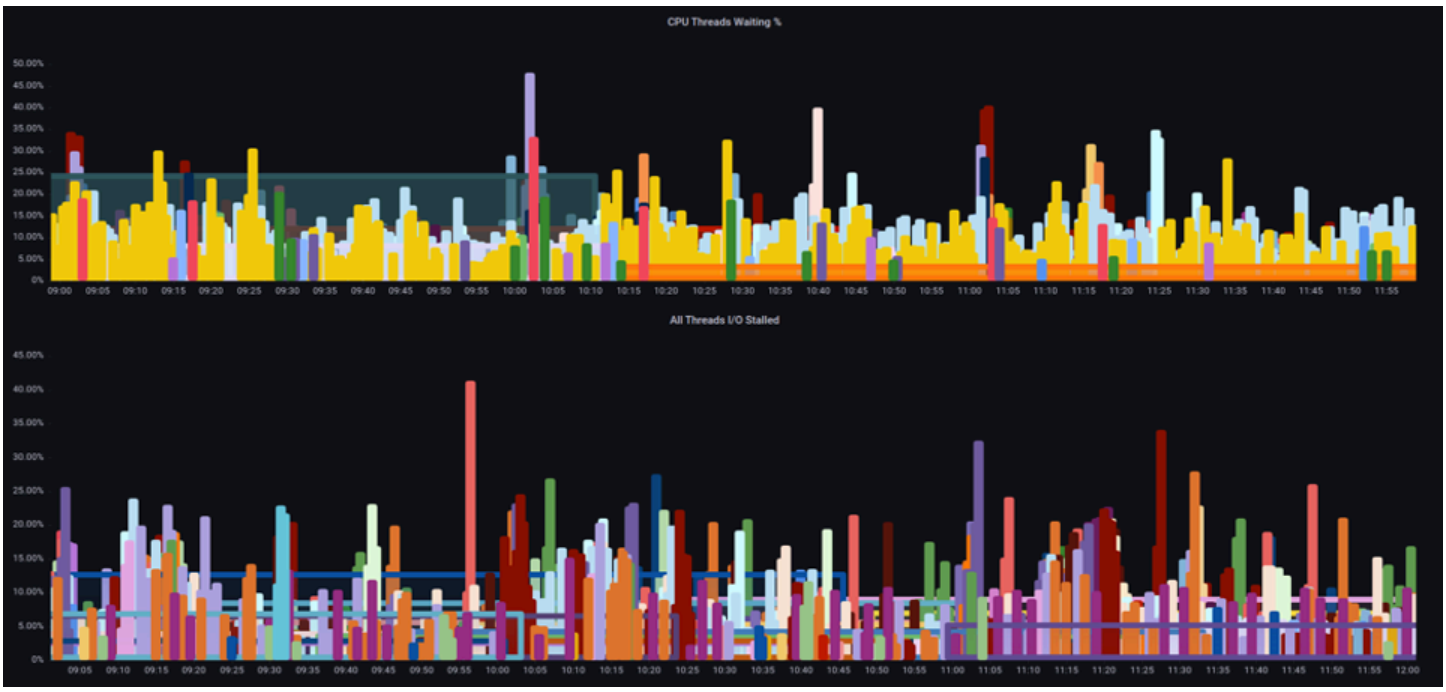
Non seulement nous ne verrions pas ce problème si nous ne considérions que l'utilisation, mais nous pourrions également attribuer cette latence à un élément sans rapport, tel que le réseau, ce qui nous conduirait sur la mauvaise voie.



Il est important de consulter les métriques de saturation, et pas seulement les métriques d'utilisation, lorsque vous augmentez le nombre total de pods exécutés sur un nœud à un moment donné, car nous pouvons facilement passer à côté du fait que nous avons sursaturé un nœud. Pour cette tâche, nous pouvons utiliser les mesures d'information relatives au décrochage sous pression, comme indiqué dans le graphique ci-dessous.

ProMQL - E/S bloquées

```
topk(3, ((irate(node_pressure_io_stalled_seconds_total[1m])) * 100))
```



> **Note**
>
> Pour en savoir plus sur les mesures de décrochage sous pression, consultez https://facebookmicrosites.github. io/psi/docs/overview*

Avec ces métriques, nous pouvons savoir si les threads attendent le processeur, ou même si tous les threads de la boîte sont bloqués en attente de ressources telles que la mémoire ou I/O. For example, we could see what percentage every thread on the instance was stalled waiting on I/O pendant une période d'une minute.

```
topk(3, ((irate(node_pressure_io_stalled_seconds_total[1m])) * 100))
```

À l'aide de cette métrique, nous pouvons voir dans le graphique ci-dessus que chaque thread était bloqué 45 % du temps d'attente I/O au point culminant, ce qui signifie que nous avons gâché tous ces cycles de processeur pendant cette minute. Le fait de comprendre que cela se produit peut nous aider à gagner beaucoup de temps sur le vCPU, rendant ainsi le dimensionnement plus efficace.

HPA V2

Il est recommandé d'utiliser la version autoscaling/v2 de l'API HPA. Les anciennes versions de l'API HPA pouvaient être bloquées dans certains cas extrêmes. Cela était également limité au fait que les pods ne doublaient que lors de chaque étape de mise à l'échelle, ce qui créait des problèmes pour les petits déploiements nécessitant une mise à l'échelle rapide.

AutoScaling/V2 nous offre une plus grande flexibilité pour inclure plusieurs critères sur lesquels nous pouvons évoluer et nous donne une grande flexibilité lors de l'utilisation de métriques personnalisées et externes (métriques autres que K8s).

À titre d'exemple, nous pouvons effectuer une mise à l'échelle sur la plus haute des trois valeurs (voir ci-dessous). Nous effectuons une mise à l'échelle si l'utilisation moyenne de tous les pods est supérieure à 50 %, si les métriques personnalisées indiquent que le nombre de paquets par seconde d'entrée dépasse une moyenne de 1 000 ou si le nombre d'objets entrants dépasse 10 000 demandes par seconde.

> **Note**
>
> C'est juste pour montrer la flexibilité de l'API d'auto-scaling. Nous vous recommandons d'éviter les règles trop complexes qui peuvent être difficiles à résoudre en production.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
```

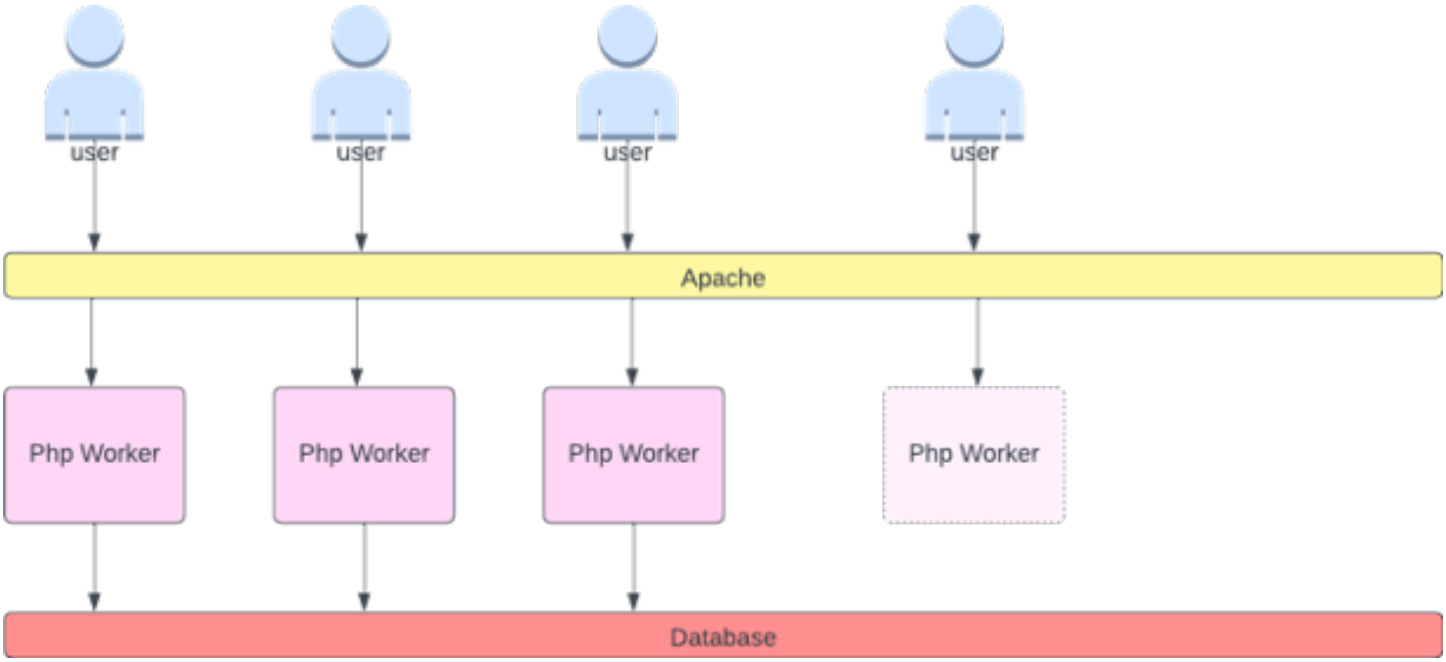
```
        averageUtilization: 50
  - type: Pods
    pods:
      metric:
        name: packets-per-second
      target:
        type: AverageValue
        averageValue: 1k
  - type: Object
    object:
      metric:
        name: requests-per-second
      describedObject:
        apiVersion: networking.k8s.io/v1
        kind: Ingress
        name: main-route
      target:
        type: Value
        value: 10k
```

Cependant, nous avons pris conscience du danger que représente l'utilisation de tels indicateurs pour des applications Web complexes. Dans ce cas, nous serions mieux servis en utilisant une métrique personnalisée ou externe qui reflète avec précision la saturation de notre application par rapport à son utilisation. HPAv2 permet cela en ayant la possibilité d'évoluer en fonction de n'importe quelle métrique, mais nous devons toujours trouver et exporter cette métrique vers Kubernetes pour l'utiliser.

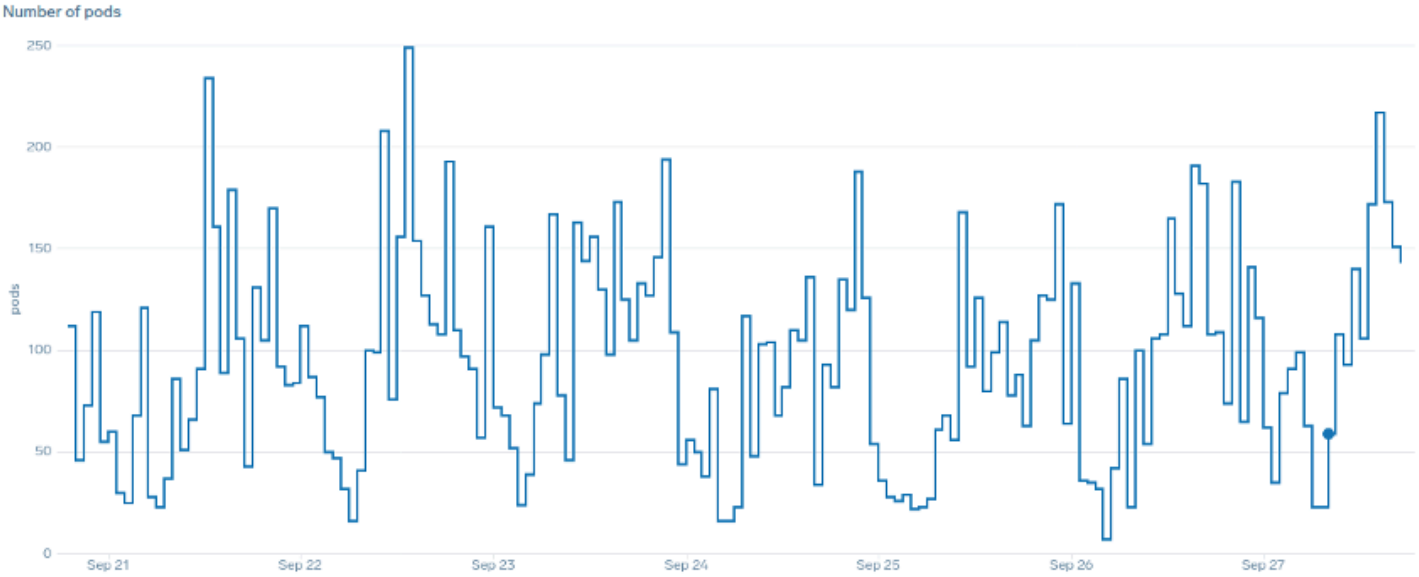
Par exemple, nous pouvons examiner le nombre de files d'attente de threads actifs dans Apache. Cela crée souvent un profil de mise à l'échelle « plus fluide » (nous reviendrons bientôt sur ce terme). Si un thread est actif, peu importe qu'il attende une couche de base de données ou qu'il traite une demande localement. Si tous les threads de l'application sont utilisés, c'est une bonne indication que l'application est saturée.

Nous pouvons utiliser cet épuisement des threads comme signal pour créer un nouveau pod avec un pool de threads entièrement disponible. Cela nous permet également de contrôler la taille de la mémoire tampon que nous voulons que l'application absorbe en période de trafic intense. Par exemple, si nous avions un pool total de 10 threads, la mise à l'échelle à 4 threads utilisés contre 8 threads utilisés aurait un impact majeur sur la mémoire tampon dont nous disposons lors du dimensionnement de l'application. Un paramètre de 4 serait judicieux pour une application qui doit évoluer rapidement sous une charge importante, alors qu'un paramètre de 8 serait plus efficace avec

nos ressources si nous avions suffisamment de temps pour le faire, étant donné que le nombre de demandes augmente lentement plutôt que fortement au fil du temps.

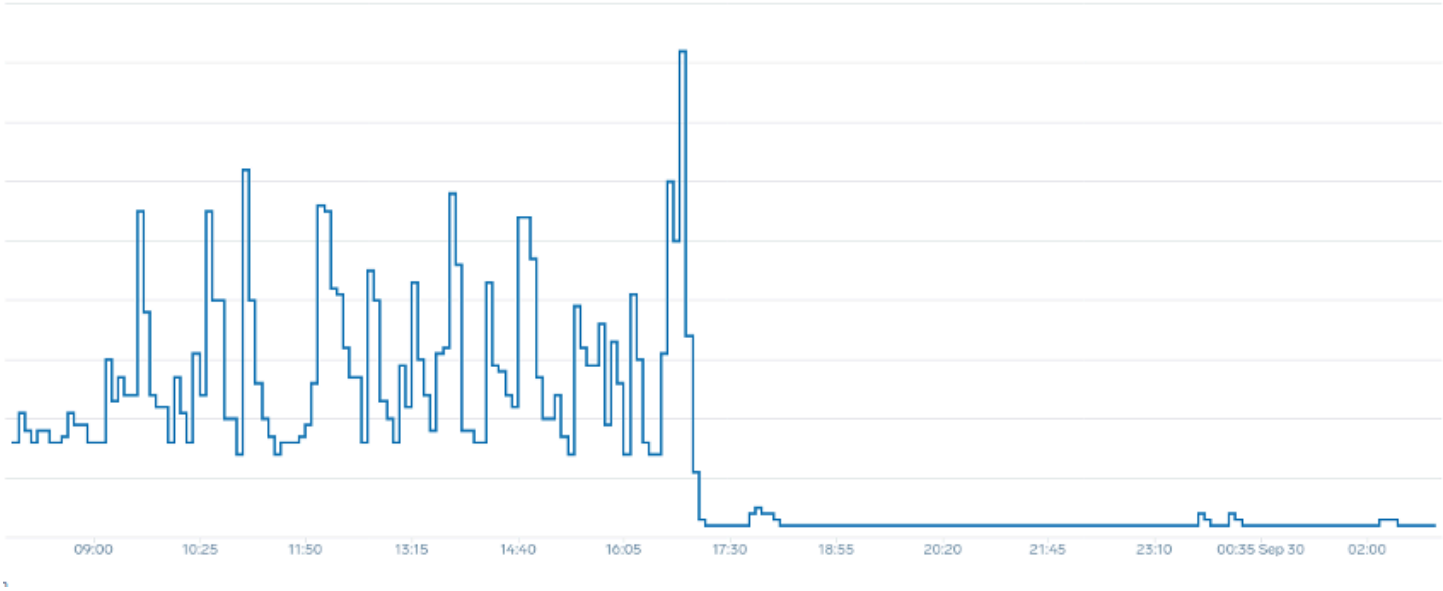


Qu'entendons-nous par le terme « fluide » lorsqu'il est question de mise à l'échelle ? Notez le graphique ci-dessous où nous utilisons le processeur comme métrique. Dans le cadre de ce déploiement, le nombre de capsules augmente en peu de temps, passant de 50 à 250 unités, pour ensuite être immédiatement réduites à nouveau. Cette mise à l'échelle très inefficace est la principale cause du taux de désabonnement des clusters.



Remarquez qu'une fois que nous avons opté pour une métrique qui reflète le point idéal de notre application (partie centrale du graphique), nous sommes en mesure de procéder à une mise

à l'échelle fluide. Notre mise à l'échelle est désormais efficace et nos pods peuvent s'adapter pleinement à la marge de manœuvre que nous avons fournie en ajustant les paramètres des demandes. Aujourd'hui, un petit groupe de capsules effectue le travail que des centaines de capsules effectuaient auparavant. Les données réelles montrent qu'il s'agit du principal facteur d'évolutivité des clusters Kubernetes.

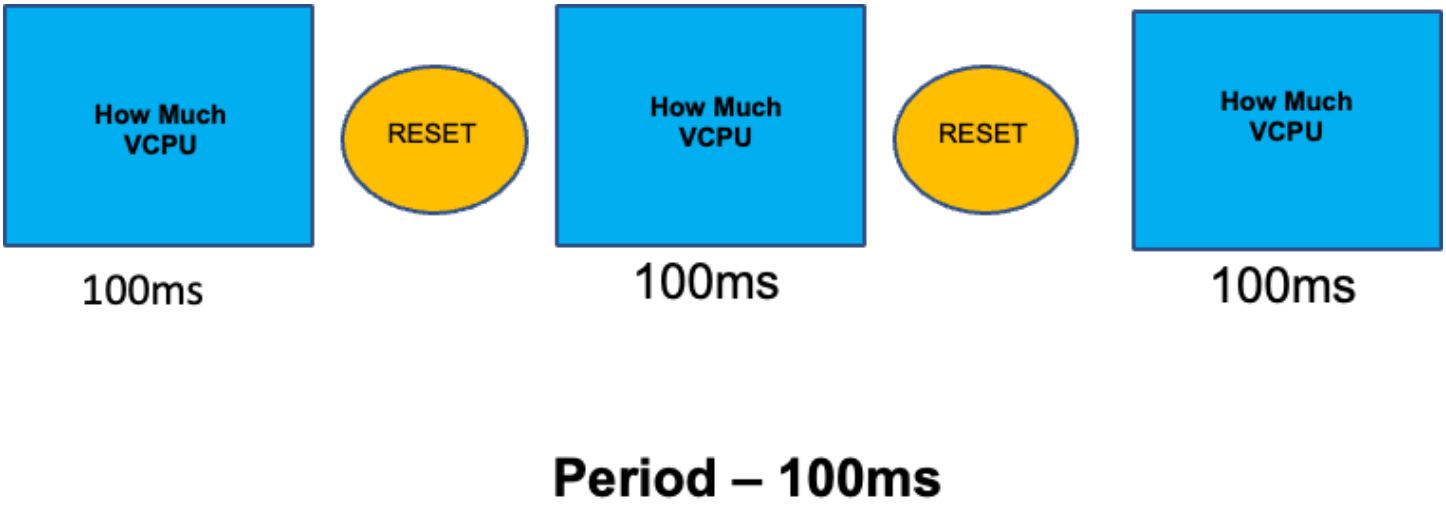


Le principal point à retenir est que l'utilisation du processeur n'est qu'une dimension des performances des applications et des nœuds. L'utilisation du processeur comme seul indicateur de santé pour nos nœuds et applications pose des problèmes d'évolutivité, de performance et de coût, deux concepts étroitement liés. Plus l'application et les nœuds sont performants, moins vous avez besoin d'évolutivité, ce qui réduit vos coûts.

La recherche et l'utilisation des mesures de saturation appropriées pour la mise à l'échelle de votre application particulière vous permettent également de surveiller et d'identifier les véritables goulots d'étranglement de cette application. Si cette étape critique est ignorée, les rapports faisant état de problèmes de performances seront difficiles, voire impossibles, à comprendre.

Configuration des limites du processeur

Pour compléter cette section sur les sujets mal compris, nous aborderons les limites du processeur. En bref, les limites sont des métadonnées associées au conteneur dont le compteur est réinitialisé toutes les 100 ms. Cela permet à Linux de suivre le nombre de ressources du processeur utilisées à l'échelle d'un nœud par un conteneur spécifique sur une période de 100 ms.



Une erreur courante lors de la définition des limites est de supposer que l'application est mono-thread et ne fonctionne que sur son vCPU « assigné ». Dans la section ci-dessus, nous avons appris que le CFS n'attribue pas de cœurs et qu'en réalité, un conteneur exécutant de grands pools de threads planifie l'utilisation de tous les processeurs virtuels disponibles sur le boîtier.

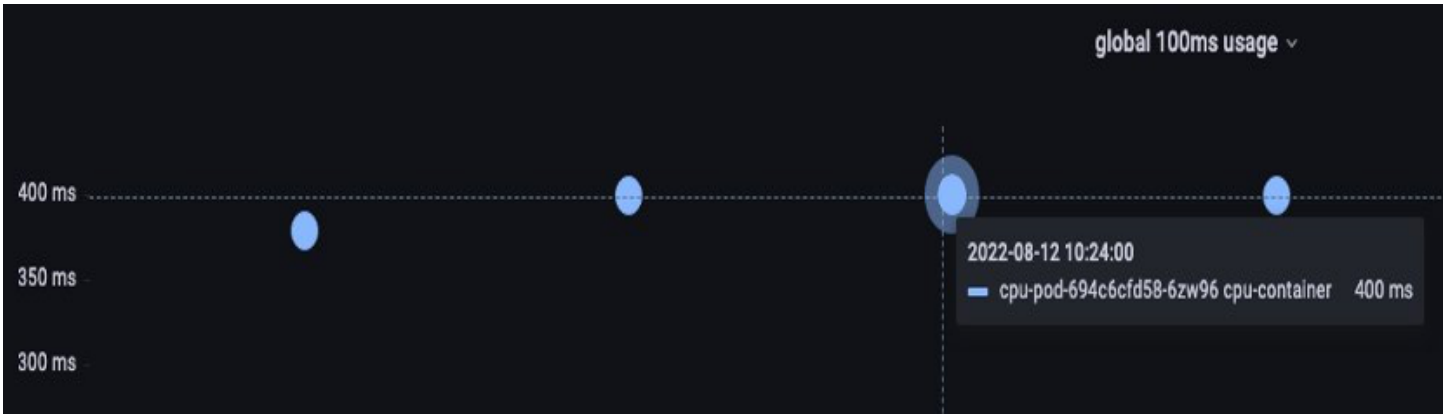
Si 64 threads du système d'exploitation s'exécutent sur 64 cœurs disponibles (du point de vue d'un nœud Linux), le temps processeur total utilisé sur une période de 100 ms sera assez élevé une fois le temps passé sur l'ensemble de ces 64 cœurs additionné. Comme cela ne se produit que pendant un processus de collecte des ordures, il peut être assez facile de rater quelque chose comme ça. C'est pourquoi il est nécessaire d'utiliser des métriques pour s'assurer que l'utilisation est correcte au fil du temps avant de tenter de fixer une limite.

Heureusement, nous avons un moyen de voir exactement quelle quantité de vCPU est utilisée par tous les threads d'une application. Nous utiliserons la métrique `container_cpu_usage_seconds_total` à cette fin.

Comme la logique de régulation se produit toutes les 100 ms et que cette métrique est une métrique par seconde, nous allons ProMQL pour qu'il corresponde à cette période de 100 ms. [Si vous souhaitez approfondir ce travail de déclaration ProMQL, veuillez consulter le blog suivant.]

Requête ProMQL :

```
topk(3, max by (pod, container)(rate(container_cpu_usage_seconds_total{image!="",
 instance="$instance"}[$__rate_interval]))) / 10
```



Une fois que nous avons le sentiment d'avoir la bonne valeur, nous pouvons fixer des limites à la production. Il devient alors nécessaire de voir si notre application est bloquée en raison d'un imprévu. Nous pouvons le faire en regardant `container_cpu_throttled_seconds_total`

```
topk(3, max by (pod, container)(rate(container_cpu_cfs_throttled_seconds_total{image!
=``""``, instance=``"$instance"``}[$__rate_interval]))) / 10
```

Mémoire

L'allocation de mémoire est un autre exemple où il est facile de confondre le comportement de planification de Kubernetes avec le comportement de Linux. CGroup Il s'agit d'un sujet plus nuancé car des changements majeurs ont été apportés à la façon dont la CGroup v2 gère la mémoire sous Linux et Kubernetes a modifié sa syntaxe pour refléter cela ; lisez ce blog pour plus de détails.

Contrairement aux demandes du processeur, les demandes de mémoire ne sont pas utilisées une fois le processus de planification terminé. En effet, nous ne pouvons pas compresser la mémoire en CGroup v1 de la même manière qu'avec le processeur. Cela ne nous laisse que des limites de mémoire, conçues pour empêcher les fuites de mémoire en mettant complètement le module hors

service. Il s'agit d'une proposition de style « tout ou rien », mais nous avons maintenant de nouvelles façons de résoudre ce problème.

Tout d'abord, il est important de comprendre que définir la bonne quantité de mémoire pour les conteneurs n'est pas aussi simple qu'il y paraît. Le système de fichiers sous Linux utilisera la mémoire comme cache pour améliorer les performances. Ce cache augmentera au fil du temps, et il peut être difficile de savoir quelle quantité de mémoire est utile pour le cache, mais qui peut être récupérée sans que cela ait un impact significatif sur les performances des applications. Cela entraîne souvent une mauvaise interprétation de l'utilisation de la mémoire.

La capacité de « compresser » la mémoire était l'un des principaux moteurs de la CGroup v2. Pour en savoir plus sur les raisons pour lesquelles la CGroup V2 était nécessaire, veuillez consulter la présentation de Chris Down, LISA21 où il explique pourquoi l'incapacité à définir correctement la mémoire minimale est l'une des raisons qui l'ont poussé à créer des métriques de CGroup v2 et de blocage sous pression.

Heureusement, Kubernetes possède désormais le concept de `memory.min` et en dessous. `memory.high` `requests.memory` Cela nous donne la possibilité de libérer de manière agressive cette mémoire cache pour que d'autres conteneurs puissent l'utiliser. Une fois que le conteneur atteint la limite maximale de mémoire, le noyau peut récupérer agressivement la mémoire de ce conteneur jusqu'à la valeur définie à. `memory.min` Cela nous donne plus de flexibilité lorsqu'un nœud est soumis à une pression de mémoire.

La question clé est la suivante : à quelle valeur `memory.min` définir ? C'est là que les métriques de blocage de la mémoire entrent en jeu. Nous pouvons utiliser ces métriques pour détecter le « battement » de mémoire au niveau du conteneur. Ensuite, nous pouvons utiliser des contrôleurs tels que fbtax pour détecter les valeurs correctes `memory.min` en recherchant ce battement de mémoire, et définir dynamiquement la `memory.min` valeur sur ce paramètre.

Résumé

Pour résumer la section, il est facile de confondre les concepts suivants :

- Utilisation et saturation
- Règles de performance Linux avec la logique du planificateur Kubernetes

Il faut prendre grand soin de séparer ces concepts. La performance et l'échelle sont étroitement liées. Une mise à l'échelle inutile crée des problèmes de performance, qui à leur tour créent des problèmes de dimensionnement.

Kubernetes Upstream SLOs

Amazon EKS exécute le même code que les versions en amont de Kubernetes et garantit que les clusters EKS fonctionnent conformément aux directives SLOs définies par la communauté Kubernetes. Le Kubernetes Scalability Special Interest Group (SIG) définit les objectifs d'évolutivité et étudie les obstacles aux performances par le biais de et. SLIs SLOs

SLIs sont la façon dont nous mesurons un système, comme les métriques ou les mesures qui peuvent être utilisées pour déterminer le « bon » fonctionnement du système, par exemple la latence ou le nombre de demandes. SLOs définissez les valeurs attendues lorsque le système fonctionne « bien », par exemple lorsque la latence des demandes reste inférieure à 3 secondes. Les Kubernetes SLOs SLIs se concentrent sur les performances des composants Kubernetes et sont totalement indépendants du service Amazon EKS SLAs qui se concentre sur la disponibilité du point de terminaison du cluster EKS.

Kubernetes possède un certain nombre de fonctionnalités qui permettent aux utilisateurs d'étendre le système avec des modules complémentaires ou des pilotes personnalisés, tels que des pilotes CSI, des webhooks d'admission et des scalers automatiques. Ces extensions peuvent avoir un impact considérable sur les performances d'un cluster Kubernetes de différentes manières. Par exemple, un webhook d'admission `failurePolicy=Ignore` peut ajouter de la latence aux requêtes d'API K8s si la cible du webhook n'est pas disponible. Le SIG de scalabilité de Kubernetes définit l'évolutivité à l'aide d'un framework « vous promettez, nous le promettons » :

> Si vous promettez de : - configurer correctement votre cluster - utiliser les fonctionnalités d'extensibilité « raisonnablement » - maintenir la charge du cluster dans les limites recommandées
>
> alors nous vous promettons que votre cluster évolue, c'est-à-dire que tous SLOs sont satisfaits.

Kubernetes SLOs

Les Kubernetes SLOs ne tiennent pas compte de tous les plugins et limitations externes susceptibles d'avoir un impact sur un cluster, tels que le dimensionnement des nœuds de travail ou les webhooks d'admission. Ils SLOs se concentrent sur les composants de Kubernetes et garantissent que les actions et les ressources de Kubernetes fonctionnent conformément aux attentes. Ils SLOs aident les développeurs de Kubernetes à s'assurer que les modifications apportées au code Kubernetes ne dégradent pas les performances de l'ensemble du système.

Le SIG de scalabilité de Kuberntes définit le SLO/ officiel suivant. SLIs L'équipe Amazon EKS effectue régulièrement des tests d'évolutivité sur les clusters EKS SLOs/SLIs afin de surveiller la dégradation des performances à mesure que des modifications sont apportées et que de nouvelles versions sont publiées.

Objectif	Définition	SLO
Latence des demandes d'API (mutation)	Latence du traitement des appels d'API mutants pour des objets uniques pour chaque paire (ressource, verbe), mesurée au 99e percentile au cours des 5 dernières minutes	Dans l'installation par défaut de Kubernetes, pour chaque paire (ressource, verbe), à l'exception des ressource s virtuelles et agrégées et des définitions de ressources personnalisées, 99e percentile par jour de cluster <= 1 s
Latence des demandes d'API (lecture seule)	Latence du traitement des appels d'API en lecture seule non diffusés pour chaque paire (ressource, étendue), mesurée au 99e percentile au cours des 5 dernières minutes	Dans l'installation par défaut de Kubernetes, pour chaque paire (ressource, étendue), à l'exception des ressource s virtuelles et agrégées et des définitions de ressource s personnalisées, 99e centile par jour de cluster : (a) <= 1s si (b) <= 30s sinon (si ou) `scope=resource` `scope=namespace` `scope=cluster`
Latence de démarrage du pod	Latence de démarrage des pods apatrides programma bles, à l'exclusion du temps nécessaire pour extraire des images et exécuter des conteneurs d'initialisation, mesurée depuis l'horodat age de création des pods	Dans l'installation par défaut de Kubernetes, 99e percentile par jour de cluster <= 5s

Objectif	Définition	SLO
	jusqu'au moment où tous leurs conteneurs sont signalés comme démarrés et observés par surveillance, mesurée au 99e centile au cours des 5 dernières minutes	

Latence des demandes d'API

Le `kube-apiserver` a été `--request-timeout` défini comme `1m0s` par défaut, ce qui signifie qu'une demande peut être exécutée pendant une minute (60 secondes) au maximum avant d'être expirée et annulée. Les SLOs valeurs définies pour la latence sont ventilées selon le type de demande effectuée, qui peut être mutante ou en lecture seule :

Mutant

Les demandes mutantes dans Kubernetes apportent des modifications à une ressource, telles que des créations, des suppressions ou des mises à jour. Ces demandes sont coûteuses car ces modifications doivent être écrites dans le backend etcd avant que l'objet mis à jour ne soit renvoyé. Etcd est un magasin clé-valeur distribué utilisé pour toutes les données du cluster Kubernetes.

Cette latence est mesurée comme le 99e percentile sur 5 minutes pour les paires (ressource, verbe) de ressources Kubernetes. Par exemple, cela permettrait de mesurer la latence pour les requêtes Create Pod et Update Node. La latence de la demande doit être inférieure ou égale à 1 seconde pour satisfaire le SLO.

Lecture seule

Les requêtes en lecture seule récupèrent une seule ressource (telle que Get Pod X) ou une collection (telle que « Get all Pods from Namespace X »). Il `kube-apiserver` maintient un cache d'objets, de sorte que les ressources demandées peuvent être renvoyées depuis le cache ou qu'il peut être nécessaire de les récupérer d'abord depuis etcd. Ces latences sont également mesurées par le 99e percentile sur 5 minutes, mais les demandes en lecture seule peuvent avoir des portées distinctes. Le SLO définit deux objectifs différents :

- Pour les demandes effectuées pour une seule ressource (c'est-à-dire`kubectl get pod -n mynamespace my-controller-xxx`), la latence de la demande doit rester inférieure ou égale à 1 seconde.
- Pour les demandes adressées à plusieurs ressources dans un espace de noms ou un cluster (par exemple,`kubectl get pods -A`), la latence doit rester inférieure ou égale à 30 secondes

Le SLO a des valeurs cibles différentes pour différentes portées de demandes, car les demandes effectuées pour une liste de ressources Kubernetes s'attendent à ce que les détails de tous les objets de la demande soient renvoyés dans le SLO. Sur les grands clusters ou les grands ensembles de ressources, cela peut entraîner des réponses de grande taille dont le retour peut prendre un certain temps. Par exemple, dans un cluster exécutant des dizaines de milliers de pods, chaque pod pesant environ 1 KiB lorsqu'il est codé en JSON, le renvoi de tous les pods du cluster comporterait 10 Mo ou plus. Les clients Kubernetes peuvent aider à réduire cette taille de réponse en utilisant le découpage pour APIList récupérer de grandes collections de ressources.

Latence de démarrage du pod

Ce SLO concerne principalement le temps qui s'écoule entre la création du pod et le moment où les conteneurs de ce pod commencent réellement à être exécutés. Pour mesurer cela, la différence par rapport à l'horodatage de création enregistré sur le Pod et lorsqu'un WATCH sur ce Pod indique que les conteneurs ont démarré est calculée (à l'exception du temps nécessaire pour extraire les images du conteneur et exécuter le conteneur d'initialisation). Pour satisfaire le SLO, le 99e centile par jour de cluster de la latence de démarrage de ce pod doit rester inférieur à 5 secondes.

Notez que ce SLO suppose que les nœuds de travail existent déjà dans ce cluster dans un état prêt pour la planification du Pod. Ce SLO ne prend pas en compte les extractions d'images ni les exécutions de conteneurs d'initialisation, et limite également le test aux « pods apatrides » qui n'exploitent pas les plugins de stockage persistants.

Métriques SLI de Kubernetes

Kubernetes améliore également l'observabilité en SLIs ajoutant des métriques Prometheus aux composants Kubernetes qui les suivent au fil du temps. SLIs À l'aide du langage de requête Prometheus (ProMQL), nous pouvons créer des requêtes qui affichent les performances du SLI au fil du temps dans des outils tels que les tableaux de bord Prometheus ou Grafana. Vous trouverez ci-dessous quelques exemples de ce qui précède. SLOs

Latence des demandes du serveur API

Métrique	Définition
apiserver_request_sli_duration_seconds	Distribution de la latence de réponse (sans compter la durée du webhook et les temps d'attente dans les files d'attente pour les priorités et l'équité) en secondes pour chaque verbe, groupe, version, ressource, sous-ressource, portée et composant.
apiserver_request_duration_seconds	Distribution de la latence de réponse en secondes pour chaque verbe, valeur d'essai, groupe, version, ressource, sous-ressource, portée et composant.

> **Note**
>
> La `apiserver_request_sli_duration_seconds` métrique est disponible à partir de Kubernetes 1.27.

Vous pouvez utiliser ces mesures pour étudier les temps de réponse du serveur d'API et déterminer s'il existe des goulots d'étranglement dans les composants Kubernetes ou dans d'autres plugins/composants. Les requêtes ci-dessous sont basées sur le tableau de bord SLO de la communauté.

Latence des demandes d'API SLI (mutante) : cette fois-ci n'inclut pas l'exécution du webhook ni le temps d'attente dans la file d'attente. `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"CREATE|DELETE|PATCH|POST|PUT", subresource!~"proxy|attach|log|exec|portforward"}[5m])) by (resource, subresource, verb, scope, le)) > 0`

Latence totale de la demande d'API (mutante) : il s'agit du temps total que la demande a pris sur le serveur d'API. Ce temps peut être plus long que le temps SLI car il inclut l'exécution du webhook et les temps d'attente relatifs à la priorité et à l'équité de l'API. `histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_bucket{verb=~"CREATE|DELETE|`

`PATCH|POST|PUT", subresource!~"proxy|attach|log|exec|portforward"}[5m])) by (resource, subresource, verb, scope, le)) > 0`

Dans ces requêtes, nous excluons les requêtes d'API de streaming qui ne sont pas renvoyées immédiatement, telles que `kubectl port-forward` or `kubectl exec` requests (`subresource!~"proxy|attach|log|exec|portforward"`), et nous filtrons uniquement les verbes Kubernetes qui modifient les objets (). `verb=~"CREATE|DELETE|PATCH|POST|PUT"` Nous calculons ensuite le 99e percentile de cette latence au cours des 5 dernières minutes.

Nous pouvons utiliser une requête similaire pour les demandes d'API en lecture seule, nous modifions simplement les verbes que nous filtrons pour inclure les actions en lecture seule `LIST` et`GET`. Il existe également différents seuils de SLO en fonction de l'étendue de la demande, par exemple pour obtenir une seule ressource ou répertorier un certain nombre de ressources.

Latence des demandes d'API SLI (lecture seule) : cette fois n'inclut pas l'exécution du webhook ni le temps d'attente dans la file d'attente. Pour une seule ressource (scope=resource, seuil=1s) `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"GET", scope=~"resource"}[5m])) by (resource, subresource, verb, scope, le))`

Pour un ensemble de ressources dans le même espace de noms (scope=namespace, threshold = 5s) `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"LIST", scope=~"namespace"}[5m])) by (resource, subresource, verb, scope, le))`

Pour un ensemble de ressources sur l'ensemble du cluster (scope=cluster, seuild=30s) `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"LIST", scope=~"cluster"}[5m])) by (resource, subresource, verb, scope, le))`

Latence totale des demandes d'API (lecture seule) : il s'agit du temps total que la demande a pris sur le serveur d'API. Ce temps peut être plus long que le temps SLI car il inclut l'exécution du webhook et les temps d'attente. Pour une seule ressource (scope=resource, seuil=1s) `histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_bucket{verb=~"GET", scope=~"resource"}[5m])) by (resource, subresource, verb, scope, le))`

Pour un ensemble de ressources dans le même espace de noms (scope=namespace, threshold = 5s) `histogram_quantile(0.99,`

```
sum(rate(apiserver_request_duration_seconds_bucket{verb=~"LIST",
scope=~"namespace"}[5m])) by (resource, subresource, verb, scope, le))
```

Pour un ensemble de ressources sur l'ensemble du cluster (scope=cluster, seuild=30s)

```
histogram_quantile(0.99,
sum(rate(apiserver_request_duration_seconds_bucket{verb=~"LIST",
scope=~"cluster"}[5m])) by (resource, subresource, verb, scope, le))
```

Les métriques SLI fournissent un aperçu des performances des composants Kubernetes en excluant le temps que les demandes passent à attendre dans les files d'attente des API Priority et Fairness, à travailler via des webhooks d'admission ou d'autres extensions Kubernetes. Les statistiques totales fournissent une vue plus globale car elles reflètent le temps que vos applications attendraient une réponse du serveur d'API. La comparaison de ces indicateurs peut donner un aperçu de l'origine des retards dans le traitement des demandes.

Latence de démarrage du pod

Métrique	Définition
kubelet_pod_start_sli_duration_seconds	Durée en secondes nécessaire au démarrage d'un pod, à l'exclusion du temps nécessair e pour extraire des images et exécuter des conteneurs d'initialisation, mesurée entre l'horodatage de création du pod et le moment où tous ses conteneurs sont signalés comme démarrés et observés par surveillance
kubelet_pod_start_duration_seconds	Durée en secondes entre le moment où kubelet voit un pod pour la première fois et celui où le pod commence à fonctionner. Cela n'inclut pas le temps nécessaire pour planifier le module ou augmenter la capacité du nœud de travail.

> **Note**
>
> `kubelet_pod_start_sli_duration_seconds`est disponible à partir de Kubernetes 1.27.

Comme pour les requêtes ci-dessus, vous pouvez utiliser ces métriques pour savoir combien de temps la mise à l'échelle des nœuds, les extractions d'images et les conteneurs d'initialisation retardent le lancement du pod par rapport aux actions Kubelet.

Latence de démarrage du pod SLI : il s'agit du temps écoulé entre la création du pod et le moment où les conteneurs d'applications ont été signalés comme étant en cours d'exécution. Cela inclut le temps nécessaire pour que la capacité du nœud de travail soit disponible et que le pod soit planifié, mais cela n'inclut pas le temps nécessaire pour extraire les images ou pour exécuter les conteneurs d'initialisation. `histogram_quantile(0.99, sum(rate(kubelet_pod_start_sli_duration_seconds_bucket[5m])) by (le))`

Latence totale au démarrage du pod : c'est le temps qu'il faut au kubelet pour démarrer le pod pour la première fois. Cela est mesuré à partir du moment où le kubelet reçoit le pod via WATCH, ce qui n'inclut pas le temps nécessaire au dimensionnement ou à la planification du nœud de travail. Cela inclut le temps nécessaire pour extraire les images et initier les conteneurs à exécuter. `histogram_quantile(0.99, sum(rate(kubelet_pod_start_duration_seconds_bucket[5m])) by (le))`

SLOs sur votre cluster

Si vous collectez les métriques Prometheus à partir des ressources Kubernetes de votre cluster EKS, vous pouvez obtenir des informations plus approfondies sur les performances des composants du plan de contrôle Kubernetes.

Le référentiel perf-tests inclut des tableaux de bord Grafana qui affichent les latences et les indicateurs de performance critiques du cluster pendant les tests. La configuration des tests de performance s'appuie sur un projet open source configuré pour collecter des métriques Kubernetes, mais vous pouvez également utiliser Amazon Managed Prometheus et Amazon Managed Grafana. kube-prometheus-stack

Si vous utilisez la solution Prometheus `kube-prometheus-stack` ou une solution similaire, vous pouvez installer le même tableau de bord pour SLOs l'observer sur votre cluster en temps réel.

1. Vous devez d'abord installer les règles Prometheus utilisées dans les tableaux de bord avec. `kubectl apply -f prometheus-rules.yaml` Vous pouvez télécharger une copie des règles ici : https://github.com/kubernetes/ perf- tests/blob/master/clusterloader2/pkg/prometheus/manifests/prometheus -rules.yaml
   a. Assurez-vous de vérifier que l'espace de noms du fichier correspond à votre environnement

   b. Vérifiez que les étiquettes correspondent à la valeur du `prometheus.prometheusSpec.ruleSelector` casque si vous utilisez `kube-prometheus-stack`

2. Vous pouvez ensuite installer les tableaux de bord dans Grafana. Les tableaux de bord json et les scripts python permettant de les générer sont disponibles ici : https://github.com/kubernetes/ perf-tests/tree/master/clusterloader2/pkg/prometheus/manifests/dashboards

   a. le `slo.json` tableau de bord affiche les performances du cluster par rapport aux Kubernetes SLOs

Sachez qu' SLOs ils se concentrent sur les performances des composants Kubernetes de vos clusters, mais qu'il existe des indicateurs supplémentaires que vous pouvez consulter et qui fournissent différentes perspectives ou informations sur votre cluster. Les projets communautaires Kubernetes tels que K ube-state-metrics peuvent vous aider à analyser rapidement les tendances de votre cluster. Les plugins et pilotes les plus courants de la communauté Kubernetes émettent également des métriques Prometheus, ce qui vous permet d'étudier des éléments tels que les autoscalers ou les planificateurs personnalisés.

Le guide des meilleures pratiques d'observabilité contient des exemples d'autres indicateurs Kubernetes que vous pouvez utiliser pour mieux comprendre.

Limites connues et Quotas de Service

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Amazon EKS peut être utilisé pour diverses charges de travail et peut interagir avec un large éventail de services AWS. Nous avons constaté que les charges de travail des clients étaient confrontées à une gamme similaire de quotas de services AWS et à d'autres problèmes entravant l'évolutivité.

Votre compte AWS possède des quotas par défaut (limite supérieure du nombre de ressources AWS que votre équipe peut demander). Chaque service AWS définit son propre quota, qui est généralement spécifique à une région. Vous pouvez demander des augmentations pour certains quotas (limites souples), tandis que d'autres quotas ne peuvent pas être augmentés (limites strictes). Vous devez tenir compte de ces valeurs lorsque vous concevez l'architecture de vos applications.

Pensez à revoir régulièrement ces limites de service et à les intégrer lors de la conception de votre application.

Vous pouvez vérifier l'utilisation de votre compte et ouvrir une demande d'augmentation de quota sur la console AWS Service Quotas ou à l'aide de l'interface de ligne de commande AWS. Reportez-vous à la documentation AWS du service AWS concerné pour plus de détails sur les Quotas de Service et sur toute autre restriction ou notification concernant leur augmentation.

> **Note**
>
> Amazon EKS Service Quotas répertorie les quotas de service et contient des liens permettant de demander des augmentations, le cas échéant.

Autres quotas de Service AWS

Nous avons constaté que les clients d'EKS étaient affectés par les quotas listés ci-dessous pour d'autres services AWS. Certains d'entre eux ne s'appliquent qu'à des cas d'utilisation ou à des configurations spécifiques, mais vous pouvez vous demander si votre solution rencontrera l'un de ces problèmes au fur et à mesure de son évolution. Les quotas sont organisés par service et chaque quota possède un identifiant au format L-XXXXXXXX que vous pouvez utiliser pour le rechercher dans la console AWS Service Quotas.

Service	Quota (L-xxxxx)	Impact	Numéro d'identification (L-xxxxx)	default
IAM	Rôles par compte	Peut limiter le nombre de clusters ou de rôles IRSA dans un compte.	FE177L-D64	1 000
IAM	OpenId connecter les fournisseurs par compte	Peut limiter le nombre de clusters par compte, OpenID Connect est utilisé par l'IRSA	L-858F3967	100

Service	Quota (L-xxxxx)	Impact	Numéro d'identification (L-xxxxx)	default
IAM	Longueur de la politique d'approbation du rôle	Peut limiter le nombre de clusters auxquels un rôle IAM est associé pour IRSA	L-C07B4B0D	2 048
VPC	Groupes de sécurité par interface réseau	Peut limiter le contrôle ou la connectivité du réseau de votre cluster	L-2 AFB9258	5
VPC	IPv4 Blocs CIDR par VPC	Peut limiter le nombre de nœuds de travail EKS	L-83 A9D CA0	5
VPC	Tableau des itinéraires par itinéraire	Peut limiter le contrôle ou la connectivité du réseau de votre cluster	L-93826ACB	50
VPC	Connexions d'appairage de VPC actives par VPC	Peut limiter le contrôle ou la connectivité du réseau de votre cluster	L-7E9ECCDB	50

Service	Quota (L-xxxxx)	Impact	Numéro d'identification (L-xxxxx)	default
VPC	Règles entrantes ou sortantes par groupe de sécurité.	Certains contrôleurs d'EKS peuvent limiter le contrôle ou la connectivité du réseau de votre cluster. Certains contrôleurs d'EKS créent de nouvelles règles	L-0F EA8095	50
VPC	VPCs par région	Peut limiter le nombre de clusters par compte ou le contrôle ou la connectivité du réseau de votre cluster	L-F678F1CE	5
VPC	Passerelles Internet par région	Peut limiter le nombre de clusters par compte ou le contrôle ou la connectivité du réseau de votre cluster	L-A4707A72	5

Service	Quota (L-xxxxx)	Impact	Numéro d'identification (L-xxxxx)	default
VPC	Interfaces réseau par région	Peut limiter le nombre de nœuds EKS Worker ou avoir un impact sur les scaling/update activités du plan de contrôle EKS.	DF5L-E4 CA3	5 000
VPC	Utilisation des adresses réseau	Peut limiter le nombre de clusters par compte ou le contrôle ou la connectivité du réseau de votre cluster	BB24L-F6E5	64 000
VPC	Utilisation des adresses réseau appairées	Peut limiter le nombre de clusters par compte ou le contrôle ou la connectivité du réseau de votre cluster	CD17FD4L-B	128 000
ELB	Écouteurs par Network Load Balancer	Peut limiter le contrôle de l'entrée du trafic vers le cluster.	L-57A373D6	50

Service	Quota (L-xxxxx)	Impact	Numéro d'identification (L-xxxxx)	default
ELB	Groupes cibles par région	Peut limiter le contrôle de l'entrée du trafic vers le cluster.	L-B22855CB	3 000
ELB	Cibles par Application Load Balancer	Peut limiter le contrôle de l'entrée du trafic vers le cluster.	L-7E6692B2	1 000
ELB	Cibles par Network Load Balancer	Peut limiter le contrôle de l'entrée du trafic vers le cluster.	L- EEF1 AD04	3 000
ELB	Cibles par zone de disponibilité et par Network Load Balancer	Peut limiter le contrôle de l'entrée du trafic vers le cluster.	L-B211E961	500
ELB	Objectifs par groupe cible et par région	Peut limiter le contrôle de l'entrée du trafic vers le cluster.	L-A0D0B863	1 000
ELB	Application Load Balancers par région	Peut limiter le contrôle de l'entrée du trafic vers le cluster.	L-53 B97 DA6	50
ELB	Classic Load Balancer par région	Peut limiter le contrôle de l'entrée du trafic vers le cluster.	L-E9E9831D	20

Service	Quota (L-xxxxx)	Impact	Numéro d'identification (L-xxxxx)	default
ELB	Équilibreurs de charge réseau par région	Peut limiter le contrôle de l'entrée du trafic vers le cluster.	L-69A177A2	50
EC2	Exécution d'instances standard à la demande (A, C, D, H, I, M, R, T, Z) (sous forme de nombre maximal de vCPU)	Peut limiter le nombre de nœuds de travail EKS	L-1216C47A	5
EC2	Toutes les demandes d'instances ponctuelles standard (A, C, D, H, I, M, R, T, Z) (sous forme de nombre maximal de vCPU)	Peut limiter le nombre de nœuds de travail EKS	L-34B43A08	5
EC2	Elastique EC2-VPC IPs	Peut limiter le nombre de NAT GWs (et donc VPCs), ce qui peut limiter le nombre de clusters dans une région	L-0263D0A3	5

Service	Quota (L-xxxxx)	Impact	Numéro d'identification (L-xxxxx)	default
EBS	Instantanés par région	Peut limiter la stratégie de sauvegarde pour les charges de travail dynamiques	L-309 BACF6	100 000
EBS	Stockage pour les volumes SSD à usage général (gp3), en TiB	Peut limiter le nombre de nœuds de travail EKS ou le PersistentVolume stockage	L-7A658B76	50
EBS	Stockage pour les volumes SSD à usage général (gp2), en TiB	Peut limiter le nombre de nœuds de travail EKS ou le PersistentVolume stockage	L-D18D FCD1	50
ECR	Référentiels enregistrés	Peut limiter le nombre de charges de travail dans vos clusters	CFEB8L-E8D	100 000
ECR	Images par référentiel	Peut limiter le nombre de charges de travail dans vos clusters	L-03A36 CE1	20 000

Service	Quota (L-xxxxx)	Impact	Numéro d'identification (L-xxxxx)	default
SecretsManager	Secrets par région	Peut limiter le nombre de charges de travail dans vos clusters	L-2F66C23C	500 000

Limitation des demandes AWS

Les services AWS mettent également en œuvre la limitation des demandes afin de garantir leur performance et leur disponibilité pour tous les clients. À l'instar des Quotas de Service, chaque service AWS maintient ses propres seuils de limitation des demandes. Pensez à consulter la documentation du service AWS correspondant si vos charges de travail doivent rapidement émettre un grand nombre d'appels d'API ou si vous remarquez des erreurs de limitation des demandes dans votre application.

Les requêtes d'API EC2 concernant le provisionnement d'interfaces réseau ou d'adresses IP EC2 peuvent être limitées dans les grands clusters ou lorsque les clusters évoluent de manière drastique. Le tableau ci-dessous présente certaines des actions d'API qui ont entraîné une limitation des demandes de nos clients. Vous pouvez consulter les valeurs par défaut de la limite de débit EC2 et les étapes à suivre pour demander une augmentation de la limite de débit dans la documentation EC2 sur la limitation du débit.

Actions mutantes	Actions en lecture seule
AssignPrivateIpAddresses	DescribeDhcpOptions
AttachNetworkInterface	DescribeInstances
CreateNetworkInterface	DescribeNetworkInterfaces
DeleteNetworkInterface	DescribeSecurityGroups
DeleteTags	DescribeTags
DetachNetworkInterface	DescribeVpcs

Actions mutantes	Actions en lecture seule
ModifyNetworkInterfaceAttribute	DescribeVolumes
UnassignPrivateIpAddresses	

Autres limites connues

- Route 53 a également une limite de débit relativement faible de 5 requêtes par seconde adressées à l'API Route 53. Si vous avez un grand nombre de domaines à mettre à jour dans le cadre d'un projet tel que le DNS externe, vous risquez de constater une limitation du débit et des retards dans la mise à jour des domaines.
  - Certains types d'instances Nitro ont une limite de 28 pièces jointes qui est partagée entre les volumes Amazon EBS, les interfaces réseau et les volumes de stockage d' NVMe instances. Si vos charges de travail génèrent de nombreux volumes EBS, la densité de pods que vous pouvez atteindre avec ces types d'instances peut être limitée.
  - Un nombre maximum de connexions peuvent être suivies par instance Ec2. Si vos charges de travail gèrent un grand nombre de connexions, vous risquez de rencontrer des échecs ou des erreurs de communication car ce maximum a été atteint. Vous pouvez utiliser les mesures de performance du `conntrack_allowance_exceeded` réseau `conntrack_allowance_available` et celles du réseau pour surveiller le nombre de connexions suivies sur vos nœuds de travail EKS.
  - Dans l'environnement EKS, la limite de stockage etcd est de 8 GiB selon les directives en amont. Veuillez surveiller la métrique `apiserver_storage_size_bytes` pour suivre la taille de la base de données ETCD. Vous pouvez vous référer aux règles d'alerte `etcdBackendQuotaLowSpace` et `etcdExcessiveDatabaseGrowth` configurer cette surveillance.

Meilleures pratiques pour les mises à niveau de clusters

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Ce guide explique aux administrateurs de clusters comment planifier et exécuter leur stratégie de mise à niveau Amazon EKS. Il décrit également comment mettre à niveau les nœuds autogérés, les groupes de nœuds gérés, les nœuds Karpenter et les nœuds Fargate. Il ne contient pas de conseils sur EKS Anywhere, les Kubernetes autogérés, les Outposts AWS ou les Zones Locales AWS.

Présentation de

Une version de Kubernetes englobe à la fois le plan de contrôle et le plan de données. Pour garantir le bon fonctionnement, le plan de contrôle et le plan de données doivent exécuter la même version mineure de Kubernetes, telle que la version 1.24. Pendant qu'AWS gère et met à niveau le plan de contrôle, vous êtes responsable de la mise à jour des nœuds de travail dans le plan de données.

- Plan de contrôle : la version du plan de contrôle est déterminée par le serveur API Kubernetes. Dans les clusters Amazon EKS, AWS se charge de gérer ce composant. Les mises à niveau du plan de contrôle peuvent être initiées via l'API AWS.
- Plan de données — La version du plan de données est associée aux versions de Kubelet exécutées sur vos nœuds individuels. Il est possible que des nœuds d'un même cluster exécutent différentes versions. Vous pouvez vérifier les versions de tous les nœuds en exécutant`kubectl get nodes`.

Avant la mise à niveau

Si vous envisagez de mettre à niveau votre version de Kubernetes dans Amazon EKS, vous devez mettre en place quelques politiques, outils et procédures importants avant de commencer une mise à niveau.

- Comprenez les politiques de dépréciation : acquérez une compréhension approfondie du fonctionnement de la politique d'obsolescence de Kubernetes. Soyez au courant de tout changement à venir susceptible d'affecter vos applications existantes. Les nouvelles versions

de Kubernetes suppriment souvent certaines fonctionnalités, ce qui peut entraîner APIs des problèmes d'exécution des applications.

- Consultez le journal des modifications de Kubernetes : examinez attentivement le journal des modifications de Kubernetes ainsi que les versions d'Amazon EKS Kubernetes afin de comprendre tout impact possible sur votre cluster, tel que les modifications majeures susceptibles d'affecter vos charges de travail.
- Évaluez la compatibilité des modules complémentaires du cluster : Amazon EKS ne met pas automatiquement à jour un module complémentaire lorsque de nouvelles versions sont publiées ou après la mise à jour de votre cluster vers une nouvelle version mineure de Kubernetes. Consultez Mettre à jour un module complémentaire pour comprendre la compatibilité de tous les modules complémentaires de cluster existants avec la version du cluster vers laquelle vous souhaitez effectuer la mise à niveau.
- Activer la journalisation du plan de contrôle : activez la journalisation du plan de contrôle pour capturer les journaux, les erreurs ou les problèmes susceptibles de survenir au cours du processus de mise à niveau. Pensez à consulter ces journaux pour détecter toute anomalie. Testez les mises à niveau de clusters dans un environnement hors production ou intégrez des tests automatisés dans votre flux de travail d'intégration continue pour évaluer la compatibilité des versions avec vos applications, vos contrôleurs et vos intégrations personnalisées.
- Découvrez eksctl pour la gestion de clusters — Envisagez d'utiliser eksctl pour gérer votre cluster EKS. Il vous permet de mettre à jour le plan de contrôle, de gérer les modules complémentaires et de gérer les mises à jour des nœuds de travailout-of-the-box.
- Optez pour le mode automatique ou les groupes de nœuds gérés : rationalisez et automatisez les mises à niveau des nœuds de travail en utilisant le mode automatique ou les groupes de nœuds gérés par EKS. Ces options simplifient le processus et réduisent les interventions manuelles.
- Utiliser le plugin kubectl Convert — Tirez parti du plugin kubectl convert pour faciliter la conversion des fichiers manifestes de Kubernetes entre les différentes versions de l'API. Cela permet de garantir que vos configurations restent compatibles avec la nouvelle version de Kubernetes.

Conservez votre cluster up-to-date

Il est essentiel de se tenir au courant des mises à jour de Kubernetes pour créer un environnement EKS sécurisé et efficace, reflétant le modèle de responsabilité partagée d'Amazon EKS. En intégrant ces stratégies à votre flux de travail opérationnel, vous vous positionnez pour maintenir up-to-date des clusters sécurisés qui tirent pleinement parti des dernières fonctionnalités et améliorations. Tactiques :

- Politique relative aux versions prises en charge : conformément à la communauté Kubernetes, Amazon EKS propose généralement trois versions actives de Kubernetes. Une version mineure de Kubernetes est prise en charge en standard dans Amazon EKS pendant les 14 premiers mois suivant sa sortie. Une fois la date de fin du support standard passée, la version bénéficie d'un support étendu pendant les 12 mois suivants. Les avis d'obsolescence sont émis au moins 60 jours avant qu'une version n'atteigne sa date de fin de support standard. Pour plus de détails, reportez-vous à la documentation sur le cycle de vie des versions d'EKS.
- Politique de mise à niveau automatique — Nous vous recommandons vivement de rester synchronisé avec les mises à jour de Kubernetes dans votre cluster EKS. Amazon EKS mettra automatiquement à niveau vers la version suivante les clusters fonctionnant sur une version Kubernetes dont le cycle de vie de 26 mois (14 mois de support standard plus 12 mois de support étendu) est terminé. Notez que vous pouvez désactiver le support étendu. Le fait de ne pas procéder à une mise à niveau proactive avant qu'une version ne end-of-life déclenche une mise à niveau automatique, ce qui pourrait perturber vos charges de travail et vos systèmes. Pour plus d'informations, consultez la version EKS FAQs.
- Créez des runbooks de mise à niveau : établissez un processus bien documenté pour gérer les mises à niveau. Dans le cadre de votre approche proactive, développez des runbooks et des outils spécialisés adaptés à votre processus de mise à niveau. Cela améliore non seulement votre préparation, mais simplifie également les transitions complexes. Prenez l'habitude de mettre à niveau vos clusters au moins une fois par an. Cette pratique vous permet de suivre les avancées technologiques en cours, renforçant ainsi l'efficacité et la sécurité de votre environnement.

Consultez le calendrier de publication d'EKS

Consultez le calendrier de publication d'EKS Kubernetes pour savoir quand de nouvelles versions seront disponibles et quand le support pour des versions spécifiques prendra fin. En général, EKS publie trois versions mineures de Kubernetes par an, et chaque version mineure est prise en charge pendant environ 14 mois.

Consultez également les informations relatives à la version de Kubernetes en amont.

Comprendre comment le modèle de responsabilité partagée s'applique aux mises à niveau de clusters

Vous êtes chargé de lancer la mise à niveau à la fois du plan de contrôle du cluster et du plan de données. Découvrez comment lancer une mise à niveau. Lorsque vous lancez une mise à niveau de

cluster, AWS gère la mise à niveau du plan de contrôle du cluster. Vous êtes responsable de la mise à niveau du plan de données, y compris les modules Fargate et les addons. Vous devez valider et planifier les mises à niveau pour les charges de travail exécutées sur votre cluster afin de garantir que leur disponibilité et leurs opérations ne sont pas affectées après la mise à niveau du cluster

Mettez à niveau les clusters sur place

EKS prend en charge une stratégie de mise à niveau du cluster sur place. Cela permet de maintenir les ressources du cluster et de maintenir la cohérence de la configuration du cluster (par exemple, point de terminaison d'API, OIDC ENIs, équilibreurs de charge). Cela perturbe moins les utilisateurs du cluster et utilisera les charges de travail et les ressources existantes du cluster sans que vous ayez à redéployer les charges de travail ou à migrer des ressources externes (par exemple, DNS, stockage).

Lorsque vous effectuez une mise à niveau de cluster sur place, il est important de noter qu'une seule mise à niveau de version mineure peut être exécutée à la fois (par exemple, de la version 1.24 à la version 1.25).

Cela signifie que si vous devez mettre à jour plusieurs versions, une série de mises à niveau séquentielles sera nécessaire. La planification des mises à niveau séquentielles est plus compliquée et présente un risque accru d'interruptions de service. Dans ce cas, voirthe section called “Évaluez les Blue/Green clusters comme alternative aux mises à niveau de clusters sur place”.

Améliorez votre plan de contrôle et votre plan de données en séquence

Pour mettre à niveau un cluster, vous devez effectuer les actions suivantes :

1. Consultez les notes de mise à jour de Kubernetes et d'EKS.
2. Effectuez une sauvegarde du cluster. (facultatif)
3. Identifiez et corrigez l'utilisation des API déconseillée ou supprimée dans vos charges de travail.
4. Assurez-vous que les groupes de nœuds gérés, s'ils sont utilisés, sont sur la même version de Kubernetes que le plan de contrôle. Les groupes de nœuds gérés par EKS et les nœuds créés par EKS Fargate Profiles prennent en charge deux versions mineures d'écart entre le plan de contrôle et le plan de données pour les versions 1.27 et antérieures de Kubernetes. À partir de la version 1.28, les groupes de nœuds gérés par EKS et les nœuds créés par EKS Fargate Profiles prennent

en charge 3 versions mineures entre le plan de contrôle et le plan de données. Par exemple, si la version de votre plan de contrôle EKS est 1.28, vous pouvez utiliser en toute sécurité des versions de kubelet aussi anciennes que 1.25. Si votre version d'EKS est 1.27, la version la plus ancienne de kubelet que vous pouvez utiliser est la 1.25.

5. Mettez à niveau le plan de contrôle du cluster à l'aide de la console AWS ou de l'interface de ligne de commande.
6. Vérifiez la compatibilité des modules complémentaires. Mettez à niveau vos modules complémentaires Kubernetes et vos contrôleurs personnalisés, selon les besoins.
7. Mettez à jour kubectl.
8. Mettez à niveau le plan de données du cluster. Mettez à niveau vos nœuds vers la même version mineure de Kubernetes que votre cluster mis à niveau.

> **Tip**
>
> Si votre cluster a été créé à l'aide du mode automatique EKS, vous n'avez pas besoin de mettre à niveau votre plan de données de cluster. Après la mise à niveau de votre plan de contrôle, le mode automatique d'EKS commencera à mettre à jour progressivement les nœuds gérés tout en respectant tous les budgets d'interruption des modules. Assurez-vous de surveiller ces mises à jour pour vérifier la conformité avec vos exigences opérationnelles.

Utilisez la documentation EKS pour créer une liste de contrôle de mise à niveau

La documentation de la version d'EKS Kubernetes inclut une liste détaillée des modifications apportées à chaque version. Créez une liste de contrôle pour chaque mise à niveau.

Pour obtenir des conseils spécifiques sur la mise à niveau d'une version d'EKS, consultez la documentation pour connaître les modifications et considérations importantes relatives à chaque version.

- Consultez les notes de publication des versions de Kubernetes sur le support standard
- Consultez les notes de mise à jour des versions de Kubernetes sur le support étendu

Mettez à niveau les modules complémentaires et les composants à l'aide de l'API Kubernetes

Avant de mettre à niveau un cluster, vous devez connaître les versions des composants Kubernetes que vous utilisez. Inventoriez les composants du cluster et identifiez les composants qui utilisent directement l'API Kubernetes. Cela inclut les composants critiques du cluster tels que les agents de surveillance et de journalisation, les autoscalers de cluster, les pilotes de stockage de conteneurs (par exemple EBS CSI, EFS CSI), les contrôleurs d'entrée et toute autre charge de travail ou module complémentaire qui repose directement sur l'API Kubernetes.

> **Tip**
>
> Les composants critiques du cluster sont souvent installés dans un espace de `*-system` noms

```
kubectl get ns | grep '-system'
```

Une fois que vous avez identifié les composants qui reposent sur l'API Kubernetes, consultez leur documentation pour connaître la compatibilité des versions et les exigences de mise à niveau. Par exemple, consultez la documentation d'AWS Load Balancer Controller pour connaître la compatibilité des versions. Il peut être nécessaire de mettre à niveau certains composants ou de modifier la configuration avant de procéder à une mise à niveau du cluster. Parmi les composants essentiels à vérifier, citons CoreDNS, kube-proxy, VPC CNI et les pilotes de stockage.

Les clusters contiennent souvent de nombreuses charges de travail qui utilisent l'API Kubernetes et sont nécessaires aux fonctionnalités de charge de travail telles que les contrôleurs d'entrée, les systèmes de livraison continue et les outils de surveillance. Lorsque vous mettez à niveau un cluster EKS, vous devez également mettre à niveau vos modules complémentaires et vos outils tiers pour vous assurer qu'ils sont compatibles.

Consultez les exemples suivants de modules complémentaires courants et la documentation de mise à niveau correspondante :

- Amazon VPC CNI : pour connaître la version recommandée du module complémentaire Amazon VPC CNI pour chaque version de cluster, consultez Mettre à jour le plug-in Amazon VPC CNI pour le module complémentaire autogéré Kubernetes. Lorsqu'il est installé en tant que module complémentaire Amazon EKS, il ne peut être mis à niveau qu'une seule version mineure à la fois.

- kube-proxy : voir Mise à jour du module complémentaire autogéré Kubernetes kube-proxy.
- CoreDNS : voir Mise à jour du module complémentaire autogéré CoreDNS.
- AWS Load Balancer Controller : Le AWS Load Balancer Controller doit être compatible avec la version EKS que vous avez déployée. Consultez le guide d'installation pour plus d'informations.
- Pilote Amazon Elastic Block Store (Amazon EBS) Container Storage Interface (CSI) : pour obtenir des informations sur l'installation et la mise à niveau, consultez Gérer le pilote Amazon EBS CSI en tant que module complémentaire Amazon EKS.
- Pilote Amazon Elastic File System (Amazon EFS) Container Storage Interface (CSI) : pour obtenir des informations sur l'installation et la mise à niveau, consultez le pilote Amazon EFS CSI.
- Serveur de métriques Kubernetes : pour plus d'informations, consultez metrics-server on. GitHub
- Kubernetes Cluster Autoscaler : pour mettre à niveau la version de Kubernetes Cluster Autoscaler, modifiez la version de l'image lors du déploiement. Le Cluster Autoscaler est étroitement lié au planificateur Kubernetes. Vous devrez toujours le mettre à niveau lors de la mise à niveau du cluster. Passez en revue les GitHub versions pour trouver l'adresse de la dernière version correspondant à votre version mineure de Kubernetes.
- Karpenter : pour obtenir des informations sur l'installation et la mise à niveau, consultez la documentation de Karpenter.

> **Tip**
>
> Il n'est pas nécessaire de mettre à niveau manuellement les fonctionnalités d'Amazon EKS Auto Mode, notamment les fonctionnalités de mise à l'échelle automatique du calcul, de stockage par blocs et d'équilibrage de charge.

Vérifiez les exigences de base d'EKS avant la mise à niveau

AWS a besoin de certaines ressources sur votre compte pour terminer le processus de mise à niveau. Si ces ressources ne sont pas présentes, le cluster ne peut pas être mis à niveau. La mise à niveau d'un plan de contrôle nécessite les ressources suivantes :

1. Adresses IP disponibles : Amazon EKS a besoin d'un maximum de cinq adresses IP disponibles provenant des sous-réseaux que vous avez spécifiés lors de la création du cluster afin de mettre à jour le cluster. Si ce n'est pas le cas, mettez à jour la configuration de votre cluster pour inclure de nouveaux sous-réseaux de cluster avant de procéder à la mise à jour de version.

2. Rôle IAM EKS : le rôle IAM du plan de contrôle est toujours présent dans le compte avec les autorisations nécessaires.
3. Si le chiffrement secret est activé sur votre cluster, assurez-vous que le rôle IAM du cluster est autorisé à utiliser la clé AWS Key Management Service (AWS KMS).

Vérifiez les adresses IP disponibles

Pour mettre à jour le cluster, Amazon EKS requiert jusqu'à cinq adresses IP disponibles correspondant aux sous-réseaux que vous avez spécifiés lors de la création de votre cluster.

Pour vérifier que vos sous-réseaux disposent de suffisamment d'adresses IP pour mettre à niveau le cluster, vous pouvez exécuter la commande suivante :

```
CLUSTER=<cluster name>
aws ec2 describe-subnets --subnet-ids \
  $(aws eks describe-cluster --name ${CLUSTER} \
  --query 'cluster.resourcesVpcConfig.subnetIds' \
  --output text) \
  --query 'Subnets[*].[SubnetId,AvailabilityZone,AvailableIpAddressCount]' \
  --output table

----------------------------------------------------
|                  DescribeSubnets                 |
+---------------------------+--------------+-------+
|  subnet-067fa8ee8476abbd6 |  us-east-1a  |  8184 |
|  subnet-0056f7403b17d2b43 |  us-east-1b  |  8153 |
|  subnet-09586f8fb3addbc8c |  us-east-1a  |  8120 |
|  subnet-047f3d276a22c6bce |  us-east-1b  |  8184 |
+---------------------------+--------------+-------+
```

Le VPC CNI Metrics Helper peut être utilisé pour créer un tableau de bord CloudWatch pour les métriques VPC. Amazon EKS recommande de mettre à jour les sous-réseaux du cluster à l'aide de l'API UpdateClusterConfiguration « » avant de commencer une mise à niveau de version de Kubernetes si vous manquez d'adresses IP dans les sous-réseaux initialement spécifiés lors de la création du cluster. Vérifiez que les nouveaux sous-réseaux qui vous seront fournis :

- appartiennent au même ensemble de ceux AZs sélectionnés lors de la création du cluster.
- appartiennent au même VPC fourni lors de la création du cluster

Envisagez d'associer des blocs d'adresse CIDR supplémentaires si les adresses IP du bloc d'adresse CIDR VPC existant sont épuisées. AWS permet d'associer des blocs CIDR supplémentaires à votre VPC de cluster existant, élargissant ainsi efficacement votre pool d'adresses IP. Cette extension peut être réalisée en introduisant des plages d'adresses IP privées supplémentaires (RFC 1918) ou, si nécessaire, des plages d'adresses IP publiques (non RFC 1918). Vous devez ajouter de nouveaux blocs d'adresse CIDR VPC et laisser l'actualisation du VPC se terminer avant qu'Amazon EKS puisse utiliser le nouveau CIDR. Ensuite, vous pouvez mettre à jour les sous-réseaux en fonction des blocs CIDR nouvellement configurés pour le VPC.

Vérifier le rôle EKS IAM

Pour vérifier que le rôle IAM est disponible et que la politique d'acceptation des rôles est correcte dans votre compte, vous pouvez exécuter les commandes suivantes :

```
CLUSTER=<cluster name>
ROLE_ARN=$(aws eks describe-cluster --name ${CLUSTER} \
  --query 'cluster.roleArn' --output text)
aws iam get-role --role-name ${ROLE_ARN##*/} \
  --query 'Role.AssumeRolePolicyDocument'

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "eks.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Migrer vers les modules complémentaires EKS

Amazon EKS installe automatiquement des modules complémentaires tels que le plug-in Amazon VPC CNI pour Kubernetes et CoreDNS pour `kube-proxy` chaque cluster. Les modules complémentaires peuvent être autogérés ou installés en tant que modules complémentaires Amazon EKS. Les modules complémentaires Amazon EKS constituent un autre moyen de gérer les modules complémentaires à l'aide de l'API EKS.

Vous pouvez utiliser les modules complémentaires Amazon EKS pour mettre à jour les versions à l'aide d'une seule commande. Par exemple :

```
aws eks update-addon –cluster-name my-cluster –addon-name vpc-cni –addon-version
 version-number \
--service-account-role-arn arn:aws:iam::111122223333:role/role-name –configuration-
values '{}' –resolve-conflicts PRESERVE
```

Vérifiez si vous avez des modules complémentaires EKS avec :

```
aws eks list-addons --cluster-name <cluster name>
```

> **Warning**
>
> Les modules complémentaires EKS ne sont pas automatiquement mis à niveau lors d'une mise à niveau du plan de contrôle. Vous devez lancer les mises à jour des modules complémentaires EKS et sélectionner la version souhaitée.

- Il vous incombe de sélectionner une version compatible parmi toutes les versions disponibles. Consultez les instructions relatives à la compatibilité des versions complémentaires.
- Les modules complémentaires Amazon EKS ne peuvent être mis à niveau qu'une seule version mineure à la fois.

Apprenez-en davantage sur les composants disponibles sous forme de modules complémentaires EKS et sur la façon de démarrer.

Découvrez comment fournir une configuration personnalisée à un module complémentaire EKS.

Identifiez et corrigez l'utilisation supprimée de l'API avant de mettre à niveau le plan de contrôle

Vous devez identifier l'utilisation de l'API supprimée APIs avant de mettre à niveau votre plan de contrôle EKS. Pour ce faire, nous vous recommandons d'utiliser des outils capables de vérifier un cluster en cours d'exécution ou des fichiers manifestes Kubernetes statiques et rendus.

L'exécution de la vérification par rapport aux fichiers manifestes statiques est généralement plus précise. S'ils sont exécutés sur des clusters actifs, ces outils peuvent renvoyer des faux positifs.

Une API Kubernetes obsolète ne signifie pas qu'elle a été supprimée. Consultez la politique de dépréciation de Kubernetes pour comprendre comment la suppression d'API affecte vos charges de travail.

Informations sur les clusters

Cluster Insights est une fonctionnalité qui fournit des informations sur les problèmes susceptibles d'avoir une incidence sur la capacité de mise à niveau d'un cluster EKS vers des versions plus récentes de Kubernetes. Ces résultats sont sélectionnés et gérés par Amazon EKS et proposent des recommandations sur la manière d'y remédier. En tirant parti de Cluster Insights, vous pouvez minimiser les efforts consacrés à la mise à niveau vers les nouvelles versions de Kubernetes.

Pour consulter les informations d'un cluster EKS, vous pouvez exécuter la commande suivante :

```
aws eks list-insights --region <region-code> --cluster-name <my-cluster>

{
    "insights": [
        {
            "category": "UPGRADE_READINESS",
            "name": "Deprecated APIs removed in Kubernetes v1.29",
            "insightStatus": {
                "status": "PASSING",
                "reason": "No deprecated API usage detected within the last 30 days."
            },
            "kubernetesVersion": "1.29",
            "lastTransitionTime": 1698774710.0,
            "lastRefreshTime": 1700157422.0,
            "id": "123e4567-e89b-42d3-a456-579642341238",
            "description": "Checks for usage of deprecated APIs that are scheduled for
 removal in Kubernetes v1.29. Upgrading your cluster before migrating to the updated
 APIs supported by v1.29 could cause application impact."
        }
    ]
}
```

Pour obtenir un résultat plus descriptif sur les informations reçues, vous pouvez exécuter la commande suivante :

```
aws eks describe-insight --region <region-code> --id <insight-id> --cluster-name <my-
cluster>
```

Vous avez également la possibilité de consulter les informations dans la console Amazon EKS. Après avoir sélectionné votre cluster dans la liste des clusters, les résultats d'analyse se trouvent sous l'`Upgrade Insights`onglet.

Si vous trouvez un aperçu du cluster dans`"status": ERROR`, vous devez résoudre le problème avant d'effectuer la mise à niveau du cluster. Exécutez la `aws eks describe-insight` commande qui partagera les conseils de correction suivants :

Ressources concernées :

```
"resources": [
      {
        "insightStatus": {
          "status": "ERROR"
        },
        "kubernetesResourceUri": "/apis/policy/v1beta1/podsecuritypolicies/null"
      }
]
```

APIs obsolète :

```
"deprecationDetails": [
      {
        "usage": "/apis/flowcontrol.apiserver.k8s.io/v1beta2/flowschemas",
        "replacedWith": "/apis/flowcontrol.apiserver.k8s.io/v1beta3/flowschemas",
        "stopServingVersion": "1.29",
        "clientStats": [],
        "startServingReplacementVersion": "1.26"
      }
]
```

Mesures recommandées à prendre :

```
"recommendation": "Update manifests and API clients to use newer Kubernetes APIs if
 applicable before upgrading to Kubernetes v1.26."
```

L'utilisation des informations du cluster via la console EKS ou la CLI permet d'accélérer le processus de mise à niveau réussie des versions du cluster EKS. Pour en savoir plus, consultez les ressources suivantes :* Documents officiels d'EKS * Blog de lancement de Cluster Insights.

K ube-no-trouble

K ube-no-trouble est un utilitaire de ligne de commande open source avec la commandekubent. Lorsque vous l'exécutez `kubent` sans aucun argument, il utilise votre KubeConfig contexte actuel, scanne le cluster et imprime un rapport avec ce qui APIs sera obsolète et supprimé.

```
kubent

4:17PM INF >>> Kube No Trouble `kubent` <<<
4:17PM INF version 0.7.0 (git sha d1bb4e5fd6550b533b2013671aa8419d923ee042)
4:17PM INF Initializing collectors and retrieving data
4:17PM INF Target K8s version is 1.24.8-eks-ffeb93d
4:l INF Retrieved 93 resources from collector name=Cluster
4:17PM INF Retrieved 16 resources from collector name="Helm v3"
4:17PM INF Loaded ruleset name=custom.rego.tmpl
4:17PM INF Loaded ruleset name=deprecated-1-16.rego
4:17PM INF Loaded ruleset name=deprecated-1-22.rego
4:17PM INF Loaded ruleset name=deprecated-1-25.rego
4:17PM INF Loaded ruleset name=deprecated-1-26.rego
4:17PM INF Loaded ruleset name=deprecated-future.rego
__________________________________________________________________________________________
>>> Deprecated APIs removed in 1.25 <<<
------------------------------------------------------------------------------------------
KIND                NAMESPACE     NAME             API_VERSION      REPLACE_WITH
 (SINCE)
PodSecurityPolicy   <undefined>   eks.privileged   policy/v1beta1   <removed> (1.21.0)
```

Il peut également être utilisé pour analyser les fichiers manifestes statiques et les packages Helm. Il est recommandé de l'exécuter dans `kubent` le cadre d'un processus d'intégration continue (CI) afin d'identifier les problèmes avant le déploiement des manifestes. L'analyse des manifestes est également plus précise que celle des clusters actifs.

Kube-no-trouble fournit un exemple de compte de service et de rôle dotés des autorisations appropriées pour analyser le cluster.

Pluton

Une autre option est Pluto, qui est similaire `kubent` car elle prend en charge l'analyse d'un cluster en direct, de fichiers manifestes, de graphiques de barre et possède une GitHub action que vous pouvez inclure dans votre processus CI.

```
pluto detect-all-in-cluster

NAME             KIND                VERSION          REPLACEMENT   REMOVED
 DEPRECATED   REPL AVAIL
eks.privileged   PodSecurityPolicy   policy/v1beta1                 false     true
    true
```

Ressources

Pour vérifier que votre cluster n'utilise pas la version obsolète APIs avant la mise à niveau, vous devez surveiller :

- métrique `apiserver_requested_deprecated_apis` depuis Kubernetes v1.19 :

```
kubectl get --raw /metrics | grep apiserver_requested_deprecated_apis

apiserver_requested_deprecated_apis{group="policy",removed_release="1.25",resource="podsecurity
 1
```

- événements dans les journaux d'audit `k8s.io/deprecated` définis sur `true` :

```
CLUSTER="<cluster_name>"
QUERY_ID=$(aws logs start-query \
 --log-group-name /aws/eks/${CLUSTER}/cluster \
 --start-time $(date -u --date="-30 minutes" "+%s") # or date -v-30M "+%s" on MacOS \
 --end-time $(date "+%s") \
 --query-string 'fields @message | filter `annotations.k8s.io/deprecated`="true"' \
 --query queryId --output text)

echo "Query started (query id: $QUERY_ID), please hold ..." && sleep 5 # give it some
 time to query

aws logs get-query-results --query-id $QUERY_ID
```

Quelles lignes de sortie seront utilisées si APIs elles sont obsolètes :

```
{
    "results": [
        [
```

```
            {
                "field": "@message",
                "value": "{\"kind\":\"Event\",\"apiVersion\":\"audit.k8s.io/v1\",
\"level\":\"Request\",\"auditID\":\"8f7883c6-b3d5-42d7-967a-1121c6f22f01\",\"stage
\":\"ResponseComplete\",\"requestURI\":\"/apis/policy/v1beta1/podsecuritypolicies?
allowWatchBookmarks=true\\u0026resourceVersion=4131\\u0026timeout=9m19s\
\u0026timeoutSeconds=559\\u0026watch=true\",\"verb\":\"watch\",\"user\":{\"username
\":\"system:apiserver\",\"uid\":\"8aabfade-da52-47da-83b4-46b16cab30fa\",
\"groups\":[\"system:masters\"]},\"sourceIPs\":[\"::1\"],\"userAgent\":\"kube-
apiserver/v1.24.16 (linux/amd64) kubernetes/af930c1\",\"objectRef\":{\"resource
\":\"podsecuritypolicies\",\"apiGroup\":\"policy\",\"apiVersion\":\"v1beta1\"},
\"responseStatus\":{\"metadata\":{},\"code\":200},\"requestReceivedTimestamp\":
\"2023-10-04T12:36:11.849075Z\",\"stageTimestamp\":\"2023-10-04T12:45:30.850483Z\",
\"annotations\":{\"authorization.k8s.io/decision\":\"allow\",\"authorization.k8s.io/
reason\":\"\",\"k8s.io/deprecated\":\"true\",\"k8s.io/removed-release\":\"1.25\"}}"
            },
[...]
```

Mettez à jour les charges de travail Kubernetes. Utilisez kubectl-convert pour mettre à jour les manifestes

Après avoir identifié les charges de travail et les manifestes à mettre à jour, vous devrez peut-être modifier le type de ressource dans vos fichiers manifestes (par exemple PodSecurityPolicies en PodSecurityStandards). Cela nécessitera une mise à jour de la spécification de la ressource et des recherches supplémentaires en fonction de la ressource remplacée.

Si le type de ressource reste le même mais que la version de l'API doit être mise à jour, vous pouvez utiliser la `kubectl-convert` commande pour convertir automatiquement vos fichiers manifestes. Par exemple, pour convertir un ancien déploiement en`apps/v1`. Pour plus d'informations, consultez Installer le plugin de conversion kubectl sur le site Web de Kubernetes.

```
kubectl-convert -f <file> --output-version <group>/<version>
```

Configurez PodDisruptionBudgets et topologySpreadConstraints garantissez la disponibilité de vos charges de travail pendant la mise à niveau du plan de données

Assurez-vous que vos charges de travail sont correctes PodDisruptionBudgetset topologySpreadConstraintsqu'elles sont disponibles pendant la mise à niveau du plan de données.

Toutes les charges de travail ne nécessitent pas le même niveau de disponibilité. Vous devez donc valider l'échelle et les exigences de votre charge de travail.

Assurez-vous que les charges de travail sont réparties dans plusieurs zones de disponibilité et sur plusieurs hôtes avec des écarts de topologie pour garantir que les charges de travail migreront automatiquement vers le nouveau plan de données sans incident.

Voici un exemple de charge de travail pour laquelle 80 % des répliques seront toujours disponibles et qui seront réparties entre les zones et les hôtes

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: myapp
spec:
  minAvailable: "80%"
  selector:
    matchLabels:
      app: myapp
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 10
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - image: public.ecr.aws/eks-distro/kubernetes/pause:3.2
        name: myapp
        resources:
          requests:
            cpu: "1"
            memory: 256M
      topologySpreadConstraints:
      - labelSelector:
```

```
          matchLabels:
            app: host-zone-spread
        maxSkew: 2
        topologyKey: kubernetes.io/hostname
        whenUnsatisfiable: DoNotSchedule
      - labelSelector:
          matchLabels:
            app: host-zone-spread
        maxSkew: 2
        topologyKey: topology.kubernetes.io/zone
        whenUnsatisfiable: DoNotSchedule
```

AWS Resilience Hub a ajouté Amazon Elastic Kubernetes Service (Amazon EKS) en tant que ressource prise en charge. Resilience Hub fournit un emplacement unique pour définir, valider et suivre la résilience de vos applications afin que vous puissiez éviter les temps d'arrêt inutiles causés par des perturbations du logiciel, de l'infrastructure ou des opérations.

Utilisez Managed Node Groups ou Karpenter pour simplifier les mises à niveau du plan de données

Managed Node Groups et Karpenter simplifient tous deux les mises à niveau des nœuds, mais ils adoptent des approches différentes.

Les groupes de nœuds gérés automatisent le provisionnement et la gestion du cycle de vie des nœuds. Cela signifie que vous pouvez créer, mettre à jour automatiquement ou résilier des nœuds en une seule opération.

Dans la configuration par défaut, Karpenter crée automatiquement de nouveaux nœuds à l'aide de la dernière AMI optimisée EKS compatible. À mesure qu'EKS publie la mise à jour d'EKS Optimized AMIs ou que le cluster est mis à niveau, Karpenter commence automatiquement à utiliser ces images. Karpenter implémente également Node Expiry pour mettre à jour les nœuds.

Karpenter peut être configuré pour une utilisation personnalisée AMIs. Si vous utilisez le custom AMIs avec Karpenter, vous êtes responsable de la version de kubelet.

Confirmer la compatibilité des versions avec les nœuds existants et le plan de contrôle

Avant de procéder à une mise à niveau de Kubernetes dans Amazon EKS, il est essentiel de garantir la compatibilité entre vos groupes de nœuds gérés, vos nœuds autogérés et le plan de contrôle. La compatibilité est déterminée par la version de Kubernetes que vous utilisez et varie en fonction des différents scénarios. Tactiques :

- Kubernetes v1.28+ — * À partir de la version 1.28 de Kubernetes, il existe une politique de version plus clémente pour les composants principaux. Plus précisément, l'écart pris en charge entre le serveur d'API Kubernetes et le kubelet a été étendu d'une version mineure, passant de n-2 à n-3. Par exemple, si la version de votre plan de contrôle EKS est 1.28, vous pouvez utiliser en toute sécurité des versions de kubelet aussi anciennes que 1.25. Cette distorsion de version est prise en charge sur AWS Fargate, les groupes de nœuds gérés et les nœuds autogérés. Nous vous recommandons vivement de conserver les versions de votre Amazon Machine Image (AMI) up-to-date pour des raisons de sécurité. Les anciennes versions de kubelet peuvent présenter des risques de sécurité en raison de vulnérabilités et d'expositions communes potentielles (CVEs), qui pourraient l'emporter sur les avantages liés à l'utilisation d'anciennes versions de kubelet.
- Kubernetes < v1.28 — Si vous utilisez une version antérieure à la v1.28, l'écart pris en charge entre le serveur d'API et le kubelet est n-2. Par exemple, si votre version d'EKS est 1.27, la version la plus ancienne de kubelet que vous pouvez utiliser est 1.25. Cette distorsion de version s'applique à AWS Fargate, aux groupes de nœuds gérés et aux nœuds autogérés.

Activer l'expiration des nœuds pour les nœuds gérés par Karpenter

Karpenter met en œuvre les mises à niveau des nœuds en utilisant le concept d'expiration des nœuds. Cela réduit la planification requise pour les mises à niveau des nœuds. Lorsque vous définissez une valeur pour ttlSecondsUntilExpired dans votre fournisseur, cela active l'expiration du nœud. Une fois que les nœuds ont atteint l'âge défini en quelques secondes, ils sont vidés et supprimés en toute sécurité. Cela est vrai même s'ils sont utilisés, ce qui vous permet de remplacer les nœuds par des instances mises à niveau récemment provisionnées. Lorsqu'un nœud est remplacé, Karpenter utilise la dernière version optimisée pour AMIs EKS. Pour plus d'informations, consultez la section Déprovisionnement sur le site Web de Karpenter.

Karpenter n'ajoute pas automatiquement de l'instabilité à cette valeur. Pour éviter une interruption excessive de la charge de travail, définissez un budget d'interruption des pods, comme indiqué dans la documentation de Kubernetes.

Si vous configurez ttlSecondsUntilExpired sur un fournisseur, cela s'applique aux nœuds existants associés au fournisseur.

Utiliser la fonctionnalité Drift pour les nœuds gérés par Karpenter

La fonction Drift de Karpenter peut automatiquement mettre à niveau les nœuds approvisionnés par Karpenter afin de rester synchronisés avec le plan de contrôle EKS. Karpenter Drift doit actuellement être activé à l'aide d'un portail de fonctionnalités. La configuration par défaut de Karpenter utilise la dernière AMI optimisée pour EKS pour la même version majeure et mineure que le plan de contrôle du cluster EKS.

Une fois la mise à niveau d'un cluster EKS terminée, la fonction Karpenter's Drift détecte que les nœuds approvisionnés par Karpenter utilisent EKS optimisé AMIs pour la version précédente du cluster, et ferme, vide et remplace automatiquement ces nœuds. Pour faciliter le déplacement des pods vers de nouveaux nœuds, suivez les meilleures pratiques de Kubernetes en définissant des quotas de ressources appropriés et en utilisant des budgets d'interruption des pods (PDB). Le déprovisionnement de Karpenter préinstallera les nœuds de remplacement en fonction des demandes de ressources du pod et respectera ces exigences lors du PDBs déprovisionnement des nœuds.

Utilisez eksctl pour automatiser les mises à niveau pour les groupes de nœuds autogérés

Les groupes de nœuds autogérés sont des instances EC2 déployées dans votre compte et connectées au cluster en dehors du service EKS. Ils sont généralement déployés et gérés par une forme d'outillage d'automatisation. Pour mettre à niveau des groupes de nœuds autogérés, vous devez vous référer à la documentation de vos outils.

Par exemple, eksctl prend en charge la suppression et le drainage des nœuds autogérés.

Certains outils courants incluent :

- eksctl
- KOP

- Plans EKS

Backup le cluster avant de procéder à la mise à niveau

Les nouvelles versions de Kubernetes apportent des modifications importantes à votre cluster Amazon EKS. Une fois que vous avez mis à niveau un cluster, vous ne pouvez pas le rétrograder.

Velero est un outil open source soutenu par la communauté qui peut être utilisé pour effectuer des sauvegardes de clusters existants et les appliquer à un nouveau cluster.

Notez que vous ne pouvez créer de nouveaux clusters que pour les versions de Kubernetes actuellement prises en charge par EKS. Si la version que votre cluster exécute actuellement est toujours prise en charge et qu'une mise à niveau échoue, vous pouvez créer un nouveau cluster avec la version d'origine et restaurer le plan de données. Notez que les ressources AWS, y compris IAM, ne sont pas incluses dans la sauvegarde de Velero. Ces ressources devraient être recréées.

Redémarrer les déploiements de Fargate après la mise à niveau du plan de contrôle

Pour mettre à niveau les nœuds du plan de données Fargate, vous devez redéployer les charges de travail. Vous pouvez identifier les charges de travail exécutées sur les nœuds Fargate en répertoriant tous les pods avec l'option. `-o wide` Tout nom de nœud commençant par `fargate-` devra être redéployé dans le cluster.

Évaluez les Blue/Green clusters comme alternative aux mises à niveau de clusters sur place

Certains clients préfèrent adopter une stratégie de blue/green mise à niveau. Cela peut avoir des avantages, mais comporte également des inconvénients qui doivent être pris en compte.

Les avantages incluent :

- Possibilité de changer plusieurs versions d'EKS à la fois (par exemple 1.23 à 1.25)
- Possibilité de revenir à l'ancien cluster
- Crée un nouveau cluster qui peut être géré avec des systèmes plus récents (par exemple Terraform)

- Les charges de travail peuvent être migrées individuellement

Certains inconvénients incluent :

- Modification du point de terminaison de l'API et de l'OIDC nécessitant la mise à jour des consommateurs (par exemple kubectl et CI/CD)
- Nécessite l'exécution de 2 clusters en parallèle pendant la migration, ce qui peut s'avérer coûteux et limiter la capacité de la région
- Une meilleure coordination est nécessaire si les charges de travail dépendent les unes des autres pour être migrées ensemble
- Les équilibreurs de charge et les DNS externes ne peuvent pas facilement s'étendre sur plusieurs clusters

Bien que cette stratégie soit possible, elle est plus coûteuse qu'une mise à niveau sur place et nécessite plus de temps pour la coordination et la migration des charges de travail. Cela peut être nécessaire dans certaines situations et doit être planifié avec soin.

Avec des degrés élevés d'automatisation et des systèmes déclaratifs similaires GitOps, cela peut être plus facile à faire. Vous devrez prendre des précautions supplémentaires pour les charges de travail dynamiques afin que les données soient sauvegardées et migrées vers de nouveaux clusters.

Consultez ces articles de blog pour plus d'informations :

- Mise à niveau du cluster Kubernetes : la stratégie de déploiement bleu-vert
- Migration de clusters Amazon EKS bleu/vert ou canari pour les charges de travail ArgoCD apatrides

Suivez les modifications majeures prévues dans le projet Kubernetes — Think ahead

Ne vous contentez pas de regarder la prochaine version. Passez en revue les nouvelles versions de Kubernetes au fur et à mesure de leur publication et identifiez les modifications majeures. Par exemple, certaines applications utilisaient directement l'API docker, et la prise en charge de l'interface CRI (Container Runtime Interface) pour Docker (également connue sous le nom de Dockershim) a été supprimée dans Kubernetes. `1.24` Il faut plus de temps pour se préparer à ce type de changement.

Passez en revue toutes les modifications documentées pour la version vers laquelle vous effectuez la mise à niveau et notez les étapes de mise à niveau requises. Notez également toutes les exigences ou procédures spécifiques aux clusters gérés par Amazon EKS.

- Journal des modifications de Kubernetes

Conseils spécifiques sur les suppressions de fonctionnalités

Suppression de Dockershim en 1.25 - Utilisez le détecteur pour Docker Socket (DDS)

L'AMI optimisée EKS pour la version 1.25 ne prend plus en charge Dockershim. Si vous êtes dépendant de Dockershim, par exemple si vous montez le socket Docker, vous devrez supprimer ces dépendances avant de passer à la version 1.25 de vos nœuds de travail.

Trouvez les instances dans lesquelles vous êtes dépendant du socket Docker avant de passer à la version 1.25. Nous vous recommandons d'utiliser Detector for Docker Socket (DDS), un plugin kubectl. .

Suppression de PodSecurityPolicy la version 1.25 - Migrer vers les normes de sécurité des pods ou vers une solution policy-as-code

`PodSecurityPolicy`était obsolète dans Kubernetes 1.21 et a été supprimé dans Kubernetes 1.25. Si vous l'utilisez PodSecurityPolicy dans votre cluster, vous devez migrer vers les normes de sécurité Kubernetes Pod (PSS) intégrées ou vers une policy-as-code solution avant de mettre à niveau votre cluster vers la version 1.25 afin d'éviter toute interruption de vos charges de travail.

AWS a publié une FAQ détaillée dans la documentation d'EKS.

Passez en revue les meilleures pratiques en matière de normes de sécurité (PSS) et d'admission à la sécurité des pods (PSA).

Consultez le billet de blog PodSecurityPolicy Deprecation sur le site Web de Kubernetes.

Obsolète du pilote de stockage intégré dans la version 1.23 - Migrer vers des pilotes CSI (Container Storage Interface)

L'interface de stockage de conteneurs (CSI) a été conçue pour aider Kubernetes à remplacer ses mécanismes de pilotes de stockage intégrés à l'arborescence existants. La fonction de migration de

l'interface de stockage du conteneur Amazon EBS est activée par défaut sur Amazon EKS `1.23` et sur les versions ultérieures des clusters. Si vos pods s'exécutent sur une version `1.22` ou un cluster antérieur, vous devez installer le pilote Amazon EBS CSI avant de mettre à jour votre cluster vers la version `1.23` afin d'éviter toute interruption de service.

Consultez les questions fréquemment posées sur la migration vers Amazon EBS CSI.

Ressources supplémentaires

ClowdHaus Conseils de mise à niveau EKS

ClowdHaus EKS Upgrade Guidance est une CLI destinée à faciliter la mise à niveau des clusters Amazon EKS. Il peut analyser un cluster pour détecter tout problème potentiel à résoudre avant la mise à niveau.

GoNoGo

GoNoGoest un outil de phase alpha permettant de déterminer le niveau de fiabilité des mises à niveau des modules complémentaires de votre cluster.

Meilleures pratiques en matière d'optimisation des coûts

L'optimisation des coûts consiste à obtenir les résultats de votre entreprise au prix le plus bas. En suivant la documentation de ce guide, vous optimiserez vos charges de travail Amazon EKS.

Consignes générales

Dans le cloud, il existe un certain nombre de directives générales qui peuvent vous aider à optimiser les coûts de vos microservices :

- Assurez-vous que les charges de travail exécutées sur Amazon EKS sont indépendantes des types d'infrastructure spécifiques pour l'exécution de vos conteneurs. Cela vous donnera une plus grande flexibilité en ce qui concerne leur exécution sur les types d'infrastructure les moins coûteux. Lorsque vous utilisez Amazon EKS avec EC2, il peut y avoir des exceptions lorsque des charges de travail nécessitent un type d' EC2 instance spécifique, comme un GPU ou d'autres types d'instances, en raison de la nature de la charge de travail.
- Sélectionnez des instances de conteneur au profil optimal : profilez vos environnements de production ou de pré-production et surveillez les indicateurs critiques tels que le processeur et la mémoire, à l'aide de services tels qu'Amazon CloudWatch Container Insights pour Amazon EKS ou d'outils tiers disponibles dans l'écosystème Kubernetes. Cela nous permettra d'allouer la bonne quantité de ressources et d'éviter le gaspillage de ressources.
- Profitez des différentes options d'achat disponibles sur AWS pour exécuter EKS EC2, par exemple On-Demand, Spot et Savings Plan.

Meilleures pratiques d'optimisation des coûts chez EKS

Il existe trois domaines généraux de bonnes pratiques en matière d'optimisation des coûts dans le cloud :

- Ressources rentables (Auto Scaling, Down Scaling, politiques et options d'achat)
- Sensibilisation aux dépenses (à l'aide d'AWS et d'outils tiers)
- Optimisation au fil du temps (bonne taille)

Comme pour toute directive, il y a des compromis. Assurez-vous de travailler avec votre organisation pour comprendre les priorités relatives à cette charge de travail et les meilleures pratiques les plus importantes.

Comment utiliser ce guide

Ce guide est destiné aux équipes DevOps chargées de la mise en œuvre et de la gestion des clusters EKS et des charges de travail qu'ils prennent en charge. Le guide est organisé en différents domaines de bonnes pratiques pour en faciliter la consommation. Chaque rubrique contient une liste de recommandations, d'outils à utiliser et de meilleures pratiques pour optimiser les coûts de vos clusters EKS. Il n'est pas nécessaire de lire les sujets dans un ordre particulier.

Principaux services AWS et fonctionnalités de Kubernetes

L'optimisation des coûts est prise en charge par les services et fonctionnalités AWS suivants :

- EC2 Types d'instances, Savings Plan (et instances réservées) et instances ponctuelles, à des prix différents.
- Auto Scaling ainsi que les politiques Auto Scaling natives de Kubernetes. Pensez à Savings Plan (instances précédemment réservées) pour des charges de travail prévisibles. Utilisez des magasins de données gérés tels que EBS et EFS pour garantir l'élasticité et la durabilité des données d'application.
- Le tableau de bord de la console Billing and Cost Management ainsi qu'AWS Cost Explorer fournissent une vue d'ensemble de votre utilisation d'AWS. Utilisez AWS Organizations pour obtenir des informations de facturation détaillées. Les détails de plusieurs outils tiers ont également été partagés.
- Amazon CloudWatch Container Metrics fournit des mesures relatives à l'utilisation des ressources par le cluster EKS. Outre le tableau de bord Kubernetes, plusieurs outils de l'écosystème Kubernetes peuvent être utilisés pour réduire le gaspillage.

Ce guide inclut un ensemble de recommandations que vous pouvez utiliser pour améliorer l'optimisation des coûts de votre cluster Amazon EKS.

Commentaires

Ce guide est publié GitHub afin de recueillir les commentaires directs et les suggestions de l'ensemble de la communauté EKS/Kubernetes. Si vous avez une bonne pratique que vous pensez

que nous devrions inclure dans le guide, veuillez signaler un problème ou soumettre un PR dans le GitHub référentiel. Notre intention est de mettre à jour le guide périodiquement au fur et à mesure que de nouvelles fonctionnalités sont ajoutées au service ou lorsqu'une nouvelle bonne pratique évolue.

Cadre d'optimisation des coûts

AWS Cloud Economics est une discipline qui aide les clients à accroître leur efficacité et à réduire leurs coûts grâce à l'adoption de technologies informatiques modernes telles qu'Amazon EKS. La discipline recommande de suivre une méthodologie appelée « cadre de gestion financière dans le cloud (CFM) » qui comprend 4 piliers :

Le pilier See : mesure et responsabilité

Le pilier See est un ensemble fondamental d'activités et de technologies qui définissent comment mesurer, surveiller et responsabiliser les dépenses liées au cloud. Elle est souvent appelée « observabilité », « instrumentation » ou « télémétrie ». Les capacités et les limites de l'infrastructure « d'observabilité » dictent ce qui peut être optimisé. Obtenir une image claire de vos coûts est une première étape essentielle de l'optimisation des coûts, car vous devez savoir par où commencer. Ce type de visibilité orientera également les types d'activités que vous devrez effectuer pour optimiser davantage votre environnement.

Voici un bref aperçu de nos meilleures pratiques pour le pilier See :

- Définissez et gérez une stratégie de balisage pour vos charges de travail.
  - Utilisez le balisage d'instance : le balisage des clusters EKS vous permet de voir les coûts de chaque cluster et de les répartir dans vos rapports sur les coûts et l'utilisation.
- Établissez des rapports et surveillez l'utilisation d'EKS en utilisant des technologies telles que Kubecost.
  - Activez les tableaux de bord Cloud Intelligence. En étiquetant correctement les ressources et en utilisant des visualisations, vous pouvez mesurer et estimer les coûts.
- Répartissez les coûts du cloud aux applications, aux secteurs d'activité (LoBs) et aux sources de revenus.
- Définissez, mesurez et diffusez efficiency/value KPIs avec les parties prenantes de l'entreprise. Par exemple, créez un KPI « métrique unitaire » qui mesure le coût par transaction. Par exemple, un service de covoiturage peut avoir un KPI pour le « coût par trajet ».

Pour plus de détails sur les technologies et activités recommandées associées à ce pilier, veuillez consulter la section Optimisation des coûts - Observabilité de ce guide.

Le pilier de l'épargne : optimisation des coûts

Ce pilier est basé sur les technologies et les capacités développées dans le pilier « Voir ». Les activités suivantes relèvent généralement de ce pilier :

- Identifiez et éliminez les déchets dans votre environnement.
- Architecte et concepteur au service de la rentabilité.
- Choisissez la meilleure option d'achat, par exemple les instances à la demande par rapport aux instances Spot.
- Adaptez-vous en fonction de l'évolution des services : à mesure que les services AWS évoluent, la manière d'utiliser efficacement ces services peut changer. Soyez prêt à vous adapter pour tenir compte de ces changements.

Ces activités étant opérationnelles, elles dépendent fortement des caractéristiques de votre environnement. Posez-vous la question suivante : quels sont les principaux facteurs de coûts ? Quelle est la valeur commerciale de vos différents environnements ? Quelles sont les options d'achat et les choix d'infrastructure, par exemple les types de familles d'instances, les mieux adaptés à chaque environnement ?

Vous trouverez ci-dessous une liste hiérarchisée des facteurs de coûts les plus courants pour les clusters EKS :

1. Coûts de calcul : la combinaison de plusieurs types de familles d'instances, d'options d'achat et l'équilibre entre évolutivité et disponibilité nécessitent une attention particulière. Pour plus d'informations, consultez les recommandations de la section Optimisation des coûts - Calcul de ce guide.
2. Coûts de mise en réseau : l'utilisation de 3 AZs pour les clusters EKS peut potentiellement augmenter les coûts du trafic inter-AZ. Pour nos recommandations sur la manière de trouver un équilibre entre les exigences de haute disponibilité et la réduction des coûts du trafic réseau, veuillez consulter la section Optimisation des coûts - Mise en réseau de ce guide.
3. Coûts de stockage : en fonction de la stateful/stateless nature des charges de travail dans les clusters EKS et de la manière dont les différents types de stockage sont utilisés, le stockage peut être considéré comme faisant partie de la charge de travail. Pour les considérations relatives aux coûts de stockage EKS, veuillez consulter la section Optimisation des coûts - Stockage de ce guide.

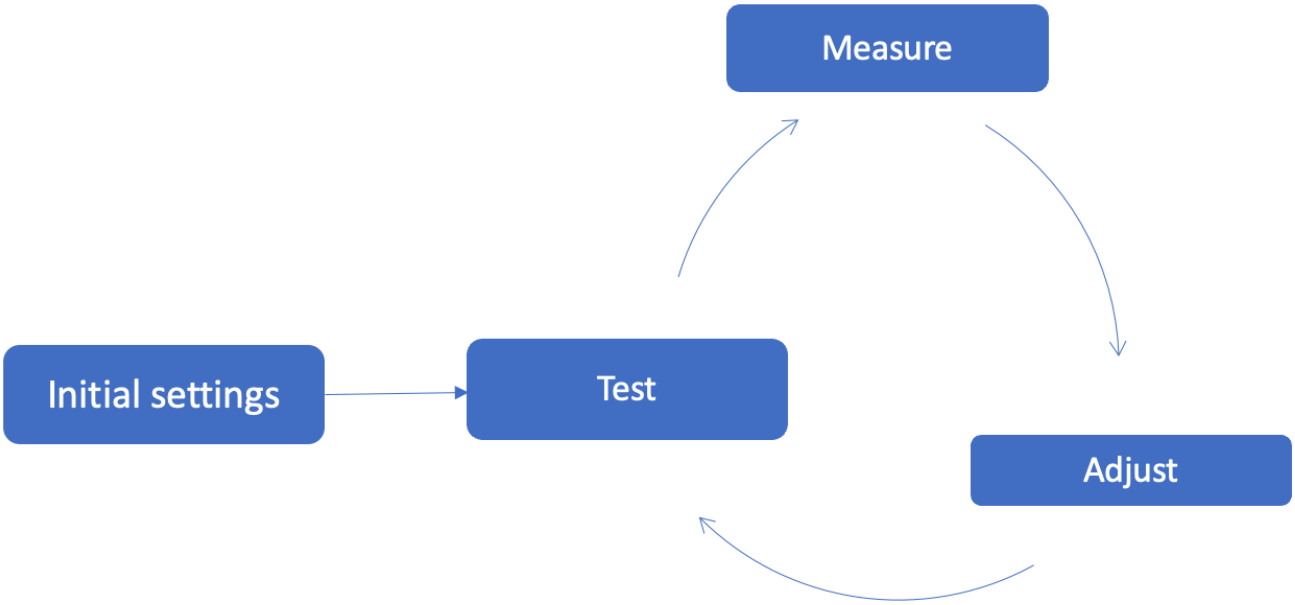
Le pilier du plan : planification et prévisions

Une fois les recommandations du pilier See mises en œuvre, les clusters sont optimisés en permanence. Au fur et à mesure que l'on acquiert de l'expérience dans l'exploitation efficace des clusters, les activités de planification et de prévision peuvent se concentrer sur :

- Budgétiser et prévoir les coûts du cloud de manière dynamique.
- Quantification de la valeur commerciale fournie par les services de conteneurs EKS.
- Intégrer la gestion des coûts du cluster EKS à la planification de la gestion financière informatique.

Le pilier Run

L'optimisation des coûts est un processus continu qui implique un ensemble d'améliorations progressives :



Obtenir le parrainage exécutif pour ce type d'activités est crucial pour intégrer l'optimisation du cluster EKS dans les « FinOps » efforts de l'organisation. Il permet l'alignement des parties prenantes grâce à une compréhension commune des coûts du cluster EKS, à la mise en œuvre de garde-fous en matière de coûts du cluster EKS et à la garantie que l'outillage, l'automatisation et les activités évoluent en fonction des besoins de l'organisation.

Références

- Économie du cloud AWS, gestion financière du cloud

Sensibilisation aux dépenses

La connaissance des dépenses consiste à comprendre qui, où et quoi est à l'origine des dépenses dans votre cluster EKS. L'obtention d'une image précise de ces données vous aidera à mieux connaître vos dépenses et à mettre en évidence les domaines à corriger.

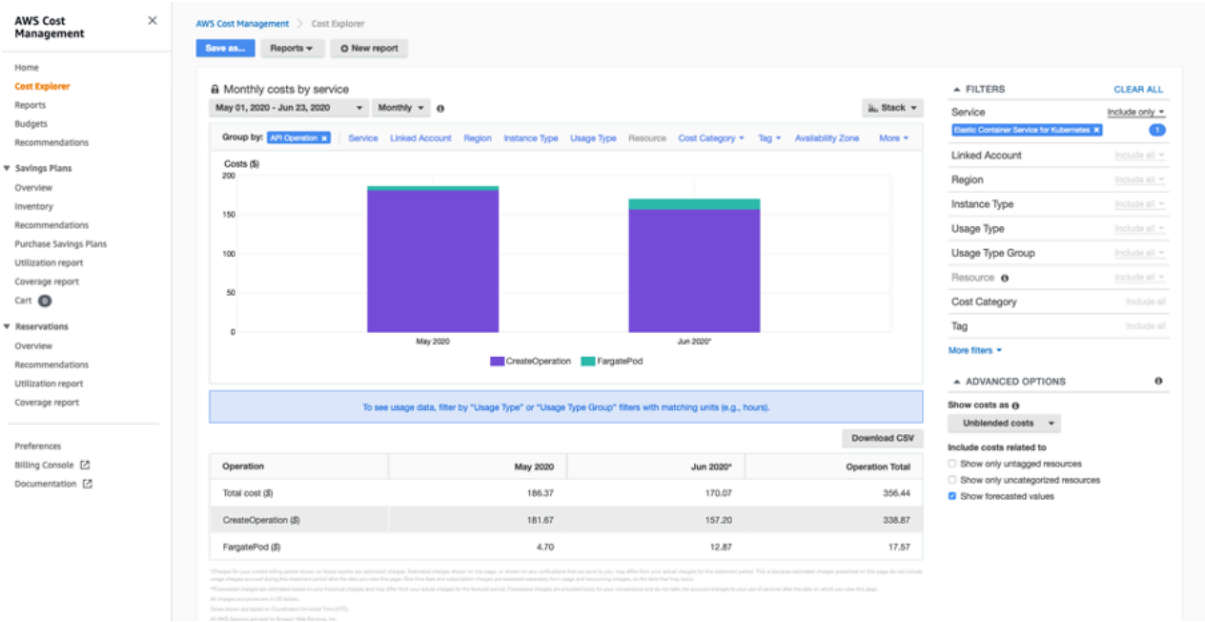
Recommandations

Utiliser Cost Explorer

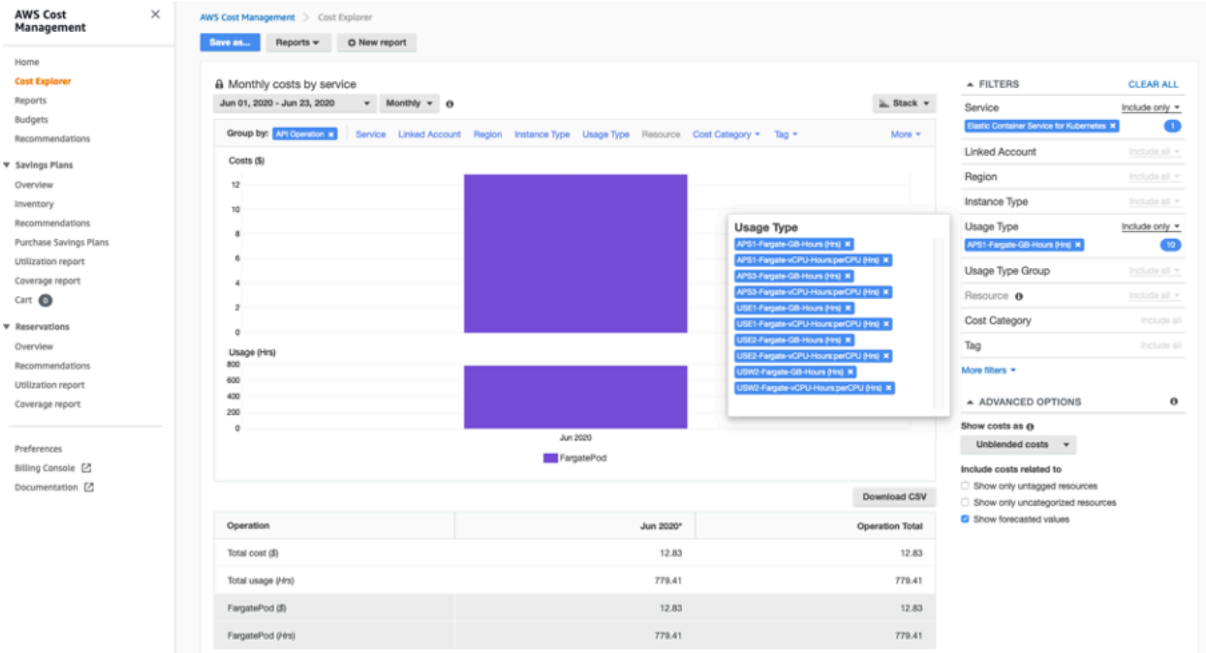
AWS Cost Explorer possède une easy-to-use interface qui vous permet de visualiser, de comprendre et de gérer vos coûts et votre utilisation d'AWS au fil du temps. Vous pouvez analyser les données de coût et d'utilisation à différents niveaux à l'aide des filtres disponibles dans Cost Explorer.

Coûts liés au plan de contrôle EKS et à EKS Fargate

À l'aide des filtres, nous pouvons vérifier les coûts engagés pour les coûts EKS sur le plan de contrôle et le module Fargate, comme indiqué dans le schéma ci-dessous :



À l'aide des filtres, nous pouvons demander les coûts agrégés engagés pour les Fargate Pods dans toutes les régions d'EKS, ce qui inclut à la fois les heures de processeur virtuel par processeur et les Go d'heures, comme le montre le schéma ci-dessous :



Marquage des ressources

Amazon EKS prend en charge l'ajout de balises AWS à vos clusters Amazon EKS. Cela permet de contrôler facilement l'accès à l'API EKS pour gérer vos clusters. Les balises ajoutées à un cluster EKS sont spécifiques à la ressource du cluster AWS EKS, elles ne se propagent pas aux autres ressources AWS utilisées par le cluster, telles que les EC2 instances ou les équilibreurs de charge. Aujourd'hui, le balisage des clusters est pris en charge pour tous les clusters EKS nouveaux et existants via l'API, la console et SDKs.

AWS Fargate est une technologie qui fournit une capacité de calcul adaptée à la demande pour les conteneurs. Avant de pouvoir planifier des pods sur Fargate dans votre cluster, vous devez définir au moins un profil Fargate qui indique quels pods doivent utiliser Fargate lors de leur lancement.

Ajouter et répertorier des balises dans un cluster EKS :

```
$ aws eks tag-resource --resource-arn arn:aws:eks:us-west-2:xxx:cluster/ekscluster1 --
tags team=devops,env=staging,bu=cio,costcenter=1234
$ aws eks list-tags-for-resource --resource-arn arn:aws:eks:us-west-2:xxx:cluster/
ekscluster1
{
    "tags": {
        "bu": "cio",
        "env": "staging",
        "costcenter": "1234",
        "team": "devops"
```

```
    }
}
```

Une fois que vous avez activé les balises de répartition des coûts dans AWS Cost Explorer, AWS utilise les balises de répartition des coûts pour organiser vos coûts de ressources dans votre rapport de répartition des coûts, afin de vous permettre de classer et de suivre plus facilement vos coûts AWS.

Les identifications n'ont pas de signification sémantique pour Amazon EKS et sont interprétées strictement comme des chaînes de caractères. Par exemple, vous pouvez définir un ensemble de balises pour vos clusters Amazon EKS afin de vous aider à suivre le propriétaire et le niveau de pile de chaque cluster.

Utiliser AWS Trusted Advisor

AWS Trusted Advisor propose un ensemble complet de vérifications et de recommandations relatives aux meilleures pratiques dans cinq catégories : optimisation des coûts, sécurité, tolérance aux pannes, performances et limites de service.

Pour optimiser les coûts, Trusted Advisor aide à éliminer les ressources inutilisées et inutilisées et recommande de prendre des engagements en matière de capacité réservée. Les principales mesures qui aideront Amazon EKS concerneront notamment les EC2 instances peu utilisées, les adresses IP élastiques non associées, les équilibreurs de charge inactifs et les volumes EBS sous-utilisés. La liste complète des contrôles est disponible sur https://aws.amazon.com/premiumsupport/technology/trusted-advisor/best -practice-checklist/.

The Trusted Advisor fournit également des recommandations sur les Savings Plans et les EC2 instances réservées pour les instances et Fargate, qui vous permet de vous engager à utiliser un montant d'utilisation constant en échange de tarifs réduits.

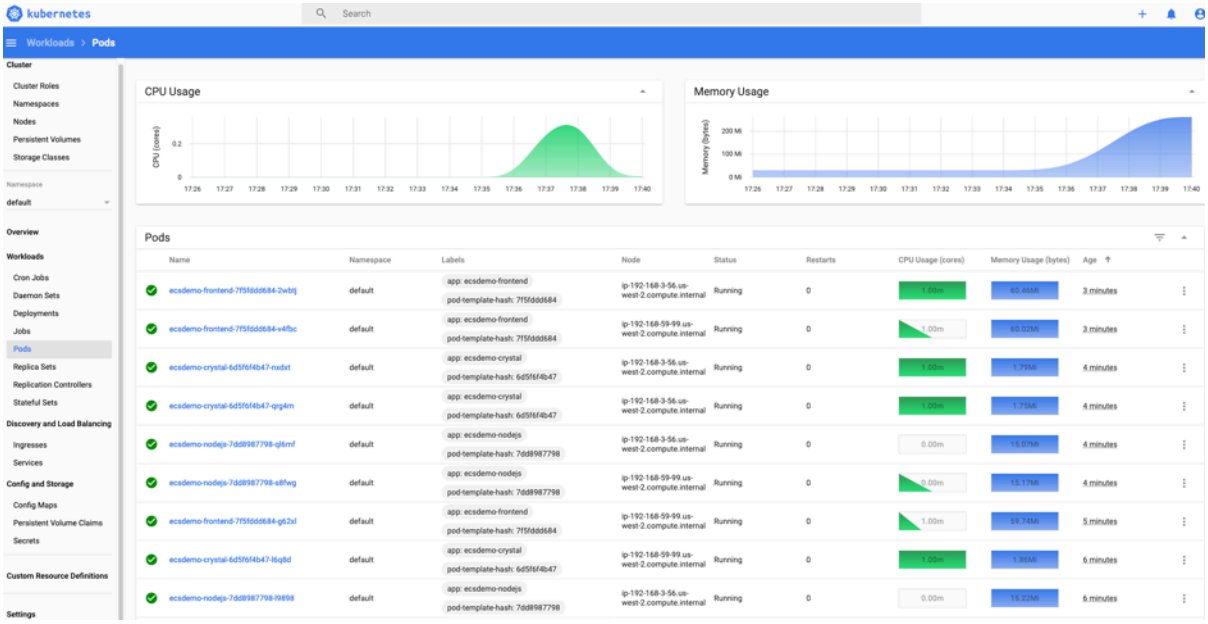
> **Note**
>
> Les recommandations de Trusted Advisor sont des recommandations génériques et ne sont pas spécifiques à EKS.

Utiliser le tableau de bord Kubernetes

Tableau de bord Kubernetes

Le tableau de bord Kubernetes est une interface utilisateur Web à usage général pour les clusters Kubernetes, qui fournit des informations sur le cluster Kubernetes, notamment l'utilisation des ressources au niveau du cluster, du nœud et du pod. Le déploiement du tableau de bord Kubernetes sur un cluster Amazon EKS est décrit dans la documentation Amazon EKS.

Le tableau de bord fournit des informations détaillées sur l'utilisation des ressources pour chaque nœud et chaque pod, ainsi que des métadonnées détaillées sur les pods, les services, les déploiements et les autres objets Kubernetes. Ces informations consolidées offrent une visibilité sur votre environnement Kubernetes.



commandes kubectl top et describe

Afficher les métriques d'utilisation des ressources avec les commandes kubectl top et kubectl describe. kubectl top affichera l'utilisation actuelle du processeur et de la mémoire pour les pods ou les nœuds de votre cluster, ou pour un pod ou un nœud spécifique. La commande kubectl describe fournira des informations plus détaillées sur un nœud ou un pod spécifique.

```
$ kubectl top pods
$ kubectl top nodes
$ kubectl top pod pod-name --namespace mynamespace --containers
```

À l'aide de la commande top, la sortie affiche la quantité totale de CPU (en cœurs) et de mémoire (en MiB) utilisée par le nœud, ainsi que les pourcentages de capacité allouable du nœud que ces chiffres représentent. Vous pouvez ensuite passer au niveau suivant, le niveau du conteneur dans les pods, en ajoutant un indicateur --containers.

```
$ kubectl describe node <node>
$ kubectl describe pod <pod>
```

kubectl describe renvoie le pourcentage de capacité totale disponible représenté par chaque demande ou limite de ressource.

kubectl top and describe, suivez l'utilisation et la disponibilité des ressources critiques telles que le processeur, la mémoire et le stockage sur les pods, nœuds et conteneurs Kubernetes. Cette prise de conscience aidera à comprendre l'utilisation des ressources et à contrôler les coûts.

Utilisez CloudWatch Container Insights

Utilisez CloudWatch Container Insights pour collecter, agréger et résumer les métriques et les journaux de vos applications conteneurisées et de vos microservices. Container Insights est disponible pour Amazon Elastic Kubernetes Service EC2 sur Amazon et pour les plateformes Kubernetes sur Amazon. EC2 Les métriques incluent l'utilisation des ressources telles que l'UC, la mémoire, le disque et le réseau.

L'installation des insights est décrite dans la documentation.

CloudWatch crée des métriques agrégées au niveau du cluster, du nœud, du pod, de la tâche et du service sous forme de CloudWatch métriques.

La requête suivante affiche une liste de nœuds, triée par utilisation moyenne du processeur par nœud

```
STATS avg(node_cpu_utilization) as avg_node_cpu_utilization by NodeName
| SORT avg_node_cpu_utilization DESC
```

Utilisation du processeur par nom de conteneur

```
stats pct(container_cpu_usage_total, 50) as CPUPercMedian by kubernetes.container_name
| filter Type="Container"
```

Utilisation du disque par nom de conteneur

```
stats floor(avg(container_filesystem_usage/1024)) as container_filesystem_usage_avg_kb
 by InstanceId, kubernetes.container_name, device
| filter Type="ContainerFS"
| sort container_filesystem_usage_avg_kb desc
```

D'autres exemples de requêtes sont fournis dans la documentation Container Insights.

Cette prise de conscience aidera à comprendre l'utilisation des ressources et à contrôler les coûts.

Utiliser Kubecost pour la sensibilisation et les conseils en matière de dépenses

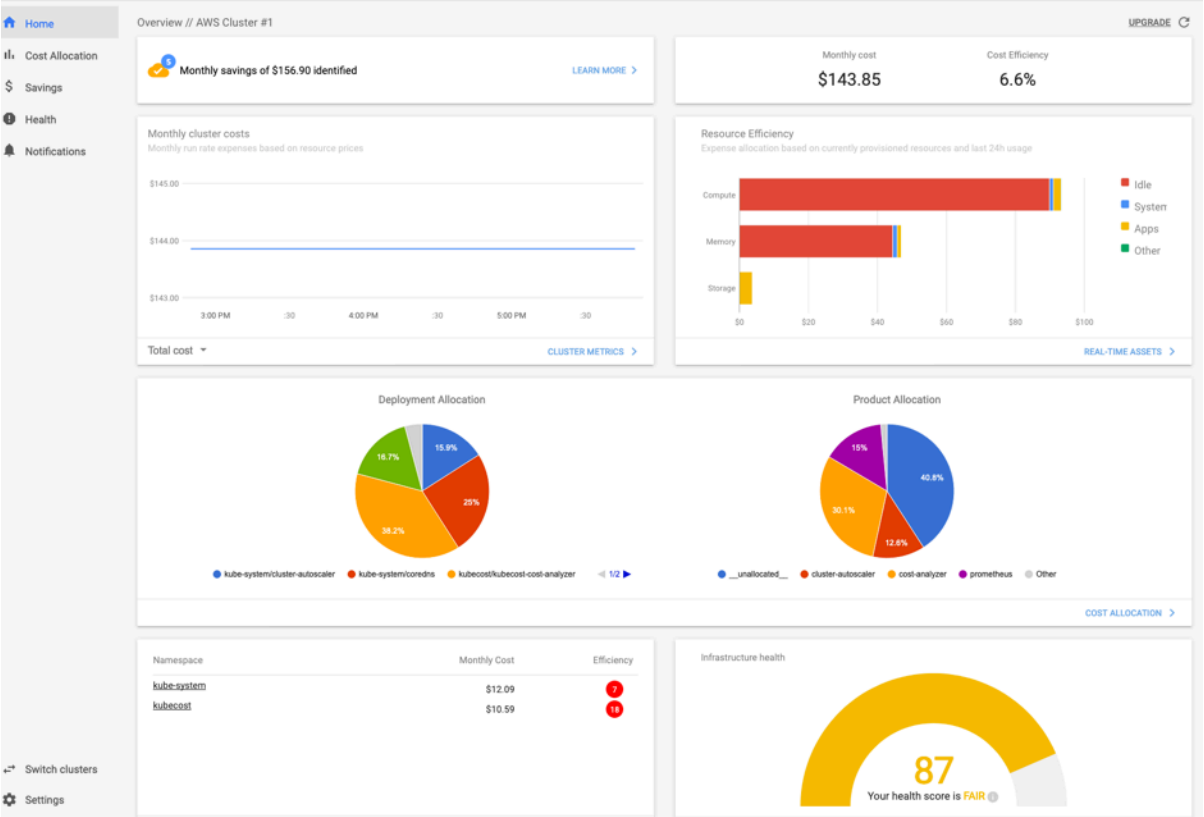
Des outils tiers tels que kubecost peuvent également être déployés sur Amazon EKS pour obtenir une visibilité sur le coût de fonctionnement de votre cluster Kubernetes. Consultez ce blog AWS pour le suivi des coûts à l'aide de Kubecost

Déploiement de kubecost à l'aide de Helm 3 :

```
$ curl -sSL https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 |
 bash
$ helm version --short
v3.2.1+gfe51cd1
$ helm repo add stable https://kubernetes-charts.storage.googleapis.com/
$ helm repo add stable https://kubernetes-charts.storage.googleapis.com/c^C
$ kubectl create namespace kubecost
namespace/kubecost created
$ helm repo add kubecost https://kubecost.github.io/cost-analyzer/
"kubecost" has been added to your repositories

$ helm install kubecost kubecost/cost-analyzer --namespace kubecost --set
 kubecostToken="aGRoZEBqc2pzLmNvbQ==xm343yadf98"
NAME: kubecost
LAST DEPLOYED: Mon May 18 08:49:05 2020
NAMESPACE: kubecost
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
---------------------------------------------------Kubecost has been successfully
 installed. When pods are Ready, you can enable port-forwarding with the following
 command:

    kubectl port-forward --namespace kubecost deployment/kubecost-cost-analyzer 9090

Next, navigate to http://localhost:9090 in a web browser.
$ kubectl port-forward --namespace kubecost deployment/kubecost-cost-analyzer 9090

NOTE: If you are using Cloud 9 or have a need to forward it to a different port like
 8080, issue the following command
```

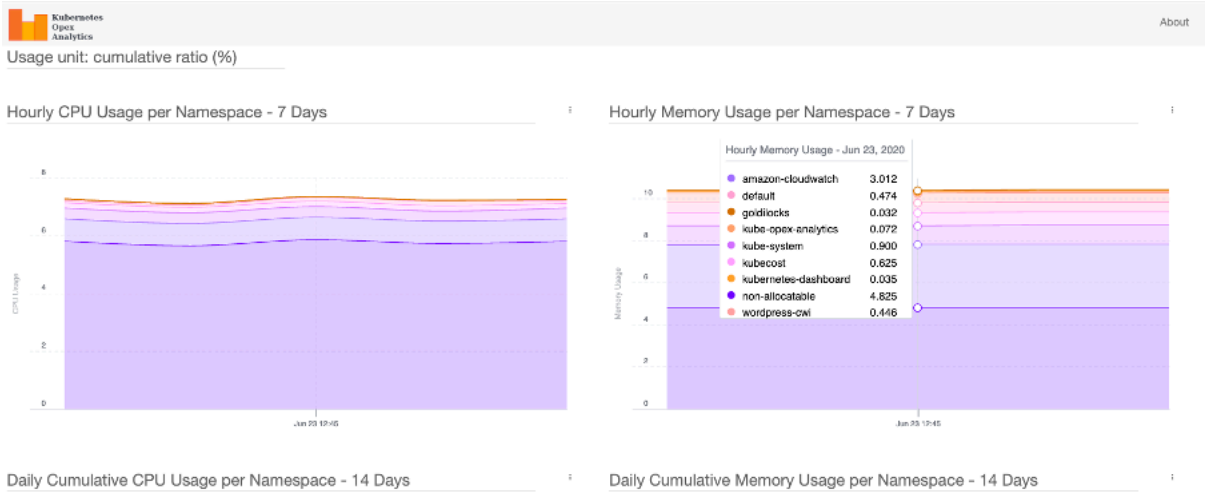
```
$ kubectl port-forward --namespace kubecost deployment/kubecost-cost-analyzer 8080:9090
```

Tableau de bord Kubecost -



Utiliser l'outil d'analyse de la répartition des coûts et de la planification des capacités de Kubernetes

Kubernetes Opex Analytics est un outil destiné à aider les entreprises à suivre les ressources consommées par leurs clusters Kubernetes afin d'éviter les surpaiements. Pour ce faire, il génère des rapports d'utilisation à court (7 jours), à moyen (14 jours) et à long terme (12 mois) fournissant des informations pertinentes sur la quantité de ressources dépensée par chaque projet au fil du temps.



Échelle de yoga

Yotascale aide à répartir avec précision les coûts de Kubernetes. La fonction d'allocation des coûts de Yotascale Kubernetes utilise les données de coûts réels, qui incluent les remises sur les instances réservées et les prix des instances au comptant, au lieu d'estimations génériques des taux du marché, pour déterminer l'empreinte financière totale de Kubernetes.

Plus de détails peuvent être trouvés sur leur site Web.

Conseiller Alcide

Alcide est un partenaire technologique avancé du réseau de partenaires AWS (APN). Alcide Advisor permet de s'assurer que la configuration de votre cluster, de vos nœuds et de vos pods Amazon EKS est réglée pour fonctionner conformément aux meilleures pratiques de sécurité et aux directives internes. Alcide Advisor est un service sans agent d'audit et de conformité Kubernetes conçu pour garantir un DevSecOps flux fluide et sécurisé en durcissant la phase de développement avant de passer à la production.

Vous trouverez plus de détails dans ce billet de blog.

Autres outils

Collecte des déchets Kubernetes

Le rôle du ramasse-miettes de Kubernetes est de supprimer certains objets qui avaient autrefois un propriétaire, mais qui n'en ont plus.

Nombre de Fargate

Fargatecount est un outil utile qui permet aux clients AWS de suivre, à l'aide d'une CloudWatch métrique personnalisée, le nombre total de pods EKS déployés sur Fargate dans une région spécifique d'un compte spécifique. Cela permet de suivre tous les pods Fargate exécutés sur un cluster EKS.

Popeye - Un désinfectant pour clusters Kubernetes

Popeye - Un désinfectant de cluster Kubernetes est un utilitaire qui analyse le cluster Kubernetes en direct et signale les problèmes potentiels liés aux ressources et aux configurations déployées. Il nettoie votre cluster en fonction de ce qui est déployé et non de ce qui se trouve sur le disque. En scannant votre cluster, il détecte les erreurs de configuration et vous aide à vous assurer que les meilleures pratiques sont en place

Ressources

Consultez les ressources suivantes pour en savoir plus sur les meilleures pratiques en matière d'optimisation des coûts.

Documentation et blogs

- Amazon EKS prend en charge le balisage

Outils

- Qu'est-ce qu'AWS Billing and Cost Management ?
- Informations sur les CloudWatch conteneurs Amazon
- Comment suivre les coûts dans les clusters Amazon EKS à locataires multiples à l'aide de Kubecost
- Kubecost
- Kube Opsview
- Analyses Opex de Kubernetes

Calcul et mise à l'échelle automatique

En tant que développeur, vous évaluerez les besoins en ressources de votre application, par exemple en termes de processeur et de mémoire, mais si vous ne les ajustez pas continuellement,

elles risquent de devenir obsolètes, ce qui peut augmenter vos coûts et détériorer les performances et la fiabilité. Il est plus important d'ajuster en permanence les besoins en ressources d'une application que de les satisfaire correctement du premier coup.

Les meilleures pratiques mentionnées ci-dessous vous aideront à créer et à exploiter des charges de travail sensibles aux coûts qui permettent d'obtenir des résultats commerciaux tout en minimisant les coûts et en permettant à votre organisation de maximiser son retour sur investissement. Voici un ordre d'importance élevé pour optimiser les coûts de calcul de votre cluster :

1. Dimensionnez correctement les charges de travail
2. Réduction de la capacité inutilisée
3. Optimisez les types de capacité de calcul (par exemple Spot) et les accélérateurs (par exemple GPUs)

Dimensionnez correctement vos charges de travail

Dans la plupart des clusters EKS, l'essentiel des coûts provient des EC2 instances utilisées pour exécuter vos charges de travail conteneurisées. Vous ne serez pas en mesure de dimensionner correctement vos ressources informatiques si vous ne comprenez pas vos exigences en matière de charges de travail. C'est pourquoi il est essentiel que vous utilisiez les demandes et les limites appropriées et que vous modifiiez ces paramètres si nécessaire. En outre, les dépendances, telles que la taille de l'instance et la sélection du stockage, peuvent affecter les performances de la charge de travail, ce qui peut avoir diverses conséquences imprévues sur les coûts et la fiabilité.

Les demandes doivent correspondre à l'utilisation réelle. Si les demandes d'un conteneur sont trop élevées, la capacité sera inutilisée, ce qui représente un facteur important dans le coût total du cluster. Chaque conteneur d'un pod, par exemple l'application et les sidecars, doit avoir ses propres demandes et limites définies pour garantir que les limites globales des pods sont aussi précises que possible.

Utilisez des outils tels que Goldilocks, KRR et Kubecost pour estimer les demandes de ressources et les limites pour vos conteneurs. En fonction de la nature des applications, des performance/cost exigences et de leur complexité, vous devez déterminer quels indicateurs sont les meilleurs pour évoluer, à quel moment les performances de votre application se dégradent (point de saturation) et comment ajuster les demandes et les limites en conséquence. Veuillez vous référer à la section Taille correcte de l'application pour obtenir de plus amples informations à ce sujet.

Nous vous recommandons d'utiliser le Horizontal Pod Autoscaler (HPA) pour contrôler le nombre de répliques de votre application à exécuter, le Vertical Pod Autoscaler (VPA) pour ajuster le nombre de demandes et les limites dont votre application a besoin par réplique, et un autoscaler de nœuds tel que Karpenter ou Cluster Autoscaler pour ajuster en permanence le nombre total de nœuds de votre cluster. Les techniques d'optimisation des coûts à l'aide de Karpenter et Cluster Autoscaler sont décrites dans une section ultérieure de ce document.

Le Vertical Pod Autoscaler peut ajuster les demandes et les limites attribuées aux conteneurs afin que les charges de travail s'exécutent de manière optimale. Vous devez exécuter le VPA en mode audit afin qu'il n'apporte pas automatiquement de modifications et ne redémarre pas vos pods. Il proposera des modifications en fonction des indicateurs observés. Toute modification affectant les charges de travail de production doit d'abord être examinée et testée dans un environnement hors production, car elle peut avoir un impact sur la fiabilité et les performances de votre application.

Réduire la consommation

Le meilleur moyen d'économiser de l'argent est de fournir moins de ressources. L'un des moyens d'y parvenir consiste à ajuster les charges de travail en fonction de leurs exigences actuelles. Vous devez commencer tout effort d'optimisation des coûts en vous assurant que vos charges de travail répondent à leurs exigences et évoluent de manière dynamique. Cela nécessitera d'obtenir des métriques à partir de vos applications et de définir des configurations telles que `PodDisruptionBudgets`Pod Readiness Gates pour garantir que votre application peut évoluer dynamiquement à la hausse et à la baisse en toute sécurité. Il est important de noter qu'une restriction PodDisruptionBudgets peut empêcher Cluster Autoscaler et Karpenter de réduire les nœuds, étant donné que Cluster Autoscaler et Karpenter respectent tous les deux. PodDisruptionBudgets La valeur « minAvailable » dans le PodDisruptionBudget doit toujours être inférieure au nombre de pods dans le déploiement et vous devez garder une bonne mémoire tampon entre les deux, par exemple dans un déploiement de 6 pods où vous souhaitez qu'un minimum de 4 pods fonctionnent en permanence, réglez le « minAvailable » sur 4. PodDisruptionBidget Cela permettra à Cluster Autoscaler et Karpenter de vider et d'expulser en toute sécurité les pods des nœuds sous-utilisés lors d'un événement de réduction des nœuds. Reportez-vous au document FAQ de Cluster Autoscaler.

L'Horizontal Pod Autoscaler est un autoscaler de charge de travail flexible qui peut ajuster le nombre de répliques nécessaires pour répondre aux exigences de performance et de fiabilité de votre application. Il dispose d'un modèle flexible permettant de définir quand augmenter ou diminuer en fonction de diverses mesures telles que le processeur, la mémoire ou de mesures personnalisées, telles que la profondeur de la file d'attente, le nombre de connexions à un pod, etc.

Le serveur de métriques Kubernetes permet une mise à l'échelle en réponse à des indicateurs intégrés tels que l'utilisation du processeur et de la mémoire, mais si vous souhaitez effectuer une mise à l'échelle en fonction d'autres indicateurs, tels que la profondeur des files d'attente Amazon CloudWatch ou SQS, vous devez envisager des projets de mise à l'échelle automatique pilotés par des événements tels que KEDA. Reportez-vous à ce billet de blog pour savoir comment utiliser KEDA avec CloudWatch les métriques. Si vous n'êtes pas certain des indicateurs à surveiller et sur lesquels vous baser, consultez les meilleures pratiques en matière de surveillance des indicateurs importants.

La réduction de la consommation de charge de travail crée une capacité excédentaire dans un cluster et, avec une configuration d'autoscaling appropriée, vous pouvez automatiquement réduire le nombre de nœuds et réduire vos dépenses totales. Nous vous recommandons de ne pas essayer d'optimiser la capacité de calcul manuellement. Le planificateur Kubernetes et les autoscalers de nœuds ont été conçus pour gérer ce processus à votre place.

Réduction de la capacité inutilisée

Une fois que vous avez déterminé la taille appropriée pour les applications, réduisant ainsi le nombre de demandes excédentaires, vous pouvez commencer à réduire la capacité de calcul allouée. Vous devriez être en mesure de le faire de manière dynamique si vous avez pris le temps de dimensionner correctement vos charges de travail dans les sections ci-dessus. Deux autoscalers de nœuds principaux sont utilisés avec Kubernetes dans AWS.

Karpenter et Cluster Autoscaler

Karpenter et Kubernetes Cluster Autoscaler augmenteront le nombre de nœuds de votre cluster en fonction de la création ou de la suppression de pods et de l'évolution des exigences de calcul. L'objectif principal des deux est le même, mais Karpenter adopte une approche différente pour le provisionnement et le déprovisionnement de la gestion des nœuds, ce qui peut contribuer à réduire les coûts et à optimiser l'utilisation à l'échelle du cluster.

À mesure que la taille des clusters augmente et que la variété des charges de travail augmente, il devient de plus en plus difficile de préconfigurer les groupes de nœuds et les instances. Tout comme pour les demandes de charge de travail, il est important de définir une base de référence initiale et de l'ajuster en permanence selon les besoins.

Si vous utilisez Cluster Autoscaler, il respectera les valeurs « minimum » et « maximum » de chaque groupe Auto Scaling (ASG) et ajustera uniquement la valeur « souhaitée ». Il est important de faire attention lorsque vous définissez ces valeurs pour l'ASG sous-jacent, car Cluster Autoscaler ne sera pas en mesure de réduire un ASG au-delà de son nombre « minimum ». Définissez le nombre

« souhaité » comme le nombre de nœuds dont vous avez besoin pendant les heures normales de bureau et le nombre « minimum » comme le nombre de nœuds dont vous avez besoin en dehors des heures de bureau. Reportez-vous au document FAQ de Cluster Autoscaler.

Expandeur de priorité Cluster Autoscaler

Le Kubernetes Cluster Autoscaler fonctionne en redimensionnant des groupes de nœuds (appelés groupes de nœuds) vers le haut ou vers le bas au fur et à mesure que les applications augmentent ou diminuent. Si vous ne dimensionnez pas les charges de travail de manière dynamique, le Cluster Autoscaler ne vous aidera pas à économiser de l'argent. Le Cluster Autoscaler nécessite qu'un administrateur du cluster crée des groupes de nœuds à l'avance pour que les charges de travail soient consommées. Les groupes de nœuds doivent être configurés pour utiliser des instances ayant le même « profil », c'est-à-dire à peu près la même quantité de processeur et de mémoire.

Vous pouvez avoir plusieurs groupes de nœuds et le Cluster Autoscaler peut être configuré pour définir des niveaux de dimensionnement prioritaires. Chaque groupe de nœuds peut contenir des nœuds de tailles différentes. Les groupes de nœuds peuvent avoir différents types de capacité et l'extenseur de priorité peut d'abord être utilisé pour dimensionner les groupes les moins coûteux.

Vous trouverez ci-dessous un exemple d'extrait de configuration de cluster qui utilise un `ConfigMap` pour prioriser la capacité réservée avant d'utiliser des instances à la demande. Vous pouvez utiliser la même technique pour prioriser les instances Graviton ou Spot par rapport aux autres types.

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-cluster
managedNodeGroups:
  - name: managed-ondemand
    minSize: 1
    maxSize: 7
    instanceType: m5.xlarge
  - name: managed-reserved
    minSize: 2
    maxSize: 10
    instanceType: c5.2xlarge
```

```
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: cluster-autoscaler-priority-expander
  namespace: kube-system
data:
  priorities: |-
    10:
      - .*ondemand.*
    50:
      - .*reserved.*
```

L'utilisation de groupes de nœuds peut aider les ressources de calcul sous-jacentes à faire ce que l'on attend par défaut, par exemple en répartissant les nœuds AZs, mais toutes les charges de travail n'ont pas les mêmes exigences ou attentes. Il est donc préférable de laisser les applications déclarer leurs besoins de manière explicite. Pour plus d'informations sur Cluster Autoscaler, consultez la section des meilleures pratiques.

Déplanificateur

Le Cluster Autoscaler peut ajouter et supprimer de la capacité des nœuds d'un cluster en fonction de la nécessité de planifier de nouveaux pods ou de la sous-utilisation de nœuds. Il ne prend pas en compte de manière globale le placement des pods une fois qu'ils ont été planifiés sur un nœud. Si vous utilisez le Cluster Autoscaler, vous devez également consulter le déplanificateur Kubernetes pour éviter de gaspiller de la capacité dans votre cluster.

Si vous avez 10 nœuds dans un cluster et que chaque nœud est utilisé à 60 %, vous n'utilisez pas 40 % de la capacité allouée dans le cluster. Avec le Cluster Autoscaler, vous pouvez définir le seuil d'utilisation par nœud à 60 %, mais cela n'essaiera de réduire qu'un seul nœud lorsque l'utilisation est tombée en dessous de 60 %.

Avec le déplanificateur, il peut examiner la capacité et l'utilisation du cluster une fois que les pods ont été planifiés ou que des nœuds ont été ajoutés au cluster. Il tente de maintenir la capacité totale du cluster au-dessus d'un seuil spécifié. Il peut également supprimer des pods en fonction de l'altération des nœuds ou de nouveaux nœuds qui rejoignent le cluster pour s'assurer que les pods fonctionnent dans leur environnement informatique optimal. Notez que le déplanificateur ne planifie pas le remplacement des pods expulsés mais s'appuie sur le planificateur par défaut pour cela.

Consolidation de Karpenter

Karpenter adopte une approche « sans groupe » pour la gestion des nœuds. Cette approche est plus flexible pour les différents types de charge de travail et nécessite moins de configuration initiale

pour les administrateurs de clusters. Au lieu de prédéfinir des groupes et de dimensionner chaque groupe en fonction des besoins des charges de travail, Karpenter utilise des fournisseurs et des modèles de nœuds pour définir de manière générale le type d' EC2 instances pouvant être créées et les paramètres relatifs aux instances au fur et à mesure de leur création.

Le bin packing est une pratique qui consiste à utiliser une plus grande partie des ressources de l'instance en répartissant davantage de charges de travail sur un nombre réduit d'instances de taille optimale. Bien que cela contribue à réduire vos coûts de calcul en provisionnant uniquement les ressources utilisées par vos charges de travail, cela comporte un compromis. Le démarrage de nouvelles charges de travail peut prendre plus de temps car de la capacité doit être ajoutée au cluster, en particulier lors d'événements de grande envergure. Tenez compte de l'équilibre entre l'optimisation des coûts, les performances et la disponibilité lors de la configuration de l'emballage en bacs.

Karpenter peut effectuer une surveillance continue et un binpack afin d'améliorer l'utilisation des ressources des instances et de réduire vos coûts de calcul. Karpenter peut également sélectionner un nœud de travail plus rentable pour votre charge de travail. Cela peut être réalisé en activant l'indicateur « consolidation » sur true dans le fournisseur (exemple d'extrait de code ci-dessous). L'exemple ci-dessous montre un exemple de fournisseur qui permet la consolidation. Au moment de la rédaction de ce guide, Karpenter ne remplacera pas une instance Spot en cours d'exécution par une instance Spot moins chère. Pour plus de détails sur la consolidation de Karpenter, consultez ce blog.

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
  name: enable-binpacking
spec:
  consolidation:
    enabled: true
```

Pour les charges de travail susceptibles de ne pas être interruptibles, par exemple les tâches par lots de longue durée sans point de contrôle, pensez à annoter les pods avec l'annotation. `do-not-evict` En refusant l'expulsion des modules, vous dites à Karpenter qu'il ne doit pas supprimer volontairement les nœuds contenant ce module. Toutefois, si un `do-not-evict` pod est ajouté à un nœud alors que celui-ci se vide, les pods restants seront tout de même expulsés, mais ce pod bloquera la terminaison jusqu'à ce qu'il soit retiré. Dans les deux cas, le nœud sera bouclé pour empêcher que des travaux supplémentaires ne soient planifiés sur le nœud. Vous trouverez ci-dessous un exemple montrant comment définir l'annotation :

```
apiVersion: v1
kind: Pod
metadata:
  name: label-demo
  labels:
    environment: production
  annotations: +
    "karpenter.sh/do-not-evict": "true"
spec:
  containers:

* name: nginx
image: nginx
ports:
 ** containerPort: 80
```

Supprimez les nœuds sous-utilisés en ajustant les paramètres du Cluster Autoscaler

L'utilisation des nœuds est définie comme la somme des ressources demandées divisée par la capacité. Par défaut, `scale-down-utilization-threshold` il est défini sur 50 %. Ce paramètre peut être utilisé conjointement avec et`scale-down-unneeded-time`, qui détermine la durée pendant laquelle un nœud doit être inutile avant de pouvoir être réduit. La valeur par défaut est de 10 minutes. Les pods toujours en cours d'exécution sur un nœud réduit seront planifiés sur d'autres nœuds par kube-scheduler. L'ajustement de ces paramètres peut aider à supprimer les nœuds sous-utilisés, mais il est important de tester d'abord ces valeurs afin de ne pas forcer le cluster à se réduire prématurément.

Vous pouvez empêcher la réduction de la taille en veillant à ce que les pods dont l'expulsion coûte cher soient protégés par une étiquette reconnue par le Cluster Autoscaler. Pour ce faire, assurez-vous que les pods dont l'expulsion coûte cher comportent l'annotation`cluster-autoscaler.kubernetes.io/safe-to-evict=false`. Voici un exemple de fichier yaml pour définir l'annotation :

```
apiVersion: v1
kind: Pod
metadata:
  name: label-demo
  labels:
    environment: production
  annotations: +
    "cluster-autoscaler.kubernetes.io/safe-to-evict": "false"
```

```
spec:
  containers:

* name: nginx
image: nginx
ports:
 ** containerPort: 80
```

Étiquetez les nœuds avec Cluster Autoscaler et Karpenter

Les balises de ressources AWS sont utilisées pour organiser vos ressources et pour suivre vos coûts AWS de manière détaillée. Ils ne sont pas directement corrélés aux étiquettes Kubernetes pour le suivi des coûts. Il est recommandé de commencer par l'étiquetage des ressources Kubernetes et d'utiliser des outils tels que Kubecost pour obtenir des rapports sur les coûts d'infrastructure basés sur les étiquettes Kubernetes apposées sur les pods, les espaces de noms, etc.

Les nœuds de travail doivent disposer de balises pour afficher les informations de facturation dans AWS Cost Explorer. Avec Cluster Autoscaler, balisez vos nœuds de travail au sein d'un groupe de nœuds gérés à l'aide d'un modèle de lancement. Pour les groupes de nœuds autogérés, balisez vos instances à l'aide du groupe de dimensionnement EC2 automatique. Pour les instances mises en service par Karpenter, balisez-les à l'aide de spec.tags dans le modèle de nœud.

Clusters à locataires multiples

Lorsque vous travaillez sur des clusters partagés par différentes équipes, vous risquez de ne pas avoir de visibilité sur les autres charges de travail exécutées sur le même nœud. Bien que les demandes de ressources puissent aider à isoler certains problèmes liés aux « voisins bruyants », tels que le partage du processeur, elles peuvent ne pas isoler toutes les limites des ressources, telles que I/O l'épuisement du disque. Toutes les ressources consommables d'une charge de travail ne peuvent pas être isolées ou limitées. Les charges de travail qui consomment des ressources partagées à des taux plus élevés que les autres charges de travail doivent être isolées en fonction des entorses et des tolérances des nœuds. Une autre technique avancée pour une telle charge de travail est l'épinglage du processeur, qui garantit un processeur exclusif au lieu d'un processeur partagé pour le conteneur.

L'isolation des charges de travail au niveau d'un nœud peut s'avérer plus coûteuse, mais il est possible de planifier des BestEfforttâches ou de réaliser des économies supplémentaires en utilisant des instances réservées, des processeurs Graviton ou Spot.

Les clusters partagés peuvent également être soumis à des contraintes de ressources au niveau du cluster, telles que l'épuisement des adresses IP, les limites de service Kubernetes ou les demandes

de dimensionnement des API. Vous devriez consulter le guide des meilleures pratiques d'évolutivité pour vous assurer que vos clusters respectent ces limites.

Vous pouvez isoler les ressources au niveau d'un espace de noms ou d'un approvisionneur Karpenter. Les quotas de ressources permettent de définir des limites quant au nombre de ressources que les charges de travail peuvent consommer dans un espace de noms. Cela peut être un bon garde-fou initial, mais il doit être évalué en permanence pour s'assurer qu'il n'empêche pas artificiellement l'augmentation des charges de travail.

Les fournisseurs Karpenter peuvent définir des limites sur certaines des ressources consommables d'un cluster (par exemple, processeur, GPU), mais vous devrez configurer les applications clientes pour utiliser le fournisseur approprié. Cela peut empêcher un seul fournisseur de créer trop de nœuds dans un cluster, mais il convient de l'évaluer en permanence pour s'assurer que la limite n'est pas trop basse et, par conséquent, empêcher les charges de travail de s'étendre.

Mise à l'échelle automatique planifiée

Il se peut que vous deviez réduire la taille de vos clusters le week-end et en dehors des heures de bureau. Cela est particulièrement pertinent pour les clusters de test et non liés à la production pour lesquels vous souhaitez réduire à zéro lorsqu'ils ne sont pas utilisés. Des solutions telles que cluster-turndown peuvent réduire le nombre de répliques à zéro en fonction d'un calendrier cron. Vous pouvez également y parvenir avec Karpenter, comme indiqué dans le blog AWS suivant.

Optimisation des types de capacité de calcul

Après avoir optimisé la capacité de calcul totale de votre cluster et utilisé le bin packing, vous devez examiner le type de calcul que vous avez fourni dans vos clusters et le mode de paiement de ces ressources. AWS propose des plans d'économies de calcul qui peuvent réduire le coût de votre calcul. Nous les classerons selon les types de capacité suivants :

- Spot
- Savings Plans
- À la demande
- Fargate

Chaque type de capacité comporte des compromis différents en termes de frais de gestion, de disponibilité et d'engagements à long terme, et vous devrez choisir celui qui convient le mieux à votre environnement. Aucun environnement ne doit reposer sur un seul type de capacité et vous pouvez

combiner plusieurs types d'exécution dans un seul cluster afin d'optimiser les exigences et les coûts de charge de travail spécifiques.

Spot

Le type de capacité ponctuelle approvisionne EC2 les instances à partir de la capacité inutilisée d'une zone de disponibilité. Spot propose les remises les plus importantes (jusqu'à 90 %), mais ces instances peuvent être interrompues lorsqu'elles sont nécessaires ailleurs. En outre, il n'est pas toujours possible de fournir de nouvelles instances Spot et les instances Spot existantes peuvent être récupérées moyennant un préavis d'interruption de 2 minutes. Si le processus de démarrage ou d'arrêt de votre application est long, les instances Spot ne sont peut-être pas la meilleure option.

Le calcul ponctuel doit utiliser une grande variété de types d'instances afin de réduire le risque de ne pas disposer de capacité ponctuelle disponible. Les interruptions d'instance doivent être gérées pour arrêter les nœuds en toute sécurité. Les nœuds approvisionnés avec Karpenter ou faisant partie d'un groupe de nœuds gérés prennent automatiquement en charge les notifications d'interruption d'instance. Si vous utilisez des nœuds autogérés, vous devrez exécuter le gestionnaire de terminaison de nœuds séparément pour arrêter correctement les instances ponctuelles.

Il est possible d'équilibrer les instances ponctuelles et à la demande dans un seul cluster. Avec Karpenter, vous pouvez créer des approvisionneurs pondérés afin de trouver un équilibre entre les différents types de capacité. Avec Cluster Autoscaler, vous pouvez créer des groupes de nœuds mixtes avec des instances ponctuelles, à la demande ou réservées.

Voici un exemple d'utilisation de Karpenter pour prioriser les instances Spot par rapport aux instances à la demande. Lorsque vous créez un fournisseur, vous pouvez spécifier Spot, On-Demand ou les deux (comme indiqué ci-dessous). Lorsque vous spécifiez les deux, et si le pod ne précise pas explicitement s'il doit utiliser Spot ou On-Demand, Karpenter donne la priorité à Spot lorsqu'il approvisionne un nœud avec une stratégie d'allocation. price-capacity-optimization

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
  name: spot-prioritized
spec:
  requirements:
    - key: "karpenter.sh/capacity-type"
      operator: In
        values: ["spot", "on-demand"]
```

Savings Plans, instances réservées et AWS EDP

Vous pouvez réduire vos dépenses informatiques en utilisant un plan d'économies de calcul. Les plans d'épargne offrent des prix réduits pour un engagement d'un ou trois ans d'utilisation du calcul. L'utilisation peut s'appliquer aux EC2 instances d'un cluster EKS, mais s'applique également à toute utilisation de calcul telle que Lambda et Fargate. Grâce aux plans d'épargne, vous pouvez réduire les coûts tout en continuant à choisir n'importe quel type d' EC2 instance pendant votre période d'engagement.

Un plan d'économies de calcul peut réduire vos EC2 coûts jusqu'à 66 % sans nécessiter d'engagement quant aux types d'instances, aux familles ou aux régions que vous souhaitez utiliser. Les économies sont automatiquement appliquées aux instances lorsque vous les utilisez.

EC2 Instance Savings Plans permet de réaliser jusqu'à 72 % d'économies sur le calcul avec un engagement d'utilisation dans une région et une EC2 famille spécifiques, par exemple les instances de la famille C. Vous pouvez transférer l'utilisation vers n'importe quelle zone de la région, utiliser n'importe quelle génération de la famille d'instances, par exemple c5 ou c6, et utiliser n'importe quelle taille d'instance au sein de la famille. La réduction sera automatiquement appliquée à toute instance de votre compte correspondant aux critères du plan d'épargne.

Les instances réservées sont similaires à EC2 Instance Savings Plan, mais elles garantissent également la capacité dans une zone de disponibilité ou une région et réduisent les coûts (jusqu'à 72 %) par rapport aux instances à la demande. Une fois que vous aurez calculé la capacité réservée dont vous aurez besoin, vous pourrez sélectionner la durée pour laquelle vous souhaitez les réserver (1 an ou 3 ans). Les remises seront automatiquement appliquées lorsque vous exécuterez ces EC2 instances dans votre compte.

Les clients ont également la possibilité de souscrire un contrat d'entreprise avec AWS. Les contrats d’entreprise offrent aux clients la possibilité de personnaliser les accords qui répondent le mieux à leurs besoins. Les clients peuvent bénéficier de remises sur les tarifs basés sur le programme AWS EDP (Enterprise Discount Program). Pour plus d'informations sur les contrats d'entreprise, veuillez contacter votre représentant commercial AWS.

À la demande

Les EC2 instances à la demande présentent l'avantage d'une disponibilité sans interruption, par rapport aux instances ponctuelles, et de l'absence d'engagement à long terme, par rapport aux plans d'épargne. Si vous souhaitez réduire les coûts d'un cluster, vous devez réduire votre utilisation des EC2 instances à la demande.

Après avoir optimisé vos exigences en matière de charge de travail, vous devriez être en mesure de calculer une capacité minimale et maximale pour vos clusters. Ce chiffre peut changer au fil du temps, mais il diminue rarement. Envisagez d'utiliser un Savings Plan pour tout ce qui est inférieur au minimum, et optez pour une capacité qui n'affectera pas la disponibilité de votre application. Tout autre élément qui ne peut pas être utilisé en permanence ou qui est requis pour des raisons de disponibilité peut être utilisé à la demande.

Comme indiqué dans cette section, le meilleur moyen de réduire votre consommation est de consommer moins de ressources et d'utiliser au maximum les ressources que vous fournissez. Avec le Cluster Autoscaler, vous pouvez supprimer les nœuds sous-utilisés à l'aide du paramètre. `scale-down-utilization-threshold` Avec Karpenter, il est recommandé d'activer la consolidation.

Pour identifier manuellement les types d' EC2 instances qui peuvent être utilisés avec vos charges de travail, vous devez utiliser ec2-instance-selector, qui peut afficher les instances disponibles dans chaque région ainsi que les instances compatibles avec EKS. Exemple d'utilisation pour des instances dotées d'une architecture de processus x86, de 4 Go de mémoire, de 2 V CPUs et disponibles dans la région us-east-1.

```
ec2-instance-selector --memory 4 --vcpus 2 --cpu-architecture x86_64 \
  -r us-east-1 --service eks
c5.large
c5a.large
c5ad.large
c5d.large
c6a.large
c6i.large
t2.medium
t3.medium
t3a.medium
```

Pour les environnements non liés à la production, vous pouvez automatiquement réduire la taille des clusters pendant les heures non utilisées, telles que la nuit et le week-end. Le cluster-turndown du projet kubecost est un exemple de contrôleur capable de réduire automatiquement votre cluster en fonction d'un calendrier défini.

Ordinateur Fargate

Le calcul Fargate est une option de calcul entièrement gérée pour les clusters EKS. Il permet d'isoler les pods en planifiant un pod par nœud dans un cluster Kubernetes. Il vous permet de dimensionner

vos nœuds de calcul en fonction des besoins en CPU et en RAM de votre charge de travail afin de contrôler étroitement l'utilisation de la charge de travail dans un cluster.

Fargate peut faire évoluer des charges de travail allant de 2,5 vCPU avec 0,5 Go de mémoire à 16 vCPU avec 120 Go de mémoire. Le nombre de variations de taille des pods disponibles est limité et vous devez comprendre comment votre charge de travail s'adapte le mieux à une configuration Fargate. Par exemple, si votre charge de travail nécessite 1 vCPU avec 0,5 Go de mémoire, le plus petit pod Fargate sera un vCPU avec 2 Go de mémoire.

Bien que Fargate présente de nombreux avantages, tels que l' EC2 absence de gestion d'instance ou de système d'exploitation, il peut nécessiter une capacité de calcul supérieure à celle des instances EC2 traditionnelles, car chaque pod déployé est isolé en tant que nœud distinct dans le cluster. Cela nécessite davantage de duplication pour des éléments tels que le Kubelet, les agents de journalisation et tout ce DaemonSets que vous déployez généralement sur un nœud. DaemonSets ne sont pas pris en charge dans Fargate et ils devront être convertis en pods « `sidecars » et exécutés en même temps que l'application.

Fargate ne peut pas tirer parti du bin packing ou du surprovisionnement du processeur, car la limite de la charge de travail est un nœud qui n'est pas extensible ou partageable entre les charges de travail. Fargate EC2 vous permettra de gagner du temps dans la gestion des instances, ce qui a un coût en soi, mais les coûts liés au processeur et à la mémoire peuvent être plus élevés que ceux EC2 des autres types de capacité. Les modules Fargate peuvent tirer parti d'un plan d'économies de calcul pour réduire les coûts à la demande.

Optimisation de l'utilisation du calcul

Une autre façon d'économiser de l'argent sur votre infrastructure informatique consiste à utiliser un calcul plus efficace pour la charge de travail. Cela peut être dû à des solutions informatiques à usage général plus performantes, comme les processeurs Graviton, qui sont jusqu'à 20 % moins chers et 60 % plus économes en énergie qu'un processeur x86, ou à des accélérateurs spécifiques à la charge de travail tels que et. GPUs FPGAs Vous devrez créer des conteneurs pouvant fonctionner sur l'architecture ARM et configurer des nœuds dotés des accélérateurs adaptés à vos charges de travail.

EKS est capable d'exécuter des clusters à architecture mixte (par exemple amd64 et arm64) et si vos conteneurs sont compilés pour plusieurs architectures, vous pouvez tirer parti des processeurs Graviton avec Karpenter en autorisant les deux architectures dans votre fournisseur. Pour maintenir des performances constantes, il est toutefois recommandé de conserver chaque charge de travail

sur une architecture informatique unique et de n'utiliser une architecture différente que si aucune capacité supplémentaire n'est disponible.

Les fournisseurs peuvent être configurés avec plusieurs architectures et les charges de travail peuvent également demander des architectures spécifiques dans leurs spécifications de charge de travail.

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
  name: default
spec:
  requirements:
  - key: "kubernetes.io/arch"
    operator: In
    values: ["arm64", "amd64"]
```

Avec Cluster Autoscaler, vous devez créer un groupe de nœuds pour les instances Graviton et définir des tolérances de nœuds sur votre charge de travail afin d'utiliser la nouvelle capacité.

GPUs et FPGAs peut augmenter considérablement les performances de votre charge de travail, mais celle-ci devra être optimisée pour utiliser l'accélérateur. De nombreux types de charges de travail pour l'apprentissage automatique et l'intelligence artificielle peuvent être utilisés GPUs pour le calcul, et les instances peuvent être ajoutées à un cluster et montées dans une charge de travail à l'aide de demandes de ressources.

```
spec:
  template:
    spec:
    - containers:
      ...
      resources:
          limits:
            nvidia.com/gpu: "1"
```

Certains matériels GPU peuvent être partagés entre plusieurs charges de travail afin qu'un seul GPU puisse être provisionné et utilisé. Pour savoir comment configurer le partage de charge de travail par GPU, consultez le plug-in du périphérique GPU virtuel pour plus d'informations. Vous pouvez également consulter les blogs suivants :

- Partage de GPU sur Amazon EKS avec le découpage temporel et les instances accélérées NVIDIA EC2
- Optimisation de l'utilisation des GPU avec le GPU multi-instances (MIG) de NVIDIA sur Amazon EKS : exécution d'un plus grand nombre de pods par GPU pour des performances améliorées

Optimisation des coûts - Mise en réseau

L'architecture des systèmes pour une haute disponibilité (HA) est une bonne pratique pour garantir la résilience et la tolérance aux pannes. Dans la pratique, cela signifie répartir vos charges de travail et l'infrastructure sous-jacente sur plusieurs zones de disponibilité (AZs) dans une région AWS donnée. La mise en place de ces caractéristiques pour votre environnement Amazon EKS améliorera la fiabilité globale de votre système. Parallèlement à cela, vos environnements EKS seront probablement également composés d'une variété de constructions (c'est-à-dire VPCs), de composants (c'est-à-dire) et d'intégrations (par exemple, ECR et autres registres de conteneurs ELBs).

La combinaison de systèmes à haute disponibilité et d'autres composants spécifiques à des cas d'utilisation peut jouer un rôle important dans la manière dont les données sont transférées et traitées. Cela aura à son tour un impact sur les coûts liés au transfert et au traitement des données.

Les pratiques détaillées ci-dessous vous aideront à concevoir et à optimiser vos environnements EKS afin d'atteindre un meilleur rapport coût-efficacité pour différents domaines et cas d'utilisation.

Communication d'un pod à un autre

En fonction de votre configuration, la communication réseau et le transfert de données entre les pods peuvent avoir un impact significatif sur le coût global d'exécution des charges de travail Amazon EKS. Cette section abordera différents concepts et approches visant à atténuer les coûts liés à la communication entre les pods, tout en tenant compte des architectures à haute disponibilité (HA), des performances des applications et de la résilience.

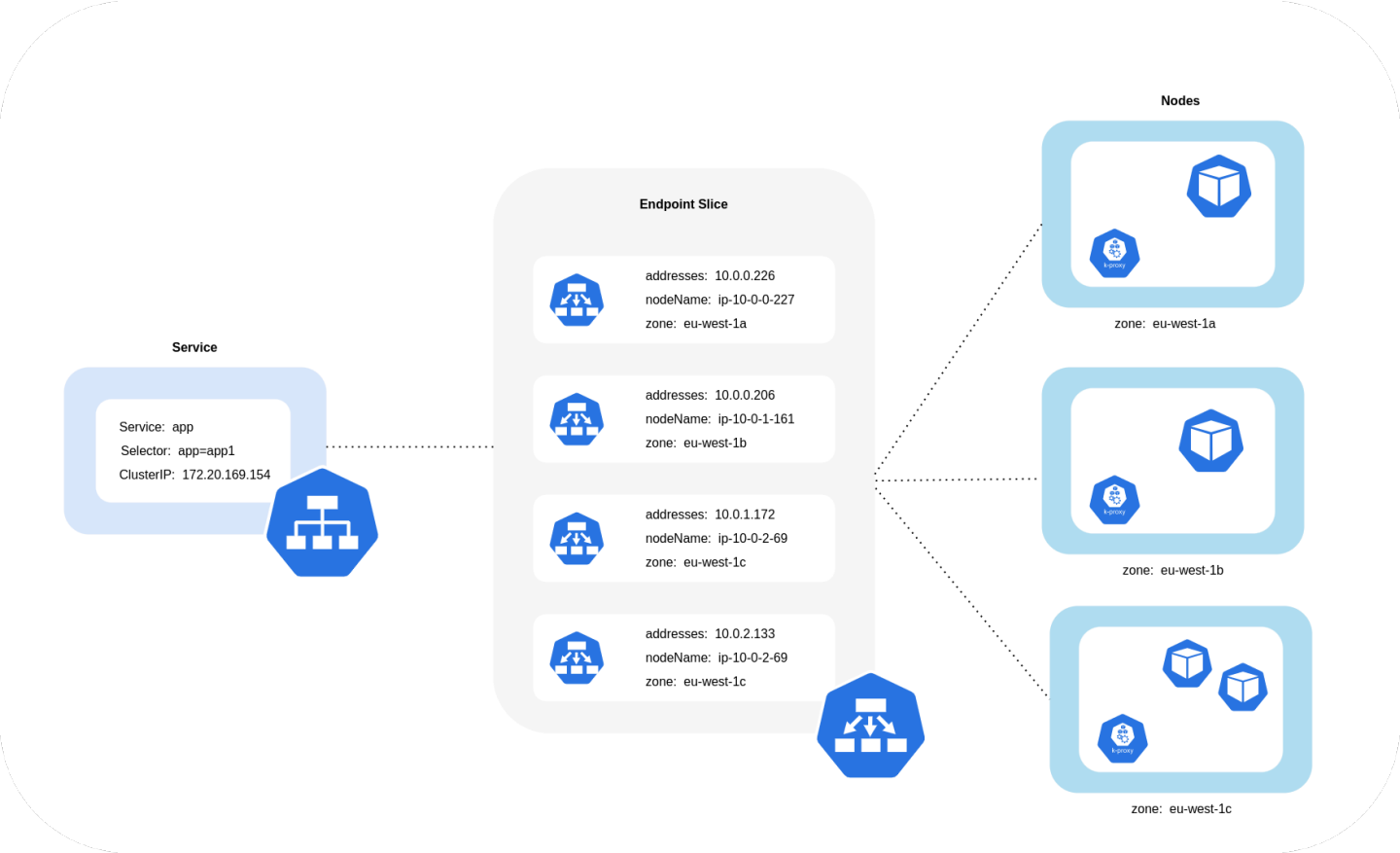
Limiter le trafic à une zone de disponibilité

Le projet Kubernetes a commencé très tôt à développer des constructions sensibles à la topologie, notamment des étiquettes telles que kubernetes. io/hostname, topology.kubernetes.io/region, and topology.kubernetes.io/zoneattribué aux nœuds pour activer des fonctionnalités telles que la répartition de la charge de travail entre les domaines de défaillance et les approvisionneurs

de volumes sensibles à la topologie. Après avoir obtenu leur diplôme dans Kubernetes 1.17, les étiquettes ont également été utilisées pour permettre des fonctionnalités de routage adaptées à la topologie pour la communication entre pods.

Vous trouverez ci-dessous quelques stratégies permettant de contrôler le volume de trafic cross-AZ entre les pods de votre cluster EKS afin de réduire les coûts et de minimiser la latence.

Si vous souhaitez obtenir une visibilité précise de la quantité de trafic cross-AZ entre les pods de votre cluster (par exemple, la quantité de données transférées en octets), consultez cet article.



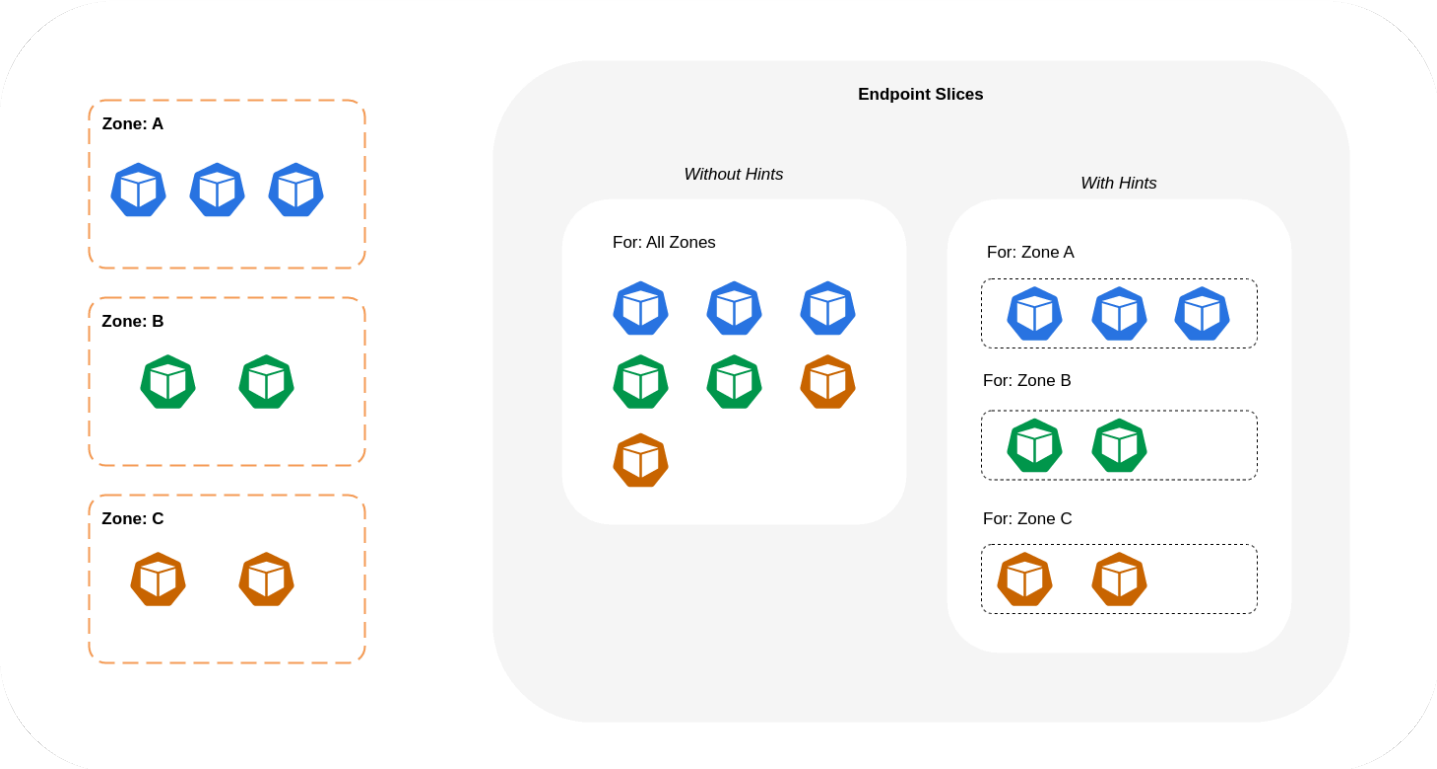
Comme le montre le schéma ci-dessus, les services sont la couche d'abstraction réseau stable qui reçoit le trafic destiné à vos pods. Lorsqu'un service est créé, plusieurs EndpointSlices sont créés. Chacun EndpointSlice possède une liste de points de terminaison contenant un sous-ensemble d'adresses Pod ainsi que les nœuds sur lesquels ils s'exécutent et toute information topologique supplémentaire. Lors de l'utilisation de l'Amazon VPC CNI, kube-proxy, un daemonset exécuté sur chaque nœud, maintient les règles réseau pour permettre la communication avec le pod et la découverte de services (une alternative basée sur eBPF CNIs peut ne pas utiliser kube-proxy mais fournir un comportement équivalent). Il joue le rôle de routage interne, mais il le fait en fonction de ce qu'il consomme du produit créé EndpointSlices.

Sur EKS, kube-proxy utilise principalement les règles NAT iptables (ou IPVS, NFTablescomme alternative) pour la distribution du trafic entre tous les pods du cluster, quel que soit leur nœud ou leur emplacement AZ. Cette distribution par défaut peut entraîner un routage du trafic inter-AZ, ce qui peut entraîner une augmentation de la latence pour les applications sensibles et des frais de transfert de données inter-AZ dans les déploiements de grande envergure.

Utilisation du routage adapté à la topologie (anciennement connu sous le nom de topologie sensible à la topologie)

Lorsque le routage tenant compte de la topologie est activé et implémenté sur un service Kubernetes, le EndpointSlice contrôleur alloue proportionnellement des points de terminaison aux différentes zones dans lesquelles votre cluster est réparti. Pour chacun de ces points de terminaison, le EndpointSlice contrôleur définira également un indice pour la zone. Les conseils décrivent la zone pour laquelle un point de terminaison doit desservir le trafic. `kube-proxy`acheminera ensuite le trafic d'une zone vers un point de terminaison en fonction des indications appliquées.

Le schéma ci-dessous montre comment EndpointSlices les indices sont organisés de manière à savoir vers quelle destination ils doivent se rendre en fonction de leur point d'origine zonal. `kube-proxy` Sans indications, il n'existe pas d'allocation ou d'organisation de ce type et le trafic sera acheminé par proxy vers différentes destinations zonales, quelle que soit sa provenance.



Dans certains cas, le EndpointSlice contrôleur peut appliquer un indice pour une zone différente, ce qui signifie que le point de terminaison peut finir par desservir le trafic provenant d'une autre zone. La raison en est d'essayer de maintenir une répartition uniforme du trafic entre les points de terminaison situés dans différentes zones.

Vous trouverez ci-dessous un extrait de code expliquant comment activer le routage adapté à la topologie pour un service.

```
apiVersion: v1
kind: Service
metadata:
  name: orders-service
  namespace: ecommerce
  annotations:
    service.kubernetes.io/topology-mode: Auto
spec:
  selector:
    app: orders
  type: ClusterIP
  ports:

* protocol: TCP
port: 3003
targetPort: 3003
```

La capture d'écran ci-dessous montre le résultat obtenu lorsque le EndpointSlice contrôleur a correctement appliqué un indice à un point de terminaison pour une réplique de Pod exécutée dans l'Arizona`eu-west-1a`.

```
YAML(default/express-test-ldl6r)
addressType: IPv4
apiVersion: discovery.k8s.io/v1
endpoints:
- addresses:
  - 10.0.0.226
  conditions:
    ready: true
    serving: true
    terminating: false
  hints:
    forZones:
    - name: eu-west-1a
  nodeName: ip-10-0-0-227.eu-west-1.compute.internal
  targetRef:
    kind: Pod
    name: express-test-67795c7d85-pj4k2
    namespace: default
    uid: e0655b58-0a2c-41cd-9f3e-a925256e07bf
  zone: eu-west-1a
```

> **Note**
>
> Il est important de noter que le routage tenant compte de la topologie est toujours en version bêta. Cette fonctionnalité fonctionne de manière plus prévisible avec des charges de travail réparties uniformément dans la topologie du cluster, car le contrôleur alloue les points de terminaison de manière proportionnelle entre les zones, mais peut ignorer les assignations d'indices lorsque les ressources des nœuds d'une zone sont trop déséquilibrées pour éviter une surcharge excessive. Par conséquent, il est fortement recommandé de l'utiliser en conjonction avec des contraintes de planification qui augmentent la disponibilité d'une application, telles que les contraintes de propagation de la topologie des pods. Notez que les indices peuvent également ne pas être attribués lorsque la capacité fluctue entre les zones, par exemple lors de l'utilisation d'instances ponctuelles Amazon EC2, car les interruptions ou les remplacements ne sont pas détectés en temps réel lors du calcul de la distribution proportionnelle.

Utilisation de la distribution du trafic

Introduite dans Kubernetes 1.30 et rendue généralement disponible dans la version 1.33, la distribution du trafic offre une alternative plus simple au routage basé sur la topologie pour la préférence du trafic dans la même zone. Bien que Topology Aware Routing tente d'utiliser une approche intelligente du routage du trafic pour éviter de surcharger les points de terminaison, cela a entraîné un comportement imprévisible. La distribution du trafic privilégie plutôt la prévisibilité. L' PreferClose option demande à kube-proxy de créer des règles qui acheminent d'abord le trafic vers les points de terminaison de même zone en fonction de l'indice zonal défini par le contrôleur. EndpointSlice Lorsqu'aucun point de terminaison de même zone n'est disponible, il suffit de répartir le trafic entre n'importe quel point de terminaison du cluster pour le service. Cette fonctionnalité est conçue pour les charges de travail qui acceptent le compromis consistant à optimiser la proximité plutôt que la tentative de répartition uniforme de la charge proposée par le routage basé sur la topologie.

Vous trouverez ci-dessous un extrait de code expliquant comment activer la distribution du trafic pour un service.

```
apiVersion: v1
kind: Service
metadata:
  name: orders-service
  namespace: ecommerce
spec:
  trafficDistribution: PreferClose
  selector:
    app: orders
  type: ClusterIP
  ports:

* protocol: TCP
port: 3003
targetPort: 3003
```

Lors de l'activation de la distribution du trafic, un défi courant se pose : les points de terminaison d'une même zone de zone peuvent être surchargés si la majeure partie du trafic provient de cette même zone. Cette surcharge peut créer des problèmes importants :

- Un seul HPA (Horizontal Pod Autoscaler) gérant un déploiement multi-AZ peut réagir en répartissant les pods entre différents. AZs Cependant, cette action ne permet pas de remédier efficacement à l'augmentation de la charge dans la zone affectée.

- Cette situation peut à son tour entraîner une inefficacité des ressources. Lorsque des autoscalers de clusters tels que Karpenter détectent le scale-out entre différents modules AZs, ils peuvent approvisionner des nœuds supplémentaires dans les nœuds non affectés AZs, ce qui entraîne une allocation de ressources inutile.

Pour relever ce défi :

- Créez des déploiements distincts par zone, qui auraient leur propre capacité HPAs à évoluer indépendamment les uns des autres.
- Tirez parti des contraintes de répartition topologique pour garantir la répartition de la charge de travail au sein du cluster, ce qui permet d'éviter la surcharge des terminaux dans les zones à fort trafic.

Utilisation d'autoscalers : provisionner des nœuds vers une zone de disponibilité spécifique

Nous vous recommandons vivement d'exécuter vos charges de travail dans des environnements à haute disponibilité répartis sur plusieurs AZs environnements. Cela améliore la fiabilité de vos applications, en particulier en cas d'incident lié à un problème avec un AZ. Si vous êtes prêt à sacrifier la fiabilité pour réduire les coûts liés au réseau, vous pouvez limiter vos nœuds à une seule AZ.

Pour exécuter tous vos pods dans la même zone de zone, configurez les nœuds de travail dans la même zone ou planifiez les pods sur les nœuds de travail exécutés sur la même zone de zone. Pour approvisionner des nœuds au sein d'une seule AZ, définissez un groupe de nœuds avec des sous-réseaux appartenant à la même AZ avec Cluster Autoscaler (CA). Pour Karpenter, utilisez `topology.kubernetes.io/zone` et spécifiez l'AZ dans laquelle vous souhaitez créer les nœuds de travail. Par exemple, l'extrait de code Karpenter Provisioner ci-dessous approvisionne les nœuds de l'AZ us-west-2a.

Charpentier

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
name: single-az
spec:
  requirements:

* key: "topology.kubernetes.io/zone"`
```

```
operator: In
values: ["us-west-2a"]
```

Cluster Autoscaler (CA)

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-ca-cluster
  region: us-east-1
  version: "1.21"
availabilityZones:

* us-east-1a
managedNodeGroups:
* name: managed-nodes
labels:
  role: managed-nodes
instanceType: t3.medium
minSize: 1
maxSize: 10
desiredCapacity: 1
...
```

Utilisation de l'attribution des pods et de l'affinité des nœuds

Sinon, si vous avez plusieurs nœuds de travail exécutés en plusieurs AZs, chaque nœud portera l'étiquette topology.kubernetes.io/zone avec la valeur de son AZ (par exemple us-west-2a ou us-west-2b). Vous pouvez utiliser `nodeSelector` ou `nodeAffinity` planifier des pods pour les nœuds dans une seule zone de disponibilité. Par exemple, le fichier manifeste suivant planifiera le Pod dans un nœud exécuté dans AZ us-west-2a.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  nodeSelector:
    topology.kubernetes.io/zone: us-west-2a
  containers:
```

```
* name: nginx
image: nginx
imagePullPolicy: IfNotPresent
```

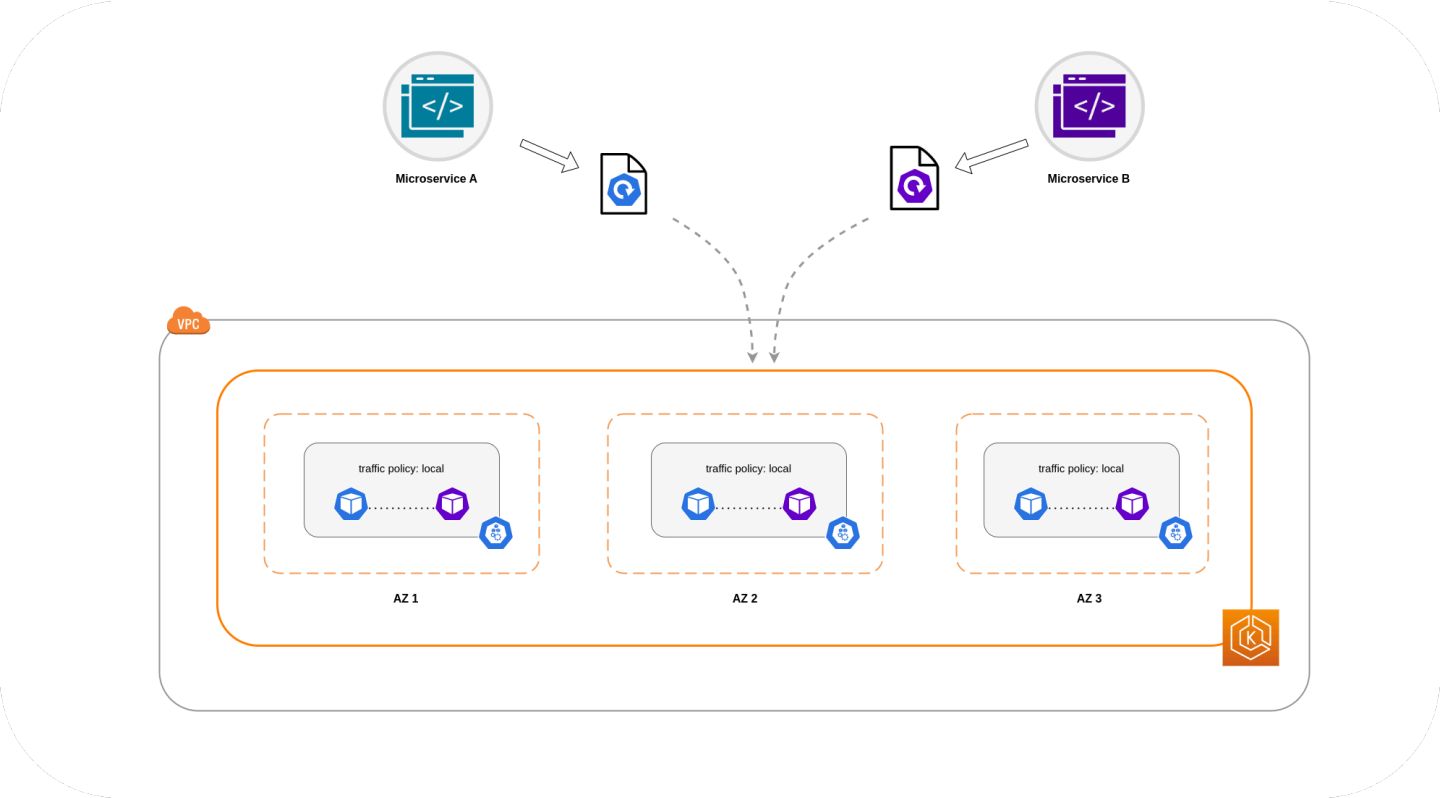
Restreindre le trafic vers un nœud

Dans certains cas, il ne suffit pas de restreindre le trafic au niveau d'une zone. Outre la réduction des coûts, vous devrez peut-être également réduire la latence du réseau entre certaines applications qui communiquent fréquemment entre elles. Afin d'optimiser les performances du réseau et de réduire les coûts, vous devez trouver un moyen de limiter le trafic vers un nœud spécifique. Par exemple, le microservice A doit toujours communiquer avec le microservice B sur le nœud 1, même dans les configurations à haute disponibilité (HA). Le fait que le microservice A sur le nœud 1 communique avec le microservice B sur le nœud 2 peut avoir un impact négatif sur les performances souhaitées pour les applications de cette nature, en particulier si le nœud 2 se trouve dans une zone de disponibilité complètement distincte.

Utilisation de la politique de trafic interne du service

Afin de limiter le trafic réseau du Pod à un nœud, vous pouvez utiliser la politique de trafic interne du service. Par défaut, le trafic envoyé au service d'une charge de travail sera distribué de manière aléatoire entre les différents points de terminaison générés. Ainsi, dans une architecture HA, cela signifie que le trafic provenant du microservice A peut être dirigé vers n'importe quelle réplique du microservice B sur un nœud donné à travers les différents nœuds. AZs Toutefois, lorsque la politique de trafic interne du service est définie sur`Local`, le trafic sera limité aux points de terminaison situés sur le nœud d'où provient le trafic. Cette politique impose l'utilisation exclusive de points de terminaison locaux aux nœuds. Par conséquent, les coûts liés au trafic réseau pour cette charge de travail seront inférieurs à ceux d'une distribution à l'échelle du cluster. De plus, la latence sera plus faible, ce qui rendra votre application plus performante.

> **Note**
>
> Il est important de noter que cette fonctionnalité ne peut pas être combinée avec le routage tenant compte de la topologie dans Kubernetes.



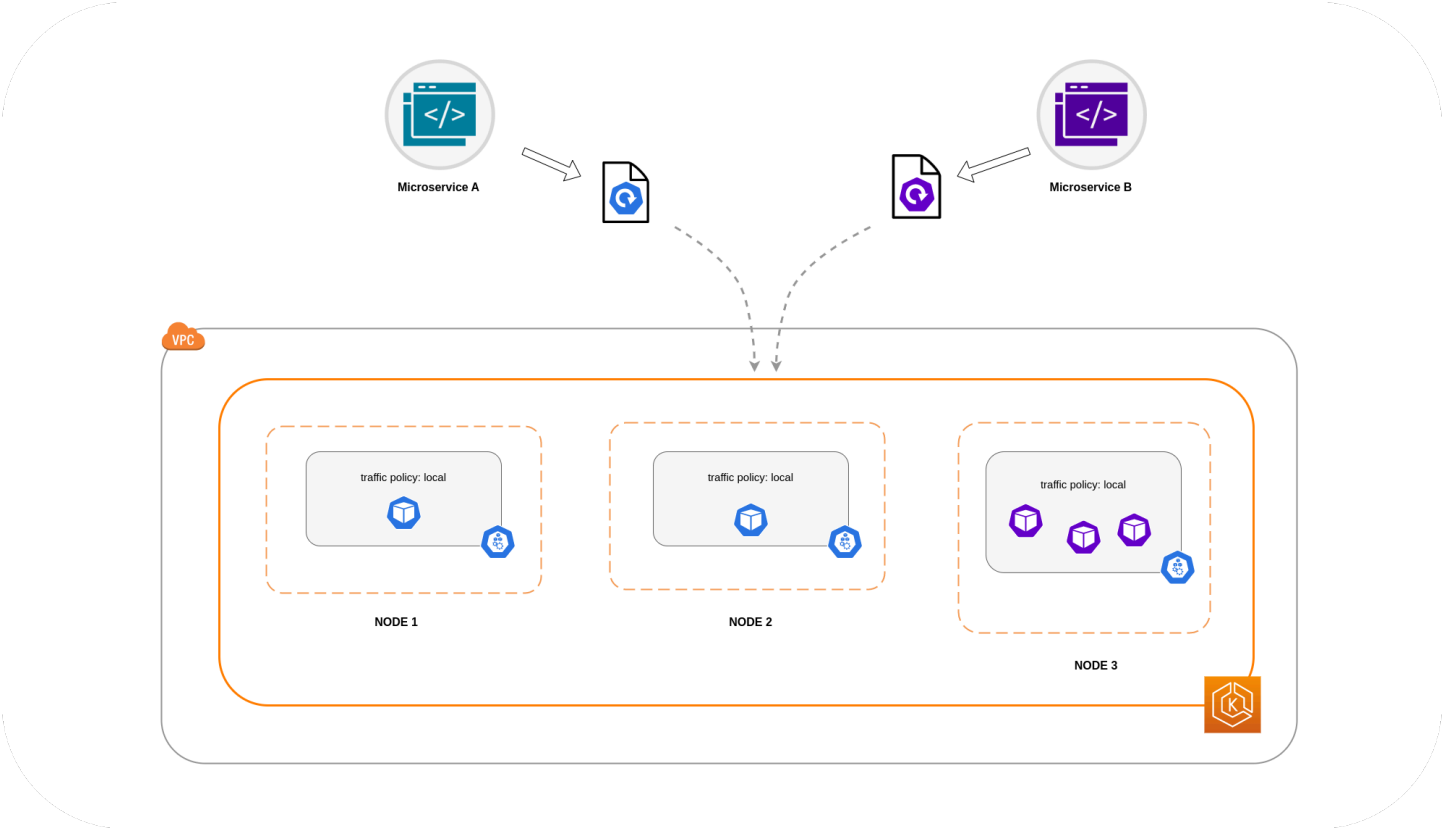
Vous trouverez ci-dessous un extrait de code expliquant comment définir la politique de trafic interne pour un service.

```
apiVersion: v1
kind: Service
metadata:
  name: orders-service
  namespace: ecommerce
spec:
  selector:
    app: orders
  type: ClusterIP
  ports:

* protocol: TCP
port: 3003
targetPort: 3003
  internalTrafficPolicy: Local
```

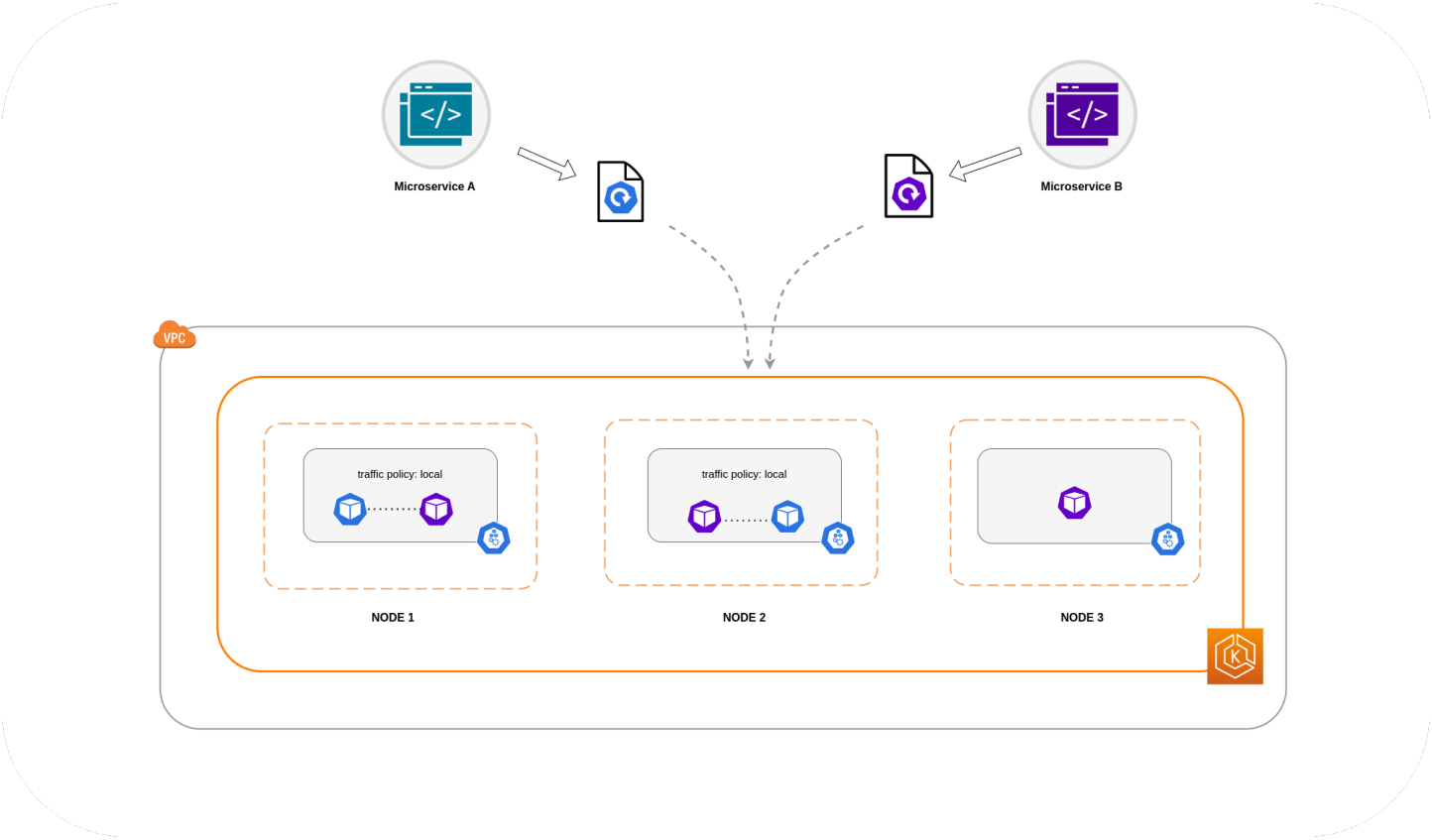
Pour éviter que votre application ne se comporte de manière inattendue en raison d'une baisse de trafic, vous devez envisager les approches suivantes :

- Exécutez suffisamment de répliques pour chacun des pods communicants
- Disposez d'une répartition relativement uniforme des pods en utilisant les contraintes de propagation topologique
- Utiliser les règles d'affinité des pods pour la colocalisation des pods communicants

Dans cet exemple, vous avez 2 répliques du microservice A et 3 répliques du microservice B. Si les répliques du microservice A sont réparties entre les nœuds 1 et 2 et que le microservice B possède ses 3 répliques sur le nœud 3, il ne sera pas en mesure de communiquer en raison de la politique de trafic interne. `Local` Lorsqu'aucun point de terminaison local n'est disponible, le trafic est supprimé.



Si le microservice B possède 2 de ses 3 répliques sur les nœuds 1 et 2, il y aura une communication entre les applications homologues. Mais vous auriez toujours une réplique isolée de Microservice B sans aucune réplique homologue avec laquelle communiquer.



> **Note**
>
> Dans certains scénarios, une réplique isolée telle que celle illustrée dans le schéma ci-dessus peut ne pas être préoccupante si elle répond toujours à un objectif (par exemple, répondre aux demandes provenant du trafic entrant externe).

Utilisation de la politique de trafic interne du service avec des contraintes de dispersion topologique

L'utilisation de la politique de trafic interne associée aux contraintes de dispersion topologique peut être utile pour garantir que vous disposez du nombre approprié de répliques pour communiquer des microservices sur différents nœuds.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: express-test
spec:
  replicas: 6
  selector:
```

```
    matchLabels:
      app: express-test
  template:
    metadata:
      labels:
        app: express-test
        tier: backend
    spec:
      topologySpreadConstraints:
      - maxSkew: 1
        topologyKey: "topology.kubernetes.io/zone"
        whenUnsatisfiable: ScheduleAnyway
        labelSelector:
          matchLabels:
            app: express-test
```

Utilisation de la politique de trafic interne du service avec les règles d'affinité des pods

Une autre approche consiste à utiliser les règles d'affinité des pods lors de l'utilisation de la politique de trafic interne du service. Grâce à l'affinité des pods, vous pouvez influencer le planificateur pour qu'il colocalise certains pods en raison de leurs communications fréquentes. En appliquant des contraintes de planification strictes (`requiredDuringSchedulingIgnoredDuringExecution`) à certains pods, vous obtiendrez de meilleurs résultats en matière de colocation de pods lorsque le planificateur place des pods sur des nœuds.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: graphql
  namespace: ecommerce
  labels:
    app.kubernetes.io/version: "0.1.6"
    ...
    spec:
      serviceAccountName: graphql-service-account
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
```

```
                - orders
            topologyKey: "kubernetes.io/hostname"
```

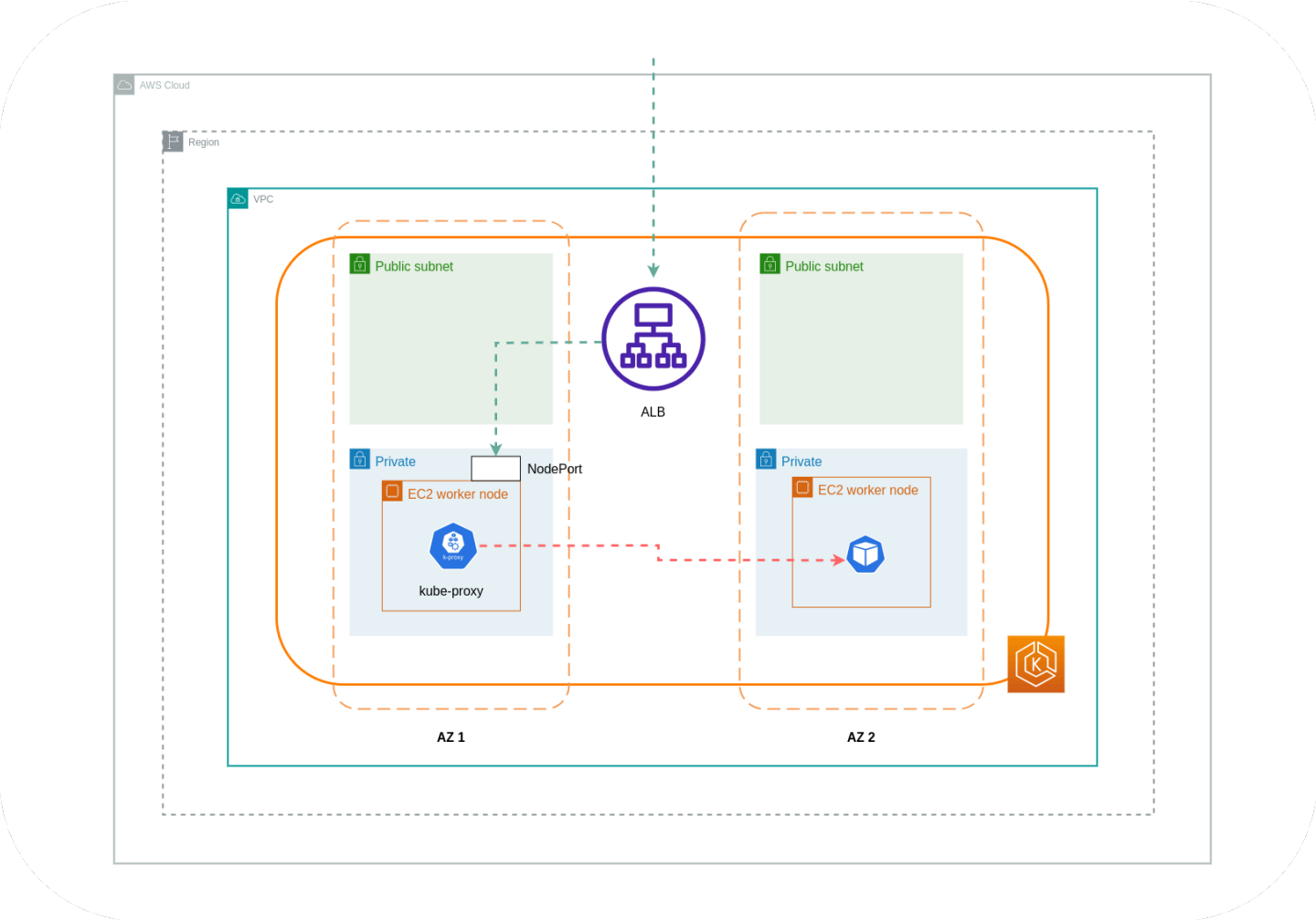
Communication entre le Load Balancer et le Pod

Les charges de travail EKS sont généralement dirigées par un équilibreur de charge qui distribue le trafic aux pods concernés de votre cluster EKS. Votre architecture peut comprendre des équilibreurs de charge internes orientés vers l' and/or extérieur. En fonction de votre architecture et de la configuration du trafic réseau, la communication entre les équilibreurs de charge et les pods peut contribuer de manière significative aux frais de transfert de données.

Vous pouvez utiliser le AWS Load Balancer Controller pour gérer automatiquement la création de ressources ELB (ALB et NLB). Les frais de transfert de données que vous devrez payer dans le cadre de telles configurations dépendront du chemin emprunté par le trafic réseau. Le contrôleur AWS Load Balancer prend en charge deux modes de trafic réseau, le mode instance et le mode IP.

Lorsque vous utilisez le mode instance, un nœud NodePort sera ouvert sur chaque nœud de votre cluster EKS. L'équilibreur de charge transfère ensuite le trafic de manière uniforme sur les nœuds. Si le Pod de destination est exécuté sur un nœud, aucun frais de transfert de données ne sera encouru. Toutefois, si le pod de destination se trouve sur un nœud distinct et dans une zone de zone différente de celle NodePort où le trafic est reçu, il y aura un saut réseau supplémentaire entre le kube-proxy et le pod de destination. Dans un tel scénario, des frais de transfert de données inter-AZ seront facturés. En raison de la répartition uniforme du trafic entre les nœuds, il est fort probable que des frais de transfert de données supplémentaires soient associés aux sauts de trafic réseau entre zones des kube-proxies vers les pods de destination concernés.

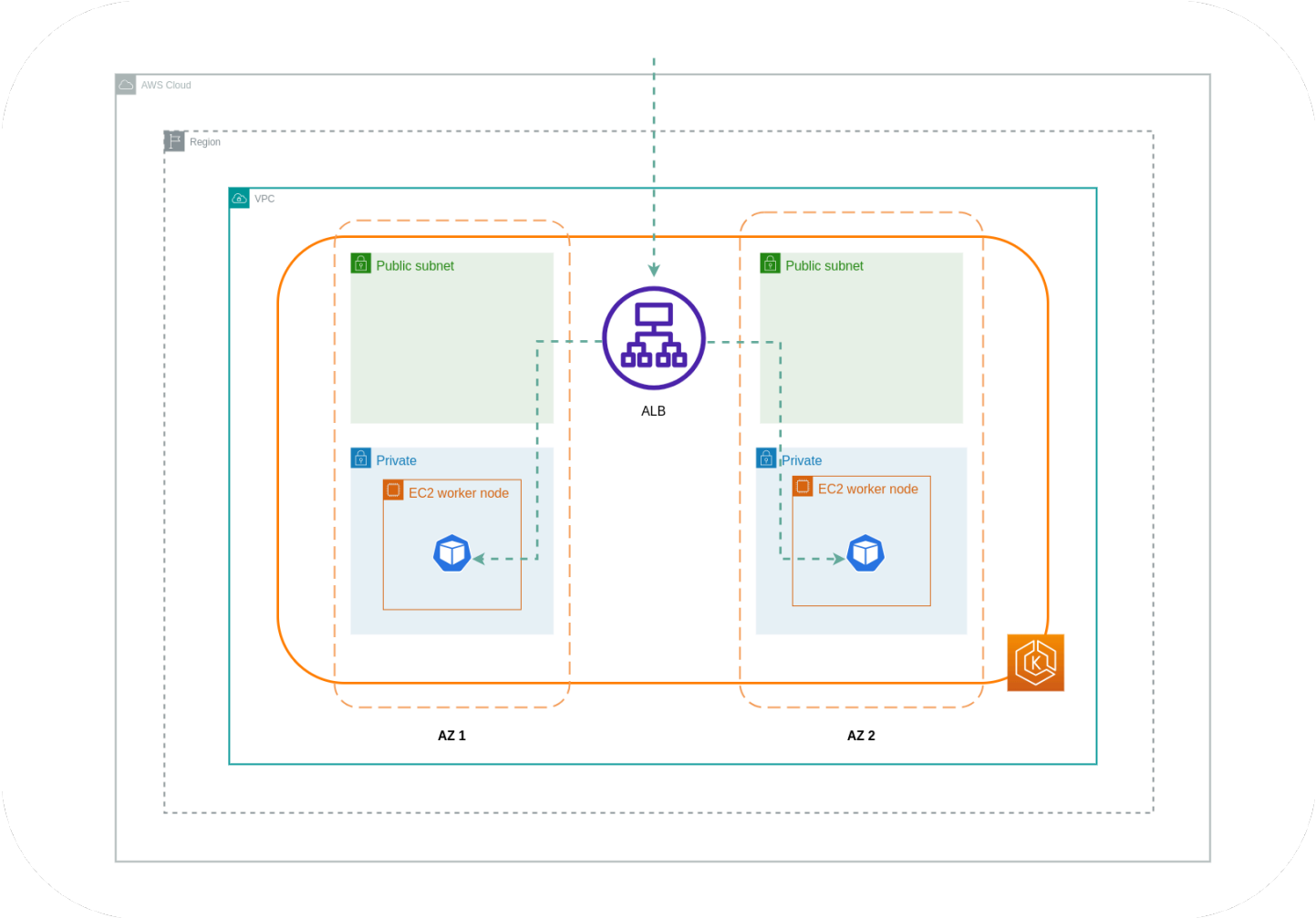
Le schéma ci-dessous représente un chemin réseau pour le trafic circulant de l'équilibreur de charge vers le Pod de destination NodePort, puis depuis celui-ci `kube-proxy` vers le Pod de destination sur un nœud distinct dans une autre zone de disponibilité. Il s'agit d'un exemple du paramètre du mode instance.



Lorsque le mode IP est utilisé, le trafic réseau est transmis par proxy depuis l'équilibreur de charge directement vers le Pod de destination. Par conséquent, cette approche n'entraîne aucun frais de transfert de données.

> **Note**
>
> Il est recommandé de configurer votre équilibreur de charge en mode trafic IP afin de réduire les frais de transfert de données. Pour cette configuration, il est également important de vous assurer que votre équilibreur de charge est déployé sur tous les sous-réseaux de votre VPC.

Le schéma ci-dessous décrit les chemins réseau pour le trafic circulant entre l'équilibreur de charge et les Pods en mode IP du réseau.



Transfert de données depuis le registre des conteneurs

Amazon ECR

Le transfert de données vers le registre privé Amazon ECR est gratuit. Le transfert de données à l'intérieur de la région est gratuit, mais le transfert de données vers Internet et entre les régions sera facturé aux tarifs de transfert de données Internet des deux côtés du transfert.

Vous devez utiliser la fonction de réplication d'image ECRs intégrée pour répliquer les images de conteneur pertinentes dans la même région que vos charges de travail. De cette façon, la réplication serait facturée une seule fois, et toutes les extractions d'images de la même région (intra-région) seraient gratuites.

Vous pouvez réduire davantage les coûts de transfert de données associés à l'extraction d'images depuis l'ECR (transfert de données sortant) en utilisant les points de terminaison VPC d'interface pour vous connecter aux référentiels ECR de la région. L'approche alternative consistant à se

connecter au point de terminaison AWS public d'ECR (via une passerelle NAT et une passerelle Internet) entraînera des coûts de traitement et de transfert de données plus élevés. La section suivante abordera plus en détail la réduction des coûts de transfert de données entre vos charges de travail et les services AWS.

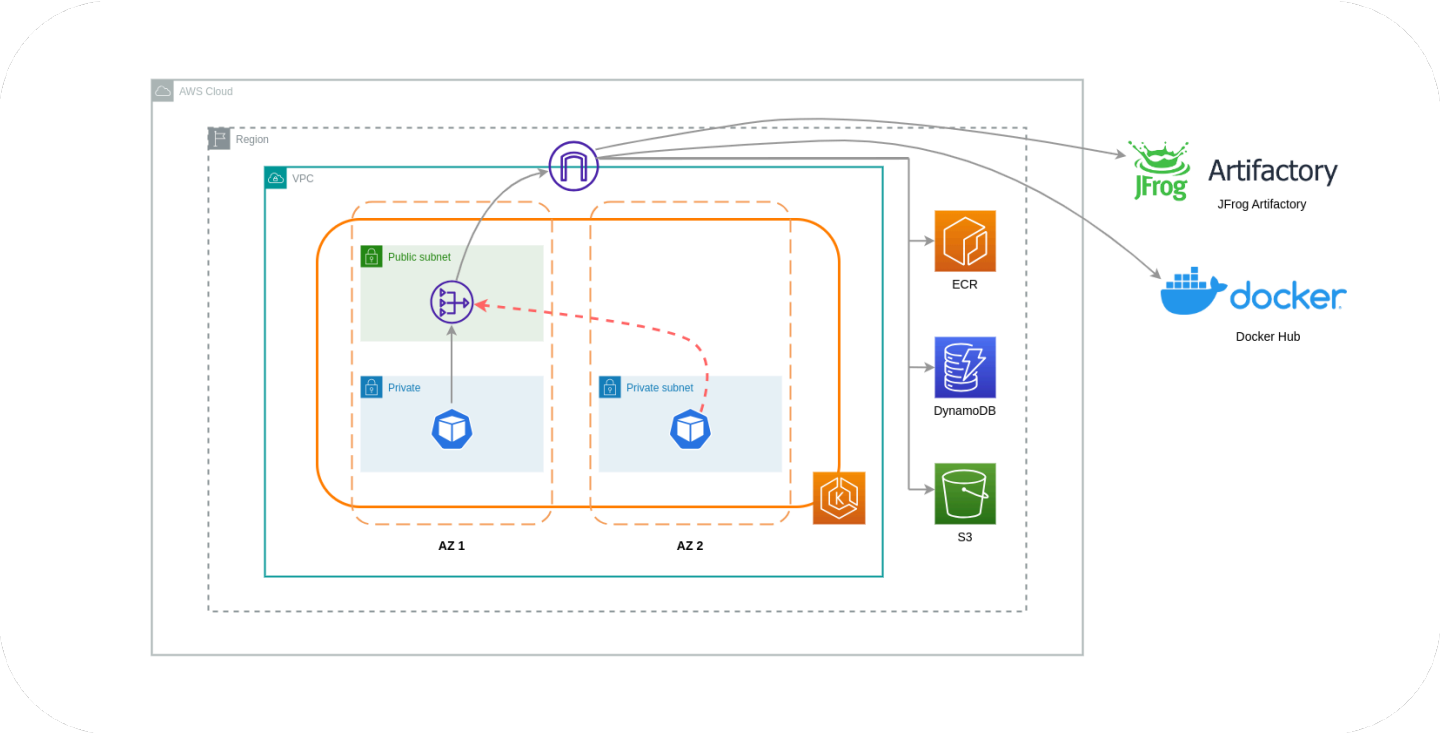
Si vous exécutez des charges de travail avec des images particulièrement volumineuses, vous pouvez créer vos propres Amazon Machine Images (AMIs) personnalisées à partir d'images de conteneur prémises en cache. Cela peut réduire le temps d'extraction initial de l'image et les coûts potentiels de transfert de données d'un registre de conteneurs vers les nœuds de travail EKS.

Transfert de données vers Internet et les services AWS

Il est courant d'intégrer les charges de travail Kubernetes à d'autres services AWS ou à des outils et plateformes tiers via Internet. L'infrastructure réseau sous-jacente utilisée pour acheminer le trafic vers et depuis la destination concernée peut avoir un impact sur les coûts engagés dans le processus de transfert de données.

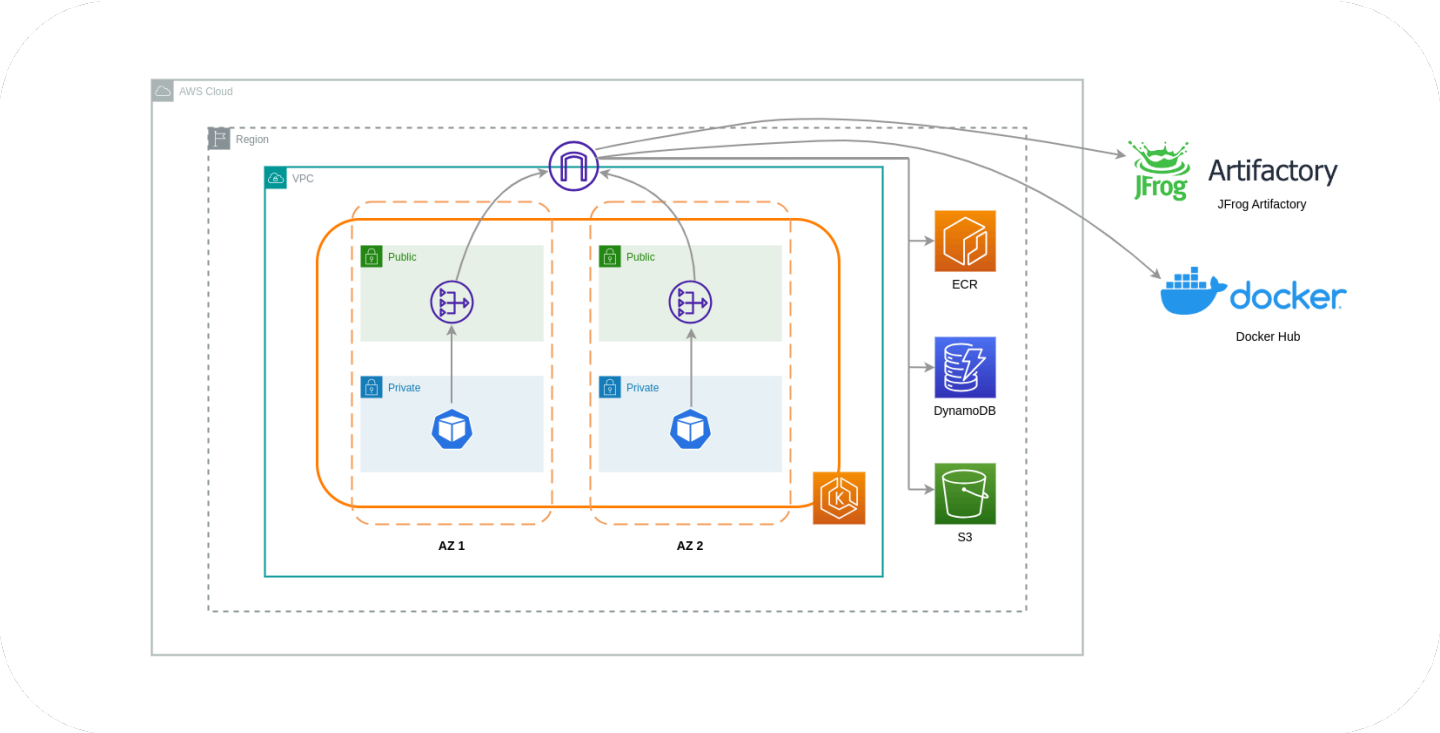
Utilisation de passerelles NAT

Les passerelles NAT sont des composants réseau qui effectuent la traduction d'adresses réseau (NAT). Le schéma ci-dessous décrit les pods d'un cluster EKS communiquant avec d'autres services AWS (Amazon ECR, DynamoDB et S3) et des plateformes tierces. Dans cet exemple, les Pods s'exécutent séparément AZs dans des sous-réseaux privés. Pour envoyer et recevoir du trafic depuis Internet, une passerelle NAT est déployée sur le sous-réseau public d'une AZ, permettant à toutes les ressources possédant des adresses IP privées de partager une seule adresse IP publique pour accéder à Internet. Cette passerelle NAT communique à son tour avec le composant Internet Gateway, permettant aux paquets d'être envoyés à leur destination finale.



Lorsque vous utilisez des passerelles NAT pour de tels cas d'utilisation, vous pouvez minimiser les coûts de transfert de données en déployant une passerelle NAT dans chaque zone de disponibilité. De cette façon, le trafic acheminé vers Internet passera par la passerelle NAT dans la même zone de zone, évitant ainsi le transfert de données entre zones de zone de zone. Cependant, même si vous économiserez sur le coût du transfert de données inter-AZ, cette configuration implique le coût d'une passerelle NAT supplémentaire dans votre architecture.

Cette approche recommandée est illustrée dans le schéma ci-dessous.



Utilisation de points de terminaison de VPC

Pour réduire davantage les coûts dans de telles architectures, vous devez utiliser des points de terminaison VPC pour établir la connectivité entre vos charges de travail et les services AWS. Les points de terminaison VPC vous permettent d'accéder aux services AWS depuis un VPC sans data/network que les paquets ne transitent par Internet. Tout le trafic est interne et reste au sein du réseau AWS. Il existe deux types de points de terminaison VPC : les points de terminaison VPC d'interface (pris en charge par de nombreux services AWS) et les points de terminaison VPC Gateway (uniquement pris en charge par S3 et DynamoDB).

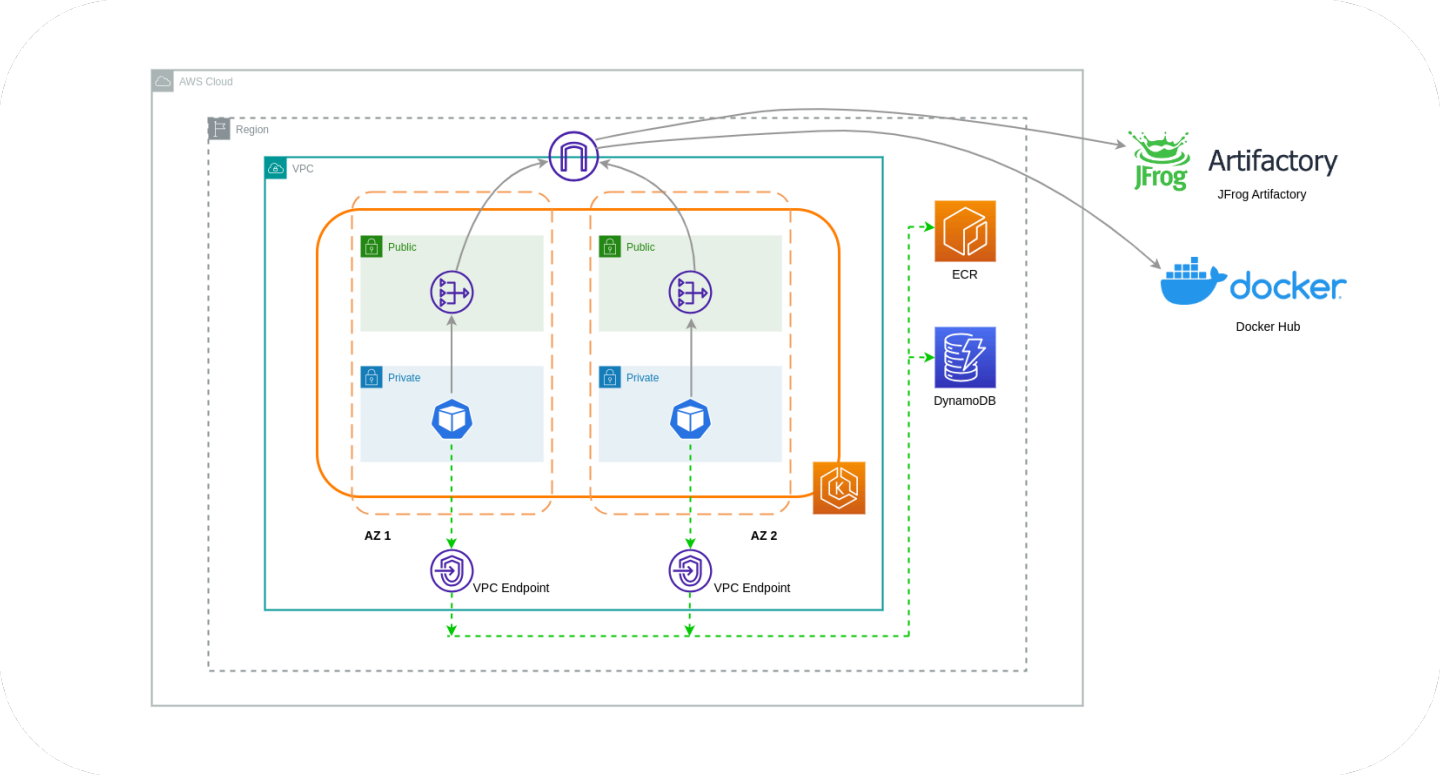
Points de terminaison VPC Gateway

Aucun coût horaire ou de transfert de données n'est associé aux points de terminaison VPC Gateway. Lorsque vous utilisez des points de terminaison VPC Gateway, il est important de noter qu'ils ne sont pas extensibles au-delà des limites des VPC. Ils ne peuvent pas être utilisés dans le peering VPC, les réseaux VPN ou via Direct Connect.

Points de terminaison VPC d'interface

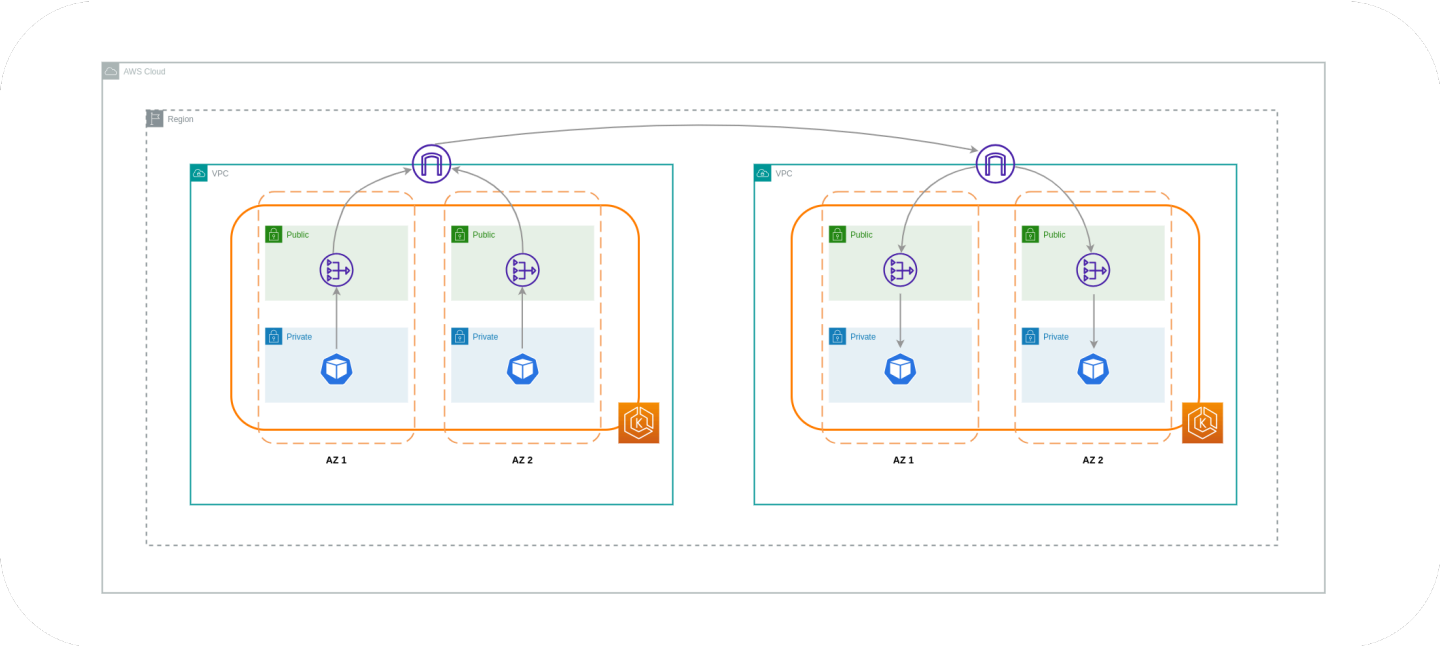
Les points de terminaison VPC sont facturés à l'heure et sont soumis à des frais supplémentaires associés au traitement des données via l'ENI sous-jacent. Notez que le transfert de données inter-AZ n'est pas facturé.

Le schéma ci-dessous montre les pods communiquant avec les services AWS via des points de terminaison VPC.



Transfert de données entre VPCs

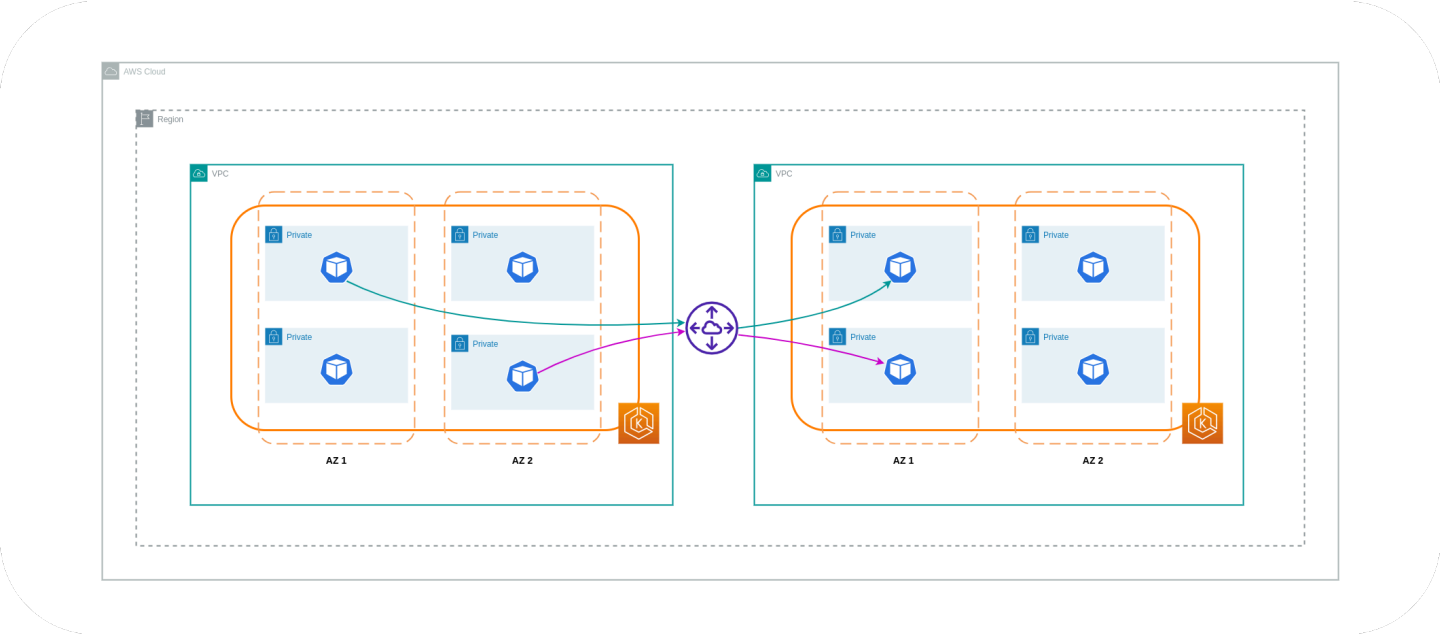
Dans certains cas, vous pouvez avoir des charges de travail dans des VPC distincts (au sein de la même région AWS) qui doivent communiquer entre eux. Cela peut être accompli en autorisant le trafic à traverser l'Internet public via des passerelles Internet connectées aux VPC respectifs. Une telle communication peut être activée en déployant des composants d'infrastructure tels que des instances EC2, des passerelles NAT ou des instances NAT dans des sous-réseaux publics. Cependant, une configuration incluant ces composants entraînera des frais pour les processing/transferring données entrantes et sortantes du VPCs. Si le trafic à destination et en provenance de la ligne séparée VPCs se déplace AZs, le transfert de données sera soumis à des frais supplémentaires. Le schéma ci-dessous décrit une configuration qui utilise des passerelles NAT et des passerelles Internet pour établir une communication entre des charges de travail de différents types. VPCs



Connexions d'appairage de VPC

Pour réduire les coûts liés à de tels cas d'utilisation, vous pouvez utiliser le peering VPC. Avec une connexion d'appairage VPC, aucun frais de transfert de données n'est facturé pour le trafic réseau qui reste dans la même zone de distribution. Si le trafic se croise AZs, des frais seront encourus. Néanmoins, l'approche de peering VPC est recommandée pour une communication rentable entre des charges de travail distinctes au sein d'une même VPCs région AWS. Cependant, il est important de noter que le peering VPC est principalement efficace pour la connectivité VPC 1:1, car il ne permet pas la mise en réseau transitive.

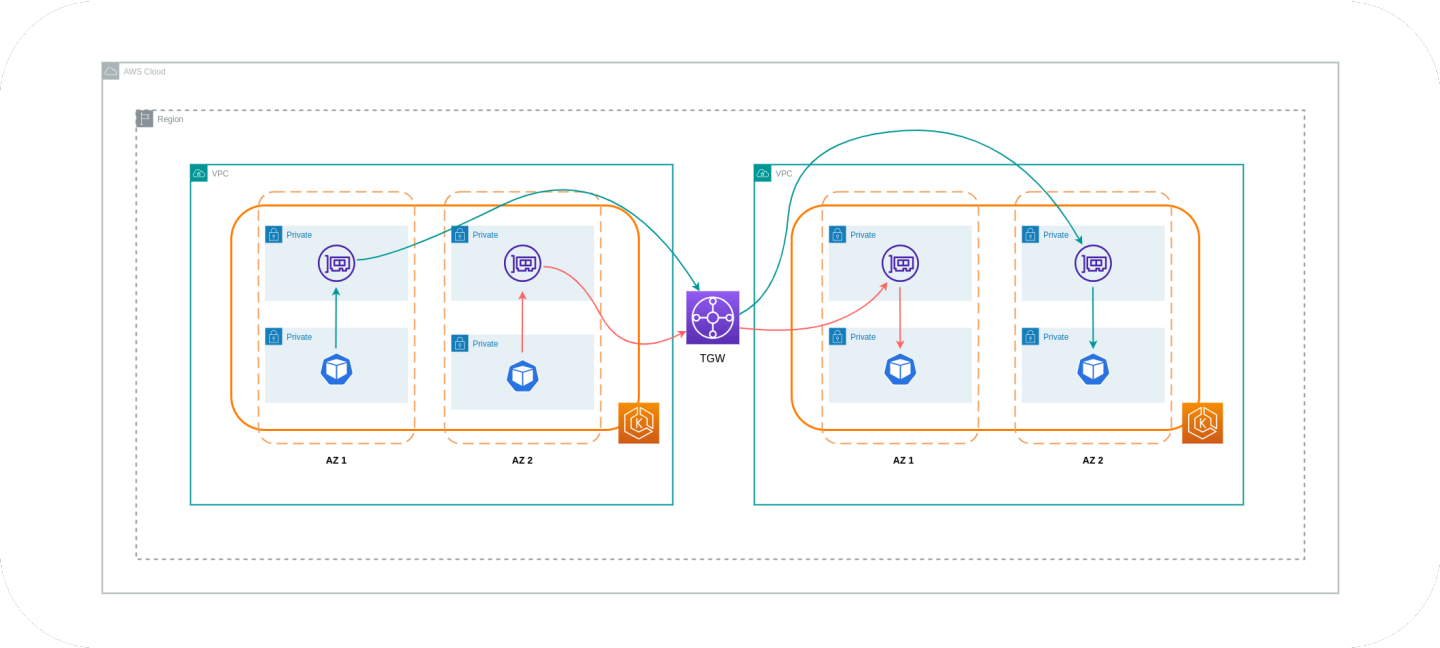
Le schéma ci-dessous est une représentation de haut niveau de la communication des charges de travail via une connexion d'appairage VPC.



Connexions réseau transitives

Comme indiqué dans la section précédente, les connexions d'appairage VPC ne permettent pas la connectivité réseau transitive. Si vous souhaitez en connecter 3 ou plus VPCs avec des exigences de réseau transitives, vous devez utiliser un Transit Gateway (TGW). Cela vous permettra de surmonter les limites de l'appairage VPC ou de surmonter toute surcharge opérationnelle associée à la présence de plusieurs connexions d'appairage VPC entre plusieurs. VPCs Vous êtes facturé sur une base horaire et pour les données envoyées au TGW. Aucun coût de destination n'est associé au trafic inter-AZ qui passe par le TGW.

Le schéma ci-dessous montre le trafic inter-AZ passant par un TGW entre des charges de travail situées dans des régions AWS différentes VPCs mais au sein de la même région AWS.



Utilisation d'un Service Mesh

Les maillages de service offrent de puissantes fonctionnalités réseau qui peuvent être utilisées pour réduire les coûts liés au réseau dans vos environnements de clusters EKS. Cependant, vous devez examiner attentivement les tâches opérationnelles et la complexité qu'un maillage de services introduira dans votre environnement si vous en adoptez un.

Limiter le trafic aux zones de disponibilité

Utilisation de la distribution pondérée par localité d'Istio

Istio vous permet d'appliquer des politiques réseau au trafic après le routage. Cela se fait à l'aide de règles de destination telles que la distribution pondérée par localité. Grâce à cette fonctionnalité, vous pouvez contrôler le poids (exprimé en pourcentage) du trafic pouvant atteindre une certaine destination en fonction de son origine. La source de ce trafic peut provenir d'un équilibreur de charge externe (ou public) ou d'un Pod au sein du cluster lui-même. Lorsque tous les points de terminaison du Pod seront disponibles, la localité sera sélectionnée sur la base d'un algorithme d'équilibrage de charge pondéré. Si certains points de terminaison ne fonctionnent pas correctement ou ne sont pas disponibles, le poids de la localité sera automatiquement ajusté pour refléter cette modification des points de terminaison disponibles.

> **Note**
>
> Avant d'implémenter la distribution pondérée par localité, vous devez commencer par comprendre les modèles de trafic de votre réseau et les implications que la politique des règles de destination peut avoir sur le comportement de votre application. Il est donc important de mettre en place des mécanismes de suivi distribués avec des outils tels qu'AWS X-Ray ou Jaeger.

Les règles de destination Istio détaillées ci-dessus peuvent également être appliquées pour gérer le trafic entre un équilibreur de charge et les pods de votre cluster EKS. Les règles de distribution pondérées par localité peuvent être appliquées à un service qui reçoit du trafic provenant d'un équilibreur de charge à haute disponibilité (en particulier la passerelle d'entrée). Ces règles vous permettent de contrôler la quantité de trafic à destination en fonction de son origine zonale, c'est-à-dire l'équilibreur de charge dans ce cas. S'il est configuré correctement, le trafic de sortie entre zones sera réduit par rapport à un équilibreur de charge qui répartit le trafic de manière uniforme ou aléatoire entre les répliques de Pod de différentes manières. AZs

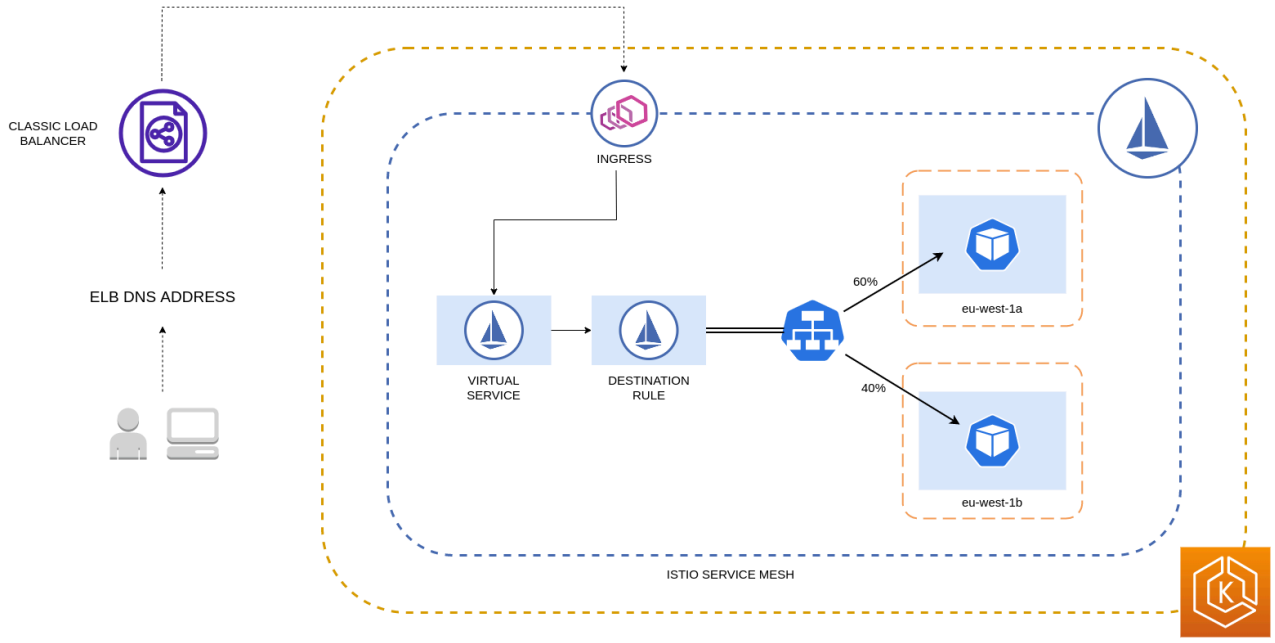
Vous trouverez ci-dessous un exemple de bloc de code d'une ressource Destination Rule dans Istio. Comme on peut le voir ci-dessous, cette ressource spécifie des configurations pondérées pour le trafic entrant provenant de 3 pays différents AZs de la `eu-west-1` région. Ces configurations indiquent que la majorité du trafic entrant (70 % dans ce cas) en provenance d'une AZ donnée doit être transmise par proxy à une destination située dans la même AZ d'où il provient.

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: express-test-dr
spec:
  host: express-test.default.svc.cluster.local
  trafficPolicy:
    loadBalancer:                      +
      localityLbSetting:
        distribute:
        - from: eu-west-1/eu-west-1a/  +
          to:
            "eu-west-1/eu-west-1a/_": 70
            "eu-west-1/eu-west-1b/_": 20
            "eu-west-1/eu-west-1c/_": 10
        - from: eu-west-1/eu-west-1b/_  +
```

```
        to:
          "eu-west-1/eu-west-1a/_": 20
          "eu-west-1/eu-west-1b/_": 70
          "eu-west-1/eu-west-1c/_": 10
      - from: eu-west-1/eu-west-1c/_  +
        to:
          "eu-west-1/eu-west-1a/_": 20
          "eu-west-1/eu-west-1b/_": 10
          "eu-west-1/eu-west-1c/*": 70**
  connectionPool:
    http:
      http2MaxRequests: 10
      maxRequestsPerConnection: 10
  outlierDetection:
    consecutiveGatewayErrors: 1
    interval: 1m
    baseEjectionTime: 30s
```

> **Note**
>
> Le poids minimum pouvant être distribué à la destination est de 1 %. La raison en est de conserver les régions et zones de basculement au cas où les terminaux de la destination principale deviendraient défectueux ou indisponibles.

Le schéma ci-dessous décrit un scénario dans lequel il existe un équilibreur de charge hautement disponible dans la région eu-west-1 et une distribution pondérée par localité est appliquée. La politique de règle de destination de ce diagramme est configurée pour envoyer 60 % du trafic en provenance d'eu-west-1a vers des pods situés dans la même zone, tandis que 40 % du trafic en provenance d'eu-west-1a doit être dirigé vers des pods situés dans eu-west-1b.



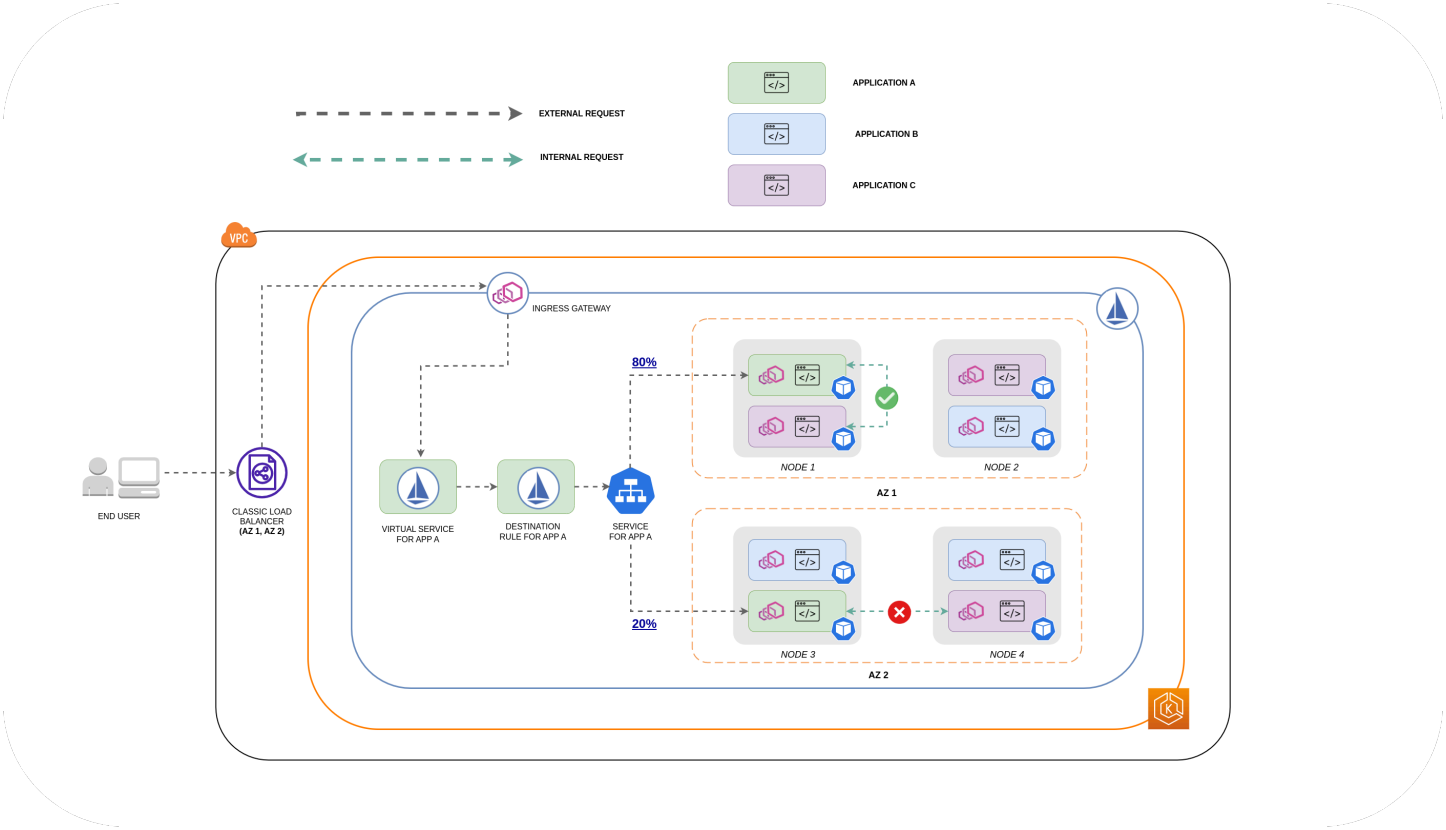
Limiter le trafic aux zones de disponibilité et aux nœuds

Utilisation de la politique de trafic interne du service avec Istio

Pour atténuer les coûts réseau associés au trafic entrant externe et au trafic interne entre les pods, vous pouvez combiner les règles de destination d'Istio et la politique de trafic interne du service Kubernetes. La manière de combiner les règles de destination d'Istio avec la politique de trafic interne du service dépendra largement de 3 éléments :

- Le rôle des microservices
- Schémas du trafic réseau sur les microservices
- Comment les microservices doivent être déployés dans la topologie du cluster Kubernetes

Le schéma ci-dessous montre à quoi ressemblerait le flux réseau dans le cas d'une demande imbriquée et comment les politiques susmentionnées contrôleraient le trafic.



1. L'utilisateur final fait une demande à APP A, qui à son tour envoie une demande imbriquée à APP C. Cette demande est d'abord envoyée à un équilibreur de charge hautement disponible, qui possède des instances dans AZ 1 et AZ 2, comme le montre le schéma ci-dessus.
2. La demande entrante externe est ensuite acheminée vers la bonne destination par le service virtuel Istio.
3. Une fois la demande acheminée, la règle de destination Istio contrôle le volume de trafic destiné à la demande en AZs fonction de son origine (AZ 1 ou AZ 2).
4. Le trafic est ensuite dirigé vers le service pour l'APP A, puis est transmis par proxy aux points de terminaison du Pod respectifs. Comme le montre le schéma, 80 % du trafic entrant est envoyé aux points de terminaison Pod dans l'AZ 1, et 20 % du trafic entrant est envoyé à l'AZ 2.
5. APP A envoie ensuite une demande interne à APP C. Le service APP C dispose d'une politique de trafic interne activée (`internalTrafficPolicy`: Local`).
6. La demande interne de l'APP A (sur le NŒUD 1) à l'APP C est réussie en raison du point de terminaison local disponible pour l'APP C.
7. La demande interne de l'APP A (sur le NŒUD 3) à l'APP C échoue car aucun point de terminaison local au nœud n'est disponible pour l'APP C. Comme le montre le schéma, APP C n'a pas de répliques sur NODE 3. * *

Les captures d'écran ci-dessous ont été capturées à partir d'un exemple réel de cette approche. La première série de captures d'écran montre une demande externe réussie adressée à un `graphql` et une demande imbriquée envoyée avec succès `graphql` à une `orders` réplique colocalisée sur le nœud. `ip-10-0-0-151.af-south-1.compute.internal`

```
test-results.txt
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-0-147.af-south-1.compute.i
NAME                          READY   STATUS    RESTARTS   AGE   IP          NODE
products-8567b458c8-brgdp     2/2     Running   0          64m   10.0.0.83   ip-10-0-0-147.af-south-1.com
----------------------------------------------------------------------
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-1-153.af-south-1.compute.i
NAME                          READY   STATUS    RESTARTS   AGE   IP          NODE
products-8567b458c8-5shq8     2/2     Running   0          65m   10.0.1.4    ip-10-0-1-153.af-south-1.comp
----------------------------------------------------------------------
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-1-68.af-south-1.compute.in
NAME                        READY   STATUS    RESTARTS   AGE   IP          NODE
orders-684bfbd6d9-5dxcf     2/2     Running   0          22m   10.0.1.145  ip-10-0-1-68.af-south-1.compu
----------------------------------------------------------------------
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-0-151.af-south-1.compute.i
NAME                        READY   STATUS    RESTARTS   AGE   IP          NODE
graphql-8f877c686-k7k9b     2/2     Running   0          18m   10.0.0.239  ip-10-0-0-151.af-south-1.comp
orders-684bfbd6d9-rc2zp     2/2     Running   0          18m   10.0.0.182  ip-10-0-0-151.af-south-1.comp
```

GET Get Orders and Pr... | No Environment

/ Microservices / GraphQL BFF / **Get Orders and Products** | Save

GET http://a9470edcf13854485a12a8fecfdabfd6-1187114975.af-south-1.elb.amazonaws.com/v1/graphql | Send

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings | Cookies

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | No schema

QUERY

```
{
    orders {
        id
        orderFor
        product {
            id
            name
        }
    }
```

GRAPHQL VARIABLES

```
1
```

Body | Cookies | Headers (7) | Test Results | Status: 200 OK | Time: 144 ms | Size: 682 B | Save Response

Pretty | Raw | Preview | Visualize | JSON

```
{
    "data": {
        "orders": [
            {
                "id": "1",
                "orderFor": "Bruce Wayne",
                "product": {
                    "id": "1a",
                    "name": "DSLR Camera"
                }
            },
            {
                "id": "2",
```

Avec Istio, vous pouvez vérifier et exporter les statistiques de tous les clusters et points de terminaison en amont connus de vos proxys. Cela peut permettre de se faire une idée du flux réseau ainsi que de la part de distribution entre les services d'une charge de travail. En reprenant le même exemple, les `orders` points de terminaison connus du `graphql` proxy peuvent être obtenus à l'aide de la commande suivante :

```
kubectl exec -it deploy/graphql -n ecommerce -c istio-proxy -- curl localhost:15000/
clusters | grep orders
```

```
...
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**rq_error::0**
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**rq_success::119**
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**rq_timeout::0**
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**rq_total::119**
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**health_flags::healthy**
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**region::af-south-1**
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**zone::af-south-1b**
...
```

Dans ce cas, le `graphql` proxy ne connaît que le `orders` point de terminaison de la réplique avec laquelle il partage un nœud. Si vous supprimez le `internalTrafficPolicy: Local` paramètre du service des commandes et que vous réexécutez une commande comme celle ci-dessus, les résultats renverront tous les points de terminaison des répliques répartis sur les différents nœuds. De plus, en examinant `rq_total` les points de terminaison respectifs, vous remarquerez une part relativement uniforme de la distribution sur le réseau. Par conséquent, si les points de terminaison sont associés à des services en amont exécutés dans des AZs environnements différents, cette distribution du réseau entre les zones entraînera des coûts plus élevés.

Comme indiqué dans une section précédente ci-dessus, vous pouvez co-localiser des pods qui communiquent fréquemment en utilisant l'affinité des pods.

```
...
spec:
...
  template:
    metadata:
      labels:
        app: graphql
        role: api
        workload: ecommerce
```

```
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - orders
            topologyKey: "kubernetes.io/hostname"
      nodeSelector:
        managedBy: karpenter
        billing-team: ecommerce
...
```

Lorsque les `orders` répliques `graphql` et ne coexistent pas sur le même nœud (`ip-10-0-0-151.af-south-1.compute.internal`), la première demande `graphql` est réussie, comme indiqué `200 response code` dans la capture d'écran de Postman ci-dessous, tandis que la deuxième demande imbriquée de `graphql` à `orders` échoue avec un. `503 response code`

```
test-results.txt
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-0-147.af-south-1.compute.i
NAME                        READY   STATUS    RESTARTS   AGE    IP          NODE
products-8567b458c8-brgdp   2/2     Running   0          114m   10.0.0.83   ip-10-0-0-147.af-south-1.co
----------------------------------------------------------------------
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-1-153.af-south-1.compute.i
NAME                        READY   STATUS    RESTARTS   AGE    IP         NODE
products-8567b458c8-5shq8   2/2     Running   0          114m   10.0.1.4   ip-10-0-1-153.af-south-1.com
----------------------------------------------------------------------
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-1-68.af-south-1.compute.in
NAME                      READY   STATUS    RESTARTS   AGE   IP          NODE
orders-684bfbd6d9-5dxcf   2/2     Running   0          72m   10.0.1.145  ip-10-0-1-68.af-south-1.compu
----------------------------------------------------------------------
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-0-151.af-south-1.compute.i
NAME                      READY   STATUS    RESTARTS   AGE   IP          NODE
graphql-8f877c686-k7k9b   2/2     Running   0          68m   10.0.0.239  ip-10-0-0-151.af-south-1.comp
```

```
QUERY                                                GRAPHQL VARIABLES
{
    orders {
        id
        orderFor
        product {
            id
            name
        }
Body  Cookies  Headers (7)  Test Results            Status: 200 OK  Time: 730 ms
Pretty  Raw  Preview  Visualize  JSON
{
    "errors": [
        {
            "message": "Request failed with status code 503",
            "locations": [
                {
                    "line": 2,
                    "column": 5
                }
            ],
            "path": [
                "orders"
            ]
        }
```

Ressources supplémentaires

- Gestion de la latence et des coûts de transfert de données sur EKS à l'aide d'Istio
- Exploration de l'effet des indices tenant compte de la topologie sur le trafic réseau dans Amazon Elastic Kubernetes Service
- Obtenir de la visibilité sur les octets réseau de votre Amazon EKS Cross-AZ d'un point à l'autre

- Optimisez le trafic AZ avec Istio
- Optimisez le trafic AZ avec un routage adapté à la topologie
- Optimisez les coûts et les performances de Kubernetes grâce à la politique de trafic interne des services
- Optimisez les coûts et les performances de Kubernetes grâce à la politique de trafic interne d'Istio and Service
- Présentation des coûts de transfert des données pour les architectures courantes
- Comprendre les coûts de transfert de données pour les services de conteneurs AWS

Stockage

Présentation

Il existe des scénarios dans lesquels vous souhaiterez peut-être exécuter des applications qui doivent préserver les données à court ou à long terme. Pour de tels cas d'utilisation, les volumes peuvent être définis et montés par des Pods afin que leurs conteneurs puissent accéder à différents mécanismes de stockage. Kubernetes prend en charge différents types de volumes pour le stockage éphémère et persistant. Le choix du stockage dépend largement des exigences de l'application. Chaque approche a des implications financières, et les pratiques décrites ci-dessous vous aideront à optimiser les coûts pour les charges de travail nécessitant une certaine forme de stockage dans vos environnements EKS.

Volumes éphémères

Les volumes éphémères sont destinés aux applications qui nécessitent des volumes locaux transitoires mais qui n'ont pas besoin que les données soient conservées après les redémarrages. Cela inclut les exigences relatives à l'espace de travail, à la mise en cache et aux données d'entrée en lecture seule, telles que les données de configuration et les secrets. Vous trouverez plus de détails sur les volumes éphémères de Kubernetes ici. La plupart des volumes éphémères (par exemple EmptyDir, ConfigMap, DownwardAPI, secret, hostpath) sont sauvegardés par des périphériques inscriptibles connectés localement (généralement le disque racine) ou par de la RAM. Il est donc important de choisir le volume hôte le plus rentable et le plus performant.

Utilisation des volumes EBS

Nous vous recommandons de commencer par gp3 comme volume racine de l'hôte. Il s'agit du dernier volume SSD à usage général proposé par Amazon EBS et propose également un prix inférieur (jusqu'à 20 %) par Go par rapport aux volumes gp2.

Utilisation d'Amazon EC2 Instance Stores

Les magasins d' EC2 instances Amazon fournissent un stockage temporaire au niveau des blocs pour vos EC2 instances. Le stockage fourni par les magasins d' EC2 instance est accessible via des disques physiquement connectés aux hôtes. Contrairement à Amazon EBS, vous ne pouvez attacher des volumes de stockage d'instance que lorsque l'instance est lancée, et ces volumes n'existent que pendant la durée de vie de l'instance. Ils ne peuvent pas être détachés et rattachés à d'autres instances. Pour en savoir plus sur les magasins d' EC2 instances Amazon, cliquez ici. Aucun frais supplémentaire n'est associé à un volume de stockage d'instance. Cela les rend (volumes de stockage d'instance) plus rentables que les EC2 instances générales comportant de grands volumes EBS.

Pour utiliser des volumes de stockage locaux dans Kubernetes, vous devez partitionner, configurer et formater les disques à l'aide des EC2 données utilisateur d'Amazon afin que les volumes puissent être montés selon les spécifications du pod HostPath. Vous pouvez également tirer parti du Local Persistent Volume Static Provisioner pour simplifier la gestion du stockage local. Le fournisseur statique de volumes persistants locaux vous permet d'accéder aux volumes de stockage d'instance locaux via l'interface Kubernetes PersistentVolumeClaim (PVC) standard. En outre, il fournira PersistentVolumes (PVs) qui contient des informations sur l'affinité des nœuds pour planifier les pods sur les bons nœuds. Bien qu'il utilise Kubernetes PersistentVolumes, les volumes de stockage d' EC2 instance sont de nature éphémère. Les données écrites sur des disques éphémères ne sont disponibles que pendant la durée de vie de l'instance. Lorsque l'instance est interrompue, les données le sont également. Veuillez consulter ce blog pour plus de détails.

N'oubliez pas que lorsque vous utilisez des volumes de stockage d' EC2 instance Amazon, la limite totale d'IOPS est partagée avec l'hôte et elle lie les pods à un hôte spécifique. Vous devez examiner attentivement vos exigences en matière de charge de travail avant d'adopter les volumes de stockage d' EC2 instance Amazon.

Volumes persistants

Kubernetes est généralement associé à l'exécution d'applications apatrides. Cependant, il existe des scénarios dans lesquels vous souhaiterez peut-être exécuter des microservices qui doivent préserver

des données ou des informations persistantes d'une demande à l'autre. Les bases de données sont un exemple courant de tels cas d'utilisation. Cependant, les capsules et les contenants ou processus qu'elles contiennent sont de nature éphémère. Pour conserver les données au-delà de la durée de vie d'un pod, vous pouvez PVs définir l'accès au stockage à un emplacement spécifique indépendant du pod. Les coûts associés dépendent PVs fortement du type de stockage utilisé et de la manière dont les applications l'utilisent.

Différents types d'options de stockage compatibles avec Kubernetes sur PVs Amazon EKS sont répertoriés ici. Les options de stockage décrites ci-dessous sont Amazon EBS, Amazon EFS, Amazon FSx for Lustre et Amazon FSx for NetApp ONTAP.

Volumes Amazon Elastic Block Store (EBS)

Les volumes Amazon EBS peuvent être utilisés sous forme de Kubernetes PVs pour fournir des volumes de stockage au niveau des blocs. Ils conviennent parfaitement aux bases de données qui reposent sur des lectures et écritures aléatoires et aux applications gourmandes en débit qui effectuent des lectures et des écritures longues et continues. Le pilote Amazon Elastic Block Store Container Storage Interface (CSI) permet aux clusters Amazon EKS de gérer le cycle de vie des volumes Amazon EBS pour les volumes persistants. L'interface de stockage de conteneurs permet et facilite l'interaction entre Kubernetes et un système de stockage. Lorsqu'un pilote CSI est déployé sur votre cluster EKS, vous pouvez accéder à ses fonctionnalités via les ressources de stockage natives de Kubernetes, telles que les volumes persistants (PVs), les réclamations de volumes persistants () et les classes de stockage (PVCs). SCs Ce lien fournit des exemples pratiques d'interaction avec les volumes Amazon EBS à l'aide du pilote Amazon EBS CSI.

Choisir le bon volume

Nous vous recommandons d'utiliser la dernière génération de stockage par blocs (gp3) car elle offre un juste équilibre entre prix et performances. Il vous permet également d'adapter les IOPS et le débit du volume indépendamment de la taille du volume sans avoir à fournir de capacité de stockage par blocs supplémentaire. Si vous utilisez actuellement des volumes gp2, nous vous recommandons vivement de migrer vers des volumes gp3. Le billet de blog Migration de clusters Amazon EKS de volumes EBS gp2 vers gp3 explique comment migrer de gp2 à gp3 sur des clusters Amazon EKS avec sauvegarde et restauration à l'aide de la fonctionnalité CSI Volume Snapshots, qui nécessite une interruption des applications.

Amazon EBS permet de modifier les caractéristiques du volume, telles que la taille du volume, les IOPS et le débit en ligne. Grâce à cette fonctionnalité, il est possible de migrer de gp2 vers gp3 sans interruption de service de l'application en utilisant soit des annotations PVC, comme décrit dans ce

blog, qui nécessite le pilote EBS CSI v1.19.0+, soit en commençant par Amazon EKS v1.31 et le pilote EBS CSI 1.35 en utilisant l'API décrite ici. VolumeAttributesClass

Lorsque vous avez des applications qui nécessitent des performances supérieures et des volumes supérieurs à ce que peut supporter un seul volume gp3, vous devriez envisager d'utiliser io2 block express. Ce type de stockage est idéal pour votre déploiement le plus important, le plus I/O intensif et le plus critique, tel que SAP HANA ou d'autres grandes bases de données nécessitant une faible latence. N'oubliez pas que les performances EBS d'une instance sont limitées par les limites de performances de l'instance, de sorte que toutes les instances ne prennent pas en charge les volumes io2 block express. Vous pouvez consulter les types d'instances pris en charge et d'autres considérations dans ce document.

Un seul volume gp3 peut prendre en charge jusqu'à 16 000 IOPS maximum, 1 000 débits maximum, 16 MiB/s TiB maximum. La dernière génération de volume SSD IOPS provisionné qui fournit jusqu'à 256 000 IOPS, 4 000 Mbits/s, un débit et 64 TiB.

Parmi ces options, vous devez adapter au mieux les performances et le coût de votre stockage aux besoins de vos applications.

Surveillez et optimisez au fil du temps

Il est important de comprendre les performances de base de votre application et de les surveiller pour les volumes sélectionnés afin de vérifier si elles répondent à vos attentes requirements/expectations ou si elles sont surapprovisionnées (par exemple, un scénario dans lequel les IOPS allouées ne sont pas pleinement utilisées).

Au lieu d'allouer un volume important dès le début, vous pouvez augmenter progressivement la taille du volume au fur et à mesure que vous accumulez des données. Vous pouvez redimensionner les volumes de manière dynamique à l'aide de la fonctionnalité de redimensionnement des volumes du pilote CSI Amazon Elastic Block Store (). aws-ebs-csi-driver N'oubliez pas que vous pouvez uniquement augmenter la taille du volume EBS.

Pour identifier et supprimer les volumes EBS en suspens, vous pouvez utiliser la catégorie d'optimisation des coûts d'AWS Trusted Advisor. Cette fonctionnalité vous permet d'identifier les volumes non attachés ou ceux dont l'activité d'écriture est très faible pendant un certain temps. Popeye, un outil open source natif du cloud et en lecture seule, analyse les clusters Kubernetes en direct et signale les problèmes potentiels liés aux ressources et aux configurations déployées. Par exemple, il peut rechercher les fichiers inutilisés PVs PVCs et vérifier s'ils sont liés ou s'il existe une erreur de montage du volume.

Pour en savoir plus sur la surveillance, veuillez consulter le guide d'observabilité de l'optimisation des coûts d'EKS.

Une autre option que vous pouvez envisager est celle des recommandations relatives aux volumes Amazon EBS d'AWS Compute Optimizer. Cet outil identifie automatiquement la configuration de volume optimale et le niveau de performance requis. Par exemple, il peut être utilisé pour des paramètres optimaux relatifs aux IOPS provisionnées, à la taille des volumes et aux types de volumes EBS en fonction de l'utilisation maximale des 14 derniers jours. Il quantifie également les économies mensuelles potentielles découlant de ses recommandations. Vous pouvez consulter ce blog pour plus de détails.

Politique de conservation des sauvegardes

Vous pouvez sauvegarder les données de vos volumes Amazon EBS en prenant des point-in-time instantanés. Le pilote Amazon EBS CSI prend en charge les instantanés de volume. Vous pouvez apprendre à créer un instantané et à restaurer un PV EBS en suivant les étapes décrites ici.

Les instantanés suivants sont des sauvegardes incrémentielles, ce qui signifie que seuls les blocs de l'appareil qui ont changé après votre dernier instantané sont enregistrés. Cela réduit le temps nécessaire pour créer l'instantané, ainsi que les coûts de stockage en ne dupliquant pas les données. Cependant, l'augmentation du nombre d'anciens instantanés EBS sans politique de conservation appropriée peut entraîner des coûts imprévus lors d'une exploitation à grande échelle. Si vous sauvegardez directement des volumes Amazon EBS via l'API AWS, vous pouvez tirer parti d'Amazon Data Lifecycle Manager (DLM) qui fournit une solution de gestion du cycle de vie automatisée et basée sur des règles pour les instantanés Amazon Elastic Block Store (EBS) et les Amazon Machine Images () soutenues par EBS. AMIs La console facilite l'automatisation de la création, de la conservation et de la suppression des instantanés EBS et. AMIs

> **Note**
>
> Il n'existe actuellement aucun moyen d'utiliser Amazon DLM via le pilote Amazon EBS CSI.

Dans un environnement Kubernetes, vous pouvez utiliser un outil open source appelé Velero pour sauvegarder vos volumes persistants EBS. Vous pouvez définir un indicateur TTL lors de la planification de la tâche pour faire expirer les sauvegardes. Voici un guide de Velero à titre d'exemple.

Amazon Elastic File System (EFS)

Amazon Elastic File System (EFS) est un système de fichiers entièrement élastique sans serveur qui vous permet de partager des données de fichiers à l'aide d'une interface de système de fichiers et d'une sémantique de système de fichiers standard pour un large éventail de charges de travail et d'applications. Wordpress et Drupal, les outils de développement tels que JIRA et Git, les systèmes de bloc-notes partagés tels que Jupyter et les répertoires personnels sont des exemples de charges de travail et d'applications.

L'un des principaux avantages d'Amazon EFS est qu'il peut être monté par plusieurs conteneurs répartis sur plusieurs nœuds et plusieurs zones de disponibilité. Un autre avantage est que vous ne payez que pour le stockage que vous utilisez. Les systèmes de fichiers EFS s'agrandissent et se réduisent automatiquement au fur et à mesure que vous ajoutez et supprimez des fichiers, ce qui élimine le besoin de planification des capacités.

Pour utiliser Amazon EFS dans Kubernetes, vous devez utiliser le pilote Amazon Elastic File System Container Storage Interface (CSI),. aws-efs-csi-driver Actuellement, le conducteur peut créer des points d'accès de manière dynamique. Cependant, le système de fichiers Amazon EFS doit d'abord être provisionné et fourni en entrée dans le paramètre de classe de stockage Kubernetes.

Choisir la bonne classe de stockage EFS

Amazon EFS propose quatre classes de stockage.

Deux classes de stockage standard :

- Norme Amazon EFS
- Accès standard et peu fréquent à Amazon EFS (EFS Standard-IA)

Deux classes de stockage à zone unique :

- Zone Amazon EFS One
- Accès peu fréquent à Amazon EFS One Zone (EFS One Zone-IA)

Les classes de stockage à accès peu fréquent (IA) sont optimisées en termes de coûts pour les fichiers auxquels on n'accède pas tous les jours. Grâce à la gestion du cycle de vie d'Amazon EFS, vous pouvez déplacer des fichiers auxquels vous n'avez pas accédé pendant la durée de la politique de cycle de vie (7, 14, 30, 60 ou 90 jours) vers les classes de stockage IA, ce qui peut réduire les

coûts de stockage jusqu'à 92 % par rapport aux classes de stockage EFS Standard et EFS One Zone respectivement.

Avec EFS Intelligent-Tiering, la gestion du cycle de vie surveille les modèles d'accès de votre système de fichiers et déplace automatiquement les fichiers vers la classe de stockage la plus optimale.

> **Note**
>
> aws-efs-csi-driver n'a actuellement aucun contrôle sur la modification des classes de stockage, la gestion du cycle de vie ou la hiérarchisation intelligente. Ils doivent être configurés manuellement dans la console AWS ou via l'EFS APIs.

> **Note**
>
> aws-efs-csi-driver n'est pas compatible avec les images de conteneur basées sur Windows.

> **Note**
>
> Il existe un problème de mémoire connu lorsque vol-metrics-opt-in(pour émettre des métriques de volume) est activée en raison de la DiskUsagefonction qui consomme une quantité de mémoire proportionnelle à la taille de votre système de fichiers. Actuellement, nous recommandons de désactiver l'option `-- vol-metrics-opt-in `sur les systèmes de fichiers volumineux pour éviter de consommer trop de mémoire. Voici un lien vers un problème sur GitHub pour plus de détails.

Amazon FSx pour Lustre

Lustre est un système de fichiers parallèle à hautes performances couramment utilisé dans les charges de travail nécessitant un débit de latence pouvant atteindre des centaines de millisecondes GB/s et inférieures à la milliseconde par opération. Il est utilisé pour des scénarios tels que la formation à l'apprentissage automatique, la modélisation financière, le HPC et le traitement vidéo. Amazon FSx for Lustre fournit un stockage partagé entièrement géré alliant évolutivité et performance, parfaitement intégré à Amazon S3.

Vous pouvez utiliser des volumes de stockage persistants Kubernetes soutenus par FSx for Lustre à l'aide du pilote CSI FSx for Lustre d'Amazon EKS ou de votre cluster Kubernetes autogéré sur AWS. Consultez la documentation Amazon EKS pour plus de détails et des exemples.

Lien vers Amazon S3

Il est recommandé de lier un référentiel de données à long terme hautement durable résidant sur Amazon S3 à votre système de fichiers FSx for Lustre. Une fois liés, les grands ensembles de données sont chargés latéralement selon les besoins depuis Amazon S3 vers les systèmes FSx de fichiers Lustre. Vous pouvez également réexécuter vos analyses et vos résultats dans S3, puis supprimer votre système de fichiers Lustre.

Choisir les bonnes options de déploiement et de stockage

FSx for Lustre propose différentes options de déploiement. La première option s'appelle scratch et ne réplique pas les données, tandis que la seconde option est appelée persistante, ce qui, comme son nom l'indique, conserve les données.

La première option (scratch) peut être utilisée pour réduire le coût du traitement temporaire des données à court terme. L'option de déploiement persistant est conçue pour un stockage à long terme qui réplique automatiquement les données au sein d'une zone de disponibilité AWS. Il prend également en charge le stockage SSD et HDD.

Vous pouvez configurer le type de déploiement souhaité dans les paramètres de StorageClass Kubernetes du système de fichiers FSx for lustre. Voici un lien qui fournit des exemples de modèles.

> **Note**
>
> Pour les charges de travail sensibles à la latence ou les charges de travail nécessitant les plus hauts niveaux d'IOP/débit, vous devez choisir le stockage SSD. Pour les charges de travail axées sur le débit qui ne sont pas sensibles à la latence, vous devez choisir le stockage sur disque dur.

Activer la compression des données

Vous pouvez également activer la compression des données sur votre système de fichiers en spécifiant « LZ4 » comme type de compression de données. Une fois activé, tous les fichiers nouvellement écrits seront automatiquement compressés FSx pour Lustre avant d'être écrits sur le

disque et décompressés lors de leur lecture. LZ4 l'algorithme de compression de données est sans perte, de sorte que les données d'origine peuvent être entièrement reconstruites à partir des données compressées.

Vous pouvez configurer le type de compression des données comme indiqué dans LZ4 les paramètres du système de fichiers StorageClass Kubernetes FSx for Lustre. La compression est désactivée lorsque la valeur est définie sur NONE, qui est la valeur par défaut. Ce lien fournit des exemples de modèles.

> **Note**
>
> Amazon FSx for Lustre n'est pas compatible avec les images de conteneurs basées sur Windows.

Amazon FSx pour NetApp ONTAP

Amazon FSx for NetApp ONTAP est un espace de stockage partagé entièrement géré basé sur le système NetApp de fichiers ONTAP. FSx for ONTAP fournit un stockage de fichiers partagé riche en fonctionnalités, rapide et flexible, largement accessible à partir d'instances de calcul Linux, Windows et macOS exécutées sur AWS ou sur site.

Amazon FSx for NetApp ONTAP prend en charge deux niveaux de stockage : 1 niveau principal et 2 niveaux de pool de capacité.

Le niveau principal est un niveau basé sur des SSD hautes performances et provisionné pour les données actives sensibles à la latence. Le niveau du pool de capacité entièrement élastique est optimisé en termes de coûts pour les données rarement consultées, évolue automatiquement au fur et à mesure que les données sont hiérarchisées et offre une capacité pratiquement illimitée en pétaoctets. Vous pouvez activer la compression et la déduplication des données sur le stockage du pool de capacité et réduire davantage la capacité de stockage consommée par vos données. NetAppLa FabricPool fonctionnalité native basée sur des règles surveille en permanence les modèles d'accès aux données, transférant automatiquement les données de manière bidirectionnelle entre les niveaux de stockage afin d'optimiser les performances et les coûts.

NetAppAstra Trident fournit une orchestration dynamique du stockage à l'aide d'un pilote CSI qui permet aux clusters Amazon EKS de gérer le cycle de vie des volumes persistants soutenus PVs par Amazon FSx pour NetApp les systèmes de fichiers ONTAP. Pour commencer, consultez la section Utiliser Astra Trident avec Amazon FSx pour NetApp ONTAP dans la documentation d'Astra Trident.

Autres considérations

Minimiser la taille de l'image du conteneur

Une fois les conteneurs déployés, les images des conteneurs sont mises en cache sur l'hôte sous forme de couches multiples. En réduisant la taille des images, la quantité de stockage requise sur l'hôte peut être réduite.

En utilisant dès le départ des images de base allégées, telles que des images à gratter ou des images de conteneur sans distribution (qui contiennent uniquement votre application et ses dépendances d'exécution), vous pouvez réduire les coûts de stockage en plus d'autres avantages connexes tels que la réduction de la surface d'attaque et des temps d'extraction des images plus courts.

Vous devriez également envisager d'utiliser des outils open source, tels que Slim.ai, qui fournit un moyen simple et sécurisé de créer des images minimales.

Plusieurs couches de packages, d'outils, de dépendances d'applications et de bibliothèques peuvent facilement augmenter la taille de l'image du conteneur. En utilisant des versions en plusieurs étapes, vous pouvez copier des artefacts de manière sélective d'une étape à l'autre, en excluant tout ce qui n'est pas nécessaire de l'image finale. Vous pouvez consulter d'autres bonnes pratiques en matière de création d'images ici.

Une autre chose à prendre en compte est la durée de conservation des images mises en cache. Vous souhaiterez peut-être nettoyer les images périmées du cache d'images lorsqu'une certaine quantité de disque est utilisée. Cela vous permettra de vous assurer que vous disposez de suffisamment d'espace pour le fonctionnement de l'hôte. Par défaut, le kubelet collecte les déchets sur les images non utilisées toutes les cinq minutes et sur les conteneurs non utilisés toutes les minutes.

Pour configurer les options pour la collecte de conteneurs et d'images inutilisés, ajustez le kubelet à l'aide d'un fichier de configuration et modifiez les paramètres liés à la collecte de déchets à l'aide du type de `KubeletConfiguration`ressource.

Vous pouvez en savoir plus à ce sujet dans la documentation de Kubernetes.

Observabilité

Introduction

Les outils d'observabilité vous aident à détecter, corriger et étudier efficacement vos charges de travail. Le coût des données de télémétrie augmente naturellement à mesure que vous utilisez EKS. Il peut parfois être difficile de trouver un équilibre entre vos besoins opérationnels et de mesurer ce qui compte pour votre entreprise et de maîtriser les coûts d'observabilité. Ce guide se concentre sur les stratégies d'optimisation des coûts pour les trois piliers de l'observabilité : les logs, les métriques et les traces. Chacune de ces meilleures pratiques peut être appliquée indépendamment pour répondre aux objectifs d'optimisation de votre organisation.

Journalisation

La journalisation joue un rôle essentiel dans la surveillance et le dépannage des applications de votre cluster. Plusieurs stratégies peuvent être utilisées pour optimiser les coûts d'exploitation forestière. Les meilleures pratiques répertoriées ci-dessous incluent l'examen de vos politiques de conservation des journaux afin de mettre en œuvre des contrôles précis sur la durée de conservation des données de journal, l'envoi des données de journal vers différentes options de stockage en fonction de leur importance et l'utilisation du filtrage des journaux pour affiner les types de messages de journal stockés. La gestion efficace de la télémétrie des journaux peut permettre à vos environnements de réaliser des économies.

Plan de contrôle EKS

Optimisez les journaux de votre plan de contrôle

Le plan de contrôle Kubernetes est un ensemble de composants qui gèrent les clusters et ces composants envoient différents types d'informations sous forme de flux de journaux à un groupe de journaux sur Amazon. CloudWatch Bien que l'activation de tous les types de journaux du plan de contrôle présente des avantages, vous devez connaître les informations contenues dans chaque journal et les coûts associés au stockage de toutes les données télémétriques des journaux. Les frais d'ingestion et de stockage CloudWatch des données Logs standard vous sont facturés pour les journaux envoyés à Amazon CloudWatch Logs depuis vos clusters. Avant de les activer, déterminez si chaque flux de log est nécessaire.

Par exemple, dans les clusters non liés à la production, activez de manière sélective des types de journaux spécifiques, tels que les journaux du serveur API, uniquement à des fins d'analyse, puis

désactivez-les par la suite. Toutefois, pour les clusters de production, où vous ne pourrez peut-être pas reproduire les événements et où la résolution des problèmes nécessite davantage d'informations de journal, vous pouvez activer tous les types de journaux. D'autres détails sur la mise en œuvre de l'optimisation des coûts du plan de contrôle figurent dans ce billet de blog.

Diffuser les journaux vers S3

Une autre bonne pratique d'optimisation des coûts consiste à diffuser les journaux du plan de contrôle vers S3 via CloudWatch des abonnements Logs. L'utilisation CloudWatch des abonnements Logs vous permet de transférer les journaux de manière sélective vers S3, ce qui permet un stockage à long terme plus rentable que la conservation des journaux indéfiniment CloudWatch. Par exemple, pour les clusters de production, vous pouvez créer un groupe de journaux critique et tirer parti des abonnements pour diffuser ces journaux vers S3 après 15 jours. Cela vous permettra d'accéder rapidement aux journaux à des fins d'analyse, mais également de réaliser des économies en transférant les journaux vers un espace de stockage plus rentable.

> **Important**
>
> Depuis le 5 septembre 2023, les journaux EKS sont classés comme des journaux vendus dans Amazon Logs. CloudWatch Les Vended Logs sont des journaux de service AWS spécifiques publiés nativement par les services AWS pour le compte du client et disponibles à des prix réduits sur le volume. Consultez la page de CloudWatch tarification d'Amazon pour en savoir plus sur la tarification de Vended Logs.

Plan de données EKS

Conservation de journal

La politique CloudWatch de conservation par défaut d'Amazon consiste à conserver les journaux indéfiniment et à ne jamais expirer, ce qui entraîne des coûts de stockage applicables à votre région AWS. Afin de réduire les coûts de stockage, vous pouvez personnaliser la politique de rétention pour chaque groupe de journaux en fonction de vos exigences en matière de charge de travail.

Dans un environnement de développement, une longue période de conservation peut ne pas être nécessaire. Toutefois, dans un environnement de production, vous pouvez définir une politique de rétention plus longue pour répondre aux exigences de dépannage, de conformité et de planification des capacités. Par exemple, si vous utilisez une application de commerce électronique pendant la haute saison des fêtes, si le système est soumis à une charge de travail plus importante et que des

problèmes peuvent survenir qui ne sont pas immédiatement perceptibles, vous devez définir une durée de conservation des journaux plus longue pour un dépannage détaillé et une analyse post-événement.

Vous pouvez configurer vos périodes de rétention dans la CloudWatch console AWS ou dans l'API AWS avec une durée comprise entre 1 jour et 10 ans en fonction de chaque groupe de journaux. Le fait de disposer d'une période de conservation flexible permet de réduire les coûts de stockage des journaux, tout en conservant les journaux critiques.

Options de stockage des journaux

Le stockage est un facteur important des coûts d'observabilité. Il est donc essentiel d'optimiser votre stratégie de stockage des journaux. Vos stratégies doivent correspondre aux exigences de vos charges de travail tout en préservant les performances et l'évolutivité. L'une des stratégies pour réduire les coûts de stockage des journaux consiste à tirer parti des compartiments AWS S3 et de ses différents niveaux de stockage.

Transférer les journaux directement vers S3

Envisagez de transférer les journaux moins critiques, tels que les environnements de développement, directement vers S3 plutôt que vers Cloudwatch. Cela peut avoir un impact immédiat sur les coûts de stockage des journaux. Une option consiste à transférer les journaux directement vers S3 à l'aide de Fluentbit. Vous définissez cela dans la `[OUTPUT]` section, la destination où les journaux des conteneurs FluentBit sont transmis à des fins de conservation. Consultez les paramètres de configuration supplémentaires ici.

```
[OUTPUT]
        Name eks_to_s3
        Match application.*
        bucket $S3_BUCKET name
        region us-east-2
        store_dir /var/log/fluentbit
        total_file_size 30M
        upload_timeout 3m
```

Transférer les journaux CloudWatch uniquement à des fins d'analyse à court terme

Pour les journaux plus critiques, tels que les environnements de production dans lesquels vous devrez peut-être effectuer une analyse immédiate des données, pensez à les transférer vers CloudWatch. Vous définissez cela dans la `[OUTPUT]` section, la destination où les journaux

des conteneurs FluentBit sont transmis à des fins de conservation. Consultez les paramètres de configuration supplémentaires ici.

```
[OUTPUT]
        Name eks_to_cloudwatch_logs
        Match application.*
        region us-east-2
        log_group_name fluent-bit-cloudwatch
        log_stream_prefix from-fluent-bit-
        auto_create_group On
```

Toutefois, cela n'aura pas d'effet immédiat sur vos économies. Pour réaliser des économies supplémentaires, vous devrez exporter ces journaux vers Amazon S3.

Exporter vers Amazon S3 depuis CloudWatch

Pour stocker les CloudWatch journaux Amazon à long terme, nous vous recommandons d'exporter vos CloudWatch journaux Amazon EKS vers Amazon Simple Storage Service (Amazon S3). Vous pouvez transférer les journaux vers le compartiment Amazon S3 en créant une tâche d'exportation via la console ou l'API. Une fois que vous l'avez fait, Amazon S3 propose de nombreuses options pour réduire encore les coûts. Vous pouvez définir vos propres règles de cycle de vie Amazon S3 pour déplacer vos journaux vers une classe de stockage adaptée à vos besoins, ou tirer parti de la classe de stockage Amazon S3 Intelligent-Tiering pour qu'AWS déplace automatiquement les données vers un stockage à long terme en fonction de votre modèle d'utilisation. Veuillez consulter ce blog pour plus de détails. Par exemple, pour votre environnement de production, les journaux sont conservés CloudWatch pendant plus de 30 jours avant d'être exportés vers le compartiment Amazon S3. Vous pouvez ensuite utiliser Amazon Athena pour interroger les données du compartiment Amazon S3 si vous devez consulter les journaux ultérieurement.

Réduire les niveaux de journalisation

Pratiquez la journalisation sélective pour votre application. Vos applications et vos nœuds génèrent des journaux par défaut. Pour les journaux de vos applications, ajustez les niveaux de journalisation en fonction de la criticité de la charge de travail et de l'environnement. Par exemple, l'application Java ci-dessous génère des `INFO` journaux, ce qui est la configuration d'application par défaut typique et, selon le code, peut entraîner un volume élevé de données de journal.

```
import org.apache.log4j.*;

public class LogClass {
```

```
   private static org.apache.log4j.Logger log = Logger.getLogger(LogClass.class);

public static void main(String[] args) {
      log.setLevel(Level.INFO);

   log.debug("This is a DEBUG message, check this out!");
   log.info("This is an INFO message, nothing to see here!");
   log.warn("This is a WARN message, investigate this!");
   log.error("This is an ERROR message, check this out!");
   log.fatal("This is a FATAL message, investigate this!");    } }
```

Dans un environnement de développement, modifiez votre niveau de journalisation sur`DEBUG`, car cela peut vous aider à résoudre les problèmes ou à détecter les problèmes potentiels avant qu'ils ne soient mis en production.

```
      log.setLevel(Level.DEBUG);
```

Dans un environnement de production, pensez à modifier votre niveau de journalisation à `ERROR` ou`FATAL`. Cela ne produira le journal que lorsque votre application contient des erreurs, ce qui réduit le volume de sortie du journal et vous aide à vous concentrer sur les données importantes concernant le statut de votre application.

```
      log.setLevel(Level.ERROR);
```

Vous pouvez affiner les niveaux de journalisation des différents composants Kubernetes. Par exemple, si vous utilisez Bottlerocket comme système d'exploitation EKS Node, certains paramètres de configuration vous permettent d'ajuster le niveau du journal des processus kubelet. Vous trouverez ci-dessous un extrait de ce paramètre de configuration. Notez le niveau de journalisation par défaut de 2 qui ajuste la verbosité de journalisation du processus. `kubelet`

```
[settings.kubernetes]
log-level = "2"
image-gc-high-threshold-percent = "85"
image-gc-low-threshold-percent = "80"
```

Pour un environnement de développement, vous pouvez définir un niveau de journalisation supérieur à 2 afin de visualiser des événements supplémentaires, ce qui est utile pour le débogage. Pour un environnement de production, vous pouvez définir le niveau sur 0 afin de n'afficher que les événements critiques.

Tirez parti des filtres

Lorsque vous utilisez une configuration EKS Fluentbit par défaut pour envoyer des journaux de conteneur à Cloudwatch, FluentBit capture et envoie TOUS les journaux de conteneurs d'applications enrichis de métadonnées Kubernetes à Cloudwatch, comme indiqué dans le bloc de configuration ci-dessous. `[INPUT]`

```
 [INPUT]
     Name                tail
     Tag                 application.*
     Exclude_Path        /var/log/containers/cloudwatch-agent*, /var/log/containers/
fluent-bit*, /var/log/containers/aws-node*, /var/log/containers/kube-proxy*
     Path                /var/log/containers/*.log
     Docker_Mode         On
     Docker_Mode_Flush   5
     Docker_Mode_Parser  container_firstline
     Parser              docker
     DB                  /var/fluent-bit/state/flb_container.db
     Mem_Buf_Limit       50MB
     Skip_Long_Lines     On
     Refresh_Interval    10
     Rotate_Wait         30
     storage.type        filesystem
     Read_from_Head      ${READ_FROM_HEAD}
```

La `[INPUT]` section ci-dessus traite de l'ingestion de tous les journaux du conteneur. Cela peut générer une grande quantité de données qui ne sont peut-être pas nécessaires. Le filtrage de ces données peut réduire la quantité de données de journal envoyées, réduisant CloudWatch ainsi vos coûts. Vous pouvez appliquer un filtre à vos journaux avant qu'ils ne soient renvoyés vers CloudWatch. Fluentbit le définit dans la `[FILTER]` section. Par exemple, empêcher l'ajout des métadonnées Kubernetes aux événements du journal peut réduire le volume de votre journal.

```
    [FILTER]
        Name                nest
        Match               application.*
        Operation           lift
        Nested_under        kubernetes
        Add_prefix          Kube.

    [FILTER]
        Name                modify
```

```
        Match               application.*
        Remove              Kube.<Metadata_1>
        Remove              Kube.<Metadata_2>
        Remove              Kube.<Metadata_3>

    [FILTER]
        Name                nest
        Match               application.*
        Operation           nest
        Wildcard            Kube.*
        Nested_under        kubernetes
        Remove_prefix       Kube.
```

Métriques

Les métriques fournissent des informations précieuses concernant les performances de votre système. En consolidant tous les indicateurs relatifs au système ou aux ressources disponibles dans un emplacement centralisé, vous pouvez comparer et analyser les données de performance. Cette approche centralisée vous permet de prendre des décisions stratégiques plus éclairées, telles que l'augmentation ou la réduction des ressources. En outre, les indicateurs jouent un rôle crucial dans l'évaluation de l'état des ressources, vous permettant de prendre des mesures proactives si nécessaire. En général, les coûts d'observabilité augmentent en fonction de la collecte et de la conservation des données de télémétrie. Vous trouverez ci-dessous quelques stratégies que vous pouvez mettre en œuvre pour réduire le coût de la télémétrie métrique : collecter uniquement les mesures importantes, réduire la cardinalité de vos données de télémétrie et affiner la granularité de votre collecte de données de télémétrie.

Surveillez ce qui compte et collectez uniquement ce dont vous avez besoin

La première stratégie de réduction des coûts consiste à réduire le nombre de mesures que vous collectez et, par conséquent, à réduire les coûts de rétention.

1. Commencez par revenir à vos exigences et/ou à celles de vos parties prenantes afin de déterminer les indicateurs les plus importants. Les indicateurs de réussite sont différents pour chacun ! Sachez à quoi ressemble une belle apparence et mesurez en conséquence.
2. Envisagez d'étudier en profondeur les charges de travail que vous soutenez et d'identifier ses indicateurs de performance clés (KPIs), également appelés « signaux d'or ». Elles devraient être conformes aux exigences des entreprises et des parties prenantes. Le calcul SLIs et SLOs l' SLAs utilisation d'Amazon CloudWatch et de Metric Math sont essentiels pour gérer la fiabilité

des services. Suivez les meilleures pratiques décrites dans ce guide pour surveiller et maintenir efficacement les performances de votre environnement EKS.

3. Passez ensuite en revue les différentes couches de l'infrastructure pour connecter et corréler le cluster EKS, le nœud et les indicateurs d'infrastructure supplémentaires à votre charge KPIs de travail. Stockez vos indicateurs commerciaux et opérationnels dans un système dans lequel vous pouvez les corréler et tirer des conclusions en fonction des impacts observés sur les deux.
4. EKS expose les métriques du plan de contrôle, du cluster kube-state-metrics, des pods et des nœuds. La pertinence de tous ces indicateurs dépend de vos besoins, mais il est probable que vous n'ayez pas besoin de tous les indicateurs sur les différentes couches. Vous pouvez utiliser ce guide des indicateurs essentiels d'EKS comme référence pour surveiller l'état général d'un cluster EKS et vos charges de travail.

Voici un exemple de configuration de Prometheus Scrape dans lequel nous utilisons le pour ne conserver que les métriques `relabel_config` de Kubelet et pour supprimer toutes les métriques `metric_relabel_config` de conteneur.

```
  kubernetes_sd_configs:
  - role: endpoints
    namespaces:
      names:
      - kube-system
  bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
  tls_config:
    insecure_skip_verify: true
  relabel_configs:
  - source_labels: [__meta_kubernetes_service_label_k8s_app]
    regex: kubelet
    action: keep

  metric_relabel_configs:
  - source_labels: [__name__]
    regex: container_(network_tcp_usage_total|network_udp_usage_total|tasks_state|
cpu_load_average_10s)
    action: drop
```

Réduire la cardinalité le cas échéant

La cardinalité fait référence au caractère unique des valeurs des données en combinaison avec leurs dimensions (par exemple, les étiquettes Prometheus) pour un ensemble de mesures spécifique. Les

métriques de cardinalité élevée ont de nombreuses dimensions et chaque combinaison de métriques de dimensions présente un caractère unique plus élevé. Une cardinalité plus élevée entraîne une augmentation de la taille des données de télémétrie métrique et des besoins de stockage, ce qui augmente les coûts.

Dans l'exemple de cardinalité élevée ci-dessous, nous voyons que la métrique, la latence, comporte des dimensions, requestiD, customerID et service et que chaque dimension possède de nombreuses valeurs uniques. La cardinalité est la mesure de la combinaison du nombre de valeurs possibles par dimension. Dans Prometheus, chaque ensemble de dimensions/étiquettes uniques est considéré comme une nouvelle métrique. Une cardinalité élevée signifie donc un plus grand nombre de métriques.

Dans les environnements EKS comportant de nombreuses métriques et dimensions/étiquettes par métrique (cluster, espace de noms, service, pod, conteneur, etc.), la cardinalité a tendance à augmenter. Afin d'optimiser les coûts, considérez soigneusement la cardinalité des indicateurs que vous collectez. Par exemple, si vous agrégez une métrique spécifique à des fins de visualisation au niveau du cluster, vous pouvez supprimer les étiquettes supplémentaires situées sur une couche inférieure, telles que l'étiquette de l'espace de noms.

Afin d'identifier les métriques de cardinalité élevées dans Prometheus, vous pouvez exécuter la requête PROMQL suivante pour déterminer quelles cibles de scrape ont le plus grand nombre de métriques (cardinalité) :

```
topk_max(5, max_over_time(scrape_samples_scraped[1h]))
```

et la requête PROMQL suivante peut vous aider à déterminer quelles cibles de scrape présentent les taux de désabonnement (combien de nouvelles séries de métriques ont été créées au cours d'un scrape donné) les plus élevés :

```
topk_max(5, max_over_time(scrape_series_added[1h]))
```

Si vous utilisez grafana, vous pouvez utiliser l'outil Mimir de Grafana Lab pour analyser vos tableaux de bord Grafana et les règles de Prometheus afin d'identifier les métriques de haute cardinalité non utilisées. Suivez ce guide pour savoir comment utiliser les `mimirtool analyze prometheus` commandes `mimirtool analyze` et pour identifier les métriques actives qui ne sont pas référencées dans vos tableaux de bord.

Tenez compte de la granularité métrique

La collecte de métriques à une granularité plus élevée, par exemple chaque seconde au lieu de chaque minute, peut avoir un impact important sur le volume de données télémétriques collectées et stockées, ce qui augmente les coûts. Déterminez des intervalles de collecte raisonnables ou mesurez des intervalles de collecte entre une granularité suffisante pour détecter les problèmes transitoires et un niveau suffisamment bas pour être rentable. Réduisez la granularité des métriques utilisées pour la planification des capacités et l'analyse de fenêtres temporelles étendues.

Vous trouverez ci-dessous un extrait de la configuration par défaut d'AWS Distro for Opentelemetry (ADOT) EKS Addon Collector.

> **Important**
>
> l'intervalle global de grattage de Prometheus est fixé à 15 s. Cet intervalle de récupération peut être augmenté, ce qui entraîne une diminution de la quantité de données métriques collectées dans Prometheus.

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: my-collector-amp

...

config: |
    extensions:
      sigv4auth:
        region: "+++<YOUR_AWS_REGION>+++" service: "aps"+++</YOUR_AWS_REGION>+++

 receivers:
   #
   # Scrape configuration for the Prometheus Receiver
   # This is the same configuration used when Prometheus is installed using the
 community Helm chart
   #
   prometheus:
     config:
       global:   scrape_interval: 15s
         scrape_timeout: 10s
```

Tracing

Le principal coût associé au traçage provient de la génération du stockage des traces. Avec le traçage, l'objectif est de recueillir suffisamment de données pour diagnostiquer et comprendre les aspects liés aux performances. Cependant, comme les coûts des traces de X-Ray sont basés sur les données transmises à X-Ray, l'effacement des traces une fois qu'elles ont été transmises ne réduira pas vos coûts. Examinons les moyens de réduire les coûts de suivi tout en conservant les données nécessaires pour effectuer une analyse appropriée.

Appliquer les règles d'échantillonnage

La fréquence d'échantillonnage des rayons X est prudente par défaut. Définissez des règles d'échantillonnage qui vous permettent de contrôler la quantité de données que vous collectez. Cela améliorera l'efficacité des performances tout en réduisant les coûts. En diminuant le taux d'échantillonnage, vous pouvez collecter des traces à partir de la demande uniquement pour ce dont vos charges de travail ont besoin, tout en maintenant une structure de coûts inférieure.

Par exemple, vous avez une application Java dont vous souhaitez déboguer les traces de toutes les demandes pour une route problématique.

Configuration via le SDK pour charger les règles d'échantillonnage à partir d'un document JSON

```
{
"version": 2,
  "rules": [
    {
"description": "debug-eks",
      "host": "*",
      "http_method": "PUT",
      "url_path": "/history/*",
      "fixed_target": 0,
      "rate": 1,
      "service_type": "debug-eks"
    }
  ],
  "default": {
"fixed_target": 1,
    "rate": 0.1
  }
}
```

Par le biais de la console

Appliquer Tail Sampling avec AWS Distro for OpenTelemetry (ADOT)

ADOT Tail Sampling vous permet de contrôler le volume de traces ingérées dans le service. Cependant, Tail Sampling vous permet de définir les politiques d'échantillonnage une fois que toutes les étapes de la demande ont été terminées plutôt qu'au début. Cela limite encore davantage la quantité de données brutes transférées CloudWatch, réduisant ainsi les coûts.

Par exemple, si vous collectez 1 % du trafic vers une page de destination et 10 % des demandes vers une page de paiement, cela peut vous laisser 300 traces sur une période de 30 minutes. Avec une règle ADOT Tail Sampling qui filtre les erreurs spécifiques, vous pourriez vous retrouver avec 200 traces, ce qui réduit le nombre de traces stockées.

```
processors:
  groupbytrace:
    wait_duration: 10s
    num_traces: 300
    tail_sampling:
    decision_wait: 1s # This value should be smaller than wait_duration
    policies:
      - ..... # Applicable policies**
  batch/tracesampling:
    timeout: 0s # No need to wait more since this will happen in previous processors
    send_batch_max_size: 8196 # This will still allow us to limit the size of the
 batches sent to subsequent exporters

service:
  pipelines:
    traces/tailsampling:
      receivers: [otlp]
      processors: [groupbytrace, tail_sampling, batch/tracesampling]
      exporters: [awsxray]
```

Tirez parti des options de stockage Amazon S3

Vous devez utiliser le compartiment AWS S3 et ses différentes classes de stockage pour stocker les traces. Exportez les traces vers S3 avant l'expiration de la période de conservation. Utilisez les règles du cycle de vie d'Amazon S3 pour déplacer les données de suivi vers la classe de stockage qui répond à vos besoins.

Par exemple, si vous avez des traces datant de 90 jours, Amazon S3 Intelligent-Tiering peut automatiquement déplacer les données vers un stockage à long terme en fonction de vos habitudes d'utilisation. Vous pouvez utiliser Amazon Athena pour interroger les données dans Amazon S3 si vous devez vous référer aux traces ultérieurement. Cela peut encore réduire vos coûts de suivi distribué.

Ressources supplémentaires :

- Guide des meilleures pratiques en matière d'observabilité
- Collecte de statistiques sur les meilleures pratiques
- AWS re:Invent 2022 - Bonnes pratiques en matière d'observabilité chez Amazon () COP343
- AWS re:Invent 2022 - Observabilité : meilleures pratiques pour les applications modernes () COP344

Meilleures pratiques pour Windows

Ce guide fournit des conseils sur l'exécution de conteneurs et de nœuds Windows.

Rubriques

- Amazon EKS a optimisé la gestion des AMI Windows
- Configurer GMSA pour les pods et les conteneurs Windows
- Renforcement des nœuds de travail Windows
- Numérisation d'images de conteneurs
- Version et licence de Windows Server
- Logging
- Contrôle
- Réseau Windows
- Éviter les erreurs OOM
- Appliquer des correctifs aux serveurs et aux conteneurs Windows
- Exécution de charges de travail hétérogènes
- Contextes de sécurité des modules
- Options de stockage permanent
- Renforcement des images de conteneurs Windows

Amazon EKS a optimisé la gestion des AMI Windows

Les versions optimisées pour Windows Amazon EKS AMIs sont basées sur Windows Server 2019 et Windows Server 2022. Elles sont configurées pour servir d'image de base pour les nœuds Amazon EKS. Par défaut, ils AMIs incluent les composants suivants :

- kubelet
- kube-proxy
- Authenticator AWS IAM pour Kubernetes
- csi-proxy
- containerd

Vous pouvez récupérer par programmation l'identifiant Amazon Machine Image (AMI) pour Amazon EKS optimisé AMIs en interrogeant l'API AWS Systems Manager Parameter Store. Ce paramètre vous évite d'avoir à rechercher manuellement l'AMI optimisée pour Amazon EKS IDs. Pour plus d'informations sur l'API Systems Manager Parameter Store, consultez GetParameter. Votre compte utilisateur doit disposer de l'autorisation ssm : GetParameter IAM pour récupérer les métadonnées AMI optimisées pour Amazon EKS.

L'exemple suivant récupère l'ID d'AMI de la dernière AMI optimisée pour Amazon EKS pour Windows Server 2019 LTSC Core. Le numéro de version indiqué dans le nom de l'AMI correspond à la version de Kubernetes correspondante pour laquelle elle est préparée.

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-2019-
English-Core-EKS_Optimized-1.21/image_id --region us-east-1 --query "Parameter.Value"
 --output text
```

Exemple de sortie :

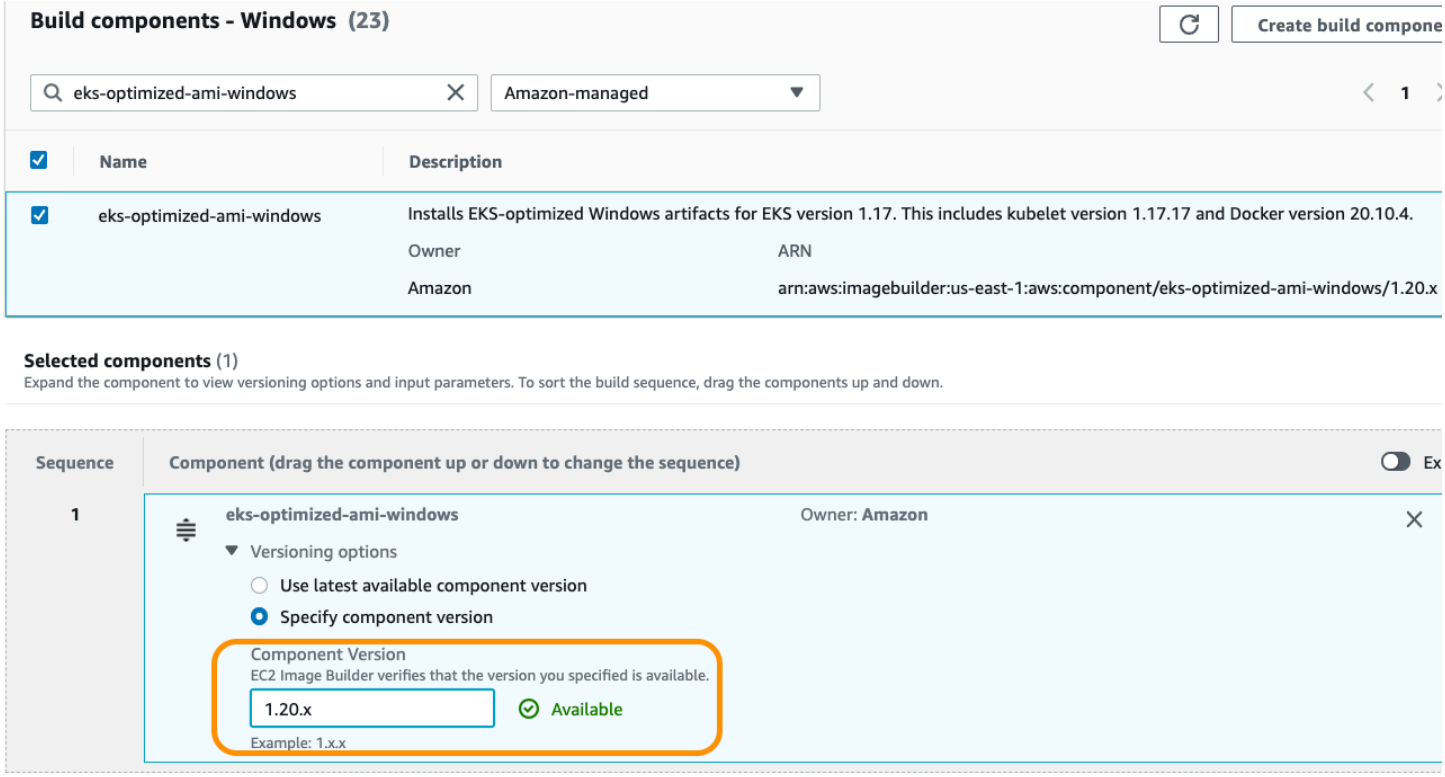
```
ami-09770b3eec4552d4e
```

Gestion de votre propre AMI Windows optimisée pour Amazon EKS

Une étape essentielle vers les environnements de production consiste à maintenir la même version d'AMI Windows optimisée pour Amazon EKS et de Kubelet sur l'ensemble du cluster Amazon EKS.

L'utilisation de la même version sur l'ensemble du cluster Amazon EKS réduit le temps consacré au dépannage et améliore la cohérence du cluster. Amazon EC2 Image Builder permet de créer et de gérer des fenêtres personnalisées optimisées pour Amazon EKS AMIs à utiliser dans un cluster Amazon EKS.

Utilisez Amazon EC2 Image Builder pour choisir entre les versions de Windows Server, les dates de sortie de l'AMI AWS Windows Server et la version de construction du système d' and/or exploitation. L'étape des composants de génération vous permet de choisir entre les artefacts Windows optimisés EKS existants et les versions de kubelet. Pour plus d'informations : https://docs.aws.amazon.com/eks/ latest/userguide/eks - custom-ami-windows .html



REMARQUE : Avant de sélectionner une image de base, consultez la section Version et licence de Windows Server pour obtenir des informations importantes concernant les mises à jour des canaux de publication.

Configuration d'un lancement plus rapide pour une optimisation EKS personnalisée AMIs

Lorsque vous utilisez une AMI personnalisée optimisée pour Windows Amazon EKS, les nœuds de travail Windows peuvent être lancés jusqu'à 65 % plus rapidement en activant la fonctionnalité de lancement rapide. Cette fonctionnalité gère un ensemble de snapshots préprovisionnés dans lesquels la spécialisation Sysprep est spécialisée, les étapes Windows Out of Box Experience (OOBE) et les redémarrages requis sont déjà terminés. Ces instantanés sont ensuite utilisés lors des lancements suivants, ce qui réduit le temps nécessaire pour étendre ou remplacer les nœuds. Fast Launch ne peut être activé que pour AMIs vous par le biais de la console EC2 ou de la CLI AWS et le nombre de snapshots conservés est configurable.

REMARQUE : Fast Launch n'est pas compatible avec l'AMI optimisée EKS par défaut fournie par Amazon. Créez une AMI personnalisée comme indiqué ci-dessus avant de tenter de l'activer.

Pour plus d'informations : AWS Windows AMIs : configurez votre AMI pour un lancement plus rapide

Mise en cache des couches de base Windows sur mesure AMIs

Les images de conteneur Windows sont plus grandes que leurs homologues Linux. Si vous exécutez une application conteneurisée basée sur .NET Framework, la taille moyenne des images est d'environ 8,24 Go. Lors de la planification du pod, l'image du conteneur doit être entièrement extraite et extraite sur le disque avant que le pod n'atteigne le statut Running.

Au cours de ce processus, le moteur d'exécution du conteneur (containerd) extrait et extrait l'image complète du conteneur sur le disque. L'opération pull est un processus parallèle, ce qui signifie que le moteur d'exécution du conteneur extrait les couches d'image du conteneur en parallèle. En revanche, l'opération d'extraction s'effectue dans un processus séquentiel et elle est I/O intensive. De ce fait, l'image du conteneur peut prendre plus de 8 minutes pour être complètement extraite et prête à être utilisée par le moteur d'exécution du conteneur (containerd). Par conséquent, le temps de démarrage du pod peut prendre plusieurs minutes.

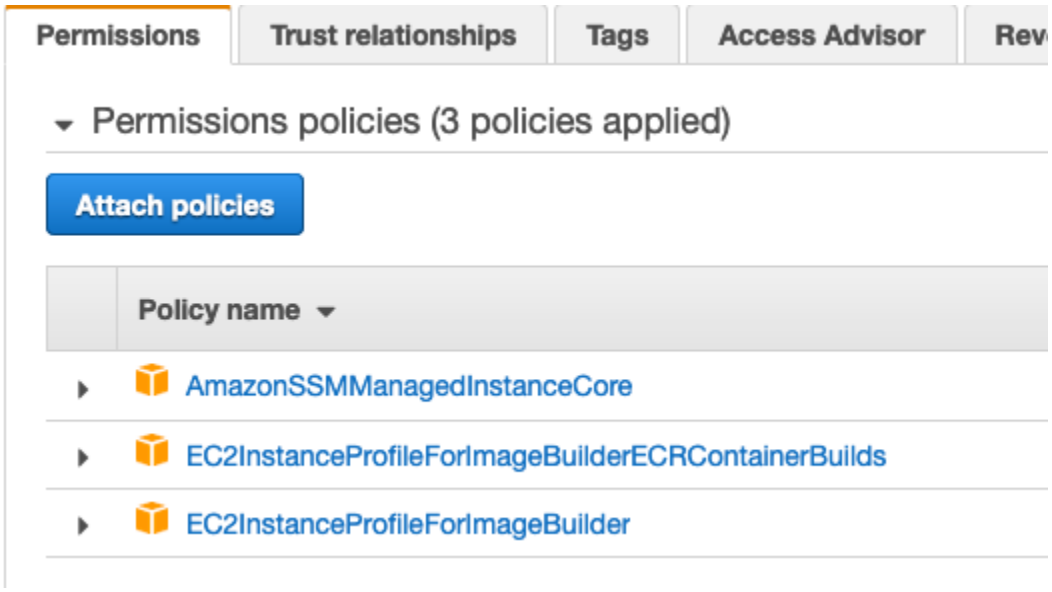
Comme indiqué dans la rubrique Appliquer des correctifs à Windows Server and Container, il existe une option permettant de créer une AMI personnalisée avec EKS. Pendant la préparation de l'AMI, vous pouvez ajouter un composant EC2 Image Builder supplémentaire pour extraire localement toutes les images de conteneur Windows nécessaires, puis générer l'AMI. Cette stratégie réduira considérablement le temps pendant lequel un pod atteint le statut En cours d'exécution.

Sur Amazon EC2 Image Builder, créez un composant pour télécharger les images nécessaires et joignez-le à la recette d'image. L'exemple suivant extrait une image spécifique d'un référentiel ECR.

```
name: ContainerdPull
description: This component pulls the necessary containers images for a cache strategy.
schemaVersion: 1.0

phases:
  - name: build
    steps:
      - name: containerdpull
        action: ExecutePowerShell
        inputs:
          commands:
            - Set-ExecutionPolicy Unrestricted -Force
            - (Get-ECRLoginCommand).Password | docker login --username AWS --password-
stdin 111000111000.dkr.ecr.us-east-1.amazonaws.com
            - ctr image pull mcr.microsoft.com/dotnet/framework/aspnet:latest
```

```
            - ctr image pull 111000111000.dkr.ecr.us-east-1.amazonaws.com/
myappcontainerimage:latest
```

Pour vous assurer que le composant suivant fonctionne comme prévu, vérifiez si le rôle IAM utilisé par EC2 Image builder (EC2InstanceProfileForImageBuilder) possède les politiques associées :



Billet de blog

Dans le billet de blog suivant, vous découvrirez étape par étape comment implémenter une stratégie de mise en cache pour les fenêtres AMIs Amazon EKS personnalisées :

Accélération des délais de lancement des conteneurs Windows grâce au générateur d'images EC2 et à la stratégie de cache d'images

Configurer GMSA pour les pods et les conteneurs Windows

Qu'est-ce qu'un compte GMSA

Les applications Windows telles que les applications .NET utilisent souvent Active Directory comme fournisseur d'identité, en authorization/authentication utilisant le protocole NTLM ou Kerberos.

Un serveur d'applications permettant d'échanger des tickets Kerberos avec Active Directory doit être joint à un domaine. Les conteneurs Windows ne prennent pas en charge les jointures de domaines et cela n'aurait pas beaucoup de sens car les conteneurs sont des ressources éphémères, ce qui alourdit le pool RID d'Active Directory.

Toutefois, les administrateurs peuvent tirer parti des comptes Active Directory GMSA pour négocier une authentification Windows pour des ressources telles que les conteneurs Windows, le NLB et les batteries de serveurs.

Conteneur Windows et cas d'utilisation du GMSA

Les applications qui s'appuient sur l'authentification Windows et s'exécutent en tant que conteneurs Windows bénéficient de la GMSA, car le nœud Windows est utilisé pour échanger le ticket Kerberos au nom du conteneur. Deux options sont disponibles pour configurer le nœud de travail Windows afin de prendre en charge l'intégration de la GMSA :

1 - Nœuds de travail Windows joints à un domaine

Dans cette configuration, le nœud de travail Windows est joint au domaine Active Directory, et le compte AD Computer des nœuds de travail Windows est utilisé pour s'authentifier auprès d'Active Directory et récupérer l'identité GMSA à utiliser avec le pod.

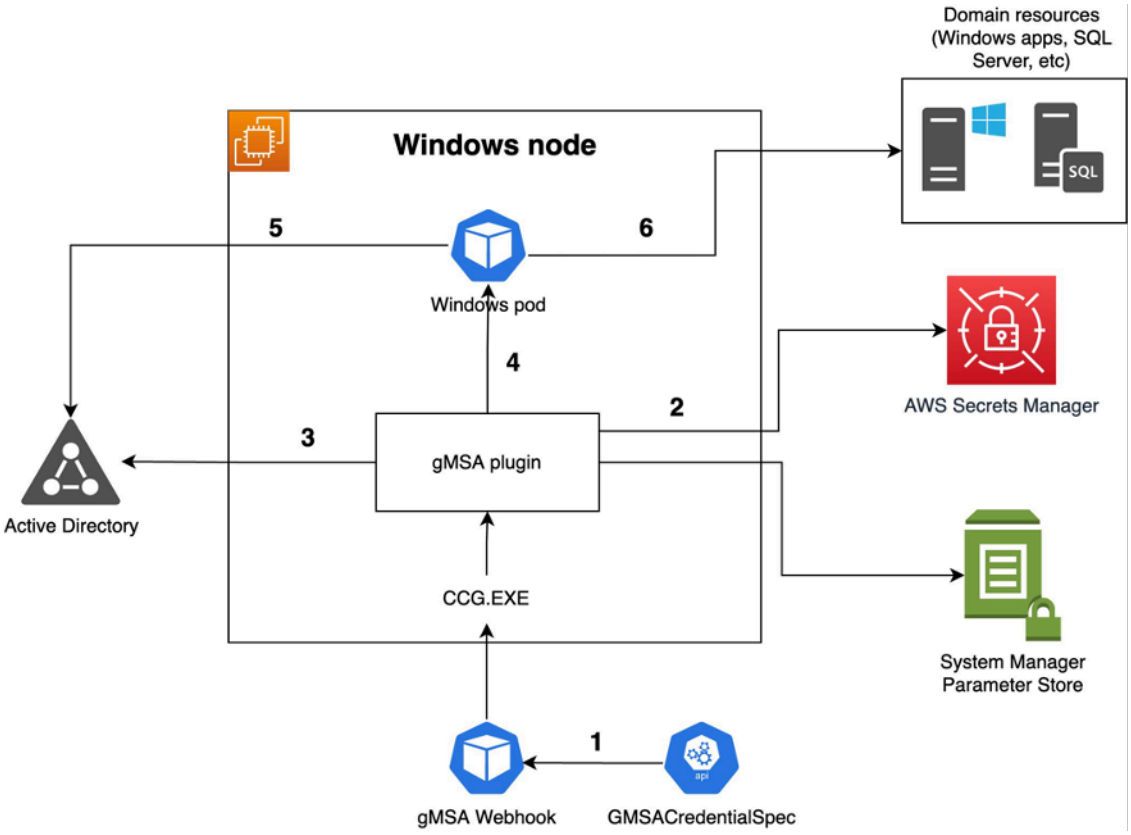
Dans le cadre de l'approche jointe à un domaine, vous pouvez facilement gérer et renforcer vos nœuds de travail Windows à l'aide d'Active Directory existant GPOs ; toutefois, cela génère des surcharges opérationnelles et des retards lors de l'adhésion des nœuds de travail Windows au cluster Kubernetes, car cela nécessite des redémarrages supplémentaires lors du démarrage des nœuds et le nettoyage du garage Active Directory une fois que le cluster Kubernetes a mis fin aux nœuds.

Dans le billet de blog suivant, vous trouverez des informations détaillées step-by-step sur la mise en œuvre de l'approche du nœud de travail Windows joint à un domaine :

Authentification Windows sur les modules Windows Amazon EKS

2 - Nœuds de travail Windows sans domaine

Dans cette configuration, le nœud de travail Windows n'est pas joint au domaine Active Directory et une identité « portable » (utilisateur/mot de passe) est utilisée pour s'authentifier auprès d'Active Directory et récupérer l'identité GMSA à utiliser avec le pod.



L'identité portable est celle d'un utilisateur Active Directory ; l'identité (utilisateur/mot de passe) est stockée sur AWS Secrets Manager ou AWS System Manager Parameter Store, et un plugin développé par AWS appelé ccg_plugin sera utilisé pour récupérer cette identité auprès d'AWS Secrets Manager ou d'AWS System Manager Parameter Store et la transmettre à containerd pour récupérer l'identité gMSA et la rendre disponible pour le pod.

Dans cette approche sans domaine, vous pouvez bénéficier de l'absence d'interaction avec Active Directory lors du démarrage du nœud de travail Windows lorsque vous utilisez GMSA et de la réduction de la charge opérationnelle pour les administrateurs Active Directory.

Dans le billet de blog suivant, vous trouverez des informations détaillées step-by-step sur la mise en œuvre de l'approche du nœud de travail Windows sans domaine :

Authentification Windows sans domaine pour les modules Windows Amazon EKS

Remarque importante

Bien que le pod puisse utiliser un compte GMSA, il est également nécessaire de configurer l'application ou le service en conséquence pour prendre en charge l'authentification Windows. Par

exemple, pour configurer Microsoft IIS pour prendre en charge l'authentification Windows, vous devez le préparer via dockerfile :

```
RUN Install-WindowsFeature -Name Web-Windows-Auth -IncludeAllSubFeature
RUN Import-Module WebAdministration; Set-ItemProperty 'IIS:\AppPools\SiteName' -name
 processModel.identityType -value 2
RUN Import-Module WebAdministration; Set-WebConfigurationProperty -Filter '/
system.webServer/security/authentication/anonymousAuthentication' -Name Enabled -Value
 False -PSPath 'IIS:\' -Location 'SiteName'
RUN Import-Module WebAdministration; Set-WebConfigurationProperty -Filter '/
system.webServer/security/authentication/windowsAuthentication' -Name Enabled -Value
 True -PSPath 'IIS:\' -Location 'SiteName'
```

Renforcement des nœuds de travail Windows

Le renforcement du système d'exploitation est une combinaison de configuration du système d'exploitation, de correctifs et de suppression de packages logiciels inutiles, dans le but de verrouiller un système et de réduire la surface d'attaque. Il est recommandé de préparer votre propre AMI Windows optimisée pour EKS avec les configurations renforcées requises par votre entreprise.

AWS fournit chaque mois une nouvelle AMI Windows optimisée pour EKS contenant les derniers correctifs de sécurité pour Windows Server. Cependant, il incombe toujours à l'utilisateur de renforcer son AMI en appliquant les configurations de système d'exploitation nécessaires, qu'il utilise des groupes de nœuds autogérés ou gérés.

Microsoft propose une gamme d'outils tels que Microsoft Security Compliance Toolkit et Security Baselines, qui vous aident à renforcer vos capacités en fonction de vos besoins en matière de politiques de sécurité. Des benchmarks CIS sont également disponibles et devraient être implémentés au-dessus d'une AMI Windows optimisée Amazon EKS pour les environnements de production.

Réduction de la surface d'attaque avec Windows Server Core

Windows Server Core est une option d'installation minimale disponible dans le cadre de l'AMI Windows optimisée EKS. Le déploiement de Windows Server Core présente plusieurs avantages. Tout d'abord, son encombrement disque est relativement faible, soit 6 Go sur Server Core contre 10 Go sur Windows Server avec expérience de bureau. Ensuite, sa surface d'attaque est plus petite en raison de sa base de code plus petite et de sa disponibilité APIs.

AWS fournit AMIs chaque mois à ses clients de nouveaux Windows optimisés pour Amazon EKS, contenant les derniers correctifs de sécurité Microsoft, quelle que soit la version prise en charge par Amazon EKS. La meilleure pratique consiste à remplacer les nœuds de travail Windows par de nouveaux nœuds basés sur la dernière AMI optimisée pour Amazon EKS. Tout nœud fonctionnant pendant plus de 45 jours sans mise à jour en place ou remplacement de nœud ne respecte pas les meilleures pratiques en matière de sécurité.

Éviter les connexions RDP

Le protocole RDP (Remote Desktop Protocol) est un protocole de connexion développé par Microsoft pour fournir aux utilisateurs une interface graphique leur permettant de se connecter à un autre ordinateur Windows via un réseau.

Il est recommandé de traiter vos nœuds de travail Windows comme s'il s'agissait d'hôtes éphémères. Cela signifie qu'il n'y a aucune connexion de gestion, aucune mise à jour et aucun dépannage. Toute modification ou mise à jour doit être implémentée sous la forme d'une nouvelle AMI personnalisée et remplacée par la mise à jour d'un groupe Auto Scaling. Consultez la section Appliquer des correctifs aux serveurs et aux conteneurs Windows et la gestion optimisée des AMI Windows par Amazon EKS.

Désactivez les connexions RDP sur les nœuds Windows pendant le déploiement en transmettant la valeur false à la propriété ssh, comme dans l'exemple ci-dessous :

```
nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
```

Si l'accès au nœud Windows est nécessaire, utilisez le gestionnaire de session AWS System Manager pour établir une PowerShell session sécurisée via la console AWS et l'agent SSM. Pour découvrir comment implémenter la solution, regardez Accès sécurisé aux instances Windows à l'aide d'AWS Systems Manager Session Manager.

Pour utiliser le gestionnaire de session System Manager, une politique IAM supplémentaire doit être appliquée au rôle IAM utilisé pour lancer le nœud de travail Windows. Voici un exemple où l'Amazon SSMManaged InstanceCore est spécifié dans le manifeste du `eksctl` cluster :

```
 nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
  iam:
    attachPolicyARNs:
      - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
      - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
      - arn:aws:iam::aws:policy/ElasticLoadBalancingFullAccess
      - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
      - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
```

Amazon Inspector

> Amazon Inspector est un service d'évaluation automatique de la sécurité qui permet d'améliorer la sécurité et la conformité des applications déployées sur AWS. Amazon Inspector évalue automatiquement les applications pour détecter leur exposition, leurs vulnérabilités et les écarts par rapport aux meilleures pratiques. Après avoir effectué une évaluation, Amazon Inspector produit une liste détaillée des résultats de sécurité classés par niveau de gravité. Ces résultats peuvent être examinés directement ou dans le cadre de rapports d'évaluation détaillés disponibles via la console ou l'API Amazon Inspector.

Amazon Inspector peut être utilisé pour exécuter l'évaluation CIS Benchmark sur le nœud de travail Windows et il peut être installé sur un Windows Server Core en effectuant les tâches suivantes :

1. Téléchargez le fichier .exe suivant : https://inspector-agent.amazonaws.com/windows/ installer/latest/AWSAgentInstall .exe
2. Transférez l'agent vers le nœud de travail Windows.
3. Exécutez la commande suivante PowerShell pour installer l'agent Amazon Inspector : `.\AWSAgentInstall.exe /install`

Vous trouverez ci-dessous le résultat après le premier essai. Comme vous pouvez le constater, il a généré des résultats basés sur la base de données CVE. Vous pouvez l'utiliser pour renforcer vos nœuds de travail ou créer une AMI basée sur les configurations renforcées.

Target name	EKS-Windows-Nodes
Template name	EKS-Windows
Start	Yesterday at 4:16 PM (GMT-5) (18 hours ago)
End	Yesterday at 5:20 PM (GMT-5) (17 hours ago)
Status	Analysis complete
Rules package	Common Vulnerabilities and Exposures-1.1
AWS agent ID	i-
Auto scaling group	win2004uscentraltime-NodeGroup-1NOC3JPXYWAHG
Finding	Instance i-0b is vulnerable to CVE-2021-1655
Severity	High
Description	Windows CSC Service Elevation of Privilege Vulnerability This CVE ID is unique from CVE-2021-1652, CVE-2021-1653, CVE-2021-1654, CVE-2021-1659, CVE-2021-1688, CVE-2021-1693.
Recommendation	Use your Operating System's update feature to update package (see the CVE details). For more information see https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-1655

Pour plus d'informations sur Amazon Inspector, notamment sur la façon d'installer les agents Amazon Inspector, de configurer l'évaluation CIS Benchmark et de générer des rapports, regardez la vidéo Améliorer la sécurité et la conformité des charges de travail Windows avec Amazon Inspector.

Amazon GuardDuty

> Amazon GuardDuty est un service de détection des menaces qui surveille en permanence les activités malveillantes et les comportements non autorisés afin de protéger vos comptes AWS, vos charges de travail et les données stockées dans Amazon S3. Avec le cloud, la collecte et l'agrégation des activités du compte et du réseau sont simplifiées, mais l'analyse continue des données du journal des événements pour détecter les menaces potentielles peut prendre du temps pour les équipes de sécurité.

En utilisant Amazon, GuardDuty vous avez une visibilité sur les activités malveillantes visant les nœuds de travail Windows, telles que les attaques par force brute RDP et Port Probe.

Regardez la GuardDuty vidéo sur la détection des menaces pour les charges de travail Windows à l'aide d'Amazon pour découvrir comment implémenter et exécuter des benchmarks CIS sur une AMI Windows EKS optimisée

Sécurité dans Amazon EC2 pour Windows

Découvrez les meilleures pratiques de sécurité pour les instances Windows Amazon EC2 afin de mettre en œuvre des contrôles de sécurité à chaque couche.

Numérisation d'images de conteneurs

La numérisation d'images est une fonctionnalité d'évaluation automatique des vulnérabilités qui permet d'améliorer la sécurité des images de conteneur de votre application en les scannant pour détecter un large éventail de vulnérabilités du système d'exploitation.

À l'heure actuelle, l'Amazon Elastic Container Registry (ECR) est uniquement en mesure de scanner l'image d'un conteneur Linux pour détecter les vulnérabilités. Cependant, il existe des outils tiers qui peuvent être intégrés à un CI/CD pipeline existant pour la numérisation d'images de conteneurs Windows.

- Ancrage
- PaloAlto Nuage Prisma
- Trend Micro - Contrôle intelligent approfondi de la sécurité

Pour en savoir plus sur la façon d'intégrer ces solutions à Amazon Elastic Container Repository (ECR), consultez :

- Anchore, numérisation d'images sur Amazon Elastic Container Registry (ECR)
- PaloAlto, numérisation d'images sur Amazon Elastic Container Registry (ECR)
- TrendMicro, numérisation d'images sur Amazon Elastic Container Registry (ECR)

Version et licence de Windows Server

Version Windows Server

Une AMI Windows optimisée pour Amazon EKS est basée sur les éditions Windows Server 2019 et 2022 Datacenter sur le Long-Term Servicing Channel (LTSC). La version Datacenter ne limite pas le nombre de conteneurs exécutés sur un nœud de travail. Pour plus d'informations : https://docs.microsoft.com/en-us/ virtualization/windowscontainers/about/faq

Canal de maintenance à long terme (LTSC)

Anciennement appelé « Long-Term Servicing Branch », il s'agit du modèle de version que vous connaissez déjà, dans le cadre duquel une nouvelle version majeure de Windows Server est publiée tous les 2 à 3 ans. Les utilisateurs ont droit à 5 ans de support standard et à 5 ans de support étendu.

Licences

Lorsque vous lancez une instance Amazon EC2 avec une AMI basée sur Windows Server, Amazon prend en charge les coûts de licence et la conformité des licences pour vous.

Logging

Les applications conteneurisées dirigent généralement les journaux des applications vers STDOUT. Le moteur d'exécution du conteneur intercepte ces journaux et en fait quelque chose, généralement en écrivant dans un fichier. L'emplacement de stockage de ces fichiers dépend de l'environnement d'exécution et de la configuration du conteneur.

L'une des différences fondamentales avec les pods Windows est qu'ils ne génèrent pas de STDOUT. Vous pouvez exécuter LogMonitorpour récupérer l'ETW (suivi des événements pour Windows), les journaux d'événements Windows et d'autres journaux spécifiques à l'application à partir de conteneurs Windows en cours d'exécution et de canaux de sortie de journal formatés vers STDOUT. Ces journaux peuvent ensuite être diffusés en utilisant fluent-bit ou fluentd vers la destination de votre choix, telle qu'Amazon. CloudWatch

Le mécanisme de collecte des STDOUT/STDERR journaux récupère les journaux des pods Kubernetes. A DaemonSetest un moyen courant de collecter des journaux à partir de conteneurs. Il vous permet de gérer le journal routing/filtering/enrichment indépendamment de l'application. Un fluentd DaemonSet peut être utilisé pour diffuser ces journaux et tout autre journal généré par une application vers un agrégateur de journaux souhaité.

Des informations plus détaillées sur le streaming de journaux depuis des charges de travail Windows vers des applications CloudWatch sont expliquées ici

Recommandations en matière de journalisation

Les meilleures pratiques générales de journalisation ne sont pas différentes lors de l'exploitation de charges de travail Windows dans Kubernetes.

- Enregistrez toujours les entrées de journal structurées (JSON/SYSLOG), ce qui facilite la gestion des entrées de journal car il existe de nombreux analyseurs préécrits pour de tels formats structurés.
- Centralisation des journaux : des conteneurs de journalisation dédiés peuvent être utilisés spécifiquement pour recueillir et transférer les messages de journal de tous les conteneurs vers une destination

- Limitez la verbosité du journal, sauf lors du débogage. La verbosité exerce un stress important sur l'infrastructure forestière et des événements importants peuvent être perdus dans le bruit.
- Enregistrez toujours les informations de l'application ainsi que l'identifiant de la transaction ou de la demande à des fins de traçabilité. Les objets Kubernetes ne portent pas le nom de l'application. Par exemple, le nom d'un pod `windows-twryrqyw` peut ne pas avoir de sens lors du débogage des journaux. Cela facilite la traçabilité et le dépannage des applications avec vos journaux agrégés.

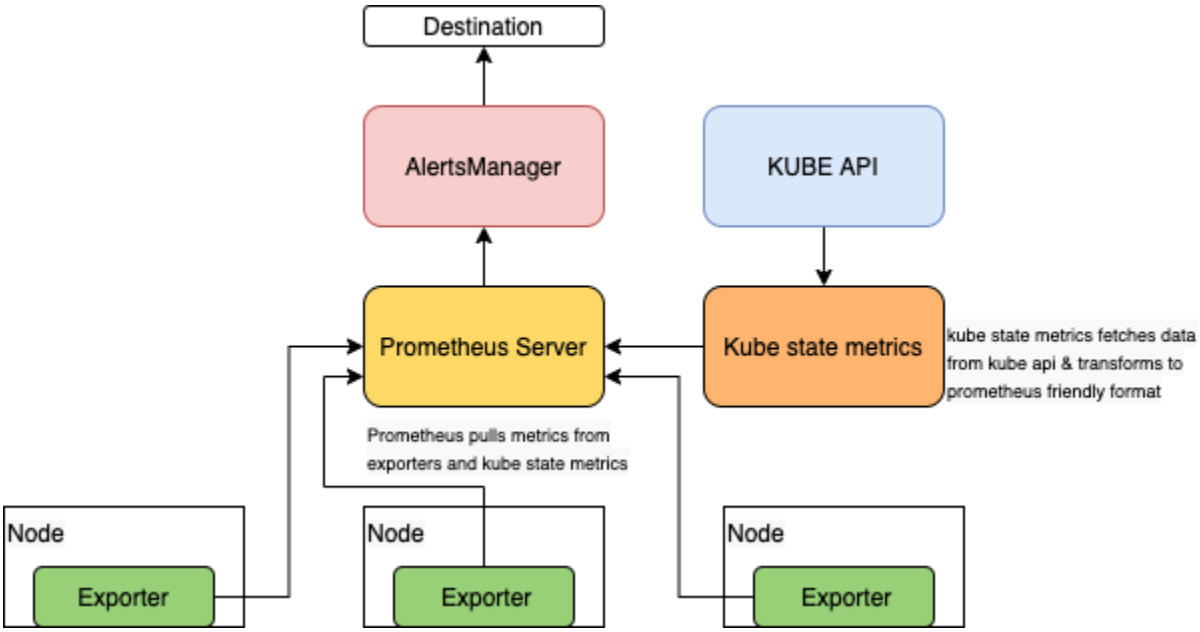
  La façon dont vous générez ces transaction/correlation identifiants dépend de la structure de programmation. Mais un modèle très courant consiste à utiliser un aspect/intercepteur de journalisation, qui peut utiliser le MDC (Mapped Diagnostic Context) pour injecter un transaction/correlation identifiant unique à chaque demande entrante, comme suit :

```
import org.slf4j.MDC;
import java.util.UUID;
Class LoggingAspect { //interceptor

    @Before(value = "execution(* *.*(..))")
    func before(...) {
        transactionId = generateTransactionId();
        MDC.put(CORRELATION_ID, transactionId);
    }

    func generateTransactionId() {
        return UUID.randomUUID().toString();
    }
}
```

Contrôle

Prometheus, un projet CNCF gradué, est de loin le système de surveillance le plus populaire avec une intégration native dans Kubernetes. Prometheus collecte des métriques relatives aux conteneurs, aux pods, aux nœuds et aux clusters. De plus, Prometheus vous permet AlertsManager de programmer des alertes pour vous avertir en cas de problème dans votre cluster. Prometheus stocke les données métriques sous forme de séries chronologiques identifiées par le nom de la métrique et par des paires. key/value Prometheus inclut un moyen d'effectuer des requêtes à l'aide d'un langage appelé ProMQL, abréviation de Prometheus Query Language.

L'architecture de haut niveau de la collecte de métriques Prometheus est illustrée ci-dessous :



Prometheus utilise un mécanisme d'extraction et extrait les métriques des cibles à l'aide d'exportateurs et de l'API Kubernetes à l'aide des métriques kube state. Cela signifie que les applications et les services doivent exposer un point de terminaison HTTP (S) contenant des métriques au format Prometheus. Prometheus, conformément à sa configuration, extraira ensuite périodiquement les métriques de ces points de terminaison HTTP (S).

Un exportateur vous permet de consommer des métriques tierces sous forme de métriques au format Prometheus. Un exportateur Prometheus est généralement déployé sur chaque nœud. Pour une liste complète des exportateurs, veuillez consulter les exportateurs de Prometheus. Bien que l'exportateur de nœuds soit adapté à l'exportation du matériel hôte et des métriques du système d'exploitation pour les nœuds Linux, il ne fonctionnera pas pour les nœuds Windows.

Dans un cluster EKS à nœuds mixtes avec des nœuds Windows, lorsque vous utilisez le diagramme de barre Prometheus stable, vous verrez des pods défaillants sur les nœuds Windows, car cet exportateur n'est pas conçu pour Windows. Vous devrez traiter le pool de travailleurs Windows séparément et installer l'exportateur Windows sur le groupe de nœuds de travail Windows.

Pour configurer la surveillance Prometheus pour les nœuds Windows, vous devez télécharger et installer l'exportateur WMI sur le serveur Windows lui-même, puis configurer les cibles dans la configuration Scrape du fichier de configuration Prometheus. La page des versions fournit tous les programmes d'installation .msi disponibles, avec les ensembles de fonctionnalités et les corrections de bogues correspondants. Le programme d'installation configurera le windows_exporter en tant que service Windows et créera une exception dans le pare-feu Windows. Si le programme d'installation

est exécuté sans aucun paramètre, l'exportateur s'exécutera avec les paramètres par défaut pour les collecteurs, les ports, etc. activés.

Vous pouvez consulter la section sur les meilleures pratiques de planification de ce guide qui suggère d'utiliser taints/tolerations ou RuntimeClass de déployer de manière sélective l'exportateur de nœuds uniquement sur les nœuds Linux, tandis que l'exportateur Windows est installé sur les nœuds Windows lorsque vous démarrez le nœud ou à l'aide d'un outil de gestion de configuration de votre choix (par exemple chef, Ansible, SSM, etc.).

Notez que, contrairement aux nœuds Linux où l'exportateur de nœuds est installé en tant que daemonset, sur les nœuds Windows, l'exportateur WMI est installé sur l'hôte lui-même. L'exportateur exportera des métriques telles que l'utilisation du processeur, de la mémoire et du disque I/O et pourra également être utilisé pour surveiller les sites et applications IIS, les interfaces réseau et les services.

Le windows_exporter exposera toutes les métriques des collecteurs activés par défaut. Il s'agit de la méthode recommandée pour collecter des métriques afin d'éviter les erreurs. Toutefois, pour une utilisation avancée, une liste facultative de collecteurs peut être transmise au windows_exporter pour filtrer les métriques. Le paramètre collect [], dans la configuration de Prometheus, vous permet de le faire.

Les étapes d'installation par défaut pour Windows incluent le téléchargement et le démarrage de l'exportateur en tant que service pendant le processus de démarrage avec des arguments, tels que les collecteurs que vous souhaitez filtrer.

```
> Powershell Invoke-WebRequest https://github.com/prometheus-community/
windows_exporter/releases/download/v0.13.0/windows_exporter-0.13.0-amd64.msi -OutFile
 <DOWNLOADPATH>

> msiexec /i <DOWNLOADPATH>
 ENABLED_COLLECTORS="cpu,cs,logical_disk,net,os,system,container,memory"
```

Par défaut, les métriques peuvent être supprimées sur le point de terminaison /metrics sur le port 9182. À ce stade, Prometheus peut consommer les métriques en ajoutant le scrape_config suivant à la configuration de Prometheus

```
scrape_configs:
    - job_name: "prometheus"
      static_configs:
        - targets: ['localhost:9090']
```

```
    ...
    - job_name: "wmi_exporter"
      scrape_interval: 10s
      static_configs:
        - targets: ['<windows-node1-ip>:9182', '<windows-node2-ip>:9182', ...]
```

La configuration de Prometheus est rechargée à l'aide de

```
> ps aux | grep prometheus
> kill HUP <PID>
```

Une méthode meilleure et recommandée pour ajouter des cibles consiste à utiliser une définition de ressource personnalisée appelée [ServiceMonitor, qui fait partie de l'opérateur Prometheus], qui fournit la définition d' ServiceMonitor un objet et un contrôleur qui activera la configuration Prometheus requise et créera automatiquement ServiceMonitors la configuration Prometheus requise.

Le ServiceMonitor, qui spécifie de manière déclarative comment les groupes de services Kubernetes doivent être surveillés, est utilisé pour définir une application dont vous souhaitez extraire des métriques dans Kubernetes. ServiceMonitor Nous y indiquons les étiquettes Kubernetes que l'opérateur peut utiliser pour identifier le service Kubernetes, qui à son tour identifie les pods que nous souhaitons surveiller.

Pour en tirer parti ServiceMonitor, créez un objet Endpoint pointant vers des cibles Windows spécifiques, un service headless et un ServiceMontor pour les nœuds Windows.

```
apiVersion: v1
kind: Endpoints
metadata:
  labels:
    k8s-app: wmiexporter
  name: wmiexporter
  namespace: kube-system
subsets:
- addresses:
  - ip: NODE-ONE-IP
    targetRef:
      kind: Node
      name: NODE-ONE-NAME
  - ip: NODE-TWO-IP
    targetRef:
```

```
      kind: Node
      name: NODE-TWO-NAME
  - ip: NODE-THREE-IP
    targetRef:
      kind: Node
      name: NODE-THREE-NAME
  ports:
  - name: http-metrics
    port: 9182
    protocol: TCP

---
apiVersion: v1
kind: Service ##Headless Service
metadata:
  labels:
    k8s-app: wmiexporter
  name: wmiexporter
  namespace: kube-system
spec:
  clusterIP: None
  ports:
  - name: http-metrics
    port: 9182
    protocol: TCP
    targetPort: 9182
  sessionAffinity: None
  type: ClusterIP

---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor ##Custom ServiceMonitor Object
metadata:
  labels:
    k8s-app: wmiexporter
  name: wmiexporter
  namespace: monitoring
spec:
  endpoints:
  - interval: 30s
    port: http-metrics
  jobLabel: k8s-app
  namespaceSelector:
    matchNames:
```

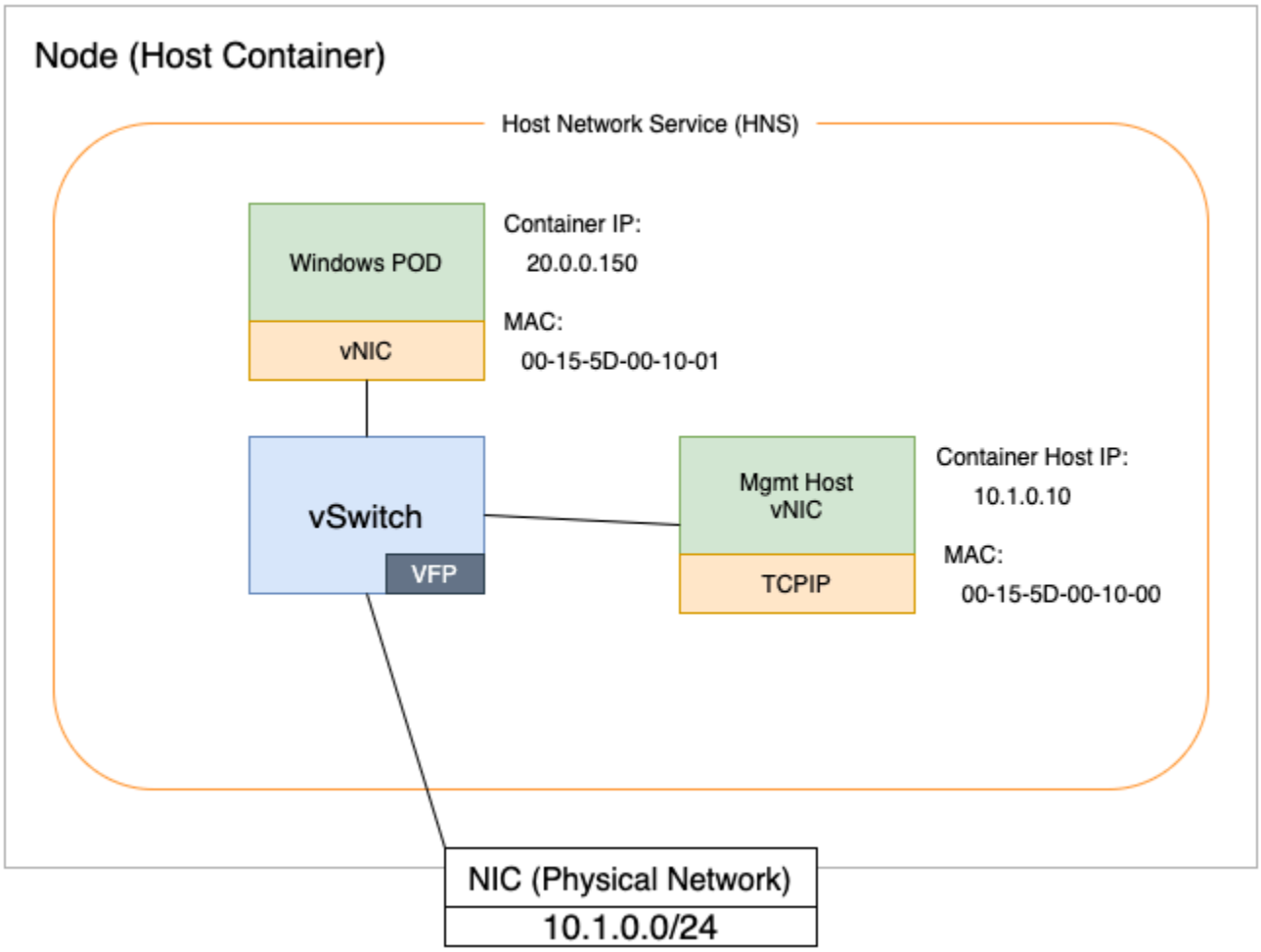
```
    - kube-system
  selector:
    matchLabels:
      k8s-app: wmiexporter
```

Pour plus de détails sur l'opérateur et son utilisation ServiceMonitor, consultez la documentation officielle de l'opérateur. Notez que Prometheus prend en charge la découverte dynamique des cibles à l'aide de nombreuses options de découverte de services.

Réseau Windows

Présentation de Windows Container Networking

Les conteneurs Windows sont fondamentalement différents des conteneurs Linux. Les conteneurs Linux utilisent des constructions Linux telles que les espaces de noms, le système de fichiers union et les cgroups. Sous Windows, ces constructions sont extraites de containerd par le Host Compute Service (HCS). HCS agit comme une couche d'API située au-dessus de l'implémentation du conteneur sous Windows. Les conteneurs Windows tirent également parti du Host Network Service (HNS) qui définit la topologie du réseau sur un nœud.



Du point de vue de la mise en réseau, HCS et HNS font fonctionner les conteneurs Windows comme des machines virtuelles. Par exemple, chaque conteneur possède un adaptateur réseau virtuel (vNIC) connecté à un commutateur virtuel Hyper-V (vSwitch), comme indiqué dans le schéma ci-dessus.

Gestion des adresses IP

Dans Amazon EKS, un nœud utilise son Elastic Network Interface (ENI) pour se connecter à un réseau AWS VPC. À l'heure actuelle, un seul ENI par nœud de travail Windows est pris en charge. La gestion des adresses IP pour les nœuds Windows est effectuée par le contrôleur de ressources VPC qui s'exécute dans le plan de contrôle. Vous trouverez plus de détails sur le flux de travail pour la gestion des adresses IP des nœuds Windows ici.

Le nombre de modules qu'un nœud de travail Windows peut prendre en charge dépend de la taille du nœud et du nombre d' IPv4 adresses disponibles. Vous pouvez calculer l' IPv4 adresse disponible sur le nœud comme ci-dessous :

- Par défaut, seules IPv4 les adresses secondaires sont attribuées à l'ENI. Dans un tel cas :

  ```
  Total IPv4 addresses available for Pods = Number of supported IPv4 addresses in the
   primary interface - 1
  ```

  Nous en soustrayons une du nombre total, car une IPv4 adresse sera utilisée comme adresse principale de l'ENI et ne peut donc pas être attribuée aux Pods.
- Si le cluster a été configuré pour une densité de pods élevée en activant la fonctionnalité de délégation de préfixes, alors-

  ```
  Total IPv4 addresses available for Pods = (Number of supported IPv4 addresses in the
   primary interface - 1) * 16
  ```

  Ici, au lieu d'allouer des IPv4 adresses secondaires, le VPC Resource Controller les `/28` `prefixes` allouera et, par conséquent, le nombre total d'adresses IPv4 disponibles sera multiplié par 16.

À l'aide de la formule ci-dessus, nous pouvons calculer le nombre maximum de pods pour un serveur de travail Windows nodé sur la base d'une instance m5.large, comme ci-dessous :

- Par défaut, lors de l'exécution en mode IP secondaire,

  ```
  10 secondary IPv4 addresses per ENI - 1 = 9 available IPv4 addresses
  ```
- Lorsque vous utilisez `prefix delegation` -

  ```
  (10 secondary IPv4 addresses per ENI - 1) * 16 = 144 available IPv4 addresses
  ```

Pour plus d'informations sur le nombre d'adresses IP qu'un type d'instance peut prendre en charge, consultez la section Adresses IP par interface réseau et par type d'instance.

Le flux du trafic réseau est un autre élément clé à prendre en compte. Windows présente un risque d'épuisement des ports sur les nœuds comportant plus de 100 services. Lorsque cette condition se présente, les nœuds commencent à générer des erreurs avec le message suivant :

« Échec de la création de la politique : échec de l' hcnCreateLoadéquilibreur dans Win32 : le port spécifié existe déjà. »

Pour résoudre ce problème, nous utilisons Direct Server Return (DSR). Le DSR est une implémentation de la distribution asymétrique de la charge réseau. En d'autres termes, le trafic de demande et de réponse utilise des chemins réseau différents. Cette fonctionnalité accélère la communication entre les pods et réduit le risque d'épuisement des ports. Nous recommandons donc d'activer le DSR sur les nœuds Windows.

Le DSR est activé par défaut dans Windows Server SAC EKS Optimized AMIs. Pour Windows Server 2019 LTSC EKS Optimized AMIs, vous devez l'activer lors du provisionnement de l'instance à l'aide du script ci-dessous et en utilisant Windows Server 2019 Full ou Core comme famille d'amis dans le groupe de nœuds. `eksctl` Consultez l'AMI personnalisée eksctl pour plus d'informations.

```
nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
```

Pour utiliser le DSR dans Windows Server 2019 et versions ultérieures, vous devez spécifier les indicateurs kube-proxy suivants lors du démarrage de l'instance. Vous pouvez le faire en ajustant le script userdata associé au modèle de lancement des groupes de nœuds autogérés.

```
<powershell>
[string]$EKSBinDir = "$env:ProgramFiles\Amazon\EKS"
[string]$EKSBootstrapScriptName = 'Start-EKSBootstrap.ps1'
[string]$EKSBootstrapScriptFile = "$EKSBinDir\$EKSBootstrapScriptName"
(Get-Content $EKSBootstrapScriptFile).replace('"--proxy-mode=kernelspace",', '"--proxy-
mode=kernelspace", "--feature-gates WinDSR=true", "--enable-dsr",') | Set-Content
 $EKSBootstrapScriptFile
& $EKSBootstrapScriptFile -EKSClusterName "eks-windows" -APIServerEndpoint "https://
<REPLACE-EKS-CLUSTER-CONFIG-API-SERVER>" -Base64ClusterCA "<REPLACE-EKSCLUSTER-
CONFIG-DETAILS-CA>" -DNSClusterIP "172.20.0.10" -KubeletExtraArgs "--node-
labels=alpha.eksctl.io/cluster-name=eks-windows,alpha.eksctl.io/nodegroup-name=windows-
ng-ltsc2019 --register-with-taints=" 3>&1 4>&1 5>&1 6>&1
</powershell>
```

L'activation du DSR peut être vérifiée en suivant les instructions du blog Microsoft Networking et du laboratoire Windows Containers on AWS.

```
PS C:\temp> (Get-Content C:\temp\pmsx1eau.0ia\policy.txt | ConvertFrom-JSON).Policies


ExternalPort : 53
InternalPort : 53
IsDSR        : True
Protocol     : 17
Type         : ELB
VIPs         : {172.20.0.10}

ExternalPort : 53
InternalPort : 53
IsDSR        : True
Protocol     : 6
Type         : ELB
VIPs         : {172.20.0.10}

ExternalPort : 443
InternalPort : 443
IsDSR        : True
Protocol     : 6
Type         : ELB
VIPs         : {172.20.0.1}
```

S'il est crucial de préserver vos IPv4 adresses disponibles et de minimiser le gaspillage pour votre sous-réseau, il est généralement recommandé d'éviter d'utiliser le mode de délégation de préfixes, comme indiqué dans Mode préfixe pour Windows - Quand éviter. Si l'utilisation de la délégation de préfixes est toujours souhaitée, vous pouvez prendre des mesures pour optimiser l'utilisation des IPv4 adresses dans votre sous-réseau. Consultez la section Configuration des paramètres pour la délégation de préfixes pour obtenir des instructions détaillées sur la manière d'affiner le processus de demande et d'allocation d' IPv4 adresses. L'ajustement de ces configurations peut vous aider à trouver un équilibre entre la conservation des IPv4 adresses et les avantages de la délégation de préfixes en termes de densité de modules.

Lorsque vous utilisez le paramètre par défaut d'attribution d' IPv4 adresses secondaires, aucune configuration n'est actuellement prise en charge pour manipuler la manière dont le contrôleur de ressources VPC demande et IPv4 alloue les adresses. Plus précisément, `minimum-ip-target` `warm-ip-target` ils ne sont pris en charge que pour le mode de délégation de préfixes. Notez également qu'en mode IP secondaire, en fonction des adresses IP disponibles sur l'interface, le contrôleur de ressources VPC alloue généralement 3 IPv4 adresses inutilisées au nœud en votre nom afin de rester au chaud et IPs d'accélérer le démarrage du pod. Si vous souhaitez minimiser le gaspillage d'adresses IP non utilisées, vous pouvez planifier davantage de pods sur un nœud Windows donné afin d'utiliser autant de capacité d'adresses IP de l'ENI que possible. Plus précisément, vous pouvez éviter que le warm ne soit inutilisé IPs si toutes les adresses IP de l'ENI

sont déjà utilisées par le nœud et les pods en cours d'exécution. Une autre solution pour vous aider à résoudre les problèmes liés à la disponibilité des adresses IP dans vos sous-réseaux pourrait consister à augmenter la taille de votre sous-réseau ou à séparer vos nœuds Windows en leurs propres sous-réseaux dédiés.

En outre, il est important de noter que cela n' IPv6 est pas pris en charge sur les nœuds Windows pour le moment.

Options d'interface réseau de conteneurs (CNI)

Le AWSVPC CNI est le plugin CNI de facto pour les nœuds de travail Windows et Linux. Bien que le AWSVPC CNI réponde aux besoins de nombreux clients, il peut arriver que vous deviez envisager des alternatives, comme un réseau superposé pour éviter l'épuisement des adresses IP. Dans ces cas, le Calico CNI peut être utilisé à la place du AWSVPC CNI. Project Calico est un logiciel open source développé par Tigera. Ce logiciel inclut un CNI qui fonctionne avec EKS. Les instructions d'installation de Calico CNI dans EKS se trouvent sur la page d'installation du projet Calico EKS.

Politiques du réseau

Il est considéré comme une bonne pratique de passer du mode de communication ouverte par défaut entre les pods de votre cluster Kubernetes à la limitation de l'accès en fonction des politiques réseau. Le projet open source Calico bénéficie d'un solide soutien pour les politiques réseau qui fonctionnent à la fois avec les nœuds Linux et Windows. Cette fonctionnalité est distincte et ne dépend pas de l'utilisation du Calico CNI. Nous recommandons donc d'installer Calico et de l'utiliser pour la gestion des politiques réseau.

Les instructions d'installation de Calico dans EKS sont disponibles sur la page Installation de Calico sur Amazon EKS.

En outre, les conseils fournis dans la section Réseau du guide des meilleures pratiques d'Amazon EKS en matière de sécurité s'appliquent également aux clusters EKS dotés de nœuds de travail Windows. Toutefois, certaines fonctionnalités telles que les « groupes de sécurité pour les pods » ne sont pas prises en charge par Windows pour le moment.

Éviter les erreurs OOM

Windows n'a pas de tueur de out-of-memory processus comme Linux. Windows traite toujours toutes les allocations de mémoire en mode utilisateur comme virtuelles, et les fichiers de page sont

obligatoires. Le résultat net est que Windows n'atteindra pas les limites de mémoire de la même manière que Linux. Les processus seront redirigés vers le disque au lieu d'être interrompus par manque de mémoire (OOM). Si la mémoire est surprovisionnée et que toute la mémoire physique est épuisée, la pagination peut ralentir les performances.

Système de réservation et mémoire Kubelet

Différent de Linux où la réservation de ressources de `--kubelet-reserve` capture pour les démons du système Kubernetes tels que Kubelet, Container Runtime, etc. et la réservation de ressources de `--system-reserve` capture pour les démons du système d'exploitation tels que sshd, udev, etc. Sous Windows, ces indicateurs ne capturent pas et ne définissent pas de limites de mémoire sur Kubelet ou sur les processus exécutés sur le nœud.

Cependant, vous pouvez combiner ces indicateurs NodeAllocatablepour réduire la capacité du nœud avec la limite de ressources de mémoire du manifeste du pod afin de contrôler l'allocation de mémoire par pod. Cette stratégie vous permet de mieux contrôler l'allocation de mémoire ainsi que d'un mécanisme de minimisation out-of-memory (OOM) sur les nœuds Windows.

Sur les nœuds Windows, il est recommandé de réserver au moins 2 Go de mémoire pour le système d'exploitation et le processus. `--kubelet-reserve` and/or `--system-reserve`À utiliser pour réduire NodeAllocatable.

En suivant la documentation sur les nœuds Windows autogérés par Amazon EKS, utilisez le CloudFormation modèle pour lancer un nouveau groupe de nœuds Windows avec des personnalisations de la configuration de Kubelet. CloudFormation Il possède un élément appelé `BootstrapArguments` qui est identique à`KubeletExtraArgs`. À utiliser avec les indicateurs et valeurs suivants :

```
--kube-reserved memory=0.5Gi,ephemeral-storage=1Gi --system-
reserved memory=1.5Gi,ephemeral-storage=1Gi --eviction-hard
 memory.available<200Mi,nodefs.available<10%"
```

Si eksctl est l'outil de déploiement, consultez la documentation suivante pour personnaliser la configuration de kubelet https://eksctl. io/usage/customizing-le-kubelet/

Exigences relatives à la mémoire des conteneurs Windows

Selon la documentation Microsoft, une image de base Windows Server pour NANO nécessite au moins 30 Mo, tandis que Server Core nécessite 45 Mo. Ces chiffres augmentent à mesure que vous

ajoutez des composants Windows tels que le .NET Framework, les services Web tels qu'IIS et les applications.

Il est essentiel que vous connaissiez la quantité minimale de mémoire requise par votre image de conteneur Windows, c'est-à-dire l'image de base et ses couches d'application, et que vous la définissiez comme celle du conteneur resources/requests dans la spécification du pod. Vous devez également définir une limite pour éviter que les pods ne consomment toute la mémoire disponible sur les nœuds en cas de problème lié à l'application.

Dans l'exemple ci-dessous, lorsque le planificateur Kubernetes essaie de placer un pod sur un nœud, les demandes du pod sont utilisées pour déterminer quel nœud dispose de suffisamment de ressources disponibles pour la planification.

```
spec:
 - name: iis
   image: mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019
   resources:
     limits:
       cpu: 1
       memory: 800Mi
     requests:
       cpu: .1
       memory: 128Mi
```

Conclusion

L'utilisation de cette approche minimise les risques d'épuisement de la mémoire mais ne l'empêche pas de se produire. À l'aide d'Amazon CloudWatch Metrics, vous pouvez configurer des alertes et des mesures correctives en cas d'épuisement de la mémoire.

Appliquer des correctifs aux serveurs et aux conteneurs Windows

L'application de correctifs à Windows Server est une tâche de gestion standard pour les administrateurs Windows. Cela peut être accompli à l'aide de différents outils tels qu'Amazon System Manager - Patch Manager, WSUS, System Center Configuration Manager et bien d'autres. Toutefois, les nœuds Windows d'un cluster Amazon EKS ne doivent pas être traités comme des serveurs Windows ordinaires. Ils doivent être traités comme des serveurs immuables. En termes simples, évitez de mettre à jour un nœud existant, lancez-en simplement un nouveau sur la base d'une nouvelle AMI mise à jour.

EC2 Image Builder vous permet d' AMIs automatiser la création en créant des recettes et en ajoutant des composants.

L'exemple suivant montre les composants, qui peuvent être des composants préexistants créés par AWS (gérés par Amazon) ainsi que les composants que vous créez (Owned by me). Portez une attention particulière au composant géré par Amazon appelé update-windows. Il met à jour Windows Server avant de générer l'AMI via le pipeline EC2 Image Builder.

Associated components

Search components

Component name	Version	Description	Date created	Platform	Owner	ARN
eks-optimized-ami-windows	1.20.0	Installs EKS-optimized Windows artifacts for EKS version 1.20. This includes kubelet version 1.20.4 and Docker version 20.10.6.	Aug 18, 2021 6:33 PM	Windows	Amazon	arn:aws:imagebuilder:us-east-1:aws:component/eks-optimized-ami-windows/1.20.0/1
update-windows	1.0.1	Updates Windows with the latest security updates.	Aug 18, 2021 6:33 PM	Windows	Amazon	arn:aws:imagebuilder:us-east-1:aws:component/update-windows/1.0.1/1
Docker Pull	1.0.1	-	Dec 22, 2021 12:08 PM	Windows	[redacted]	arn:aws:imagebuilder:us-east-[redacted]component/docker-pull/1.0.1/1
simple-boot-test-windows	1.0.0	Executes a simple boot test.	Nov 30, 2019 11:39 PM	Windows	Amazon	arn:aws:imagebuilder:us-east-1:aws:component/simple-boot-test-windows/1.0.0/1

EC2 Image Builder vous permet de créer des AMI basées sur Amazon Managed AMIs Public et de les personnaliser pour répondre aux besoins de votre entreprise. Vous pouvez ensuite les AMIs associer à des modèles de lancement, ce qui vous permet de lier une nouvelle AMI au groupe Auto Scaling créé par le groupe de nœuds EKS. Une fois cette opération terminée, vous pouvez commencer à mettre fin aux nœuds Windows existants et de nouveaux nœuds seront lancés en fonction de la nouvelle AMI mise à jour.

Pousser et extraire des images Windows

Amazon publie une version optimisée d'EKS AMIs qui inclut deux images de conteneur Windows mises en cache.

```
mcr.microsoft.com/windows/servercore
mcr.microsoft.com/windows/nanoserver
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mcr.microsoft.com/windows/servercore	ltsc2019	617dd5ead5b8	6 weeks ago	5.74GB
mcr.microsoft.com/windows/nanoserver	1809	8334af2fa7ba	6 weeks ago	257MB
[redacted].dkr.ecr.us-east-1.amazonaws.com/fluentd-windows-coreltsc	latest	721afca2c725	7 weeks ago	6.96GB
fluent/fluentd	v1.14-windows-ltsc2019-1	721afca2c725	7 weeks ago	6.96GB
amazonaws.com/eks/pause-windows	latest	6392f69ae6e7	10 months ago	255MB

Les images mises en cache sont mises à jour suite aux mises à jour du système d'exploitation principal. Lorsque Microsoft publie une nouvelle mise à jour Windows qui affecte directement l'image de base du conteneur Windows, la mise à jour sera lancée sous la forme d'une mise à jour Windows ordinaire sur le système d'exploitation principal. Le maintien de l'environnement up-to-date offre un environnement plus sécurisé au niveau du nœud et du conteneur.

La taille d'une image de conteneur Windows influence push/pull les opérations, ce qui peut ralentir le démarrage du conteneur. La mise en cache des images de conteneurs Windows permet de réaliser I/O des opérations coûteuses (extraction de fichiers) lors de la création de la version de l'AMI plutôt que lors du lancement du conteneur. Par conséquent, toutes les couches d'image nécessaires seront extraites sur l'AMI et seront prêtes à être utilisées, ce qui accélérera le lancement d'un conteneur Windows et pourra commencer à accepter du trafic. Lors d'une opération push, seules les couches qui composent votre image sont chargées dans le référentiel.

L'exemple suivant montre que sur Amazon ECR, les images de fluentd-windows-sac2004 ne font que 390,18 Mo. Il s'agit du volume de téléchargement effectué lors de l'opération push.

L'exemple suivant montre une image Windows Ltsc Fluentd envoyée vers un référentiel Amazon ECR. La taille de la couche stockée dans ECR est de 533,05 Mo.

Image details

Image URI

[redacted].dkr.ecr.us-east-1.amazonaws.com/fluentd-windows-coreltsc

Digest

sha256:2ddb820abe[redacted]638185ee46c454cb856b9d03f257960cdf2e6de9c

Image tags	Repository
latest	fluentd-windows-coreltsc
Pushed at	**Size (MB)**
December 22, 2021, 13:47:36 (UTC-05)	533.05

La sortie ci-dessous de`docker image ls`, la taille du fluentd v1.14-windows-ltsc2019-1 est de 6,96 Go sur le disque, mais cela ne signifie pas qu'il a téléchargé et extrait cette quantité de données.

En pratique, pendant l'opération d'extraction, seuls les 533,05 Mo compressés seront téléchargés et extraits.

```
REPOSITORY                                                          TAG
            IMAGE ID       CREATED         SIZE
111122223333.dkr.ecr.us-east-1.amazonaws.com/fluentd-windows-coreltsc   latest
            721afca2c725   7 weeks ago     6.96GB
```

```
fluent/fluentd                                                          v1.14-windows-
ltsc2019-1   721afca2c725   7 weeks ago     6.96GB
amazonaws.com/eks/pause-windows                                         latest
            6392f69ae6e7   10 months ago   255MB
```

La colonne de taille indique la taille totale de l'image, 6,96 Go. En le décomposant :

- Image de base LTSC Windows Server Core 2019 = 5,74 Go
- Image de base non compressée Fluentd = 6,96 Go
- Différence sur le disque = 1,2 Go
- ECR de l'image finale compressée Fluentd = 533,05 Mo

L'image de base existe déjà sur le disque local, ce qui signifie que la quantité totale sur le disque est de 1,2 Go supplémentaire. La prochaine fois que vous verrez la quantité GBs dans la colonne de taille, ne vous inquiétez pas trop, il est probable que plus de 70 % se trouvent déjà sur le disque sous forme d'image de conteneur mise en cache.

Référence

Accélération des délais de lancement des conteneurs Windows grâce au générateur d'images EC2 et à la stratégie de cache d'images

Exécution de charges de travail hétérogènes

Kubernetes prend en charge les clusters hétérogènes dans lesquels vous pouvez avoir un mélange de nœuds Linux et Windows dans le même cluster. Au sein de ce cluster, vous pouvez avoir un mélange de pods qui s'exécutent sous Linux et de pods qui s'exécutent sous Windows. Vous pouvez même exécuter plusieurs versions de Windows dans le même cluster. Cependant, plusieurs facteurs (tels que mentionnés ci-dessous) devront être pris en compte lors de la prise de cette décision.

Meilleures pratiques PODs d'attribution aux nœuds

Afin de conserver les charges de travail Linux et Windows sur leurs nœuds spécifiques au système d'exploitation respectifs, vous devez utiliser une combinaison de sélecteurs de nœuds et d'altérations/tolérances. L'objectif principal de la planification des charges de travail dans un environnement hétérogène est d'éviter de compromettre la compatibilité des charges de travail Linux existantes.

Veiller à ce que les charges de travail spécifiques au système d'exploitation arrivent sur le conteneur hôte approprié

Les utilisateurs peuvent s'assurer que les conteneurs Windows peuvent être planifiés sur l'hôte approprié à l'aide de NodeSelectors. Tous les nœuds Kubernetes portent aujourd'hui les libellés par défaut suivants :

```
kubernetes.io/os = [windows|linux]
kubernetes.io/arch = [amd64|arm64|...]
```

Si une spécification de Pod n'inclut pas de NodeSelector`"kubernetes.io/os": windows`, le Pod peut être planifié sur n'importe quel hôte, Windows ou Linux. Cela peut être problématique car un conteneur Windows ne peut fonctionner que sous Windows et un conteneur Linux ne peut fonctionner que sous Linux.

Dans les environnements d'entreprise, il n'est pas rare de disposer d'un grand nombre de déploiements préexistants pour les conteneurs Linux, ainsi que d'un écosystème de off-the-shelf configurations, comme les cartes Helm. Dans ces situations, vous pouvez hésiter à apporter des modifications aux NodeSelectors d'un déploiement. L'alternative est d'utiliser Taints.

Par exemple : `--register-with-taints='os=windows:NoSchedule'`

Si vous utilisez EKS, eksctl propose des moyens d'appliquer des taches via ClusterConfig :

```
NodeGroups:
  - name: windows-ng
    amiFamily: WindowsServer2022FullContainer
    ...
    labels:
      nodeclass: windows2022
    taints:
      os: "windows:NoSchedule"
```

En altérant tous les nœuds Windows, le planificateur ne planifiera pas les pods sur ces nœuds à moins qu'ils ne tolèrent cette altération. Exemple de manifeste Pod :

```
nodeSelector:
    kubernetes.io/os: windows
tolerations:
```

```
    - key: "os"
      operator: "Equal"
      value: "windows"
      effect: "NoSchedule"
```

Gestion de plusieurs versions de Windows dans le même cluster

L'image de base du conteneur Windows utilisée par chaque pod doit correspondre à la même version de construction du noyau que le nœud. Si vous souhaitez utiliser plusieurs versions de Windows Server dans le même cluster, vous devez définir des étiquettes de nœuds supplémentaires, des NodeSelectors ou utiliser une étiquette appelée windows-build.

Kubernetes 1.17 ajoute automatiquement une nouvelle étiquette node.kubernetes.io/windows-build afin de simplifier la gestion de plusieurs versions de Windows dans le même cluster. Si vous utilisez une ancienne version, il est recommandé d'ajouter cette étiquette manuellement aux nœuds Windows.

Cette étiquette indique les numéros de version majeure, mineure et de version de Windows qui doivent correspondre pour des raisons de compatibilité. Vous trouverez ci-dessous les valeurs utilisées aujourd'hui pour chaque version de Windows Server.

Il est important de noter que Windows Server passe au canal LTSC (Long-Term Servicing Channel) en tant que canal de publication principal. Le canal semi-annuel (SAC) Windows Server a été retiré le 9 août 2022. Il n'y aura aucune future version SAC de Windows Server.

Nom de produit	Numéro (s) de version
Serveur LTSC 2022 complet	10,0.20348
Noyau du serveur 2019 LTSC	10,0.17763

Il est possible de vérifier la version de build du système d'exploitation à l'aide de la commande suivante :

```
kubectl get nodes -o wide
```

La sortie KERNEL-VERSION correspond à la version de build du système d'exploitation Windows.

```
NAME                          STATUS   ROLES    AGE   VERSION               INTERNAL-
IP   EXTERNAL-IP     OS-IMAGE                         KERNEL-VERSION
 CONTAINER-RUNTIME
ip-10-10-2-235.ec2.internal   Ready    <none>   23m   v1.24.7-eks-fb459a0
 10.10.2.235   3.236.30.157    Windows Server 2022 Datacenter   10.0.20348.1607
         containerd://1.6.6
ip-10-10-31-27.ec2.internal   Ready    <none>   23m   v1.24.7-eks-fb459a0
 10.10.31.27   44.204.218.24   Windows Server 2019 Datacenter   10.0.17763.4131
         containerd://1.6.6
ip-10-10-7-54.ec2.internal    Ready    <none>   31m   v1.24.11-eks-a59e1f0   10.10.7.54
    3.227.8.172     Amazon Linux 2                    5.10.173-154.642.amzn2.x86_64
 containerd://1.6.19
```

L'exemple ci-dessous applique un NodeSelector supplémentaire au manifeste du pod afin de correspondre à la version Windows correcte lors de l'exécution de différentes versions du système d'exploitation de groupes de nœuds Windows.

```
nodeSelector:
    kubernetes.io/os: windows
    node.kubernetes.io/windows-build: '10.0.20348'
tolerations:
    - key: "os"
    operator: "Equal"
    value: "windows"
    effect: "NoSchedule"
```

Simplification NodeSelector et tolérance dans les manifestes Pod à l'aide de RuntimeClass

Vous pouvez également l'utiliser RuntimeClass pour simplifier le processus d'utilisation des colorants et des tolérances. Cela peut être accompli en créant un RuntimeClass objet qui est utilisé pour encapsuler ces souillures et tolérances.

Créez un RuntimeClass en exécutant le manifeste suivant :

```
apiVersion: node.k8s.io/v1beta1
kind: RuntimeClass
metadata:
  name: windows-2022
handler: 'docker'
```

```
scheduling:
  nodeSelector:
    kubernetes.io/os: 'windows'
    kubernetes.io/arch: 'amd64'
    node.kubernetes.io/windows-build: '10.0.20348'
  tolerations:
  - effect: NoSchedule
    key: os
    operator: Equal
    value: "windows"
```

Une fois le Runtimeclass créé, attribuez-le en utilisant comme spécification sur le manifeste du Pod :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: iis-2022
  labels:
    app: iis-2022
spec:
  replicas: 1
  template:
    metadata:
      name: iis-2022
      labels:
        app: iis-2022
    spec:
      runtimeClassName: windows-2022
      containers:
      - name: iis
```

Support pour les groupes de nœuds gérés

Pour aider les clients à exécuter leurs applications Windows de manière plus rationalisée, AWS a lancé le support du groupe de nœuds gérés (MNG) Amazon EKS pour les conteneurs Windows le 15 décembre 2022. Pour aider à aligner les équipes opérationnelles, Windows MNGs est activé à l'aide des mêmes flux de travail et outils que Linux MNGs. Les versions complètes et principales de la famille AMI (Amazon Machine Image) de Windows Server 2019 et 2022 sont prises en charge.

Les familles d'AMI suivantes sont prises en charge pour les groupes de nœuds gérés (MNG).

Famille AMI
Windows_Core_2019_x86_64
Windows_Full_2019_x86_64
Windows_Core_2022_x86_64
Windows_Full_2022_x86_64

Documentations supplémentaires

Documentation officielle AWS : https://docs.aws.amazon.com/eks/ latest/userguide/windows - support.html

Pour mieux comprendre le fonctionnement de Pod Networking (CNI), consultez le lien suivant : - networking.html https://docs.aws.amazon.com/eks/ latest/userguide/pod

Blog AWS sur le déploiement d'un groupe de nœuds gérés pour Windows sur EKS : https:// aws.amazon.com/blogs/ containers/ -/deploying-amazon-eks-windowsmanaged-node-groups

Contextes de sécurité des modules

Les politiques de sécurité des pods (PSP) et les normes de sécurité des pods (PSS) sont les deux principaux moyens de renforcer la sécurité dans Kubernetes. Notez qu'elle PodSecurityPolicy est obsolète depuis Kubernetes v1.21 et sera supprimée dans la version v1.25. Pod Security Standard (PSS) est l'approche recommandée par Kubernetes pour renforcer la sécurité à l'avenir.

Une politique de sécurité des pods (PSP) est une solution native de Kubernetes permettant de mettre en œuvre des politiques de sécurité. La PSP est une ressource au niveau du cluster qui contrôle les aspects sensibles à la sécurité de la spécification du Pod. À l'aide de la politique de sécurité des pods, vous pouvez définir un ensemble de conditions que les pods doivent remplir pour être acceptés par le cluster. La fonctionnalité PSP est disponible depuis les débuts de Kubernetes et est conçue pour empêcher la création de pods mal configurés sur un cluster donné.

Pour plus d'informations sur les politiques de sécurité des pods, veuillez consulter la documentation de Kubernetes. Selon la politique de dépréciation de Kubernetes, les anciennes versions ne seront plus prises en charge neuf mois après la dépréciation de la fonctionnalité.

D'autre part, les normes de sécurité des pods (PSS), qui constituent l'approche de sécurité recommandée et sont généralement mises en œuvre à l'aide de contextes de sécurité, sont définies dans le cadre des spécifications du pod et du conteneur dans le manifeste du pod. Le PSS est la norme officielle que l'équipe du projet Kubernetes a définie pour répondre aux meilleures pratiques en matière de sécurité pour les Pods. Il définit des politiques telles que les politiques de base (minimalement restrictives, par défaut), privilégiées (non restrictives) et restreintes (les plus restrictives).

Nous vous recommandons de commencer par le profil de référence. Le profil de base du PSS fournit un équilibre solide entre la sécurité et les frictions potentielles, nécessitant une liste minimale d'exceptions, il constitue un bon point de départ pour la sécurité des charges de travail. Si vous l'utilisez actuellement, PSPs nous vous recommandons de passer au PSS. Vous trouverez plus de détails sur les politiques PSS dans la documentation de Kubernetes. Ces politiques peuvent être appliquées à l'aide de plusieurs outils, notamment ceux de l'OPA et de Kyverno. Par exemple, Kyverno fournit la collection complète des politiques PSS ici.

Les paramètres du contexte de sécurité permettent d'accorder des privilèges pour sélectionner des processus, d'utiliser des profils de programme pour restreindre les fonctionnalités à des programmes individuels, d'autoriser l'augmentation des privilèges, de filtrer les appels système, entre autres choses.

Les pods Windows de Kubernetes présentent certaines limites et se différencient des charges de travail standard basées sur Linux en ce qui concerne les contextes de sécurité.

Windows utilise un objet Job par conteneur avec un filtre d'espace de noms système pour contenir tous les processus d'un conteneur et fournir une isolation logique par rapport à l'hôte. Il est impossible d'exécuter un conteneur Windows sans le filtrage des espaces de noms en place. Cela signifie que les privilèges système ne peuvent pas être revendiqués dans le contexte de l'hôte et que les conteneurs privilégiés ne sont donc pas disponibles sous Windows.

Les seules options de contexte de sécurité Windows documentées `windowsOptions` sont les suivantes, tandis que les autres sont des options générales de contexte de sécurité.

Pour obtenir la liste des attributs de contexte de sécurité pris en charge par Windows et Linux, veuillez consulter la documentation officielle ici.

Les paramètres spécifiques au Pod sont appliqués à tous les conteneurs. Si elles ne sont pas spécifiées, les options du PodSecurityContext seront utilisées. Si cette valeur est définie à la fois dans SecurityContext et PodSecurityContext, la valeur spécifiée dans SecurityContext est prioritaire.

Par exemple, le paramètre runAsUser Nom pour les pods et les conteneurs, qui est une option Windows, est un équivalent approximatif du runAsUser paramètre spécifique à Linux. Dans le manifeste suivant, le contexte de sécurité spécifique au pod est appliqué à tous les conteneurs.

```
apiVersion: v1
kind: Pod
metadata:
  name: run-as-username-pod-demo
spec:
  securityContext:
    windowsOptions:
      runAsUserName: "ContainerUser"
  containers:
  - name: run-as-username-demo

  nodeSelector:
    kubernetes.io/os: windows
```

Alors que dans ce qui suit, le contexte de sécurité au niveau du conteneur remplace le contexte de sécurité au niveau du pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: run-as-username-container-demo
spec:
  securityContext:
    windowsOptions:
      runAsUserName: "ContainerUser"
  containers:
  - name: run-as-username-demo
    ..
    securityContext:
        windowsOptions:
            runAsUserName: "ContainerAdministrator"
  nodeSelector:
    kubernetes.io/os: windows
```

Exemples de valeurs acceptables pour le champ runAsUser Nom : ContainerAdministrator, ContainerUser, NT AUTHORITY \ NETWORK SERVICE, NT AUTHORITY \ LOCAL SERVICE

Il est généralement judicieux d'exécuter vos conteneurs avec des pods ContainerUser pour Windows. Les utilisateurs ne sont pas partagés entre le conteneur et l' ContainerAdministrator hôte, mais ils ont des privilèges supplémentaires dans le conteneur. Notez qu'il existe des limites de nom d'utilisateur à connaître.

Un bon exemple de cas d'utilisation ContainerAdministrator consiste à définir PATH. Vous pouvez utiliser la directive USER pour cela, comme suit :

```
USER ContainerAdministrator
RUN setx /M PATH "%PATH%;C:/your/path"
USER ContainerUser
```

Notez également que les secrets sont écrits en texte clair sur le volume du nœud (par rapport à tmpfs/in-memory sous Linux). Cela signifie que vous devez faire deux choses

- Utiliser le fichier ACLs pour sécuriser l'emplacement du fichier secret
- Utilisez le chiffrement au niveau du volume en utilisant BitLocker

Options de stockage permanent

Qu'est-ce qu'un plugin intégré à l'arbre par rapport à un plugin out-of-tree de volume ?

Avant l'introduction de l'interface de stockage de conteneurs (CSI), tous les plugins de volume étaient intégrés à l'arborescence, ce qui signifie qu'ils étaient créés, liés, compilés et livrés avec les principaux binaires Kubernetes et étendaient l'API Kubernetes principale. Cela signifiait que l'ajout d'un nouveau système de stockage à Kubernetes (un plugin de volume) nécessitait de vérifier le code dans le référentiel de code principal de Kubernetes.

Out-of-tree les plugins de volume sont développés indépendamment de la base de code Kubernetes et sont déployés (installés) sur les clusters Kubernetes en tant qu'extensions. Cela permet aux fournisseurs de mettre à jour les pilotes out-of-band, c'est-à-dire indépendamment du cycle de publication de Kubernetes. Cela est largement possible car Kubernetes a créé une interface de stockage ou CSI qui fournit aux fournisseurs un moyen standard d'interfaçage avec k8s.

Vous pouvez en savoir plus sur les classes de stockage et les pilotes CSI d'Amazon Elastic Kubernetes Services (EKS) sur .html https://docs.aws.amazon.com/eks/ latest/userguide/storage

Plug-in In-Tree Volume pour Windows

Les volumes Kubernetes permettent de déployer des applications soumises à des exigences de persistance des données sur Kubernetes. La gestion des volumes persistants consiste en des conteneurs provisioning/de-provisioning/resizing of volumes, attaching/detaching a volume to/from a Kubernetes node, and mounting/dismounting a volume to/from individuels placés dans un pod. Le code permettant d'implémenter ces actions de gestion des volumes pour un backend ou un protocole de stockage spécifique est fourni sous la forme d'un plug-in de volume Kubernetes (plug-ins de volume intégrés à l'arbre). Sur Amazon Elastic Kubernetes Services (EKS), la classe suivante de plug-ins de volume Kubernetes est prise en charge sous Windows :

Plugin de volume intégré à l'arbre : Store awsElasticBlock

Pour utiliser le plugin de volume In-tree sur les nœuds Windows, il est nécessaire d'en créer un supplémentaire StorageClass pour utiliser NTFS en tant que FSType. Sur EKS, la valeur par défaut StorageClass utilise ext4 comme FSType par défaut.

A StorageClass permet aux administrateurs de décrire les « classes » de stockage qu'ils proposent. Les différentes classes peuvent correspondre à quality-of-service des niveaux, à des politiques de sauvegarde ou à des politiques arbitraires déterminées par les administrateurs du cluster. Kubernetes n'a aucune opinion quant à ce que représentent les classes. Ce concept est parfois appelé « profils » dans d'autres systèmes de stockage.

Vous pouvez le vérifier en exécutant la commande suivante :

```
kubectl describe storageclass gp2
```

Sortie :

```
Name:            gp2
IsDefaultClass:  Yes
Annotations:     kubectl.kubernetes.io/last-applied-
configuration={"apiVersion":"storage.k8s.io/v1","kind":"StorageClas
","metadata":{"annotations":{"storageclass.kubernetes.io/is-default-
class":"true"},"name":"gp2"},"parameters":{"fsType"
"ext4","type":"gp2"},"provisioner":"kubernetes.io/aws-
ebs","volumeBindingMode":"WaitForFirstConsumer"}
,storageclass.kubernetes.io/is-default-class=true
Provisioner:           kubernetes.io/aws-ebs
```

```
Parameters:            fsType=ext4,type=gp2
AllowVolumeExpansion:  <unset>
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     WaitForFirstConsumer
Events:                <none>
```

Pour créer le nouveau fichier StorageClass compatible avec NTFS, utilisez le manifeste suivant :

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gp2-windows
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  fsType: ntfs
volumeBindingMode: WaitForFirstConsumer
```

Créez le StorageClass en exécutant la commande suivante :

```
kubectl apply -f NTFSStorageClass.yaml
```

L'étape suivante consiste à créer une réclamation de volume persistante (PVC).

Un PersistentVolume (PV) est un élément de stockage du cluster qui a été provisionné par un administrateur ou approvisionné dynamiquement à l'aide de PVC. Il s'agit d'une ressource du cluster, tout comme un nœud est une ressource du cluster. Cet objet d'API capture les détails de la mise en œuvre du stockage, qu'il s'agisse d'un système NFS, iSCSI ou cloud-provider-specific d'un système de stockage.

Un PersistentVolumeClaim (PVC) est une demande de stockage par un utilisateur. Les réclamations peuvent demander une taille et des modes d'accès spécifiques (par exemple, elles peuvent être montées ReadWriteOnce ReadOnlyMany ou ReadWriteMany).

Les utilisateurs ont besoin PersistentVolumes de différents attributs, tels que les performances, pour différents cas d'utilisation. Les administrateurs de clusters doivent être en mesure de proposer une variété de solutions PersistentVolumes qui ne se limitent pas à la taille et aux modes d'accès, sans exposer les utilisateurs aux détails de la mise en œuvre de ces volumes. Pour répondre à ces besoins, il existe des StorageClass ressources.

Dans l'exemple ci-dessous, le PVC a été créé dans les fenêtres de l'espace de noms.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ebs-windows-pv-claim
  namespace: windows
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gp2-windows
  resources:
    requests:
      storage: 1Gi
```

Créez le PVC en exécutant la commande suivante :

```
kubectl apply -f persistent-volume-claim.yaml
```

Le manifeste suivant crée un Windows Pod, le configure en VolumeMount tant que `C:\Data` et utilise le PVC comme espace de stockage attaché`C:\Data`.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: windows-server-ltsc2019
  namespace: windows
spec:
  selector:
    matchLabels:
      app: windows-server-ltsc2019
      tier: backend
      track: stable
  replicas: 1
  template:
    metadata:
      labels:
        app: windows-server-ltsc2019
        tier: backend
        track: stable
    spec:
      containers:
```

```
      - name: windows-server-ltsc2019
        image: mcr.microsoft.com/windows/servercore:ltsc2019
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
        volumeMounts:
        - mountPath: "C:\\data"
          name: test-volume
      volumes:
        - name: test-volume
          persistentVolumeClaim:
            claimName: ebs-windows-pv-claim
      nodeSelector:
        kubernetes.io/os: windows
        node.kubernetes.io/windows-build: '10.0.17763'
```

Testez les résultats en accédant au module Windows via PowerShell :

```
kubectl exec -it podname powershell -n windows
```

Dans le Windows Pod, exécutez : `ls`

Sortie :

```
PS C:\> ls


    Directory: C:\


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----          3/8/2021   1:54 PM                data
d-----          3/8/2021   3:37 PM                inetpub
d-r---          1/9/2021   7:26 AM                Program Files
d-----          1/9/2021   7:18 AM                Program Files (x86)
d-r---          1/9/2021   7:28 AM                Users
d-----          3/8/2021   3:36 PM                var
d-----          3/8/2021   3:36 PM                Windows
-a----         12/7/2019   4:20 AM           5510 License.txt
```

Le répertoire de données est fourni par le volume EBS.

Out-of-tree pour Windows

Le code associé aux plugins CSI est livré sous forme de out-of-tree scripts et de binaires qui sont généralement distribués sous forme d'images de conteneur et déployés à l'aide de constructions Kubernetes standard telles que et. DaemonSets StatefulSets Les plugins CSI gèrent un large éventail d'actions de gestion des volumes dans Kubernetes. Les plugins CSI se composent généralement de plugins de nœud (qui s'exécutent sur chaque nœud en tant que DaemonSet) et de plugins de contrôleur.

Les plug-ins de nœuds CSI (en particulier ceux associés à des volumes persistants exposés sous forme de périphériques en mode bloc ou sur un système de fichiers partagé) doivent effectuer diverses opérations privilégiées, telles que l'analyse des périphériques de disque, le montage de systèmes de fichiers, etc. Ces opérations sont différentes pour chaque système d'exploitation hôte. Pour les nœuds de travail Linux, les plugins de nœuds CSI conteneurisés sont généralement déployés en tant que conteneurs privilégiés. Pour les nœuds de travail Windows, les opérations privilégiées pour les plug-ins de nœuds CSI conteneurisés sont prises en charge à l'aide de csi-proxy, un binaire autonome géré par la communauté qui doit être préinstallé sur chaque nœud Windows.

L'AMI Windows optimisée Amazon EKS inclut le proxy CSI à partir d'avril 2022. Les clients peuvent utiliser le pilote CSI SMB sur les nœuds Windows pour accéder au serveur de fichiers Amazon FSx pour Windows, à Amazon FSx pour les partages NetApp ONTAP SMB et à and/or AWS Storage Gateway — File Gateway.

Le blog suivant contient des informations sur la mise en œuvre de la configuration du pilote SMB CSI pour utiliser le serveur de fichiers Amazon FSx pour Windows en tant que stockage persistant pour les modules Windows.

Serveur FSx de fichiers Amazon pour Windows

Une option consiste à utiliser le serveur de fichiers Amazon FSx pour Windows via une fonctionnalité SMB appelée SMB Global Mapping, qui permet de monter un partage SMB sur l'hôte, puis de transmettre les répertoires de ce partage dans un conteneur. Le conteneur n'a pas besoin d'être configuré avec un serveur, un partage, un nom d'utilisateur ou un mot de passe spécifiques. Tout cela est géré sur l'hôte. Le conteneur fonctionnera de la même manière que s'il était entreposé localement.

> Le mappage global des PME est transparent pour l'orchestrateur et il est monté, HostPath ce qui peut impliquer des problèmes de sécurité.

Dans l'exemple ci-dessous, le chemin `G:\Directory\app-state` est un partage SMB sur le nœud Windows.

```
apiVersion: v1
kind: Pod
metadata:
  name: test-fsx
spec:
  containers:
  - name: test-fsx
    image: mcr.microsoft.com/windows/servercore:ltsc2019
    command:
      - powershell.exe
      - -command
      - "Add-WindowsFeature Web-Server; Invoke-WebRequest -UseBasicParsing
 -Uri 'https://dotnetbinaries.blob.core.windows.net/servicemonitor/2.0.1.6/
ServiceMonitor.exe' -OutFile 'C:\\ServiceMonitor.exe'; echo '<html><body><br/
><br/><marquee><H1>Hello EKS!!!<H1><marquee></body><html>' > C:\\inetpub\\wwwroot\
\default.html; C:\\ServiceMonitor.exe 'w3svc'; "
    volumeMounts:
      - mountPath: C:\dotnetapp\app-state
        name: test-mount
  volumes:
    - name: test-mount
      hostPath:
        path: G:\Directory\app-state
        type: Directory
  nodeSelector:
      beta.kubernetes.io/os: windows
      beta.kubernetes.io/arch: amd64
```

Le blog suivant contient des informations sur la mise en œuvre de la configuration d'Amazon FSx pour Windows File Server en tant que stockage persistant pour Windows Pods.

Renforcement des images de conteneurs Windows

Êtes-vous en train de renforcer les images de vos conteneurs Windows ? Au fil des ans, j'ai travaillé avec des clients du monde entier pour les aider à migrer les charges de travail existantes vers des conteneurs, en particulier les charges de travail Windows. Avec plus de 20 ans d'expérience, j'ai vu des entreprises consacrer des efforts et des ressources considérables au renforcement de leurs

serveurs Windows, en mettant en œuvre tout, des benchmarks CIS à la protection antivirus des environnements d'exécution pour protéger les données sensibles.

Cependant, une tendance inquiétante est apparue. À mesure que ces machines virtuelles hautement sécurisées sont modernisées en conteneurs, de nombreuses pratiques de renforcement critiques sont négligées. Les meilleures pratiques de sécurité Windows, qu'il s'agisse de l'image de base (système d'exploitation) ou des services Web tels que IIS, sont souvent négligées, l'accent étant mis uniquement sur la sécurisation de l'hôte du conteneur. Il est essentiel de reconnaître que même si les conteneurs fonctionnent dans des espaces de noms isolés, ils partagent toujours des primitives de noyau avec l'hôte. Les attaquants sont généralement plus intéressés par les mouvements latéraux que par le ciblage direct de l'hôte du conteneur, ce qui leur permet d'exploiter les faibles paramètres de sécurité du conteneur et d'accéder à des données sensibles.

L'objectif de ces documentations est de mettre en évidence quelques paramètres de sécurité essentiels que vous devez implémenter spécifiquement pour les conteneurs Windows hébergeant des `ASP.NET` sites Web sur IIS. Nous nous concentrerons sur quatre domaines clés :

- Politiques de sécurité des comptes
- Politiques d’audit
- Bonnes pratiques en matière de sécurité IIS
- Principe du moindre privilège

Nous allons commencer par découvrir pourquoi chacune de ces configurations de sécurité est essentielle pour protéger vos conteneurs Windows, en examinant les risques spécifiques qu'elles atténuent et les avantages qu'elles offrent en matière de sécurité. Nous allons maintenant passer en revue un extrait de code qui montre comment implémenter correctement ces configurations dans votre Dockerfile, afin de garantir que votre conteneur est renforcé contre les menaces potentielles. Enfin, nous analyserons chaque paramètre en détail, en proposant une explication complète de sa fonction, de son impact sur la sécurité des conteneurs et de la manière dont il contribue à protéger vos applications. Cette approche vous montrera non seulement comment appliquer ces meilleures pratiques, mais vous permettra également de comprendre pourquoi elles sont essentielles au maintien d'une posture de sécurité robuste dans les environnements conteneurisés.

1. Configurer les politiques de compte (mot de passe ou verrouillage) à l'aide des politiques de sécurité locales et du registre

Windows Server Core est une option d'installation minimale disponible dans le cadre de l' [AMI Windows optimisée EKS] (https://docs.aws.amazon.com/eks/latest/userguide/eks- optimized-windows-ami .html). La configuration des politiques de compte (mot de passe ou verrouillage) à l'aide des politiques de sécurité locales et du registre renforce la sécurité du système en appliquant des règles de mot de passe et de verrouillage robustes. Ces politiques obligent les utilisateurs à créer des mots de passe forts d'une longueur et d'une complexité minimales définies, afin de les protéger contre les attaques courantes liées aux mots de passe.

En définissant un âge maximum pour les mots de passe, les utilisateurs sont invités à mettre régulièrement à jour leurs mots de passe, ce qui réduit le risque de compromission de leurs informations d'identification. Les politiques de verrouillage ajoutent un niveau de protection supplémentaire en verrouillant temporairement les comptes après un certain nombre de tentatives de connexion infructueuses, ce qui contribue à prévenir les attaques par force brute. La configuration de ces paramètres via le registre Windows permet aux administrateurs d'appliquer ces mesures de sécurité au niveau du système, garantissant ainsi l'uniformité et la conformité dans l'ensemble de l'organisation. L'application de ces politiques de compte dans un conteneur Windows est essentielle pour garantir la cohérence de la sécurité, même si les conteneurs sont souvent éphémères et destinés à des charges de travail isolées :

Cohérence de sécurité

- Conformité : L'application de politiques de mots de passe et de règles de verrouillage cohérentes dans les conteneurs permet de garantir la conformité en matière de sécurité, en particulier dans les environnements qui nécessitent des contrôles d'accès stricts (par exemple, conformité aux réglementations telles que HIPAA, PCI-DSS).
- Conteneurs renforcés : l'application de ces paramètres garantit que votre conteneur Windows est renforcé contre les accès non autorisés ou les attaques basées sur des mots de passe, en alignant le niveau de sécurité de votre conteneur sur les politiques de sécurité générales du système.

Protection contre les attaques par force brute

- Verrouillage du compte : ces paramètres permettent de se protéger contre les tentatives de connexion par force brute en verrouillant les comptes après un certain nombre de tentatives de

connexion infructueuses. Cela empêche les attaquants d'essayer un nombre illimité de mots de passe.

- Complexité des mots de passe : le fait d'exiger des mots de passe complexes d'une longueur suffisante réduit le risque d'exploitation de mots de passe faibles, même dans des environnements conteneurisés isolés.

Scénarios multi-utilisateurs

- Si votre application conteneurisée est conçue pour gérer plusieurs utilisateurs ou nécessite une authentification utilisateur, l'application de politiques de mot de passe garantit que les comptes utilisateurs du conteneur respectent des règles de sécurité strictes, limitant l'accès aux seuls utilisateurs autorisés.

Conteneurs Windows persistants

- Bien que les conteneurs soient généralement considérés comme éphémères, certains conteneurs Windows peuvent exécuter des services à long terme ou gérer les utilisateurs. Il est donc important d'appliquer des politiques de sécurité appropriées, similaires à celles d'un serveur Windows classique.

Cohérence dans les environnements hybrides

- Si vous utilisez à la fois des machines virtuelles et des conteneurs dans votre infrastructure, l'application des mêmes politiques de sécurité (par exemple, des password/lockout politiques) dans tous les environnements garantit des normes de sécurité uniformes, simplifiant ainsi la gouvernance et la gestion.

En résumé, l'application de ces politiques de compte dans les conteneurs Windows garantit que ceux-ci ne constituent pas un point faible de votre stratégie de sécurité, en vous protégeant contre les attaques par mot de passe et en garantissant la cohérence dans l'ensemble de votre environnement.

Fichier Docker :

```
# Configure account policies for password complexity and lockout
RUN powershell -Command \
      "Write-Output 'Configuring Account Policies (Password/Lockout)...'; \
```

```
NET ACCOUNTS /MINPWLEN:14 /MAXPWAGE:60 /MINPWAGE:14 /LOCKOUTTHRESHOLD:5
```

Explication :

Cette section configure les politiques de compte pour les paramètres de mot de passe et de verrouillage via le registre Windows. Ces politiques contribuent à renforcer la sécurité en contrôlant les exigences relatives aux mots de passe et les seuils de verrouillage des comptes.

1. MinimumPasswordLength (MINPWLEN) = 14 Ce paramètre définit le nombre minimum de caractères pour un mot de passe. La plage est comprise entre 0 et 14 caractères ; la valeur par défaut est de six caractères.
2. MaximumPasswordAge (MAXPWAGE) = 60 Ce paramètre définit le nombre maximal de jours pendant lesquels un mot de passe est valide. Aucune limite n'est spécifiée en utilisant UNLIMITED. /MAXPWAGE ne peut pas être inférieur à /MINPWAGE. La plage est comprise entre 1 et 999 ; la valeur par défaut est de 90 jours
3. Seuil de verrouillage (LOCKOUTTHRESHOLD) = 5 Ce paramètre définit le seuil d'échec des tentatives de connexion. Après 5 tentatives incorrectes, le compte sera verrouillé.

Ces paramètres permettent d'améliorer la sécurité des mots de passe et de prévenir les attaques par force brute en appliquant des politiques de mots de passe strictes et en verrouillant les comptes après un certain nombre de tentatives de connexion infructueuses.

2. Politiques d’audit

Les politiques d'audit sont importantes pour les conteneurs Windows car elles fournissent une visibilité essentielle sur les événements de sécurité, tels que les tentatives de connexion et l'utilisation de privilèges, aidant à détecter les accès non autorisés, à surveiller l'activité des utilisateurs et à garantir le respect des normes réglementaires. Même dans la nature éphémère des conteneurs, les journaux d'audit sont essentiels pour l'investigation des incidents, la détection proactive des menaces et le maintien d'une posture de sécurité cohérente dans les environnements conteneurisés.

Surveillance de la sécurité et conformité :

- Suivi des activités des utilisateurs : les politiques d'audit permettent aux administrateurs de surveiller les activités des utilisateurs, telles que les tentatives de connexion et l'utilisation des privilèges, au sein du conteneur. Cela est essentiel pour détecter les accès non autorisés ou les comportements suspects.

- Conformité réglementaire : de nombreuses organisations sont tenues de consigner les événements de sécurité pour se conformer aux réglementations telles que la loi HIPAA, la norme PCI-DSS et le RGPD. L'activation des politiques d'audit dans les conteneurs garantit le respect de ces exigences, même dans les environnements conteneurisés.

Enquête sur l'incident :

- Criminalistique et analyse : si une application ou un service conteneurisé est compromis, les journaux d'audit peuvent fournir des informations précieuses pour l'analyse post-incident. Ils aident les équipes de sécurité à suivre les actions entreprises par les attaquants ou à identifier la manière dont une violation s'est produite.
- Détection en temps réel : les journaux d'audit permettent aux administrateurs de configurer des alertes en temps réel pour les événements critiques (par exemple, tentatives de connexion infructueuses, augmentation des privilèges). Cette surveillance proactive permet de détecter les attaques à un stade précoce et d'accélérer les temps de réponse.

Cohérence entre les environnements :

- Posture de sécurité uniforme : en appliquant des politiques d'audit dans les conteneurs via le registre, vous garantissez des pratiques de sécurité cohérentes dans les environnements conteneurisés et non conteneurisés. Cela permet d'éviter que les conteneurs ne deviennent un angle mort pour la surveillance de la sécurité.
- Visibilité dans les environnements hybrides : pour les entreprises qui utilisent à la fois des serveurs et des conteneurs Windows traditionnels, les politiques d'audit fournissent une visibilité et un contrôle similaires sur toutes les plateformes, ce qui rend la gestion plus facile et plus efficace.

Suivi des opérations privilégiées :

- Audit de l'utilisation des privilèges : dans les environnements de conteneurs où les applications s'exécutent avec des privilèges élevés ou dans lesquels des tâches administratives sont effectuées, l'audit des opérations privilégiées garantit la responsabilisation. Vous pouvez enregistrer les personnes qui ont accédé à des ressources sensibles ou effectué des tâches critiques à l'intérieur du conteneur.
- Prévenir les abus de privilèges : en surveillant l'utilisation des privilèges, vous pouvez détecter les cas où des utilisateurs non autorisés tentent d'augmenter leurs privilèges ou d'accéder à des zones restreintes au sein du conteneur, ce qui permet de prévenir les attaques internes ou externes.

Détection des tentatives d'accès non autorisées :

- Tentatives d'ouverture de session infructueuses : l'activation de politiques d'audit pour les tentatives de connexion infructueuses permet d'identifier les attaques par force brute ou les tentatives non autorisées d'accès à des applications conteneurisées. Cela permet de savoir qui essaie d'accéder au système et à quelle fréquence.
- Surveillance du verrouillage des comptes : l'audit des événements de verrouillage des comptes permet aux administrateurs de détecter et d'étudier les blocages potentiels causés par des activités suspectes ou malveillantes.

Sécurité permanente, même dans des environnements éphémères :

- Éphémère mais sécurisé : bien que les conteneurs soient éphémères, ce qui signifie qu'ils peuvent être supprimés et recréés fréquemment, l'audit joue toujours un rôle clé pour garantir que les événements de sécurité sont capturés pendant le fonctionnement du conteneur. Cela garantit que les événements de sécurité critiques sont enregistrés pendant toute la durée du cycle de vie du conteneur.

Journalisation centralisée :

- Transfert des journaux vers des systèmes centralisés : les conteneurs peuvent être intégrés à des systèmes de journalisation centralisés (par exemple, ELK stack, AWS CloudWatch) pour capturer les journaux d'audit provenant de plusieurs instances de conteneurs. Cela permet une meilleure analyse et une meilleure corrélation des événements de sécurité dans l'ensemble de votre infrastructure.

Fichier Docker :

```
# Configure audit policies for logging security events
RUN powershell -Command \
    "Write-Host 'Configuring Audit Policy..'; \
    Set-ItemProperty -Path 'HKLM:\\SYSTEM\\CurrentControlSet\\Control\\Lsa' -Name
 'SCENoApplyLegacyAuditPolicy' -Value 0; \
    auditpol /set /category:"Logon/Logoff" /subcategory:"Logon" /failure:enable

# Creates STDOUT on Windows Containers (check GitHub LogMonitor:: https://github.com/
microsoft/windows-container-tools/blob/main/LogMonitor/README.md)
COPY LogMonitor.exe LogMonitorConfig.json 'C:\\LogMonitor\\'
```

```
WORKDIR /LogMonitor
```

Explication :

Cette section configure les politiques d'audit en utilisant les modifications du registre. Les politiques d'audit contrôlent les événements de sécurité enregistrés par Windows, ce qui permet de surveiller et de détecter les tentatives d'accès non autorisées.

1. SCENoApplyLegacyAuditPolicy = 0 Cela désactive le format de stratégie d'audit existant, permettant ainsi des politiques d'audit plus détaillées introduites dans les versions ultérieures de Windows. Cela est important pour les configurations d'audit modernes.
2. Sous-catégorie Auditpol : « Connexion » Ce paramètre permet d'auditer les événements de connexion réussis et d'échec. La valeur 3 signifie que Windows enregistrera à la fois les tentatives de connexion réussies et échouées. Cela permet de contrôler qui accède au système et de détecter les tentatives de connexion infructueuses.

Ces politiques d'audit sont essentielles à la surveillance de la sécurité et à la conformité, car elles fournissent des journaux détaillés des événements de sécurité importants tels que les tentatives de connexion et l'utilisation d'opérations privilégiées.

3. Bonnes pratiques de sécurité IIS pour les conteneurs Windows

La mise en œuvre des meilleures pratiques IIS dans les conteneurs Windows est importante pour plusieurs raisons, afin de garantir la sécurité, les performances et l'évolutivité de vos applications. Bien que les conteneurs fournissent une isolation et un environnement léger, ils nécessitent tout de même une configuration appropriée pour éviter les vulnérabilités et les problèmes opérationnels. Voici pourquoi il est essentiel de suivre les meilleures pratiques pour IIS dans les conteneurs Windows :

Sécurité

- Prévention des vulnérabilités courantes : IIS est souvent la cible d'attaques telles que le cross-site scripting (XSS), le clickjacking et la divulgation d'informations. La mise en œuvre d'en-têtes de sécurité (par exemple X-Content-Type-Options X-Frame-Options,, et Strict-Transport-Security) permet de protéger votre application contre ces menaces.
- L'isolation ne suffit pas : les conteneurs sont isolés, mais une instance IIS mal configurée peut exposer des informations sensibles, telles que les détails de la version du serveur, les listes de répertoires ou les communications non chiffrées. En désactivant des fonctionnalités telles que

la navigation dans les annuaires et en supprimant l'en-tête de la version IIS, vous minimisez la surface d'attaque.

- Chiffrement et HTTPS : les meilleures pratiques, telles que le renforcement des connexions HTTPS uniquement, garantissent que les données en transit sont cryptées, protégeant ainsi les informations sensibles contre toute interception.

Performances

- Utilisation efficace des ressources : les meilleures pratiques d'IIS, telles que l'activation de la compression dynamique et statique, réduisent l'utilisation de la bande passante et améliorent les temps de chargement. Ces optimisations sont particulièrement importantes dans les environnements conteneurisés, où les ressources sont partagées entre les conteneurs et le système hôte.
- Journalisation optimisée : une configuration correcte de la journalisation (par exemple, en incluant l' X-Forwarded-Foren-tête) garantit que vous pouvez suivre l'activité des clients tout en minimisant les frais de journalisation inutiles. Cela vous permet de recueillir des données pertinentes pour le dépannage sans dégrader les performances.

Scalabilité et maintenabilité

- Cohérence entre les environnements : en suivant les meilleures pratiques, vous vous assurez que votre configuration IIS est cohérente sur plusieurs instances de conteneur. Cela simplifie le dimensionnement et garantit que lorsque de nouveaux conteneurs sont déployés, ils respectent les mêmes directives de sécurité et de performance.
- Configurations automatisées : les meilleures pratiques en matière de Dockerfiles, telles que la définition des autorisations de dossier et la désactivation des fonctionnalités inutiles, garantissent que chaque nouveau conteneur est automatiquement configuré correctement. Cela réduit les interventions manuelles et réduit le risque d'erreur humaine.

Conformité

- Respect des exigences réglementaires : de nombreux secteurs ont des exigences réglementaires strictes (par exemple, PCI-DSS, HIPAA) qui imposent des mesures de sécurité spécifiques, telles que les communications cryptées (HTTPS) et l'enregistrement des demandes des clients. Le respect des meilleures pratiques d'IIS dans les conteneurs permet de garantir la conformité à ces normes.

- Auditabilité : La mise en œuvre de politiques d'audit et de journalisation sécurisée permet la traçabilité des événements, essentielle dans les audits. Par exemple, la journalisation de l' X-Forwarded-Foren-tête garantit que les adresses IP des clients sont correctement enregistrées dans les architectures basées sur un proxy.

Minimiser les risques dans les environnements partagés

- Éviter les erreurs de configuration : les conteneurs partagent le noyau de l'hôte et, bien qu'ils soient isolés les uns des autres, une instance IIS mal configurée peut révéler des vulnérabilités ou créer des problèmes de performance. Les meilleures pratiques garantissent le fonctionnement optimal de chaque instance IIS, réduisant ainsi le risque de problèmes entre conteneurs.
- Accès au moindre privilège : la définition des autorisations appropriées pour les dossiers et les fichiers du conteneur (par exemple, en utilisant Set-Acl in PowerShell) garantit que les utilisateurs et les processus du conteneur n'ont que l'accès nécessaire, réduisant ainsi le risque d'augmentation des privilèges ou de falsification des données.

Résilience dans des environnements éphémères

- Nature éphémère des contenants : Les contenants ont souvent une courte durée de vie et sont souvent reconstruits. L'application des meilleures pratiques IIS garantit que chaque conteneur est configuré de manière sécurisée et cohérente, quel que soit le nombre de redéploiements. Cela permet d'éviter l'introduction de mauvaises configurations au fil du temps.
- Atténuer les erreurs de configuration potentielles : en appliquant automatiquement les meilleures pratiques (par exemple, en désactivant les protocoles ou les en-têtes faibles), le risque d'une mauvaise configuration lors des redémarrages ou des mises à jour des conteneurs est minimisé.

Fichier Docker :

```
# Enforce HTTPS (disable HTTP) -- Only if container is target for SSL termination
RUN powershell -Command \
    "$httpBinding = Get-WebBinding -Name 'Default Web Site' -Protocol http | Where-
Object { $_.bindingInformation -eq '*:80:' }; \
    if ($httpBinding) { Remove-WebBinding -Name 'Default Web Site' -Protocol http -Port
 80; } \
    $httpsBinding = Get-WebBinding -Name 'Default Web Site' -Protocol https | Where-
Object { $_.bindingInformation -eq '*:443:' }; \
```

```
    if (-not $httpsBinding) { New-WebBinding -Name 'Default Web Site' -Protocol https -
Port 443 -IPAddress '*'; }"

# Use secure headers
RUN powershell -Command \
    "Write-Host 'Adding security headers...'; \
    Add-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 'system.applicationHost/sites/siteDefaults/logFile/customFields' -name
 "." -value @{logFieldName='X-Forwarded-For';sourceName='X-Forwarded-
For';sourceType='RequestHeader'}; \
    Add-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/httpProtocol/customHeaders" -name "." -value @{name='Strict-
Transport-Security';value='max-age=31536000; includeSubDomains'}; \
    Add-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/httpProtocol/customHeaders" -name "." -value @{name='X-Content-Type-
Options';value='nosniff'}; \
    Add-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/httpProtocol/customHeaders" -name "." -value @{name='X-XSS-
Protection';value='1; mode=block'}; \
    Add-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/httpProtocol/customHeaders" -name "." -value @{name='X-Frame-
Options';value='DENY'};"

# Disable IIS version disclosure
RUN powershell -Command \
    "Write-Host 'Disabling IIS version disclosure...'; \
    Import-Module WebAdministration; \
    Set-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/security/requestFiltering" -name "removeServerHeader" -value
 "true";"

# Set IIS Logging Best Practices
RUN powershell -Command \
    Set-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/directoryBrowse" -name "enabled" -value "false"; \
    Set-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/httpErrors" -name "existingResponse" -value "PassThrough"; \

# Enable IIS dynamic and static compression to optimize performance
RUN powershell -Command \
    "Write-Host 'Enabling IIS compression...'; \
    Enable-WindowsOptionalFeature -Online -FeatureName IIS-HttpCompressionDynamic; \
    Import-Module WebAdministration; \
```

```
    Set-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/urlCompression" -name "doDynamicCompression" -value "true"; \
    Set-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/urlCompression" -name "doStaticCompression" -value "true"

# Ensure proper folder permissions using PowerShell's Set-Acl

RUN powershell -Command \
    "Write-Host 'Setting folder permissions for IIS...'; \
    $path = 'C:\\inetpub\\wwwroot'; \
    $acl = Get-Acl $path; \
    $iusr = New-Object System.Security.Principal.NTAccount('IIS_IUSRS'); \
    $rule = New-Object System.Security.AccessControl.FileSystemAccessRule($iusr,
 'ReadAndExecute', 'ContainerInherit, ObjectInherit', 'None', 'Allow'); \
    $acl.SetAccessRule($rule); \
    $users = New-Object System.Security.Principal.NTAccount('Users'); \
    $rule2 = New-Object System.Security.AccessControl.FileSystemAccessRule($users,
 'ReadAndExecute', 'ContainerInherit, ObjectInherit', 'None', 'Allow'); \
    $acl.SetAccessRule($rule2); \
    Set-Acl -Path $path -AclObject $acl"
```

Explication :

Cette commande configure IIS pour enregistrer l' X-Forwarded-Foren-tête, qui est couramment utilisé pour capturer l'adresse IP du client d'origine lorsqu'une demande passe par un proxy ou un équilibreur de charge. Par défaut, IIS enregistre uniquement l'adresse IP de l'équilibreur de charge ou du proxy inverse. L'ajout de ce champ de journal personnalisé permet de suivre la véritable adresse IP du client à des fins d'audit de sécurité, d'analyse et de résolution des problèmes.

1. X-Forwarded-For en-tête couramment utilisé pour capturer l'adresse IP d'origine du client lorsqu'une demande passe par un proxy ou un équilibreur de charge. Par défaut, IIS enregistre uniquement l'adresse IP de l'équilibreur de charge ou du proxy inverse. L'ajout de ce champ de journal personnalisé permet de suivre la véritable adresse IP du client à des fins d'audit de sécurité, d'analyse et de résolution des problèmes.
2. Strict-Transport-Security (HSTS) Garantit que les navigateurs communiquent uniquement via HTTPS. Le max-age=31536000 indique que cette politique est appliquée pendant 1 an et qu'elle includeSubDomains s'applique à tous les sous-domaines.
3. X-Content-Type-Options empêche les navigateurs de « renifler par MIME » une réponse différente du type de contenu déclaré. Cela permet de prévenir certains types d'attaques.
4. X-XSS-Protection active la protection par script intersite (XSS) dans les navigateurs.

5. X-Frame-Options empêche l'intégration de la page dans les iframes, protégeant ainsi contre les attaques de clickjacking.
6. Désactiver la divulgation de la version d'IIS Cette commande désactive l'en-tête du serveur dans les réponses HTTP, qui indique par défaut la version d'IIS utilisée. Le masquage de ces informations permet de réduire le risque que des attaquants identifient et ciblent des vulnérabilités spécifiques à la version IIS.
7. Activer les connexions HTTPS uniquement Cette section (commentée) applique les connexions HTTPS et désactive le HTTP. S'il n'est pas commenté, le Dockerfile configurera IIS pour écouter uniquement sur le port 443 (HTTPS) et supprimera la liaison HTTP par défaut sur le port 80. Cela est utile lorsque vous mettez fin au protocole SSL à l'intérieur du conteneur et garantit que tout le trafic est crypté.
8. Désactiver la navigation dans les répertoires Empêche IIS d'afficher une liste de répertoires lorsqu'aucun document par défaut n'est présent. Cela évite d'exposer la structure interne des fichiers aux utilisateurs.
9. Le transfert des pages d'erreur personnalisées garantit que si l'application dispose de sa propre gestion des erreurs, IIS laissera passer les pages d'erreur de l'application au lieu d'afficher les pages d'erreur IIS par défaut.
10. Le mode d'erreur détaillé configure IIS pour afficher des messages d'erreur détaillés pour les demandes locales uniquement, ce qui aide les développeurs à diagnostiquer les problèmes sans exposer des informations sensibles aux utilisateurs externes.
11. Garantir des autorisations de dossier appropriées Ce bloc configure les autorisations de dossier pour la racine Web IIS (C:\inetpub\wwwroot). Il définit les autorisations de lecture et d'exécution pour les groupes IIS_IUSRS et Users, garantissant que ces utilisateurs peuvent accéder au dossier mais pas modifier les fichiers. La définition des autorisations appropriées minimise le risque d'accès non autorisé ou de falsification des fichiers hébergés par le serveur Web.

Le respect des meilleures pratiques IIS dans les conteneurs Windows garantit la sécurité, les performances et l'évolutivité de vos applications conteneurisées. Ces pratiques permettent de prévenir les vulnérabilités, d'optimiser l'utilisation des ressources, de garantir la conformité et de maintenir la cohérence entre les instances de conteneur. Même si les conteneurs sont conçus pour être isolés, une configuration appropriée est nécessaire pour minimiser les risques et garantir la fiabilité de votre application dans les environnements de production.

4. Principe du moindre privilège

Le principe du moindre privilège (PoLP) est crucial pour les conteneurs Windows pour plusieurs raisons importantes, notamment pour améliorer la sécurité et minimiser les risques dans les environnements conteneurisés. Selon ce principe, un système ou une application doit fonctionner avec le niveau minimum d'autorisations nécessaires pour fonctionner correctement. Voici pourquoi c'est important dans les conteneurs Windows :

Minimiser la surface d'attaque

- Les conteneurs exécutent souvent des applications qui interagissent avec différents composants du système, et plus une application possède de privilèges, plus son accès à ces composants est large. En limitant les autorisations du conteneur à ce qui est nécessaire, PolP réduit considérablement la surface d'attaque, ce qui rend plus difficile pour un attaquant d'exploiter le conteneur s'il est compromis.

Limiter l'impact des conteneurs compromis

- Si un conteneur Windows est compromis, l'exécution d'applications dotées de privilèges excessifs (par exemple, accès administrateur ou root) peut permettre à un attaquant de prendre le contrôle de fichiers système critiques ou d'augmenter les privilèges sur l'hôte du conteneur. En appliquant le protocole PolP, même si un conteneur est piraté, l'attaquant est limité dans ce qu'il peut faire, ce qui empêche toute nouvelle escalade et l'accès à des ressources sensibles ou à d'autres conteneurs.

Protection dans les environnements multilocataires

- Dans les environnements cloud ou d'entreprise, plusieurs conteneurs peuvent être exécutés sur la même infrastructure physique ou virtuelle. PoLP garantit qu'un conteneur compromis n'a pas la capacité d'accéder aux ressources ou aux données appartenant à d'autres locataires. Cette isolation est essentielle au maintien de la sécurité dans les environnements partagés à locataires multiples et à la protection contre les mouvements latéraux entre les conteneurs.

Atténuer l'escalade des privilèges

- Les conteneurs qui s'exécutent avec des privilèges élevés peuvent être utilisés par des attaquants pour augmenter les privilèges au sein du système. PoLP atténue ce risque en limitant l'accès

du conteneur aux ressources du système, empêchant ainsi les actions non autorisées ou l'augmentation des privilèges au-delà de l'environnement du conteneur.

Conformité et audit

- De nombreuses normes réglementaires et cadres de sécurité (par exemple, PCI DSS, HIPAA, RGPD) exigent que les systèmes adhèrent à la PolP afin de limiter l'accès aux données sensibles. L'exécution de conteneurs Windows avec des privilèges restreints aide les entreprises à se conformer à ces réglementations et garantit que les applications n'ont accès qu'aux ressources dont elles ont spécifiquement besoin.

Réduction du risque de mauvaise configuration

- Lorsque les conteneurs s'exécutent avec des privilèges inutiles, même une erreur de configuration mineure peut entraîner de graves failles de sécurité. Par exemple, si un conteneur exécuté en tant qu'administrateur est accidentellement exposé à Internet, un attaquant pourrait prendre le contrôle du système. PoLP aide à prévenir de tels risques en utilisant par défaut des privilèges limités, ce qui réduit le danger des erreurs de configuration.

Posture de sécurité des conteneurs améliorée

- En suivant PoLP, les conteneurs sont mieux isolés du système hôte sous-jacent et les uns des autres. Cela garantit que l'application conteneurisée est moins susceptible d'accéder aux fichiers ou aux processus système ou de les modifier en dehors de son périmètre défini, préservant ainsi l'intégrité du système d'exploitation hôte et des autres charges de travail.

Fichier Docker :

```
# Strongly recommended that when deploying a Windows server container to any multi-
tenant environment that your application runs via the ContainerUser account
USER ContainerUser
```

Explication :

Dans cette section, la ContainerUser commande USER indique que l'application contenue dans le conteneur Windows doit s'exécuter sous le ContainerUser compte plutôt que sous le compte administrateur par défaut.

Voici pourquoi cela est important, en particulier dans un environnement mutualisé :

1. Principe du moindre privilège : le ContainerUser compte est un utilisateur non administratif doté de privilèges limités. L'exécution de l'application sous ce compte respecte le principe du moindre privilège, ce qui permet de minimiser le risque d'exploitation. Si un attaquant compromettait l'application, il aurait un accès limité au système, réduisant ainsi les dommages potentiels.
2. Sécurité renforcée : dans les environnements multilocataires, les conteneurs peuvent partager la même infrastructure sous-jacente. Son exécution ContainerUser garantit que même si un conteneur est compromis, il ne disposera pas de privilèges administratifs lui permettant d'accéder aux fichiers système critiques ou à d'autres conteneurs ou de les modifier. Cela réduit considérablement la surface d'attaque.
3. Éviter l'accès root : par défaut, les conteneurs peuvent fonctionner avec des autorisations élevées (comme l'accès root dans les conteneurs Linux), ce qui peut être dangereux en cas d'exploitation. Son utilisation ContainerUser garantit que l'application ne s'exécute pas avec des droits administratifs inutiles, ce qui rend plus difficile pour les attaquants d'augmenter les privilèges.
4. Bonnes pratiques pour les environnements à locataires multiples : dans les environnements où plusieurs utilisateurs ou organisations partagent la même infrastructure (dans le cloud, par exemple), la sécurité est essentielle. L'exécution d'applications avec des autorisations restreintes empêche l'application d'un client d'affecter les autres, protégeant ainsi les données et les ressources sensibles sur l'ensemble de la plateforme.

La ContainerUser commande USER garantit que l'application s'exécute avec des privilèges minimaux, renforçant ainsi la sécurité dans les environnements multilocataires en limitant les dommages qui pourraient être causés si le conteneur était compromis. Il s'agit d'une bonne pratique pour empêcher tout accès non autorisé ou toute augmentation de privilèges dans un environnement conteneurisé.

Le principe du moindre privilège est essentiel pour les conteneurs Windows car il limite l'impact potentiel des failles de sécurité, réduit la surface d'attaque et empêche l'accès non autorisé aux composants critiques du système. En exécutant des applications conteneurisées avec uniquement les autorisations nécessaires, les entreprises peuvent améliorer de manière significative la sécurité et la stabilité de leurs environnements de conteneurs, en particulier dans les infrastructures mutualisées et partagées.

Réflexions finales : pourquoi la sécurisation de vos conteneurs Windows est indispensable dans le contexte actuel des menaces

Dans le monde numérique en évolution rapide d'aujourd'hui, où les menaces sont de plus en plus sophistiquées et abondantes, la sécurisation de vos conteneurs Windows n'est pas simplement une recommandation, c'est une nécessité absolue. Les conteneurs constituent un moyen léger et flexible d'empaqueter et de déployer des applications, mais ils ne sont pas à l'abri des failles de sécurité. À mesure que de plus en plus d'entreprises adoptent les conteneurs pour rationaliser leur infrastructure, ils deviennent également une cible potentielle de cyberattaques s'ils ne sont pas correctement sécurisés.

Internet est inondé de menaces diverses, qu'il s'agisse d'acteurs malveillants ciblant des vulnérabilités non corrigées ou de robots automatisés qui détectent les erreurs de configuration. Si les mesures de sécurité appropriées ne sont pas mises en place, les conteneurs peuvent être exploités pour exposer des données sensibles, augmenter les privilèges ou servir de points d'entrée à des attaques susceptibles de compromettre l'ensemble de votre infrastructure. La sécurité des conteneurs est donc aussi essentielle que la sécurisation de toute autre partie de votre environnement.

Lorsque vous utilisez des conteneurs Windows, de nombreuses bonnes pratiques de sécurité traditionnelles s'appliquent toujours. La mise en œuvre de politiques de compte robustes, la sécurisation des configurations IIS, l'application du protocole HTTPS, l'utilisation de règles de pare-feu strictes et l'application du moindre privilège d'accès aux fichiers critiques sont autant de mesures clés qui garantissent la résilience du conteneur face aux attaques. En outre, des audits et des journaux réguliers fournissent une visibilité sur ce qui se passe à l'intérieur du conteneur, ce qui vous permet de détecter les activités suspectes avant qu'elles ne se transforment en véritable incident.

La sécurisation des conteneurs Windows est également conforme aux exigences réglementaires qui imposent de protéger les données sensibles et de garantir l'intégrité des applications. À mesure que les architectures natives du cloud et conteneurisées gagnent en popularité, garantir la sécurité à chaque niveau, de l'image de base au conteneur en cours d'exécution, vous aidera à protéger vos opérations et à conserver la confiance des clients.

En résumé, l'essor des applications conteneurisées, associé au nombre croissant de cybermenaces, fait de la sécurité des conteneurs un aspect non négociable de la gestion des infrastructures modernes. En adhérant aux meilleures pratiques et en surveillant en permanence les vulnérabilités, les entreprises peuvent profiter de l'agilité et de l'efficacité des conteneurs Windows sans

compromettre la sécurité. Dans cet environnement riche en menaces, la sécurisation de vos conteneurs Windows n'est pas une simple option, c'est une nécessité.

Bonnes pratiques pour les déploiements hybrides

Ce guide fournit des conseils sur l'exécution de déploiements dans des environnements sur site ou de périphérie avec EKS Hybrid Nodes ou EKS Anywhere.

Nous avons actuellement publié des guides sur les sujets suivants :

- Meilleures pratiques pour les nœuds hybrides EKS et les déconnexions réseau

Nœuds hybrides EKS et déconnexions réseau

L'architecture EKS Hybrid Nodes peut être une nouveauté pour les clients habitués à gérer des clusters Kubernetes locaux entièrement dans leurs propres centres de données ou emplacements périphériques. Avec les nœuds hybrides EKS, le plan de contrôle Kubernetes s'exécute dans une région AWS et seuls les nœuds s'exécutent sur site, ce qui permet d'obtenir une architecture de cluster Kubernetes « étendue » ou « étendue ».

Cela conduit à une question courante : « Que se passe-t-il si mes nœuds sont déconnectés du plan de contrôle Kubernetes ? »

Dans ce guide, nous répondons à cette question en passant en revue les sujets suivants. Il est recommandé de valider la stabilité et la fiabilité de vos applications par le biais de déconnexions réseau, car chaque application peut se comporter différemment en fonction de ses dépendances, de sa configuration et de son environnement. Consultez le eks-hybrid-examples GitHub référentiel aws-samples/ pour connaître la configuration, les procédures et les résultats des tests auxquels vous pouvez vous référer pour tester les déconnexions réseau avec les nœuds hybrides EKS et vos propres applications. Le GitHub dépôt contient également des détails supplémentaires sur les tests utilisés pour valider le comportement expliqué dans ce guide.

- Meilleures pratiques en matière de stabilité grâce aux déconnexions du réseau
- Comportement de basculement du pod Kubernetes lors de déconnexions réseau
- Trafic réseau des applications via des déconnexions réseau
- Informations d'identification de l'hôte via des déconnexions réseau

Meilleures pratiques en matière de stabilité grâce aux déconnexions du réseau

Réseau à haute disponibilité

La meilleure approche pour éviter les déconnexions réseau entre les nœuds hybrides et le plan de contrôle Kubernetes consiste à utiliser des connexions redondantes et résilientes entre votre environnement sur site et AWS. Reportez-vous au kit de résilience AWS Direct Connect et à la documentation AWS Site-to-Site VPN pour plus d'informations sur l'architecture de réseaux hybrides à haute disponibilité avec ces solutions.

Applications à haute disponibilité

Lorsque vous concevez l'architecture des applications, tenez compte de vos domaines de défaillance et des effets des différents types de pannes. Kubernetes fournit des mécanismes intégrés pour déployer et gérer des répliques d'applications sur des domaines de nœuds, de zones et régionaux. L'utilisation de ces mécanismes dépend de l'architecture de votre application, de vos environnements et de vos exigences de disponibilité. Par exemple, les applications sans état peuvent souvent être déployées avec plusieurs répliques et peuvent être déplacées entre des hôtes et des capacités d'infrastructure arbitraires, et vous pouvez utiliser des sélecteurs de nœuds et des contraintes de répartition topologique pour exécuter des instances de l'application dans différents domaines. Pour plus de détails sur les techniques au niveau des applications permettant de créer des applications résilientes sur Kubernetes, reportez-vous au guide des meilleures pratiques EKS.

Kubernetes évalue les informations zonales relatives aux nœuds déconnectés du plan de contrôle Kubernetes pour déterminer s'il convient de déplacer les pods vers d'autres nœuds. Si tous les nœuds d'une zone sont inaccessibles, Kubernetes annule les expulsions de pods pour les nœuds de cette zone. La meilleure pratique consiste à attribuer une zone à chaque nœud en fonction de son centre de données ou de son emplacement physique si vous effectuez un déploiement avec des nœuds exécutés dans plusieurs centres de données ou emplacements physiques. Lorsque vous exécutez EKS avec des nœuds dans le cloud, cette étiquette de zone est automatiquement appliquée par AWS cloud-controller-manager. Cependant, a n' cloud-controller-managerest pas utilisé avec les nœuds hybrides, vous pouvez donc transmettre ces informations via votre configuration kubelet. Vous trouverez ci-dessous un exemple de configuration d'une zone dans la configuration de votre nœud pour les nœuds hybrides. La configuration est transmise lorsque vous connectez vos nœuds hybrides à votre cluster à l'aide de la CLI (`nodeadm`) des nœuds hybrides. Pour plus d'informations sur l'`topology.kubernetes.io/zone`étiquette, consultez la documentation de Kubernetes. Pour

plus d'informations sur la CLI des nœuds hybrides, consultez la référence nodeadm sur les nœuds hybrides.

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    region: my-region
  kubelet:
    flags:
       - --node-labels=topology.kubernetes.io/zone=dc1
  hybrid:
    ...
```

Surveillance réseau

Si vous utilisez AWS Direct Connect ou AWS Site-to-Site VPN pour votre connectivité hybride, vous pouvez tirer parti des CloudWatch alarmes, des journaux et des mesures pour observer l'état de votre connexion hybride et diagnostiquer les problèmes. Pour plus d'informations, consultez Surveillance des ressources AWS Direct Connect et Surveillance d'une connexion Site-to-Site VPN AWS.

Il est recommandé de créer des alarmes pour les `NodeNotReady` événements signalés par l' node-lifecycle-controllerexécution sur le plan de contrôle EKS, qui signalent qu'un nœud hybride est peut-être en train de se déconnecter du réseau. Vous pouvez créer cette alarme en activant la journalisation du plan de contrôle EKS pour le Controller Manager et en créant un filtre métrique CloudWatch pour le message « Enregistrement du message d'événement de changement de statut pour le nœud » avec le status = « NodeNotReady ». Après avoir créé un filtre métrique, vous pouvez créer une alarme pour ce filtre en fonction des seuils souhaités. Pour plus d'informations, consultez la section Alarme relative aux journaux dans la CloudWatch documentation.

Vous pouvez utiliser les métriques intégrées Transit Gateway (TGW) et Virtual Private Gateway (VGW) pour observer le trafic réseau entrant et sortant de votre TGW ou VGW. Vous pouvez créer des alarmes pour ces métriques afin de détecter les scénarios dans lesquels le trafic réseau chute en dessous des niveaux normaux, indiquant un problème réseau potentiel entre les nœuds hybrides et le plan de contrôle EKS. Les métriques TGW et VGW sont décrites dans le tableau suivant.

Passerelle	Métrique	Description
Passerelle de transit	BytesIn	Les octets reçus par TGW depuis la pièce jointe (plan de contrôle EKS vers nœuds hybrides)
Passerelle de transit	BytesOut	Les octets envoyés par TGW à la pièce jointe (nœuds hybrides vers le plan de contrôle EKS)
Passerelle privée virtuelle	TunnelDataIn	Les octets envoyés depuis le côté AWS de la connexion via le tunnel VPN vers la passerelle client (plan de contrôle EKS vers les nœuds hybrides)
Passerelle privée virtuelle	TunnelDataOut	Les octets reçus du côté AWS de la connexion via le tunnel VPN depuis la passerelle client (nœuds hybrides vers le plan de contrôle EKS)

Vous pouvez également utiliser CloudWatch Network Monitor pour mieux comprendre vos connexions hybrides afin de réduire le temps moyen de restauration et de déterminer si les problèmes de réseau proviennent d'AWS ou de votre environnement. CloudWatch Network Monitor peut être utilisé pour visualiser la perte de paquets et la latence dans vos connexions réseau hybrides, définir des alertes et des seuils, puis prendre des mesures pour améliorer les performances de votre réseau. Pour plus d'informations, consultez la section Utilisation d'Amazon CloudWatch Network Monitor.

EKS propose plusieurs options pour surveiller l'état de santé de vos clusters et applications. Pour l'état du cluster, vous pouvez utiliser le tableau de bord d'observabilité de la console EKS pour détecter, résoudre et résoudre rapidement les problèmes. Vous pouvez également utiliser Amazon Managed Service pour Prometheus, AWS Distro for Open Telemetry (ADOT) et pour la surveillance des clusters CloudWatch , des applications et des infrastructures. Pour plus d'informations sur les options d'observabilité d'EKS, voir Surveiller les performances de votre cluster et consulter les journaux.

Dépannage local

Pour vous préparer aux déconnexions réseau entre les nœuds hybrides et le plan de contrôle EKS, vous pouvez configurer des backends de surveillance et de journalisation secondaires afin de maintenir l'observabilité des applications lorsque les services AWS régionaux ne sont pas accessibles. Par exemple, vous pouvez configurer le collecteur AWS Distro for Open Telemetry

(ADOT) pour envoyer des métriques et des journaux à plusieurs backends. Vous pouvez également utiliser des outils locaux, tels que la `crictl` CLI, pour interagir localement avec les pods et les conteneurs en remplacement d'autres clients compatibles avec l'API Kubernetes qui interrogent généralement le point de terminaison du serveur d'API Kubernetes. `kubectl` Pour plus d'informations`crictl`, consultez la `crictl`documentation dans les cri-tools GitHub. Quelques `crictl` commandes utiles sont répertoriées ci-dessous.

Répertoriez les pods exécutés sur l'hôte :

```
crictl pods
```

Répertoriez les conteneurs exécutés sur l'hôte :

```
crictl ps
```

Répertoriez les images exécutées sur l'hôte :

```
crictl images
```

Obtenez les journaux d'un conteneur en cours d'exécution sur l'hôte :

```
crictl logs CONTAINER_NAME
```

Obtenez des statistiques sur les pods exécutés sur l'hôte :

```
crictl statsp
```

Trafic réseau des applications

Lorsque vous utilisez des nœuds hybrides, il est important de prendre en compte et de comprendre les flux réseau du trafic de vos applications et les technologies que vous utilisez pour exposer vos applications en externe à votre cluster. Les différentes technologies d'équilibrage de charge et d'entrée des applications se comportent différemment lors des déconnexions du réseau. Par exemple, si vous utilisez la fonctionnalité BGP Control Plane de Cilium pour l'équilibrage de charge des applications, la session BGP de vos pods et services peut être interrompue lors des déconnexions du réseau. Cela se produit parce que la fonctionnalité du haut-parleur BGP est intégrée à l'agent Cilium, qui redémarre en permanence lorsqu'il est déconnecté du plan de contrôle Kubernetes. La raison du redémarrage est due à l'échec du bilan de santé de Cilium, car son état de santé est associé à l'accès au plan de contrôle Kubernetes (voir CFP : #31702 avec une amélioration

opt-in dans Cilium v1.17). De même, si vous utilisez des équilibreurs de charge d'application (ALB) ou des équilibreurs de charge réseau (NLB) pour le trafic d'applications provenant de la région AWS, ce trafic peut être temporairement interrompu si votre environnement sur site perd la connectivité avec la région AWS. Il est recommandé de vérifier que les technologies que vous utilisez pour l'équilibrage de charge et l'entrée restent stables pendant les déconnexions du réseau avant de les déployer en production. L'exemple du eks-hybrid-examples GitHub référentiel aws-samples/ utilise MetalLB pour l'équilibrage de charge en mode L2, qui reste stable lors des déconnexions réseau entre les nœuds hybrides et le plan de contrôle EKS.

Passez en revue les dépendances sur les services AWS distants

Lorsque vous utilisez des nœuds hybrides, soyez conscient des dépendances que vous assumez vis-à-vis des services AWS régionaux externes à votre environnement sur site ou périphérique. Les exemples incluent l'accès à Amazon S3 ou Amazon RDS pour les données d'application, l'utilisation d'Amazon Managed Service pour Prometheus CloudWatch ou pour les métriques et les journaux, l'utilisation d'équilibreurs de charge d'applications et de réseaux pour le trafic provenant de régions et le retrait de conteneurs depuis Amazon Elastic Container Registry. Ces services ne seront pas accessibles lors des déconnexions réseau entre votre environnement sur site et AWS. Si votre environnement sur site est sujet à des déconnexions réseau avec AWS, passez en revue votre utilisation des services AWS et assurez-vous que la perte de connexion à ces services ne compromet pas la stabilité statique de vos applications.

Régler le comportement de basculement du pod Kubernetes

Il existe des options permettant d'ajuster le comportement de basculement des pods lors des déconnexions réseau pour les applications qui ne sont pas portables entre les hôtes ou pour les environnements aux ressources limitées qui ne disposent pas de capacité de réserve pour le basculement des pods. En général, il est important de prendre en compte les besoins en ressources de vos applications et de disposer d'une capacité suffisante pour qu'une ou plusieurs instances de l'application puissent basculer vers un autre hôte en cas de défaillance d'un nœud.

- Option 1 - Utilisation DaemonSets : Cette option s'applique aux applications qui peuvent et doivent s'exécuter sur tous les nœuds du cluster. DaemonSets sont automatiquement configurés pour tolérer les souillures inaccessibles, qui maintiennent les DaemonSet pods liés à leurs nœuds en cas de déconnexion du réseau.
- Option 2 - Régler `tolerationSeconds` pour éviter les altérations inaccessibles : vous pouvez régler la durée pendant laquelle vos pods restent liés aux nœuds lors des déconnexions du réseau. Pour ce faire, configurez les modules d'application de manière à ce qu'ils tolèrent

cette odeur inaccessible avec l'`NoExecute`effet pendant une durée que vous spécifiez (`tolerationSecondsdans` les spécifications de l'application). Avec cette option, en cas de déconnexion du réseau, les pods de votre application restent liés aux nœuds jusqu'à leur `tolerationSeconds` expiration. Réfléchissez bien à cette question, car si vous `tolerationSeconds` augmentez le nombre d'hôtes inaccessibles, `NoExecute` cela signifie que les pods fonctionnant sur des hôtes inaccessibles peuvent mettre plus de temps à être transférés vers d'autres hôtes sains et accessibles.

- Option 3 : contrôleur personnalisé : vous pouvez créer et exécuter un contrôleur personnalisé (ou un autre logiciel) qui surveille Kubernetes pour détecter toute trace d'effet indésirable. `NoExecute` Lorsque cette altération est détectée, le contrôleur personnalisé peut vérifier les métriques spécifiques à l'application pour évaluer l'état de santé de l'application. Si l'application est saine, le contrôleur personnalisé peut supprimer la souillure inaccessible, empêchant ainsi l'expulsion des pods des nœuds lors des déconnexions du réseau.

Vous trouverez ci-dessous un exemple de configuration d'un déploiement en cas d'inaccessibilité. `tolerationSeconds` Dans l'exemple, `tolerationSeconds` est défini sur `1800` (30 minutes), ce qui signifie que les pods exécutés sur des nœuds inaccessibles ne seront expulsés que si la déconnexion du réseau dure plus de 30 minutes.

```
apiVersion: apps/v1
kind: Deployment
metadata:
...
spec:
...
      tolerations:
      - key: "node.kubernetes.io/unreachable"
        operator: "Exists"
        effect: "NoExecute"
        tolerationSeconds: 1800
```

Basculement du pod Kubernetes via des déconnexions réseau

Nous commençons par un examen des concepts, composants et paramètres clés qui influencent le comportement de Kubernetes lors des déconnexions réseau entre les nœuds et le plan de contrôle Kubernetes. EKS est conforme à Kubernetes en amont, de sorte que tous les concepts, composants et paramètres Kubernetes décrits ici s'appliquent aux déploiements d'EKS et de nœuds hybrides EKS.

Des améliorations ont été apportées à EKS spécifiquement pour améliorer le comportement de basculement des pods lors de déconnexions réseau. Pour plus d'informations, consultez les GitHub problèmes #131294 et #131481 dans le référentiel Kubernetes en amont.

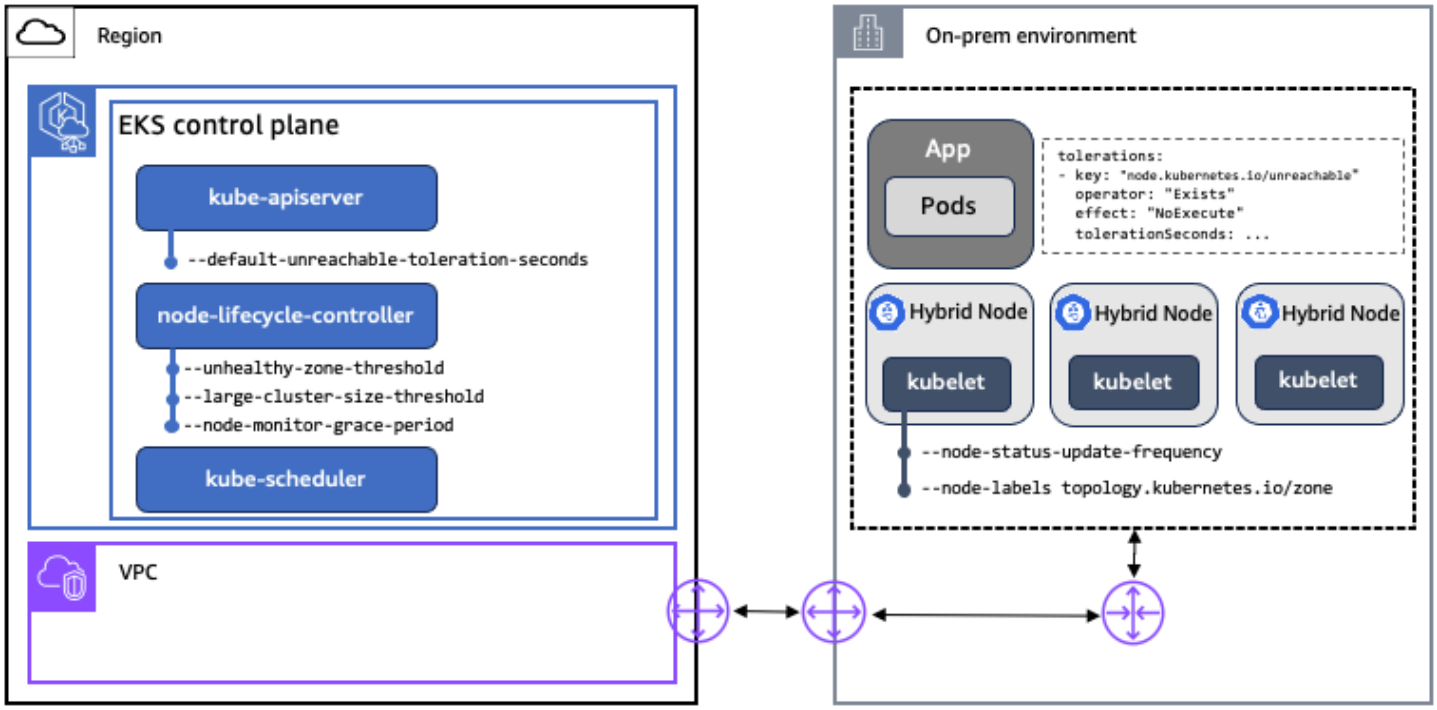
Concepts

Contraintes et tolérances : les altérations et les tolérances sont utilisées dans Kubernetes pour contrôler la planification des pods sur les nœuds. Les taches sont définies par le node-lifecycle-controller pour indiquer que les nœuds ne sont pas éligibles à la planification ou que les pods de ces nœuds doivent être expulsés. Lorsque les nœuds sont inaccessibles en raison d'une déconnexion du réseau, le node.kubernetes node-lifecycle-controller s'applique. io/unreachable taint with a NoSchedule effect, and with a NoExecute effect if certain conditions are met. The node.kubernetes.io/unreachabletaint correspond au NodeCondition Ready being Unknown. Les utilisateurs peuvent spécifier des tolérances pour les taches au niveau de l'application dans le. PodSpec

- NoSchedule: Aucun nouveau pod n'est prévu sur le nœud contaminé à moins qu'il n'ait une tolérance correspondante. Les pods déjà actifs sur le nœud ne sont pas expulsés.
- NoExecute: Les capsules qui ne tolèrent pas la souillure sont immédiatement expulsées. Les pods qui tolèrent cette odeur (sans spécifier TolerationSeconds) restent liés pour toujours. Les capsules qui tolèrent l'odeur avec une valeur de TolerationSeconds spécifiée restent limitées pendant la durée spécifiée. Une fois ce délai écoulé, le contrôleur du cycle de vie du nœud évacue les pods du nœud.

Locations de nœuds : Kubernetes utilise l'API Lease pour communiquer les pulsations des nœuds Kubelet au serveur d'API Kubernetes. Pour chaque nœud, il existe un objet Lease dont le nom correspond. En interne, chaque battement de cœur du kubelet met à jour le champ spec.RenewTime de l'objet Lease. Le plan de contrôle Kubernetes utilise l'horodatage de ce champ pour déterminer la disponibilité des nœuds. Si les nœuds sont déconnectés du plan de contrôle Kubernetes, ils ne peuvent pas mettre à jour Spec.RenewTime pour leur bail, et le plan de contrôle interprète cela comme le Ready étant Unknown. NodeCondition

Éléments



Composant	Sous-composant	Description
Plan de contrôle Kubernetes	kube-api-server	Le serveur d'API est un composant essentiel du plan de contrôle Kubernetes qui expose l'API Kubernetes.
Plan de contrôle Kubernetes	node-lifecycle-controller	L'un des contrôleurs qu'il fait kube-controller-manager fonctionner. Il est chargé de détecter les problèmes liés aux nœuds et d'y répondre.
Plan de contrôle Kubernetes	planificateur Kube	Un composant du plan de contrôle qui surveille les nouveaux pods sans nœud assigné et sélectionne un nœud sur lequel ils pourront s'exécuter.
Nœuds Kubernetes	kubelet	Un agent qui s'exécute sur chaque nœud du cluster. Le kubelet surveille PodSpecs et s'assure que les récipients décrits dans ces documents fonctionnent et PodSpecs sont sains.

Paramètres de configuration

Compos	Paramètre	Description	K8s par défaut	EKS par défaut	Configurable dans EKS
kube-api-server	default-unreachable-toleration-seconds	Indique `tolerationSeconds` la tolérance `unreachable:NoExecute` qui est ajoutée par défaut à chaque pod qui ne dispose pas déjà d'une telle tolérance.	300	300	Non
node-lifecycle-controller	node-monitor-grace-period	Durée pendant laquelle un nœud peut ne pas répondre avant d'être marqué comme étant défectueux. Doit être N fois supérieur à celui de kubelet`nodeStatusUpdateFrequency` , où N est le nombre de tentatives autorisées pour que le kubelet publie le statut du nœud.	40	40	Non
node-lifecycle-controller	large-cluster-size-threshold	Le nombre de nœuds auxquels le cluster est node-lifecycle-controller considéré comme étant important pour la logique d'éviction. `--secondary-node-eviction-rate` est remplacé par 0 pour les clusters de cette taille ou moins.	50	100 000	Non
node-lifecycle-controller	unhealthy-zone-threshold	Pourcentage de nœuds d'une zone qui doivent être « Non prêts » pour que cette zone soit considérée comme non saine.	55 %	55 %	Non
kubelet	node-status-update-frequency	Fréquence à laquelle le kubelet publie l'état du nœud sur le plan de contrôle. Doit être compatible avec `nodeMonit`	10	10	Oui

Compos	Paramètre	Description	K8s par défaut	EKS par défaut	Configurable dans EKS
		`orGracePeriod` in node-lifecycle-controller.			
kubelet	étiquettes de nœuds	Étiquettes à ajouter lors de l'enregistrement du nœud dans le cluster. L'étiquette `topology.kubernetes.io/zone` peut être spécifiée avec des nœuds hybrides pour regrouper les nœuds en zones.	Aucune	Aucune	Oui

Basculement du pod Kubernetes via des déconnexions réseau

Le comportement décrit ici suppose que les pods s'exécutent en tant que déploiements Kubernetes avec des paramètres par défaut, et qu'EKS est utilisé comme fournisseur Kubernetes. Le comportement réel peut varier en fonction de votre environnement, du type de déconnexion réseau, des applications, des dépendances et de la configuration du cluster. Le contenu de ce guide a été validé à l'aide d'une application spécifique, d'une configuration de cluster et d'un sous-ensemble de plug-ins. Il est vivement recommandé de tester le comportement dans votre propre environnement et avec vos propres applications avant de passer à la production.

En cas de déconnexion réseau entre les nœuds et le plan de contrôle Kubernetes, le kubelet de chaque nœud déconnecté ne peut pas communiquer avec le plan de contrôle Kubernetes. Par conséquent, le kubelet ne peut pas expulser les pods sur ces nœuds tant que la connexion n'est pas rétablie. Cela signifie que les pods exécutés sur ces nœuds avant la déconnexion du réseau continuent de fonctionner pendant la déconnexion, en supposant qu'aucune autre défaillance ne provoque leur arrêt. En résumé, vous pouvez obtenir une stabilité statique lors des déconnexions réseau entre les nœuds et le plan de contrôle Kubernetes, mais vous ne pouvez pas effectuer d'opérations de mutation sur vos nœuds ou vos charges de travail tant que la connexion n'est pas rétablie.

Cinq scénarios principaux produisent différents comportements de basculement des pods en fonction de la nature de la déconnexion du réseau. Dans tous les scénarios, le cluster redevient sain sans intervention de l'opérateur une fois que les nœuds sont reconnectés au plan de contrôle Kubernetes.

Les scénarios ci-dessous décrivent les résultats attendus sur la base de nos observations, mais ces résultats peuvent ne pas s'appliquer à toutes les configurations d'applications et de clusters possibles.

Scénario 1 : interruption complète du cluster

Résultat attendu : les pods situés sur des nœuds inaccessibles ne sont pas expulsés et continuent de fonctionner sur ces nœuds.

Une interruption complète du cluster signifie que tous les nœuds du cluster sont déconnectés du plan de contrôle Kubernetes. Dans ce scénario, node-lifecycle-controller le plan de contrôle détecte que tous les nœuds du cluster sont inaccessibles et annule toute expulsion de pods.

Les administrateurs du cluster verront l'état de tous les nœuds `Not Ready` lors de la déconnexion. L'état du pod ne change pas et aucun nouveau pod n'est programmé sur aucun nœud pendant la déconnexion et la reconnexion ultérieure.

Scénario 2 : perturbation complète de la zone

Résultat attendu : les pods situés sur des nœuds inaccessibles ne sont pas expulsés et continuent de fonctionner sur ces nœuds.

Une interruption de zone complète signifie que tous les nœuds de la zone sont déconnectés du plan de contrôle Kubernetes. Dans ce scénario, node-lifecycle-controller le plan de contrôle détecte que tous les nœuds de la zone sont inaccessibles et annule toute expulsion de pods.

Les administrateurs du cluster verront l'état de tous les nœuds `Not Ready` lors de la déconnexion. L'état du pod ne change pas et aucun nouveau pod n'est programmé sur aucun nœud pendant la déconnexion et la reconnexion ultérieure.

Scénario 3 : perturbation de la zone majoritaire

Résultat attendu : les pods situés sur des nœuds inaccessibles ne sont pas expulsés et continuent de fonctionner sur ces nœuds.

Une interruption de zone majoritaire signifie que la plupart des nœuds d'une zone donnée sont déconnectés du plan de contrôle Kubernetes. Les zones de Kubernetes sont définies par des nœuds portant le même label. `topology.kubernetes.io/zone` Si aucune zone n'est définie dans le cluster, une perturbation majeure signifie que la majorité des nœuds de l'ensemble du cluster sont déconnectés. Par défaut, une majorité est définie par le node-lifecycle-controller « s »`unhealthy-`

`zone-threshold`, qui est défini à 55 % dans Kubernetes et EKS. Étant donné que ce paramètre `large-cluster-size-threshold` est défini sur 100 000 dans EKS, si 55 % ou plus des nœuds d'une zone sont inaccessibles, les expulsions de pods sont annulées (étant donné que la plupart des clusters sont bien inférieurs à 100 000 nœuds).

Les administrateurs du cluster verront la majorité des nœuds de la zone avoir un statut `Not Ready` lors de la déconnexion, mais le statut des pods ne changera pas et ils ne seront pas reprogrammés sur les autres nœuds.

Notez que le comportement ci-dessus s'applique uniquement aux clusters de plus de trois nœuds. Dans les clusters de trois nœuds ou moins, les pods situés sur des nœuds inaccessibles sont planifiés pour être expulsés, et les nouveaux pods sont planifiés sur des nœuds sains.

Au cours des tests, nous avons parfois observé que des pods étaient expulsés d'un seul nœud inaccessible lors de déconnexions du réseau, même lorsque la majorité des nœuds de la zone étaient inaccessibles. Nous étudions toujours une éventuelle condition de race dans Kubernetes node-lifecycle-controller à l'origine de ce comportement.

Scénario 4 : Perturbation de la zone minoritaire

Résultat attendu : les pods sont expulsés des nœuds inaccessibles, et de nouveaux pods sont programmés sur les nœuds éligibles disponibles.

Une interruption minoritaire signifie qu'un faible pourcentage de nœuds d'une zone sont déconnectés du plan de contrôle Kubernetes. Si aucune zone n'est définie dans le cluster, une interruption minoritaire signifie que la minorité de nœuds de l'ensemble du cluster est déconnectée. Comme indiqué, la minorité est définie par le `unhealthy-zone-threshold` paramètre de node-lifecycle-controller, qui est de 55 % par défaut. Dans ce scénario, si la déconnexion du réseau dure plus de 5 minutes et 40 secondes et `node-monitor-grace-period` que moins de 55 % des nœuds d'une zone sont inaccessibles, de nouveaux pods sont programmés sur des nœuds sains tandis que les pods situés sur des nœuds inaccessibles sont marqués pour expulsion. `default-unreachable-toleration-seconds`

Les administrateurs de clusters verront de nouveaux pods créés sur des nœuds sains, et les pods sur des nœuds déconnectés seront affichés sous la forme`Terminating`. N'oubliez pas que, même si les pods des nœuds déconnectés ont un `Terminating` statut, ils ne sont pas complètement expulsés tant que le nœud ne se reconnecte pas au plan de contrôle Kubernetes.

Scénario 5 : redémarrage du nœud lors d'une interruption du réseau

Résultat attendu : les pods situés sur des nœuds inaccessibles ne sont pas démarrés tant que les nœuds ne sont pas reconnectés au plan de contrôle Kubernetes. Le basculement du pod suit la logique décrite dans les scénarios 1 à 3, en fonction du nombre de nœuds inaccessibles.

Un redémarrage d'un nœud lors d'une interruption du réseau signifie qu'une autre panne (telle qu'un cycle d'alimentation, un out-of-memory événement ou un autre problème) s'est produite sur un nœud en même temps qu'une déconnexion du réseau. Les pods qui s'exécutaient sur ce nœud lorsque la déconnexion du réseau a commencé ne sont pas automatiquement redémarrés lors de la déconnexion si le kubelet a également redémarré. Le kubelet interroge le serveur d'API Kubernetes au démarrage pour savoir quels pods il doit exécuter. Si le kubelet ne peut pas atteindre le serveur API en raison d'une déconnexion du réseau, il ne peut pas récupérer les informations nécessaires pour démarrer les pods.

Dans ce scénario, les outils de dépannage locaux tels que la `crictl` CLI ne peuvent pas être utilisés pour démarrer les pods manuellement afin de « briser la vitre ». Kubernetes supprime généralement les pods défaillants et en crée de nouveaux plutôt que de redémarrer les pods existants (voir #10213 dans le dépôt GitHub containerd pour plus de détails). Les pods statiques sont le seul objet de charge de travail Kubernetes contrôlé par le kubelet et peuvent être redémarrés dans ces scénarios. Cependant, il n'est généralement pas recommandé d'utiliser des pods statiques pour les déploiements d'applications. Déployez plutôt plusieurs répliques sur différents hôtes pour garantir la disponibilité des applications en cas de défaillances simultanées multiples, telles qu'une panne de nœud ou une déconnexion du réseau entre vos nœuds et le plan de contrôle Kubernetes.

Trafic réseau des applications via des déconnexions réseau

Les rubriques de cette page sont liées à la mise en réseau des clusters Kubernetes et au trafic des applications lors des déconnexions réseau entre les nœuds et le plan de contrôle Kubernetes.

Cilium

Cilium dispose de plusieurs modes de gestion des adresses IP (IPAM), d'encapsulation, d'équilibrage de charge et de routage de clusters. Les modes validés dans ce guide utilisaient Cluster Scope IPAM, la superposition VXLAN, l'équilibrage de charge BGP et le kube-proxy. Le cilium a également été utilisé sans équilibrage de charge BGP, en le remplaçant par un équilibrage de charge MetalLB L2.

La base de l'installation Cilium se compose de l'opérateur Cilium et des agents Cilium. L'opérateur Cilium s'exécute en tant que déploiement et enregistre les définitions de ressources personnalisées

Cilium (CRDs), gère l'IPAM et synchronise les objets du cluster avec le serveur API Kubernetes, entre autres fonctionnalités. Les agents Cilium s'exécutent sur chaque nœud en tant que DaemonSet et gèrent les programmes eBPF afin de contrôler les règles réseau applicables aux charges de travail exécutées sur le cluster.

En général, le routage intégré au cluster configuré par Cilium reste disponible et en place pendant les déconnexions du réseau, ce qui peut être confirmé en observant les flux de trafic internes au cluster et les règles de table IP (iptables) pour le réseau du pod.

```
ip route show table all | grep cilium
```

```
10.86.2.0/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16 mtu 1450
10.86.2.64/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16 mtu 1450
10.86.2.128/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16 mtu 1450
10.86.2.192/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16 mtu 1450
10.86.3.0/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16
10.86.3.16 dev cilium_host proto kernel scope link
...
```

Cependant, lors des déconnexions du réseau, l'opérateur Cilium et les agents Cilium redémarrent en raison du couplage de leurs bilans de santé avec l'état de la connexion avec le serveur API Kubernetes. On s'attend à voir ce qui suit dans les journaux de l'opérateur Cilium et des agents Cilium lors des déconnexions du réseau. Pendant les déconnexions du réseau, vous pouvez utiliser des outils tels que la `crictl` CLI pour observer les redémarrages de ces composants, y compris leurs journaux.

```
msg="Started gops server" address="127.0.0.1:9890" subsys=gops
msg="Establishing connection to apiserver" host="https://<k8s-cluster-ip>:443"
 subsys=k8s-client
msg="Establishing connection to apiserver" host="https://<k8s-cluster-ip>:443"
 subsys=k8s-client
msg="Unable to contact k8s api-server" error="Get \"https://<k8s-cluster-ip>:443/
api/v1/namespaces/kube-system\": dial tcp <k8s-cluster-ip>:443: i/o timeout"
 ipAddr="https://<k8s-cluster-ip>:443" subsys=k8s-client
msg="Start hook failed" function="client.(*compositeClientset).onStart
 (agent.infra.k8s-client)" error="Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/
kube-system\": dial tcp <k8s-cluster-ip>:443: i/o timeout"
msg="Start failed" error="Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/kube-
system\": dial tcp <k8s-cluster-ip>:443: i/o timeout" duration=1m5.003834026s
msg=Stopping
```

```
msg="Stopped gops server" address="127.0.0.1:9890" subsys=gops
msg="failed to start: Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/kube-system
\": dial tcp <k8s-cluster-ip>:443: i/o timeout" subsys=daemon
```

Si vous utilisez la fonctionnalité du plan de contrôle BGP de Cilium pour l'équilibrage de charge des applications, la session BGP pour vos pods et services peut être interrompue lors des déconnexions réseau car la fonctionnalité des haut-parleurs BGP est intégrée à l'agent Cilium, et l'agent Cilium redémarre en permanence lorsqu'il est déconnecté du plan de contrôle Kubernetes. Pour plus d'informations, consultez le guide d'utilisation du plan de contrôle Cilium BGP dans la documentation de Cilium. En outre, si vous rencontrez une panne simultanée lors d'une déconnexion du réseau, telle qu'un cycle d'alimentation ou un redémarrage de machine, les routes Cilium ne seront pas préservées grâce à ces actions, bien qu'elles soient recréées lorsque le nœud se reconnecte au plan de contrôle Kubernetes et que Cilium redémarre.

Calico

Prochainement

Métal B

MetalLB dispose de deux modes d'équilibrage de charge : le mode L2 et le mode BGP. Consultez la documentation de MetalLB pour plus de détails sur le fonctionnement de ces modes d'équilibrage de charge et leurs limites. La validation de ce guide a utilisé MetalLB en mode L2, où une machine du cluster prend possession du service Kubernetes et utilise ARP pour rendre les adresses IP de l'équilibreur IPv4 de charge accessibles sur le réseau local. Lors de l'exécution de MetalLB, un contrôleur est responsable de l'attribution des adresses IP et des haut-parleurs exécutés sur chaque nœud sont responsables des services publicitaires auxquels les adresses IP sont attribuées. Le contrôleur MetalLB fonctionne en tant que déploiement et les haut-parleurs MetalLB fonctionnent en tant que. DaemonSet Lors des déconnexions réseau, le contrôleur MetalLB et les haut-parleurs ne surveillent pas les ressources du cluster sur le serveur d'API Kubernetes, mais continuent de fonctionner. Plus important encore, les Services qui utilisent MetalLB pour la connectivité externe restent disponibles et accessibles pendant les déconnexions du réseau.

kube-proxy

Dans les clusters EKS, kube-proxy s'exécute en tant que serveur DaemonSet sur chaque nœud et est chargé de gérer les règles du réseau afin de permettre la communication entre les services et les pods en traduisant les adresses IP des services en adresses IP des pods sous-jacents. Les règles

des tables IP (iptables) configurées par kube-proxy sont maintenues pendant les déconnexions du réseau, le routage au sein du cluster continue de fonctionner et les pods kube-proxy continuent de fonctionner.

Vous pouvez observer les règles de kube-proxy avec les commandes iptables suivantes. La première commande montre que les paquets passant par la `PREROUTING` chaîne sont dirigés vers la `KUBE-SERVICES` chaîne.

```
iptables -t nat -L PREROUTING
```

```
Chain PREROUTING (policy ACCEPT)
target         prot opt source      destination
KUBE-SERVICES  all  --  anywhere    anywhere      /* kubernetes service portals */
```

En inspectant la `KUBE-SERVICES` chaîne, nous pouvons voir les règles des différents services du cluster.

```
Chain KUBE-SERVICES (2 references)
target                     prot opt source      destination
KUBE-SVL-NZTS37XDTDNXGCKJ  tcp  --  anywhere    172.16.189.136  /* kube-system/hubble-
peer:peer-service cluster IP /
KUBE-SVC-2BINP2AXJOTI3HJ5  tcp  --  anywhere    172.16.62.72    / default/metallb-
webhook-service cluster IP /
KUBE-SVC-LRNEBRA3Z5YGJ4QC  tcp  --  anywhere    172.16.145.111  / default/redis-leader
 cluster IP /
KUBE-SVC-I7SKRZYQ7PWYV5X7  tcp  --  anywhere    172.16.142.147  / kube-system/eks-
extension-metrics-api:metrics-api cluster IP /
KUBE-SVC-JD5MR3NA4I4DYORP  tcp  --  anywhere    172.16.0.10     / kube-system/kube-
dns:metrics cluster IP /
KUBE-SVC-TCOU7JCQXEZGVUNU  udp  --  anywhere    172.16.0.10     / kube-system/kube-
dns:dns cluster IP /
KUBE-SVC-ERIFXISQEP7F7OF4  tcp  --  anywhere    172.16.0.10     / kube-system/kube-
dns:dns-tcp cluster IP /
KUBE-SVC-ENODL3HWJ5BZY56Q  tcp  --  anywhere    172.16.7.26     / default/frontend
 cluster IP /
KUBE-EXT-ENODL3HWJ5BZY56Q  tcp  --  anywhere    <LB-IP>    / default/frontend
 loadbalancer IP /
KUBE-SVC-NPX46M4PTMTKRN6Y  tcp  --  anywhere    172.16.0.1      / default/
kubernetes:https cluster IP /
KUBE-SVC-YU5RV2YQWHLZ5XPR  tcp  --  anywhere    172.16.228.76   / default/redis-
follower cluster IP /
```

```
KUBE-NODEPORTS             all  --  anywhere    anywhere        / kubernetes service
 nodeports; NOTE: this must be the last rule in this chain */
```

En inspectant la chaîne du service frontal de l'application, nous pouvons voir les adresses IP des pods qui soutiennent le service.

```
iptables -t nat -L KUBE-SVC-ENODL3HWJ5BZY56Q
```

```
Chain KUBE-SVC-ENODL3HWJ5BZY56Q (2 references)
target                     prot opt source    destination
KUBE-SEP-EKXE7ASH7Y74BGBO  all  --  anywhere  anywhere    /* default/frontend ->
 10.86.2.103:80 / statistic mode random probability 0.33333333349
KUBE-SEP-GCY3OUXWSVMSEAR6  all  --  anywhere  anywhere    / default/frontend ->
 10.86.2.179:80 / statistic mode random probability 0.50000000000
KUBE-SEP-6GJJR3EF5AUP2WBU  all  --  anywhere  anywhere    / default/frontend ->
 10.86.3.47:80 */
```

Les messages du journal kube-proxy suivants sont attendus lors des déconnexions du réseau lorsqu'il tente de surveiller le serveur d'API Kubernetes pour détecter les mises à jour des ressources des nœuds et des points de terminaison.

```
"Unhandled Error" err="k8s.io/client-go/informers/factory.go:160: Failed to watch
 *v1.Node: failed to list *v1.Node: Get \"https://<k8s-endpoint>/api/v1/nodes?
fieldSelector=metadata.name%3D<node-name>&resourceVersion=2241908\": dial tcp <k8s-
ip>:443: i/o timeout" logger="UnhandledError"
"Unhandled Error" err="k8s.io/client-go/informers/factory.go:160: Failed to watch
 *v1.EndpointSlice: failed to list *v1.EndpointSlice: Get \"https://<k8s-endpoint>/
apis/discovery.k8s.io/v1/endpointslices?labelSelector=%21service.kubernetes.io
%2Fheadless%2C%21service.kubernetes.io%2Fservice-proxy-name&resourceVersion=2242090\":
 dial tcp <k8s-ip>:443: i/o timeout" logger="UnhandledError"
```

CoreDNS

Par défaut, les pods des clusters EKS utilisent l'adresse IP du cluster CoreDNS comme serveur de noms pour les requêtes DNS internes au cluster. Dans les clusters EKS, CoreDNS s'exécute en tant que déploiement sur des nœuds. Avec les nœuds hybrides, les pods peuvent continuer à communiquer avec le CoreDNS pendant les déconnexions du réseau lorsque des répliques de CoreDNS s'exécutent localement sur des nœuds hybrides. Si vous avez un cluster EKS avec des nœuds dans le cloud et des nœuds hybrides dans votre environnement sur site, il est recommandé de disposer d'au moins une réplique CoreDNS dans chaque environnement. CoreDNS continue

de répondre aux requêtes DNS pour les enregistrements créés avant la déconnexion du réseau et continue à exécuter la reconnexion réseau pour garantir une stabilité statique.

Les messages de journal CoreDNS suivants sont attendus lors des déconnexions du réseau lorsqu'il tente de répertorier des objets depuis le serveur d'API Kubernetes.

```
Failed to watch *v1.Namespace: failed to list *v1.Namespace: Get "https://<k8s-cluster-
ip>:443/api/v1/namespaces?resourceVersion=2263964": dial tcp <k8s-cluster-ip>:443: i/o
 timeout
Failed to watch *v1.Service: failed to list *v1.Service: Get "https://<k8s-cluster-
ip>:443/api/v1/services?resourceVersion=2263966": dial tcp <k8s-cluster-ip>:443: i/o
 timeout
Failed to watch *v1.EndpointSlice: failed to list *v1.EndpointSlice: Get "https://<k8s-
cluster-ip>:443/apis/discovery.k8s.io/v1/endpointslices?resourceVersion=2263896": dial
 tcp <k8s-cluster-ip>: i/o timeout
```

Informations d'identification de l'hôte via des déconnexions réseau

EKS Hybrid Nodes est intégré aux activations hybrides d'AWS Systems Manager (SSM) et à AWS IAM Roles Anywhere pour les informations d'identification IAM temporaires utilisées pour authentifier le nœud auprès du plan de contrôle EKS. SSM et IAM Roles Anywhere actualisent automatiquement les informations d'identification temporaires qu'ils gèrent sur les hôtes locaux. Il est recommandé d'utiliser un seul fournisseur d'informations d'identification pour tous les nœuds hybrides de votre cluster, qu'il s'agisse d'activations hybrides SSM ou d'IAM Roles Anywhere, mais pas les deux.

Activations hybrides SSM

Les informations d'identification temporaires fournies par SSM sont valides pendant une heure. Vous ne pouvez pas modifier la durée de validité des informations d'identification lorsque vous utilisez SSM comme fournisseur d'informations d'identification. Les informations d'identification temporaires sont automatiquement modifiées par SSM avant leur expiration, et cette rotation n'affecte pas le statut de vos nœuds ou applications. Toutefois, en cas de déconnexion réseau entre l'agent SSM et le point de terminaison régional SSM, SSM n'est pas en mesure d'actualiser les informations d'identification et celles-ci peuvent expirer.

SSM utilise un retard exponentiel pour les nouvelles tentatives d'actualisation des informations d'identification s'il n'est pas en mesure de se connecter aux points de terminaison régionaux SSM. Dans les versions de l'agent SSM `3.3.808.0` et ultérieures (publiées en août 2024), le retard exponentiel est plafonné à 30 minutes. Selon la durée de la déconnexion de votre réseau, l'actualisation des informations d'identification par SSM peut prendre jusqu'à 30 minutes, et les

nœuds hybrides ne se reconnecteront pas au plan de contrôle EKS tant que les informations d'identification ne seront pas actualisées. Dans ce scénario, vous pouvez redémarrer l'agent SSM pour forcer l'actualisation des informations d'identification. Comme effet secondaire du comportement actuel d'actualisation des informations d'identification SSM, les nœuds peuvent se reconnecter à des moments différents selon le moment où l'agent SSM de chaque nœud parvient à actualiser ses informations d'identification. De ce fait, vous pouvez assister à un basculement du pod depuis des nœuds qui ne sont pas encore reconnectés vers des nœuds déjà reconnectés.

Obtenez la version de l'agent SSM. Vous pouvez également consulter la section Fleet Manager de la console SSM :

```
# AL2023, RHEL
yum info amazon-ssm-agent
# Ubuntu
snap list amazon-ssm-agent
```

Redémarrez l'agent SSM :

```
# AL2023, RHEL
systemctl restart amazon-ssm-agent
# Ubuntu
systemctl restart snap.amazon-ssm-agent.amazon-ssm-agent
```

Afficher les journaux des agents SSM :

```
tail -f /var/log/amazon/ssm/amazon-ssm-agent.log
```

Messages de journal attendus lors des déconnexions du réseau :

```
INFO [CredentialRefresher] Credentials ready
INFO [CredentialRefresher] Next credential rotation will be in 29.995040663666668
 minutes
ERROR [CredentialRefresher] Retrieve credentials produced error: RequestError: send
 request failed
INFO [CredentialRefresher] Sleeping for 35s before retrying retrieve credentials
ERROR [CredentialRefresher] Retrieve credentials produced error: RequestError: send
 request failed
INFO [CredentialRefresher] Sleeping for 56s before retrying retrieve credentials
ERROR [CredentialRefresher] Retrieve credentials produced error: RequestError: send
 request failed
```

```
INFO [CredentialRefresher] Sleeping for 1m24s before retrying retrieve credentials
```

Rôles Anywhere IAM

Les informations d'identification temporaires fournies par IAM Roles Anywhere sont valides pendant une heure par défaut. Vous pouvez configurer la durée de validité des informations d'identification avec IAM Roles Anywhere via le `durationSeconds`champ de votre profil IAM Roles Anywhere. La durée maximale de validité des informations d'identification est de 12 heures. Le `MaxSessionDuration`paramètre de votre rôle IAM Hybrid Nodes doit être supérieur à celui `durationSeconds` de votre profil IAM Roles Anywhere.

Lorsque vous utilisez IAM Roles Anywhere comme fournisseur d'informations d'identification pour vos nœuds hybrides, la reconnexion au plan de contrôle EKS après une déconnexion du réseau a généralement lieu quelques secondes après la restauration du réseau, car le kubelet appelle `aws_signing_helper credential-process` pour obtenir des informations d'identification à la demande. Bien que cela ne soit pas directement lié aux nœuds hybrides ou aux déconnexions du réseau, vous pouvez configurer des notifications et des alertes relatives à l'expiration des certificats lorsque vous utilisez IAM Roles Anywhere. Pour plus d'informations, voir Personnaliser les paramètres de notification dans IAM Roles Anywhere.

Bonnes pratiques en matière d'exécution des charges AI/ML de travail

> **Tip**
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

La mise en œuvre des meilleures pratiques lors de l'exécution de AI/ML charges de travail sur EKS peut garantir que ces charges de travail sont performantes, rentables, résilientes et dotées de ressources appropriées. Les meilleures pratiques sont réparties dans les sections générales suivantes : calcul, mise en réseau, stockage, observabilité et performances.

Commentaires

Ce guide est publié GitHub afin de recueillir les commentaires et suggestions directs de l'ensemble de la EKS/Kubernetes communauté. Si vous avez une bonne pratique que vous pensez que nous devrions inclure dans le guide, veuillez signaler un problème ou soumettre un PR dans le GitHub référentiel. Notre intention est de mettre à jour le guide périodiquement à mesure que de nouvelles fonctionnalités sont ajoutées au service ou lorsqu'une nouvelle bonne pratique évolue.

Calcul et mise à l'échelle automatique

> **Tip**
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Optimisation des ressources GPU et gestion des coûts

Planifiez les charges de travail en fonction des exigences en matière de GPU à l'aide d'étiquettes connues

Pour les AI/ML charges de travail sensibles aux différentes caractéristiques du GPU (par exemple, GPU, mémoire GPU), nous recommandons de spécifier les exigences du GPU à l'aide d'étiquettes de

planification connues prises en charge par les types de nœuds utilisés avec Karpenter et les groupes de nœuds gérés. Si vous ne les définissez pas, les pods peuvent être planifiés sur des instances dont les ressources GPU sont insuffisantes, ce qui peut entraîner des défaillances ou une dégradation des performances. Nous vous recommandons d'utiliser NodeSelector ou Node Affinity pour spécifier sur quel nœud un pod doit s'exécuter et de définir les ressources de calcul (processeur, mémoire, GPUs etc.) dans la section des ressources du pod.

Exemple

Par exemple, en utilisant le sélecteur de nœud de nom du GPU lors de l'utilisation de Karpenter :

```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod-example
spec:
  containers:
  - name: ml-workload
    image: <image>
    resources:
      limits:
        nvidia.com/gpu: 1  # Request one NVIDIA GPU
  nodeSelector:
    karpenter.k8s.aws/instance-gpu-name: "l40s"  # Run on nodes with NVIDIA L40S GPUs
```

Utilisez le plugin Kubernetes Device pour exposer GPUs

Pour effectuer une exposition GPUs sur des nœuds, le pilote GPU NVIDIA doit être installé sur le système d'exploitation du nœud et le runtime du conteneur doit être configuré pour permettre au planificateur Kubernetes d'attribuer des pods aux nœuds disponibles. GPUs Le processus de configuration du plug-in d'appareil NVIDIA Kubernetes dépend de l'AMI accélérée EKS que vous utilisez :

- AMI accélérée Bottlerocket : cette AMI inclut le pilote GPU NVIDIA et le plug-in pour appareil NVIDIA Kubernetes est préinstallé et prêt à l'emploi, permettant ainsi un support GPU prêt à l'emploi. Aucune configuration supplémentaire n'est requise pour accéder GPUs au planificateur Kubernetes.
- AL2023 AMI accélérée : cette AMI inclut le pilote GPU NVIDIA, mais le plug-in de périphérique NVIDIA Kubernetes n'est pas préinstallé. Vous devez installer et configurer le plug-in de l'appareil séparément, généralement via un DaemonSet. Notez que si vous utilisez eksctl pour créer votre

cluster et que vous spécifiez un type d'instance GPU (par exemple,`g5.xlarge`) dans votre ClusterConfig cluster, il `eksctl` sélectionnera automatiquement l'AMI appropriée et installera le plug-in de périphérique NVIDIA Kubernetes. Pour en savoir plus, consultez le support du GPU dans la documentation eksctl.

Si vous décidez d'utiliser l'opérateur EKS Accelerated AMIs et NVIDIA GPU pour gérer des composants tels que le plug-in de périphérique NVIDIA Kubernetes, prenez note de désactiver la gestion du pilote GPU NVIDIA et du kit d'outils NVIDIA Container conformément aux pilotes de GPU NVIDIA préinstallés et à la documentation NVIDIA Container Toolkit NVIDIA.

Pour vérifier que le plug-in de périphérique NVIDIA est actif et GPUs correctement exposé, exécutez :

```
kubectl describe node | grep nvidia.com/gpu
```

Cette commande vérifie si la `nvidia.com/gpu` ressource est dans la capacité du nœud et si les ressources sont allouables. Par exemple, un nœud doté d'un processeur graphique devrait s'afficher`nvidia.com/gpu: 1`. Consultez le guide de planification du GPU Kubernetes pour plus d'informations.

Utilisez de nombreux types d'instances EC2

L'utilisation d'autant de types d'instances EC2 que possible est une bonne pratique importante pour l'évolutivité sur Amazon EKS, comme indiqué dans la the section called “Plan de données” section. Cette recommandation s'applique également aux instances dotées d'un matériel accéléré (par exemple, GPUs). Si vous créez un cluster qui n'utilise qu'un seul type d'instance et que vous essayez d'augmenter le nombre de nœuds au-delà de la capacité de la région, vous risquez de recevoir une erreur ICE (insuffisance de capacité), indiquant qu'aucune instance n'est disponible. Il est important de comprendre les caractéristiques uniques de vos AI/ML charges de travail avant de procéder à une diversification arbitraire. Passez en revue les types d'instances disponibles à l'aide de l'outil EC2 Instance Type Explorer pour générer une liste des types d'instances correspondant à vos besoins de calcul spécifiques, et évitez de limiter arbitrairement le type d'instances pouvant être utilisées dans votre cluster.

Les instances de calcul accéléré sont proposées dans différents modèles d'achat pour s'adapter aux charges de travail à court, moyen terme et en régime permanent. Pour les charges de travail flexibles et tolérantes aux pannes à court terme, pour lesquelles vous souhaiteriez éviter de faire une réservation, optez pour les instances Spot. Les blocs de capacité, les instances à la demande et les plans d'économie vous permettent de fournir des instances de calcul accélérées pour une durée de

charge de travail à moyen et long terme. Pour augmenter les chances d'accéder à la capacité requise dans le cadre de votre option d'achat préférée, il est recommandé d'utiliser une liste variée de types d'instances et de zones de disponibilité. Sinon, si vous rencontrez un modèle ICEs d'achat spécifique, réessayez en utilisant un autre modèle.

Exemple L'exemple suivant montre comment permettre à un Karpenter de NodePool provisionner des instances G et P supérieures aux générations 3 (par exemple, p3). Pour plus d'informations, consultez la section Capacité de mise à l'échelle.

```
- key: karpenter.k8s.aws/instance-category
  operator: In
  values: ["g", "p"] # Diversifies across G-series and P-series
- key: karpenter.k8s.aws/instance-generation
  operator: Gt
  values: ["3"] # Selects instance generations greater than 3
```

Pour en savoir plus sur l'utilisation des instances Spot pour les GPU, consultez « Envisagez d'utiliser des instances Spot Amazon EC2 GPUs pour Karpenter » ci-dessous.

Envisagez d'utiliser des instances Spot Amazon EC2 pour Karpenter GPUs

Les instances Amazon EC2 Spot vous permettent de tirer parti de la capacité EC2 inutilisée dans le cloud AWS et sont disponibles avec une réduction allant jusqu'à 90 % par rapport aux prix à la demande. Les instances Spot Amazon EC2 peuvent être interrompues moyennant un préavis de deux minutes lorsqu'EC2 a besoin de récupérer sa capacité. Pour plus d'informations, consultez la section Instances Spot dans le guide de l'utilisateur Amazon EC2. Amazon EC2 Spot peut être un excellent choix pour les charges de travail tolérantes aux pannes, apatrides et flexibles (durée et type d'instance). Pour en savoir plus sur les circonstances dans lesquelles utiliser les instances Spot, consultez les meilleures pratiques en matière d'instances Spot EC2. Vous pouvez également utiliser des instances Spot pour les AI/ML charges de travail si elles sont compatibles avec Spot.

Cas d'utilisation

Les charges de travail optimisées peuvent être le Big Data, les charges de travail conteneurisées, le CI/CD, les serveurs Web sans état, le calcul haute performance (HPC) et les charges de travail de rendu. Les instances Spot ne sont pas adaptées aux charges de travail rigides, dynamiques, intolérantes aux pannes ou étroitement couplées entre les nœuds d'instance (par exemple, les charges de travail comportant des processus parallèles dont les calculs dépendent fortement les uns des autres et nécessitent une communication constante entre les nœuds, telles que les applications

informatiques à hautes performances basées sur le protocole MPI, telles que la dynamique des fluides ou les bases de données distribuées présentant des interdépendances complexes). Voici les cas d'utilisation spécifiques que nous recommandons (sans ordre particulier) :

- Inférence en ligne en temps réel : utilisez les instances Spot pour une mise à l'échelle optimisée en termes de coûts pour vos charges de travail d'inférence en temps réel, à condition que vos charges de travail soient adaptées à la localisation. En d'autres termes, le temps d'inférence est inférieur à deux minutes, l'application tolère les pannes et peut s'exécuter sur différents types d'instances. Garantissez une haute disponibilité grâce à la diversité des instances (par exemple, entre plusieurs types d'instances et zones de disponibilité) ou à des réservations, tout en mettant en œuvre une tolérance aux pannes au niveau de l'application pour gérer les interruptions ponctuelles potentielles.
- Réglage des hyperparamètres : utilisez les instances Spot pour exécuter des tâches de réglage exploratoires de manière opportuniste, car les interruptions peuvent être tolérées sans perte significative, en particulier pour les expériences de courte durée.
- Augmentation des données : utilisez les instances Spot pour effectuer des tâches de prétraitement et d'augmentation des données qui peuvent redémarrer à partir des points de contrôle en cas d'interruption, ce qui les rend idéales pour la disponibilité variable de Spot.
- Modèles de réglage précis : utilisez des instances Spot pour effectuer des réglages précis grâce à des mécanismes de point de contrôle robustes permettant de revenir au dernier état enregistré, minimisant ainsi l'impact des interruptions d'instance.
- Inférence par lots : utilisez les instances Spot pour traiter de gros lots de demandes d'inférence hors ligne de non-real-time manière à ce que les tâches puissent être interrompues et reprises, en optimisant les économies réalisées par Spot et en gérant les interruptions potentielles par le biais de nouvelles tentatives ou de la diversification.
- Sous-ensembles de formation opportunistes : utilisez des instances ponctuelles pour des charges de travail de formation marginales ou expérimentales (par exemple, des modèles plus petits avec 10 millions de paramètres), où les interruptions sont acceptables et où des optimisations de l'efficacité telles que la diversification entre les types d'instances ou les régions peuvent être appliquées, bien que cela ne soit pas recommandé pour la formation à l'échelle de la production en raison de perturbations potentielles.

Considérations

Pour utiliser des instances Spot pour des charges de travail accélérées sur Amazon EKS, il convient de prendre en compte un certain nombre de points essentiels (sans ordre particulier) :

- Utilisez Karpenter pour gérer les instances Spot avec la consolidation avancée activée. En spécifiant karpenter.sh/capacity type comme « spot » dans votre Karpenter NodePool, Karpenter fournira des instances Spot par défaut sans aucune configuration supplémentaire. Toutefois, pour permettre une Spot-to-Spot consolidation avancée, qui remplace les nœuds Spot sous-utilisés par des alternatives Spot moins coûteuses, vous devez activer le SpotToSpotConsolidation portail des fonctionnalités en définissant --feature-gates SpotToSpotConsolidation =true dans les arguments du contrôleur Karpenter ou via la variable d'environnement FEATURE_GATES. Karpenter utilise la stratégie price-capacity-optimizedd'allocation pour approvisionner les instances EC2. En fonction des NodePool exigences et des contraintes liées aux modules, Karpenter intègre des modules non planifiables et envoie un ensemble varié de types d'instances à l'API Amazon EC2 Fleet. Vous pouvez utiliser l'outil EC2 Instance Type Explorer pour générer une liste de types d'instances correspondant à vos besoins de calcul spécifiques.
- Assurez-vous que les charges de travail sont apatrides, tolérantes aux pannes et flexibles. Les charges de travail doivent être apatrides, tolérantes aux pannes et flexibles en termes de taille. instance/GPU Cela permet une reprise fluide après une interruption de Spot, et la flexibilité de l'instance vous permet de rester sur Spot plus longtemps. Activez la gestion des interruptions ponctuelles dans Karpenter en configurant la valeur Settings.interruptionQueue Helm avec le nom de la file d'attente AWS SQS pour détecter les événements d'interruption ponctuelle. Par exemple, lors de l'installation via Helm, utilisez --set « settings.interruptionQueue=$ {CLUSTER_NAME} ». Pour voir un exemple, consultez le guide Getting Started with Karpenter. Lorsque Karpenter remarque un événement d'interruption Spot, il déconnecte, dilue, vide et arrête automatiquement le ou les nœuds avant l'événement d'interruption afin de maximiser le délai de grâce de résiliation des pods. Dans le même temps, Karpenter démarrera immédiatement un nouveau nœud afin qu'il soit prêt dès que possible.
- Évitez de trop restreindre la sélection du type d'instance. Vous devez éviter autant que possible de restreindre les types d'instances. En ne limitant pas les types d'instances, vous augmentez les chances d'acquérir de la capacité Spot à grande échelle avec une fréquence d'interruption des instances Spot plus faible à moindre coût. Par exemple, évitez de vous limiter à des types spécifiques (par exemple, g5.xlarge). Envisagez de spécifier un ensemble diversifié de catégories et de générations d'instances à l'aide de clés telles que karpenter.k8s. aws/instance-category and karpenter.k8s.aws/instance-generation. Karpenter enables easier diversification of on-demand and Spot instance capacity across multiple instance types and Availability Zones (AZs). Moreover, if your AI/MLLa charge de travail nécessite un nombre spécifique ou limité d'accélérateurs, mais elle est flexible selon les régions. Vous pouvez utiliser le Spot Placement Score pour identifier dynamiquement la région optimale pour déployer votre charge de travail avant le lancement.

- Élargissez les NodePool exigences pour inclure un plus grand nombre de familles d'instances EC2 similaires. Chaque pool d'instances Spot est constitué d'une capacité d'instance EC2 inutilisée pour un type d'instance spécifique dans une zone de disponibilité (AZ) spécifique. Lorsque Karpenter essaie de provisionner un nouveau nœud, il sélectionne un type d'instance qui correspond aux exigences NodePool de Karpenter. Si aucun type d'instance compatible ne possède de capacité Spot dans aucune zone de disponibilité, le provisionnement échoue. Pour éviter ce problème, autorisez les instances NVIDIA de la série G (génération 4 ou supérieure) de NVIDIA, quelle que soit leur taille et leur zone de disponibilité (AZs), tout en tenant compte des besoins matériels tels que la mémoire GPU ou le ray tracing. Comme les instances peuvent être de différents types, vous devez vous assurer que votre charge de travail est capable de s'exécuter sur chaque type et que les performances que vous obtenez répondent à vos besoins.
- Tirez parti de toutes les zones de disponibilité d'une région. La capacité disponible varie en fonction de la zone de disponibilité (AZ). Un type d'instance spécifique peut être indisponible dans une zone mais abondant dans une autre. Chaque combinaison unique d'un type d'instance et d'une zone de disponibilité constitue un pool de capacité ponctuel distinct. En demandant des capacités AZs dans l'ensemble d'une région répondant à vos NodePool besoins en matière de Karpenter, vous recherchez efficacement plus de pools à la fois. Cela maximise le nombre de pools de capacités Spot et augmente donc la probabilité d'acquérir de la capacité Spot. Pour ce faire, dans votre NodePool configuration, omettez complètement la clé topology.kubernetes.io/zone pour permettre à Karpenter de sélectionner parmi toutes les options disponibles AZs dans la région, ou listez explicitement à l' AZs aide de l'opérateur : In et fournissez les valeurs (par exemple, us-west-2a).
- Envisagez d'utiliser le Spot Placement Score (SPS) pour avoir une idée de la probabilité d'accéder avec succès à la capacité requise à l'aide d'instances Spot. Le Spot Placement Score (SPS) est un outil qui fournit un score pour vous aider à évaluer les chances de succès d'une demande Spot. Lorsque vous utilisez SPS, vous spécifiez d'abord vos besoins en calcul pour vos instances Spot, puis Amazon EC2 renvoie les 10 principales régions ou zones de disponibilité AZs () dans lesquelles votre demande Spot est susceptible de réussir. Les régions et les zones de disponibilité sont évaluées sur une échelle de 1 à 10. Un score de 10 indique que votre demande Spot a de fortes chances de réussir, mais qu'il n'est pas garanti qu'elle aboutisse. Un score de 1 indique que votre demande Spot a peu de chances d'aboutir. Le même score peut être renvoyé pour différentes régions ou zones de disponibilité. Pour en savoir plus, consultez les instructions relatives à la création d'un tableau de bord de suivi des scores de placement ponctuel sur AWS. Comme la capacité du spot fluctue en permanence, SPS vous aidera à identifier la combinaison de types d'instances et de régions la mieux adaptée à vos contraintes de charge de travail (flexibilité, performance, taille, etc.). AZs Si votre AI/ML charge de travail nécessite un nombre spécifique ou

limité d'accélérateurs, mais qu'elle est flexible entre les régions, vous pouvez utiliser le score de placement Spot pour identifier dynamiquement la région optimale pour déployer votre charge de travail avant le lancement. Pour vous aider à déterminer automatiquement la probabilité d'acquérir de la capacité Spot, nous vous proposons des conseils pour créer un tableau de bord de suivi SPS. Cette solution surveille les scores SPS au fil du temps à l'aide d'une configuration YAML pour des configurations diversifiées (par exemple, les exigences relatives aux instances GPUs), stocke les métriques et fournit des tableaux de bord pour comparer les configurations. CloudWatch Définissez des tableaux de bord par charge de travail pour évaluer les besoins en matière de vCPU, de mémoire et de GPU, afin de garantir des configurations optimales pour les clusters EKS, notamment en envisageant d'utiliser d'autres régions AWS. Pour en savoir plus, consultez Comment fonctionne le score de placement Spot.

- Gérez les interruptions et testez Spot avec élégance. Pour un pod dont la période de résiliation est supérieure à deux minutes, l'ancien nœud sera interrompu avant que ces pods ne soient reprogrammés, ce qui peut avoir un impact sur la disponibilité de la charge de travail. Tenez compte du préavis d'interruption ponctuelle de deux minutes lors de la conception de vos applications, implémentez le point de contrôle dans les applications de longue durée (par exemple, enregistrez la progression vers le stockage persistant comme Amazon S3) pour reprendre après une interruption, prolongez le terminationGracePeriod délai de 30 secondes dans les spécifications du Pod pour laisser plus de temps à un arrêt progressif et gérez les interruptions en utilisant les signaux and/or SIGTERM de votre application pour des activités d'arrêt progressif telles que le nettoyage, la sauvegarde de l'état et la fermeture de connexion. Pour les charges de travail en temps réel, où le temps de mise à l'échelle est important et où les charges de travail prennent plus de deux minutes pour que l'application soit prête à traiter le trafic, envisagez d'optimiser les temps de démarrage des conteneurs et de chargement des modèles de machine learning en passant en revue the section called “Stockage” les meilleures pratiques. the section called “Performance” Pour tester un nœud de remplacement, utilisez AWS Fault Injection Service (FIS) pour simuler des interruptions ponctuelles.

Outre ces bonnes pratiques Spot fondamentales, tenez compte de ces facteurs lors de la gestion des charges de travail du GPU sur Amazon EKS. Contrairement aux charges de travail basées sur le processeur, les charges de travail sur le processeur graphique sont particulièrement sensibles aux détails matériels tels que les capacités du processeur graphique et la mémoire disponible du processeur graphique. Les charges de travail du GPU peuvent être limitées par les types d'instances qu'ils peuvent utiliser, avec moins d'options disponibles par rapport à. CPUs Dans un premier temps, déterminez si votre charge de travail est flexible en termes d'instance. Si vous ne savez pas combien de types d'instances votre charge de travail peut utiliser, testez-les individuellement pour garantir leur

compatibilité et leur fonctionnalité. Déterminez dans quelle mesure vous pouvez être flexible pour diversifier autant que possible, tout en confirmant que la diversification permet de maintenir la charge de travail et en comprenant les impacts sur les performances (par exemple, sur le débit ou le délai d'exécution). Dans le cadre de la diversification de vos charges de travail, tenez compte des points suivants :

- Vérifiez la compatibilité entre CUDA et le framework. Vos charges de travail GPU peuvent être optimisées pour du matériel ou des types de GPU spécifiques (par exemple, V100 en p3 contre A100 en p4), ou écrites pour des versions CUDA spécifiques pour des bibliothèques telles que. Assurez-vous donc de vérifier la TensorFlow compatibilité de vos charges de travail. Cette compatibilité est essentielle pour éviter les erreurs d'exécution, les pannes, les défaillances d'accélération du GPU (par exemple, des versions CUDA incompatibles avec des frameworks similaires PyTorch ou TensorFlow susceptibles d'empêcher l'exécution) ou pour pouvoir tirer parti de fonctionnalités matérielles telles que FP16/precision. INT8
- Mémoire GPU. Assurez-vous d'évaluer les besoins en mémoire de vos modèles et d'établir le profil de l'utilisation de la mémoire de votre modèle pendant l'exécution à l'aide d'outils tels que l'exportateur DCGM et de définir la mémoire GPU minimale requise pour le type d'instance dans des étiquettes connues telles que karpenter.k8s.aws/. instance-gpu-memory La VRAM du GPU varie selon le type d'instance (par exemple, NVIDIA T4 dispose de 16 Go, A10G de 24 Go, V100 de 16 à 32 Go) et les modèles ML (par exemple, les modèles de langage volumineux) peuvent dépasser la mémoire disponible, provoquant des erreurs (OOM) ou des pannes. out-of-memory Pour les instances Spot dans EKS, cela peut limiter la diversification. Par exemple, vous ne pouvez pas inclure de types de mémoire VRAM inférieure si votre modèle ne convient pas, ce qui peut limiter l'accès aux pools de capacité et augmenter le risque d'interruption. Notez que pour l'inférence à un seul GPU ou à un seul nœud (par exemple, plusieurs pods planifiés sur le même nœud pour utiliser ses ressources GPU), cela peut limiter la diversification, car vous ne pouvez inclure que des types d'instance dotés d'une VRAM suffisante dans votre configuration Spot.
- Précision et performance en virgule flottante. Les architectures GPU Nvidia n'ont pas toutes la même précision en virgule flottante (par exemple, FP16/INT8). Évaluez les performances des types de base (CUDA/Tensor/RT) et la précision à virgule flottante requises pour vos charges de travail. Utiliser un processeur graphique moins cher et moins performant ne signifie pas nécessairement qu'il est meilleur. Pensez donc à évaluer les performances en termes de travail effectué dans un délai précis afin de comprendre l'impact de la diversification.

Scénario : Diversification pour les charges de travail d'inférence en temps réel

Pour une charge de travail d'inférence en ligne en temps réel sur les instances Spot, vous pouvez configurer un Karpenter NodePool afin de diversifier les familles et les générations d'instances GPU compatibles. Cette approche garantit une haute disponibilité en s'appuyant sur plusieurs pools de Spot, tout en maintenant les performances grâce à des contraintes sur les capacités, la mémoire et l'architecture du GPU. Il prend en charge l'utilisation d'alternatives lorsque la capacité de l'instance est limitée, en minimisant les interruptions et en optimisant la latence d'inférence. Cet exemple NodePool indique qu'il faut utiliser des instances des séries g et p supérieures à 3, dotées de plus de 20 Go de mémoire graphique.

Exemple

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-inference-spot
spec:
  template:
    metadata:
      labels:
        role: gpu-spot-worker
    spec:
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot"] # Use Spot Instances
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["g", "p"] # Diversifies across G-series and P-series
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["3"] # Selects instance generations greater than 3
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"] # Specifies AMD64 architecture, compatible with NVIDIA GPUs
        - key: karpenter.k8s.aws/instance-gpu-memory
          operator: Gt
          values: ["20480"] # Ensures more than 20GB (20480 MiB) total GPU memory
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      nodeClassRef:
        name: gpu-inference-ec2
```

```
        group: karpenter.k8s.aws
        kind: EC2NodeClass
      expireAfter: 720h
  limits:
    cpu: 100
    memory: 100Gi
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
    consolidateAfter: 5m # Enables consolidation of underutilized nodes after 5 minutes
```

Mettre en œuvre le point de contrôle pour les tâches de formation de longue durée

Le point de contrôle est une technique de tolérance aux pannes qui consiste à enregistrer périodiquement l'état d'un processus, afin de lui permettre de reprendre à partir du dernier point enregistré en cas d'interruption. Dans le domaine de l'apprentissage automatique, il est généralement associé à la formation, dans le cadre de laquelle les tâches de longue durée peuvent permettre de réduire le poids des modèles et d'optimiser les états des optimiseurs afin de reprendre l'entraînement après des défaillances, telles que des problèmes matériels ou des interruptions d'instance ponctuelle.

Vous utilisez des points de contrôle pour enregistrer l'état des modèles d'apprentissage automatique (ML) pendant l'entraînement. Les points de contrôle sont des instantanés du modèle et peuvent être configurés par les fonctions de rappel de cadres ML. Vous pouvez utiliser les points de contrôle enregistrés pour redémarrer une tâche d'entraînement à partir du dernier point de contrôle enregistré. À l'aide des points de contrôle, vous enregistrez les instantanés de votre modèle en cours de formation en raison d'une interruption inattendue de la tâche ou de l'instance de formation. Cela vous permet de reprendre l'entraînement du modèle à l'avenir à partir d'un point de contrôle. Outre la mise en œuvre d'un système de résilience des nœuds, nous recommandons de mettre en œuvre des points de contrôle pour atténuer l'impact des interruptions, notamment celles causées par des pannes matérielles ou des interruptions des instances Spot Amazon EC2.

Sans points de contrôle, les interruptions peuvent entraîner une perte de temps de calcul et de progression, ce qui est coûteux pour les tâches de formation de longue durée. Le point de contrôle permet aux tâches d'enregistrer régulièrement leur état (par exemple, les poids du modèle et les états de l'optimiseur) et de reprendre à partir du dernier point de contrôle (dernier lot traité) après une interruption. Pour implémenter le point de contrôle, concevez votre application de manière à traiter les données par lots importants et à enregistrer les résultats intermédiaires dans un stockage persistant, tel qu'un bucket Amazon S3 via le pilote CSI Mountpoint for Amazon S3 pendant que la formation progresse.

Cas d'utilisation

Le point de contrôle est particulièrement utile dans des scénarios spécifiques pour équilibrer la tolérance aux pannes avec la surcharge de performance. Envisagez d'utiliser le point de contrôle dans les cas suivants :

- La durée des tâches dépasse quelques heures : pour les tâches de formation de longue durée (par exemple, > 1 à 2 heures pour les petits modèles, ou days/weeks pour les grands modèles de base comportant des milliards de paramètres), où la perte de progression due à des interruptions est coûteuse. Les emplois plus courts ne justifient peut-être pas les I/O frais généraux.
- Pour les instances Spot ou les pannes matérielles : dans les environnements sujets à des interruptions, tels qu'EC2 Spot (préavis de 2 minutes) ou à des défaillances matérielles (par exemple, des erreurs de mémoire du GPU), le point de contrôle permet une reprise rapide, ce qui rend Spot viable et permet de réaliser des économies sur les charges de travail tolérantes aux pannes.
- Entraînement distribué à grande échelle : pour les configurations comportant hundreds/thousands des accélérateurs (par exemple, >100 GPUs), où le temps moyen entre les défaillances diminue de façon linéaire avec l'échelle. Utilisez le model/data parallélisme pour gérer l'accès simultané aux points de contrôle et éviter les redémarrages complets.
- Modèles à grande échelle nécessitant de fortes ressources : dans le cadre de la formation LLM à l'échelle du pétaoctet, où les défaillances sont inévitables en raison de la taille du cluster, les approches à plusieurs niveaux (local rapide toutes les 5 à 30 minutes pour les transitoires, durable toutes les heures pour les pannes majeures) optimisent le temps de restauration par rapport à l'efficacité.

Utilisez les blocs de capacité ML pour garantir la capacité des instances P et Trainium

Les blocs de capacité pour le ML vous permettent de réserver des instances GPU très recherchées, en particulier des instances P (par exemple, p6-b200, p5, p5e, p5en, p4d, p4de) et des instances Trainium (par exemple, trn1, trn2), pour démarrer presque immédiatement ou à une date future afin de prendre en charge vos charges de travail d'apprentissage automatique (ML) de courte durée. Ces réservations sont idéales pour garantir la capacité nécessaire aux tâches de calcul intensives, telles que la formation et le réglage des modèles. La tarification d'EC2 Capacity Blocks comprend des frais de réservation et des frais de système d'exploitation. Pour en savoir plus sur la tarification, consultez la section relative à la tarification des blocs de capacité EC2 pour le ML.

Pour réserver GPUs pour les AI/ML charges de travail sur Amazon EKS afin de garantir une capacité prévisible, nous vous recommandons d'utiliser les blocs de capacité ML pour les réservations de capacité à court terme ou à la demande (ODCRs) pour une assurance de capacité à usage général.

- Les ODCR vous permettent de réserver de la capacité d'instance EC2 (par exemple, des instances GPU telles que g5 ou p5) dans une zone de disponibilité spécifique pendant une certaine durée, garantissant ainsi la disponibilité, même en cas de forte demande. ODCRs vous n'avez aucun engagement à long terme, mais vous payez le tarif à la demande pour la capacité réservée, qu'elle soit utilisée ou inutilisée. Dans EKS, ODCRs sont pris en charge par des types de nœuds tels que Karpenter et des groupes de nœuds gérés. Pour établir des priorités ODCRs dans Karpenter, configurez le NodeClass pour utiliser le `capacityReservationSelectorTerms` champ. Consultez la NodePools documentation de Karpenter.
- Les blocs de capacité sont un mécanisme de réservation spécialisé pour les instances GPU (par exemple, p5, p4d) ou Trainium (trn1, trn2), conçu pour les charges de travail ML à court terme telles que la formation de modèles, le réglage fin ou l'expérimentation. Vous réservez des capacités pour une période définie (généralement de 24 heures à 182 jours) à compter d'une date future, en ne payant que pour le temps réservé. Ils sont prépayés, nécessitent une planification préalable en fonction des besoins en capacité et ne prennent pas en charge la mise à l'échelle automatique, mais ils sont hébergés dans EC2 pour un réseau à faible latence. UltraClusters Ils facturent uniquement pour la période réservée. Pour en savoir plus, reportez-vous à la section Rechercher et acheter des blocs de capacité, ou commencez par configurer des groupes de nœuds gérés avec des blocs de capacité en suivant les instructions de la section Créer un groupe de nœuds gérés avec des blocs de capacité pour le ML.

Réservez de la capacité via l'AWS Management Console et configurez vos nœuds pour utiliser des blocs de capacité ML. Planifiez les réservations en fonction des plannings de charge de travail et testez-les dans un cluster intermédiaire. Reportez-vous à la documentation sur les blocs de capacité pour plus d'informations.

Envisagez des réservations de capacité à la demande, Amazon EC2 ponctuelles ou à la demande (ODCR) pour les instances G Amazon EC2

Pour les instances G Amazon EC2, considérez les différentes options d'achat proposées par On-Demand, les instances ponctuelles Amazon EC2 et les réservations de capacité à la demande. ODCRsvous permettent de réserver la capacité d'instance EC2 dans une zone de disponibilité spécifique pendant une certaine durée, garantissant ainsi la disponibilité même en cas de forte demande. Contrairement aux blocs de capacité ML, qui ne sont disponibles que pour les instances P et Trainium, les ODCR peuvent être utilisés pour un plus large éventail de types d'instances, y compris les instances G, ce qui les rend adaptés aux charges de travail nécessitant des capacités GPU différentes, telles que l'inférence ou les graphiques. Lorsque vous utilisez des instances Spot

Amazon EC2, il est essentiel de pouvoir varier les types d'instances, les tailles et les zones de disponibilité pour rester sur place plus longtemps.

ODCRs vous n'avez aucun engagement à long terme, mais vous payez le tarif à la demande pour la capacité réservée, qu'elle soit utilisée ou inutilisée. ODCRs peuvent être créés pour une utilisation immédiate ou planifiés pour une date future, offrant ainsi une flexibilité dans la planification des capacités. Dans Amazon EKS, ODCRs sont pris en charge par des types de nœuds tels que Karpenter et des groupes de nœuds gérés. Pour établir des priorités ODCRs dans Karpenter, configurez le NodeClass pour utiliser le `capacityReservationSelectorTerms` champ. Consultez la NodePools documentation de Karpenter. Pour plus d'informations sur la création ODCRs, notamment sur les commandes CLI, reportez-vous à la section Démarrage de la réservation de capacité à la demande.

Envisagez d'autres types et tailles d'instances accélérés

La sélection de l'instance accélérée et de la taille appropriées est essentielle pour optimiser à la fois les performances et les coûts de vos charges de travail ML sur Amazon EKS. Par exemple, les différentes familles d'instances de GPU ont des performances et des capacités différentes, telles que la mémoire GPU. Pour vous aider à choisir l'option la plus rentable, consultez les instances de GPU disponibles sur la page Types d'instances EC2 sous Accelerated Computing. Évaluez plusieurs types et tailles d'instances afin de trouver celle qui convient le mieux à vos exigences spécifiques en matière de charge de travail. Tenez compte de facteurs tels que le nombre GPUs, la mémoire et les performances du réseau. En sélectionnant avec soin le type et la taille d'instance GPU appropriés, vous pouvez optimiser l'utilisation des ressources et la rentabilité de vos clusters EKS.

Si vous utilisez une instance de GPU dans un nœud EKS, le `nvidia-device-plugin-daemonset` pod figurera par défaut dans l'`kube-system`espace de noms. Pour savoir rapidement si vous utilisez pleinement le ou les GPU de votre instance, vous pouvez utiliser nvidia-smi comme indiqué ici :

```
kubectl exec nvidia-device-plugin-daemonset-xxxxx \
  -n kube-system -- nvidia-smi \
  --query-
gpu=index,power.draw,power.limit,temperature.gpu,utilization.gpu,utilization.memory,memory.free
 \
  --format=csv -l 5
```

- Si `utilization.memory` cette valeur est proche de 100 %, cela signifie que vos codes sont probablement liés à la mémoire. Cela signifie que le GPU (mémoire) est pleinement utilisé, mais cela pourrait suggérer qu'une optimisation plus poussée des performances devrait être étudiée.

- Si la `utilization.gpu` valeur est proche de 100 %, cela ne signifie pas nécessairement que le GPU est pleinement utilisé. Une meilleure métrique à examiner est le ratio de `power.draw` à`power.limit`. Si ce ratio est supérieur ou égal à 100 %, vos codes utilisent pleinement la capacité de calcul du GPU.
- Le `-l 5` drapeau indique de sortir les métriques toutes les 5 secondes. Dans le cas d'un seul type d'instance de GPU, l'indicateur de requête d'index n'est pas nécessaire.

Pour en savoir plus, consultez les instances GPU dans la documentation AWS.

Optimisez l'allocation des ressources GPU grâce au découpage temporel, au MIG et à l'allocation fractionnée du GPU

Les limites de ressources statiques dans Kubernetes (par exemple, le nombre de processeurs, de mémoire, de GPU) peuvent entraîner un provisionnement excessif ou une sous-utilisation, en particulier pour les charges de travail dynamiques telles que l'inférence. AI/ML Il est important de sélectionner le bon processeur graphique. Pour les charges de travail à faible volume ou trop élevées, le découpage en tranches de temps permet à plusieurs charges de travail de partager un seul GPU en partageant ses ressources de calcul, ce qui peut améliorer l'efficacité et réduire le gaspillage. Le partage du GPU peut être réalisé grâce à différentes options :

- Tirez parti des sélecteurs de nœuds et de l'affinité des nœuds pour influencer la planification : assurez-vous que les nœuds provisionnés et les pods sont planifiés en fonction de la GPUs charge de travail (par exemple,) `karpenter.k8s.aws/instance-gpu-name: "a100"`
- Tranchage dans le temps : planifie les charges de travail pour partager les ressources de calcul d'un GPU au fil du temps, permettant ainsi une exécution simultanée sans partitionnement physique. C'est idéal pour les charges de travail soumises à des exigences de calcul variables, mais qui peuvent ne pas isoler la mémoire.
- GPU multi-instance (MIG) : MIG permet de partitionner un seul GPU NVIDIA en plusieurs instances isolées et est compatible avec NVIDIA Ampere (par exemple, GPU A100), NVIDIA Hopper (par exemple, GPU H100) et NVIDIA Blackwell (par exemple, Blackwell). GPUs GPUs Chaque instance MIG reçoit des ressources de calcul et de mémoire dédiées, ce qui permet le partage des ressources dans des environnements mutualisés ou des charges de travail nécessitant des garanties de ressources, ce qui vous permet d'optimiser l'utilisation des ressources du GPU, y compris des scénarios tels que le service de plusieurs modèles avec des tailles de lots différentes par le biais d'un découpage temporel.

- Allocation fractionnée du GPU : utilise une planification logicielle pour allouer des parties du calcul ou de la mémoire d'un GPU aux charges de travail, offrant ainsi une flexibilité pour les charges de travail dynamiques. Le planificateur NVIDIA KAI, qui fait partie de la plateforme Run:AI, permet cela en permettant aux pods de demander des ressources GPU fractionnées.

Pour activer ces fonctionnalités dans EKS, vous pouvez déployer le plugin NVIDIA Device, qui expose les ressources GPUs sous forme de ressources planifiables et prend en charge le time-slicing et le MIG. Pour en savoir plus, consultez les sections Time-Slicing GPUs dans Kubernetes et Partage de GPU sur Amazon EKS avec le time-slicing NVIDIA et les instances EC2 accélérées.

Exemple

Par exemple, pour activer le time-slicing avec le plugin NVIDIA Device :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nvidia-device-plugin-config
  namespace: kube-system
data:
  config.yaml: |
    version: v1
    sharing:
      timeSlicing:
        resources:
        - name: nvidia.com/gpu
          replicas: 4  # Allow 4 pods to share each GPU
```

Exemple

Par exemple, pour utiliser KAI Scheduler pour l'allocation fractionnée du GPU, déployez-le aux côtés de l'opérateur GPU NVIDIA et spécifiez les ressources GPU fractionnées dans la spécification du pod :

```
apiVersion: v1
kind: Pod
metadata:
  name: fractional-gpu-pod-example
  annotations:
    gpu-fraction: "0.5"  # Annotation for 50% GPU
```

```
  labels:
    runai/queue: "default"  # Required queue assignment
spec:
  containers:
  - name: ml-workload
    image: nvcr.io/nvidia/pytorch:25.04-py3
    resources:
      limits:
        nvidia.com/gpu: 1
  nodeSelector:
    nvidia.com/gpu: "true"
  schedulerName: kai-scheduler
```

Résilience des nœuds et gestion des tâches de formation

Mettre en œuvre des contrôles de santé des nœuds avec restauration automatique

Pour les tâches de formation distribuées sur Amazon EKS qui nécessitent des communications fréquentes entre nœuds, telles que la formation de modèles multi-GPU sur plusieurs nœuds, les problèmes matériels tels que les pannes du GPU ou de l'EFA peuvent perturber les tâches de formation. Ces interruptions peuvent entraîner une perte de progression en matière de formation et une augmentation des coûts, en particulier pour les AI/ML charges de travail de longue durée qui reposent sur un matériel stable.

Pour renforcer la résilience face aux défaillances matérielles, telles que les pannes de GPU dans les clusters EKS exécutant des charges de travail GPU, nous vous recommandons de tirer parti de l'agent de surveillance des nœuds EKS avec Auto Repair ou d'Amazon SageMaker HyperPod. Alors que l'agent de surveillance des nœuds EKS avec réparation automatique fournit des fonctionnalités telles que la surveillance de l'état des nœuds et la réparation automatique à l'aide des mécanismes Kubernetes standard, il SageMaker HyperPod offre une résilience ciblée et des fonctionnalités supplémentaires spécialement conçues pour la formation ML à grande échelle, telles que des bilans de santé approfondis et la reprise automatique des tâches.

- L'agent de surveillance des nœuds EKS avec Node Auto Repair surveille en permanence l'état des nœuds en lisant les journaux et en appliquant NodeConditions des conditions standard telles que `Ready` les conditions spécifiques au matériel accéléré afin d'identifier les problèmes tels que les pannes de GPU ou de réseau. Lorsqu'un nœud est jugé insalubre, Node Auto Repair le cordon et le remplace par un nouveau nœud. La replanification des pods et le redémarrage des tâches s'appuient sur les mécanismes standard de Kubernetes et sur la politique de redémarrage des tâches.

- Les contrôles de santé SageMaker HyperPodapprofondis et l'agent de surveillance de l'état de santé surveillent en permanence l'état de santé des instances basées sur le GPU et Trainium. Il est adapté aux AI/ML charges de travail et utilise des étiquettes (par exemple node-health-status) pour gérer l'état des nœuds. Lorsqu'un nœud est jugé défectueux, HyperPod déclenche le remplacement automatique du matériel défectueux, par exemple GPUs. Il détecte les défaillances liées au réseau pour EFA grâce à ses contrôles de santé de base par défaut et prend en charge la reprise automatique des tâches de formation interrompues, permettant ainsi aux tâches de continuer depuis le dernier point de contrôle, minimisant ainsi les interruptions pour les tâches de machine learning à grande échelle.

Pour l'agent de surveillance des nœuds EKS avec réparation automatique et les SageMaker HyperPod clusters utilisant EFA, afin de surveiller les métriques spécifiques à EFA, telles que les erreurs RDMA (Remote Direct Memory Access) et les pertes de paquets, assurez-vous que le pilote AWS EFA est installé. En outre, nous vous recommandons de déployer le module complémentaire CloudWatch Observability ou d'utiliser des outils tels que DCGM Exporter avec Prometheus et Grafana pour surveiller l'EFA, le GPU et, pour SageMaker HyperPod les indicateurs spécifiques liés à ses fonctionnalités.

Désactiver Karpenter Consolidation pour les charges de travail sensibles aux interruptions

Pour les charges de travail sensibles aux interruptions, telles que le traitement, les tâches de AI/ML prédiction à grande échelle ou la formation, nous recommandons d'ajuster les politiques de consolidation de Karpenter afin d'éviter les perturbations lors de l'exécution des tâches. La fonction de consolidation de Karpenter optimise automatiquement les coûts des clusters en mettant fin aux nœuds sous-utilisés ou en les remplaçant par des alternatives moins coûteuses. Cependant, même lorsqu'une charge de travail utilise entièrement un GPU, Karpenter peut consolider les nœuds s'il identifie un type d'instance de taille appropriée et moins coûteux répondant aux exigences du module, ce qui entraîne des interruptions de travail.

La politique de `WhenEmptyOrUnderutilized` consolidation peut mettre fin aux nœuds prématurément, ce qui entraîne des délais d'exécution plus longs. Par exemple, les interruptions peuvent retarder la reprise des tâches en raison de la replanification des modules ou du rechargement des données, ce qui peut s'avérer coûteux pour les tâches d'inférence par lots de longue durée. Pour atténuer ce problème, vous pouvez définir la valeur `WhenEmpty` et `consolidationPolicy` configurer une `consolidateAfter` durée, telle qu'une heure, pour conserver les nœuds pendant les pics de charge de travail. Par exemple :

```
disruption:
  consolidationPolicy: WhenEmpty
  consolidateAfter: 60m
```

Cette approche améliore la latence au démarrage des modules pour les charges de travail d'inférence par lots complexes et autres tâches sensibles aux interruptions, telles que le traitement des données d'inférence en ligne en temps réel ou la formation de modèles, où le coût de l'interruption l'emporte sur les économies de coûts de calcul. Karpenter NodePool Disruption Budgets est une autre fonctionnalité permettant de gérer les interruptions de Karpenter. Avec les budgets, vous pouvez vous assurer que pas plus d'un certain nombre de nœuds ne seront perturbés dans le nœud choisi NodePool à un moment donné. Vous pouvez également utiliser des budgets d'interruption pour éviter que tous les nœuds ne soient perturbés à un moment donné (par exemple aux heures de pointe). Pour en savoir plus, consultez la documentation de Karpenter Consolidation.

Utiliser ttlSecondsAfter Finished pour nettoyer automatiquement les tâches Kubernetes

Nous vous recommandons de configurer `ttlSecondsAfterFinished` les tâches Kubernetes dans Amazon EKS pour qu'elles suppriment automatiquement les objets de tâche terminés. Les objets de travail persistants consomment les ressources du cluster, telles que la mémoire du serveur d'API, et compliquent la surveillance en encombrant les tableaux de bord (par exemple, Grafana, Amazon). CloudWatch Par exemple, la définition d'un TTL d'une heure garantit que les tâches sont supprimées peu de temps après leur fin, ce qui permet de garder votre cluster bien rangé. Pour plus de détails, reportez-vous à la section Nettoyage automatique des tâches terminées.

Configurer la préemption des tâches de faible priorité pour les tâches et les charges de travail de priorité plus élevée

Pour les charges de AI/ML travail à priorité mixte sur Amazon EKS, vous pouvez configurer la préemption des tâches à faible priorité afin de garantir que les tâches les plus prioritaires (par exemple, l'inférence en temps réel) reçoivent des ressources rapidement. Sans préemption, les charges de travail peu prioritaires telles que les processus par lots (par exemple, inférence par lots, traitement des données), les services non batch (par exemple, tâches en arrière-plan, tâches cron) ou les tâches gourmandes en CPU et en mémoire (par exemple, les services Web) peuvent retarder les pods critiques en occupant des nœuds. La préemption permet à Kubernetes d'expulser les pods de faible priorité lorsque les pods de priorité élevée ont besoin de ressources, garantissant ainsi une allocation efficace des ressources sur les nœuds dotés de ou de mémoire. GPUs CPUs Nous vous recommandons d'utiliser Kubernetes `PriorityClass` pour attribuer des priorités et contrôler le comportement `PodDisruptionBudget` d'expulsion.

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: low-priority
value: 100
---
spec:
  priorityClassName: low-priority
```

Consultez la documentation relative à la priorité et à la préemption de Kubernetes pour plus d'informations.

Mise à l'échelle et performances des applications

Adaptez la capacité de calcul aux charges de travail ML avec Karpenter ou Static Nodes

Pour garantir une capacité de calcul rentable et réactive pour les flux de travail d'apprentissage automatique (ML) sur Amazon EKS, nous vous recommandons d'adapter votre stratégie de provisionnement des nœuds aux caractéristiques de votre charge de travail et à vos engagements en matière de coûts. Vous trouverez ci-dessous deux approches à envisager : le just-in-time dimensionnement avec Karpenter et les groupes de nœuds statiques pour la capacité réservée.

- Just-in-time scalers de plans de données tels que Karpenter : pour les flux de travail ML dynamiques soumis à des exigences de calcul variables (par exemple, inférence basée sur le GPU suivie d'un tracé basé sur le processeur), nous recommandons d'utiliser des scalers de plans de données tels que Karpenter. just-in-time
- Utilisez des groupes de nœuds statiques pour des charges de travail prévisibles : pour des charges de travail ML prévisibles et stables ou lors de l'utilisation d'instances réservées, les groupes de nœuds gérés par EKS peuvent contribuer à garantir que la capacité réservée est entièrement provisionnée et utilisée, maximisant ainsi les économies. Cette approche est idéale pour des types d'instances spécifiques commis via RIs ou ODCRs.

Exemple

Il s'agit d'un exemple de Karpenter diversifié NodePoolqui permet de lancer des instances g Amazon EC2 lorsque la génération d'instances est supérieure à trois.

```
apiVersion: karpenter.sh/v1
```

```
kind: NodePool
metadata:
  name: gpu-inference
spec:
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["on-demand"]
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["g"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["3"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
  limits:
    cpu: "1000"
    memory: "4000Gi"
    nvidia.com/gpu: "10"  *# Limit the total number of GPUs to 10 for the NodePool*
  disruption:
    consolidationPolicy: WhenEmpty
    consolidateAfter: 60m
    expireAfter: 720h
```

Exemple

Exemple d'utilisation de groupes de nœuds statiques pour une charge de travail d'entraînement :

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: ml-cluster
  region: us-west-2
```

```
managedNodeGroups:
  - name: gpu-node-group
    instanceType: p4d.24xlarge
    minSize: 2
    maxSize: 2
    desiredCapacity: 2
    taints:
      - key: nvidia.com/gpu
        effect: NoSchedule
```

Utilisez des contraintes et des tolérances pour empêcher que des charges de travail non accélérées ne soient planifiées sur des instances accélérées

La planification de charges de travail non accélérées sur les ressources GPU n'est pas efficace en termes de calcul. Nous vous recommandons d'utiliser des restrictions et des tolérances pour garantir que les pods de charges de travail non accélérées ne sont pas planifiés sur des nœuds inappropriés. Consultez la documentation de Kubernetes pour plus d'informations.

Échelle basée sur les performances du modèle

Pour les charges de travail d'inférence, nous recommandons d'utiliser Kubernetes Event-Driven Autoscaling (KEDA) pour effectuer une mise à l'échelle en fonction des indicateurs de performance du modèle, tels que les demandes d'inférence ou le débit de jetons, avec des périodes de recharge appropriées. Les politiques de dimensionnement statique peuvent surprovisionner ou sous-allouer les ressources, ce qui a un impact sur les coûts et la latence. Pour en savoir plus, consultez la documentation KEDA.

Allocation dynamique des ressources pour une gestion avancée du GPU

L'allocation dynamique des ressources (DRA) représente une avancée fondamentale dans la gestion des ressources GPU Kubernetes. Le DRA va au-delà des limites traditionnelles des plug-ins d'appareils pour permettre un partage sophistiqué du GPU, une connaissance de la topologie et une coordination des ressources entre les nœuds. Disponible dans la version 1.33 d'Amazon EKS, DRA répond aux défis critiques liés aux AI/ML charges de travail en fournissant les éléments suivants :

- Allocation précise du GPU
- Mécanismes de partage avancés, tels que le service multi-processus (MPS) et le GPU multi-instance (MIG)

- Support pour les architectures matérielles de nouvelle génération, notamment NVIDIA GB200 UltraServers

L'allocation GPU traditionnelle traite les ressources GPUs comme des entiers opaques, ce qui entraîne une sous-utilisation significative (souvent 30 à 40 % dans les clusters de production). Cela se produit parce que les charges de travail bénéficient d'un accès exclusif à l'intégralité, GPUs même lorsqu'elles ne nécessitent que des ressources fractionnaires. DRA transforme ce modèle en introduisant une allocation déclarative structurée qui fournit au planificateur Kubernetes une visibilité complète sur les caractéristiques matérielles et les exigences de charge de travail. Cela permet de prendre des décisions de placement intelligentes et de partager efficacement les ressources.

Avantages de l'utilisation de DRA au lieu du plug-in d'appareil NVIDIA

Le plug-in pour appareil NVIDIA (à partir de la version`0.12.0`) prend en charge les mécanismes de partage du GPU, notamment le time-slicing, le MPS et le MIG. Cependant, il existe des limites architecturales que le DRA corrige.

Limitations des plug-ins pour appareils NVIDIA

- Configuration statique : les configurations de partage du GPU (répliques temporelles et paramètres MPS) nécessitent une préconfiguration à l'échelle du cluster. `ConfigMaps` Il est donc difficile de proposer différentes stratégies de partage pour différentes charges de travail.
- Sélection granulaire limitée : alors que le plug-in de l'appareil expose les caractéristiques du GPU par le biais d'étiquettes de nœuds, les charges de travail ne peuvent pas demander de manière dynamique des configurations GPU spécifiques (taille de mémoire et capacités de calcul) dans le cadre de la décision de planification.
- Aucune coordination des ressources entre nœuds : impossible de gérer les ressources GPU distribuées sur plusieurs nœuds ou d'exprimer des exigences topologiques complexes, telles que les NVLink domaines pour des systèmes tels que NVIDIA. GB200
- Contraintes du planificateur : le planificateur Kubernetes traite les ressources du GPU comme des entiers opaques, ce qui limite sa capacité à prendre des décisions tenant compte de la topologie ou à gérer des dépendances de ressources complexes.
- Complexité de la configuration : la mise en place de différentes stratégies de partage nécessite un étiquetage multiple `ConfigMaps` et minutieux des nœuds, ce qui crée une complexité opérationnelle.

Solutions avec DRA

- Sélection dynamique des ressources : le DRA permet aux charges de travail de spécifier des exigences détaillées (mémoire GPU, versions des pilotes et attributs spécifiques) au `resourceclaims` moment de la demande. Cela permet une mise en correspondance des ressources plus flexible.
- Connaissance de la topologie : grâce à des paramètres structurés et à des sélecteurs de périphériques, le DRA répond à des exigences complexes telles que la communication GPU entre nœuds et les interconnexions cohérentes en mémoire.
- Gestion des ressources entre nœuds : `computeDomains` permet de coordonner les ressources GPU distribuées sur plusieurs nœuds, ce qui est essentiel pour les systèmes tels que GB200 les canaux IMEX.
- Configuration spécifique à la charge de travail : chacune `ResourceClaim` définit des stratégies et des configurations de partage différentes, ce qui permet un contrôle précis par charge de travail plutôt que des paramètres à l'échelle du cluster.
- Intégration améliorée du planificateur : DRA fournit au planificateur des informations détaillées sur les appareils et permet de prendre des décisions de placement plus intelligentes en fonction de la topologie matérielle et des caractéristiques des ressources.

Important : DRA ne remplace pas entièrement le plug-in de l'appareil NVIDIA. Le pilote NVIDIA DRA fonctionne conjointement avec le plug-in de l'appareil pour fournir des fonctionnalités améliorées. Le plug-in de l'appareil continue de gérer la découverte et la gestion de base du GPU, tandis que le DRA ajoute des fonctionnalités avancées d'allocation et de planification.

Instances prises en charge par DRA et leurs fonctionnalités

La prise en charge du DRA varie en fonction de la famille d'instances Amazon EC2 et de l'architecture GPU, comme indiqué dans le tableau suivant.

Famille d’instances	Type de GPU	Trancher le temps	Support MIG	Support MPS	Support IMEX	Cas d’utilisation
G5	NVIDIA A10 G	Oui	Non	Oui	Non	Charges de travail graphiques et d'inférence

Famille d'instances	Type de GPU	Trancher le temps	Support MIG	Support MPS	Support IMEX	Cas d'utilisation
G6	NVIDIA L4	Oui	Non	Oui	Non	Inférence basée sur l'IA et traitement vidéo
G6e	NVIDIA L40	Oui	Non	Oui	Non	Entraînement, inférence et graphisme
P4D/P4de	NVIDIA A100	Oui	Oui	Oui	Non	Formation à grande échelle et HPC
P5	NVIDIA H100	Oui	Oui	Oui	Non	Formation sur les modèles de base
P6	NVIDIA B200	Oui	Oui	Oui	Non	Modèles comportant des milliards ou des billions de paramètres, formation distribuée et inférence
P6e	NVIDIA GB200	Oui	Oui	Oui	Oui	Modèles comportant des milliards ou des billions de paramètres, formation distribuée et inférence

Les descriptions de chaque fonctionnalité du tableau sont les suivantes :

- Tranchage dans le temps : permet à plusieurs charges de travail de partager les ressources de calcul du GPU au fil du temps.
- GPU multi-instance (MIG) : partitionnement au niveau du matériel qui crée des instances de GPU isolées.
- Service multiprocessus (MPS) : permet l'exécution simultanée de plusieurs processus CUDA sur un seul GPU.

- Internode Memory Exchange (IMEX) : communication cohérente en mémoire entre les nœuds pour. GB200 UltraServers

Ressources supplémentaires

Pour plus d'informations sur les pilotes Kubernetes DRA et NVIDIA DRA, consultez les ressources suivantes sur : GitHub

- Kubernetes dynamic-resource-allocation
- Proposition d'amélioration de Kubernetes pour DRA
- Pilote NVIDIA DRA pour GPUs
- Exemples de NVIDIA DRA et démarrage rapide

Configurez l'allocation dynamique des ressources pour une gestion avancée du GPU

La rubrique suivante explique comment configurer l'allocation dynamique des ressources (DRA) pour une gestion avancée du GPU.

Conditions préalables

Avant d'implémenter le DRA sur Amazon EKS, assurez-vous que votre environnement répond aux exigences suivantes.

Configuration du cluster

- Cluster Amazon EKS en cours d'exécution (version `1.33` ou ultérieure)
- Groupes de nœuds gérés par Amazon EKS (le DRA n'est actuellement pris en charge que par les groupes de nœuds gérés optimisés pour Bottlerocket NVIDIA AMIs, AL2023 et non par Karpenter)
- Nœuds de travail compatibles avec le GPU NVIDIA dotés des types d'instances appropriés

Composants requis

- Version du plugin pour appareil NVIDIA `0.17.1` ou version ultérieure
- Version du pilote NVIDIA DRA `25.3.0` ou ultérieure

Étape 1 : créer un cluster avec un groupe de nœuds compatible DRA à l'aide d'eksctl

1. Créez un fichier de configuration de cluster nommé `dra-eks-cluster.yaml` :

```
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: dra-eks-cluster
  region: us-west-2
  version: '1.33'

managedNodeGroups:
- name: gpu-dra-nodes
  amiFamily: AmazonLinux2023
  instanceType: g6.12xlarge
  desiredCapacity: 2
  minSize: 1
  maxSize: 3

  labels:
    node-type: "gpu-dra"
    nvidia.com/gpu.present: "true"

  taints:
  - key: nvidia.com/gpu
    value: "true"
    effect: NoSchedule
```

2. Créez le cluster :

```
eksctl create cluster -f dra-eks-cluster.yaml
```

Étape 2 : Déployer le plug-in pour appareil NVIDIA

Déployez le plug-in pour appareil NVIDIA pour activer la découverte de base du GPU :

1. Ajoutez le référentiel Helm du plugin pour appareils NVIDIA :

```
helm repo add nvidia https://nvidia.github.io/k8s-device-plugin
```

```
helm repo update
```

2. Créez des valeurs personnalisées pour le plug-in de l'appareil :

```
cat <<EOF > nvidia-device-plugin-values.yaml
gfd:
  enabled: true
nfd:
  enabled: true
tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
EOF
```

3. Installez le plug-in pour appareil NVIDIA :

```
helm install nvidia-device-plugin nvidia/nvidia-device-plugin \
 --namespace nvidia-device-plugin \
 --create-namespace \
 --version v0.17.1 \
 --values nvidia-device-plugin-values.yaml
```

Étape 3 : Déploiement du pilote NVIDIA DRA Helm (graphique Helm)

1. Créez un fichier de `dra-driver-values.yaml` valeurs pour le pilote DRA :

```
---
nvidiaDriverRoot: /

gpuResourcesEnabledOverride: true

resources:
  gpus:
    enabled: true
  computeDomains:
    enabled: true  # Enable for GB200 IMEX support

controller:
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
```

```
    effect: NoSchedule

kubeletPlugin:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: "nvidia.com/gpu.present"
            operator: In
            values: ["true"]
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
```

2. Ajoutez le référentiel NVIDIA NGC Helm :

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
```

3. Installez le pilote NVIDIA DRA :

```
helm install nvidia-dra-driver nvidia/nvidia-dra-driver-gpu \
 --version="25.3.0-rc.2" \
 --namespace nvidia-dra-driver \
 --create-namespace \
 --values dra-driver-values.yaml
```

Étape 4 : vérifier l'installation du DRA

1. Vérifiez que les ressources de l'API DRA sont disponibles :

```
kubectl api-resources | grep resource.k8s.io/v1beta1
```

Le résultat attendu est le suivant :

```
deviceclasses resource.k8s.io/v1beta1 false DeviceClass
resourceclaims resource.k8s.io/v1beta1 true ResourceClaim
resourceclaimtemplates resource.k8s.io/v1beta1 true ResourceClaimTemplate
resourceslices resource.k8s.io/v1beta1 false ResourceSlice
```

2. Vérifiez les classes d'appareils disponibles :

```
kubectl get deviceclasses
```

Voici un exemple de résultat attendu :

```
NAME                                        AGE
compute-domain-daemon.nvidia.com            4h39m
compute-domain-default-channel.nvidia.com   4h39m
gpu.nvidia.com                              4h39m
mig.nvidia.com                              4h39m
```

Lorsqu'une instance de GPU G6 nouvellement créée rejoint votre cluster Amazon EKS avec le DRA activé, les actions suivantes se produisent :

- Le pilote NVIDIA DRA détecte automatiquement le GPU A10G et en crée deux `resourceslices` sur ce nœud.
- La `gpu.nvidia.com` tranche enregistre le périphérique GPU A10G physique avec ses spécifications (mémoire, capacité de calcul, etc.).
- Étant donné que l'A10G ne prend pas en charge le partitionnement MIG, la `compute-domain.nvidia.com` tranche crée un domaine de calcul unique représentant l'ensemble du contexte de calcul du GPU.
- Ils `resourceslices` sont ensuite publiés sur le serveur d'API Kubernetes, ce qui rend les ressources GPU disponibles pour la planification. `resourceclaims`

  Le planificateur DRA peut désormais allouer intelligemment ce GPU aux pods qui demandent des ressources GPU`resourceclaimtemplates`, offrant ainsi une gestion des ressources plus flexible par rapport aux approches traditionnelles de plug-in d'appareil. Cela se fait automatiquement sans intervention manuelle. Le nœud devient simplement disponible pour les charges de travail du GPU une fois que le pilote DRA a terminé le processus de découverte et d'enregistrement des ressources.

  Lorsque vous exécutez la commande suivante :

  ```
  kubectl get resourceslices
  ```

  Voici un exemple de résultat attendu :

```
NAME                                                                NODE
            DRIVER                        POOL                                AGE
ip-100-64-129-47.ec2.internal-compute-domain.nvidia.com-rwsts
 ip-100-64-129-47.ec2.internal    compute-domain.nvidia.com
 ip-100-64-129-47.ec2.internal    35m
ip-100-64-129-47.ec2.internal-gpu.nvidia.com-6kndg
 ip-100-64-129-47.ec2.internal    gpu.nvidia.com
 ip-100-64-129-47.ec2.internal    35m
```

Passez au the section called “Planifiez une charge de travail GPU simple à l'aide de l'allocation dynamique des ressources”.

Planifiez une charge de travail GPU simple à l'aide de l'allocation dynamique des ressources

Pour planifier une charge de travail GPU simple à l'aide de l'allocation dynamique des ressources (DRA), procédez comme suit. Avant de poursuivre, assurez-vous d'avoir suivithe section called “Configurez l'allocation dynamique des ressources pour une gestion avancée du GPU”.

1. Créez un fichier de base `ResourceClaimTemplate` pour l'allocation du GPU avec un fichier nommé `basic-gpu-claim-template.yaml` :

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: gpu-test1

---
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  namespace: gpu-test1
  name: single-gpu
spec:
  spec:
    devices:
      requests:
      - name: gpu
```

```
        deviceClassName: gpu.nvidia.com
```

2. Appliquez le modèle :

```
kubectl apply -f basic-gpu-claim-template.yaml
```

3. Vérifiez le statut :

```
kubectl get resourceclaimtemplates -n gpu-test1
```

   Voici un exemple de sortie :

```
NAME         AGE
single-gpu   9m16s
```

4. Créez un Pod qui utilise le `ResourceClaimTemplate` avec un fichier nommé `basic-gpu-pod.yaml` :

```
---
apiVersion: v1
kind: Pod
metadata:
  namespace: gpu-test1
  name: gpu-pod
  labels:
    app: pod
spec:
  containers:
  - name: ctr0
    image: ubuntu:22.04
    command: ["bash", "-c"]
    args: ["nvidia-smi -L; trap 'exit 0' TERM; sleep 9999 & wait"]
    resources:
      claims:
      - name: gpu0
  resourceClaims:
  - name: gpu0
    resourceClaimTemplateName: single-gpu
  nodeSelector:
    NodeGroupType: gpu-dra
    nvidia.com/gpu.present: "true"
  tolerations:
```

```
  - key: "nvidia.com/gpu"
    operator: "Exists"
    effect: "NoSchedule"
```

5. Appliquez et surveillez le Pod :

```
kubectl apply -f basic-gpu-pod.yaml
```

6. Vérifiez l'état du Pod :

```
kubectl get pod -n gpu-test1
```

Voici un exemple de sortie attendue :

```
NAME      READY   STATUS    RESTARTS   AGE
gpu-pod   1/1     Running   0          13m
```

7. Vérifiez le `ResourceClaim` statut :

```
kubectl get resourceclaims -n gpu-test1
```

Voici un exemple de sortie attendue :

```
NAME                 STATE                AGE
gpu-pod-gpu0-l76cg   allocated,reserved   9m6s
```

8. Consultez les journaux du pod pour consulter les informations du processeur graphique :

```
kubectl logs gpu-pod -n gpu-test1
```

Voici un exemple de sortie attendue :

```
GPU 0: NVIDIA L4 (UUID: GPU-da7c24d7-c7e3-ed3b-418c-bcecc32af7c5)
```

Continuez vers the section called “Techniques d'optimisation du GPU avec allocation dynamique des ressources” pour découvrir des techniques d'optimisation du GPU plus avancées à l'aide de la DRA.

Techniques d'optimisation du GPU avec allocation dynamique des ressources

Les charges de travail des GPU modernes nécessitent une gestion sophistiquée des ressources pour optimiser l'utilisation et la rentabilité. Le DRA permet plusieurs techniques d'optimisation avancées qui répondent à différents cas d'utilisation et à différentes capacités matérielles :

- Le découpage dans le temps permet à plusieurs charges de travail de partager les ressources de calcul du GPU au fil du temps, ce qui le rend idéal pour les charges de travail d'inférence associées à une utilisation sporadique du GPU. Pour obtenir un exemple, consultez the section called “Optimisez les charges de travail du GPU grâce au découpage en tranches de temps”.
- Le service multiprocessus (MPS) permet l'exécution simultanée de plusieurs processus CUDA sur un seul GPU avec une meilleure isolation que le découpage en tranches temporelles. Pour obtenir un exemple, consultez the section called “Optimisez les charges de travail du GPU avec MPS”.
- Le GPU multi-instance (MIG) assure le partitionnement au niveau du matériel, en créant des instances de GPU isolées dotées de ressources de calcul et de mémoire dédiées. Pour obtenir un exemple, consultez the section called “Optimisez les charges de travail du GPU avec le GPU multi-instance”.
- L'échange de mémoire entre nœuds (IMEX) permet une communication cohérente en mémoire entre les nœuds pour un entraînement distribué sur les systèmes NVIDIA. GB200 Pour obtenir un exemple, consultez the section called “Optimisez les charges de travail du GPU avec IMEX à l'aide d'instances P6e GB200”.

Ces techniques peuvent améliorer considérablement l'utilisation des ressources. Organisations signalent que l'utilisation du GPU augmente de 30 à 40 % avec l'allocation traditionnelle à 80 à 90 % avec des stratégies de partage optimisées. Le choix de la technique dépend des caractéristiques de la charge de travail, des exigences d'isolation et des capacités matérielles.

Optimisez les charges de travail du GPU grâce au découpage en tranches de temps

Le time-slicing permet à plusieurs charges de travail de partager les ressources de calcul du GPU en les planifiant pour qu'elles s'exécutent de manière séquentielle sur le même GPU physique. Il est idéal pour les charges de travail d'inférence associées à une utilisation sporadique du GPU.

Procédez comme suit.

1. Définissez un `ResourceClaimTemplate` pour le découpage temporel avec un fichier nommé : `timeslicing-claim-template.yaml`

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: timeslicing-gpu

---
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  name: timeslicing-gpu-template
  namespace: timeslicing-gpu
spec:
  spec:
    devices:
      requests:
      - name: shared-gpu
        deviceClassName: gpu.nvidia.com
      config:
      - requests: ["shared-gpu"]
        opaque:
          driver: gpu.nvidia.com
          parameters:
            apiVersion: resource.nvidia.com/v1beta1
            kind: GpuConfig
            sharing:
              strategy: TimeSlicing
```

2. Définissez un pod en utilisant le time-slicing avec un fichier nommé : `timeslicing-pod.yaml`

```
---
# Pod 1 - Inference workload
apiVersion: v1
kind: Pod
metadata:
  name: inference-pod-1
  namespace: timeslicing-gpu
  labels:
    app: gpu-inference
spec:
  restartPolicy: Never
  containers:
```

```
  - name: inference-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "-c"]
    args:
    - |
      import torch
      import time
      import os
      print(f"=== POD 1 STARTING ===")
      print(f"GPU available: {torch.cuda.is_available()}")
      print(f"GPU count: {torch.cuda.device_count()}")
      if torch.cuda.is_available():
          device = torch.cuda.current_device()
          print(f"Current GPU: {torch.cuda.get_device_name(device)}")
          print(f"GPU Memory:
 {torch.cuda.get_device_properties(device).total_memory / 1024**3:.1f} GB")
          # Simulate inference workload
          for i in range(20):
              x = torch.randn(1000, 1000).cuda()
              y = torch.mm(x, x.t())
              print(f"Pod 1 - Iteration {i+1} completed at {time.strftime('%H:%M:
%S')}")
              time.sleep(60)
      else:
          print("No GPU available!")
          time.sleep(5)
    resources:
      claims:
      - name: shared-gpu-claim
  resourceClaims:
  - name: shared-gpu-claim
    resourceClaimTemplateName: timeslicing-gpu-template
  nodeSelector:
    NodeGroupType: "gpu-dra"
    nvidia.com/gpu.present: "true"
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule


---
# Pod 2 - Training workload
apiVersion: v1
```

```
kind: Pod
metadata:
  name: training-pod-2
  namespace: timeslicing-gpu
  labels:
    app: gpu-training
spec:
  restartPolicy: Never
  containers:
  - name: training-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "-c"]
    args:
    - |
      import torch
      import time
      import os
      print(f"=== POD 2 STARTING ===")
      print(f"GPU available: {torch.cuda.is_available()}")
      print(f"GPU count: {torch.cuda.device_count()}")
      if torch.cuda.is_available():
          device = torch.cuda.current_device()
          print(f"Current GPU: {torch.cuda.get_device_name(device)}")
          print(f"GPU Memory:
 {torch.cuda.get_device_properties(device).total_memory / 1024**3:.1f} GB")
          # Simulate training workload with heavier compute
          for i in range(15):
              x = torch.randn(2000, 2000).cuda()
              y = torch.mm(x, x.t())
              loss = torch.sum(y)
              print(f"Pod 2 - Training step {i+1}, Loss: {loss.item():.2f} at
 {time.strftime('%H:%M:%S')}")
              time.sleep(5)
      else:
          print("No GPU available!")
          time.sleep(60)
    resources:
      claims:
      - name: shared-gpu-claim-2
  resourceClaims:
  - name: shared-gpu-claim-2
    resourceClaimTemplateName: timeslicing-gpu-template
  nodeSelector:
    NodeGroupType: "gpu-dra"
```

```
    nvidia.com/gpu.present: "true"
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
```

3. Appliquez le modèle et le pod :

```
kubectl apply -f timeslicing-claim-template.yaml
kubectl apply -f timeslicing-pod.yaml
```

4. Surveillez les demandes de ressources :

```
kubectl get resourceclaims -n timeslicing-gpu -w
```

Voici un exemple de sortie :

```
NAME                                        STATE                AGE
inference-pod-1-shared-gpu-claim-9p97x      allocated,reserved   21s
training-pod-2-shared-gpu-claim-2-qghnb     pending              21s
inference-pod-1-shared-gpu-claim-9p97x      pending              105s
training-pod-2-shared-gpu-claim-2-qghnb     pending              105s
inference-pod-1-shared-gpu-claim-9p97x      pending              105s
training-pod-2-shared-gpu-claim-2-qghnb     allocated,reserved   105s
inference-pod-1-shared-gpu-claim-9p97x      pending              105s
```

Premier pod (`inference-pod-1`)

- État : `allocated,reserved`
- Signification : DRA a trouvé un GPU disponible et l'a réservé pour ce Pod
- État du pod : démarre immédiatement

Deuxième module (`training-pod-2`)

- État : `pending`
- Signification : attendre que DRA configure le time-slicing sur le même GPU
- État du pod : En attente de planification
- L'État va passer de `pending` `allocated,reserved` à `running`

Optimisez les charges de travail du GPU avec MPS

Le service multiprocessus (MPS) permet l'exécution simultanée de plusieurs contextes CUDA sur un seul GPU avec une meilleure isolation que le découpage en tranches temporelles.

Procédez comme suit.

1. Définissez un `ResourceClaimTemplate` pour MPS avec un fichier nommé `mps-claim-template.yaml` :

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: mps-gpu

---
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  name: mps-gpu-template
  namespace: mps-gpu
spec:
  spec:
    devices:
      requests:
      - name: shared-gpu
        deviceClassName: gpu.nvidia.com
      config:
      - requests: ["shared-gpu"]
        opaque:
          driver: gpu.nvidia.com
          parameters:
            apiVersion: resource.nvidia.com/v1beta1
            kind: GpuConfig
            sharing:
              strategy: MPS
```

2. Définissez un Pod à l'aide de MPS avec un fichier nommé `mps-pod.yaml` :

```
---
# Single Pod with Multiple Containers sharing GPU via MPS
apiVersion: v1
```

```
kind: Pod
metadata:
  name: mps-multi-container-pod
  namespace: mps-gpu
  labels:
    app: mps-demo
spec:
  restartPolicy: Never
  containers:
  # Container 1 - Inference workload
  - name: inference-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "-c"]
    args:
    - |
      import torch
      import torch.nn as nn
      import time
      import os

      print(f"=== INFERENCE CONTAINER STARTING ===")
      print(f"Process ID: {os.getpid()}")
      print(f"GPU available: {torch.cuda.is_available()}")
      print(f"GPU count: {torch.cuda.device_count()}")

      if torch.cuda.is_available():
          device = torch.cuda.current_device()
          print(f"Current GPU: {torch.cuda.get_device_name(device)}")
          print(f"GPU Memory:
 {torch.cuda.get_device_properties(device).total_memory / 1024**3:.1f} GB")

          # Create inference model
          model = nn.Sequential(
              nn.Linear(1000, 500),
              nn.ReLU(),
              nn.Linear(500, 100)
          ).cuda()

          # Run inference
          for i in range(1, 999999):
              with torch.no_grad():
                  x = torch.randn(128, 1000).cuda()
                  output = model(x)
                  result = torch.sum(output)
```

```
                 print(f"Inference Container PID {os.getpid()}: Batch {i}, Result:
{result.item():.2f} at {time.strftime('%H:%M:%S')}")
             time.sleep(2)
     else:
         print("No GPU available!")
         time.sleep(60)
   resources:
     claims:
     - name: shared-gpu-claim
       request: shared-gpu

 # Container 2 - Training workload
 - name: training-container
   image: nvcr.io/nvidia/pytorch:25.04-py3
   command: ["python", "-c"]
   args:
   - |
     import torch
     import torch.nn as nn
     import time
     import os

     print(f"=== TRAINING CONTAINER STARTING ===")
     print(f"Process ID: {os.getpid()}")
     print(f"GPU available: {torch.cuda.is_available()}")
     print(f"GPU count: {torch.cuda.device_count()}")

     if torch.cuda.is_available():
         device = torch.cuda.current_device()
         print(f"Current GPU: {torch.cuda.get_device_name(device)}")
         print(f"GPU Memory:
{torch.cuda.get_device_properties(device).total_memory / 1024**3:.1f} GB")

         # Create training model
         model = nn.Sequential(
             nn.Linear(2000, 1000),
             nn.ReLU(),
             nn.Linear(1000, 500),
             nn.ReLU(),
             nn.Linear(500, 10)
         ).cuda()

         criterion = nn.MSELoss()
         optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

```
        # Run training
        for epoch in range(1, 999999):
            x = torch.randn(64, 2000).cuda()
            target = torch.randn(64, 10).cuda()

            optimizer.zero_grad()
            output = model(x)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()

            print(f"Training Container PID {os.getpid()}: Epoch {epoch}, Loss:
{loss.item():.4f} at {time.strftime('%H:%M:%S')}")
            time.sleep(3)
    else:
        print("No GPU available!")
        time.sleep(60)
  resources:
    claims:
    - name: shared-gpu-claim
      request: shared-gpu

 resourceClaims:
 - name: shared-gpu-claim
   resourceClaimTemplateName: mps-gpu-template

 nodeSelector:
   NodeGroupType: "gpu-dra"
   nvidia.com/gpu.present: "true"
 tolerations:
 - key: nvidia.com/gpu
   operator: Exists
   effect: NoSchedule
```

3. Appliquez le modèle et créez plusieurs pods MPS :

```
kubectl apply -f mps-claim-template.yaml
kubectl apply -f mps-pod.yaml
```

4. Surveillez les demandes de ressources :

```
kubectl get resourceclaims -n mps-gpu -w
```

Voici un exemple de sortie :

```
NAME                                                STATE                AGE
mps-multi-container-pod-shared-gpu-claim-2p9kx   allocated,reserved   86s
```

Cette configuration illustre un véritable partage de GPU à l'aide du service multiprocessus NVIDIA (MPS) via l'allocation dynamique des ressources (DRA). Contrairement au time-slicing où les charges de travail utilisent à tour de rôle le GPU de manière séquentielle, le MPS permet aux deux conteneurs de s'exécuter simultanément sur le même GPU physique. L'essentiel est que le partage DRA MPS nécessite plusieurs conteneurs au sein d'un même pod, et non plusieurs pods distincts. Une fois déployé, le pilote DRA en alloue un `ResourceClaim` au Pod et configure automatiquement le MPS pour permettre aux conteneurs d'inférence et d'entraînement de s'exécuter simultanément.

Chaque conteneur dispose de son propre espace mémoire GPU isolé et de ses propres ressources de calcul, le daemon MPS coordonnant l'accès au matériel sous-jacent. Vous pouvez vérifier que cela fonctionne en procédant comme suit :

- Vérification`nvidia-smi`, qui affichera les deux conteneurs sous forme de processus M+C (`MPS + Compute`) partageant le même périphérique GPU.
- Surveillance des journaux des deux conteneurs, qui afficheront des horodatages entrelacés prouvant une exécution simultanée.

Cette approche maximise l'utilisation du GPU en permettant aux charges de travail complémentaires de partager efficacement le matériel GPU coûteux, plutôt que de le laisser sous-utilisé par un seul processus.

Récipient 1 : **`inference-container`**

```
root@mps-multi-container-pod:/workspace# nvidia-smi
Wed Jul 16 21:09:30 2025
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 570.158.01             Driver Version: 570.158.01     CUDA Version: 12.9
   |
|-----------------------------------------+------------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
 ECC |
```

```
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute
 M. |
|                                         |                        |               MIG
 M. |
|=========================================+========================
+======================|
|   0  NVIDIA L4                      On  |   00000000:35:00.0 Off |
 0 |
| N/A   48C    P0             28W /   72W |     597MiB /  23034MiB |      0%   E.
 Process |
|                                         |                        |
 N/A |
+-----------------------------------------+------------------------
+----------------------+

+-----------------------------------------------------------------------------------------
+
| Processes:
   |
|  GPU   GI   CI              PID   Type   Process name                        GPU
 Memory |
|        ID   ID                                                               Usage
   |
|
=========================================================================================|
|    0   N/A  N/A               1    M+C   python
 246MiB |
+-----------------------------------------------------------------------------------------
+
```

Conteneur 2 : **`training-container`**

```
root@mps-multi-container-pod:/workspace# nvidia-smi
Wed Jul 16 21:16:00 2025
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 570.158.01             Driver Version: 570.158.01     CUDA Version: 12.9
   |
|-----------------------------------------+------------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
 ECC |
```

```
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute
 M. |
|                                         |                        |               MIG
 M. |
|=========================================+========================
+======================|
|   0  NVIDIA L4                      On  |   00000000:35:00.0 Off |
 0 |
| N/A   51C    P0             28W /   72W |     597MiB /  23034MiB |      0%   E.
 Process |
|                                         |                        |
 N/A |
+-----------------------------------------+------------------------
+----------------------+

+-----------------------------------------------------------------------------------------
+
| Processes:
   |
|  GPU   GI   CI              PID   Type   Process name                        GPU
 Memory |
|        ID   ID                                                               Usage
   |
|
=========================================================================================|
|    0   N/A  N/A               1    M+C   python
 314MiB |
+-----------------------------------------------------------------------------------------
+
```

Optimisez les charges de travail du GPU avec le GPU multi-instance

Le GPU multi-instance (MIG) assure le partitionnement au niveau du matériel, en créant des instances de GPU isolées dotées de ressources de calcul et de mémoire dédiées.

L'utilisation du partitionnement MIG dynamique avec différents profils nécessite l'opérateur GPU NVIDIA. L'opérateur GPU NVIDIA utilise MIG Manager pour créer des profils MIG et redémarre les instances GPU telles que P4D, P4De, P5, P6, etc. pour appliquer les modifications de configuration. L'opérateur GPU inclut des fonctionnalités complètes de gestion MIG via le composant MIG Manager, qui surveille les modifications apportées aux étiquettes des nœuds et applique automatiquement la configuration MIG appropriée. Lorsqu'une modification du profil MIG est demandée, l'opérateur arrête automatiquement tous les clients GPU, applique la nouvelle géométrie de partition et redémarre

les services concernés. Ce processus nécessite le redémarrage d'un nœud pour les instances de GPU afin de garantir des transitions d'état correctes entre les différents états du GPU. C'est pourquoi l'activation `WITH0REBOOT=true` dans la configuration de MIG Manager est essentielle à la réussite des déploiements MIG.

Vous avez besoin du pilote NVIDIA DRA et de l'opérateur GPU NVIDIA pour travailler avec MIG dans Amazon EKS. Vous n'avez pas besoin de NVIDIA Device Plugin ni de DCGM Exporter en plus de cela, car ils font partie du NVIDIA GPU Operator. Comme les pilotes NVIDIA EKS AMIs sont préinstallés, nous avons désactivé le déploiement des pilotes par l'opérateur du GPU afin d'éviter les conflits et de tirer parti des pilotes optimisés déjà présents sur les instances. Le pilote NVIDIA DRA gère l'allocation dynamique des ressources pour les instances MIG, tandis que l'opérateur GPU gère le cycle de vie complet du GPU. Cela inclut la configuration MIG, la fonctionnalité du plug-in de l'appareil, la surveillance via DCGM et la découverte des fonctionnalités des nœuds. Cette approche intégrée fournit une solution complète pour la gestion des GPU d'entreprise, avec des capacités d'isolation au niveau du matériel et d'allocation dynamique des ressources.

Étape 1 : Déployer l'opérateur GPU NVIDIA

1. Ajoutez le référentiel NVIDIA GPU Operator :

```
helm repo add nvidia https://nvidia.github.io/gpu-operator
helm repo update
```

2. Créez un `gpu-operator-values.yaml` fichier :

```
driver:
  enabled: false

mig:
  strategy: mixed

migManager:
  enabled: true
  env:
    - name: WITH_REBOOT
      value: "true"
  config:
    create: true
    name: custom-mig-parted-configs
    default: "all-disabled"
    data:
```

```
      config.yaml: |-
        version: v1
        mig-configs:
          all-disabled:
            - devices: all
              mig-enabled: false

          # P4D profiles (A100 40GB)
          p4d-half-balanced:
            - devices: [0, 1, 2, 3]
              mig-enabled: true
              mig-devices:
                "1g.5gb": 2
                "2g.10gb": 1
                "3g.20gb": 1
            - devices: [4, 5, 6, 7]
              mig-enabled: false

          # P4DE profiles (A100 80GB)
          p4de-half-balanced:
            - devices: [0, 1, 2, 3]
              mig-enabled: true
              mig-devices:
                "1g.10gb": 2
                "2g.20gb": 1
                "3g.40gb": 1
            - devices: [4, 5, 6, 7]
              mig-enabled: false

devicePlugin:
  enabled: true
  config:
    name: ""
    create: false
    default: ""

toolkit:
  enabled: true

nfd:
  enabled: true

gfd:
  enabled: true
```

```
dcgmExporter:
  enabled: true
  serviceMonitor:
    enabled: true
    interval: 15s
    honorLabels: false
    additionalLabels:
      release: kube-prometheus-stack

nodeStatusExporter:
  enabled: false

operator:
  defaultRuntime: containerd
  runtimeClass: nvidia
  resources:
    limits:
      cpu: 500m
      memory: 350Mi
    requests:
      cpu: 200m
      memory: 100Mi

daemonsets:
  tolerations:
    - key: "nvidia.com/gpu"
      operator: "Exists"
      effect: "NoSchedule"
  nodeSelector:
    accelerator: nvidia
  priorityClassName: system-node-critical
```

3. Installez GPU Operator à l'aide du `gpu-operator-values.yaml` fichier :

```
helm install gpu-operator nvidia/gpu-operator \
  --namespace gpu-operator \
  --create-namespace \
  --version v25.3.1 \
  --values gpu-operator-values.yaml
```

Ce graphique Helm déploie les composants suivants et plusieurs profils MIG :

- Plug-in de périphérique (planification des ressources GPU)

- DCGM Exporter (métriques et surveillance du GPU)
- Node Feature Discovery (NFD - étiquetage du matériel)
- Découverte des fonctionnalités du GPU (GFD - étiquetage spécifique au GPU)
- MIG Manager (partitionnement GPU multi-instances)
- Container Toolkit (environnement d'exécution du conteneur GPU)
- Operator Controller (gestion du cycle de vie)

4. Vérifiez les pods de déploiement :

```
kubectl get pods -n gpu-operator
```

Voici un exemple de sortie :

```
NAME                                                              READY   STATUS
 RESTARTS        AGE
gpu-feature-discovery-27rdq                                       1/1     Running
 0               3h31m
gpu-operator-555774698d-48brn                                     1/1     Running
 0               4h8m
nvidia-container-toolkit-daemonset-sxmh9                          1/1     Running
 1 (3h32m ago)   4h1m
nvidia-cuda-validator-qb77g                                       0/1     Completed
 0               3h31m
nvidia-dcgm-exporter-cvzd7                                        1/1     Running
 0               3h31m
nvidia-device-plugin-daemonset-5ljm5                              1/1     Running
 0               3h31m
nvidia-gpu-operator-node-feature-discovery-gc-67f66fc557-q5wkt    1/1     Running
 0               4h8m
nvidia-gpu-operator-node-feature-discovery-master-5d8ffddcsl6s6   1/1     Running
 0               4h8m
nvidia-gpu-operator-node-feature-discovery-worker-6t4w7           1/1     Running
 1 (3h32m ago)   4h1m
nvidia-gpu-operator-node-feature-discovery-worker-9w7g8           1/1     Running
 0               4h8m
nvidia-gpu-operator-node-feature-discovery-worker-k5fgs           1/1     Running
 0               4h8m
nvidia-mig-manager-zvf54                                          1/1     Running
 1 (3h32m ago)   3h35m
```

5. Créez un cluster Amazon EKS avec un groupe de nœuds gérés par P4de pour tester les exemples MIG :

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: dra-eks-cluster
  region: us-east-1
  version: '1.33'

managedNodeGroups:
# P4DE MIG Node Group with Capacity Block Reservation
- name: p4de-mig-nodes
  amiFamily: AmazonLinux2023
  instanceType: p4de.24xlarge

  # Capacity settings
  desiredCapacity: 0
  minSize: 0
  maxSize: 1

  # Use specific subnet in us-east-1b for capacity reservation
  subnets:
    - us-east-1b

  # AL2023 NodeConfig for RAID0 local storage only
  nodeadmConfig:
    apiVersion: node.eks.aws/v1alpha1
    kind: NodeConfig
    spec:
      instance:
        localStorage:
          strategy: RAID0

  # Node labels for MIG configuration
  labels:
    nvidia.com/gpu.present: "true"
    nvidia.com/gpu.product: "A100-SXM4-80GB"
    nvidia.com/mig.config: "p4de-half-balanced"
    node-type: "p4de"
    vpc.amazonaws.com/efa.present: "true"
    accelerator: "nvidia"
```

```
  # Node taints
  taints:
    - key: nvidia.com/gpu
      value: "true"
      effect: NoSchedule

  # EFA support
  efaEnabled: true

  # Placement group for high-performance networking
  placementGroup:
    groupName: p4de-placement-group
    strategy: cluster

  # Capacity Block Reservation (CBR)
  # Ensure CBR ID matches the subnet AZ with the Nodegroup subnet
  spot: false
  capacityReservation:
    capacityReservationTarget:
      capacityReservationId: "cr-abcdefghij"  # Replace with your capacity
 reservation ID
```

NVIDIA GPU Operator utilise l'étiquette ajoutée aux nœuds `nvidia.com/mig.config: "p4de-half-balanced"` et partitionne le GPU avec le profil donné.

6. Connectez-vous à l'`p4de`instance.
7. Exécutez la commande suivante :

```
nvidia-smi -L
```

Vous devriez voir l'exemple de sortie suivant :

```
[root@ip-100-64-173-145 bin]# nvidia-smi -L
GPU 0: NVIDIA A100-SXM4-80GB (UUID: GPU-ab52e33c-be48-38f2-119e-b62b9935925a)
  MIG 3g.40gb     Device  0: (UUID: MIG-da972af8-a20a-5f51-849f-bc0439f7970e)
  MIG 2g.20gb     Device  1: (UUID: MIG-7f9768b7-11a6-5de9-a8aa-e9c424400da4)
  MIG 1g.10gb     Device  2: (UUID: MIG-498adad6-6cf7-53af-9d1a-10cfd1fa53b2)
  MIG 1g.10gb     Device  3: (UUID: MIG-3f55ef65-1991-571a-ac50-0dbf50d80c5a)
GPU 1: NVIDIA A100-SXM4-80GB (UUID: GPU-0eabeccc-7498-c282-0ac7-d3c09f6af0c8)
  MIG 3g.40gb     Device  0: (UUID: MIG-80543849-ea3b-595b-b162-847568fe6e0e)
  MIG 2g.20gb     Device  1: (UUID: MIG-3af1958f-fac4-59f1-8477-9f8d08c55029)
```

```
  MIG 1g.10gb     Device  2: (UUID: MIG-401088d2-716f-527b-a970-b1fc7a4ac6b2)
  MIG 1g.10gb     Device  3: (UUID: MIG-8c56c75e-5141-501c-8f43-8cf22f422569)
GPU 2: NVIDIA A100-SXM4-80GB (UUID: GPU-1c7a1289-243f-7872-a35c-1d2d8af22dd0)
  MIG 3g.40gb     Device  0: (UUID: MIG-e9b44486-09fc-591a-b904-0d378caf2276)
  MIG 2g.20gb     Device  1: (UUID: MIG-ded93941-9f64-56a3-a9b1-a129c6edf6e4)
  MIG 1g.10gb     Device  2: (UUID: MIG-6c317d83-a078-5c25-9fa3-c8308b379aa1)
  MIG 1g.10gb     Device  3: (UUID: MIG-2b070d39-d4e9-5b11-bda6-e903372e3d08)
GPU 3: NVIDIA A100-SXM4-80GB (UUID: GPU-9a6250e2-5c59-10b7-2da8-b61d8a937233)
  MIG 3g.40gb     Device  0: (UUID: MIG-20e3cd87-7a57-5f1b-82e7-97b14ab1a5aa)
  MIG 2g.20gb     Device  1: (UUID: MIG-04430354-1575-5b42-95f4-bda6901f1ace)
  MIG 1g.10gb     Device  2: (UUID: MIG-d62ec8b6-e097-5e99-a60c-abf8eb906f91)
  MIG 1g.10gb     Device  3: (UUID: MIG-fce20069-2baa-5dd4-988a-cead08348ada)
GPU 4: NVIDIA A100-SXM4-80GB (UUID: GPU-5d09daf0-c2eb-75fd-3919-7ad8fafa5f86)
GPU 5: NVIDIA A100-SXM4-80GB (UUID: GPU-99194e04-ab2a-b519-4793-81cb2e8e9179)
GPU 6: NVIDIA A100-SXM4-80GB (UUID: GPU-c1a1910f-465a-e16f-5af1-c6aafe499cd6)
GPU 7: NVIDIA A100-SXM4-80GB (UUID: GPU-c2cfafbc-fd6e-2679-e955-2a9e09377f78)
```

NVIDIA GPU Operator a correctement appliqué le profil `p4de-half-balanced` MIG à votre instance P4DE, créant ainsi des partitions GPU au niveau du matériel telles que configurées. Voici comment fonctionne le partitionnement :

L'opérateur du GPU a appliqué cette configuration à partir de votre profil MIG intégré :

```
p4de-half-balanced:
  - devices: [0, 1, 2, 3]        # First 4 GPUs: MIG enabled
    mig-enabled: true
    mig-devices:
      "1g.10gb": 2               # 2x small instances (10GB each)
      "2g.20gb": 1               # 1x medium instance (20GB)
      "3g.40gb": 1               # 1x large instance (40GB)
  - devices: [4, 5, 6, 7]        # Last 4 GPUs: Full GPUs
    mig-enabled: false
```

À partir de vos `nvidia-smi -L` résultats, voici ce que l'opérateur du GPU a créé :

- compatible MiG GPUs (0-3) : partitionné du matériel
  - GPU 0 : NVIDIA A100- SXM4 -80 Go
    - Appareil MIG 3g.40 Go 0 — Charges de travail importantes (40 Go de mémoire, 42) SMs
    - Appareil 1 MIG 2 g/20 Go — Charges de travail moyennes (20 Go de mémoire, 28) SMs
    - Appareil MIG 1 g.10 Go 2 — Petites charges de travail (10 Go de mémoire, 14) SMs

    - Appareil MIG 1 g.10 Go 3 — Petites charges de travail (10 Go de mémoire, 14) SMs
  - GPU 1 : NVIDIA A100- -80 GO SXM4
    - Dispositif MIG 3g.40gb 0 — Disposition de partition identique
    - Appareil MIG 2g.20gb 1
    - Appareil MIG 1g.10gb 2
    - Appareil MIG 1g.10gb 3
  - GPU 2 et GPU 3 : même schéma que le GPU 0 et le GPU 1
- Complet GPUs (4-7) : pas de partitionnement MIG
  - GPU 4 : NVIDIA A100- SXM4 -80 Go — GPU complet de 80 Go
  - GPU 5 : NVIDIA A100- SXM4 -80 Go — GPU complet de 80 Go
  - GPU 6 : NVIDIA A100- SXM4 -80 Go — GPU complet de 80 Go
  - GPU 7 : NVIDIA A100- SXM4 -80 Go — GPU complet de 80 Go

Une fois que l'opérateur GPU NVIDIA a créé les partitions MIG, le pilote NVIDIA DRA détecte automatiquement ces instances isolées du matériel et les rend disponibles pour une allocation dynamique des ressources dans Kubernetes. Le pilote DRA découvre chaque instance MIG avec son profil spécifique (1g.10 Go, 2g.20 Go, 3g.40 Go) et les expose en tant que ressources planifiables via la classe de périphérique. `mig.nvidia.com`

Le pilote DRA surveille en permanence la topologie MIG et tient à jour un inventaire des instances disponibles dans l'ensemble. GPUs Lorsqu'un Pod demande un profil MIG spécifique via un`ResourceClaimTemplate`, le pilote DRA sélectionne intelligemment une instance MIG appropriée parmi n'importe quel GPU disponible, permettant ainsi une véritable mutualisation au niveau du matériel. Cette allocation dynamique permet à plusieurs charges de travail isolées de s'exécuter simultanément sur le même GPU physique tout en maintenant des limites de ressources strictes et des garanties de performances.

Étape 2 : Tester l'allocation des ressources MIG

Passons maintenant à quelques exemples pour montrer comment DRA alloue dynamiquement des instances MIG à différentes charges de travail. Déployez `resourceclaimtemplates` et testez les pods pour voir comment le pilote DRA répartit les charges de travail entre les partitions MIG disponibles, permettant ainsi à plusieurs conteneurs de partager les ressources du GPU avec une isolation au niveau du matériel.

1. Créez `mig-claim-template.yaml` pour contenir le MIG `resourceclaimtemplates` :

```
apiVersion: v1
kind: Namespace
metadata:
  name: mig-gpu

---
# Template for 3g.40gb MIG instance (Large training)
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  name: mig-large-template
  namespace: mig-gpu
spec:
  spec:
    devices:
      requests:
      - name: mig-large
        deviceClassName: mig.nvidia.com
        selectors:
        - cel:
            expression: |
              device.attributes['gpu.nvidia.com'].profile == '3g.40gb'

---
# Template for 2g.20gb MIG instance (Medium training)
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  name: mig-medium-template
  namespace: mig-gpu
spec:
  spec:
    devices:
      requests:
      - name: mig-medium
        deviceClassName: mig.nvidia.com
        selectors:
        - cel:
            expression: |
              device.attributes['gpu.nvidia.com'].profile == '2g.20gb'
```

```
---
# Template for 1g.10gb MIG instance (Small inference)
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  name: mig-small-template
  namespace: mig-gpu
spec:
  spec:
    devices:
      requests:
      - name: mig-small
        deviceClassName: mig.nvidia.com
        selectors:
        - cel:
            expression: |
              device.attributes['gpu.nvidia.com'].profile == '1g.10gb'
```

2. Appliquez les trois modèles :

```
kubectl apply -f mig-claim-template.yaml
```

3. Exécutez la commande suivante :

```
kubectl get resourceclaimtemplates -n mig-gpu
```

   Voici un exemple de sortie :

```
NAME                  AGE
mig-large-template    71m
mig-medium-template   71m
mig-small-template    71m
```

4. Créez `mig-pod.yaml` pour planifier plusieurs tâches afin d'en tirer parti `resourceclaimtemplates` :

```
---
# ConfigMap containing Python scripts for MIG pods
apiVersion: v1
kind: ConfigMap
metadata:
  name: mig-scripts-configmap
```

```
  namespace: mig-gpu
data:
  large-training-script.py: |
    import torch
    import torch.nn as nn
    import torch.optim as optim
    import time
    import os

    print(f"=== LARGE TRAINING POD (3g.40gb) ===")
    print(f"Process ID: {os.getpid()}")
    print(f"GPU available: {torch.cuda.is_available()}")
    print(f"GPU count: {torch.cuda.device_count()}")

    if torch.cuda.is_available():
        device = torch.cuda.current_device()
        print(f"Using GPU: {torch.cuda.get_device_name(device)}")
        print(f"GPU Memory: {torch.cuda.get_device_properties(device).total_memory /
 1e9:.1f} GB")

        # Large model for 3g.40gb instance
        model = nn.Sequential(
            nn.Linear(2048, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 10)
        ).cuda()

        optimizer = optim.Adam(model.parameters())
        criterion = nn.CrossEntropyLoss()

        print(f"Model parameters: {sum(p.numel() for p in model.parameters())}")

        # Training loop
        for epoch in range(100):
            # Large batch for 3g.40gb
            x = torch.randn(256, 2048).cuda()
            y = torch.randint(0, 10, (256,)).cuda()

            optimizer.zero_grad()
            output = model(x)
```

```
            loss = criterion(output, y)
            loss.backward()
            optimizer.step()

            if epoch % 10 == 0:
                print(f"Large Training - Epoch {epoch}, Loss: {loss.item():.4f}, GPU
Memory: {torch.cuda.memory_allocated()/1e9:.2f}GB")
            time.sleep(3)

        print("Large training completed on 3g.40gb MIG instance")

 medium-training-script.py: |
   import torch
   import torch.nn as nn
   import torch.optim as optim
   import time
   import os

   print(f"=== MEDIUM TRAINING POD (2g.20gb) ===")
   print(f"Process ID: {os.getpid()}")
   print(f"GPU available: {torch.cuda.is_available()}")
   print(f"GPU count: {torch.cuda.device_count()}")

   if torch.cuda.is_available():
       device = torch.cuda.current_device()
       print(f"Using GPU: {torch.cuda.get_device_name(device)}")
       print(f"GPU Memory: {torch.cuda.get_device_properties(device).total_memory /
1e9:.1f} GB")

       # Medium model for 2g.20gb instance
       model = nn.Sequential(
           nn.Linear(1024, 512),
           nn.ReLU(),
           nn.Linear(512, 256),
           nn.ReLU(),
           nn.Linear(256, 10)
       ).cuda()

       optimizer = optim.Adam(model.parameters())
       criterion = nn.CrossEntropyLoss()

       print(f"Model parameters: {sum(p.numel() for p in model.parameters())}")

       # Training loop
```

```
        for epoch in range(100):
            # Medium batch for 2g.20gb
            x = torch.randn(128, 1024).cuda()
            y = torch.randint(0, 10, (128,)).cuda()

            optimizer.zero_grad()
            output = model(x)
            loss = criterion(output, y)
            loss.backward()
            optimizer.step()

            if epoch % 10 == 0:
                print(f"Medium Training - Epoch {epoch}, Loss: {loss.item():.4f}, GPU
Memory: {torch.cuda.memory_allocated()/1e9:.2f}GB")
            time.sleep(4)

        print("Medium training completed on 2g.20gb MIG instance")

 small-inference-script.py: |
   import torch
   import torch.nn as nn
   import time
   import os

   print(f"=== SMALL INFERENCE POD (1g.10gb) ===")
   print(f"Process ID: {os.getpid()}")
   print(f"GPU available: {torch.cuda.is_available()}")
   print(f"GPU count: {torch.cuda.device_count()}")

   if torch.cuda.is_available():
       device = torch.cuda.current_device()
       print(f"Using GPU: {torch.cuda.get_device_name(device)}")
       print(f"GPU Memory: {torch.cuda.get_device_properties(device).total_memory /
1e9:.1f} GB")

       # Small model for 1g.10gb instance
       model = nn.Sequential(
           nn.Linear(512, 256),
           nn.ReLU(),
           nn.Linear(256, 10)
       ).cuda()

       print(f"Model parameters: {sum(p.numel() for p in model.parameters())}")
```

```
        # Inference loop
        for i in range(200):
            with torch.no_grad():
                # Small batch for 1g.10gb
                x = torch.randn(32, 512).cuda()
                output = model(x)
                prediction = torch.argmax(output, dim=1)

                if i % 20 == 0:
                    print(f"Small Inference - Batch {i}, Predictions:
 {prediction[:5].tolist()}, GPU Memory: {torch.cuda.memory_allocated()/1e9:.2f}GB")
            time.sleep(2)

        print("Small inference completed on 1g.10gb MIG instance")

---
# Pod 1: Large training workload (3g.40gb)
apiVersion: v1
kind: Pod
metadata:
  name: mig-large-training-pod
  namespace: mig-gpu
  labels:
    app: mig-large-training
    workload-type: training
spec:
  restartPolicy: Never
  containers:
  - name: large-training-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "/scripts/large-training-script.py"]
    volumeMounts:
    - name: script-volume
      mountPath: /scripts
      readOnly: true
    resources:
      claims:
      - name: mig-large-claim
  resourceClaims:
  - name: mig-large-claim
    resourceClaimTemplateName: mig-large-template
  nodeSelector:
    node.kubernetes.io/instance-type: p4de.24xlarge
    nvidia.com/gpu.present: "true"
```

```
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
  volumes:
  - name: script-volume
    configMap:
      name: mig-scripts-configmap
      defaultMode: 0755

---
# Pod 2: Medium training workload (2g.20gb) - can run on SAME GPU as Pod 1
apiVersion: v1
kind: Pod
metadata:
  name: mig-medium-training-pod
  namespace: mig-gpu
  labels:
    app: mig-medium-training
    workload-type: training
spec:
  restartPolicy: Never
  containers:
  - name: medium-training-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "/scripts/medium-training-script.py"]
    volumeMounts:
    - name: script-volume
      mountPath: /scripts
      readOnly: true
    resources:
      claims:
      - name: mig-medium-claim
  resourceClaims:
  - name: mig-medium-claim
    resourceClaimTemplateName: mig-medium-template
  nodeSelector:
    node.kubernetes.io/instance-type: p4de.24xlarge
    nvidia.com/gpu.present: "true"
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
  volumes:
```

```
  - name: script-volume
    configMap:
      name: mig-scripts-configmap
      defaultMode: 0755

---
# Pod 3: Small inference workload (1g.10gb) - can run on SAME GPU as Pod 1 & 2
apiVersion: v1
kind: Pod
metadata:
  name: mig-small-inference-pod
  namespace: mig-gpu
  labels:
    app: mig-small-inference
    workload-type: inference
spec:
  restartPolicy: Never
  containers:
  - name: small-inference-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "/scripts/small-inference-script.py"]
    volumeMounts:
    - name: script-volume
      mountPath: /scripts
      readOnly: true
    resources:
      claims:
      - name: mig-small-claim
  resourceClaims:
  - name: mig-small-claim
    resourceClaimTemplateName: mig-small-template
  nodeSelector:
    node.kubernetes.io/instance-type: p4de.24xlarge
    nvidia.com/gpu.present: "true"
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
  volumes:
  - name: script-volume
    configMap:
      name: mig-scripts-configmap
      defaultMode: 0755
```

5. Appliquez cette spécification, qui devrait déployer trois pods :

```
kubctl apply -f mig-pod.yaml
```

Ces pods doivent être programmés par le pilote DRA.

6. Consultez les journaux du pilote DRA Pod et vous verrez un résultat similaire à celui-ci :

```
I0717 21:50:22.925811 1 driver.go:87] NodePrepareResource is called: number of
 claims: 1
I0717 21:50:22.932499 1 driver.go:129] Returning newly prepared devices
 for claim '933e9c72-6fd6-49c5-933c-a896407dc6d1': [&Device{RequestNames:
[mig-large],PoolName:ip-100-64-173-145.ec2.internal,DeviceName:gpu-0-
mig-9-4-4,CDIDeviceIDs:[k8s.gpu.nvidia.com/device=**gpu-0-mig-9-4-4**],}]
I0717 21:50:23.186472 1 driver.go:87] NodePrepareResource is called: number of
 claims: 1
I0717 21:50:23.191226 1 driver.go:129] Returning newly prepared devices
 for claim '61e5ddd2-8c2e-4c19-93ae-d317fecb44a4': [&Device{RequestNames:
[mig-medium],PoolName:ip-100-64-173-145.ec2.internal,DeviceName:gpu-2-
mig-14-0-2,CDIDeviceIDs:[k8s.gpu.nvidia.com/device=**gpu-2-mig-14-0-2**],}]
I0717 21:50:23.450024 1 driver.go:87] NodePrepareResource is called: number of
 claims: 1
I0717 21:50:23.455991 1 driver.go:129] Returning newly prepared devices
 for claim '1eda9b2c-2ea6-401e-96d0-90e9b3c111b5': [&Device{RequestNames:
[mig-small],PoolName:ip-100-64-173-145.ec2.internal,DeviceName:gpu-1-
mig-19-2-1,CDIDeviceIDs:[k8s.gpu.nvidia.com/device=**gpu-1-mig-19-2-1**],}]
```

7. Vérifiez le `resourceclaims` pour connaître l'état du Pod :

```
kubectl get resourceclaims -n mig-gpu -w
```

Voici un exemple de sortie :

```
NAME                                          STATE                AGE
mig-large-training-pod-mig-large-claim-6dpn8     pending              0s
mig-large-training-pod-mig-large-claim-6dpn8     pending              0s
mig-large-training-pod-mig-large-claim-6dpn8     allocated,reserved   0s
mig-medium-training-pod-mig-medium-claim-bk596   pending              0s
mig-medium-training-pod-mig-medium-claim-bk596   pending              0s
mig-medium-training-pod-mig-medium-claim-bk596   allocated,reserved   0s
mig-small-inference-pod-mig-small-claim-d2t58    pending              0s
mig-small-inference-pod-mig-small-claim-d2t58    pending              0s
```

```
mig-small-inference-pod-mig-small-claim-d2t58    allocated,reserved   0s
```

Comme vous pouvez le constater, tous les pods sont passés de « en attente » à « `allocated,reserved` par le pilote DRA ».

8. Exécutez `nvidia-smi` depuis le nœud. Vous remarquerez que trois processeurs Python sont en cours d'exécution :

```
root@ip-100-64-173-145 bin]# nvidia-smi
+-----------------------------------------------------------------------------------------------
+
| NVIDIA-SMI 570.158.01 Driver Version: 570.158.01 CUDA Version: 12.8 |
|-----------------------------------------+------------------------
+----------------------+
| GPU Name Persistence-M | Bus-Id Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf Pwr:Usage/Cap | Memory-Usage | GPU-Util Compute M. |
| | | MIG M. |
|=========================================+========================
+======================|
| 0 NVIDIA A100-SXM4-80GB On | 00000000:10:1C.0 Off | On |
| N/A 63C P0 127W / 400W | 569MiB / 81920MiB | N/A Default |
| | | Enabled |
+-----------------------------------------+------------------------
+----------------------+
| 1 NVIDIA A100-SXM4-80GB On | 00000000:10:1D.0 Off | On |
| N/A 56C P0 121W / 400W | 374MiB / 81920MiB | N/A Default |
| | | Enabled |
+-----------------------------------------+------------------------
+----------------------+
| 2 NVIDIA A100-SXM4-80GB On | 00000000:20:1C.0 Off | On |
| N/A 63C P0 128W / 400W | 467MiB / 81920MiB | N/A Default |
| | | Enabled |
+-----------------------------------------+------------------------
+----------------------+
| 3 NVIDIA A100-SXM4-80GB On | 00000000:20:1D.0 Off | On |
| N/A 57C P0 118W / 400W | 249MiB / 81920MiB | N/A Default |
| | | Enabled |
+-----------------------------------------+------------------------
+----------------------+
| 4 NVIDIA A100-SXM4-80GB On | 00000000:90:1C.0 Off | 0 |
| N/A 51C P0 77W / 400W | 0MiB / 81920MiB | 0% Default |
| | | Disabled |
```

```
+-----------------------------------------+-----------------------
+----------------------+
| 5 NVIDIA A100-SXM4-80GB On | 00000000:90:1D.0 Off | 0 |
| N/A 46C P0 69W / 400W | 0MiB / 81920MiB | 0% Default |
| | | Disabled |
+-----------------------------------------+-----------------------
+----------------------+
| 6 NVIDIA A100-SXM4-80GB On | 00000000:A0:1C.0 Off | 0 |
| N/A 52C P0 74W / 400W | 0MiB / 81920MiB | 0% Default |
| | | Disabled |
+-----------------------------------------+-----------------------
+----------------------+
| 7 NVIDIA A100-SXM4-80GB On | 00000000:A0:1D.0 Off | 0 |
| N/A 47C P0 72W / 400W | 0MiB / 81920MiB | 0% Default |
| | | Disabled |
+-----------------------------------------+-----------------------
+----------------------+



+-----------------------------------------------------------------------------------------------
+
| MIG devices: |
+------------------+--------------------------------+-----------
+-----------------------+
| GPU GI CI MIG | Memory-Usage | Vol| Shared |
| ID ID Dev | BAR1-Usage | SM Unc| CE ENC DEC OFA JPG |
| | | ECC| |
|==================+================================+===========
+=======================|
| 0 2 0 0 | 428MiB / 40192MiB | 42 0 | 3 0 2 0 0 |
| | 2MiB / 32767MiB | | |
+------------------+--------------------------------+-----------
+-----------------------+
| 0 3 0 1 | 71MiB / 19968MiB | 28 0 | 2 0 1 0 0 |
| | 0MiB / 16383MiB | | |
+------------------+--------------------------------+-----------
+-----------------------+
| 0 9 0 2 | 36MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 0MiB / 8191MiB | | |
+------------------+--------------------------------+-----------
+-----------------------+
| 0 10 0 3 | 36MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 0MiB / 8191MiB | | |
```

```
+------------------+----------------------------------+-----------
+-----------------------+
| 1 1 0 0 | 107MiB / 40192MiB | 42 0 | 3 0 2 0 0 |
| | 0MiB / 32767MiB | | |
+------------------+----------------------------------+-----------
+-----------------------+
| 1 5 0 1 | 71MiB / 19968MiB | 28 0 | 2 0 1 0 0 |
| | 0MiB / 16383MiB | | |
+------------------+----------------------------------+-----------
+-----------------------+
| 1 13 0 2 | 161MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 2MiB / 8191MiB | | |
+------------------+----------------------------------+-----------
+-----------------------+
| 1 14 0 3 | 36MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 0MiB / 8191MiB | | |
+------------------+----------------------------------+-----------
+-----------------------+
| 2 1 0 0 | 107MiB / 40192MiB | 42 0 | 3 0 2 0 0 |
| | 0MiB / 32767MiB | | |
+------------------+----------------------------------+-----------
+-----------------------+
| 2 5 0 1 | 289MiB / 19968MiB | 28 0 | 2 0 1 0 0 |
| | 2MiB / 16383MiB | | |
+------------------+----------------------------------+-----------
+-----------------------+
| 2 13 0 2 | 36MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 0MiB / 8191MiB | | |
+------------------+----------------------------------+-----------
+-----------------------+
| 2 14 0 3 | 36MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 0MiB / 8191MiB | | |
+------------------+----------------------------------+-----------
+-----------------------+
| 3 1 0 0 | 107MiB / 40192MiB | 42 0 | 3 0 2 0 0 |
| | 0MiB / 32767MiB | | |
+------------------+----------------------------------+-----------
+-----------------------+
| 3 5 0 1 | 71MiB / 19968MiB | 28 0 | 2 0 1 0 0 |
| | 0MiB / 16383MiB | | |
+------------------+----------------------------------+-----------
+-----------------------+
| 3 13 0 2 | 36MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 0MiB / 8191MiB | | |
```

```
+------------------+----------------------------------+-----------
+-----------------------+
| 3 14 0 3 | 36MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 0MiB / 8191MiB | | |
+------------------+----------------------------------+-----------
+-----------------------+



+-----------------------------------------------------------------------------------------
+
| Processes: |
| GPU GI CI PID Type Process name GPU Memory |
| ID ID Usage |
|
=========================================================================================|
**| 0 2 0 64080 C python 312MiB |
| 1 13 0 64085 C python 118MiB |
| 2 5 0 64073 C python 210MiB |**
+-----------------------------------------------------------------------------------------
+
```

Optimisez les charges de travail du GPU avec IMEX à l'aide d'instances P6e GB200

L'IMEX (Internode Memory Exchange) permet une communication cohérente en mémoire entre les nœuds pour un entraînement distribué sur NVIDIA. GB200 UltraServers

Procédez comme suit.

1. Définissez un `ComputeDomain` pour l'entraînement multi-nœuds avec un fichier nommé `imex-compute-domain.yaml` :

```
apiVersion: resource.nvidia.com/v1beta1
kind: ComputeDomain
metadata:
  name: distributed-training-domain
  namespace: default
spec:
  numNodes: 2
  channel:
    resourceClaimTemplate:
      name: imex-channel-template
```

2. Définissez un Pod en utilisant les canaux IMEX avec un fichier nommé `imex-pod.yaml` :

```
apiVersion: v1
kind: Pod
metadata:
  name: imex-distributed-training
  namespace: default
  labels:
    app: imex-training
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: nvidia.com/gpu.clique
            operator: Exists
  containers:
  - name: distributed-training
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["bash", "-c"]
    args:
    - |
      echo "=== IMEX Channel Verification ==="
      ls -la /dev/nvidia-caps-imex-channels/
      echo ""

      echo "=== GPU Information ==="
      nvidia-smi
      echo ""

      echo "=== NCCL Test (if available) ==="
      python -c "
      import torch
      import torch.distributed as dist
      import os

      print(f'CUDA available: {torch.cuda.is_available()}')
      print(f'CUDA device count: {torch.cuda.device_count()}')

      if torch.cuda.is_available():
          for i in range(torch.cuda.device_count()):
              print(f'GPU {i}: {torch.cuda.get_device_name(i)}')
```

```
      # Check for IMEX environment variables
      imex_vars = [k for k in os.environ.keys() if 'IMEX' in k or 'NVLINK' in k]
      if imex_vars:
          print('IMEX Environment Variables:')
          for var in imex_vars:
              print(f'  {var}={os.environ[var]}')

      print('IMEX channel verification completed')
      "

      # Keep container running for inspection
      sleep 3600
    resources:
      claims:
      - name: imex-channel-0
      - name: imex-channel-1
  resourceClaims:
  - name: imex-channel-0
    resourceClaimTemplateName: imex-channel-template
  - name: imex-channel-1
    resourceClaimTemplateName: imex-channel-template
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
```

> **Note**
>
> Cela nécessite des GB200 instances P6e.

3. Déployez IMEX en appliquant les modèles `ComputeDomain` et :

```
kubectl apply -f imex-claim-template.yaml
kubectl apply -f imex-compute-domain.yaml
kubectl apply -f imex-pod.yaml
```

4. Vérifiez le `ComputeDomain` statut.

```
kubectl get computedomain distributed-training-domain
```

5. Surveillez le déploiement du daemon IMEX.

```
kubectl get pods -n nvidia-dra-driver -l resource.nvidia.com/computeDomain
```

6. Vérifiez les canaux IMEX dans le Pod :

```
kubectl exec imex-distributed-training -- ls -la /dev/nvidia-caps-imex-channels/
```

7. Consultez les journaux du Pod :

```
kubectl logs imex-distributed-training
```

Voici un exemple de résultat attendu :

```
=== IMEX Channel Verification ===
total 0
drwxr-xr-x. 2 root root 80 Jul 8 10:45 .
drwxr-xr-x. 6 root root 380 Jul 8 10:45 ..
crw-rw-rw-. 1 root root 241, 0 Jul 8 10:45 channel0
crw-rw-rw-. 1 root root 241, 1 Jul 8 10:45 channel1
```

Pour plus d'informations, consultez l'exemple NVIDIA sur GitHub.

Réseaux

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Envisagez une bande passante réseau plus élevée ou un adaptateur Elastic Fabric pour les applications nécessitant une communication inter-nœuds élevée

Pour les charges de travail de formation distribuées sur Amazon EKS nécessitant de fortes communications entre nœuds, pensez à sélectionner des instances dotées d'une bande passante réseau plus importante ou un adaptateur Elastic Fabric (EFA). Des performances réseau insuffisantes peuvent entraver le transfert de données et ralentir les tâches d'apprentissage

automatique telles que la formation multiGPU distribuée. Notez que les charges de travail d'inférence ne présentent généralement pas un niveau élevé de communication entre les nœuds.

Exemple

Par exemple, en utilisant Karpenter :

```
apiVersion: v1
kind: Pod
metadata:
  name: ml-workload
spec:
  nodeSelector:
    karpenter.k8s.aws/instance-network-bandwidth: "100000"  # 100 Gbps in Mbps
    node.kubernetes.io/instance-type: p5.48xlarge  # EFA-enabled instance
  containers:
  - name: training-job
    image: `763104351884.dkr.ecr.us-west-2.amazonaws.com/pytorch-inference:2.6.0-gpu-
py312-cu124-ubuntu22.04-ec2-v1.6`
    resources:
      limits:
        vpc.amazonaws.com/efa: 1  # Requires EFA device plugin
```

Assurez-vous que des outils tels que MPI et NCCL sont installés dans l'image de votre conteneur afin de tirer parti de l'EFA pour les tâches de formation.

Augmentez le nombre d'adresses IP disponibles pour accélérer le lancement des pods

Dans EKS, chaque pod a besoin d'une adresse IP provenant du bloc d'adresse CIDR VPC. À mesure que votre cluster évolue avec un plus grand nombre de nœuds et de pods, vous risquez d'épuiser les adresses IP ou de ralentir les performances, mais l'activation de la délégation de préfixes peut atténuer ces problèmes en préallouant des plages d'adresses IP et en réduisant les appels d'API EC2, ce qui permet d'accélérer les temps de lancement des pods et d'améliorer l'évolutivité.

L'activation de la délégation de préfixes après la création de votre cluster permet à l'interface réseau du conteneur VPC (CNI) d'attribuer des préfixes IP (/28, chacun donnant 16 adresses IP) aux interfaces réseau sur les instances EC2. Cela signifie que chaque nœud peut prendre en charge un plus grand nombre de pods, réduisant ainsi le risque de pénurie d'adresses IP. Par exemple, sur une `c5.4xlarge` instance, vous pouvez prendre en charge jusqu'à 110 pods avec délégation de préfixe.

Bien que la délégation de préfixes soit essentielle pour optimiser l'utilisation de l'IP dans les environnements comportant de nombreux petits pods, les AI/ML charges de travail utilisent souvent des pods moins nombreux et plus grands (par exemple, un pod par GPU). L'activation de la délégation de préfixes permet au VPC CNI de préallouer un préfixe pour accélérer le démarrage du pod en maintenant un pool de chaleur. Cela signifie que les adresses IP sont facilement disponibles, ce qui réduit le temps nécessaire à l'initialisation du pod par rapport à l'allocation à la demande en mode sans préfixe. Dans de tels cas, les économies d'adresse IP réalisées grâce à l'activation de la délégation de préfixes offrent des avantages en termes de performances pour les charges AI/ML de travail. En réduisant le nombre d'appels d'API EC2 requis pour la configuration des adresses IP et la préallocation des plages d'adresses IP, l'utilisation de la délégation de préfixes permet d'accélérer les temps de lancement des pods, ce qui est particulièrement avantageux pour faire évoluer rapidement les charges de travail. AI/ML

Pour activer la délégation de préfixes :

```
kubectl set env daemonset/aws-node -n kube-system ENABLE_PREFIX_DELEGATION=true
```

Assurez-vous de planifier correctement les sous-réseaux VPC afin d'éviter l'épuisement des adresses IP, en particulier lors de déploiements de grande envergure, et gérez les blocs CIDR pour éviter les chevauchements entre eux. VPCs Pour en savoir plus, consultez Optimisation de l'utilisation des adresses IP et attribution d'adresses IP supplémentaires aux nœuds Amazon EKS avec des préfixes.

Sécurité

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Sécurité et conformité

Envisagez S3 avec KMS pour un stockage conforme au chiffrement

Sauf indication contraire de votre part, tous les compartiments S3 utilisent SSE-S3 par défaut pour chiffrer les objets au repos. Toutefois, vous pouvez choisir de configurer les compartiments pour utiliser le chiffrement côté serveur avec les clés AWS Key Management Service (AWS KMS) (SSE-KMS) à la place. Les contrôles de sécurité dans AWS KMS peuvent vous aider à respecter les

exigences de conformité liées au chiffrement. Vous pouvez utiliser ces clés KMS pour protéger les données dans les compartiments Simple Storage Service (Amazon S3). Lorsque vous utilisez le chiffrement SSE-KMS avec un compartiment S3, les clés AWS KMS doivent se trouver dans la même région que le compartiment.

Configurez vos compartiments à usage général pour utiliser les clés de compartiment S3 pour SSE-KMS, afin de réduire les coûts de vos demandes AWS KMS jusqu'à 99 % en diminuant le trafic de demandes d'Amazon S3 vers AWS KMS. Les clés de compartiment S3 sont toujours activées pour `GET` les `PUT` opérations dans un compartiment de répertoire et ne peuvent pas être désactivées.

Notez qu'Amazon S3 Express One Zone utilise un type de compartiment spécifique appelé compartiment d'annuaire S3. Les buckets d'annuaire sont exclusivement destinés à la classe de stockage S3 Express One Zone et permettent un accès à haute performance avec une faible latence. Pour configurer le chiffrement de compartiment par défaut sur un compartiment d'annuaire S3, utilisez l'interface de ligne de commande AWS et spécifiez l'ID ou l'ARN de la clé KMS, et non l'alias, comme dans l'exemple suivant :

Example

```
aws s3api put-bucket-encryption --bucket my-directory-bucket --server-side-encryption-
configuration \
   '{"Rules": [{"ApplyServerSideEncryptionByDefault": {"SSEAlgorithm": "aws:kms",
 "KMSMasterKeyID": "1234abcd-12ab-34cd-56ef-1234567890ab"}}]}'
```

Assurez-vous que le rôle IAM de votre pod EKS dispose des autorisations KMS (par exemple`kms:Decrypt`) pour accéder aux objets chiffrés. Testez cela dans un environnement intermédiaire en téléchargeant un exemple de modèle dans le compartiment, en le montant dans un module (par exemple, via le pilote Mountpoint S3 CSI) et en vérifiant que le module peut lire les données cryptées sans erreur. Journaux d'audit via AWS CloudTrail pour confirmer la conformité aux exigences de chiffrement. Consultez la documentation KMS pour les détails de configuration et la gestion des clés.

Stockage

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Gestion et stockage des données

Déployez des modèles d'IA sur des pods à l'aide d'un pilote CSI

Les charges de travail IA/ML nécessitent souvent l'accès à de grands artefacts de modèles (par exemple, poids entraînés, configurations), et les pods ont besoin d'un moyen fiable et évolutif pour y accéder sans les intégrer dans des images de conteneur, ce qui peut augmenter la taille des images et les temps d'extraction du registre des conteneurs. Pour réduire les frais opérationnels liés à la gestion des montages de volumes, nous recommandons de déployer des modèles d'IA sur des pods en installant les services de stockage Amazon (par exemple, S3, FSx pour Lustre, FSx pour OpenZFS, EFS) en tant que volumes persistants (PVs) à l'aide de leurs pilotes CSI respectifs. Pour plus de détails sur la mise en œuvre, consultez les rubriques suivantes de cette section.

Optimisation du stockage pour les caches de modèles ML sur EKS

Il est essentiel de tirer parti d'une solution de stockage optimale pour minimiser la latence de démarrage des pods et des applications, réduire l'utilisation de la mémoire, obtenir les niveaux de performance souhaités pour accélérer les charges de travail et garantir l'évolutivité des charges de travail de machine learning. Les charges de travail ML reposent souvent sur des fichiers modèles (poids), qui peuvent être volumineux et nécessiter un accès partagé aux données entre les pods ou les nœuds. Le choix de la solution de stockage optimale dépend des caractéristiques de votre charge de travail, telles que l'efficacité d'un nœud unique, l'accès à plusieurs nœuds, les exigences de latence, les contraintes de coûts ainsi que les exigences en matière d'intégration des données (comme dans le cas d'un référentiel de données Amazon S3). Nous vous recommandons d'évaluer les différentes solutions de stockage en fonction de vos charges de travail afin de déterminer laquelle répond à vos exigences. Nous vous proposons les options suivantes pour vous aider à évaluer en fonction de vos exigences en matière de charge de travail.

Le pilote EKS CSI prend en charge les services de stockage AWS suivants. Chacun possède son propre pilote CSI et possède ses propres atouts pour les flux de travail d'IA et de ML :

- Mountpoint pour Amazon S3
- Amazon FSx pour Lustre
- Amazon FSx pour OpenZFS
- Amazon EFS
- Amazon EBS

Le choix du service de stockage AWS dépend de votre architecture de déploiement, de votre échelle, de vos exigences en matière de performances et de votre stratégie en matière de coûts. Les pilotes CSI de stockage doivent être installés sur votre cluster EKS, ce qui permet au pilote CSI de créer et de gérer des volumes persistants (PV) en dehors du cycle de vie d'un pod. À l'aide du pilote CSI, vous pouvez créer des définitions PV des services de stockage AWS pris en charge en tant que ressources de cluster EKS. Les pods peuvent ensuite accéder à ces volumes de stockage pour leurs volumes de données en créant une réclamation de volume persistante (PVC) pour le PV. En fonction du service de stockage AWS et de votre scénario de déploiement, un seul PVC (et le PV associé) peut être attaché à plusieurs pods pour une charge de travail. Par exemple, pour la formation ML, les données d'entraînement partagées sont stockées sur un PV et accessibles par plusieurs Pods ; pour une inférence en ligne en temps réel, les modèles LLM sont mis en cache sur un PV et accessibles par plusieurs Pods. Des exemples de fichiers YAML PV et PVC pour les services de stockage AWS sont fournis ci-dessous pour vous aider à démarrer.

Surveillance des performances Les performances médiocres du disque peuvent retarder la lecture des images des conteneurs, augmenter le temps de latence au démarrage du pod et dégrader le débit d'inférence ou de formation. Utilisez Amazon CloudWatch pour surveiller les indicateurs de performance de vos services de stockage AWS. Lorsque vous identifiez des problèmes de performance, modifiez les paramètres de configuration de votre stockage afin d'optimiser les performances.

Scénario : charge de travail de plusieurs instances GPU

Amazon FSx for Lustre : dans les scénarios où vous disposez de plusieurs environnements d'instances de calcul GPU EC2 avec des charges de travail dynamiques sensibles à la latence et à haut débit de bande passante, tels que la formation distribuée et le service de modèles, et que vous avez besoin d'une intégration native au référentiel de données Amazon S3, nous recommandons Amazon for Lustre. FSx FSx for Lustre fournit un système de fichiers parallèle hautes performances entièrement géré, conçu pour les charges de travail intensives telles que le calcul haute performance (HPC) et le Machine Learning.

Vous pouvez installer le pilote CSI FSx for Lustre pour monter des FSx systèmes de fichiers sur EKS en tant que volume persistant (PV), puis le déployer FSx pour le système de fichiers Lustre en tant que cache haute performance autonome ou en tant que système de fichiers lié au S3 pour agir comme cache haute performance pour les données S3, fournissant ainsi un débit rapide I/O et élevé pour l'accès aux données entre vos instances de calcul GPU. FSx for Lustre peut être déployé avec les options de stockage Scratch-SSD ou Persistent-SSD :

- Stockage sur SSD Scratch : recommandé pour les charges de travail éphémères ou de courte durée (heures), avec une capacité de débit fixe par To allouée.
- Stockage SSD permanent : recommandé pour les charges de travail critiques de longue durée nécessitant le plus haut niveau de disponibilité, par exemple les simulations HPC, l'analyse de mégadonnées ou la formation au Machine Learning. Avec le stockage SSD persistant, vous pouvez configurer à la fois la capacité de stockage et la capacité de débit (par To) requises.

Considérations relatives aux performances :

- Module d'administration FSx pour gérer le système de fichiers Lustre : configurez un module « administratif » sur lequel le client Lustre est installé et sur lequel le système de fichiers FSx est monté. Cela permettra à un point d'accès de peaufiner le système de fichiers FSx, et également dans les situations où vous devez préchauffer le système de fichiers FSx avec vos données d'entraînement ML ou vos modèles LLM avant de démarrer vos instances de calcul GPU. Cela est particulièrement important si votre architecture utilise des GPU/compute instances Amazon EC2 basées sur Spot, dans lesquelles vous pouvez utiliser le module d'administration pour « réchauffer » ou « précharger » les données souhaitées dans le système de fichiers FSx, afin que les données soient prêtes à être traitées lorsque vous exécutez vos instances Amazon EC2 basées sur Spot.
- Elastic Fabric Adapter (EFA) : les types de déploiement de stockage SSD persistants prennent en charge Elastic Fabric Adapter (EFA), où l'utilisation d'EFA est idéale pour les charges de travail basées sur des GPU à hautes performances et basées sur le débit. Notez que FSx for Lustre prend en charge le GPUDirect stockage NVIDIA (GDS), technologie qui crée un chemin de données direct entre le stockage local ou distant et la mémoire du GPU, afin de permettre un accès plus rapide aux données.
- Compression : activez la compression des données sur le système de fichiers si certains types de fichiers peuvent être compressés. Cela peut contribuer à améliorer les performances, car la compression des données réduit la quantité de données transférées entre les serveurs FSx de fichiers Lustre et le stockage.
- Configuration du découpage du système de fichiers Lustre :
  - Bandage des données : permet à Luster de distribuer FSx les données d'un fichier sur plusieurs cibles de stockage d'objets (OSTs) au sein d'un système de fichiers Lustre, ce qui maximise l'accès parallèle et le débit, en particulier pour les tâches de formation ML à grande échelle.
  - Système de fichiers autonome : Par défaut, une configuration de découpage Lustre à 4 composants est créée pour vous via la fonctionnalité PFL (Progressive File Layouts) de for

Lustre. FSx Dans la plupart des scénarios, il n'est pas nécessaire de mettre à jour le comptage et la taille des bandes PFL Lustre par défaut. Si vous devez ajuster le découpage des données Lustre, vous pouvez le régler manuellement en vous référant aux paramètres de découpage d'un système de fichiers FSx pour Lustre.

- Système de fichiers lié au S3 : les fichiers importés dans le système de FSx fichiers à l'aide de l'intégration native Amazon S3 (Data Repository Association ou DRA) n'utilisent pas la mise en page PFL par défaut, mais utilisent plutôt la mise en page dans les paramètres du système de fichiers. `ImportedFileChunkSize` Les fichiers importés au format S3 dont la taille est supérieure à la taille `ImportedFileChunkSize` seront stockés sur plusieurs OSTs avec un nombre de bandes basé sur la valeur `ImportedFileChunkSize` définie (1 Go par défaut). Si vous avez des fichiers volumineux, nous vous recommandons de régler ce paramètre sur une valeur supérieure.
- Emplacement : déployez un système de fichiers FSx pour Lustre dans la même zone de disponibilité que vos nœuds de calcul ou de GPU afin de permettre un accès aux données avec le plus faible temps de latence et d'éviter les modèles d'accès entre zones de disponibilité. Si vous avez plusieurs nœuds GPU situés dans différentes zones de disponibilité, nous vous recommandons de déployer un système de FSx fichiers dans chaque zone de disponibilité pour un accès aux données à faible latence.

Exemple

Définition du volume persistant (PV) pour un système de fichiers FSx pour Lustre, à l'aide du provisionnement statique (lorsque l' FSx instance a déjà été provisionnée).

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: fsx-pv
spec:
  capacity:
    storage: 1200Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  mountOptions:
    - flock
  persistentVolumeReclaimPolicy: Recycle
  csi:
    driver: fsx.csi.aws.com
```

```
    volumeHandle: [FileSystemId of FSx instance]
    volumeAttributes:
      dnsname: [DNSName of FSx instance]
      mountname: [MountName of FSx instance]
```

Exemple

Définition de la demande de volume persistant pour le PV appelée `fsx-pv` :

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: fsx-claim
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 1200Gi
  volumeName: fsx-pv
```

Exemple

Configurez un pod pour utiliser une réclamation de volume persistante de `fsx-claim` :

```
apiVersion: v1
kind: Pod
metadata:
  name: fsx-app
spec:
  containers:
  - name: app
    image: amazonlinux:2023
    command: ["/bin/sh"]
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: fsx-claim
```

Pour des exemples complets, consultez les exemples FSx de pilotes Lustre dans GitHub. Surveillez les indicateurs de performance FSx d'Amazon for Lustre à l'aide d'Amazon CloudWatch. Lorsque des problèmes de performance sont identifiés, ajustez vos paramètres de configuration selon vos besoins.

Scénario : charge de travail d'une instance de GPU unique

Mountpoint pour Amazon S3 avec pilote CSI : vous pouvez monter un compartiment S3 sous forme de volume dans vos pods à l'aide du pilote CSI Mountpoint pour Amazon S3. Cette méthode permet un contrôle d'accès précis sur les pods autorisés à accéder à des compartiments S3 spécifiques. Chaque pod possède sa propre instance de point de montage et son propre cache local (5 à 10 Go), ce qui permet d'isoler les performances de chargement et de lecture des modèles entre les pods. Cette configuration prend en charge l'authentification au niveau du pod avec les rôles IAM pour les comptes de service (IRSA) et le versionnement indépendant des modèles pour différents modèles ou clients. Le compromis est une augmentation de l'utilisation de la mémoire et du trafic d'API, car chaque pod émet des appels d'API S3 et gère son propre cache.

Exemple partiel de déploiement d'un Pod YAML avec pilote CSI :

```
# CSI driver dynamically mounts the S3 bucket for each pod

volumes:
  - name: s3-mount
    csi:
      driver: s3.csi.aws.com
      volumeAttributes:
        bucketName: your-s3-bucket-name
        mountOptions: "--allow-delete"  # Optional
        region: us-west-2

containers:
  - name: inference
    image: your-inference-image
    volumeMounts:
      - mountPath: /models
        name: s3-mount
volumeMounts:
  - name: model-cache
    mountPath: /models
volumes:
  - name: model-cache
    hostPath:
```

```
      path: /mnt/s3-model-cache
```

Considérations relatives aux performances :

- Mise en cache des données : Mountpoint for S3 peut mettre en cache le contenu afin de réduire les coûts et d'améliorer les performances lors de lectures répétées d'un même fichier. Reportez-vous à la section Configuration de la mise en cache pour connaître les options et les paramètres de mise en cache.
- Taille partielle de l'objet : lorsque vous stockez et accédez à des fichiers d'une taille supérieure à 72 Go, reportez-vous à la section Configuration des performances de Mountpoint pour comprendre comment configurer les paramètres et les paramètres de `--write-part-size` ligne de commande en fonction de votre profil de données `--read-part-size` et de vos exigences en matière de charge de travail.
- Le cache partagé est conçu pour les objets d'une taille maximale de 1 Mo. Il ne supporte pas les objets de grande taille. Utilisez l'option de cache local pour mettre en cache des objets NVMe ou des volumes EBS sur le nœud EKS.
- Frais de demande d'API : lorsque vous effectuez un grand nombre d'opérations sur les fichiers avec Mountpoint pour S3, les frais de demande d'API peuvent devenir une partie des coûts de stockage. Pour atténuer ce problème, si une forte cohérence n'est pas requise, activez toujours la mise en cache des métadonnées et définissez la `metadata-ttl` période pour réduire le nombre d'opérations d'API à S3.

Pour plus de détails, consultez le pilote CSI Mountpoint pour Amazon S3 dans la documentation officielle Amazon EKS. Nous vous recommandons de surveiller les indicateurs de performance d'Amazon S3 à l'aide des CloudWatch indicateurs Amazon en cas de goulots d'étranglement et d'ajuster votre configuration si nécessaire.

Stockage partagé persistant Amazon FSx pour OpenZFS

Pour les scénarios impliquant plusieurs instances de calcul GPU EC2 avec des charges de travail sensibles à la latence nécessitant une haute disponibilité, des performances élevées, une sensibilité aux coûts et le déploiement de plusieurs pods pour différentes applications, nous recommandons Amazon pour OpenZFS. FSx Certains exemples de charge de travail incluent l'inférence en temps réel, l'apprentissage par renforcement et la formation de réseaux antagonistes génératifs. FSx pour OpenZFS est particulièrement utile pour les charges de travail nécessitant un accès performant à une structure de répertoires ciblée contenant de petits fichiers utilisant de petits modèles d'accès aux données d'E/S. En outre, OpenZFS offre la flexibilité nécessaire FSx pour adapter les performances

indépendamment de la capacité de stockage, ce qui vous aide à atteindre une rentabilité optimale en adaptant la taille du stockage aux besoins réels tout en maintenant les niveaux de performance requis.

Le pilote CSI natif FSx d'OpenZFS permet de créer plusieurs PVCs ou un seul système de fichiers en créant plusieurs volumes. Cela permet de réduire les frais de gestion et d'optimiser l'utilisation du débit et des IOPS du système de fichiers grâce à des déploiements de modules d'applications consolidés sur un seul système de fichiers. En outre, il inclut des fonctionnalités d'entreprise telles que les instantanés sans copie, les clones sans copie et les quotas d'utilisateurs et de groupes qui peuvent être provisionnés de manière dynamique via le pilote CSI.

FSx car OpenZFS prend en charge trois types de déploiement différents lors de sa création :

- Mono-AZ : option la plus économique avec des latences inférieures à la milliseconde, mais qui ne fournit aucune haute disponibilité au niveau du système de fichiers ou de la zone de disponibilité. Recommandé pour les charges de travail de développement et de test ou celles présentant une haute disponibilité au niveau de la couche application.
- Mono-AZ (HA) : fournit une haute disponibilité au niveau du système de fichiers avec des latences inférieures à la milliseconde. Recommandé pour les charges de travail les plus performantes qui nécessitent une haute disponibilité.
- Multi-AZ : assure une haute disponibilité au niveau du système de fichiers ainsi que dans toutes les zones de disponibilité. Recommandé pour les charges de travail à hautes performances qui nécessitent une disponibilité accrue dans toutes les zones de disponibilité.

Considérations relatives aux performances :

- Type de déploiement : si la disponibilité supplémentaire entre les zones de disponibilité n'est pas une exigence, envisagez d'utiliser le type de déploiement mono-AZ (HA). Ce type de déploiement fournit jusqu'à 100 % du débit pour les écritures, maintient des latences inférieures à la milliseconde et les systèmes de fichiers Gen2 disposent d'un NVMe cache supplémentaire pour stocker jusqu'à plusieurs téraoctets de données fréquemment consultées. Les systèmes de fichiers multi-AZ fournissent jusqu'à 75 % du débit pour les écritures avec une latence accrue afin de tenir compte du trafic inter-AZ.
- Débit et IOPS : le débit et les IOPS configurés pour le système de fichiers peuvent être augmentés ou diminués après le déploiement. Vous pouvez fournir jusqu'à 10 % GB/s of disk throughput providing up to 21GB/s de l'accès aux données mises en cache. Les IOPS peuvent être augmentées jusqu'à 400 000 à partir du disque et le cache peut fournir plus d'un million d'IOPS.

Notez que le dimensionnement du débit d'un système de fichiers mono-AZ entraîne une brève interruption du système de fichiers car il n'existe aucune haute disponibilité. La mise à l'échelle du débit d'un système de fichiers mono-AZ (HA) ou multi-AZ peut être effectuée sans interruption de service. Les IOPS du SSD peuvent être ajustés une fois toutes les six heures.

- Classe de stockage : FSx pour OpenZFS, prend en charge à la fois la classe de stockage SSD et la classe de stockage Intelligent-Tiering. Pour les AI/ML charges de travail, il est recommandé d'utiliser la classe de stockage SSD qui fournit des performances cohérentes à la charge de travail en maintenant les processeurs/GPU aussi occupés que possible.
- Compression : activez l'algorithme LZ4 de compression si vous avez une charge de travail qui peut être compressée. Cela réduit la quantité de données que chaque fichier consomme dans le cache, ce qui permet de diffuser davantage de données directement à partir du cache sous forme de débit réseau et d'IOPS réduisant la charge sur le disque SSD.
- Taille d'enregistrement : la plupart des AI/ML charges de travail gagneront à conserver la taille d'enregistrement par défaut de 128 Ko. Cette valeur ne doit être réduite que si l'ensemble de données est constitué de fichiers volumineux (supérieurs à 10 Go) avec un accès constant à de petits blocs inférieur à 128 Ko depuis l'application.

Une fois le système de fichiers créé, un volume racine associé est automatiquement créé par le service. Il est recommandé de stocker les données dans les volumes enfants du volume racine du système de fichiers. À l'aide du pilote CSI FSx pour OpenZFS, vous créez une réclamation de volume persistant associée pour créer dynamiquement le volume enfant.

Exemples :

Définition de classe de stockage (SC) pour un volume FSx pour OpenZFS, utilisée pour créer un volume enfant du volume racine ($ROOT_VOL_ID) sur un système de fichiers existant et exporter le volume vers le VPC CIDR ($VPC_CIDR) à l'aide du protocole NFS v4.2.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fsxz-vol-sc
provisioner: fsx.openzfs.csi.aws.com
parameters:
  ResourceType: "volume"
  ParentVolumeId: '"$ROOT_VOL_ID"'
  CopyTagsToSnapshots: 'false'
  DataCompressionType: '"LZ4"'
```

```
  NfsExports: '[{"ClientConfigurations": [{"Clients": "$VPC_CIDR", "Options":
 ["rw","crossmnt","no_root_squash"]}]}]'
  ReadOnly: 'false'
  RecordSizeKiB: '128'
  Tags: '[{"Key": "Name", "Value": "AI-ML"}]'
  OptionsOnDeletion: '["DELETE_CHILD_VOLUMES_AND_SNAPSHOTS"]'
reclaimPolicy: Delete
allowVolumeExpansion: false
mountOptions:
  - nfsvers=4.2
  - rsize=1048576
  - wsize=1048576
  - timeo=600
  - nconnect=16
  - async
```

Une réclamation de volume persistante (PVC) créée dynamiquement par rapport à ce qui fsxz-vol-sc a été créé ci-dessus. Notez que la capacité de stockage allouée est de 1 Go, ce qui est requis FSx pour les volumes OpenZFS, comme indiqué dans la FAQ sur le pilote CSI. Le volume bénéficiera de la pleine capacité allouée au système de fichiers avec cette configuration. Si la capacité du volume doit être limitée, vous pouvez le faire en utilisant des quotas d'utilisateurs ou de groupes.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: dynamic-vol-pvc
  namespace: example
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: fsxz-vol-sc
  resources:
    requests:
      storage: 1Gi
```

Configurez un pod pour monter un volume en utilisant le Persistent Volume Claim (PVC) de dynamic-vol-pvc :

```
kind: Pod
apiVersion: v1
metadata:
  name: fsx-app
```

```
  namespace: example
spec:
  volumes:
    - name: dynamic-vol-pv
      persistentVolumeClaim:
        claimName: dynamic-vol-pvc
  containers:
    - name: app
      image: amazonlinux:2023
      command: ["/bin/sh"]
      volumeMounts:
        - mountPath: "/mnt/fsxz"
          name: dynamic-vol-pv
```

Amazon EFS pour les caches de modèles partagés

Dans les scénarios où vous disposez d'un environnement d'instances de calcul GPU EC2 multiples et que vous avez des charges de travail dynamiques nécessitant un accès partagé aux modèles entre plusieurs nœuds et zones de disponibilité (par exemple, inférence en ligne en temps réel avec Karpenter) avec des besoins modérés en termes de performances et d'évolutivité, nous vous recommandons d'utiliser un système de fichiers Amazon Elastic File System (EFS) comme volume persistant via le pilote EFS CSI. Amazon EFS est un système de fichiers NFS basé sur le cloud entièrement géré, hautement disponible et évolutif qui permet aux instances et aux conteneurs EC2 de bénéficier d'un stockage de fichiers partagé, avec des performances constantes, et où aucun provisionnement initial du stockage n'est requis. Utilisez EFS comme volume modèle et montez le volume en tant que système de fichiers partagé en définissant un volume persistant sur le cluster EKS. Chaque demande de volume persistant (PVC) soutenue par un système de fichiers EFS est créée en tant que point d'accès EFS au système de fichiers EFS. EFS permet à plusieurs nœuds et pods d'accéder aux mêmes fichiers de modèle, éliminant ainsi le besoin de synchroniser les données avec le système de fichiers de chaque nœud. Installez le pilote EFS CSI pour intégrer EFS à EKS.

Vous pouvez déployer un système de fichiers Amazon EFS avec les modes de débit suivants :

- Débit en rafale : adapte le débit à la taille du système de fichiers, adapté à des charges de travail variées avec des rafales occasionnelles.
- Débit provisionné : débit dédié, idéal pour les tâches de formation au machine learning cohérentes avec des besoins de performance prévisibles dans des limites limites.
- Débit élastique (recommandé pour le ML) : s'adapte automatiquement en fonction de la charge de travail, rentabilité pour différentes charges de travail ML.

Pour consulter les spécifications de performances, consultez les spécifications de performance d'Amazon EFS.

Considérations relatives aux performances :

- Utilisez Elastic Throughput pour différentes charges de travail.
- Utilisez la classe de stockage standard pour les charges de travail ML actives.

Pour des exemples complets d'utilisation du système de fichiers Amazon EFS en tant que volume persistant au sein de votre cluster EKS et de vos pods, reportez-vous aux exemples de pilotes EFS CSI dans GitHub. Surveillez les indicateurs de performance d'Amazon EFS à l'aide d'Amazon CloudWatch. Lorsque des problèmes de performance sont identifiés, ajustez vos paramètres de configuration selon vos besoins.

Utilisez S3 Express One Zone pour les flux de travail orientés objet et sensibles à la latence

Pour les AI/ML charges de travail sensibles à la latence sur Amazon EKS, telles que la formation de modèles à grande échelle, l'inférence ou les analyses hautes performances, nous recommandons d'utiliser S3 Express One Zone pour le stockage et la récupération de modèles à hautes performances. S3 Express One Zone propose un espace de noms hiérarchique, comme un système de fichiers, dans lequel il suffit de le télécharger dans un compartiment de répertoire, ce qui permet de « tout stocker » tout en maintenant une vitesse élevée. Cela est particulièrement utile si vous êtes habitué aux flux de travail orientés objet. Sinon, si vous êtes plus habitué aux systèmes de fichiers (par exemple, compatibles POSIX), vous pouvez préférer Amazon FSx for Lustre ou OpenZFS. Amazon S3 Express One Zone stocke les données dans une seule zone de disponibilité (AZ) à l'aide de compartiments de répertoire et offre une latence inférieure à celle des compartiments S3 standard, qui distribuent les données entre plusieurs. AZs Pour de meilleurs résultats, assurez-vous de colocaliser votre ordinateur EKS dans la même zone AZ que votre bucket Express One Zone. Pour en savoir plus sur les différences entre S3 Express One Zone, consultez la section Différences entre les compartiments d'annuaire.

Pour accéder à S3 Express One Zone avec la sémantique du système de fichiers, nous vous recommandons d'utiliser le pilote CSI Mountpoint S3, qui monte les compartiments S3 (y compris Express One Zone) en tant que système de fichiers local. Cela traduit les opérations sur les fichiers (par exemple, ouverture, lecture, écriture) en appels d'API S3, fournissant un accès haut débit optimisé pour les charges de travail intensives en lecture provenant de plusieurs clients et les écritures séquentielles sur de nouveaux objets. Pour plus de détails sur les opérations prises en

charge et les limitations (par exemple, aucune conformité totale avec POSIX, mais les ajouts et les renommages pris en charge dans Express One Zone), consultez la documentation sémantique de Mountpoint.

Avantages en termes de performances

- Fournit un accès aux données jusqu'à 10 fois plus rapide que le S3 Standard, avec une latence constante d'une milliseconde à un chiffre et des coûts de demande jusqu'à 80 % inférieurs.
- S'adapte pour traiter des centaines de milliers, voire des millions de requêtes par seconde et par compartiment de répertoire, évitant ainsi les ralentissements ou les baisses de tension observés dans la norme S3 lors de charges extrêmes (par exemple, en provenance de clusters comportant des dizaines, voire des centaines de milliers de GPUs/CPUs réseaux saturés).
- Utilise un mécanisme d'authentification basé sur les sessions. Authentifiez-vous une fois pour obtenir un jeton de session, puis effectuez des opérations répétées à grande vitesse sans surcharger l'authentification par demande. Ceci est optimisé pour les charges de travail telles que les points de contrôle fréquents ou le chargement de données.

Cas d'utilisation recommandés

- Mise en cache : L'un des principaux cas d'utilisation du pilote CSI Mountpoint S3 avec S3 Express One Zone est la mise en cache. La première instance lit les données depuis S3 Standard (usage général) et les met en cache dans Express One Zone à faible latence. Les lectures ultérieures effectuées par d'autres clients accèdent plus rapidement aux données mises en cache, ce qui est idéal pour les scénarios à nœuds multiples dans lesquels plusieurs nœuds EKS lisent les mêmes données (par exemple, des ensembles de données d'entraînement partagés). Cela peut améliorer les performances jusqu'à 7 fois pour les accès répétés et réduire les coûts de calcul. Pour les charges de travail nécessitant une conformité totale avec POSIX (par exemple, verrouillage de fichiers et modifications sur place), envisagez Amazon FSx for Lustre ou OpenZFS comme alternatives.
- AI/ML Formation et inférence à grande échelle : idéal pour les charges de travail comportant des centaines ou des milliers de nœuds de calcul (par exemple, GPUs dans les clusters EKS) où la régulation S3 à usage général peut entraîner des retards et gaspiller des ressources informatiques coûteuses. Par exemple, les chercheurs en LLM ou les organisations utilisant un modèle quotidien tests/checkpoints bénéficient d'un accès rapide et fiable sans perturber le S3 régional. Pour les charges de travail à petite échelle (par exemple, des dizaines de nœuds), S3 Standard ou d'autres classes de stockage peuvent suffire.

- Pipelines de données : Load/prepare modèles, données d'entraînement archivées ou points de contrôle des flux. Si votre équipe préfère le stockage d'objets aux systèmes de fichiers traditionnels (par exemple, en raison de sa familiarité avec S3), utilisez-le au lieu de procéder à des modifications techniques pour des options conformes à POSIX, comme FSx pour Lustre.

Considérations

- Résilience : la conception mono-AZ offre une durabilité de 99,999999999 % (identique à celle du S3 standard, grâce à la redondance au sein de l'AZ) mais une disponibilité inférieure (99,95 % conçu, 99,9 % SLA) par rapport aux classes multi-AZ (disponibilité de 99,99 %). Il est moins résistant aux défaillances de l'AZ. À utiliser pour les données recréables ou mises en cache. Envisagez la réplication multi-AZ ou les sauvegardes pour les charges de travail critiques.
- Support des API et des fonctionnalités : prend en charge un sous-ensemble de S3 APIs (par exemple, aucune politique de cycle de vie ni réplication) ; peut nécessiter des modifications mineures de l'application pour l'authentification de session ou la gestion des objets.
- Intégration EKS : colocalisez votre EKS pods/nodes dans la même zone que le compartiment d'annuaire afin de minimiser la latence du réseau. Utilisez Mountpoint pour Amazon S3 ou les pilotes CSI pour un accès natif à Kubernetes.
- Tests : testez la latence de récupération dans un environnement hors production pour valider les gains de performances. Surveillez les ralentissements dans les scénarios S3 standard (par exemple, saturation élevée du GPU) et comparez.

La classe de stockage S3 Express One Zone est disponible dans plusieurs régions et s'intègre à EKS pour les charges de travail nécessitant un accès aux objets sans attendre le stockage. Pour en savoir plus, consultez Commencer à utiliser S3 Express One Zone.

Observabilité

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Surveillance et observabilité

Explication des métriques du GPU

La métrique d'utilisation du GPU indique si le GPU a exécuté du travail pendant la fenêtre d'exemple. Cette métrique capture le pourcentage de temps pendant lequel le GPU a exécuté au moins une instruction, mais elle ne révèle pas l'efficacité avec laquelle le GPU a utilisé son matériel. Un GPU contient plusieurs multiprocesseurs de streaming (SMs), qui sont des unités de traitement parallèle qui exécutent les instructions. Un taux d'utilisation de 100 % peut signifier que le GPU a exécuté de lourdes charges de travail parallèles sur tous ses SMs composants, ou cela peut signifier qu'une seule petite instruction a activé le GPU au cours de la période d'échantillonnage. Pour comprendre l'utilisation réelle, vous devez examiner les métriques du GPU à plusieurs niveaux de l'architecture matérielle. Chaque multiprocesseur de streaming est construit à partir de différents types de cœurs, et chaque couche présente des caractéristiques de performance différentes. Des indicateurs de haut niveau (utilisation du processeur graphique, utilisation de la mémoire, puissance du processeur graphique et température du processeur, visibles via nvidia-smi) indiquent si l'appareil est actif. Des indicateurs plus approfondis (utilisation du SM, activité du SM et utilisation du noyau tensoriel) révèlent l'efficacité avec laquelle le GPU utilise ses ressources.

Ciblez une consommation d'énergie élevée du GPU

La sous-utilisation GPUs gaspille la capacité de calcul et augmente les coûts car les charges de travail n'impliquent pas tous les composants du GPU simultanément. Pour les AI/ML charges de travail sur Amazon EKS, suivez la consommation d'énergie du GPU en tant que proxy afin d'identifier l'activité réelle du GPU. L'utilisation du GPU indique le pourcentage de temps pendant lequel le GPU exécute un noyau, mais ne révèle pas si les multiprocesseurs de streaming, les contrôleurs de mémoire et les cœurs tensoriels sont tous actifs en même temps. La consommation d'énergie met en évidence cet écart, car le matériel entièrement engagé consomme beaucoup plus d'énergie que le matériel utilisant des noyaux légers ou restant inactif entre les tâches. Comparez la consommation d'énergie à la puissance thermique nominale (TDP) du GPU pour détecter la sous-utilisation, puis déterminez si votre charge de travail est entravée par le prétraitement du processeur, les E/S réseau ou des tailles de lots inefficaces.

CloudWatch Configurez Container Insights sur Amazon EKS pour identifier les pods, les nœuds ou les charges de travail consommant peu d'énergie du processeur graphique. Cet outil s'intègre directement à Amazon EKS et vous permet de surveiller la consommation d'énergie du GPU et d'ajuster la planification des pods ou les types d'instances lorsque la consommation d'énergie tombe en dessous de vos niveaux cibles. Si vous avez besoin d'une visualisation avancée ou

de tableaux de bord personnalisés, utilisez DCGM-Exporter de NVIDIA avec Prometheus et Grafana pour une surveillance native de Kubernetes. Les deux approches mettent en évidence des indicateurs clés de NVIDIA tels que `nvidia_smi_power_draw` (consommation d'énergie du GPU) et `nvidia_smi_temperature_gpu` (température du GPU). Pour obtenir la liste des métriques, explorez https://docs.aws.amazon.com/AmazonCloudWatch/ latest/monitoring/CloudWatch -Agent-NVIDIA-GPU.htm. Recherchez des modèles tels qu'une faible consommation d'énergie constante pendant des heures spécifiques ou pour des tâches particulières. Ces tendances vous aident à identifier les domaines dans lesquels consolider les charges de travail ou ajuster l'allocation des ressources.

Les limites de ressources statiques dans Kubernetes (telles que le nombre de processeurs, de mémoire et de GPU) entraînent souvent un provisionnement excessif ou une sous-utilisation, en particulier pour les charges de AI/ML travail dynamiques telles que l'inférence lorsque la demande fluctue. Analysez vos tendances d'utilisation et consolidez vos charges de travail en réduisant le nombre de vos charges de GPUs travail. Assurez-vous que chaque GPU est pleinement utilisé avant d'en allouer d'autres. Cette approche permet de réduire le gaspillage et les coûts. Pour obtenir des conseils détaillés sur l'optimisation des stratégies de planification et de partage, consultez les meilleures pratiques d'EKS en matière de calcul et d'autoscaling

Observabilité et métriques

Utilisation d'outils de surveillance et d'observabilité pour vos charges de travail AI/ML

Les AI/ML services modernes nécessitent une coordination entre l'infrastructure, la modélisation et la logique des applications. Les ingénieurs de plateforme gèrent l'infrastructure et le stack d'observabilité. Ils collectent, stockent et visualisent les métriques. AI/ML les ingénieurs définissent des métriques spécifiques au modèle et surveillent les performances sous différentes charges et distributions de données. Les développeurs d'applications consomment APIs, acheminent les demandes et suivent les indicateurs de niveau de service et les interactions des utilisateurs. Sans pratiques d'observabilité unifiées, ces équipes travaillent en silos et passent à côté de signaux critiques concernant l'état et les performances du système. L'établissement d'une visibilité partagée entre les environnements permet à toutes les parties prenantes de détecter les problèmes à un stade précoce et de maintenir un service fiable.

L'optimisation des clusters Amazon EKS pour les AI/ML charges de travail présente des défis de surveillance uniques, notamment en ce qui concerne la gestion de la mémoire du GPU. Sans surveillance adéquate, les entreprises sont confrontées à des erreurs out-of-memory (OOM), à une inefficacité des ressources et à des coûts inutiles. Une surveillance efficace garantit de meilleures

performances, une meilleure résilience et une réduction des coûts pour les clients d'EKS. Utilisez une approche holistique qui combine trois niveaux de surveillance. Tout d'abord, surveillez les métriques granulaires du GPU à l'aide de NVIDIA DCGM Exporter pour suivre la consommation d'énergie du GPU, la température du GPU, l'activité du SM, l'occupation du SM et les erreurs XID. Ensuite, surveillez les frameworks de service d'inférence tels que Ray et vLLM pour obtenir des informations sur les charges de travail distribuées grâce à leurs métriques natives. Troisièmement, collectez des informations au niveau de l'application pour suivre les indicateurs personnalisés spécifiques à votre charge de travail. Cette approche en couches vous offre une visibilité allant de l'utilisation du matériel aux performances des applications.

Outils et cadres

Plusieurs outils et frameworks fournissent des out-of-the-box métriques natives pour surveiller les AI/ML charges de travail. Ces métriques intégrées éliminent le besoin d'une instrumentation personnalisée et réduisent le temps de configuration. Les métriques se concentrent sur les aspects de performance tels que la latence, le débit et la génération de jetons, qui sont essentiels pour le service d'inférence et l'analyse comparative. L'utilisation de métriques natives vous permet de commencer à surveiller immédiatement sans créer de pipelines de collecte personnalisés.

- vLLM : moteur de service à haut débit pour les grands modèles linguistiques (LLMs) qui fournit des mesures natives telles que la latence des demandes et l'utilisation de la mémoire.
- Ray : infrastructure informatique distribuée qui émet des mesures pour les charges de travail évolutives de l'IA, notamment les temps d'exécution des tâches et l'utilisation des ressources.
- Hugging Face Text Generation Inference (TGI) : une boîte à outils pour le déploiement et le LLMs service, avec des métriques intégrées pour les performances d'inférence.
- NVIDIA genai-perf : outil en ligne de commande permettant d'évaluer les modèles d'IA génératifs, de mesurer le débit, la latence et les indicateurs spécifiques au LLM, tels que les demandes traitées à des intervalles de temps spécifiques.

Méthodes d'observabilité

Nous recommandons de mettre en œuvre tout mécanisme d'observabilité supplémentaire de l'une des manières suivantes.

CloudWatch Container Insights Si votre entreprise préfère les outils natifs d'AWS avec une configuration minimale, nous recommandons CloudWatch Container Insights. Il s'intègre à l'exportateur NVIDIA DCGM pour collecter les métriques du GPU et propose des tableaux de

bord prédéfinis pour des informations rapides. Activé par l'installation du module complémentaire CloudWatch Observability sur votre cluster, Container Insights déploie et gère le cycle de vie de l'exportateur NVIDIA DCGM, qui collecte les métriques du GPU à partir des pilotes de Nvidia et les expose à celles-ci. CloudWatch

Après avoir installé Container Insights, il détecte CloudWatch automatiquement NVIDIA GPUs dans votre environnement et collecte des indicateurs de santé et de performance critiques. Ces statistiques apparaissent sur des out-of-the-box tableaux de bord sélectionnés. Vous pouvez également intégrer Ray et vLLM à CloudWatch l'aide de l' CloudWatch agent unifié pour envoyer leurs métriques natives. Cette approche unifiée simplifie l'observabilité dans les environnements EKS et permet aux équipes de se concentrer sur le réglage des performances et l'optimisation des coûts au lieu de créer une infrastructure de surveillance.

Pour obtenir la liste complète des métriques disponibles, consultez les métriques Amazon EKS et Kubernetes Container Insights. Pour step-by-step obtenir des conseils sur la mise en œuvre de la surveillance du GPU, reportez-vous à la section Obtenir des informations opérationnelles pour les charges de travail des GPU NVIDIA à l'aide d'Amazon CloudWatch Container Insights. Pour des exemples pratiques d'optimisation de la latence d'inférence, consultez Optimisation de la réactivité de l'IA : guide pratique de l'inférence optimisée pour la latence d'Amazon Bedrock.

Prometheus et Grafana gérés Si votre entreprise a besoin de tableaux de bord personnalisés et de fonctionnalités de visualisation avancées, déployez Prometheus avec le NVIDIA DCGM-Exporter et Grafana pour la surveillance native de Kubernetes. Prometheus récupère et stocke les métriques du GPU à partir du DCGM-Exporter, tandis que Grafana fournit des fonctionnalités de visualisation et d'alerte flexibles. Cette approche vous permet de mieux contrôler la conception du tableau de bord et la rétention des métriques par rapport à CloudWatch Container Insights.

Vous pouvez étendre cette pile de surveillance en intégrant des frameworks open source tels que Ray et VLLM Ray et VLLM pour exporter leurs métriques natives vers Prometheus. Vous pouvez également connecter Grafana à une source de données AWS X-Ray pour visualiser les traces distribuées et identifier les obstacles aux performances dans votre pipeline d'inférence. Cette combinaison fournit une end-to-end visibilité à partir des métriques au niveau du GPU via les flux de demandes au niveau de l'application.

Pour step-by-step obtenir des conseils sur le déploiement de cette pile de surveillance, reportez-vous à la section Surveillance des charges de travail du GPU sur Amazon EKS à l'aide des services open source gérés par AWS.

Envisagez de surveiller la formation de base et de peaufiner les indicateurs

Surveillez les indicateurs d'entraînement de base pour suivre l'état et les performances de votre cluster Amazon EKS ainsi que les charges de travail d'apprentissage automatique qui y sont exécutées. Les charges de travail de formation ont des exigences de surveillance différentes de celles des charges de travail d'inférence, car elles s'exécutent sur de longues périodes, consomment les ressources différemment et nécessitent une visibilité sur la convergence des modèles et l'efficacité du pipeline de données. Les indicateurs ci-dessous vous aident à identifier les goulets d'étranglement, à optimiser l'allocation des ressources et à garantir le succès des tâches de formation. Pour step-by-step obtenir des conseils sur la mise en œuvre de cette approche de surveillance, reportez-vous à la section Introduction à l'observation des charges de travail d'apprentissage automatique sur Amazon EKS.

Mesures d'utilisation des ressources

Surveillez les indicateurs d'utilisation des ressources pour vérifier que vos ressources sont correctement consommées. Ces indicateurs vous aident à identifier les goulets d'étranglement et les causes profondes des problèmes de performance.

- Processeur, mémoire, réseau, puissance du processeur graphique et température du processeur graphique : surveillez ces indicateurs pour vous assurer que les ressources allouées répondent aux demandes de charge de travail et identifier les opportunités d'optimisation. Suivez des indicateurs tels que gpu_memory_usage_bytes pour identifier les modèles de consommation de mémoire et détecter les pics d'utilisation. Calculez des percentiles tels que le 95e percentile (P95) pour comprendre les exigences de mémoire les plus élevées pendant l'entraînement. Cette analyse vous aide à optimiser les modèles et l'infrastructure afin d'éviter les erreurs OOM et de réduire les coûts.
- Occupation du SM, FPxx activité du SM, activité : surveillez ces métriques pour comprendre comment la ressource sous-jacente du GPU est utilisée. Objectif 0,8 pour l'activité SM en règle générale.
- Utilisation des ressources des nœuds et des pods : suivez l'utilisation des ressources au niveau du nœud et du pod afin d'identifier les problèmes de contention des ressources et les goulots d'étranglement potentiels. Vérifiez si les nœuds approchent des limites de capacité, ce qui peut retarder la planification des modules et ralentir les tâches de formation.
- Utilisation des ressources par rapport aux demandes et aux limites : comparez l'utilisation réelle des ressources aux demandes et limites configurées afin de déterminer si votre cluster peut gérer les charges de travail actuelles et les charges de travail futures. Cette comparaison indique si

vous devez ajuster les allocations de ressources pour éviter les erreurs OOM ou le gaspillage de ressources.

- Métriques internes à partir de frameworks ML - Capturez les métriques de formation et de convergence internes à partir de frameworks de ML tels que TensorFlow et PyTorch. Ces indicateurs incluent les courbes de perte, le taux d'apprentissage, le temps de traitement par lots et la durée des étapes de formation. Visualisez ces indicateurs à l'aide d'outils TensorBoard ou similaires pour suivre la convergence des modèles et identifier les inefficiences en matière de formation.

Mesures de performance du modèle

Surveillez les indicateurs de performance des modèles pour vérifier que votre processus de formation produit des modèles qui répondent aux exigences de précision et de gestion. Ces indicateurs vous aident à déterminer quand arrêter l'entraînement, à comparer les versions des modèles et à identifier la dégradation des performances.

- Exactitude, précision, rappel et score F1 : suivez ces indicateurs pour comprendre les performances de votre modèle sur les données de validation. Calculez le score F1 sur un ensemble de validation après chaque période d'entraînement pour évaluer si le modèle s'améliore et quand il atteint des niveaux de performance acceptables.
- Mesures spécifiques à l'entreprise et KPIs — Définissez et suivez les mesures qui mesurent directement la valeur commerciale de vos AI/ML initiatives. Pour un système de recommandation, suivez des indicateurs tels que le taux de clics ou le taux de conversion pour vous assurer que le modèle génère les résultats commerciaux escomptés.
- Performances au fil du temps : comparez les indicateurs de performance entre les versions des modèles et les cycles d'entraînement pour identifier les tendances et détecter les dégradations. Vérifiez si les nouvelles versions des modèles maintiennent ou améliorent les performances par rapport aux modèles de base. Cette comparaison historique vous permet de décider s'il convient de déployer de nouveaux modèles ou d'étudier les problèmes de formation.

Qualité des données et mesures de dérive

Surveillez la qualité des données et les mesures de dérive pour vous assurer que vos données d'entraînement restent cohérentes et représentatives. La dérive des données peut entraîner une dégradation des performances des modèles au fil du temps, tandis que les problèmes de qualité des données peuvent empêcher les modèles de converger ou produire des résultats peu fiables.

- Propriétés statistiques des données d'entrée : suivez les propriétés statistiques telles que la moyenne, l'écart type et la distribution des entités en entrée dans le temps afin de détecter les dérives ou les anomalies des données. Vérifiez si la distribution des fonctionnalités change de manière significative par rapport à vos données d'entraînement de base. Par exemple, si la moyenne d'une caractéristique critique change de plus de deux écarts types, déterminez si votre pipeline de données a changé ou si la source de données sous-jacente a changé.
- Détection de la dérive des données et alertes — Mettez en œuvre des mécanismes automatisés pour détecter les problèmes de qualité des données et les signaler avant qu'ils n'aient une incidence sur la formation. Utilisez des tests statistiques tels que le test de Kolmogorov-Smirnov ou le test du Khi carré pour comparer les distributions de données actuelles avec vos données d'entraînement d'origine. Configurez des alertes lorsque les tests détectent une dérive significative afin de réentraîner les modèles avec des données mises à jour ou d'étudier les problèmes liés au pipeline de données.

Mesures de latence et de débit

Surveillez les indicateurs de latence et de débit pour identifier les goulots d'étranglement dans votre pipeline de formation et optimiser l'utilisation des ressources. Ces indicateurs vous aident à comprendre où le temps est consacré pendant la formation et sur quoi concentrer les efforts d'optimisation.

- End-to-End Latence des pipelines de formation ML : mesurez le temps total pendant lequel les données circulent dans l'ensemble de votre pipeline de formation, de l'ingestion des données à la mise à jour du modèle. Suivez cette métrique d'un cycle d'entraînement à l'autre pour déterminer si les modifications apportées au pipeline améliorent ou dégradent les performances. Une latence élevée indique souvent des goulots d'étranglement dans le chargement des données, le prétraitement ou la communication réseau entre les nœuds.
- Débit de formation et taux de traitement : suivez le volume de données que votre pipeline de formation traite par unité de temps afin de garantir une utilisation efficace des ressources. Surveillez les indicateurs tels que les échantillons traités par seconde ou les lots terminés par minute. Un faible débit par rapport à la capacité de votre matériel indique que le chargement des données, le prétraitement ou le calcul des modèles sont inefficaces et gâchent les cycles du GPU.
- Latence de sauvegarde et de restauration des points de contrôle : surveillez le temps nécessaire pour enregistrer les points de contrôle du modèle dans le stockage (S3, EFS FSx) et restaurez-les dans la mémoire du GPU ou du processeur lors de la reprise des tâches ou de la reprise après une panne. Les opérations lentes aux points de contrôle prolongent le temps de reprise des tâches

et augmentent les coûts. Suivez la taille du point de contrôle, la durée des sauvegardes, la durée de restauration et le nombre d'échecs afin d'identifier les opportunités d'optimisation telles que la compression ou l'accélération des niveaux de stockage.

- Temps de chargement et de prétraitement des données : mesurez le temps passé à charger les données depuis le stockage et à appliquer des transformations de prétraitement. Comparez ce temps au temps de calcul du modèle pour déterminer si votre entraînement est lié aux données ou au calcul. Si le chargement des données occupe plus de 20 % du temps total de formation, envisagez d'optimiser votre pipeline de données avec la mise en cache, le préchargement ou un stockage plus rapide.

Taux d'erreur et défaillances

Surveillez les taux d'erreur et les défaillances tout au long de votre pipeline de formation afin de maintenir la fiabilité et d'éviter le gaspillage de ressources informatiques. Les erreurs non détectées peuvent entraîner l'échec silencieux des tâches d'entraînement, produire des modèles non valides ou faire perdre des heures de temps au GPU avant que vous ne remarquiez de problèmes.

- Surveillance des erreurs de pipeline — Suivez les erreurs à toutes les étapes de votre pipeline de ML, y compris le prétraitement des données, la formation des modèles et les opérations aux points de contrôle. Enregistrez les types d'erreurs, les fréquences et les composants concernés pour identifier rapidement les problèmes. Les erreurs courantes incluent les incohérences entre les formats de données, out-of-memory les échecs lors du prétraitement et les échecs de sauvegarde des points de contrôle dus aux limites de stockage. Configurez des alertes lorsque les taux d'erreur dépassent les seuils de référence afin de pouvoir enquêter avant que les erreurs ne se répercutent en cascade.
- Analyse des erreurs récurrentes — Identifiez et étudiez les modèles d'erreurs récurrentes afin de prévenir les défaillances futures et d'améliorer la fiabilité du pipeline. Analysez les journaux pour déterminer si des échantillons de données, des tailles de lots ou des configurations de formation spécifiques sont systématiquement à l'origine de défaillances. Par exemple, si certains types de données d'entrée déclenchent des erreurs de prétraitement, ajoutez des contrôles de validation plus tôt dans le pipeline ou mettez à jour votre logique de nettoyage des données. Suivez le temps moyen entre les défaillances (MTBF) pour déterminer si la fiabilité de votre pipeline s'améliore au fil du temps. »

Métriques spécifiques à Kubernetes et EKS

Surveillez les indicateurs Kubernetes et EKS pour vous assurer que votre infrastructure de cluster reste saine et qu'elle peut prendre en charge vos charges de travail de formation. Ces indicateurs vous aident à détecter les problèmes d'infrastructure avant qu'ils n'entraînent des échecs de formation ou une dégradation des performances.

- Mesures de l'état du cluster Kubernetes : surveillez l'état et l'état des objets Kubernetes, notamment les pods, les nœuds, les déploiements et les services. Suivez l'état des pods pour identifier les pods bloqués dans des états de boucle en attente, en panne ou en panne. Surveillez l'état des nœuds pour détecter les problèmes tels que la pression du disque, la pression de la mémoire ou l'indisponibilité du réseau. Utilisez kubectl ou des outils de surveillance pour vérifier ces indicateurs en permanence et configurer des alertes lorsque les pods ne démarrent pas ou que les nœuds deviennent impossibles à planifier.
- Mesures d'exécution du pipeline de formation — Suivez les cycles réussis et échoués du pipeline, la durée des tâches, le temps d'achèvement des étapes et les erreurs d'orchestration. Vérifiez si les tâches de formation sont achevées dans les délais prévus et si les taux d'échec augmentent au fil du temps. Suivez des indicateurs tels que le taux de réussite, la durée moyenne des tâches et le délai avant l'échec. Ces indicateurs vous aident à déterminer si des problèmes d'infrastructure, de configuration ou de qualité des données sont à l'origine d'échecs de formation.
- Mesures des services AWS : suivez les métriques des services AWS qui prennent en charge votre infrastructure EKS et vos charges de travail de formation. Surveillez les métriques S3 telles que la latence des demandes, les taux d'erreur et le débit pour garantir la cohérence des performances de chargement des données. Suivez les indicateurs de volume EBS, notamment les IOPS, le débit et la longueur des files d'attente, afin de détecter les goulots d'étranglement liés au stockage. Surveillez les journaux de flux VPC et les métriques du réseau pour identifier les problèmes de connectivité entre les nœuds ou avec les services externes.
- Mesures du plan de contrôle Kubernetes : surveillez le serveur d'API, le planificateur, le gestionnaire de contrôleurs et la base de données etcd pour détecter les problèmes de performances ou les défaillances qui affectent les opérations du cluster. Suivez la latence des demandes du serveur API, le taux de demandes et le taux d'erreur pour garantir que le plan de contrôle répond rapidement aux demandes de planification. Surveillez la taille de la base de données etcd, la durée des validations et les changements de leader pour détecter les problèmes de stabilité. La latence élevée du serveur API ou les changements fréquents de leader ETCD peuvent retarder la planification des modules et prolonger les délais de démarrage des tâches de formation.

Journaux des applications et des instances

Collectez et analysez les journaux des applications et des instances pour diagnostiquer les problèmes que les indicateurs ne peuvent expliquer à eux seuls. Les journaux fournissent un contexte détaillé sur les erreurs, les changements d'état et les événements du système qui vous aident à comprendre pourquoi les tâches de formation échouent ou fonctionnent mal. La corrélation entre les journaux et les métriques vous permet d'identifier les causes profondes plus rapidement.

- Journaux des applications : collectez les journaux des applications provenant de vos tâches de formation, de vos pipelines de données et de vos frameworks de machine learning afin d'identifier les goulots d'étranglement et de diagnostiquer les défaillances. Ces journaux capturent des informations détaillées sur l'exécution des tâches, notamment les erreurs de chargement des données, les échecs d'initialisation du modèle, les erreurs de sauvegarde des points de contrôle et les avertissements spécifiques au framework. Corrélez les horodatages des journaux avec les pics métriques pour comprendre les causes de la dégradation des performances ou des défaillances. Par exemple, si l'utilisation du GPU baisse soudainement, consultez les journaux des applications pour détecter les erreurs indiquant des blocages du pipeline de données ou des échecs de prétraitement. Utilisez des outils de journalisation centralisés tels que CloudWatch Logs ou Fluent Bit pour agréger les journaux de tous les pods et les rendre consultables.
- Journaux d'instance : collectez des journaux au niveau de l'instance, tels que les journaux du système et les résultats dmesg, pour détecter les problèmes matériels et les problèmes au niveau du noyau. Ces journaux révèlent des problèmes tels que des erreurs de pilote GPU, des défaillances d'allocation de mémoire, I/O des erreurs de disque et des problèmes d'interface réseau qui peuvent ne pas apparaître dans les journaux des applications. Corrélez les journaux d'instance avec les journaux et les métriques des applications pour déterminer si les échecs de formation sont dus à des problèmes matériels ou à des problèmes d'application. Par exemple, si une tâche de formation échoue avec une out-of-memory erreur, consultez les journaux dmesg pour détecter les messages du noyau OOM Killer indiquant si le système n'a plus de mémoire ou si l'application a dépassé ses limites de conteneur. Configurez des alertes pour les erreurs matérielles critiques, telles que les erreurs XID du GPU ou les pannes de disque, afin de pouvoir remplacer les instances défaillantes avant qu'elles n'entraînent des interruptions d'entraînement généralisées.

Les sections suivantes expliquent comment collecter les métriques décrites ci-dessus à l'aide de deux approches recommandées par AWS : CloudWatch Container Insights et Amazon Managed Prometheus Amazon Managed Prometheus with Amazon Managed Grafana. Choisissez CloudWatch Container Insights si vous préférez les outils natifs d'AWS dotés d'une configuration minimale et de tableaux de bord prédéfinis. Choisissez Amazon Managed Prometheus avec Amazon Managed

Grafana si vous avez besoin de tableaux de bord personnalisés, de fonctionnalités de visualisation avancées ou si vous souhaitez intégrer une infrastructure de surveillance existante basée sur Prometheus. Pour obtenir la liste complète des métriques Container Insights disponibles, consultez les métriques Amazon EKS et Kubernetes Container Insights.

Envisagez de surveiller les mesures d'inférence en ligne en temps réel

Dans les systèmes en temps réel, une faible latence est essentielle pour fournir des réponses rapides aux utilisateurs ou aux autres systèmes dépendants. Une latence élevée peut dégrader l'expérience utilisateur ou enfreindre les exigences de performance. Les composants qui influencent la latence d'inférence incluent le temps de chargement du modèle, le temps de prétraitement, le temps de prédiction réel, le temps de post-traitement et le temps de transmission du réseau. Nous recommandons de surveiller la latence d'inférence pour garantir des réponses à faible latence conformes aux accords de niveau de service (SLAs) et de développer des métriques personnalisées pour les éléments suivants. Effectuez des tests en fonction de la charge attendue, incluez la latence du réseau, prenez en compte les demandes simultanées et testez avec des lots de différentes tailles.

- Délai jusqu'au premier jeton (TTFT) : délai entre le moment où un utilisateur soumet une demande et celui où il reçoit le début de la réponse (le premier mot, jeton ou fragment). Par exemple, dans les chatbots, vous pouvez vérifier combien de temps il faut pour générer le premier résultat (jeton) une fois que l'utilisateur a posé une question.
- End-to-End Latence — Il s'agit du temps total entre le moment où une demande est reçue et le moment où la réponse est renvoyée. Par exemple, mesurez le délai entre la demande et la réponse.
- Jetons de sortie par seconde (TPS) : indique la rapidité avec laquelle votre modèle génère de nouveaux jetons une fois qu'il commence à répondre. Par exemple, dans les chatbots, vous pouvez suivre la vitesse de génération des modèles linguistiques pour un texte de référence.
- Taux d'erreur : suit les demandes ayant échoué, ce qui peut indiquer des problèmes de performances. Par exemple, surveillez les demandes infructueuses concernant des documents volumineux ou certains caractères.
- Débit : mesurez le nombre de demandes ou d'opérations que le système peut traiter par unité de temps. Par exemple, suivez les demandes par seconde pour gérer les pics de charge.

K/V (Key/Value) cache can be a powerful optimization technique for inference latency, particularly relevant for transformer-based models. K/V cache stores the key and value tensors from previous transformer layer computations, reducing redundant computations during autoregressive inference,

particularly in large language models (LLMs). Cache Efficiency Metrics (specifically for K/Vou un cache de session (utilisation) :

- hit/miss Taux de cache — Pour les configurations d'inférence utilisant la mise en cache (K/V ou intégration de caches), mesurez la fréquence à laquelle le cache est utile. Les faibles taux de réussite peuvent indiquer des modifications de la configuration du cache ou de la charge de travail sous-optimales, deux facteurs susceptibles d'augmenter la latence.

Dans les rubriques suivantes, nous démontrons la collecte de données pour certains des indicateurs mentionnés ci-dessus. Nous fournirons des exemples des deux approches recommandées par AWS : AWS Native CloudWatch Container Insights et Amazon Managed Prometheus open source avec Amazon Managed Grafana. Vous choisiriez l'une de ces solutions en fonction de vos besoins globaux en matière d'observabilité. Consultez les métriques Amazon EKS et Kubernetes Container Insights pour obtenir la liste complète des métriques Container Insights.

Suivi de l'utilisation de la mémoire GPU

Comme indiqué dans la the section called “Envisagez de surveiller la formation de base et de peaufiner les indicateurs” rubrique, l'utilisation de la mémoire du GPU est essentielle pour éviter les erreurs out-of-memory (OOM) et garantir une utilisation efficace des ressources. Les exemples suivants montrent comment instrumenter votre application d'entraînement pour exposer une métrique d'histogramme personnalisée et calculer l'utilisation de la mémoire P95 pour identifier les pics de consommation. `gpu_memory_usage_bytes` Veillez à effectuer le test avec un exemple de tâche de formation (par exemple, peaufiner un modèle de transformateur) dans un environnement de test.

Exemple d'AWS CloudWatch Container Insights

Cet exemple montre comment instrumenter votre application de formation pour l'exposer `gpu_memory_usage_bytes` sous forme d'histogramme en utilisant l'approche native d'AWS. Notez que votre AI/ML conteneur doit être configuré pour émettre des journaux structurés au format EMF (CloudWatch Embedded Metrics Format). CloudWatch logs analyse les données EMF et publie les métriques. Utilisez aws_embedded_metrics dans votre application d'entraînement pour envoyer des journaux structurés au format EMF à Logs, qui extrait les métriques du GPU CloudWatch .

```
from aws_embedded_metrics import metric_scope
import torch
import numpy as np

memory_usage = []
```

```
@metric_scope
def log_gpu_memory(metrics):
    # Record current GPU memory usage
    mem = torch.cuda.memory_allocated()
    memory_usage.append(mem)

    # Log as histogram metric
    metrics.set_namespace("MLTraining/GPUMemory")
    metrics.put_metric("gpu_memory_usage_bytes", mem, "Bytes", "Histogram")

    # Calculate and log P95 if we have enough data points
    if len(memory_usage) >= 10:
        p95 = np.percentile(memory_usage, 95)
        metrics.put_metric("gpu_memory_p95_bytes", p95, "Bytes")
        print(f"Current memory: {mem} bytes, P95: {p95} bytes")

# Example usage in training loop
for epoch in range(20):
    # Your model training code would go here
    log_gpu_memory()
```

Exemple de Prometheus et Grafana

Cet exemple montre comment instrumenter votre application d'entraînement pour l'exposer `gpu_memory_usage_bytes` sous forme d'histogramme à l'aide de la bibliothèque cliente Prometheus en Python.

```
from prometheus_client import Histogram
from prometheus_client import start_http_server
import pynvml

start_http_server(8080)
memory_usage = Histogram(
    'gpu_memory_usage_bytes',
    'GPU memory usage during training',
    ['gpu_index'],
    buckets=[1e9, 2e9, 4e9, 8e9, 16e9, 32e9]
)

# Function to get GPU memory usage
def get_gpu_memory_usage():
    if torch.cuda.is_available():
```

```
        # Get the current GPU device
        device = torch.cuda.current_device()

        # Get memory usage in bytes
        memory_allocated = torch.cuda.memory_allocated(device)
        memory_reserved = torch.cuda.memory_reserved(device)

        # Total memory usage (allocated + reserved)
        total_memory = memory_allocated + memory_reserved

        return device, total_memory
    else:
        return None, 0

# Get GPU memory usage
gpu_index, memory_used = get_gpu_memory_usage()
```

Suivez la durée de la demande d'inférence pour une inférence en ligne en temps réel

Comme indiqué dans la the section called “Envisagez de surveiller la formation de base et de peaufiner les indicateurs” rubrique, une faible latence est essentielle pour fournir des réponses rapides aux utilisateurs ou aux autres systèmes dépendants. Les exemples suivants montrent comment suivre une métrique d'histogramme personnalisée`inference_request_duration_seconds`, exposée par votre application d'inférence. Calculez la latence du 95e percentile (P95) pour vous concentrer sur les pires scénarios, testez avec des demandes d'inférence synthétique (par exemple, via Locust) dans un environnement intermédiaire et définissez des seuils d'alerte (par exemple, > 500 ms) pour détecter les violations des SLA.

Exemple d'AWS CloudWatch Container Insights

Cet exemple montre comment créer une métrique d'histogramme personnalisée dans votre application d'inférence pour inference_request_duration_seconds à l'aide du format de métrique intégré AWS. CloudWatch

```
import boto3
import time
from aws_embedded_metrics import metric_scope, MetricsLogger

cloudwatch = boto3.client('cloudwatch')
```

```
@metric_scope
def log_inference_duration(metrics: MetricsLogger, duration: float):
    metrics.set_namespace("ML/Inference")
    metrics.put_metric("inference_request_duration_seconds", duration, "Seconds",
 "Histogram")
    metrics.set_property("Buckets", [0.1, 0.5, 1, 2, 5])

@metric_scope
def process_inference_request(metrics: MetricsLogger):
    start_time = time.time()

    # Your inference processing code here
    # For example:
    # result = model.predict(input_data)

    duration = time.time() - start_time
    log_inference_duration(metrics, duration)

    print(f"Inference request processed in {duration} seconds")

# Example usage
process_inference_request()
```

Exemple de Prometheus et Grafana

Cet exemple montre comment créer une métrique d'histogramme personnalisée dans votre application d'inférence pour inference_request_duration_seconds à l'aide de la bibliothèque cliente Prometheus en Python :

```
from prometheus_client import Histogram
from prometheus_client import start_http_server
import time

start_http_server(8080)
request_duration = Histogram(
    'inference_request_duration_seconds',
    'Inference request latency',
    buckets=[0.1, 0.5, 1, 2, 5]
)
start_time = time.time()
# Process inference request
request_duration.observe(time.time() - start_time)
```

Dans Grafana, utilisez la requête `histogram_quantile(0.95, sum(rate(inference_request_duration_seconds_bucket[5m])) by (le, pod))` pour visualiser les tendances de latence du P95. Pour en savoir plus, consultez la documentation sur l'histogramme de Prometheus et la documentation sur le client de Prometheus.

Suivez le débit des jetons pour une inférence en ligne en temps réel

Comme indiqué dans la the section called "Envisagez de surveiller la formation de base et de peaufiner les indicateurs" rubrique, nous recommandons de surveiller le temps de traitement des jetons afin d'évaluer les performances du modèle et d'optimiser les décisions de dimensionnement. Les exemples suivants montrent comment suivre une métrique d'histogramme personnalisée`token_processing_duration_seconds`, exposée par votre application d'inférence. Calculez la durée du 95e percentile (P95) pour analyser l'efficacité du traitement, effectuez des tests avec des charges de demandes simulées (par exemple, 100 à 1 000 demandes/seconde) dans un cluster hors production et ajustez les déclencheurs KEDA pour optimiser la mise à l'échelle.

Exemple d'AWS CloudWatch Container Insights

Cet exemple montre comment créer une métrique d'histogramme personnalisée dans votre application d'inférence pour token_processing_duration_seconds à l'aide du format de métrique intégré AWS. CloudWatch Elle utilise des dimensions `() with a custom` `get_duration_bucket`fonction `set_dimension) pour classer les durées en compartiments (par exemple, « 0.01", « >1").

```
import boto3
import time
from aws_embedded_metrics import metric_scope, MetricsLogger

cloudwatch = boto3.client('cloudwatch')

@metric_scope
def log_token_processing(metrics: MetricsLogger, duration: float, token_count: int):
    metrics.set_namespace("ML/TokenProcessing")
    metrics.put_metric("token_processing_duration_seconds", duration, "Seconds")
    metrics.set_dimension("ProcessingBucket", get_duration_bucket(duration))
    metrics.set_property("TokenCount", token_count)

def get_duration_bucket(duration):
    buckets = [0.01, 0.05, 0.1, 0.5, 1]
    for bucket in buckets:
```

```
        if duration <= bucket:
            return f"<={bucket}"
    return f">{buckets[-1]}"

@metric_scope
def process_tokens(input_text: str, model, tokenizer, metrics: MetricsLogger):
    tokens = tokenizer.encode(input_text)
    token_count = len(tokens)

    start_time = time.time()
    # Process tokens (replace with your actual processing logic)
    output = model(tokens)
    duration = time.time() - start_time

    log_token_processing(metrics, duration, token_count)
    print(f"Processed {token_count} tokens in {duration} seconds")
    return output
```

Exemple de Prometheus et Grafana

Cet exemple montre comment créer une métrique d'histogramme personnalisée dans votre application d'inférence pour token_processing_duration_seconds à l'aide de la bibliothèque cliente Prometheus en Python.

```
from prometheus_client import Histogram
from prometheus_client import start_http_server
import time

start_http_server(8080)
token_duration = Histogram(
    'token_processing_duration_seconds',
    'Token processing time per request',
    buckets=[0.01, 0.05, 0.1, 0.5, 1]
)
start_time = time.time()
# Process tokens
token_duration.observe(time.time() - start_time)
```

Dans Grafana, utilisez la requête `histogram_quantile(0.95, sum(rate(token_processing_duration_seconds_bucket[5m])) by (le, pod))`` pour visualiser les tendances du temps de traitement du P95. Pour en savoir plus, consultez la documentation sur l'histogramme de Prometheus et la documentation sur le client de Prometheus.

Mesurer la latence de restauration des points de contrôle

Comme indiqué dans la the section called “Envisagez de surveiller la formation de base et de peaufiner les indicateurs” rubrique, la latence des points de contrôle est une mesure critique au cours des différentes phases du cycle de vie du modèle. Les exemples suivants montrent comment suivre une métrique d'histogramme personnalisée`checkpoint_restore_duration_seconds`, exposée par votre application. Calculez la durée du 95e percentile (P95) pour surveiller les performances de restauration, effectuez des tests avec Spot interruptions dans un cluster hors production et définissez des seuils d'alerte (par exemple, <30 secondes) pour détecter les retards.

Exemple d'AWS CloudWatch Container Insights

Cet exemple montre comment instrumenter votre application par lots pour exposer checkpoint_restore_duration_seconds sous forme d'histogramme à l'aide d'Insights : CloudWatch

```
import boto3
import time
import torch
from aws_embedded_metrics import metric_scope, MetricsLogger

@metric_scope
def log_checkpoint_restore(metrics: MetricsLogger, duration: float):
    metrics.set_namespace("ML/ModelOperations")
    metrics.put_metric("checkpoint_restore_duration_seconds", duration, "Seconds",
 "Histogram")
    metrics.set_property("Buckets", [1, 5, 10, 30, 60])
    metrics.set_property("CheckpointSource", "s3://my-bucket/checkpoint.pt")

@metric_scope
def load_checkpoint(model, checkpoint_path: str, metrics: MetricsLogger):
    start_time = time.time()

    # Load model checkpoint
    model.load_state_dict(torch.load(checkpoint_path))

    duration = time.time() - start_time
    log_checkpoint_restore(metrics, duration)

    print(f"Checkpoint restored in {duration} seconds")
```

Exemple de Prometheus et Grafana

Cet exemple montre comment instrumenter votre application batch pour l'exposer `checkpoint_restore_duration_seconds` sous forme d'histogramme à l'aide de la bibliothèque cliente Prometheus en Python :

```
from prometheus_client import Histogram
from prometheus_client import start_http_server
import torch

start_http_server(8080)
restore_duration = Histogram(
    'checkpoint_restore_duration_seconds',
    'Time to restore checkpoint',
    buckets=[1, 5, 10, 30, 60]
)
with restore_duration.time():
    model.load_state_dict(torch.load("s3://my-bucket/checkpoint.pt"))
```

Dans Grafana, utilisez la requête `histogram_quantile(0.95, sum(rate(checkpoint_restore_duration_seconds_bucket[5m]) by (le))` pour visualiser les tendances de latence de restauration du P95. Pour en savoir plus, consultez la documentation sur l'histogramme de Prometheus et la documentation sur le client de Prometheus.

Mise à l'échelle et performances des applications

> **Tip**
>
> Découvrez les meilleures pratiques grâce aux ateliers Amazon EKS.

Gestion des artefacts ML, des frameworks de service et optimisation du démarrage

Le déploiement de modèles d'apprentissage automatique (ML) sur Amazon EKS nécessite une réflexion approfondie sur la manière dont les modèles sont intégrés aux images de conteneur et aux environnements d'exécution. Cela garantit l'évolutivité, la reproductibilité et l'utilisation efficace des ressources. Cette rubrique décrit les différentes approches de gestion des artefacts du modèle ML, de sélection de frameworks de service et d'optimisation des temps de démarrage des conteneurs grâce à des techniques telles que la pré-mise en cache, toutes conçues pour réduire les temps de démarrage des conteneurs.

Gestion des artefacts du modèle ML dans les déploiements

L'une des décisions clés est de savoir comment gérer les artefacts du modèle ML (tels que les poids et les configurations) eux-mêmes. Le choix a un impact sur la taille de l'image, la vitesse de déploiement, la fréquence de mise à jour du modèle et la surcharge opérationnelle. Notez que lorsque nous parlons de stockage du « modèle », nous faisons référence aux artefacts du modèle (tels que les paramètres entraînés et les poids du modèle). Il existe différentes approches pour gérer les artefacts du modèle ML sur Amazon EKS. Chacun a ses inconvénients, et le meilleur dépend de la taille de votre modèle, de la cadence de mise à jour et des besoins en infrastructure. Envisagez les approches suivantes, de la moins recommandée à la plus recommandée :

- Intégration du modèle dans l'image du conteneur : copiez les fichiers du modèle (par exemple, .safetensors, .pth, .h5) dans l'image du conteneur (par exemple, Dockerfile) lors de la création de l'image. Le modèle fait partie de l'image immuable. Nous recommandons d'utiliser cette approche pour les petits modèles avec des mises à jour peu fréquentes. Cela garantit la cohérence et la reproductibilité, évite les délais de chargement et simplifie la gestion des dépendances, mais entraîne des tailles d'image plus importantes, ralentit les builds et les pushs, nécessite une reconstruction et un redéploiement pour les mises à jour des modèles, et n'est pas idéal pour les grands modèles en raison du débit d'extraction des registres.
- Téléchargement du modèle lors de l'exécution : au démarrage du conteneur, l'application télécharge le modèle depuis un stockage externe (par exemple, Amazon S3, soutenu par S3 CRT pour des transferts haut débit optimisés à l'aide de méthodes telles que Mountpoint pour le pilote CSI S3, AWS S3 CLI ou s5cmd OSS CLI) via des scripts dans un conteneur d'initialisation ou un point d'entrée. Nous recommandons de commencer par cette approche pour les grands modèles soumis à des mises à jour fréquentes. Cela permet aux images de conteneur de se concentrer sur le code/l'exécution, permet de mettre à jour facilement les modèles sans avoir à les reconstruire, prend en charge le contrôle des versions via les métadonnées de stockage, mais cela entraîne des défaillances réseau potentielles (nécessite une logique de nouvelle tentative), nécessite une authentification et une mise en cache.

Pour en savoir plus, consultez la section Accélérer le processus d'extraction dans le cadre de l'atelier AI on EKS.

Au service des modèles ML

Le déploiement et le service de modèles d'apprentissage automatique (ML) sur Amazon EKS nécessitent de sélectionner une approche de diffusion de modèles appropriée afin d'optimiser

la latence, le débit, l'évolutivité et la simplicité opérationnelle. Le choix dépend de votre type de modèle (par exemple, langage, modèle de vision), des exigences en matière de charge de travail (par exemple, inférence en temps réel) et de l'expertise de votre équipe. Les approches courantes incluent des configurations basées sur Python pour le prototypage, des serveurs de modèles dédiés pour les fonctionnalités de production et des moteurs d'inférence spécialisés pour des performances et une efficacité élevées. Chaque méthode implique des compromis en termes de complexité de configuration, de performances et d'utilisation des ressources. Notez que les frameworks de distribution peuvent augmenter la taille (multiple GBs) des images de conteneur en raison de dépendances, ce qui peut avoir un impact sur les temps de démarrage. Envisagez le découplage à l'aide de techniques de gestion des artefacts pour atténuer ce problème. Les options sont répertoriées de la moins recommandée à la plus recommandée :

À l'aide de frameworks Python (par exemple, FastAPI, HuggingFace Transformers with PyTorch)
Développez une application personnalisée à l'aide de frameworks Python, en intégrant des fichiers de modèle (poids, configuration, tokenizer) dans une configuration de nœud conteneurisé.

- Avantages : prototypage facile, Python uniquement sans infrastructure supplémentaire, compatible avec tous les HuggingFace modèles, déploiement simple de Kubernetes.
- Inconvénients : se limite au traitement par request/simple lots uniques, génération lente de jetons (pas de noyaux optimisés), mémoire inefficace, manque de scalabilité et de surveillance et implique de longs temps de démarrage.
- Recommandation : à utiliser pour le prototypage initial ou pour les tâches à nœud unique nécessitant une intégration logique personnalisée.

Utilisation de frameworks de service de modèles dédiés (par exemple, TensorRT-LLM, TGI)
Adoptez des serveurs spécialisés tels que TensorRT-LLM ou TGI pour l'inférence ML, la gestion du chargement, du routage et de l'optimisation des modèles. Ils prennent en charge des formats tels que les capteurs de sécurité, avec une compilation optionnelle ou des plugins.

- Avantages : Propose le traitement par lots (static/in-flight or continuous), quantization (INT8, FP8, GPTQ), hardware optimizations (NVIDIA, AMD, Intel, Inferentia), and multi-GPU support (Tensor/Pipelineparallélisme). TensorRT-LLM prend en charge divers modèles (encodeur-décodeur)LLMs, tandis que TGI tire parti de l'intégration. HuggingFace
- Inconvénients : TensorRT-LLM doit être compilé et n'est disponible que sur NVIDIA ; TGI peut être moins efficace pour le traitement par lots ; les deux ajoutent une charge de configuration et peuvent ne pas convenir à tous les types de modèles (par exemple, les modèles autres que les transformateurs).

- Recommandation : convient aux PyTorch/TensorFlow modèles nécessitant des capacités de production telles que A/B des tests ou un débit élevé avec du matériel compatible.

Utilisation de moteurs d'inférence spécialisés à haut débit (par exemple, vLLM) Utilisez des moteurs d'inférence avancés tels que vLLM, qui optimisent le service LLM, le traitement par lots en vol et la quantification (, -KV PagedAttention, AWQ), intégrés à la mise à l'échelle automatique d'EKS. INT8 FP8

- Avantages : haut débit et efficacité de la mémoire (40 à 60 % d'économies de VRAM), gestion dynamique des demandes, diffusion de jetons, prise en charge de plusieurs processeurs graphiques Tensor Parallel à nœud unique et large compatibilité matérielle.
- Inconvénients : optimisé pour les transformateurs à décodeur uniquement (par exemple, LLa MA), moins efficace pour les modèles sans transformateur, nécessite du matériel compatible (par exemple, NVIDIA GPUs) et des efforts de configuration.
- Recommandation : le meilleur choix pour l'inférence LLM à volume élevé et à faible latence sur EKS, afin d'optimiser l'évolutivité et les performances.

Optimisation des temps d'extraction des images des conteneurs

Les images de conteneurs de grande taille peuvent entraîner des retards de démarrage à froid qui ont un impact sur la latence de démarrage du pod. Pour les charges de travail sensibles à la latence, telles que les charges de travail d'inférence en temps réel redimensionnées horizontalement, le démarrage rapide du pod est essentiel. Envisagez les approches suivantes pour optimiser les temps d'extraction des images de conteneurs :

Réduction de la taille des images des conteneurs

La réduction de la taille des images du conteneur au démarrage est un autre moyen de réduire la taille des images. Vous pouvez effectuer des réductions à chaque étape du processus de création de l'image du conteneur. Pour commencer, choisissez les images de base qui contiennent le moins de dépendances requises. Lors de la création d'images, n'incluez que les bibliothèques et artefacts essentiels requis. Lorsque vous créez des images, essayez de combiner plusieurs `COPY` commandes `RUN` ou plusieurs pour créer un plus petit nombre de couches plus grandes. Pour les AI/ML frameworks, utilisez des versions en plusieurs étapes pour séparer la construction et l'exécution, en copiant uniquement les artefacts requis (par exemple, via `COPY –from=` pour les registres ou les contextes locaux) et en sélectionnant des variantes telles que des images destinées à l'exécution uniquement (par exemple, `pytorch/pytorch:2.7.1-cuda11.8-cudnn9-runtime` à 3,03 Go

contre devel à 6,66 Go). Pour en savoir plus, consultez la section Réduction de la taille de l'image du conteneur dans l'atelier AI on EKS.

Utilisation du snapshotter SOCI pour pré-extraire des images

Pour les très grandes images difficiles à minimiser, vous pouvez utiliser le snapshotter open source Seekable OCI (SOCI) configuré en mode pull and unpack en parallèle. Cette solution vous permet d'utiliser des images existantes sans reconstruire ni modifier vos pipelines de génération. Cette option est particulièrement efficace lors du déploiement de charges de travail comportant de très grandes images sur des instances de calcul EC2 hautes performances. Il fonctionne parfaitement avec les réseaux à haut débit et les configurations de stockage hautes performances, comme c'est généralement le cas pour les charges de travail AI/ML évolutives.

Le pull/unpack mode parallèle SOCI améliore les performances end-to-end d'extraction d'images grâce à des stratégies de parallélisation configurables. L'accélération de l'extraction et de la préparation des images a un impact direct sur la rapidité avec laquelle vous pouvez déployer de nouvelles charges de travail et faire évoluer efficacement votre cluster. Les extractions d'images comportent deux phases principales :

1. Extraction de couches depuis le registre vers le nœud

   Pour optimiser l'extraction des couches, SOCI crée plusieurs connexions HTTP simultanées par couche, multipliant ainsi le débit de téléchargement au-delà de la limite de connexion unique. Il divise les grandes couches en morceaux et les télécharge simultanément sur plusieurs connexions. Cette approche permet de saturer la bande passante réseau disponible et de réduire considérablement les temps de téléchargement. Cela est particulièrement utile pour les AI/ML charges de travail où une seule couche peut atteindre plusieurs gigaoctets.

2. Déballage et préparation de ces couches pour créer des conteneurs

   Pour optimiser le déballage des couches, SOCI traite plusieurs couches simultanément. Au lieu d'attendre que chaque couche soit complètement déballée avant de commencer la suivante, il utilise les cœurs de processeur disponibles pour décompresser et extraire plusieurs couches simultanément. Ce traitement parallèle transforme la phase de déballage traditionnellement liée aux E/S en une opération optimisée pour le processeur qui évolue en fonction des cœurs disponibles. Le système orchestre soigneusement cette parallélisation afin de maintenir la cohérence du système de fichiers tout en maximisant le débit.

Le mode SOCI parallel pull utilise un système de contrôle à double seuil avec des paramètres configurables pour la simultanéité des téléchargements et le déballage du parallélisme. Ce contrôle granulaire vous permet d'affiner le comportement du SOCI pour répondre à vos exigences de performance et aux conditions environnementales spécifiques. La compréhension de ces paramètres vous permet d'optimiser votre temps d'exécution pour obtenir les meilleures performances d'extraction.

Références

- Pour plus d'informations sur la solution et les compromis de réglage, consultez la documentation des fonctionnalités dans le référentiel du projet SOCI sur. GitHub
- Pour un exemple pratique avec Karpenter sur Amazon EKS, consultez le plan de Karpenter utilisant le mode parallèle du snapshotter SOCI. pull/unpack
- Pour plus d'informations sur la configuration de Bottlerocket pour le tirage parallèle, voir soci-snapshotter Parallel Pull Unpack Mode dans la documentation de Bottlerocket.o

Utilisation des instantanés EBS pour pré-extraire des images

Vous pouvez prendre un instantané Amazon Elastic Block Store (EBS) des images de conteneurs mises en cache et le réutiliser pour les nœuds de travail EKS. Cela garantit que les images sont préextraites localement au démarrage du nœud, ce qui réduit le temps d'initialisation du module. Consultez Réduire le temps de démarrage des conteneurs sur Amazon EKS avec le volume de données Bottlerocket pour plus d'informations sur l'utilisation des plans Karpenter et EKS Terraform pour les groupes de nœuds gérés.

Pour en savoir plus, consultez les sections Utilisation de snapshotter containerd et Précharger des images de conteneurs dans des volumes de données Bottlerocket avec des instantanés EBS dans l'atelier AI on EKS.

Utilisation du cache d'exécution du conteneur pour pré-extraire des images

Vous pouvez pré-extraire des images de conteneur sur des nœuds à l'aide des ressources Kubernetes (par exemple, DaemonSet ou Deployment) pour remplir le cache d'exécution du conteneur du nœud. Le cache d'exécution du conteneur est le stockage local géré par le runtime du conteneur (par exemple, containerd) où les images sont stockées après avoir été extraites d'un registre. Le pré-extraction garantit la disponibilité des images localement, évitant ainsi les retards de téléchargement lors du démarrage du pod. Cette approche est particulièrement utile lorsque les

images changent souvent (mises à jour fréquentes, par exemple), lorsque les instantanés EBS ne sont pas préconfigurés, lorsque la création d'un volume EBS prend plus de temps que l'extraction directe depuis un registre de conteneurs, ou lorsque des nœuds se trouvent déjà dans le cluster et doivent créer des pods à la demande en utilisant l'une des nombreuses images possibles.

Le pré-extraction de toutes les variantes garantit un démarrage rapide, quelle que soit l'image requise. Par exemple, dans le cadre d'une charge de travail de ML massivement parallèle nécessitant 100 000 petits modèles conçus à l'aide de 10 techniques différentes, le pré-extraction de 10 images DaemonSet sur un grand cluster (par exemple, des milliers de nœuds) minimise le temps de démarrage du pod, permettant ainsi de terminer le processus en moins de 10 secondes en évitant les extractions à la demande. L'approche du cache d'exécution des conteneurs élimine le besoin de gérer les instantanés EBS et vous permet de toujours obtenir la dernière version des images de conteneur. Toutefois DaemonSets, pour les charges de travail d'inférence en temps réel dans lesquelles les nœuds redimensionnent les in/out, new nodes added by tools like Cluster Autoscaler may schedule workload pods before the pre-pull DaemonSet completes image pulling. This can cause the initial pod on the new node to trigger the pull anyway, potentially delaying startup and impacting low-latency requirements. Additionally, kubelet image garbage collection can affect pre-pulled images by removing unused ones when disk usage exceeds certain thresholds or if they exceed a configured maximum unused age. In scale-in/out modèles, cela peut entraîner l'expulsion des images sur les nœuds inactifs, ce qui nécessite des réextractions lors des mises à l'échelle ultérieures et réduit la fiabilité du cache pour les charges de travail surchargées.

Consultez le GitHub référentiel AWS pour des exemples de pré-extraction d'images dans le cache d'exécution du conteneur.

Pensez NVMe au stockage kubelet et containerd

Envisagez de configurer `kubelet` et `containerd` d'utiliser des disques de stockage d' NVMe instance éphémères pour améliorer les performances des disques. Le processus d'extraction d'un conteneur consiste à télécharger une image de conteneur à partir d'un registre et à décompresser ses couches dans un format utilisable. Pour optimiser les I/O opérations lors de la décompression, vous devez évaluer ce qui fournit des niveaux de I/O performance et de débit supérieurs pour le type d'instance de votre hôte de conteneur : les NVMe instances sauvegardées avec stockage local par rapport aux IOPS/débit IOPS/volume EBS. Pour les instances EC2 avec stockage NVMe local, envisagez de configurer le système de fichiers sous-jacent du nœud pour kubelet ()`/var/lib/kubelet`, containerd () et Pod logs (`/var/lib/containerd/var/log/pods`) afin d'utiliser des disques de stockage d' NVMe instance éphémères pour des niveaux de performance et de débit supérieurs. I/O

Le stockage éphémère du nœud peut être partagé entre les pods qui demandent un stockage éphémère et les images des conteneurs téléchargées sur le nœud. Si vous utilisez Karpenter avec Bottlerocket ou AL2023 EKS Optimized, AMIs cela peut être configuré en EC2NodeClassconfigurant le paramètre instanceStorePolicy sur RAID0ou, si vous utilisez des groupes de nœuds gérés, en le configurant localStoragePolicy dans le cadre des NodeConfigdonnées utilisateur.

Contribuez à ce guide

Tout le monde peut contribuer au guide des meilleures pratiques. Le guide des meilleures pratiques d'EKS est rédigé dans le AsciiDoc format suivant GitHub.

Résumé pour les contributeurs existants

- Ouvrez le `bpg-docs.code-workspace`avec VS Code pour installer automatiquement l' AsciiDoc extension.
  - Pour en savoir plus sur l'AsciiDoc extension, rendez-vous sur Visual Studio Marketplace.
- Les fichiers source du site Web AWS Docs sont stockés dans `latest/bpg`
- La syntaxe est très similaire à celle de Markdown.
  - Consultez la référence de syntaxe dans la AsciiDoctor documentation.
- La plateforme Docs se déploie `latest/bpg/images` uniquement. Chacune des sections du guide comporte un lien symbolique renvoyant vers ce répertoire. Par exemple, `latest/bpg/networking/images` pointe vers`latest/bpg/images`.

Configuration d'un environnement d'édition local

Si vous prévoyez de modifier fréquemment le guide, configurez un environnement d'édition local.

Forker et cloner le dépôt

Vous devez être familiarisé avec `gitgithub`, et les éditeurs de texte. Pour plus d'informations sur la prise en main de `git` et`github`, consultez la section Prise en main de votre GitHub compte dans la GitHub documentation.

1. Consultez le guide des meilleures pratiques d'EKS sur GitHub.
2. Créez un fork du dépôt du projet. Découvrez comment créer un dépôt dans la GitHub documentation.
3. Clonez votre fork du dépôt du projet. Découvrez comment cloner votre dépôt bifurqué.

Ouvrez l'espace de travail VS Code

AWS recommande d'utiliser le code Visual Studio de Microsoft pour modifier le guide. Pour plus d'informations sur VS Code, voir Télécharger Visual Studio Code et Commencer à utiliser Visual Studio Code dans la documentation de Visual Studio Code.

1. Ouvrez VS Code.
2. Ouvrez le `bpg-docs.code-workspace` fichier depuis le dépôt cloné.
3. Si c'est la première fois que vous ouvrez cet espace de travail, acceptez l'invite d'installation de l' AsciiDoc extension. Cette extension vérifie la syntaxe des AsciiDoc fichiers et génère un aperçu en direct.
4. Accédez au `latest/bpg` répertoire. Ce répertoire contient les fichiers source déployés sur le site de documentation AWS. Les fichiers sources sont organisés par section du guide, telle que « sécurité » ou « réseau ».

Modifier un fichier

1. Ouvrez un fichier dans l'éditeur.
   - Consultez la AsciiDoc syntaxe pour savoir comment créer des en-têtes, des liens et des listes.
   - Vous pouvez utiliser la syntaxe Markdown pour mettre en forme du texte, créer des listes et des en-têtes. Vous ne pouvez pas utiliser la syntaxe Markdown pour créer des liens.
2. Ouvrez un aperçu en direct de la page.
   - Appuyez d'abord sur `ctrl-k` ou `cmd-k` (selon le clavier). Ensuite, appuyez sur`v`. Cela ouvre un aperçu en mode partagé.

AWS suggère d'utiliser des branches de fonctionnalités pour organiser vos modifications. Apprenez à créer des branches avec git.

Soumettre une pull request

Vous pouvez créer une pull request depuis le GitHub site Web ou le GitHub cli.

Découvrez comment créer une pull request à partir d'un fork en utilisant le GitHub site Web.

Apprenez à créer une pull request à l'aide de l' GitHub interface de ligne de commande.

Utilisez l'éditeur Web github.dev

L'éditeur `github.dev` Web est basé sur VS Code. C'est un excellent moyen de modifier plusieurs fichiers et de prévisualiser le contenu sans aucune configuration.

Il prend en charge l' AsciiDoc extension. Vous pouvez effectuer des opérations git à l'aide de l'interface graphique. L'éditeur Web ne possède pas de shell ou de terminal pour exécuter des commandes.

Vous devez avoir un GitHub compte. Vous serez invité à vous connecter si nécessaire.

Lancez l'éditeur GitHub Web.

Modifier une seule page

Vous pouvez rapidement mettre à jour des pages individuelles en utilisant GitHub. Chaque page contient un lien « # Modifier cette page sur GitHub » en bas.

1. Accédez à la page de ce guide que vous souhaitez modifier
2. Cliquez sur le lien « Modifier cette page sur GitHub » en bas
3. Cliquez sur l'icône en forme de crayon d'édition en haut à droite de l'afficheur de GitHub fichiers, ou appuyez sur e
4. Modifier le fichier
5. Soumettez vos modifications à l'aide du bouton « Valider les modifications... ». Ce bouton crée une GitHub pull request. Les responsables du guide examineront cette pull request. Un réviseur approuvera la pull request ou demandera des modifications.

Afficher et définir l'ID d'une page

Cette page explique comment afficher et définir l'ID de page.

L'ID de page est une chaîne unique qui identifie chaque page du site de documentation. Vous pouvez consulter l'identifiant de page dans la barre d'adresse de votre navigateur lorsque vous vous trouvez sur une page spécifique. L'ID de page est utilisé pour l'URL, le nom du fichier et pour créer des liens de référence croisée.

Par exemple, si vous consultez cette page, l'URL dans la barre d'adresse de votre navigateur ressemblera à ce qui suit :

```
https://docs.aws.amazon.com/view-set-page-id.html
```

La dernière partie de l'URL (`view-set-page-id`) est l'ID de page.

Définissez l'ID de page

Lorsque vous créez une nouvelle page, vous devez définir l'ID de page dans le fichier source. L'ID de page doit être une chaîne concise avec un trait d'union décrivant le contenu de la page.

1. Ouvrez le fichier source de votre nouvelle page dans un éditeur de texte.
2. En haut du fichier, ajoutez la ligne suivante. Il doit se trouver au-dessus du premier titre.

   ```
   [#my-new-page]
   ```

   Remplacez `my-new-page` par l'ID de page de votre nouvelle page.
3. Enregistrez le fichier.

> **Note**
>
> La page IDs doit être unique sur l'ensemble du site de documentation. Si vous essayez d'utiliser un identifiant de page existant, une erreur de compilation s'affichera.

Création d’une nouvelle page

Découvrez comment créer une nouvelle page et mettre à jour la table des matières du guide.

Création de métadonnées de page

1. Déterminez le titre de la page et le titre abrégé de la page. Le titre abrégé de la page est facultatif, mais recommandé si le titre de la page comporte plus de quelques mots.
2. Déterminez l'ID de la page. Cela doit être unique dans le guide des meilleures pratiques d'EKS. La convention consiste à utiliser toutes les minuscules et à séparer les mots par. `-`
3. Créez un nouveau fichier asciidoc, dans un dossier si nécessaire, et ajoutez-y le texte suivant :

Example

[. » sujet "] [#<page-id>] = <page-title>:info_titleabbrev : < > page-short-title

Par exemple,

Example

[. » topic "] [#scalability] = Meilleures pratiques d'évolutivité d'EKS : info_titleabbrev : Scalabilité

Ajouter à la table des matières

1. Ouvrez le fichier de la page parent dans la table des matières. Pour les nouvelles sections du guide de haut niveau, le fichier parent est`book.adoc`.
2. Au bas du fichier parent, mettez à jour et insérez la directive suivante :

   Example

   inclure : <new-filename>[leveloffset=+1]

   Par exemple,

   Example

   include : :dataplane.adoc [leveloffset=+1]

Insérer une image

1. Trouvez le préfixe d'image de la page que vous êtes en train de modifier. Vérifiez la `:imagesdir:` propriété dans l'en-tête du fichier. À titre d'exemple, `` `:imagesdir: images/reliability/ ``
2. Placez votre image dans ce chemin, tel que `latest/bpg/images/reliability`
3. Déterminez le texte alternatif approprié pour votre image. Rédigez une brève description détaillée de l'image. Par exemple, « schéma d'un VPC avec trois zones de disponibilité » est un texte alternatif approprié.
4. Mettez à jour l'exemple suivant avec le texte alternatif et le nom de fichier de l'image. Insérez à l'endroit souhaité.

Example

image : <image-filename>[< image-alt-text >]

Par exemple,

Example

image : : eks-data-plane-connectivity .jpeg [Schéma de réseau]

Vérifiez le style avec Vale

1. Installez la CLI Vale.
2. Exécutez `vale sync`
3. Installez l'extension Vale depuis Visual Studio Marketplace.
4. Redémarrez VS Code et ouvrez un AsciiDoc fichier
5. VS Code souligne le texte problématique. Apprenez à gérer les erreurs et les avertissements dans la documentation de VS Code.

Création d'un aperçu local

1. Installez l'`asciidoctor`outil sous `brew` Linux ou macOS
   - Découvrez comment installer asciidoctor cli dans la documentation. AsciiDoctor
   - Découvrez comment installer le gestionnaire de paquets Brew.
2. Ouvrez un terminal, puis naviguez vers `latest/bpg/`
3. Exécutez `asciidoctor book.adoc`
   - Passez en revue les avertissements et les erreurs de syntaxe
4. Ouvrez le fichier `book.html` de sortie.
   - Sur macOS, vous pouvez `open book.html` exécuter l'aperçu dans votre navigateur par défaut pour ouvrir l'aperçu.

AsciiDoc Aide-mémoire

Formatage de base

```
*bold text*
_italic text_
`monospace text`
```

En-têtes

```
= Document Title (Header 1)
== Header 2
=== Header 3
==== Header 4
===== Header 5
====== Header 6
```

Lists

Listes non ordonnées :

```
- Item 1
- Item 2
-- Subitem 2.1
-- Subitem 2.2
- Item 3
```

Listes ordonnées :

```
. First item
. Second item
.. Subitem 2.1
.. Subitem 2.2
. Third item
```

Liens

```
External link:  https://example.com[Link text]
Internal link: <<page-id>>
```

```
Internal link: xref:page-id[Link text]
```

Images

```
image::image-file.jpg[Alt text]
```

Blocs de code

```
 [source,python]
 ----
 def hello_world():
     print("Hello, World!")
 ----
```

Tables

Apprenez à créer une table de base.

```
[cols="1,1"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
|===
```

Des admonestations

```
NOTE: This is a note admonition.

WARNING: This is a warning admonition.

TIP: This is a tip admonition.

IMPORTANT: This is an important admonition.
```

```
CAUTION: This is a caution admonition.
```

Aperçu :

> **Note**
>
> Il s'agit d'une mise en garde.

Inclut

```
include::filename.adoc[]
```

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.