Leitfaden für bewährte Verfahren

Amazon EKS



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

Amazon EKS: Leitfaden für bewährte Verfahren

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

Die Handelsmarken und Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Marken, die nicht Amazon gehören, sind Eigentum ihrer jeweiligen Inhaber, die möglicherweise mit Amazon verbunden sind, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Einführung 1
Verwandte Leitfäden 2
Beitragen 2
Sicherheit 3
Verwendung dieses Leitfadens 3
Das Modell der geteilten Verantwortung verstehen 3
Einführung 1
Feedback 6
Weitere Informationen 6
Tools und Ressourcen 6
EKS-Automatikmodus — Sicherheit 6
Sicherheitsarchitektur 8
Häufig gestellte Fragen (FAQ) 17
Identitäts- und Zugriffsverwaltung 20
Steuerung des Zugriffs auf EKS-Cluster 21
Empfehlungen für den Cluster-Zugriff 28
Identitäten und Anmeldeinformationen für EKS-Pods 35
Empfehlungen für Identitäten und Anmeldeinformationen für EKS-Pods 44
Tools und Ressourcen 53
Pod-Sicherheit 54
Linux-Funktionen 54
Sicherheitslösungen für Pods 56
Empfehlungen 65
Tools und Ressourcen 53
Mehrmandantenfähigkeit 73
Sanfte Mehrmandantenfähigkeit 73
Kubernetes-Konstrukte 76
Mildernde Kontrollen 78
Harte Mehrmandantenfähigkeit 86
Künftige Richtungen 87
Verwaltungstools und Ressourcen für mehrere Cluster 87
Detective Controls 88
Empfehlungen 65
Tools und Ressourcen 53

Netzwerksicherheit 96
Verkehrskontrolle 96
Netzwerkverschlüsselung 96
Netzwerkrichtlinie 97
Empfehlungen 65
Sicherheitsgruppen 102
Wann sollten Netzwerkrichtlinien und wann Sicherheitsgruppen für Pods verwendet werden? 105
Durchsetzung von Service Mesh-Richtlinien oder Kubernetes-Netzwerkrichtlinien 106
ThirdParty Engines für Netzwerkrichtlinien 107
Verschlüsselung während der Übertragung 109
Tools und Ressourcen 53
Datenverschlüsselung und Verwaltung von Geheimnissen 120
Verschlüsselung im Ruhezustand 120
Verwaltung von Secrets 122
Tools und Ressourcen 53
Laufzeit-Sicherheit 125
Sicherheitskontexte und integrierte Kubernetes-Steuerelemente 126
Empfehlungen 65
Tools und Ressourcen 130
Infrastruktursicherheit 130
Empfehlungen 65
Alternativen 135
Tools und Ressourcen 53
Einhaltung gesetzlicher Vorschriften 139
Verlagerung nach links 140
Tools und Ressourcen 53
Reaktion auf Zwischenfälle und Forensik 142
Beispiel für einen Plan zur Reaktion auf Vorfälle 142
Empfehlungen 65
Tools und Ressourcen 53
Image-Sicherheit 148
Empfehlungen 65
Tools und Ressourcen 53
Strategie für mehrere Konten 159
Planung einer Kontenstrategie für mehrere Workloads für Multi-Tenant-Cluster 159

Zentralisierter EKS-Cluster 160
Dezentrale EKS-Cluster 166
Zentralisierte und dezentrale EKS-Cluster 168
Verwaltung des Cluster-Zugriffs 171
EKS-Zugriffsverwaltungsoptionen 171
Option 1: AWS IAM Identity Center mit Cluster Access Management (CAM) -API 171
Option 2: AWS IAM ist Users/Roles Kubernetes-Gruppen zugeordnet 174
Option 3: OIDC-Anbieter 174
AWS EKS Pod Identity im Vergleich zu IRSA für Workloads 175
Empfehlung 176
Automatische Clusterskalierung 177
EKS Auto Mode 177
Gründe für die Verwendung des automatischen Modus 178
Häufig gestellte Fragen 178
Karpenter 184
Empfehlungen 65
Bewährte Methoden von Karpenter 185
Erstellen NodePools 189
Pods planen 194
CoreDNS-Empfehlungen 197
Baupläne von Karpenter 198
Weitere Ressourcen 95
Cluster Autoscaler 198
-Übersicht 198
Optimierung im Hinblick auf Leistung und Skalierbarkeit 203
Optimierung im Hinblick auf Kosten und Verfügbarkeit 206
Fortschrittliche Anwendungsfälle 209
Zusätzliche Parameter 212
Weitere Ressourcen 95
Referenzen 86
Zuverlässigkeit 218
Verwendung dieses Leitfadens 3
Einführung 1
Feedback 6
Anwendungen 221
Empfehlungen 65

Horizontaler Pod-Autoscaler (HPA) 226
Vertikaler Pod-Autoscaler (VPA) 227
Anwendungen werden aktualisiert 228
Gesundheitschecks und Selbstheilung 230
Empfehlungen 231
Umgang mit Störungen 233
Empfehlungen 234
Beobachtbarkeit 236
Empfehlungen 237
Steuerebene 239
EKS-Architektur 240
Empfehlungen 241
Überwachen Sie die Kennzahlen der Kontrollebene 241
Cluster-Authentifizierung 244
Zulassung: Webhooks 246
Umgang mit Cluster-Upgrades 247
Konnektivität von Cluster-Endpunkten 248
Große Cluster ausführen 249
Zusätzliche Ressourcen: 249
Datenebene 249
Empfehlungen 65
Netzwerk 257
Kubernetes-Netzwerkmodell 257
Container-Netzwerkschnittstelle (CNI) 258
Amazon Virtual Private Cloud (VPC) CNI 259
Subnetz-Rechner 260
Überlegungen zu VPC und Subnetzen 261
-Übersicht 198
Empfehlungen 65
Amazon VPC CNI 273
-Übersicht 198
Empfehlungen 65
Optimierung der Nutzung von IP-Adressen 287
Optimieren Sie den IP-Verbrauch auf Knotenebene 288
Reduzieren Sie die IP-Erschöpfung 288
IPv6 Cluster ausführen 296

-Übersicht 198
Empfehlungen 65
Benutzerdefiniertes Netzwerk 306
Beispielkonfiguration 307
Empfehlungen 65
Präfixmodus für Linux 315
Empfehlungen 65
Präfix-Modus für Windows 320
Empfehlungen 65
Sicherheitsgruppen pro Pod 326
Empfehlungen 65
Lastausgleich 337
Auswahl des Load Balancer Balancer-Typs 338
Bereitstellung von Load Balancern 339
Wählen Sie den Load Balancer-Zieltyp 340
Konfiguration von Load Balancer Balancer-Zustandsprüfungen 342
Verfügbarkeit und Pod-Lebenszyklus 343
Referenzen 86
Anhang 348
Überwachung von Netzwerkleistungsproblemen 350
Überwachung des CoreDNS-Datenverkehrs auf DNS-Throttling-Probleme 350
Überwachung von Verzögerungen bei DNS-Abfragen mithilfe von Conntrack-Metriken 351
Andere wichtige Kennzahlen zur Netzwerkleistung 351
Erfassung der Metriken zur Überwachung von Workloads im Hinblick auf Netzwerkleistungsprobleme 352
Kube-Proxy im IPVS-Modus ausführen 360
-Übersicht 198
Skalierbarkeit 365
Verwendung dieses Leitfadens 365
Die Dimensionen der Skalierung verstehen 365
Extra große Skalierung 366
Steuerebene 367
Begrenzen Sie Arbeitslast und Node-Bursting 367
Skalieren Sie Knoten und Pods sicher herunter 368
Verwenden Sie den clientseitigen Cache, wenn Sie Kubectl ausführen 369
Deaktivieren Sie die Kubectl-Komprimierung 369

Shard Cluster Autoscaler 369
API-Priorität und Fairness 371
Ressourcen auf dem API-Server werden abgerufen 379
Datenebene 386
Automatische Knotenskalierung 386
Verwenden Sie viele verschiedene EC2-Instance-Typen 386
Bevorzugen Sie größere Knoten, um die API-Serverlast zu reduzieren 387
Verwenden Sie ähnliche Knotengrößen für eine konsistente Workload-Leistung 388
Verwenden Sie Rechenressourcen effizient 389
Automatisieren Sie Amazon Machine Image (AMI) -Updates 390
Verwenden Sie mehrere EBS-Volumes für Container 390
Vermeiden Sie Instances mit niedrigen EBS-Attach-Limits, wenn Workloads EBS-Volumes verwenden 392
Deaktivieren Sie unnötiges Logging auf der Festplatte 392
Vorhandene Patch-Instanzen, wenn die Geschwindigkeit des Betriebssystem-Updates erforderlich ist 393
Cluster-Dienste 394
CoreDNS skalieren 394
Skalieren Sie den Kubernetes Metrics Server vertikal 396
Dauer CoreDNS CoreDNS-Lameduck 396
CoreDNS-Bereitschaftstest 397
Agenten protokollieren und überwachen 397
Workloads 398
IPv6 Für Pod-Netzwerke verwenden 399
Beschränken Sie die Anzahl der Dienste pro Namespace 399
Elastic Load Balancer Balancer-Kontingente verstehen 400
Verwenden Sie Route 53, Global Accelerator oder CloudFront 400
Verwenden Sie EndpointSlices anstelle von Endpunkten 401
Verwenden Sie nach Möglichkeit unveränderliche und externe Geheimnisse 401
Beschränken Sie den Bereitstellungsverlauf 402
enableServiceLinks Standardmäßig deaktivieren 403
Beschränken Sie Webhooks mit dynamischer Zulassung pro Ressource 403
Vergleichen Sie Workloads in mehreren Clustern 404
Die Theorie hinter der Skalierung 404
Nodes im Vergleich zur Abwanderungsrate 404
Denken Sie in Abfragen pro Sekunde 405

Skalierung verteilter Komponenten 406
Engpässe im Upstream- und Downstream-Bereich 406
Skalieren nach Metriken 408
Überwachung der Kontrollebene 411
API-Server 411
Wo ist das Problem? 411
Scheduler 417
Kube Controller-Manager 419
USW 421
Knoteneffizienz und Skalierung 422
Auswahl des Knotens 423
Bin-Packen für Knoten 424
Auslastung versus Sättigung 432
CPU-Grenzwerte festlegen 440
Kubernetes SLOs 444
Kubernetes SLOs 444
Kubernetes SLI-Metriken 447
SLOs auf Ihrem Cluster 451
Bekannte Limits und Service Quotas 452
Andere AWS-Servicekontingente 453
Drosselung von AWS-Anfragen 460
Andere bekannte Grenzwerte 461
Cluster-Upgrades 463
-Übersicht 463
Vor dem Upgrade 463
Behalten Sie Ihren Cluster up-to-date 465
Sehen Sie sich den EKS-Veröffentlichungskalender an 466
Erfahren Sie, wie das Modell der gemeinsamen Verantwortung für Cluster-Upgrades gilt 466
Führen Sie ein direktes Upgrade der Cluster durch 466
Rüsten Sie Ihre Steuerungsebene und Datenebene nacheinander auf 467
Verwenden Sie die EKS-Dokumentation, um eine Upgrade-Checkliste zu erstellen 468
Aktualisieren Sie Add-Ons und Komponenten mithilfe der Kubernetes-API 468
Überprüfen Sie vor dem Upgrade die grundlegenden EKS-Anforderungen 470
Überprüfen Sie die verfügbaren IP-Adressen 470
Überprüfen Sie die EKS-IAM-Rolle 471
Migrieren Sie zu EKS Add-ons 472

Identifizieren und korrigieren Sie die fehlende API-Nutzung, bevor Sie die Steuerungsebene aktualisieren ... 473
Einblicke in Cluster ... 473
K ube-no-trouble ... 475
Pluto ... 476
Ressourcen ... 476
Aktualisieren Sie Kubernetes-Workloads. Verwenden Sie kubectl-convert, um Manifeste zu aktualisieren ... 477
Konfigurieren Sie Ihre Workloads PodDisruptionBudgets und topologySpreadConstraints stellen Sie deren Verfügbarkeit sicher, während die Datenebene aktualisiert wird ... 478
Verwenden Sie Managed Node Groups oder Karpenter, um Upgrades auf Datenebene zu vereinfachen ... 480
Bestätigen Sie die Versionskompatibilität mit vorhandenen Knoten und der Steuerungsebene . 480
Aktivieren Sie das Ablaufen von Knoten für von Karpenter verwaltete Knoten ... 481
Verwenden Sie die Drift-Funktion für von Karpenter verwaltete Knoten ... 481
Verwenden Sie eksctl, um Upgrades für selbstverwaltete Knotengruppen zu automatisieren 482
Backup den Cluster vor dem Upgrade ... 482
Starten Sie die Fargate-Bereitstellungen nach dem Upgrade der Steuerungsebene neu ... 483
Evaluieren Sie Blue/Green Cluster als Alternative zu direkten Cluster-Upgrades ... 483
Verfolgen Sie geplante größere Änderungen im Kubernetes-Projekt — denken Sie voraus ... 484
Spezifische Hinweise zum Entfernen von Funktionen ... 484
Entfernung von Dockershim in 1.25 — Verwenden Sie Detector for Docker Socket (DDS) ... 484
Entfernung von PodSecurityPolicy in 1.25 — Migration zu Pod-Sicherheitsstandards oder einer Lösung policy-as-code ... 485
Der In-Tree-Storage-Treiber in Version 1.23 ist veraltet — Migration zu Container-Storage-Interface-Treibern (CSI) ... 485
Weitere Ressourcen ... 485
ClowdHaus Anleitung zum EKS-Upgrade ... 485
GoNoGo ... 485
Kostenoptimierung ... 486
Allgemeine Richtlinien ... 486
Bewährte Methoden zur Kostenoptimierung von EKS ... 486
Verwendung dieses Leitfadens ... 3
Die wichtigsten Funktionen von AWS-Services und Kubernetes ... 487
Feedback ... 6
Framework ... 488

Die Säule See: Messung und Rechenschaftspflicht ... 488
Die Säule „Sparen“: Kostenoptimierung ... 489
Die Säule des Plans: Planung und Prognose ... 490
Die Run-Säule ... 490
Referenzen ... 86
Sensibilisierung ... 491
Empfehlungen ... 65
Andere Tools ... 499
Datenverarbeitung ... 500
Passen Sie die Größe Ihrer Workloads an ... 501
Reduzieren Sie den Verbrauch ... 502
Reduzieren Sie ungenutzte Kapazitäten ... 503
Optimieren Sie die Arten von Rechenkapazitäten ... 509
Optimieren Sie die Computernutzung ... 513
Netzwerk ... 515
Kommunikation von Pod zu Pod ... 515
Kommunikation zwischen Load Balancer und Pod ... 528
Datenübertragung aus Container Registry ... 530
Datenübertragung ins Internet und zu AWS-Services ... 531
Datenübertragung zwischen VPCs ... 534
Verwenden eines Service Mesh ... 537
Weitere Ressourcen ... 95
Speicher ... 546
Übersicht ... 198
Kurzlebige Volumen ... 546
Persistente Volumes ... 548
Weitere Überlegungen ... 555
Beobachtbarkeit ... 556
Einführung ... 556
Protokollierung ... 556
EKS-Steuerebene ... 556
EKS-Datenebene ... 557
Metriken ... 562
Nachverfolgung ... 566
Zusätzliche Ressourcen: ... 95
Windows ... 569

AMI-Verwaltung 569
Verwaltung Ihres eigenen für Amazon EKS optimierten Windows-AMI 570
Konfiguration eines schnelleren Starts für benutzerdefinierte EKS-Optimierungen AMIs 571
Zwischenspeichern von Windows-Basislayern auf benutzerdefinierten AMIs 572
Blogbeitrag 573
gMSA für Windows-Container 573
Was ist ein gMSA-Konto 573
Windows-Container und GMSA-Anwendungsfall 574
Härtung von Windows-Servern 576
Reduzierung der Angriffsfläche mit Windows Server Core 576
Vermeidung von RDP-Verbindungen 577
Amazon Inspector 578
Amazon GuardDuty 579
Sicherheit in Amazon EC2 für Windows 580
Windows-Bilder scannen 580
Windows-Versionen und Lizenzierung 580
Windows Server Version 580
Lizenzen 581
Protokollierung 581
Empfehlungen zur Protokollierung 581
Überwachung von Windows-Containern 582
Windows-Netzwerke 587
Überblick über Windows-Containernetzwerke 587
IP-Adressverwaltung 588
Optionen für Container Network Interface (CNI) 592
Netzwerkrichtlinien 592
Speicher- und Systemverwaltung 592
System- und Kubelet-Speicher reservieren 593
Speicheranforderungen für Windows-Container 594
Schlussfolgerung 594
Verwaltung der Infrastruktur 594
Windows-Images übertragen und ziehen 595
Referenz 597
Planung 597
Bewährte Methoden PODs für die Zuweisung zu Knoten 598

Sicherstellen, dass betriebssystemspezifische Workloads auf dem entsprechenden Container-Host landen ... 598
Umgang mit mehreren Windows-Builds im selben Cluster ... 599
Vereinfachung und Tolerierung NodeSelector in Pod-Manifesten mit RuntimeClass ... 600
Support für verwaltete Knotengruppen ... 601
Zusätzliche Dokumentationen ... 602
Pod-Sicherheit für Windows-Container ... 602
Speicheroptionen ... 605
Was ist ein In-Tree- oder ein out-of-tree Volume-Plugin? ... 605
In-Tree-Volume-Plugin für Windows ... 606
Out-of-tree für Windows ... 610
Amazon FSx für Windows-Dateiserver ... 610
Härten von Windows-Container-Images ... 611
1. Konfigurieren Sie Kontorichtlinien (Passwort oder Sperrung) mithilfe der lokalen Sicherheitsrichtlinien und der Registrierung ... 612
2. Audit-Richtlinien ... 615
3. Bewährte Methoden zur IIS-Sicherheit für Windows-Container ... 618
4. Prinzip der geringsten Rechte ... 624
Letzte Gedanken: Warum der Schutz Ihrer Windows-Container in der heutigen Bedrohungslandschaft ein Muss ist ... 627
Hybrid ... 629
Netzwerkunterbrechung ... 629
Best Practices ... 630
Kubernetes-Pod-Failover ... 635
Netzwerkverkehr von Anwendungen ... 642
Host-Anmeldeinformationen ... 647
AI/ML ... 650
Feedback ... 650
Datenverarbeitung ... 650
GPU-Ressourcenoptimierung und Kostenmanagement ... 650
Knotenausfallsicherheit und Verwaltung von Schulungsaufträgen ... 666
Skalierung und Leistung von Anwendungen ... 669
Dynamische Ressourcenzuweisung für erweitertes GPU-Management ... 671
Netzwerk ... 718
Ziehen Sie für Anwendungen mit hoher Kommunikation zwischen Knoten eine höhere Netzwerkbandbreite oder einen Elastic Fabric-Adapter in Betracht ... 718

Erhöhen Sie die Anzahl der verfügbaren IP-Adressen, um schnellere Pod-Startzeiten zu ermöglichen ... 719
Sicherheit ... 720
Sicherheit und Compliance ... 720
Speicher ... 721
Datenverwaltung und Speicherung ... 722
Beobachtbarkeit ... 735
Überwachung und Beobachtbarkeit ... 735
Beobachtbarkeit und Metriken ... 737
Leistung ... 754
Verwaltung von ML-Artefakten, Bereitstellung von Frameworks und Startup-Optimierung ... 754
Optimierung der Abrufzeiten für Container-Images ... 757
Ziehen Sie NVMe Kubelet- und Containerspeicher in Betracht ... 760
Beitragen ... 761
Zusammenfassung für bestehende Mitwirkende ... 761
Richten Sie eine lokale Bearbeitungsumgebung ein ... 761
Forke und klone das Repo ... 761
Öffnen Sie den VS Code Workspace ... 762
Bearbeiten einer Datei ... 762
Reichen Sie eine Pull-Anfrage ein ... 762
Verwenden Sie den webbasierten Editor github.dev ... 763
Bearbeiten Sie eine einzelne Seite ... 763
Die ID für eine Seite anzeigen und festlegen ... 763
Stellen Sie die Seiten-ID ein ... 764
Erstellen einer neuen Seite ... 764
Seitenmetadaten erstellen ... 764
Zum Inhaltsverzeichnis hinzufügen ... 765
Fügen Sie ein Bild ein ... 765
Überprüfen Sie den Stil mit Vale ... 766
Erstellen Sie eine lokale Vorschau ... 766
AsciiDoc Spickzettel ... 767
Grundlegende Formatierung ... 767
Kopfzeilen ... 767
Listen ... 767
Links ... 767
Bilder ... 768

Code-Blöcke 768
Tabellen 768
Ermahnungen 768
Umfasst 769
..... dcclxx

Leitfaden zu bewährten Methoden für Amazon EKS

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Willkommen bei den EKS Best Practices Guides. Das Hauptziel dieses Projekts besteht darin, eine Reihe von Best Practices für den Betrieb von Amazon EKS am zweiten Tag anzubieten. Wir haben uns dafür entschieden, diese Leitlinien zu veröffentlichen, GitHub damit wir sie schnell wiederholen, zeitnahe und effektive Empfehlungen für eine Vielzahl von Anliegen geben und Vorschläge aus der breiteren Community problemlos berücksichtigen können.

Derzeit haben wir Leitfäden zu den folgenden Themen veröffentlicht:

- Bewährte Methoden für die Sicherheit
- Bewährte Verfahren für Zuverlässigkeit
- Bewährte Methoden für die automatische Clusterskalierung: Karpenter
- Bewährte Methoden für Cluster-Autoscaling: Cluster-Autoscaler
- Bewährte Methoden für die automatische Clusterskalierung: EKS-Automatikmodus
- Bewährte Methoden für Netzwerke
- Bewährte Methoden für Skalierbarkeit
- Bewährte Methoden für Cluster-Upgrades
- Bewährte Methoden zur Kostenoptimierung
- Bewährte Methoden für den Betrieb von Windows-Containern
- Bewährte Methoden für Hybridbereitstellungen
- Bewährte Methoden für die Ausführung von Workloads AI/ML

Wir haben auch eine Python-basierte CLI (Command Line Interface) namens hardeneks als Open Source bereitgestellt, um einige der Empfehlungen aus diesem Handbuch zu überprüfen.

In future werden wir Leitlinien für bewährte Verfahren für Leistung, Kostenoptimierung und operative Exzellenz veröffentlichen.

Verwandte Leitfäden

Neben dem EKS-Benutzerhandbuch hat AWS mehrere weitere Leitfäden veröffentlicht, die Ihnen bei der Implementierung von EKS helfen können.

- Best-Practice-Leitfäden für EMR-Container
- Daten zu EKS
- Bewährte Methoden für AWS-Observability
- Amazon EKS Blueprints für Terraform
- Amazon EKS Blueprints Schnellstart

Beitragen

Wir empfehlen Ihnen, zu diesen Leitfäden beizutragen. Wenn Sie eine Methode eingeführt haben, die sich als wirksam erwiesen hat, teilen Sie uns diese bitte mit, indem Sie ein Problem oder eine Pull-Anfrage öffnen. Wenn du einen Fehler oder eine Schwachstelle in den von uns bereits veröffentlichten Leitlinien entdeckst, reiche bitte ebenfalls eine PR ein, um ihn zu korrigieren. Die Richtlinien für das Einreichen PRs finden Sie in unseren Richtlinien für Beiträge.

Bewährte Methoden für die Sicherheit

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Dieser Leitfaden bietet Ratschläge zum Schutz von Informationen, Systemen und Anlagen, die auf EKS angewiesen sind, und bietet gleichzeitig durch Risikobewertungen und Strategien zur Risikominderung einen Mehrwert für das Unternehmen. Die hier aufgeführten Leitlinien sind Teil einer Reihe von Best-Practice-Leitfäden, die AWS veröffentlicht, um Kunden bei der Implementierung von EKS gemäß den Best Practices zu unterstützen. Leitfäden für Leistung, betriebliche Exzellenz, Kostenoptimierung und Zuverlässigkeit werden in den kommenden Monaten verfügbar sein.

Verwendung dieses Leitfadens

Dieser Leitfaden richtet sich an Sicherheitsexperten, die für die Implementierung und Überwachung der Wirksamkeit von Sicherheitskontrollen für EKS-Cluster und die von ihnen unterstützten Workloads verantwortlich sind. Der Leitfaden ist zur leichteren Handhabung in verschiedene Themenbereiche unterteilt. Jedes Thema beginnt mit einem kurzen Überblick, gefolgt von einer Liste mit Empfehlungen und bewährten Methoden zur Sicherung Ihrer EKS-Cluster. Die Themen müssen nicht in einer bestimmten Reihenfolge gelesen werden.

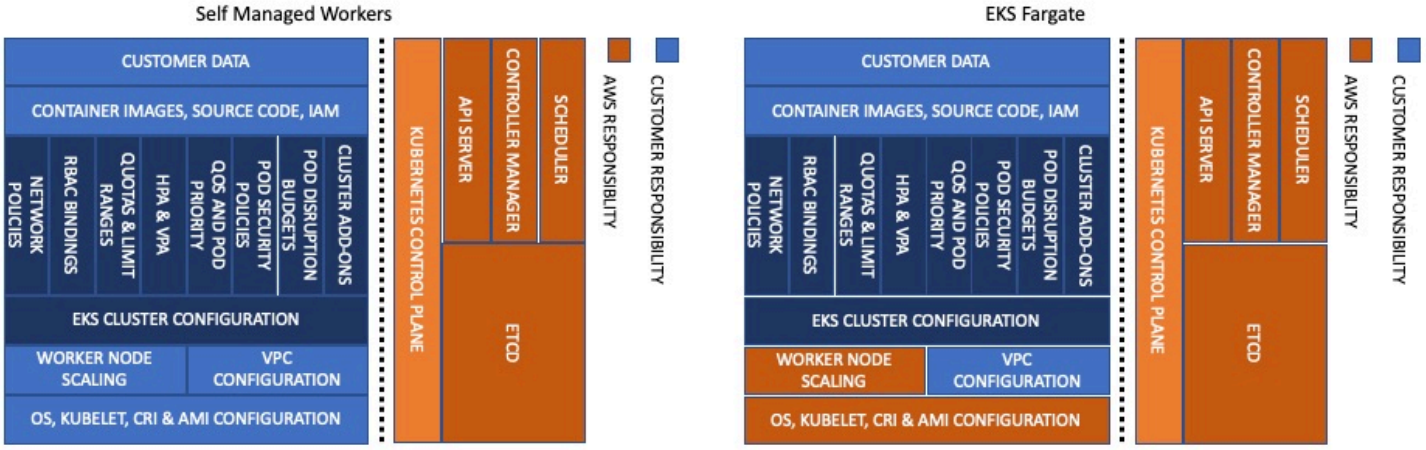
Das Modell der geteilten Verantwortung verstehen

Sicherheit und Compliance werden bei der Nutzung eines verwalteten Dienstes wie EKS als gemeinsame Verantwortung betrachtet. Im Allgemeinen ist AWS für die Sicherheit „der“ Cloud verantwortlich, während Sie als Kunde für die Sicherheit „in“ der Cloud verantwortlich sind. Mit EKS ist AWS für die Verwaltung der von EKS verwalteten Kubernetes-Steuerebene verantwortlich. Dazu gehören die Knoten der Kubernetes-Steuerebene, die ETCD-Datenbank und andere Infrastrukturen, die AWS benötigt, um einen sicheren und zuverlässigen Service bereitzustellen. Als Nutzer von EKS sind Sie weitgehend für die Themen in diesem Leitfaden verantwortlich, z. B. IAM, Pod-Sicherheit, Laufzeitsicherheit, Netzwerksicherheit usw.

Wenn es um die Sicherheit der Infrastruktur geht, übernimmt AWS zusätzliche Aufgaben, wenn Sie von selbstverwalteten Mitarbeitern zu verwalteten Knotengruppen zu Fargate wechseln. Mit Fargate

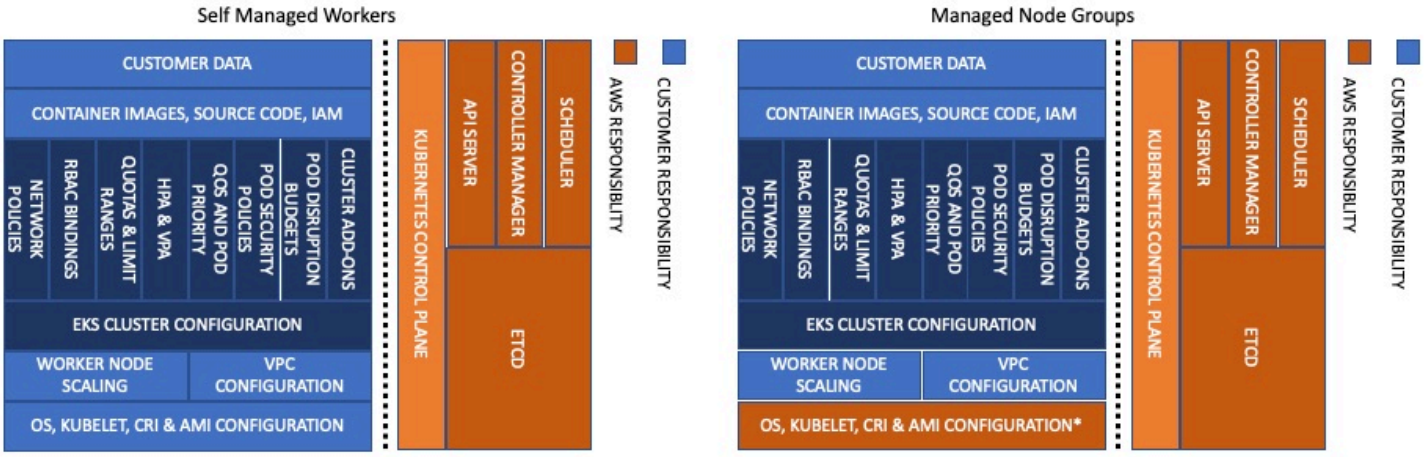
übernimmt AWS beispielsweise die Verantwortung für die Sicherung der Basisdaten, die für den Betrieb Ihrer Pods instance/runtime verwendet werden.

Modell der geteilten Verantwortung - Fargate



AWS wird auch dafür verantwortlich sein, das EKS-optimierte AMI mit Kubernetes-Patch-Versionen und Sicherheitspatches auf dem neuesten Stand zu halten. Kunden, die Managed Node Groups (MNG) verwenden, sind dafür verantwortlich, ihre Nodegroups über EKS-API, CLI, Cloudformation oder AWS-Konsole auf das neueste AMI zu aktualisieren. Im Gegensatz zu Fargate MNGs wird Ihre Infrastruktur/Ihr Cluster auch nicht automatisch skaliert. Dies kann durch den Cluster-Autoscaler oder andere Technologien wie Karpenter, natives AWS-Autoscaling, Ocean oder SpotInst Atlassians Escalator erledigt werden.

Modell der geteilten Verantwortung — MNG



Bevor Sie Ihr System entwerfen, ist es wichtig zu wissen, wo die Grenze zwischen Ihren Verantwortlichkeiten und dem Serviceanbieter (AWS) verläuft.

Weitere Informationen zum Modell der gemeinsamen Verantwortung finden Sie unter/https://aws.amazon.com/compliance/shared-responsibility-model

Einführung

Es gibt mehrere Bereiche mit bewährten Sicherheitsmethoden, die für die Verwendung eines verwalteten Kubernetes-Dienstes wie EKS relevant sind:

- Identitäts- und Zugriffsverwaltung
- Sicherheit von Pods
- Sicherheit während der Laufzeit
- Netzwerksicherheit
- Mehrmandantenfähigkeit
- Mehrere Konten für mehrere Mandanten
- Detective Controls
- Infrastruktursicherheit
- Datenverschlüsselung und Verwaltung von Geheimnissen
- Einhaltung gesetzlicher Vorschriften
- Reaktion auf Vorfälle und Forensik
- Sicherheit von Bildern

Bei der Entwicklung eines Systems müssen Sie die Auswirkungen auf die Sicherheit und die Praktiken berücksichtigen, die sich auf Ihre Sicherheitslage auswirken können. Sie müssen beispielsweise kontrollieren, wer Aktionen gegen eine Reihe von Ressourcen ausführen kann. Außerdem müssen Sie in der Lage sein, Sicherheitsvorfälle schnell zu erkennen, Ihre Systeme und Dienste vor unbefugtem Zugriff zu schützen und die Vertraulichkeit und Integrität von Daten durch Datenschutz zu wahren. Ein klar definierter und eingespielter Satz von Prozessen zur Reaktion auf Sicherheitsvorfälle wird auch Ihre Sicherheitslage verbessern. Derartige Tools und Techniken sind unabdinglich, um Ihr Unternehmen vor finanziellen Verlusten zu schützen und gesetzliche Vorgaben zu erfüllen.

AWS unterstützt Unternehmen dabei, ihre Sicherheits- und Compliance-Ziele zu erreichen, indem es eine breite Palette von Sicherheitsservices anbietet, die auf der Grundlage des Feedbacks einer Vielzahl sicherheitsbewusster Kunden weiterentwickelt wurden. Durch die Bereitstellung einer hochsicheren Grundlage können Kunden weniger Zeit mit „undifferenzierter Schwerarbeit“ und mehr Zeit mit der Erreichung ihrer Geschäftsziele verbringen.

Feedback

Dieser Leitfaden wird am GitHub veröffentlicht, um direktes Feedback und Vorschläge von der breiteren EKS/Kubernetes Community zu sammeln. Wenn Sie eine bewährte Methode haben, die wir Ihrer Meinung nach in den Leitfaden aufnehmen sollten, reichen Sie bitte ein Problem ein oder reichen Sie eine PR im GitHub Repository ein. Wir beabsichtigen, den Leitfaden regelmäßig zu aktualisieren, sobald der Service um neue Funktionen erweitert wird oder wenn sich eine neue bewährte Methode herausstellt.

Weitere Informationen

Dieses Whitepaper zur Kubernetes-Sicherheit, das von der Security Audit Working Group gesponsert wird, beschreibt die wichtigsten Aspekte der Kubernetes-Angriffsfläche und der Sicherheitsarchitektur mit dem Ziel, Sicherheitsexperten dabei zu unterstützen, fundierte Design- und Implementierungsentscheidungen zu treffen.

Die CNCF veröffentlichte auch ein Whitepaper zur Cloud-nativen Sicherheit. Das paper untersucht, wie sich die Technologielandschaft entwickelt hat, und setzt sich für die Einführung von Sicherheitspraktiken ein, die auf DevOps Prozesse und agile Methoden abgestimmt sind.

Tools und Ressourcen

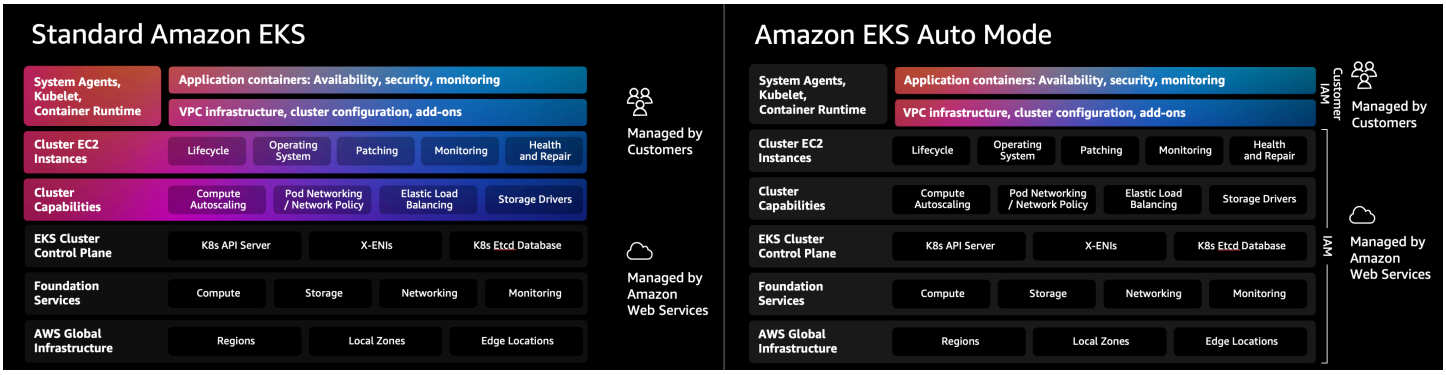
Workshop zum Eintauchen in die Sicherheit von Amazon EKS

Automatischer EKS-Modus — Sicherheit

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Amazon EKS Auto Mode bietet erweiterte Sicherheitsfunktionen, indem das Sicherheitsmanagement von AWS über die Kontrollebene hinaus auf Worker-Knoten und zentrale Cluster-Komponenten ausgedehnt wird. Dieses umfassende Sicherheitsmodell hilft Unternehmen dabei, ein starkes Sicherheitsniveau aufrechtzuerhalten und gleichzeitig den betrieblichen Aufwand zu reduzieren.

Modell mit geteilter Verantwortung — EKS Auto Mode



Zu den wichtigsten Sicherheitsverbesserungen im automatischen Modus von EKS gehören:

- Minimales containeroptimiertes Betriebssystem mit reduzierter Angriffsfläche
- Bewährte Sicherheitsmethoden durch EC2 Managed Instances
- Automatisiertes Sicherheitspatch-Management mit obligatorischer Knotenrotation
- Integrierte Knotenisolierung und Sicherheitsgrenzen
- Optimierte IAM-Integration mit nur minimal erforderlichen Berechtigungen

Der automatische Modus von EKS setzt diese Sicherheitskontrollen standardmäßig durch und hilft Unternehmen dabei, ihre Sicherheits- und Compliance-Anforderungen zu erfüllen und gleichzeitig den Clusterbetrieb zu vereinfachen. Dieser Ansatz entspricht den defense-in-depth Prinzipien und bietet mehrere Ebenen von Sicherheitskontrollen auf Infrastruktur-, Knoten- und Workload-Ebene.

> **Important**
>
> Der automatische Modus von EKS bietet zwar erweiterte Sicherheitsfunktionen, Unternehmen sollten jedoch dennoch angemessene Sicherheitskontrollen auf Anwendungsebene implementieren und die bewährten Sicherheitsmethoden für ihre Workloads, die auf dem Cluster ausgeführt werden, befolgen.

Sicherheitsarchitektur

Der automatische Modus von EKS implementiert Sicherheitskontrollen auf mehreren Ebenen der EKS-Infrastruktur, von der Steuerungsebene bis hin zu einzelnen Knoten. Das Verständnis dieser Architektur ist entscheidend für die effektive Verwaltung und Sicherung Ihrer EKS-Cluster.

Sicherheit auf der Kontrollebene

Die EKS-Steuerebene im EKS-Automatikmodus behält dieselben hohen Sicherheitsstandards wie herkömmliche EKS-Cluster bei und bietet gleichzeitig neue Sicherheitsfunktionen:

- Envelope-Verschlüsselung: Alle Kubernetes-API-Daten werden automatisch mithilfe der Envelope-Verschlüsselung verschlüsselt.
- KMS-Integration: Verwendet AWS KMS mit Kubernetes KMS-Anbieter v2, mit Optionen für AWS-eigene Schlüssel oder kundenverwaltete Schlüssel (CMK).
- Verbessertes Komponentenmanagement: Kritische Komponenten wie auto-scaling, ENI-Management und EBS-Controller werden außerhalb des Clusters verschoben und von AWS verwaltet.
- Verbesserte Sicherheitskontrollen und Auditfunktionen: Die für den automatischen Modus von EKS erforderlichen Berechtigungen, die über die standardmäßigen EKS-Cluster hinausgehen, werden vollständig über die Cluster-IAM-Rolle und nicht über einzelne Knotenrollen verwaltet.

IAM-Integration und Zugriffsmanagement

Der automatische Modus von EKS bietet eine verbesserte Integration mit AWS Identity and Access Management (IAM) über EKS Access Entries und EKS Pod Identity.

Cluster-Zugriffsverwaltung

Der automatische Modus von EKS bietet Verbesserungen bei der Cluster-Zugriffsverwaltung über die Cluster Access Management (CAM) -API:

- Standardisierte Authentifizierungsmodi durch `EKS_API`
- Verbesserte Sicherheit durch API-basiertes Zugriffsmanagement
- Vereinfachte Zugriffskontrolle mithilfe von Zugriffseinträgen und Zugriffsrichtlinien

Zugriffseinträge können erstellt werden, um den Clusterzugriff zu verwalten:

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

Important

Es ist zwar immer noch möglich, einen EKS-Automodus-Cluster mit `CONFIG_MAP_AND_API` Authentifizierungsmodus zu erstellen, dies ist jedoch nicht der Standardansatz, und es wird dringend empfohlen, den `API` Standardauthentifizierungsmodus für neue Cluster zu verwenden. `API`Die basierte Authentifizierung bietet im Vergleich zum herkömmlichen Ansatz mehr Sicherheit und eine vereinfachte Zugriffsverwaltung. ConfigMap

EKS Pod Identity

Der automatische Modus von EKS wird mit dem bereits bereitgestellten Pod Identity Agent geliefert, sodass Pods auf optimierte Weise AWS IAM-Berechtigungen erteilt werden können:

- Vereinfachtes IAM-Berechtigungsmanagement ohne OIDC-Anbieterkonfiguration
- Geringerer Betriebsaufwand im Vergleich zu IRSA
- Verbesserte Sicherheit durch Sitzungskennzeichnung und ABAC-Unterstützung

```
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 ist der empfohlene Ansatz für IAM-Berechtigungen im automatischen EKS-Modus, da er im Vergleich zu IRSA erweiterte Sicherheitsfunktionen und eine vereinfachte Verwaltung bietet.

Knoten-IAM-Rolle

Der EKS-Automodus verwendet einen neuen Modus, der nur `AmazonEKSWorkerNodeMinimalPolicy` die Berechtigungen bereitstellt, die für den Betrieb der EKS-Automodus-Knoten erforderlich sind. Diese Berechtigungen:

- Stellen Sie im Vergleich zu herkömmlichen Knotenrichtlinien weniger Berechtigungen bereit
- Halten Sie sich an das Prinzip der geringsten Rechte
- Werden automatisch von AWS verwaltet und aktualisiert

Dieser Ansatz mit minimalen Richtlinien trägt zur Verbesserung der Sicherheitslage bei, indem die für den Knoten und seine Workloads verfügbaren Berechtigungen eingeschränkt werden.

Sicherheit der Knoten

Der automatische Modus von EKS bietet mehrere signifikante Sicherheitsverbesserungen auf Knotenebene:

Sicherheit verwalteter EC2-Instances

EKS-Automodus-Knoten verwenden Amazon EC2 Managed Instances mit erweiterten Sicherheitseigenschaften:

- Durch IAM erzwungene Einschränkungen, die Operationen verhindern, die den Betrieb der Knoten durch AWS beeinträchtigen könnten
- Unveränderliche Infrastrukturmuster, bei denen Konfigurationsänderungen den Austausch von Knoten erfordern
- Obligatorischer Knotenaustausch innerhalb von 21 Tagen, um regelmäßige Sicherheitsupdates zu gewährleisten
- Eingeschränkter Zugriff auf Instanz-Metadaten IMDSv2 mithilfe kontrollierter Hop-Limits

Betriebssystemsicherheit

Das Betriebssystem ist eine benutzerdefinierte Variante von Bottlerocket, die für den EKS-Automatikmodus optimiert wurde und Folgendes beinhaltet:

- Schreibgeschütztes Root-Dateisystem
- SELinux standardmäßig aktiviert mit obligatorischen Zugriffskontrollen

- Automatische Pod-Isolierung mithilfe eindeutiger SELinux MCS-Labels
- SSH-Zugriff deaktiviert und unnötige Dienste entfernt
- Automatisierte Sicherheitspatches durch Knotenrotation

Sicherheit der Knotenkomponenten

Für die Konfiguration der Knotenkomponenten gelten bewährte Sicherheitsmethoden:

- Kubelet ist mit sicheren Standardeinstellungen konfiguriert
- Robuste Konfiguration für Container-Runtime
- Automatisierte Verwaltung und Rotation von Zertifikaten
- Eingeschränkte node-to-control-plane Kommunikation

Netzwerksicherheit

Der automatische Modus von EKS implementiert mehrere Netzwerksicherheitsfunktionen, um eine sichere Kommunikation innerhalb des Clusters und mit externen Ressourcen zu gewährleisten:

VPC CNI-Netzwerkrichtlinie

Der EKS Auto Mode nutzt die native Kubernetes-Netzwerkrichtlinien-Unterstützung des Amazon VPC CNI-Plug-ins:

- Lässt sich in die Upstream-API für Kubernetes-Netzwerkrichtlinien integrieren
- Ermöglicht eine detaillierte Steuerung der Kommunikation pod-to-pod
- Unterstützt sowohl Eingangs- als auch Ausgangsregeln

Um die Unterstützung von Netzwerkrichtlinien im automatischen EKS-Modus zu aktivieren, müssen Sie das VPC CNI-Add-on mit einem `configMap` Manifest konfigurieren. Ein Beispiel:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-network-policy: "true"
```

Es ist auch erforderlich, zu definieren, dass die Netzwerkrichtlinien-Unterstützung in der Node-Klasse konfiguriert ist, wie hier dargestellt:

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

Nach der Aktivierung können Sie Netzwerkrichtlinien zur Steuerung des Datenverkehrs erstellen:

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

Verbessertes ENI-Management

Der automatische Modus von EKS bietet verbesserte Sicherheit für das Elastic Network Interface (ENI) -Management:

- AWS-verwalteter ENI-Anschluss und Konfiguration
- Trennung des Kontrollverkehrs vom Datenverkehr
- Automatisierte IP-Adressverwaltung mit reduzierten Rechten, die für Knoten erforderlich sind

Speichersicherheit

Der automatische Modus von EKS bietet erweiterte Sicherheitsfunktionen sowohl für kurzlebigen als auch für persistenten Speicher:

Kurzlebiger Speicher

- Alle Daten, die auf ephemere Volumes geschrieben werden, werden automatisch verschlüsselt

- Verwendet den branchenüblichen kryptografischen AES-256-Algorithmus
- Verschlüsselung und Entschlüsselung werden vom Service nahtlos abgewickelt

EBS-Volumes

- Stamm- und Daten-EBS-Volumes sind immer verschlüsselt
- Volumes sind so konfiguriert, dass sie bei Beendigung der Instance gelöscht werden
- Es besteht die Möglichkeit, benutzerdefinierte KMS-Schlüssel für die Verschlüsselung anzugeben

EFS-Integration

- Support für Verschlüsselung bei der Übertragung mit EFS
- Automatische Verschlüsselung im Ruhezustand für EFS-Dateisysteme
- Integration mit EFS Access Points für verbesserte Zugriffskontrolle

> **Important**
>
> Wenn Sie EFS mit dem EKS-Automatikmodus verwenden, stellen Sie sicher, dass die entsprechenden Verschlüsselungseinstellungen auf EFS-Dateisystemebene konfiguriert sind, da der EKS-Automatikmodus die EFS-Verschlüsselung nicht direkt verwaltet.

Überwachung und Protokollierung

Der automatische Modus von EKS bietet erweiterte Überwachungs- und Protokollierungsfunktionen, damit Sie den Überblick über die Sicherheitslage und den Betriebszustand Ihres Clusters behalten können.

Steuerebenen-Protokollierung

Der automatische Modus von EKS bietet dieselben Protokollierungsfunktionen auf der Steuerungsebene wie das Standard-EKS, aktiviert jedoch standardmäßig alle Protokolle, um die Überwachung zu verbessern.

- Protokolle werden an Amazon CloudWatch Logs gesendet
- Standardmäßig aktiviert der automatische EKS-Modus alle Protokolle auf der Kontrollebene: API-Server, Audit, Authenticator, Controller-Manager und Scheduler

- Der automatische EKS-Modus ermöglicht einen detaillierten Einblick in Clusteroperationen und Sicherheitsereignisse

> **Important**
>
> Für die Protokollierung auf der Kontrollebene fallen zusätzliche Kosten für die Protokollspeicherung an CloudWatch. Überlegen Sie sich Ihre Protokollierungsstrategie sorgfältig, um ein Gleichgewicht zwischen Sicherheitsanforderungen und Kostenmanagement zu finden.

Protokollierung auf Knotenebene

Der automatische EKS-Modus verbessert die Protokollierung auf Knotenebene:

- Systemprotokolle werden automatisch gesammelt und können über Logs abgerufen werden CloudWatch
- Knotenprotokolle werden auch nach der Beendigung des Knotens aufbewahrt, was die Analyse nach einem Vorfall erleichtert
- Verbesserter Einblick in Sicherheitsereignisse und betriebliche Probleme auf Knotenebene

GuardDuty Amazon-Integration

EKS-Auto-Mode-Cluster lassen sich nahtlos in Amazon integrieren, GuardDuty um die Bedrohungserkennung zu verbessern. Zu den Merkmalen gehören:

- Automatisiertes Scannen von Auditprotokollen auf der Kontrollebene
- Laufzeitüberwachung, die für die Überwachung von Workloads aktiviert werden kann
- Integration mit bestehenden GuardDuty Ergebnissen und Warnmechanismen

Um den EKS-Automode-Schutz auf Amazon GuardDuty for Kubernetes Audit Logs zu aktivieren, können Sie den folgenden Befehl ausführen:

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

GuardDuty Amazon-Integration für Runtime-Sicherheit

Amazon GuardDuty bietet grundlegende Runtime-Sicherheitsüberwachung für EKS-Auto-Mode-Cluster und bietet umfassende Funktionen zur Bedrohungserkennung und Sicherheitsüberwachung. Diese Integration ist besonders wichtig, da sie dabei hilft, potenzielle Sicherheitsbedrohungen und böswillige Aktivitäten in Echtzeit zu identifizieren.

GuardDuty Hauptfunktionen für den EKS-Automatikmodus

- Überwachung der Laufzeit:
  - Kontinuierliche Überwachung des Laufzeitverhaltens
  - Erkennung bösartiger oder verdächtiger Aktivitäten
  - Identifizierung potenzieller Fluchtversuche aus Containern
  - Überwachung ungewöhnlicher Prozessausführungen oder Netzwerkverbindungen
- Kubernetes-spezifische Bedrohungserkennung:
  - Identifizierung verdächtiger Versuche, Pods bereitzustellen
  - Erkennung kompromittierter Container
  - Überwachung privilegierter Container-Starts
  - Identifizierung einer verdächtigen Nutzung von Dienstkonten
- Umfassende Suchtypen:
  - Policy:Kubernetes/* — Erkennt Verstöße gegen bewährte Sicherheitsverfahren
  - Impact:Kubernetes/* — Identifiziert potenziell betroffene Ressourcen
  - Discovery:Kubernetes/* — Erkennt Aufklärungsaktivitäten
  - Execution:Kubernetes/* — Identifiziert verdächtige Ausführungsmuster
  - Persistence:Kubernetes/* — Erkennt potenziell anhaltende Bedrohungen

Um den EKS-Automode-Schutz auf Amazon GuardDuty für Kubernetes Audit Logs and Runtime Monitoring zu aktivieren, können Sie den folgenden Befehl ausführen:

```
aws guardduty update-detector \
    --detector-id 12abc34d567e8fa901bc2d34e56789f0 \
    --data-sources '{
        "Kubernetes": {
            "AuditLogs": {"Enable": true},
            "RuntimeMonitoring": {"Enable": true}
        }
```

```
    }'
```

> **Important**
>
> GuardDuty Runtime Monitoring wird in EKS Auto Mode-Clustern automatisch unterstützt und bietet so eine verbesserte Sicherheitstransparenz ohne zusätzliche Konfiguration auf Knotenebene.

GuardDuty Integration der Ergebnisse

GuardDuty Die Ergebnisse können in andere AWS-Services integriert werden, um eine automatisierte Reaktion zu ermöglichen:

- EventBridge Regeln:

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

- Integration von Security Hub:

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

Bewährte Methoden für GuardDuty den EKS-Automatikmodus

1. Alle Suchtypen aktivieren:
   - Aktivieren Sie sowohl die Überwachung des Kubernetes-Auditprotokolls als auch die Laufzeitüberwachung

- Konfigurieren Sie die Ergebnisse für alle Schweregrade

2. Implementieren Sie eine automatisierte Reaktion:
   - Erstellen Sie EventBridge Regeln für Ergebnisse mit hohem Schweregrad
   - Integrieren Sie in AWS Security Hub für ein zentralisiertes Sicherheitsmanagement
   - Richten Sie gegebenenfalls automatisierte Abhilfemaßnahmen ein
3. Regelmäßige Überprüfung und Optimierung:
   - Regelmäßige Überprüfung der GuardDuty Ergebnisse
   - Passen Sie die Erkennungsschwellenwerte an Ihre Umgebung an
   - Aktualisieren Sie die Reaktionsverfahren auf der Grundlage neuer Erkennungstypen
4. Kontoübergreifende Verwaltung:
   - Erwägen Sie die Verwendung eines GuardDuty Administratorkontos für die zentrale Verwaltung
   - Ermöglichen Sie die Aggregation von Ergebnissen für mehrere Konten

> **Warning**
>
> Es GuardDuty bietet zwar eine umfassende Sicherheitsüberwachung, sollte aber Teil einer defense-in-depth Strategie sein, die auch andere Sicherheitskontrollen wie Netzwerkrichtlinien, Pod-Sicherheitsstandards und die richtige RBAC-Konfiguration umfasst.

Häufig gestellte Fragen (FAQ)

F: Wie unterscheidet sich der EKS Auto Mode in Bezug auf die Sicherheit vom Standard-EKS?
A: Der automatische EKS-Modus bietet erweiterte Sicherheit durch EC2-verwaltete Instances, automatisiertes Patchen, obligatorische Knotenrotation und integrierte Sicherheitskontrollen. Es reduziert den betrieblichen Aufwand und sorgt gleichzeitig für ein hohes Maß an Sicherheit, da AWS mehr Sicherheitsaspekte verwaltet.

F: Kann ich mit dem automatischen Modus von EKS weiterhin vorhandene Sicherheitstools und -richtlinien verwenden? A: Ja, der automatische Modus von EKS ist mit den meisten vorhandenen Sicherheitstools und -richtlinien kompatibel. Einige Sicherheitstools auf Knotenebene müssen jedoch möglicherweise angepasst werden, da die EKS-Automodus-Knoten verwaltet werden.

F: Wie setze ich Sicherheitsagenten und Überwachungstools im automatischen EKS-Modus ein? A: Im automatischen EKS-Modus sollten Sicherheitsagenten und Überwachungstools als Kubernetes-

Workloads bereitgestellt werden (in der Regel wird standardmäßig eine Pod-Instanz auf jedem Knoten bereitgestellt) DaemonSets, anstatt sie direkt auf dem Betriebssystem des Knotens zu installieren. Dieser Ansatz entspricht dem unveränderlichen Infrastrukturmodell von EKS Auto Mode. Beispiel:

```
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"]
```

F: Sind Sicherheitslösungen von Drittanbietern mit EKS Auto Mode kompatibel? A: Viele beliebte Sicherheitslösungen von Drittanbietern wurden aktualisiert, um den EKS Auto Mode zu unterstützen. Es wird jedoch immer empfohlen, die spezifischen Versions- und Bereitstellungsanforderungen mit Ihrem Sicherheitsanbieter zu überprüfen, da für die Unterstützung des EKS Auto Mode möglicherweise aktualisierte Versionen oder spezifische Bereitstellungskonfigurationen erforderlich sind.

F: Was sind die Einschränkungen für Security Agents im EKS Auto Mode? A: Zu den wichtigsten Einschränkungen gehören:

- Kein direkter Zugriff zur Änderung des Betriebssystems des Knotens
- Keine Persistenz über Knotenrotationen hinweg
- Muss mit der containerbasierten Bereitstellung kompatibel sein

- Muss die Unveränderlichkeit des Knotens respektieren
- Möglicherweise sind unterschiedliche Rechtekonfigurationen erforderlich
- Alle dauerhaften Änderungen an den Knoten sollten über die `NodePools` `NodeClasses` Ressourcen vorgenommen werden.

> **Note**
>
> Der automatische Modus von EKS kann zwar Anpassungen Ihrer Strategie zur Implementierung von Sicherheitstools erfordern, diese Änderungen führen jedoch häufig zu wartbareren und sichereren Konfigurationen, die auf Cloud-native Best Practices abgestimmt sind. Der EKS Auto Mode geht davon aus, dass er die meisten Funktionen, die er verwaltet, vollständig übernehmen wird. Daher könnten alle manuellen Änderungen, die Sie an diesen Funktionen vornehmen, vom EKS-Automodus überschrieben oder verworfen werden, sofern Ihnen diese zur Verfügung stehen.

F: Kann ich den benutzerdefinierten Modus AMIs mit dem EKS-Automatikmodus verwenden? A: Derzeit unterstützt der EKS-Automodus keine benutzerdefinierten Einstellungen AMIs. Dies ist beabsichtigt, da AWS die Sicherheit, das Patchen und die Wartung der Knoten im Rahmen des Modells der gemeinsamen Verantwortung verwaltet. Die EKS-Automodus-Knoten verwenden eine spezielle Variante von Bottlerocket, die von AWS optimiert und gewartet wird.

F: Wie oft werden Knoten im EKS-Automodus automatisch rotiert? A: Knoten im EKS-Automatikmodus haben eine maximale Lebensdauer von 21 Tagen. Sie werden vor Ablauf dieser Grenze automatisch ersetzt, sodass regelmäßige Sicherheitsupdates und Patches gewährleistet sind.

F: Kann ich zur Fehlerbehebung per SSH auf EKS-Automodus-Knoten zugreifen? A: Nein, direkter SSH-Zugriff ist im EKS-Automatikmodus nicht verfügbar. Stattdessen können Sie die NodeDiagnostic Custom Resource Definition (CRD) zum Sammeln von Systemprotokollen und Debugging-Informationen verwenden.

F: Ist die Unterstützung für Netzwerkrichtlinien im automatischen EKS-Modus standardmäßig aktiviert? A: Vorerst muss die Unterstützung von Netzwerkrichtlinien explizit über die VPC CNI-Add-On-Konfiguration aktiviert werden. Nach der Aktivierung können Sie die standardmäßigen Kubernetes-Netzwerkrichtlinien verwenden.

F: Sollte ich IRSA oder Pod Identity mit EKS Auto Mode verwenden? A: Obwohl beide unterstützt werden, ist Pod Identity der empfohlene Ansatz im EKS-Automatikmodus, da er bereits das Pod

Identity Security Agent-Add-on enthält und erweiterte Sicherheitsfunktionen und eine vereinfachte Verwaltung bietet.

F: Kann ich aws-auth weiterhin ConfigMap im automatischen EKS-Modus verwenden? A: Das `aws-auth` ConfigMap ist eine veraltete Funktion. Es wird empfohlen, den Standardansatz der API-basierten Authentifizierung zu verwenden, um die Sicherheit zu erhöhen und die Zugriffsverwaltung zu vereinfachen.

F: Wie kann ich Sicherheitsereignisse im automatischen EKS-Modus überwachen? A: Der EKS Auto Mode lässt sich in mehrere Überwachungslösungen integrieren GuardDuty, darunter CloudWatch, und CloudTrail. GuardDuty bietet eine verbesserte Runtime-Sicherheitsüberwachung speziell für EKS-Workloads.

F: Wie sammle ich Protokolle von EKS-Automodus-Knoten? A: Verwenden Sie die NodeDiagnostic CRD, die Protokolle automatisch in einen S3-Bucket hochlädt. Sie können CloudWatch Container Insights und AWS Distro auch für OpenTelemetry verwenden.

> **Note**
>
> Dieser FAQ-Bereich wird regelmäßig aktualisiert, da dem EKS Auto Mode neue Funktionen hinzugefügt werden und wir häufig gestellte Fragen von Kunden erhalten.

Identitäts- und Zugriffsverwaltung

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Identity and Access Management (IAM) ist ein AWS-Service, der zwei wesentliche Funktionen erfüllt: Authentifizierung und Autorisierung. Die Authentifizierung beinhaltet die Überprüfung einer Identität, wohingegen die Autorisierung die Aktionen regelt, die von AWS-Ressourcen ausgeführt werden können. Innerhalb von AWS kann eine Ressource ein anderer AWS-Service sein, z. B. EC2, oder ein AWS-Prinzipal wie ein IAM-Benutzer oder eine IAM-Rolle. Die Regeln für die Aktionen, die eine Ressource ausführen darf, werden als IAM-Richtlinien ausgedrückt.

Steuerung des Zugriffs auf EKS-Cluster

Das Kubernetes-Projekt unterstützt eine Vielzahl verschiedener Strategien zur Authentifizierung von Anfragen an den Kube-Apiserver-Service, z. B. Bearer-Token, X.509-Zertifikate, OIDC usw. EKS bietet derzeit native Unterstützung für die Webhook-Token-Authentifizierung, Dienstkonto-Token und seit dem 21. Februar 2021 auch die OIDC-Authentifizierung.

Die Webhook-Authentifizierungsstrategie ruft einen Webhook auf, der Inhaber-Token verifiziert. Auf EKS werden diese Bearer-Token von der AWS-CLI oder dem aws-iam-authenticatorClient generiert, wenn Sie `kubectl` Befehle ausführen. Wenn Sie Befehle ausführen, wird das Token an den Kube-Apiserver übergeben, der es an den Authentifizierungs-Webhook weiterleitet. Wenn die Anfrage wohlgeformt ist, ruft der Webhook eine vorsignierte URL auf, die in den Hauptteil des Tokens eingebettet ist. Diese URL validiert die Signatur der Anfrage und gibt Informationen über den Benutzer zurück, z. B. das Konto des Benutzers, Arn, und UserId an den Kube-Apiserver.

Um ein Authentifizierungstoken manuell zu generieren, geben Sie den folgenden Befehl in ein Terminalfenster ein:

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

Die Ausgabe sollte wie folgt aussehen:

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

Sie können ein Token auch programmgesteuert abrufen. Im Folgenden finden Sie ein in Go geschriebenes Beispiel:

```
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)
}
```

Die Ausgabe sollte so aussehen:

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

Jedes Token beginnt mit, `k8s-aws-v1.` gefolgt von einer Base64-kodierten Zeichenfolge. Wenn die Zeichenfolge dekodiert wird, sollte sie etwa so aussehen:

```
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
```

Das Token besteht aus einer vorsignierten URL, die einen Amazon-Berechtigungsnachweis und eine Signatur enthält. Weitere Informationen finden Sie unter https://docs.aws.amazon.com/STS/ latest/ APIReference/API _ GetCallerIdentity .html.

Das Token hat eine Gültigkeitsdauer (TTL) von 15 Minuten. Danach muss ein neues Token generiert werden. Dies erfolgt automatisch, wenn Sie einen Client verwenden`kubectl`, z. B. wenn Sie das

Kubernetes-Dashboard verwenden, müssen Sie ein neues Token generieren und sich jedes Mal neu authentifizieren, wenn das Token abläuft.

Sobald die Identität des Benutzers durch den AWS IAM-Service authentifiziert wurde, liest der Kube-Apiserver die `aws-auth` ConfigMap im `kube-system` Namespace, um die RBAC-Gruppe zu bestimmen, die dem Benutzer zugeordnet werden soll. Das `aws-auth` ConfigMap wird verwendet, um eine statische Zuordnung zwischen IAM-Prinzipalen, d. h. IAM-Benutzern und -Rollen, und Kubernetes-RBAC-Gruppen zu erstellen. RBAC-Gruppen können in Kubernetes oder referenziert werden. RoleBindings ClusterRoleBindings Sie ähneln IAM-Rollen insofern, als sie eine Reihe von Aktionen (Verben) definieren, die für eine Sammlung von Kubernetes-Ressourcen (Objekten) ausgeführt werden können.

CloudWatch Abfrage, die Benutzern hilft, Clients zu identifizieren, die Anfragen an den globalen STS-Endpunkt senden

Führen Sie die folgende CloudWatch Abfrage aus, um den STS-Endpunkt zu ermitteln. Wenn stsendpoint „sts.amazonaws.com“ entspricht, handelt es sich um einen globalen STS-Endpunkt. Wenn stsendpoint gleich „sts“ ist. <region>.amazonaws.com“, dann handelt es sich um einen regionalen STS-Endpunkt.

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

Cluster-Zugriffsmanager

Cluster Access Manager, heute die bevorzugte Methode zur Verwaltung des Zugriffs von AWS IAM-Prinzipalen auf Amazon EKS-Cluster, ist eine Funktion der AWS-API und eine optionale Funktion für EKS v1.23 und spätere Cluster (neu oder bereits vorhanden). Es vereinfacht die Identitätszuweisung zwischen AWS IAM und KubernetesRBACs, sodass Sie nicht zwischen AWS und Kubernetes wechseln APIs oder das `aws-auth` ConfigMap für die Zugriffsverwaltung bearbeiten müssen, wodurch der Betriebsaufwand reduziert und Fehlkonfigurationen behoben werden können. Das Tool ermöglicht es Clusteradministratoren auch, `cluster-admin` Berechtigungen zu widerrufen oder zu verfeinern, die dem AWS IAM-Prinzipal, der zur Erstellung des Clusters verwendet wurde, automatisch erteilt wurden.

Diese API basiert auf zwei Konzepten:

- Zugriffseinträge: Eine Cluster-Identität, die direkt mit einem AWS IAM-Prinzipal (Benutzer oder Rolle) verknüpft ist, der sich bei einem Amazon EKS-Cluster authentifizieren darf.
- Zugriffsrichtlinien: Sind Amazon EKS-spezifische Richtlinien, die die Autorisierung für einen Access Entry zur Durchführung von Aktionen im Amazon EKS-Cluster bereitstellen.

Beim Start unterstützt Amazon EKS nur vordefinierte und von AWS verwaltete Richtlinien. Zugriffsrichtlinien sind keine IAM-Entitäten und werden von Amazon EKS definiert und verwaltet.

Cluster Access Manager ermöglicht die Kombination von Upstream-RBAC mit Zugriffsrichtlinien, die Kubernetes AuthZ-Entscheidungen in Bezug auf API-Serveranfragen unterstützen und zulassen (aber nicht verweigern). Eine Ablehnungsentscheidung wird getroffen, wenn sowohl der vorgelagerte RBAC- als auch der Amazon EKS-Autorisierer das Ergebnis einer Anfragebewertung nicht bestimmen können.

Mit dieser Funktion unterstützt Amazon EKS drei Authentifizierungsmodi:

1. `CONFIG_MAP`um `aws-auth` configMap weiterhin ausschließlich zu verwenden.
2. `API_AND_CONFIG_MAP`um authentifizierte IAM-Prinzipale sowohl aus EKS Access Entry als auch aus der `aws-auth` configMap zu beziehen APIs und die Access-Einträge zu priorisieren. Ideal, um bestehende `aws-auth` Berechtigungen auf Access Entries zu migrieren.
3. `API`um sich ausschließlich auf EKS Access Entry zu verlassen APIs. Dies ist der neue empfohlene Ansatz.

Zunächst können Clusteradministratoren Amazon EKS-Cluster erstellen oder aktualisieren, indem sie die bevorzugte Authentifizierung auf `API_AND_CONFIG_MAP` oder `API` -methode festlegen und Zugriffseinträge definieren, um den Zugriff auf die gewünschten AWS IAM-Prinzipale zu gewähren.

```
$ 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
```

Der obige Befehl ist ein Beispiel für die Erstellung eines Amazon EKS-Clusters bereits ohne die Administratorrechte des Clustererstellers.

Es ist möglich, die Amazon EKS-Clusterkonfiguration mithilfe des `update-cluster-config` Befehls zu aktualisieren, um den `API` AuthenticationMode zu aktivieren. Um dies auf vorhandenen Clustern zu tun, müssen `CONFIG_MAP` Sie zuerst auf `API_AND_CONFIG_MAP` und dann auf aktualisieren. `API` Diese Operationen können nicht rückgängig gemacht werden, was bedeutet, dass es nicht möglich ist, von zu `API_AND_CONFIG_MAP` oder `CONFIG_MAP` und auch nicht von `API` zu zu wechseln. `API_AND_CONFIG_MAP` `CONFIG_MAP`

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

Die API unterstützt Befehle zum Hinzufügen und Widerrufen des Zugriffs auf den Cluster sowie zum Überprüfen der vorhandenen Zugriffsrichtlinien und Zugriffseinträge für den angegebenen Cluster. Die Standardrichtlinien werden RBACs wie folgt so erstellt, dass sie zu Kubernetes passen.

EKS-Zugriffsrichtlinie	Kubernetes RBAC
Amazon EKSCluster AdminPolicy	Cluster-Administrator
EKSAdminAmazon-Richtlinie	Admin.
EKSEditAmazon-Richtlinie	bearbeiten
EKSViewAmazon-Richtlinie	Ansicht

```
$ 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": []
}
```

Es sind keine Zugriffseinträge verfügbar, wenn der Cluster ohne die Administratorberechtigung für den Clustererersteller erstellt wird. Dies ist der einzige Eintrag, der standardmäßig erstellt wird.

Der **aws-auth** ConfigMap (veraltet)

Eine Möglichkeit, die Kubernetes-Integration mit der AWS-Authentifizierung durchzuführen, ist die `aws-auth` ConfigMap, die sich im Namespace befindet. `kube-system` Es ist verantwortlich für die Zuordnung der AWS-IAM-Authentifizierung (Benutzer, Gruppen und Rollen) zur rollenbasierten Zugriffskontrolle (RBAC) von Kubernetes. Der `aws-auth` ConfigMap wird während der Bereitstellungsphase automatisch in Ihrem Amazon EKS-Cluster erstellt. Es wurde ursprünglich erstellt, um Knoten den Beitritt zu Ihrem Cluster zu ermöglichen, aber wie bereits erwähnt, können Sie damit ConfigMap auch RBACs Zugriff auf IAM-Prinzipale gewähren.

Um die Ihres Clusters zu überprüfen `aws-auth` ConfigMap, können Sie den folgenden Befehl verwenden.

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

Dies ist ein Beispiel für eine Standardkonfiguration von `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
```

Die Hauptsitzung davon ConfigMap befindet sich unter dem `data` `mapRoles` Block, der im Wesentlichen aus 3 Parametern besteht.

- Gruppen: Das Kubernetes, dem die IAM-Rolle zugeordnet werden group/groups soll. Dies kann eine Standardgruppe oder eine benutzerdefinierte Gruppe sein, die in einem oder angegeben ist. `clusterrolebinding` `rolebinding` Im obigen Beispiel haben wir nur Systemgruppen deklariert.
- rolearn: Der ARN der AWS-IAM-Rolle wird der Kubernetes-Gruppe (n) hinzugefügt, wobei das folgende Format verwendet wird. `arn:<PARTITION>:iam::<AWS_ACCOUNT_ID>:role/role-name`
- username: Der Benutzername in Kubernetes, der der AWS IAM-Rolle zugeordnet werden soll. Dies kann ein beliebiger benutzerdefinierter Name sein.

Es ist auch möglich, Berechtigungen für AWS IAM-Benutzer zuzuordnen und einen neuen Konfigurationsblock für`mapUsers`, unter`data`, zu definieren `aws-auth`ConfigMap, der den rolearn-Parameter für userarn ersetzt. Als Best Practice wird jedoch immer empfohlen, stattdessen den Benutzer zu verwenden. `mapRoles`

Um Berechtigungen zu verwalten, können Sie das `aws-auth` ConfigMap Hinzufügen oder Entfernen des Zugriffs auf Ihren Amazon EKS-Cluster bearbeiten. Es ist zwar möglich, sie `aws-auth` ConfigMap manuell zu bearbeiten, es wird jedoch empfohlen, Tools wie zu verwenden`eksctl`, da es sich um eine sehr sensible Konfiguration handelt und eine ungenaue Konfiguration Sie außerhalb Ihres Amazon EKS-Clusters sperren kann. Weitere Informationen finden Sie unten im Unterabschnitt Tools verwenden, um Änderungen an ConfigMap aws-auth vorzunehmen.

Vorteile gegenüber einer basierten Zugriffsverwaltung ConfigMap

1. Geringeres Risiko von Fehlkonfigurationen: Direktes API-basiertes Management verhindert häufige Fehler im Zusammenhang mit manueller Bearbeitung. ConfigMap Dies trägt dazu

bei, versehentliche Löschungen oder Syntaxfehler zu verhindern, die Benutzer vom Cluster ausschließen könnten.

2. Verbessertes Prinzip der geringsten Rechte: Macht die Cluster-Admin-Rechte für die Identität des Clustererstellers überflüssig und ermöglicht eine detailliertere und angemessenere Zuweisung von Berechtigungen. Sie können sich dafür entscheiden, diese Berechtigung für Break-Glass-Anwendungsfälle hinzuzufügen.
3. Verbessertes Sicherheitsmodell: Bietet eine integrierte Überprüfung von Zugriffseinträgen, bevor sie angewendet werden. Darüber hinaus bietet es eine engere Integration mit AWS IAM für die Authentifizierung.
4. Optimierter Betrieb: Bietet eine intuitivere Möglichkeit, Berechtigungen mithilfe von AWS-nativen Tools zu verwalten.

Empfehlungen für den Cluster-Zugriff

Kombinieren Sie IAM Identity Center mit der CAM-API

- Vereinfachtes Management: Durch die Verwendung der Cluster Access Management API in Verbindung mit IAM Identity Center können Administratoren den EKS-Clusterzugriff zusammen mit anderen AWS-Services verwalten, sodass sie nicht zwischen verschiedenen Schnittstellen wechseln oder ConfigMaps manuell bearbeiten müssen.
- Verwenden Sie Zugriffseinträge, um die Kubernetes-Berechtigungen von IAM-Prinzipalen außerhalb des Clusters zu verwalten. Sie können den Zugriff auf den Cluster mithilfe der EKS-API, der AWS-Befehlszeilenschnittstelle, AWS SDKs CloudFormation, AWS und der AWS-Managementkonsole hinzufügen und verwalten. Für die Benutzerverwaltung können also die gleichen Tools verwendet werden wie für die Clustererstellung.
- Nutzen Sie die Automatisierung, wie in diesem Beispiel gezeigt, für die Bereitstellung von Clustern mit AWS IAM Identity Center als IdP mit CAM-API als Einstiegspunkt.
- Granulare Kubernetes-Berechtigungen können angewendet werden, indem Kubernetes-Benutzern oder Gruppen IAM-Prinzipale zugeordnet werden, die mit SSO-Identitäten über Zugriffseinträge und Zugriffsrichtlinien verknüpft sind.
- Folgen Sie zunächst den Anweisungen Authentifizierungsmodus zur Verwendung von Zugriffseinträgen ändern und anschließend Vorhandene aws-auth-Einträge zu Zugriffseinträgen migrieren. ConfigMap

Machen Sie den EKS-Cluster-Endpunkt privat

Wenn Sie einen EKS-Cluster bereitstellen, ist der API-Cluster-Endpunkt standardmäßig auf öffentlich gesetzt, d. h. auf ihn kann über das Internet zugegriffen werden. Obwohl der Endpunkt über das Internet zugänglich ist, gilt er dennoch als sicher, da alle API-Anfragen von IAM authentifiziert und anschließend von Kubernetes RBAC autorisiert werden müssen. Wenn Ihre Unternehmenssicherheitsrichtlinie jedoch vorschreibt, dass Sie den Zugriff auf die API aus dem Internet einschränken oder Sie daran hindern, Datenverkehr außerhalb der Cluster-VPC weiterzuleiten, können Sie:

- Konfigurieren Sie den EKS-Cluster-Endpunkt so, dass er privat ist. Weitere Informationen zu diesem Thema finden Sie unter Ändern des Cluster-Endpunktzugriffs.
- Lassen Sie den Cluster-Endpunkt öffentlich und geben Sie an, welche CIDR-Blöcke mit dem Cluster-Endpunkt kommunizieren können. Bei den Blöcken handelt es sich praktisch um einen Satz öffentlicher IP-Adressen auf der Whitelist, die auf den Cluster-Endpunkt zugreifen dürfen.
- Konfigurieren Sie den öffentlichen Zugriff mit einer Reihe von CIDR-Blöcken auf der Whitelist und setzen Sie den privaten Endpunktzugriff auf aktiviert. Dadurch wird der öffentliche Zugriff von einem bestimmten öffentlichen Bereich aus ermöglicht und IPs gleichzeitig der gesamte Netzwerkverkehr zwischen den Kubelets (Workers) und der Kubernetes-API über das Cross-Konto erzwungenENIs , das bei der Bereitstellung der Steuerungsebene in der Cluster-VPC bereitgestellt wird.

Verwenden Sie kein Dienstkonto-Token für die Authentifizierung

Ein Dienstkonto-Token ist ein langlebiger, statischer Berechtigungsnachweis. Wenn es kompromittiert wird, verloren geht oder gestohlen wird, kann ein Angreifer möglicherweise alle mit diesem Token verbundenen Aktionen ausführen, bis das Dienstkonto gelöscht wird. Manchmal müssen Sie möglicherweise eine Ausnahme für Anwendungen gewähren, die die Kubernetes-API von außerhalb des Clusters nutzen müssen, z. B. eine CI/CD-Pipeline-Anwendung. Wenn solche Anwendungen auf einer AWS-Infrastruktur wie EC2-Instances ausgeführt werden, sollten Sie erwägen, ein Instance-Profil zu verwenden und dieses einer Kubernetes-RBAC-Rolle zuzuordnen.

Verwenden Sie den Zugriff mit den geringsten Rechten auf AWS-Ressourcen

Einem IAM-Benutzer müssen keine Rechte für AWS-Ressourcen zugewiesen werden, um auf die Kubernetes-API zugreifen zu können. Wenn Sie einem IAM-Benutzer Zugriff auf einen EKS-Cluster

gewähren müssen, erstellen Sie `aws-auth` ConfigMap für diesen Benutzer einen Eintrag, der einer bestimmten Kubernetes-RBAC-Gruppe zugeordnet ist.

Entfernen Sie die Cluster-Admin-Berechtigungen aus dem Cluster-Ersteller-Prinzipal

Standardmäßig werden Amazon EKS-Cluster mit einer dauerhaften `cluster-admin` Berechtigung erstellt, die an den Cluster-Erstellerprinzipal gebunden ist. Mit der Cluster Access Manager-API ist es möglich, Cluster zu erstellen, ohne dass diese Berechtigung den Authentifizierungsmodus `--access-config bootstrapClusterCreatorAdminPermissions` auf`false`, bei der Verwendung `API_AND_CONFIG_MAP` oder den `API` Authentifizierungsmodus setzt. Das Widerrufen dieses Zugriffs wurde als bewährte Methode angesehen, um unerwünschte Änderungen an der Clusterkonfiguration zu vermeiden. Das Verfahren zum Widerrufen dieses Zugriffs folgt demselben Verfahren zum Widerrufen aller anderen Zugriffe auf den Cluster.

Die API bietet Ihnen die Flexibilität, nur einen IAM-Prinzipal von einer Zugriffsrichtlinie zu trennen, in diesem Fall der. `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
```

Oder Sie entfernen den mit der Berechtigung verknüpften Zugriffseintrag vollständig. `cluster-admin`

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

{
    "accessEntries": []
}

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

Dieser Zugriff kann erneut gewährt werden, falls dies bei einem Zwischenfall, einem Notfall oder einem Breakglasszenario erforderlich ist, bei dem ansonsten kein Zugriff auf den Cluster möglich ist.

Wenn der Cluster immer noch mit der `CONFIG_MAP` Authentifizierungsmethode konfiguriert `aws-auth` ConfigMap ist, sollte allen zusätzlichen Benutzern Zugriff auf den Cluster gewährt werden. Nach der `aws-auth` ConfigMap Konfiguration kann die Rolle, die der Entität zugewiesen wurde, die den Cluster erstellt hat, gelöscht und nur dann neu erstellt werden, wenn ein Vorfall, ein Notfall oder ein Breakglass-Szenario vorliegt oder wenn der Cluster beschädigt `aws-auth` ConfigMap ist und der Cluster anderweitig nicht zugänglich ist. Dies kann besonders in Produktionsclustern nützlich sein.

Verwenden Sie IAM-Rollen, wenn mehrere Benutzer identischen Zugriff auf den Cluster benötigen

Anstatt einen Eintrag für jeden einzelnen IAM-Benutzer zu erstellen, ermöglichen Sie diesen Benutzern, eine IAM-Rolle anzunehmen und diese Rolle einer Kubernetes-RBAC-Gruppe zuzuordnen. Dies wird einfacher zu verwalten sein, insbesondere wenn die Anzahl der Benutzer, die Zugriff benötigen, zunimmt.

> **Important**
>
> Beim Zugriff auf den EKS-Cluster mit zugeordneter IAM-Entität wird der beschriebene Benutzername im Benutzerfeld des Kubernetes-Audit-Logs aufgezeichnet. `aws-auth` ConfigMap Wenn Sie eine IAM-Rolle verwenden, werden die tatsächlichen Benutzer, die diese Rolle annehmen, nicht aufgezeichnet und können nicht geprüft werden.

Wenn Sie `aws-auth` configMap weiterhin als Authentifizierungsmethode verwenden und einer IAM-Rolle die RBAC-Berechtigungen von K8 zuweisen, sollten Sie\ {{}} in Ihren Benutzernamen aufnehmen. SessionName Auf diese Weise zeichnet das Audit-Protokoll den Sitzungsnamen auf, sodass Sie zusammen mit dem Protokoll nachverfolgen können, wer der tatsächliche Benutzer diese Rolle übernommen hat. CloudTrail

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

In Kubernetes 1.20 und höher ist diese Änderung nicht mehr erforderlich, da sie dem Kubernetes-Audit-Log hinzugefügt `user.extra.sessionName.0` wurde.

Verwenden Sie bei der Erstellung und den Zugriff mit den geringsten Rechten RoleBindings ClusterRoleBindings

Wie der vorherige Punkt über die Gewährung des Zugriffs auf AWS-Ressourcen, RoleBindings und ClusterRoleBindings sollte nur die Berechtigungen enthalten, die für die Ausführung einer bestimmten Funktion erforderlich sind. Vermeiden Sie die Verwendung `["*"]` in Ihren Rollen und ClusterRoles sofern dies nicht unbedingt erforderlich ist. Wenn Sie sich nicht sicher sind, welche Berechtigungen Sie zuweisen sollen, sollten Sie ein Tool wie audit2rbac verwenden, um automatisch Rollen und Bindungen auf der Grundlage der beobachteten API-Aufrufe im Kubernetes Audit Log zu generieren.

Erstellen Sie einen Cluster mithilfe eines automatisierten Prozesses

Wie in den vorherigen Schritten beschrieben, werden beim Erstellen eines Amazon EKS-Clusters dem Benutzer `API_AND_CONFIG_MAP` oder der Rolle der IAM-Entität, z. B. einem Verbundbenutzer, der den Cluster erstellt, automatisch `cluster-admin` Berechtigungen in der RBAC-Konfiguration des Clusters gewährt, wenn nicht der Verwendungs- oder `API` Authentifizierungsmodus verwendet wird und die Delegierung von `system:masters` Berechtigungen an den Cluster-Ersteller nicht deaktiviert wird. Es ist sogar eine bewährte Methode, diese Berechtigung zu entfernen, wie hier beschrieben, wenn Sie die `CONFIG_MAP` Authentifizierungsmethode verwenden, und sich darauf verlassen `aws-auth` ConfigMap, dass dieser Zugriff nicht widerrufen werden kann. Daher ist es eine gute Idee, den Cluster mit einer Pipeline zur Infrastrukturautomatisierung zu erstellen, die an eine dedizierte IAM-Rolle gebunden ist, ohne dass Rechte von anderen Benutzern oder Entitäten übernommen werden müssen, und regelmäßig die Berechtigungen und Richtlinien dieser Rolle zu überprüfen und zu überprüfen, wer Zugriff hat, um die Pipeline auszulösen. Außerdem sollte diese Rolle nicht zur Durchführung von Routineaktionen auf dem Cluster verwendet werden. Sie sollte ausschließlich für Aktionen auf Clusterebene verwendet werden, die durch die Pipeline ausgelöst werden, beispielsweise über SCM-Codeänderungen.

Erstellen Sie den Cluster mit einer dedizierten IAM-Rolle

Wenn Sie einen Amazon EKS-Cluster erstellen, werden dem Benutzer oder der Rolle der IAM-Entität, z. B. einem Verbundbenutzer, der den Cluster erstellt, automatisch `system:masters` Berechtigungen in der RBAC-Konfiguration des Clusters gewährt. Dieser Zugriff kann nicht entfernt werden und wird nicht über den verwaltet. `aws-auth` ConfigMap Daher empfiehlt es sich, den Cluster mit einer eigenen IAM-Rolle zu erstellen und regelmäßig zu überprüfen, wer diese Rolle übernehmen kann. Diese Rolle sollte nicht zur Durchführung von Routineaktionen auf dem Cluster verwendet werden. Stattdessen sollte zu diesem Zweck zusätzlichen Benutzern über den Zugriff auf den `aws-auth` ConfigMap Cluster gewährt werden. Nach der `aws-auth` ConfigMap Konfiguration

sollte die Rolle gesichert und nur im temporären Modus mit erhöhten Rechten (Breakglass) für Szenarien verwendet werden, in denen ansonsten kein Zugriff auf den Cluster möglich ist. Dies kann besonders in Clustern nützlich sein, für die kein direkter Benutzerzugriff konfiguriert ist.

Überprüfen Sie regelmäßig den Zugriff auf den Cluster

Wer Zugriff benötigt, wird sich wahrscheinlich im Laufe der Zeit ändern. Planen Sie, sie regelmäßig `aws-auth` ConfigMap zu überprüfen, um festzustellen, wem Zugriff gewährt wurde und welche Rechte ihnen zugewiesen wurden. Sie können auch Open-Source-Tools wie oder rbac-Lookup verwenden kubectl-who-can, um die Rollen zu untersuchen, die an ein bestimmtes Dienstkonto, einen bestimmten Benutzer oder eine bestimmte Gruppe gebunden sind. Wir werden uns eingehender mit diesem Thema befassen, wenn wir zum Abschnitt über Auditing kommen. Weitere Ideen finden Sie in diesem Artikel der NCC Group.

Wenn Sie sich auf **`aws-auth`** configMap verlassen, verwenden Sie Tools, um Änderungen vorzunehmen.

Ein falsch formatiertes aws-auth ConfigMap kann dazu führen, dass Sie den Zugriff auf den Cluster verlieren. Wenn Sie Änderungen an der vornehmen müssen, verwenden Sie ein Tool. ConfigMap

eksctl Die `eksctl` CLI enthält einen Befehl zum Hinzufügen von Identitätszuordnungen zur aws-auth. ConfigMap

CLI-Hilfe anzeigen:

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

Überprüfen Sie die Identitäten, die Ihrem Amazon EKS-Cluster zugeordnet sind.

```
$ 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
```

Machen Sie eine IAM-Rolle zu einem Cluster-Administrator:

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

```
...
```

Weitere Informationen finden Sie in den Dokumenten `eksctl`

aws-auth von keikoproj

`aws-auth`von keikoproj enthält sowohl eine CLI- als auch eine Go-Bibliothek.

Laden Sie die CLI-Hilfe herunter und zeigen Sie sie an:

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

Alternativ können Sie die Installation `aws-auth` mit dem Krew-Plugin-Manager für Kubectl durchführen.

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

Weitere Informationen, einschließlich der Go-Bibliothek, finden Sie in den GitHub aws-auth-Dokumenten unter.

AWS-IAM-Authentifikator-CLI

Das `aws-iam-authenticator` Projekt enthält eine CLI zur Aktualisierung derConfigMap.

Laden Sie eine Version herunter am GitHub.

Fügen Sie Clusterberechtigungen zu einer IAM-Rolle hinzu:

```
$ ./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
...
```

Alternative Ansätze für Authentifizierung und Zugriffsmanagement

IAM ist zwar die bevorzugte Methode zur Authentifizierung von Benutzern, die Zugriff auf einen EKS-Cluster benötigen, es ist jedoch möglich, einen OIDC-Identitätsanbieter zu verwenden, z. B. einen Authentifizierungsproxy und Kubernetes-Identitätswechsel zu GitHub verwenden. Beiträge zu zwei solchen Lösungen wurden im AWS Open Source-Blog veröffentlicht:

- Authentifizierung bei EKS mithilfe von GitHub Anmeldeinformationen mit Teleport
- Konsistente OIDC-Authentifizierung über mehrere EKS-Cluster hinweg mit kube-oidc-proxy

> **Important**
>
> EKS unterstützt nativ die OIDC-Authentifizierung ohne Verwendung eines Proxys. Weitere Informationen finden Sie im Launch-Blog Introducing OIDC Identity Provider Authentication for Amazon EKS. Ein Beispiel für die Konfiguration von EKS mit Dex, einem beliebten Open-Source-OIDC-Anbieter mit Konnektoren für eine Vielzahl verschiedener Authentifizierungsmethoden, finden Sie unter Verwenden von Dex & dex-k8s-authenticator zur Authentifizierung bei Amazon EKS. Wie in den Blogs beschrieben, werden die Benutzer, die username/group von einem OIDC-Anbieter authentifiziert wurden, im Kubernetes-Audit-Log angezeigt.

Sie können AWS SSO auch verwenden, um AWS mit einem externen Identitätsanbieter, z. B. Azure AD, zu verbinden. Wenn Sie sich dafür entscheiden, bietet AWS CLI v2.0 eine Option zum Erstellen eines benannten Profils, mit dem Sie ganz einfach eine SSO-Sitzung mit Ihrer aktuellen CLI-Sitzung verknüpfen und eine IAM-Rolle übernehmen können. Beachten Sie, dass Sie vor der Ausführung eine Rolle übernehmen müssen, `kubectl` da die IAM-Rolle verwendet wird, um die Kubernetes-RBAC-Gruppe des Benutzers zu ermitteln.

Identitäten und Anmeldeinformationen für EKS-Pods

Bestimmte Anwendungen, die in einem Kubernetes-Cluster ausgeführt werden, benötigen die Erlaubnis, die Kubernetes-API aufzurufen, damit sie ordnungsgemäß funktionieren. Beispielsweise muss der AWS Load Balancer Controller in der Lage sein, die Endpunkte eines Services aufzulisten. Der Controller muss außerdem in der Lage sein, AWS aufzurufen APIs , um eine ALB bereitzustellen und zu konfigurieren. In diesem Abschnitt werden wir die bewährten Methoden für die Zuweisung von Rechten und Privilegien an Pods untersuchen.

Kubernetes-Dienstkonten

Ein Dienstkonto ist ein besonderer Objekttyp, mit dem Sie einem Pod eine Kubernetes-RBAC-Rolle zuweisen können. Für jeden Namespace innerhalb eines Clusters wird automatisch ein Standarddienstkonto erstellt. Wenn Sie einen Pod in einem Namespace bereitstellen, ohne auf ein bestimmtes Dienstkonto zu verweisen, wird das Standarddienstkonto für diesen Namespace automatisch dem Pod zugewiesen, und das Secret, d. h. das Dienstkonto-Token (JWT) für dieses Dienstkonto, wird dem Pod als Volume hinzugefügt. `/var/run/secrets/kubernetes.io/serviceaccount` Durch die Dekodierung des Dienstkonto-Tokens in diesem Verzeichnis werden die folgenden Metadaten angezeigt:

```
{
  "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"
}
```

Das Standarddienstkonto hat die folgenden Berechtigungen für die Kubernetes-API.

```
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
```

Diese Rolle berechtigt nicht authentifizierte und authentifizierte Benutzer zum Lesen von API-Informationen und gilt als sicher, wenn sie öffentlich zugänglich ist.

Wenn eine Anwendung, die in einem Pod ausgeführt wird, Kubernetes aufruft APIs, muss dem Pod ein Dienstkonto zugewiesen werden, das ihm ausdrücklich die Erlaubnis erteilt, diese aufzurufen. APIs Ähnlich wie bei den Richtlinien für den Benutzerzugriff sollte die Rolle oder die ClusterRole Bindung an ein Dienstkonto auf die API-Ressourcen und Methoden beschränkt sein, die die Anwendung benötigt, um zu funktionieren, und auf nichts anderes. Um ein nicht standardmäßiges Dienstkonto zu verwenden, setzen Sie einfach das `spec.serviceAccountName` Feld eines Pods auf den Namen des Dienstkontos, das Sie verwenden möchten. Weitere Informationen zum Erstellen von Dienstkonten finden Sie unter https://kubernetes. io/docs/reference/access-authn-authz/rbac/#. service-account-permissions

> **Note**
>
> Vor Kubernetes 1.24 erstellte Kubernetes automatisch ein Geheimnis für jedes Dienstkonto. Dieses Geheimnis wurde unter/in den Pod eingebunden var/run/secrets/kubernetes.io/ serviceaccount und vom Pod zur Authentifizierung beim Kubernetes-API-Server verwendet. In Kubernetes 1.24 wird ein Dienstkonto-Token dynamisch generiert, wenn der Pod ausgeführt wird, und ist standardmäßig nur für eine Stunde gültig. Ein Geheimnis für das Dienstkonto wird nicht erstellt. Wenn Sie über eine Anwendung verfügen, die außerhalb des Clusters ausgeführt wird und sich bei der Kubernetes-API authentifizieren muss, z. B. Jenkins, müssen Sie einen geheimen Typ `kubernetes.io/service-account-token` zusammen mit einer Anmerkung erstellen, die auf das Dienstkonto verweist, z. B. `metadata.annotations.kubernetes.io/service-account.name: <SERVICE_ACCOUNT_NAME>` Auf diese Weise erstellte Geheimnisse laufen nicht ab.

IAM-Rollen für Dienstkonten (IRSA)

IRSA ist eine Funktion, mit der Sie einem Kubernetes-Dienstkonto eine IAM-Rolle zuweisen können. Es nutzt eine Kubernetes-Funktion, die als Service Account Token Volume Projection bekannt

ist. Wenn Pods mit einem Dienstkonto konfiguriert sind, das auf eine IAM-Rolle verweist, ruft der Kubernetes-API-Server beim Start den öffentlichen OIDC-Discovery-Endpunkt für den Cluster auf. Der Endpunkt signiert kryptografisch das von Kubernetes ausgegebene OIDC-Token und das daraus resultierende Token wird als Volume bereitgestellt. Dieses signierte Token ermöglicht es dem Pod, die APIs mit AWS verknüpfte IAM-Rolle aufzurufen. Wenn eine AWS-API aufgerufen wird, SDKs ruft `sts:AssumeRoleWithWebIdentity` AWS auf. Nach der Überprüfung der Tokensignatur tauscht IAM das von Kubernetes ausgestellte Token gegen temporäre AWS-Rollenanmeldedaten aus.

Bei der Verwendung von IRSA ist es wichtig, AWS-SDK-Sitzungen wiederzuverwenden, um unnötige Aufrufe von AWS STS zu vermeiden.

Die Dekodierung des (JWT-) Tokens für IRSA erzeugt eine Ausgabe, die dem Beispiel ähnelt, das Sie unten sehen:

```
{
  "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"
}
```

Dieses spezielle Token gewährt S3 nur Leseberechtigungen für den Pod, indem es eine IAM-Rolle annimmt. Wenn die Anwendung versucht, aus S3 zu lesen, wird das Token gegen einen temporären Satz von IAM-Anmeldeinformationen ausgetauscht, der dem Folgenden ähnelt:

```
{
    "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"
    }
}
```

Ein mutierender Webhook, der als Teil der EKS-Steuerebene ausgeführt wird, fügt den AWS-Rollen-ARN und den Pfad zu einer Web-Identity-Token-Datei als Umgebungsvariablen in den Pod ein. Diese Werte können auch manuell angegeben werden.

```
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
```

Das Kubelet rotiert das projizierte Token automatisch, wenn es älter als 80% seiner gesamten TTL ist oder nach 24 Stunden. Die AWS SDKs sind dafür verantwortlich, das Token neu zu laden, wenn es rotiert. Weitere Informationen zu IRSA finden Sie unter https://docs.aws.amazon.com/eks/ latest/ userguide/iam - -technical-overview.html. roles-for-service-accounts

EKS-Pod-Identitäten

EKS Pod Identities ist eine auf der re:Invent 2023 eingeführte Funktion, mit der Sie einem Kubernetes-Servicekonto eine IAM-Rolle zuweisen können, ohne für jeden Cluster in Ihrem AWS-

Konto einen Open Id Connect (OIDC) -Identitätsanbieter (IDP) konfigurieren zu müssen. Um EKS Pod Identity verwenden zu können, müssen Sie einen Agenten bereitstellen, der als Pod auf jedem geeigneten Worker-Knoten ausgeführt wird. DaemonSet Dieser Agent wird Ihnen als EKS-Add-on zur Verfügung gestellt und ist eine Voraussetzung für die Nutzung der EKS Pod Identity-Funktion. Ihre Anwendungen müssen eine unterstützte Version des AWS-SDK verwenden, um diese Funktion nutzen zu können.

Wenn EKS-Pod-Identitäten für einen Pod konfiguriert sind, mountet und aktualisiert EKS ein Pod-Identitätstoken unter`/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token`. Dieses Token wird vom AWS-SDK für die Kommunikation mit dem EKS Pod Identity Agent verwendet, der das Pod-Identitätstoken und die IAM-Rolle des Agenten verwendet, um temporäre Anmeldeinformationen für Ihre Pods zu erstellen, indem er die AssumeRoleForPodIdentityAPI aufruft. Das an Ihre Pods übermittelte Pod-Identitätstoken ist ein JWT, das von Ihrem EKS-Cluster ausgegeben und kryptografisch signiert wurde und entsprechende JWT-Ansprüche für die Verwendung mit EKS Pod Identities enthält.

Weitere Informationen zu EKS Pod Identities finden Sie in diesem Blog.

Sie müssen keine Änderungen an Ihrem Anwendungscode vornehmen, um EKS Pod Identities verwenden zu können. Unterstützte AWS-SDK-Versionen erkennen automatisch Anmeldeinformationen, die mit EKS Pod Identities zur Verfügung gestellt wurden, mithilfe der Anmeldeinformationsanbieterkette. Wie IRSA legt auch EKS-Pod-Identitäten Variablen in Ihren Pods fest, um ihnen mitzuteilen, wie sie AWS-Anmeldeinformationen finden können.

Arbeiten mit IAM-Rollen für EKS-Pod-Identitäten

- EKS-Pod-Identitäten können nur direkt eine IAM-Rolle übernehmen, die zu demselben AWS-Konto gehört wie der EKS-Cluster. Um auf eine IAM-Rolle in einem anderen AWS-Konto zuzugreifen, müssen Sie diese Rolle übernehmen, indem Sie ein Profil in Ihrer SDK-Konfiguration oder im Code Ihrer Anwendung konfigurieren.
- Wenn EKS Pod Identities für Service Accounts konfiguriert werden, muss die Person oder der Prozess, der die Pod Identity Association konfiguriert, über die `iam:PassRole` Berechtigung für diese Rolle verfügen.
- Jedem Dienstkonto kann über EKS Pod Identities nur eine IAM-Rolle zugeordnet sein. Sie können jedoch dieselbe IAM-Rolle mehreren Dienstkonten zuordnen.
- IAM-Rollen, die mit EKS-Pod-Identitäten verwendet werden, müssen es dem *`pods.eks.amazonaws.com`* Service Principal ermöglichen, sie zu übernehmen und Sitzungs-

Tags festzulegen. Im Folgenden finden Sie ein Beispiel für eine Vertrauensrichtlinie für Rollen, die es EKS Pod Identities ermöglicht, eine IAM-Rolle zu verwenden:

```
{
  "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 empfiehlt die Verwendung von Bedingungsschlüsseln, `aws:SourceOrgId` um sich vor dem dienstübergreifenden Problem Confused Deputy zu schützen. Im obigen Beispiel für eine Vertrauensrichtlinie für Rollen `ResourceOrgId` ist dies eine Variable, die der AWS-Organisations-ID der AWS-Organisation entspricht, zu der das AWS-Konto gehört. EKS übergibt einen Wert, der diesem Wert `aws:SourceOrgId` entspricht, wenn es eine Rolle mit EKS Pod Identities annimmt.

ABAC- und EKS-Pod-Identitäten

Wenn EKS Pod Identities eine IAM-Rolle annimmt, werden die folgenden Sitzungs-Tags festgelegt:

Sitzungs-Tag für EKS Pod Identities	Wert
Kubernetes-Namespace	Der Namespace, in dem der mit EKS Pod Identities verknüpfte Pod ausgeführt wird.

Sitzungs-Tag für EKS Pod Identities	Wert
kubernetes-service-account	Der Name des Kubernetes-Dienstkontos, das EKS Pod Identities zugeordnet ist
eks-cluster-arn	Der ARN des EKS-Clusters, z. `arn:${Partition}:eks:${Region}:${Account}:cluster/${ClusterName}` B. Der Cluster-ARN ist eindeutig, aber wenn ein Cluster gelöscht und in derselben Region mit demselben Namen innerhalb desselben AWS-Kontos neu erstellt wird, hat er denselben ARN.
eks-cluster-name	Name des EKS-Clusters. Bitte beachten Sie, dass die Namen der EKS-Cluster in Ihrem AWS-Konto und die Namen der EKS-Cluster in anderen AWS-Konten identisch sein können.
kubernetes-pod-name	Der Name des Pods in EKS.
kubernetes-pod-uid	Die UID des Pods in EKS.

Mit diesen Sitzungs-Tags können Sie Attribute Based Access Control (ABAC) verwenden, um nur bestimmten Kubernetes-Servicekonten Zugriff auf Ihre AWS-Ressourcen zu gewähren. Dabei ist es sehr wichtig zu verstehen, dass Kubernetes-Servicekonten nur innerhalb eines Namespace eindeutig sind und Kubernetes-Namespaces nur innerhalb eines EKS-Clusters eindeutig sind. Auf diese Sitzungs-Tags kann in AWS-Richtlinien mithilfe des `aws:PrincipalTag/<tag-key>` globalen Bedingungsschlüssels zugegriffen werden, z. B. `aws:PrincipalTag/eks-cluster-arn`

Wenn Sie beispielsweise nur einem bestimmten Dienstkonto Zugriff gewähren möchten, um auf eine AWS-Ressource in Ihrem Konto mit einer IAM- oder Ressourcenrichtlinie zuzugreifen, müssten Sie überprüfen `eks-cluster-arn` und `kubernetes-namespace` kennzeichnen sowie sicherstellen, dass nur Dienstkonten aus dem vorgesehenen Cluster Zugriff auf diese Ressource haben, da andere Cluster identische `kubernetes-service-accounts` und `kubernetes-namespaces` haben könnten. `kubernetes-service-account`

Dieses Beispiel für eine S3-Bucket-Richtlinie gewährt nur dann Zugriff auf Objekte in dem S3-Bucket, an den sie angehängt ist `kubernetes-service-accountkubernetes-namespace`, wenn,, `eks-`

`cluster-arn` alle ihre erwarteten Werte erfüllen, wobei der EKS-Cluster im AWS-Konto gehostet wird`111122223333`.

```
{
    "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"
                }
            }
        }
    ]
}
```

EKS-Pod-Identitäten im Vergleich zu IRSA

Sowohl EKS Pod Identities als auch IRSA sind bevorzugte Methoden, um temporäre AWS-Anmeldeinformationen für Ihre EKS-Pods bereitzustellen. Sofern Sie keine speziellen Anwendungsfälle für IRSA haben, empfehlen wir Ihnen, bei der Verwendung von EKS Pod Identities zu verwenden. Diese Tabelle hilft beim Vergleich der beiden Funktionen.

#	EKS-Pod-Identitäten	IRSA
Benötigen Sie die Erlaubnis, einen OIDC-IDP in Ihren AWS-Konten zu erstellen?	Nein	Ja

#	EKS-Pod-Identitäten	IRSA
Erfordert ein einzigartiges IDP-Setup pro Cluster	Nein	Ja
Legt relevante Sitzungs-Tags für die Verwendung mit ABAC fest	Ja	Nein
Erfordert einen iam: Check? PassRole	Ja	Nein
Verwendet AWS STS Quota von Ihrem AWS-Konto?	Nein	Ja
Kann auf andere AWS-Konten zugreifen	Indirekt mit Rollenverkettung	Direkt mit sts: AssumeRoleWithWebIdentity
kompatibel mit AWS SDKs	Ja	Ja
Erfordert Pod Identity Agent Daemonset auf Knoten?	Ja	Nein

Empfehlungen für Identitäten und Anmeldeinformationen für EKS-Pods

Aktualisieren Sie das aws-node-Daemonset für die Verwendung von IRSA

Derzeit ist der aws-node-Daemonset so konfiguriert, dass er eine den EC2-Instances zugewiesene Rolle verwendet, um sie Pods zuzuweisen. IPs Diese Rolle umfasst mehrere von AWS verwaltete Richtlinien, z. B. Amazoneks_CNI_Policy, die es allen Pods EC2ContainerRegistryReadOnly , die auf einem Knoten laufen, ermöglichen attach/detach ENIs, assign/unassign IP-Adressen zu verwenden oder Bilder aus ECR abzurufen. Da dies ein Risiko für Ihren Cluster darstellt, wird empfohlen, das aws-node-Daemonset so zu aktualisieren, dass es IRSA verwendet. Ein Skript dafür finden Sie im Repository für dieses Handbuch.

Der aws-node-Daemonset unterstützt EKS Pod Identities in den Versionen v1.15.5 und höher.

Beschränken Sie den Zugriff auf das Instanzprofil, das dem Worker-Knoten zugewiesen ist

Wenn Sie IRSA- oder EKS-Pod-Identitäten verwenden, wird die Anmeldeinformationskette des Pods aktualisiert, sodass zuerst IRSA- oder EKS-Pod-Identitäten verwendet werden. Der Pod kann jedoch weiterhin die Rechte des Instanzprofils erben, das dem Worker-Knoten zugewiesen ist. Für Pods, die diese Berechtigungen nicht benötigen, können Sie den Zugriff auf die Instanz-Metadaten blockieren, um sicherzustellen, dass Ihre Anwendungen nur über die Berechtigungen verfügen, die sie benötigen, und nicht ihre Knoten.

> **Warning**
>
> Durch das Blockieren des Zugriffs auf Instanz-Metadaten wird verhindert, dass Pods, die keine IRSA- oder EKS-Pod-Identitäten verwenden, die dem Worker-Knoten zugewiesene Rolle erben.

Sie können den Zugriff auf Instanz-Metadaten blockieren, indem Sie festlegen, dass die Instance IMDSv2 nur die Instance verwenden darf, und die Anzahl der Hops auf 1 aktualisieren, wie im folgenden Beispiel gezeigt. Sie können diese Einstellungen auch in die Startvorlage der Knotengruppe aufnehmen. Deaktivieren Sie Instanz-Metadaten nicht, da dies verhindert, dass Komponenten wie der Node Termination Handler und andere Dinge, die auf Instanz-Metadaten angewiesen sind, ordnungsgemäß funktionieren.

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

Wenn Sie Terraform verwenden, um Startvorlagen für die Verwendung mit verwalteten Knotengruppen zu erstellen, fügen Sie den Metadatenblock hinzu, um die Anzahl der Hops zu konfigurieren, wie in diesem Codeausschnitt dargestellt:

```
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" } …
```

Sie können den Zugriff eines Pods auf EC2-Metadaten auch blockieren, indem Sie iptables auf dem Knoten manipulieren. Weitere Informationen zu dieser Methode finden Sie unter Beschränken des Zugriffs auf den Instanz-Metadatendienst.

Wenn Sie eine Anwendung haben, die eine ältere Version des AWS-SDK verwendet, die IRSA- oder EKS-Pod-Identitäten nicht unterstützt, sollten Sie die SDK-Version aktualisieren.

Richten Sie die Vertrauensrichtlinie für IAM-Rollen für IRSA-Rollen auf den Namen des Dienstkontos, den Namespace und den Cluster ein

Die Vertrauensrichtlinie kann auf einen Namespace oder ein bestimmtes Dienstkonto innerhalb eines Namespace beschränkt werden. Wenn Sie IRSA verwenden, empfiehlt es sich, die Rollenvertrauensrichtlinie so explizit wie möglich zu gestalten, indem Sie den Namen des Dienstkontos angeben. Dadurch wird effektiv verhindert, dass andere Pods innerhalb desselben Namespace die Rolle übernehmen. Die CLI `eksctl` erledigt dies automatisch, wenn Sie sie zum Erstellen von accounts/IAM Servicerollen verwenden. Weitere Informationen finden Sie unter https://eksctl. io/usage/iamserviceaccounts/für weitere Informationen.

Wenn Sie direkt mit IAM arbeiten, fügt dies der Vertrauensrichtlinie der Rolle eine Bedingung hinzu, die anhand von Bedingungen sicherstellt, dass es sich bei dem `:sub` Anspruch um den Namespace und das Dienstkonto handelt, die Sie erwarten. Zum Beispiel hatten wir zuvor ein IRSA-Token mit dem Unteranspruch „system:serviceaccount:default:s3-read-only“. Dies `default` ist der `s3-read-only` Namespace und das Dienstkonto ist. Sie würden eine Bedingung wie die folgende verwenden, um sicherzustellen, dass nur Ihr Dienstkonto in einem bestimmten Namespace aus Ihrem Cluster diese Rolle übernehmen kann:

```
 "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"
     }
 }
```

Verwenden Sie eine IAM-Rolle pro Anwendung

Sowohl bei IRSA als auch bei EKS Pod Identity ist es eine bewährte Methode, jeder Anwendung ihre eigene IAM-Rolle zuzuweisen. Dadurch erhalten Sie eine verbesserte Isolierung, da Sie eine Anwendung ändern können, ohne dass sich dies auf eine andere auswirkt. Außerdem können Sie das Prinzip der geringsten Rechte anwenden, indem Sie einer Anwendung nur die Berechtigungen gewähren, die sie benötigt.

Wenn Sie ABAC mit EKS Pod Identity verwenden, können Sie eine gemeinsame IAM-Rolle für mehrere Dienstkonten verwenden und sich bei der Zugriffskontrolle auf deren Sitzungsattribute verlassen. Dies ist besonders nützlich, wenn Sie in großem Umfang arbeiten, da Sie mit ABAC mit weniger IAM-Rollen arbeiten können.

Wenn Ihre Anwendung Zugriff auf IMDS benötigt, verwenden Sie IMDSv2 und erhöhen Sie das Hop-Limit für EC2-Instances auf 2

IMDSv2erfordert, dass Sie eine PUT-Anfrage verwenden, um ein Sitzungstoken zu erhalten. Die erste PUT-Anfrage muss eine TTL für das Sitzungstoken enthalten. Neuere Versionen der AWS-SDKs übernehmen dies und die Verlängerung dieses Tokens automatisch. Es ist auch wichtig zu beachten, dass das Standard-Hop-Limit für EC2-Instances bewusst auf 1 gesetzt ist, um eine IP-Weiterleitung zu verhindern. Infolgedessen kann es bei Pods, die ein Sitzungstoken anfordern und auf EC2-Instances ausgeführt werden, irgendwann zu einem Timeout kommen und auf die Nutzung des IMDSv1 Datenflusses zurückgreifen. EKS fügt Unterstützung hinzu, IMDSv2 indem es sowohl Version 1 als auch Version 2 aktiviert und das Hop-Limit auf Knoten, die von eksctl oder mit den offiziellen Vorlagen bereitgestellt werden, auf 2 ändert. CloudFormation

Deaktivieren Sie das automatische Mounten von Dienstkonto-Tokens

Wenn Ihre Anwendung die Kubernetes-API nicht aufrufen muss, setzen Sie das `automountServiceAccountToken` Attribut PodSpec für Ihre Anwendung auf `false` oder patchen Sie das Standarddienstkonto in jedem Namespace, sodass es nicht mehr automatisch in Pods eingebunden wird. Beispiel:

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

Verwenden Sie spezielle Dienstkonten für jede Anwendung

Jede Anwendung sollte über ein eigenes Dienstkonto verfügen. Dies gilt für Dienstkonten für die Kubernetes-API sowie für IRSA und EKS Pod Identity.

> **Important**
>
> Wenn Sie bei Verwendung von IRSA einen blue/green Ansatz für Cluster-Upgrades verwenden, anstatt ein direktes Cluster-Upgrade durchzuführen, müssen Sie die Vertrauensrichtlinie jeder der IRSA-IAM-Rollen mit dem OIDC-Endpunkt des neuen Clusters aktualisieren. Bei einem blue/green Cluster-Upgrade erstellen Sie einen Cluster, auf

dem neben dem alten Cluster eine neuere Version von Kubernetes ausgeführt wird, und verwenden einen Load Balancer oder ein Service Mesh, um den Datenverkehr nahtlos von Diensten, die auf dem alten Cluster ausgeführt werden, auf den neuen Cluster zu verlagern. Wenn Sie blue/green Cluster-Upgrades mit EKS Pod Identity verwenden, würden Sie Pod-Identitätszuordnungen zwischen den IAM-Rollen und Dienstkonten im neuen Cluster erstellen. Und aktualisieren Sie die Vertrauensrichtlinie für IAM-Rollen, wenn Sie eine `sourceArn` Bedingung haben.

Führen Sie die Anwendung als Nicht-Root-Benutzer aus

Container werden standardmäßig als Root ausgeführt. Dadurch können sie zwar die Web-Identity-Token-Datei lesen, aber die Ausführung eines Containers als Root-Benutzer wird nicht als bewährte Methode angesehen. Als Alternative könnten Sie erwägen, das `spec.securityContext.runAsUser` Attribut dem hinzuzufügen PodSpec. Der Wert von `runAsUser` ist ein beliebiger Wert.

Im folgenden Beispiel werden alle Prozesse innerhalb des Pods unter der im `runAsUser` Feld angegebenen Benutzer-ID ausgeführt.

```
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" ]
```

Wenn Sie einen Container als Nicht-Root-Benutzer ausführen, wird verhindert, dass der Container das Token des IRSA-Dienstkontos liest, da dem Token standardmäßig 0600 [root] -Berechtigungen zugewiesen sind. Wenn Sie den SecurityContext für Ihren Container so aktualisieren, dass er fsgroup=65534 [Nobody] enthält, kann der Container das Token lesen.

```
spec:
  securityContext:
```

```
    fsGroup: 65534
```

In Kubernetes 1.19 und höher ist diese Änderung nicht mehr erforderlich und Anwendungen können das IRSA-Dienstkonto-Token lesen, ohne sie der Nobody-Gruppe hinzuzufügen.

Gewähren Sie Anwendungen den Zugriff mit den geringsten Rechten

Action Hero ist ein Hilfsprogramm, das Sie zusammen mit Ihrer Anwendung ausführen können, um die AWS-API-Aufrufe und die entsprechenden IAM-Berechtigungen zu identifizieren, die Ihre Anwendung benötigt, um ordnungsgemäß zu funktionieren. Es ähnelt IAM Access Advisor insofern, als es Ihnen hilft, den Umfang der IAM-Rollen, die Anwendungen zugewiesen sind, schrittweise einzuschränken. Weitere Informationen finden Sie in der Dokumentation zur Gewährung des Zugriffs mit den geringsten Rechten auf AWS-Ressourcen.

Erwägen Sie, eine Berechtigungsgrenze für IAM-Rollen festzulegen, die mit IRSA- und Pod-Identitäten verwendet werden. Sie können die Berechtigungsgrenze verwenden, um sicherzustellen, dass die von IRSA- oder Pod-Identitäten verwendeten Rollen eine maximale Berechtigungsstufe nicht überschreiten dürfen. Eine Beispielanleitung für die ersten Schritte mit Berechtigungsgrenzen mit einer Beispielrichtlinie für Rechtegrenzen finden Sie in diesem Github-Repo.

Überprüfen und widerrufen Sie unnötigen anonymen Zugriff auf Ihren EKS-Cluster

Idealerweise sollte der anonyme Zugriff für alle API-Aktionen deaktiviert werden. Anonymer Zugriff wird gewährt, indem ein RoleBinding oder ClusterRoleBinding für das in Kubernetes integrierte Benutzersystem erstellt wird:anonymous. Sie können das Tool rbac-lookup verwenden, um die Berechtigungen zu ermitteln, die system:anonymous user in Ihrem Cluster hat:

```
./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
```

Jede Rolle oder eine ClusterRole andere Rolle als system: public-info-viewer sollte nicht an system:anonymous user oder system:unauthenticated group gebunden sein.

Es kann einige legitime Gründe geben, den anonymen Zugriff auf bestimmte Bereiche zu aktivieren. APIs Wenn dies bei Ihrem Cluster der Fall ist, stellen Sie sicher, dass nur die spezifischen APIs Daten für anonyme Benutzer zugänglich sind. Wenn Sie diese Daten APIs ohne Authentifizierung offenlegen, wird Ihr Cluster nicht anfällig.

Vor Kubernetes/EKS Version 1.14 war die Gruppe system:unauthenticated standardmäßig mit system:discovery und system:basic-user verknüpft. ClusterRoles Beachten Sie, dass diese Berechtigungen auch dann noch auf Ihrem Cluster aktiviert sein können, wenn Sie Ihren Cluster auf Version 1.14 oder höher aktualisiert haben, da Cluster-Updates diese Berechtigungen nicht entziehen. Um zu überprüfen, welche „system:unauthenticated“ außer system: ClusterRoles haben, können public-info-viewer Sie den folgenden Befehl ausführen (erfordert jq util):

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

Und „system:unauthenticated“ kann aus allen Rollen außer „system:“ entfernt werden, indem man: 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 -
```

Alternativ können Sie es manuell mit kubectl describe und kubectl edit überprüfen und entfernen. Führen Sie den folgenden Befehl aus, um zu überprüfen, ob system:unauthenticated group über system:discovery-Berechtigungen für Ihren Cluster verfügt:

```
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
```

Führen Sie den folgenden Befehl aus, um zu überprüfen, ob system:unauthenticated group die Berechtigung system:basic-user auf Ihrem Cluster hat:

```
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
```

Wenn system:unauthenticated group auf Ihrem Cluster an system:discovery und/oder system:basic-user gebunden ist, sollten Sie diese Rollen von system:unauthenticated group trennen. ClusterRoles Bearbeiten Sie ClusterRoleBinding system:discovery mit dem folgenden Befehl:

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

Der obige Befehl öffnet die aktuelle Definition von system:discovery ClusterRoleBinding in einem Editor, wie unten gezeigt:

```
# 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
```

Löschen Sie den Eintrag für die Gruppe system:unauthenticated aus dem Abschnitt „Subjects“ im obigen Editor-Bildschirm.

Wiederholen Sie die gleichen Schritte für system:basic-user. ClusterRoleBinding

Wiederverwendung von AWS-SDK-Sitzungen mit IRSA

Wenn Sie IRSA verwenden, verwenden Anwendungen, die mit dem AWS-SDK geschrieben wurden, das an Ihre Pods gelieferte Token zum Aufrufen, um temporäre AWS-Anmeldeinformationen `sts:AssumeRoleWithWebIdentity` zu generieren. Dies unterscheidet sich von anderen AWS-Rechenservices, bei denen der Rechenservice temporäre AWS-Anmeldeinformationen direkt an die AWS-Rechenressource übermittelt, z. B. eine Lambda-Funktion. Das bedeutet, dass jedes Mal, wenn eine AWS-SDK-Sitzung initialisiert wird, ein Aufruf an AWS STS for `AssumeRoleWithWebIdentity` erfolgt. Wenn Ihre Anwendung schnell skaliert und viele AWS-SDK-Sitzungen initialisiert, kann es zu Drosselungen durch AWS STS kommen, da Ihr Code viele Aufrufe tätigen wird. `AssumeRoleWithWebIdentity`

Um dieses Szenario zu vermeiden, empfehlen wir, AWS-SDK-Sitzungen innerhalb Ihrer Anwendung wiederzuverwenden, damit keine unnötigen Aufrufe von getätigt `AssumeRoleWithWebIdentity` werden.

Im folgenden Beispielcode wird eine Sitzung mit dem Boto3-Python-SDK erstellt, und dieselbe Sitzung wird verwendet, um Clients zu erstellen und mit Amazon S3 und Amazon SQS zu interagieren. `AssumeRoleWithWebIdentity`wird nur einmal aufgerufen, und das AWS-SDK aktualisiert die Anmeldeinformationen automatisch`my_session`, wenn sie ablaufen.

```
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)
```

Wenn Sie eine Anwendung von einem anderen AWS-Rechenservice wie EC2 zu EKS mit IRSA migrieren, ist dies ein besonders wichtiges Detail. Bei anderen Rechendiensten ruft die Initialisierung einer AWS-SDK-Sitzung AWS STS nur auf, wenn Sie sie dazu anweisen.

Alternative Ansätze

IRSA- und EKS-Pod-Identitäten sind zwar die bevorzugten Methoden, um einem Pod eine AWS-Identität zuzuweisen, erfordern jedoch, dass Sie die aktuelle Version von AWS SDKs in Ihre Anwendung aufnehmen. Eine vollständige Liste der Programme SDKs, die derzeit IRSA unterstützen, finden Sie unter 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 .html. id-minimum-sdk Wenn Sie eine Anwendung haben, die Sie nicht sofort mit einem kompatiblen SDK aktualisieren können, stehen mehrere von der Community entwickelte Lösungen für die Zuweisung von IAM-Rollen zu Kubernetes-Pods zur Verfügung, darunter kube2iam und kiam. Obwohl AWS die Verwendung dieser Lösungen nicht befürwortet, duldet oder unterstützt, werden sie häufig von der gesamten Community verwendet, um ähnliche Ergebnisse zu erzielen wie IRSA und EKS Pod Identities.

Wenn Sie eine dieser Lösungen verwenden müssen, die nicht von AWS bereitgestellt werden, wenden Sie bitte die gebotene Sorgfalt an und stellen Sie sicher, dass Sie die damit verbundenen Sicherheitsauswirkungen verstehen.

Tools und Ressourcen

- Amazon EKS Security Immersion Workshop — Identity and Access Management
- Terraform EKS Blueprints Pattern — Vollständig privater Amazon EKS-Cluster
- Terraform EKS Blueprints Pattern — IAM Identity Center Single Sign-On für Amazon EKS-Cluster
- Terraform EKS Blueprints-Muster — Okta Single Sign-On für Amazon EKS-Cluster

- audit2rbac
- rbac.dev Eine Liste zusätzlicher Ressourcen, einschließlich Blogs und Tools, für Kubernetes RBAC
- Actionheld
- kube2iam
- Kiam

Sicherheit des Pods

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Die Pod-Spezifikation umfasst eine Vielzahl verschiedener Attribute, die Ihre allgemeine Sicherheitslage stärken oder schwächen können. Als Kubernetes-Experte sollte es Ihr Hauptanliegen sein, zu verhindern, dass ein Prozess, der in einem Container ausgeführt wird, die Isolationsgrenzen der Container-Laufzeit durchbricht und Zugriff auf den zugrunde liegenden Host erhält.

Linux-Funktionen

Die Prozesse, die in einem Container ausgeführt werden, werden standardmäßig im Kontext des [Linux] -Root-Benutzers ausgeführt. Obwohl die Aktionen von root innerhalb eines Containers teilweise durch die Linux-Funktionen eingeschränkt werden, die die Container-Runtime den Containern zuweist, könnten diese Standardberechtigungen es einem Angreifer ermöglichen, seine Rechte zu erweitern .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

EC2- und Fargate-Pods werden standardmäßig die oben genannten Funktionen zugewiesen. Darüber hinaus können Linux-Funktionen nur aus Fargate-Pods entfernt werden.

Pods, die als privilegiert ausgeführt werden, erben alle Linux-Funktionen, die mit Root auf dem Host verknüpft sind. Dies sollte nach Möglichkeit vermieden werden.

Knotenautorisierung

Alle Kubernetes-Worker-Knoten verwenden einen Autorisierungsmodus namens Node Authorization. Die Knotenautorisierung autorisiert alle API-Anfragen, die vom Kubelet stammen, und ermöglicht es den Knoten, die folgenden Aktionen auszuführen:

Operationen lesen:

- service
- Endpunkte
- Knoten
- Pods
- Secrets, Configmaps, persistente Volume-Claims und persistente Volumes im Zusammenhang mit Pods, die an den Knoten des Kubelets gebunden sind

Schreiboperationen:

- Knoten und Knotenstatus (aktivieren Sie das `NodeRestriction` Zulassungs-Plugin, um ein Kubelet darauf zu beschränken, seinen eigenen Knoten zu modifizieren)
- Pods und Pod-Status (aktivieren Sie das `NodeRestriction` Zugangs-Plugin, um ein Kubelet darauf zu beschränken, an sich selbst gebundene Pods zu ändern)
- Veranstaltungen

Operationen im Zusammenhang mit der Authentifizierung:

- Lese-/Schreibzugriff auf die CertificateSigningRequest (CSR-) API für TLS-Bootstrapping
- die Fähigkeit, Prüfungen zu erstellen und für delegierte Prüfungen durchzuführen TokenReview SubjectAccessReview authentication/authorization

EKS verwendet den Node Restriction Admission Controller, der es dem Knoten nur ermöglicht, einen begrenzten Satz von Knotenattributen und Pod-Objekten zu ändern, die an den Knoten gebunden sind. Dennoch kann ein Angreifer, dem es gelingt, Zugriff auf den Host zu erhalten,

weiterhin vertrauliche Informationen über die Umgebung aus der Kubernetes-API abrufen, die es ihm ermöglichen könnten, sich innerhalb des Clusters seitlich zu bewegen.

Sicherheitslösungen für Pods

Pod-Sicherheitsrichtlinie (PSP)

In der Vergangenheit wurden Ressourcen der Pod Security Policy (PSP) verwendet, um eine Reihe von Anforderungen zu spezifizieren, die Pods erfüllen mussten, bevor sie erstellt werden konnten. Seit Version 1.21 von Kubernetes sind PSP veraltet. Sie sollen in Kubernetes Version 1.25 entfernt werden.

> **Important**
>
> PSPs sind in Kubernetes Version 1.21 veraltet. Sie haben bis Version 1.25 oder etwa 2 Jahre Zeit, um auf eine Alternative umzusteigen. In diesem Dokument wird die Begründung für diese Abwertung erläutert.

Migration zu einer neuen Pod-Sicherheitslösung

Da PSPs diese Sicherheitskontrollen ab Kubernetes v1.25 entfernt wurden, müssen Clusteradministratoren und -betreiber diese Sicherheitskontrollen ersetzen. Zwei Lösungen können diesen Bedarf decken:

- Policy-as-code (PAC) -Lösungen aus dem Kubernetes-Ökosystem
- Kubernetes-Pod-Sicherheitsstandards (PSS)

Sowohl die PAC- als auch die PSS-Lösung können mit PSP koexistieren. Sie können in Clustern verwendet werden, bevor PSP entfernt wird. Dies erleichtert die Einführung bei der Migration von PSP. Bitte lies dir dieses Dokument durch, wenn du eine Migration von PSP zu PSS in Betracht ziehst.

Kyverno, eine der unten aufgeführten PAC-Lösungen, bietet in einem Blogbeitrag spezifische Leitlinien für die Migration von PSPs zu seiner Lösung, darunter analoge Richtlinien, Funktionsvergleiche und ein Migrationsverfahren. Zusätzliche Informationen und Anleitungen zur Migration nach Kyverno in Bezug auf Pod Security Admission (PSA) wurden hier im AWS-Blog veröffentlicht.

Policy-as-code (PAC)

Policy-as-code (PAC) -Lösungen bieten Leitplanken, um Cluster-Benutzer anzuleiten und unerwünschtes Verhalten durch vorgeschriebene und automatisierte Kontrollen zu verhindern. PAC verwendet Kubernetes Dynamic Admission Controllers, um den Anforderungsfluss des Kubernetes-API-Servers über einen Webhook-Aufruf abzufangen und Anforderungs-Payloads auf der Grundlage von Richtlinien, die als Code geschrieben und gespeichert wurden, zu mutieren und zu validieren. Die Mutation und Validierung erfolgt, bevor die API-Serveranfrage zu einer Änderung des Clusters führt. PAC-Lösungen verwenden Richtlinien, um Payloads von API-Serveranfragen auf der Grundlage von Taxonomie und Werten abzugleichen und darauf zu reagieren.

Für Kubernetes sind mehrere Open-Source-PAC-Lösungen verfügbar. Diese Lösungen sind nicht Teil des Kubernetes-Projekts. Sie stammen aus dem Kubernetes-Ökosystem. Einige PAC-Lösungen sind unten aufgeführt.

- OPA/Gatekeeper
- Öffnen Sie den Policy Agent (OPA)
- Kyverno
- Kubewart
- JS-Richtlinie

Weitere Informationen zu PAC-Lösungen und dazu, wie wir Ihnen bei der Auswahl der für Ihre Bedürfnisse geeigneten Lösung helfen können, finden Sie unter den folgenden Links.

- Richtlinienbasierte Gegenmaßnahmen für Kubernetes — Teil 1
- Richtlinienbasierte Gegenmaßnahmen für Kubernetes — Teil 2

Pod-Sicherheitsstandards (PSS) und Pod Security Admission (PSA)

Als Reaktion auf die PSP-Abwertung und die anhaltende Notwendigkeit, die Pod-Sicherheit mit einer integrierten Kubernetes-Lösung zu kontrollieren out-of-the-box, hat die Kubernetes Auth Special Interest Group die Pod Security Standards (PSS) und Pod Security Admission (PSA) entwickelt. Zu den Bemühungen der PSA gehört ein Webhook-Projekt für die Zugangssteuerung, das die im PSS definierten Kontrollen implementiert. Dieser Ansatz für den Zugangscontroller ähnelt dem, der in den PAC-Lösungen verwendet wird.

Laut der Kubernetes-Dokumentation definiert das PSS „drei verschiedene Richtlinien, um das gesamte Sicherheitsspektrum abzudecken. Diese Richtlinien sind kumulativ und reichen von sehr freizügig bis sehr restriktiv. `“

Diese Richtlinien sind wie folgt definiert:

- Privilegiert: Uneingeschränkte (unsichere) Richtlinie, die ein möglichst breites Maß an Berechtigungen bietet. Diese Richtlinie ermöglicht bekannte Eskalationen von Rechten. Es ist das Fehlen einer Richtlinie. Dies ist ideal für Anwendungen wie Logging-Agenten CNIs, Speichertreiber und andere systemweite Anwendungen, die privilegierten Zugriff benötigen.
- Grundeinstellung: Minimal restriktive Richtlinie, die bekannte Rechteerweiterungen verhindert. Ermöglicht die standardmäßige (minimal angegebene) Pod-Konfiguration. Die Basisrichtlinie verbietet die Verwendung von HostNetwork, HostPid, HostIPC, HostPath, HostPort, die Unfähigkeit, Linux-Funktionen hinzuzufügen, sowie mehrere andere Einschränkungen.
- Eingeschränkt: Stark eingeschränkte Richtlinie, die den aktuellen bewährten Methoden zur Pod-Hardening-Methode entspricht. Diese Richtlinie erbt von der Baseline und fügt weitere Einschränkungen hinzu, z. B. die Unfähigkeit, als Root-Benutzer oder als Root-Gruppe ausgeführt zu werden. Eingeschränkte Richtlinien können die Funktionsfähigkeit einer Anwendung beeinträchtigen. Sie sind in erster Linie auf die Ausführung sicherheitskritischer Anwendungen ausgerichtet.

Diese Richtlinien definieren Profile für die Pod-Ausführung, die in drei Stufen mit privilegiertem und eingeschränktem Zugriff unterteilt sind.

Um die im PSS definierten Kontrollen zu implementieren, arbeitet PSA in drei Modi:

- durchsetzen: Verstöße gegen die Richtlinien führen dazu, dass der Pod abgelehnt wird.
- Prüfung: Bei Verstößen gegen Richtlinien wird dem im Audit-Protokoll aufgezeichneten Ereignis eine Audit-Anmerkung hinzugefügt, andernfalls sind sie jedoch zulässig.
- Warnung: Bei Verstößen gegen Richtlinien wird eine Warnung angezeigt, die vom Benutzer angezeigt wird, sind aber ansonsten zulässig.

Diese Modi und die Profilstufen (Einschränkungsebenen) werden auf der Kubernetes-Namespace-Ebene mithilfe von Labels konfiguriert, wie im folgenden Beispiel zu sehen ist.

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

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

Wenn diese Betriebsmodi unabhängig voneinander verwendet werden, reagieren sie unterschiedlich, was zu unterschiedlichen Benutzererfahrungen führt. Der Erzwingungsmodus verhindert, dass Pods erstellt werden, wenn die jeweiligen PodSpecs gegen die konfigurierte Einschränkungsstufe verstoßen. In diesem Modus wird jedoch nicht verhindert, dass Kubernetes-Objekte, die Pods erstellen, wie z. B. Deployments, auf den Cluster angewendet werden, auch wenn die darin enthaltene PodSpec gegen die angewendete PSS verstößt. In diesem Fall wird das Deployment angewendet, wohingegen die Anwendung der Pods verhindert wird.

Dies ist eine schwierige Benutzererfahrung, da es keinen unmittelbaren Hinweis darauf gibt, dass das erfolgreich angewendete Deployment-Objekt einer fehlgeschlagenen Pod-Erstellung widerspricht. Mit den fraglichen PodSpecs werden keine Pods erstellt. Wenn Sie die Bereitstellungsressource mit überprüfen, `kubectl get deploy <DEPLOYMENT_NAME> -oyaml` wird die Meldung des ausgefallenen `.status.conditions` Pod-Elements angezeigt, wie unten dargestellt.

```
...
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
...
```

Sowohl im Audit - als auch im Warn-Modus verhindern die Pod-Einschränkungen nicht, dass Pods erstellt und gestartet werden, die die Verletzung verletzen. In diesen Modi werden jedoch Audit-Anmerkungen zu API-Server-Audit-Log-Ereignissen und Warnungen an API-Serverclients wie kubectl jeweils ausgelöst, wenn Pods sowie Objekte, die Pods erstellen, PodSpecs enthalten, die Verstöße enthalten. Eine `kubectl` Warnmeldung ist unten zu sehen.

```
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
```

Die PSA-Überwachungs - und Warnmodi sind bei der Einführung des PSS nützlich, ohne den Clusterbetrieb negativ zu beeinflussen.

Die PSA-Betriebsmodi schließen sich nicht gegenseitig aus und können kumulativ verwendet werden. Wie unten dargestellt, können die verschiedenen Modi in einem einzigen Namespace konfiguriert werden.

```
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
```

Im obigen Beispiel werden die benutzerfreundlichen Warnungen und Audit-Anmerkungen bei der Anwendung von Deployments bereitgestellt, und die Durchsetzung von Verstößen erfolgt auch auf Pod-Ebene. Tatsächlich können mehrere PSA-Labels unterschiedliche Profilebenen verwenden, wie unten dargestellt.

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

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

Im obigen Beispiel ist PSA so konfiguriert, dass alle Pods erstellt werden können, die die Basisprofilebene erfüllen, und dann vor Pods (und Objekten, die Pods erstellen) gewarnt werden, die gegen die eingeschränkte Profilebene verstoßen. Dies ist ein nützlicher Ansatz, um die möglichen Auswirkungen zu ermitteln, wenn vom Basisprofil zu eingeschränkten Profilen gewechselt wird.

Bestehende Pods

Wenn ein Namespace mit vorhandenen Pods dahingehend geändert wird, dass er ein restriktiveres PSS-Profil verwendet, werden im Audit - und Warn-Modus entsprechende Meldungen ausgegeben. Im Erzwingungsmodus werden die Pods jedoch nicht gelöscht. Die Warnmeldungen sind unten aufgeführt.

```
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
```

Ausnahmen

PSA verwendet Ausnahmen, um die Durchsetzung von Verstößen gegen Pods auszuschließen, die andernfalls verhängt worden wären. Diese Ausnahmen sind unten aufgeführt.

- Benutzernamen: Anfragen von Benutzern mit einem ausgenommenen authentifizierten (oder nachgemachten) Benutzernamen werden ignoriert.
- RuntimeClassNames: Pods und Workload-Ressourcen, die einen ausgenommenen Runtime-Klassennamen angeben, werden ignoriert.
- Namespaces: Pods und Workload-Ressourcen in einem ausgenommenen Namespace werden ignoriert.

Diese Ausnahmen werden in der PSA-Zugangscontroller-Konfiguration als Teil der API-Serverkonfiguration statisch angewendet.

In der Implementierung von Validating Webhook können die Ausnahmen innerhalb einer ConfigMapKubernetes-Ressource konfiguriert werden, die als Volume in den Container gemountet wird. 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"]
```

Wie im obigen ConfigMap YAML zu sehen ist, wurde die clusterweite Standard-PSS-Stufe für alle PSA-Modi „Audit“, „Enforce“ und „Warn“ auf „Restricted“ gesetzt. Dies betrifft alle Namespaces, mit Ausnahme der ausgenommenen:. `namespaces: ["kube-system","policy-test1"]` Darüber hinaus ist der pod-security-webhookNamespace in der unten abgebildeten ValidatingWebhookConfigurationRessource auch vom konfigurierten PSS ausgenommen.

```
...
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 wurde in Kubernetes v1.25 auf Stable umgestellt. Wenn Sie die Pod Security Admission-Funktion verwenden wollten, bevor sie standardmäßig aktiviert wurde, mussten Sie den Dynamic Admission Controller (mutierender Webhook) installieren. Die Anweisungen zur Installation und Konfiguration des Webhooks finden Sie hier.

Wählen Sie zwischen den policy-as-code Sicherheitsstandards und den Pod-Sicherheitsstandards

Die Pod Security Standards (PSS) wurden entwickelt, um die Pod Security Policy (PSP) zu ersetzen, indem sie eine Lösung bieten, die in Kubernetes integriert war und keine Lösungen aus dem Kubernetes-Ökosystem erforderte. Davon abgesehen sind policy-as-code (PAC) -Lösungen erheblich flexibler.

Die folgende Liste mit Vor- und Nachteilen soll Ihnen helfen, eine fundiertere Entscheidung über Ihre Pod-Sicherheitslösung zu treffen.

Policy-as-code (im Vergleich zu den Pod-Sicherheitsstandards)

Vorteile:

- Flexibler und detaillierter (bei Bedarf bis hin zu Ressourcenattributen)
- Es konzentriert sich nicht nur auf Pods, sondern kann auch gegen verschiedene Ressourcen und Aktionen eingesetzt werden
- Nicht nur auf Namespace-Ebene angewendet
- Ausgereifter als die Pod-Sicherheitsstandards
- Entscheidungen können auf beliebigen Angaben in der Nutzlast der API-Serveranforderung sowie auf vorhandenen Clusterressourcen und externen Daten (lösungsabhängig) basieren
- Unterstützt mutierende API-Serveranfragen vor der Validierung (lösungsabhängig)
- Kann ergänzende Richtlinien und Kubernetes-Ressourcen generieren (lösungsabhängig) — Aus Pod-Richtlinien kann Kyverno automatisch Richtlinien für übergeordnete Controller wie

Deployments generieren. Kyverno kann mithilfe von Generate Rules auch zusätzliche Kubernetes-Ressourcen generieren, „wenn eine neue Ressource erstellt wird oder wenn die Quelle aktualisiert wird“.)

- Kann verwendet werden, um nach links in die CICD-Pipelines zu wechseln, bevor der Kubernetes-API-Server aufgerufen wird (lösungsabhängig)
- Kann verwendet werden, um Verhaltensweisen zu implementieren, die nicht unbedingt sicherheitsrelevant sind, wie z. B. bewährte Verfahren, Organisationsstandards usw.
- Kann in Anwendungsfällen außerhalb von Kubernetes verwendet werden (lösungsabhängig)
- Aufgrund der Flexibilität kann die Benutzererfahrung an die Bedürfnisse der Benutzer angepasst werden

Nachteile:

- Nicht in Kubernetes integriert
- Das Erlernen, Konfigurieren und Unterstützen ist komplexer
- Für die Erstellung von Richtlinien sind möglicherweise neue erforderlich skills/languages/capabilities

Pod Security Admission (im Vergleich zu policy-as-code)

Vorteile:

- In Kubernetes integriert
- Einfacher zu konfigurieren
- Es müssen keine neuen Sprachen verwendet oder Richtlinien erstellt werden
- Wenn die Standard-Zugriffsebene des Clusters auf privilegiert konfiguriert ist, können Namespace-Labels verwendet werden, um Namespaces in die Pod-Sicherheitsprofile aufzunehmen.

Nachteile:

- Nicht so flexibel oder granular wie policy-as-code
- Nur 3 Stufen von Einschränkungen
- Hauptsächlich auf Pods konzentriert

Zusammenfassung

Wenn Sie derzeit über keine andere Pod-Sicherheitslösung als PSP verfügen und Ihr erforderlicher Sicherheitsstatus für den Pod dem in den Pod Security Standards (PSS) definierten Modell entspricht, ist es möglicherweise einfacher, statt einer Lösung das PSS einzuführen. policy-as-code Wenn Ihr Pod-Sicherheitsstatus jedoch nicht zum PSS-Modell passt oder Sie zusätzliche Kontrollen hinzufügen möchten, die über das von PSS definierte Maß hinausgehen, dann scheint eine policy-as-code Lösung besser geeignet zu sein.

Empfehlungen

Verwenden Sie mehrere Pod Security Admission (PSA) -Modi für eine bessere Benutzererfahrung

Wie bereits erwähnt, verhindert der PSA-Erzwingungsmodus, dass Pods mit PSS-Verstößen angewendet werden. Controller auf höherer Ebene, wie z. B. Bereitstellungen, werden jedoch nicht gestoppt. Tatsächlich wird die Bereitstellung erfolgreich angewendet, ohne dass es Hinweise darauf gibt, dass die Pods nicht angewendet wurden. Sie können zwar kubectl verwenden, um das Deployment-Objekt zu überprüfen und die Meldung über die fehlerhaften Pods vom PSA zu ermitteln, aber die Benutzererfahrung könnte besser sein. Um die Benutzererfahrung zu verbessern, sollten mehrere PSA-Modi (Audit, Enforce, Warn) verwendet werden.

```
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
```

Wenn im obigen Beispiel mit definiertem Erzwingungsmodus versucht wird, ein Deployment-Manifest mit PSS-Verstößen in der jeweiligen PodSpec auf den Kubernetes-API-Server anzuwenden, wird das Deployment erfolgreich angewendet, die Pods jedoch nicht. Und da die Modi Audit und Warn ebenfalls aktiviert sind, erhält der API-Server-Client eine Warnmeldung, und das Audit-Log-Ereignis des API-Servers wird ebenfalls mit einer Meldung versehen.

Schränken Sie die Container ein, die als privilegiert ausgeführt werden können

Wie bereits erwähnt, erben Container, die als privilegiert ausgeführt werden, alle Linux-Funktionen, die Root-Benutzern auf dem Host zugewiesen sind. In den seltensten Fällen benötigen Container diese Art von Rechten, um ordnungsgemäß zu funktionieren. Es gibt mehrere Methoden, mit denen die Berechtigungen und Funktionen von Containern eingeschränkt werden können.

> **Important**
>
> Fargate ist ein Starttyp, mit dem Sie „serverlose“ Container ausführen können, wobei die Container eines Pods auf einer von AWS verwalteten Infrastruktur ausgeführt werden. Mit Fargate können Sie keinen privilegierten Container ausführen oder Ihren Pod für die Verwendung von HostNetwork oder HostPort konfigurieren.

Führen Sie Prozesse in Containern nicht als Root aus

Alle Container werden standardmäßig als Root ausgeführt. Dies könnte problematisch sein, wenn ein Angreifer in der Lage ist, eine Sicherheitslücke in der Anwendung auszunutzen und Shell-Zugriff auf den laufenden Container zu erhalten. Sie können dieses Risiko auf verschiedene Weise mindern. Zuerst, indem Sie die Shell aus dem Container-Image entfernen. Zweitens fügen Sie Ihrem Dockerfile die USER-Direktive hinzu oder führen Sie die Container im Pod als Nicht-Root-Benutzer aus. Die Kubernetes-PodSpec enthält unter eine Reihe von Feldern, mit denen Sie die and/or Benutzergruppe angeben können`spec.securityContext`, unter der Ihre Anwendung ausgeführt werden soll. Diese Felder sind jeweils und. `runAsUser` `runAsGroup`

Um die Verwendung von und den `spec.securityContext` zugehörigen Elementen innerhalb von Kubernetes zu erzwingen, können PodSpec policy-as-code - oder Pod-Sicherheitsstandards zu Clustern hinzugefügt werden. Mit diesen Lösungen können Sie Richtlinien oder Profile schreiben und/oder verwenden, mit denen eingehende Payloads für Kubernetes-API-Serveranfragen validiert werden können, bevor sie dauerhaft in etcd gespeichert werden. Darüber hinaus können policy-as-code Lösungen eingehende Anfragen mutieren und in einigen Fällen neue Anfragen generieren.

Führen Sie Docker niemals in Docker aus und mounten Sie den Socket niemals im Container

Auf diese Weise können Sie zwar bequem build/run Images in Docker-Containern erstellen, aber im Grunde geben Sie die vollständige Kontrolle über den Knoten an den Prozess ab, der im Container

ausgeführt wird. Wenn Sie Container-Images auf Kubernetes erstellen müssen, verwenden Sie stattdessen Kaniko, buildah oder einen ähnlichen Build-Service. CodeBuild

> **Note**
>
> Kubernetes-Cluster, die für die CICD-Verarbeitung verwendet werden, z. B. für die Erstellung von Container-Images, sollten von Clustern isoliert werden, auf denen allgemeinere Workloads ausgeführt werden.

Beschränken Sie die Verwendung von HostPath oder, falls HostPath erforderlich ist, schränken Sie ein, welche Präfixe verwendet werden können, und konfigurieren Sie das Volume als schreibgeschützt

`hostPath`ist ein Volume, das ein Verzeichnis vom Server direkt in den Container einhängt. Selten benötigen Pods diese Art von Zugriff, aber wenn doch, müssen Sie sich der Risiken bewusst sein. Standardmäßig haben Pods, die als Root ausgeführt werden, Schreibzugriff auf das von HostPath bereitgestellte Dateisystem. Dies könnte es einem Angreifer ermöglichen, die Kubelet-Einstellungen zu ändern, symbolische Links zu Verzeichnissen oder Dateien zu erstellen, die nicht direkt vom HostPath offengelegt werden, z. B. /etc/shadow, SSH-Schlüssel zu installieren, auf dem Host bereitgestellte Geheimnisse zu lesen und andere bösartige Dinge. Um die von HostPath ausgehenden Risiken zu minimieren, konfigurieren Sie das als, zum Beispiel: `spec.containers.volumeMounts` `readOnly`

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

Sie sollten auch policy-as-code Lösungen verwenden, um die Anzahl der Verzeichnisse, die von `hostPath` Volumes verwendet werden können, einzuschränken oder die `hostPath` Nutzung ganz zu verhindern. Sie können die Pod Security Standards-Richtlinien „Baseline“ oder „Eingeschränkt“ verwenden, um die Verwendung von zu verhindern`hostPath`.

Weitere Informationen zu den Gefahren einer privilegierten Eskalation finden Sie im Blog Bad Pods: Kubernetes Pod Privilege Escalation von Seth Art.

Legen Sie Anfragen und Limits für jeden Container fest, um Ressourcenkonflikte und DoS-Angriffe zu vermeiden

Ein Pod ohne Anfragen oder Limits kann theoretisch alle auf einem Host verfügbaren Ressourcen verbrauchen. Wenn zusätzliche Pods auf einem Knoten eingeplant werden, kann es auf dem Knoten zu einer CPU- oder Speicherauslastung kommen, was dazu führen kann, dass das Kubelet Pods beendet oder aus dem Knoten entfernt. Sie können zwar nicht verhindern, dass dies alles zusammen passiert, aber die Festlegung von Anforderungen und Grenzwerten trägt dazu bei, Ressourcenkonflikte zu minimieren und das Risiko zu verringern, das von schlecht geschriebenen Anwendungen ausgeht, die übermäßig viele Ressourcen verbrauchen.

Mit `podSpec` dieser können Sie Anforderungen und Grenzwerte für CPU und Arbeitsspeicher angeben. Die CPU wird als komprimierbare Ressource angesehen, da sie überlastet sein kann. Speicher ist inkomprimierbar, d. h. er kann nicht von mehreren Containern gemeinsam genutzt werden.

Wenn Sie Anforderungen für CPU oder Arbeitsspeicher angeben, legen Sie im Wesentlichen die Speichermenge fest, die Container garantiert erhalten. Kubernetes aggregiert die Anfragen aller Container in einem Pod, um zu bestimmen, auf welchem Knoten der Pod geplant werden soll. Wenn ein Container die angeforderte Speichermenge überschreitet, kann er beendet werden, wenn der Knoten unter Speicherdruck steht.

Grenzwerte sind die maximale Menge an CPU- und Speicherressourcen, die ein Container verbrauchen darf. Sie entsprechen direkt dem `memory.limit_in_bytes` Wert der für den Container erstellten Cgroup. Ein Container, der das Speicherlimit überschreitet, wird als OOM beendet. Wenn ein Container sein CPU-Limit überschreitet, wird er gedrosselt.

> **Note**
>
> Bei der Verwendung von Containern `resources.limits` wird dringend empfohlen, dass die Nutzung von Container-Ressourcen (auch bekannt als Resource Footprints) datengesteuert und korrekt erfolgt und auf Belastungstests basiert. In Ermangelung eines genauen und vertrauenswürdigen Ressourcenbedarfs kann der Container `resources.limits` aufgefüllt werden. Er `resources.limits.memory` könnte beispielsweise um 20 bis 30% höher als die beobachtbaren Höchstwerte aufgefüllt werden, um möglichen Ungenauigkeiten bei der Speicherressourcenbegrenzung Rechnung zu tragen.

Kubernetes verwendet drei Quality of Service (QoS) -Klassen, um die Workloads zu priorisieren, die auf einem Knoten ausgeführt werden. Dazu zählen:

- garantierte
- platzfähig
- bestes Bemühen

Wenn keine Limits und Anfragen festgelegt sind, wird der Pod als Best-Effort (niedrigste Priorität) konfiguriert. Best-Effort-Pods sind die ersten, die beendet werden, wenn nicht genügend Speicher zur Verfügung steht. Wenn Grenzwerte für alle Container innerhalb des Pods festgelegt sind oder wenn die Anfragen und Grenzwerte auf dieselben Werte und nicht gleich 0 festgelegt sind, wird der Pod als garantiert konfiguriert (höchste Priorität). Garantierte Pods werden nicht beendet, es sei denn, sie überschreiten ihre konfigurierten Speichergrenzen. Wenn die Grenzwerte und Anfragen mit unterschiedlichen Werten konfiguriert sind und nicht gleich 0 sind, oder wenn ein Container innerhalb des Pods Grenzwerte festlegt und die anderen keine Grenzwerte für unterschiedliche Ressourcen haben oder keine Grenzwerte für unterschiedliche Ressourcen festgelegt haben, werden die Pods als Burstable (mittlere Priorität) konfiguriert. Diese Pods haben einige Ressourcengarantien, können aber beendet werden, sobald sie den angeforderten Speicherplatz überschreiten.

> **Important**
>
> Anfragen wirken sich nicht auf den `memory_limit_in_bytes` Wert der Cgroup des Containers aus. Das Cgroup-Limit ist auf die Menge des auf dem Host verfügbaren Speichers festgelegt. Wenn der Anforderungswert jedoch zu niedrig eingestellt wird, kann dies dazu führen, dass der Pod vom Kubelet beendet wird, wenn der Knoten Speicherauslastung ausgesetzt ist.

Klasse	Priorität	Bedingung	Zustand beenden
Garantiert	Am höchsten	Limit = Anfrage! = 0	Überschreite nur die Speichergrenzen
Burst-fähig	Medium	begrenzen! = Anfrage! = 0	Kann beendet werden, wenn der

Klasse	Priorität	Bedingung	Zustand beenden
			Anforderungsspeicher überschritten wird
Bestes Bemühen	Am niedrigsten	Limit und Anfrage nicht festgelegt	Der Erste, der getötet wird, wenn nicht genügend Speicher zur Verfügung steht

Weitere Informationen zu Ressourcen-QoS finden Sie in der Kubernetes-Dokumentation.

Sie können die Verwendung von Anfragen und Limits erzwingen, indem Sie ein Ressourcenkontingent für einen Namespace festlegen oder einen Grenzbereich erstellen. Mit einem Ressourcenkontingent können Sie die Gesamtmenge der Ressourcen, z. B. CPU und RAM, angeben, die einem Namespace zugewiesen sind. Wenn es auf einen Namespace angewendet wird, müssen Sie Anforderungen und Grenzwerte für alle Container angeben, die in diesem Namespace bereitgestellt werden. Im Gegensatz dazu bieten Ihnen Grenzbereiche eine genauere Kontrolle über die Zuweisung von Ressourcen. Mit Grenzbereichen können Sie CPU min/max - und Speicherressourcen pro Pod oder pro Container innerhalb eines Namespace angeben. Sie können sie auch verwenden, um Standardanforderungs-/Grenzwerte festzulegen, falls keine angegeben sind.

Policy-as-code Lösungen können verwendet werden, um Anfragen und Limits durchzusetzen oder sogar Ressourcenkontingente und Grenzbereiche zu erstellen, wenn Namespaces erstellt werden.

Erlauben Sie keine privilegierte Eskalation

Die privilegierte Eskalation ermöglicht es einem Prozess, den Sicherheitskontext zu ändern, in dem er ausgeführt wird. Sudo ist ein gutes Beispiel dafür, ebenso wie Binärdateien mit dem SUID- oder SGID-Bit. Die privilegierte Eskalation ist im Grunde eine Möglichkeit für Benutzer, eine Datei mit den Rechten eines anderen Benutzers oder einer anderen Gruppe auszuführen. Sie können verhindern, dass ein Container die privilegierte Eskalation verwendet, indem Sie eine policy-as-code mutierende Richtlinie implementieren, die `allowPrivilegeEscalation` auf `false` oder durch die Einstellung `securityContext.allowPrivilegeEscalation` von. `podSpec` Policy-as-code Richtlinien können auch verwendet werden, um zu verhindern, dass API-Serveranfragen erfolgreich sind, wenn falsche Einstellungen erkannt werden. Pod-Sicherheitsstandards können auch verwendet werden, um zu verhindern, dass Pods die Rechteeskalation verwenden.

Deaktivieren Sie ServiceAccount Token-Mounts

Für Pods, die nicht auf die Kubernetes-API zugreifen müssen, können Sie das automatische Mounten eines ServiceAccount Tokens in einer Pod-Spezifikation oder für alle Pods, die einen bestimmten verwenden, deaktivieren. 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
```

Deaktivieren Sie die Diensterkennung

Für Pods, die keine Clusterdienste suchen oder aufrufen müssen, können Sie die Menge der an einen Pod übergebenen Informationen reduzieren. Sie können die DNS-Richtlinie des Pods so festlegen, dass CoreDNS nicht verwendet wird und Dienste im Namespace des Pods nicht als Umgebungsvariablen verfügbar gemacht werden. Weitere Informationen zu Service-Links finden Sie in den Kubernetes-Dokumenten zu Umgebungsvariablen. Der Standardwert für die DNS-Richtlinie eines Pods ist "ClusterFirst", wodurch Cluster-DNS verwendet wird, während der nicht standardmäßige Wert „Default“ die DNS-Auflösung des zugrunde liegenden Knotens verwendet. Weitere Informationen finden Sie in den Kubernetes-Dokumenten zur Pod-DNS-Richtlinie.

Example

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-no-service-info
spec:
```

```
    dnsPolicy: Default # "Default" is not the true default value
    enableServiceLinks: false
```

Konfigurieren Sie Ihre Images mit einem schreibgeschützten Root-Dateisystem

Wenn Sie Ihre Bilder mit einem schreibgeschützten Root-Dateisystem konfigurieren, wird verhindert, dass ein Angreifer eine Binärdatei auf dem Dateisystem überschreibt, das Ihre Anwendung verwendet. Wenn Ihre Anwendung in das Dateisystem schreiben muss, sollten Sie in Erwägung ziehen, in ein temporäres Verzeichnis zu schreiben oder ein Volume anzuhängen und zu mounten. Sie können dies erzwingen, indem Sie die Pods SecurityContext wie folgt einstellen:

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

Policy-as-code und Pod-Sicherheitsstandards können verwendet werden, um dieses Verhalten durchzusetzen.

Example

Gemäß Windows können Container in Kubernetes `securityContext.readOnlyRootFilesystem` nicht auf einen Container eingestellt werden, `true` der unter Windows läuft, da Schreibzugriff erforderlich ist, damit Registrierungs- und Systemprozesse innerhalb des Containers ausgeführt werden können.

Tools und Ressourcen

- Workshop zum Eintauchen in die Sicherheit von Amazon EKS — Pod-Sicherheit
- open-policy-agent/gatekeeper-library: Die OPA Gatekeeper-Richtlinienbibliothek ist eine Bibliothek mit Richtlinien, die Sie als Ersatz verwenden OPA/Gatekeeper können. PSPs
- Richtlinienbibliothek von Kyverno
- Eine Sammlung gängiger OPA- und Kyverno-Richtlinien für EKS.
- Politische Gegenmaßnahmen: Teil 1
- Politische Gegenmaßnahmen: Teil 2
- Pod Security Policy Migrator ist ein Tool, das in OPA/Gatekeeper- oder Kyverno-Richtlinien konvertiert PSPs KubeWarden

- NeuVector Die Open-Source-Zero-Trust-Container-Sicherheitsplattform von SUSE bietet Prozess- und Dateisystemrichtlinien sowie Regeln für die Zugangskontrolle.

Isolierung von Mandanten

Wenn wir an Mehrmandantenfähigkeit denken, möchten wir oft einen Benutzer oder eine Anwendung von anderen Benutzern oder Anwendungen isolieren, die auf einer gemeinsam genutzten Infrastruktur ausgeführt werden.

Kubernetes ist ein Single-Tenant-Orchestrator, d. h. eine einzelne Instanz der Steuerungsebene wird von allen Mandanten innerhalb eines Clusters gemeinsam genutzt. Es gibt jedoch verschiedene Kubernetes-Objekte, mit denen Sie den Anschein einer Mehrmandantenfähigkeit erzeugen können. Beispielsweise können Namespaces und rollenbasierte Zugriffskontrollen (RBAC) implementiert werden, um Mandanten logisch voneinander zu isolieren. In ähnlicher Weise können Kontingente und Grenzbereiche verwendet werden, um die Menge an Clusterressourcen zu steuern, die jeder Mandant verbrauchen kann. Dennoch ist der Cluster das einzige Konstrukt, das eine starke Sicherheitsgrenze bietet. Das liegt daran, dass ein Angreifer, der Zugriff auf einen Host innerhalb des Clusters erlangt, ConfigMapsalle Secrets und Volumes abrufen kann, die auf diesem Host installiert sind. Sie könnten sich auch als Kubelet ausgeben, was es ihnen ermöglichen würde, die Attribute des Knotens zu manipulieren, der sich innerhalb des Clusters seitlich and/or bewegt.

In den folgenden Abschnitten wird erklärt, wie die Mandantenisolierung implementiert und gleichzeitig die Risiken der Verwendung eines Single-Tenant-Orchestrators wie Kubernetes minimiert werden.

Sanfte Mehrmandantenfähigkeit

Bei Soft-Multi-Tenancy verwenden Sie native Kubernetes-Konstrukte, z. B. Namespaces, Rollen und Rollenbindungen sowie Netzwerkrichtlinien, um eine logische Trennung zwischen Mandanten herzustellen. RBAC kann beispielsweise verhindern, dass Mandanten auf die Ressourcen des jeweils anderen zugreifen oder diese manipulieren. Kontingente und Grenzbereiche steuern die Menge an Clusterressourcen, die jeder Mandant verbrauchen kann, während Netzwerkrichtlinien dazu beitragen können, zu verhindern, dass Anwendungen, die in unterschiedlichen Namespaces bereitgestellt werden, miteinander kommunizieren.

Keine dieser Kontrollen verhindert jedoch, dass sich Pods verschiedener Mandanten einen Knoten teilen. Wenn eine stärkere Isolierung erforderlich ist, können Sie mithilfe einer Knotenauswahl, Anti-Affinitätsregeln, and/or Taints und Toleranzen die Planung von Pods verschiedener Mandanten auf

separaten Knoten erzwingen, die häufig als Knoten für einzelne Mandanten bezeichnet werden. In einer Umgebung mit vielen Mietern könnte dies ziemlich kompliziert und kostspielig werden.

> **Important**
>
> Die mit Namespaces implementierte sanfte Mehrmandantenfähigkeit ermöglicht es Ihnen nicht, Mandanten eine gefilterte Liste von Namespaces zur Verfügung zu stellen, da es sich bei Namespaces um einen Typ mit globalem Gültigkeitsbereich handelt. Wenn ein Mandant einen bestimmten Namespace anzeigen kann, kann er alle Namespaces innerhalb des Clusters einsehen.

> **Warning**
>
> Mit behalten Mandanten standardmäßig die Möglichkeit soft-multi-tenancy, CoreDNS für alle Dienste abzufragen, die innerhalb des Clusters ausgeführt werden. Ein Angreifer könnte dies ausnutzen, indem er dig SRV von einem beliebigen Pod im Cluster `..svc.cluster.local` aus ausführt. Wenn Sie den Zugriff auf DNS-Einträge von Diensten einschränken müssen, die in Ihren Clustern ausgeführt werden, sollten Sie die Firewall- oder Policy-Plug-ins für CoreDNS verwenden. Weitere Informationen finden Sie unter https:// github.com/coredns/ kubernetes-metadata-multi-tenancy policy# -policy.

Kiosk ist ein Open-Source-Projekt, das bei der Implementierung von Soft-Multi-Tenancy helfen kann. Es ist als eine Reihe von CRDs AND-Controllern implementiert, die die folgenden Funktionen bieten:

- Konten und Kontobenutzer zur Trennung von Mandanten in einem gemeinsamen Kubernetes-Cluster
- Self-Service-Namespace-Bereitstellung für Kontonutzer
- Kontolimits zur Sicherstellung der Servicequalität und Fairness bei der gemeinsamen Nutzung eines Clusters
- Namespace-Vorlagen für die sichere Mandantenisolierung und Self-Service-Namespace-Initialisierung

Loft ist ein kommerzielles Angebot der Entwickler von Kiosk, das die folgenden Funktionen bietet: DevSpace

- Multi-Cluster-Zugriff zur Gewährung des Zugriffs auf Bereiche in verschiedenen Clustern
- Im Ruhemodus werden Bereitstellungen in einem Bereich während Zeiten der Inaktivität herunterskaliert
- Single Sign-On mit OIDC-Authentifizierungsanbietern wie GitHub

Es gibt drei Hauptanwendungsfälle, die durch Soft-Multi-Tenancy abgedeckt werden können.

Unternehmensumgebung

Die erste ist eine Unternehmensumgebung, in der den „Mandanten“ teilweise vertraut wird, da es sich um Mitarbeiter, Auftragnehmer oder anderweitig von der Organisation autorisierte Personen handelt. Jeder Mandant ordnet sich in der Regel einer Verwaltungsabteilung wie einer Abteilung oder einem Team zu.

In dieser Art von Umgebung ist in der Regel ein Clusteradministrator für die Erstellung von Namespaces und die Verwaltung von Richtlinien verantwortlich. Sie können auch ein delegiertes Verwaltungsmodell implementieren, bei dem bestimmten Personen die Aufsicht über einen Namespace übertragen wird, sodass sie CRUD-Operationen für Objekte ausführen können, die nicht mit Richtlinien in Verbindung stehen, wie Bereitstellungen, Dienste, Pods, Jobs usw.

Die durch eine Container-Runtime bereitgestellte Isolierung kann innerhalb dieser Einstellung akzeptabel sein, oder sie muss möglicherweise durch zusätzliche Kontrollen für die Pod-Sicherheit erweitert werden. Es kann auch erforderlich sein, die Kommunikation zwischen Diensten in verschiedenen Namespaces einzuschränken, wenn eine strengere Isolierung erforderlich ist.

Kubernetes als Service

Im Gegensatz dazu kann Soft Multi-Tenancy in Einstellungen verwendet werden, in denen Sie Kubernetes als Service (KaaS) anbieten möchten. Bei KaaS wird Ihre Anwendung zusammen mit einer Sammlung von Controllern, die eine Reihe von PaaS-Diensten bereitstellenCRDs , in einem gemeinsam genutzten Cluster gehostet. Mandanten interagieren direkt mit dem Kubernetes-API-Server und dürfen CRUD-Operationen an Objekten ausführen, die nicht den Richtlinien entsprechen. Es gibt auch ein Self-Service-Element, da Mandanten möglicherweise ihre eigenen Namespaces erstellen und verwalten können. In dieser Art von Umgebung wird davon ausgegangen, dass Mandanten nicht vertrauenswürdigen Code ausführen.

Um Mandanten in dieser Art von Umgebung zu isolieren, müssen Sie wahrscheinlich strenge Netzwerkrichtlinien sowie Pod-Sandboxing implementieren. Beim Sandboxing führen Sie die

Container eines Pods in einer Mikro-VM wie Firecracker oder in einem User-Space-Kernel aus. Heute können Sie mit EKS Fargate Sandbox-Pods erstellen.

Software-as-a-Service (SaaS)

Der letzte Anwendungsfall für Soft-Multi-Tenancy ist eine Software-as-a-Service (SaaS-) Umgebung. In dieser Umgebung ist jeder Mandant einer bestimmten Instanz einer Anwendung zugeordnet, die innerhalb des Clusters ausgeführt wird. Jede Instanz hat oft ihre eigenen Daten und verwendet separate Zugriffskontrollen, die normalerweise unabhängig von Kubernetes RBAC sind.

Im Gegensatz zu den anderen Anwendungsfällen hat der Mandant in einer SaaS-Umgebung keine direkte Schnittstelle zur Kubernetes-API. Stattdessen ist die SaaS-Anwendung für die Schnittstelle mit der Kubernetes-API verantwortlich, um die erforderlichen Objekte zur Unterstützung der einzelnen Mandanten zu erstellen.

Kubernetes-Konstrukte

In jeder dieser Instanzen werden die folgenden Konstrukte verwendet, um Mandanten voneinander zu isolieren:

Namespaces

Namespaces sind für die Implementierung von Soft-Multi-Tenancy von grundlegender Bedeutung. Sie ermöglichen es Ihnen, den Cluster in logische Partitionen zu unterteilen. Kontingente, Netzwerkrichtlinien, Dienstkonten und andere Objekte, die für die Implementierung von Mehrmandantenfähigkeit erforderlich sind, sind auf einen Namespace beschränkt.

Netzwerkrichtlinien

Standardmäßig dürfen alle Pods in einem Kubernetes-Cluster miteinander kommunizieren. Dieses Verhalten kann mithilfe von Netzwerkrichtlinien geändert werden.

Netzwerkrichtlinien schränken die Kommunikation zwischen Pods mithilfe von Labels oder IP-Adressbereichen ein. In einer Umgebung mit mehreren Mandanten, in der eine strikte Netzwerkisolierung zwischen Mandanten erforderlich ist, empfehlen wir, mit einer Standardregel zu beginnen, die die Kommunikation zwischen Pods verweigert, und einer anderen Regel, die es allen Pods ermöglicht, den DNS-Server zur Namensauflösung abzufragen. Damit können Sie beginnen, tolerantere Regeln hinzuzufügen, die die Kommunikation innerhalb eines Namespaces ermöglichen. Dies kann nach Bedarf weiter verfeinert werden.

> **Note**
>
> Amazon VPC CNI unterstützt jetzt Kubernetes-Netzwerkrichtlinien, um Richtlinien zu erstellen, mit denen sensible Workloads isoliert und vor unbefugtem Zugriff geschützt werden können, wenn Kubernetes auf AWS ausgeführt wird. Das bedeutet, dass Sie alle Funktionen der Network Policy API in Ihrem Amazon EKS-Cluster nutzen können. Diese Ebene der detaillierten Kontrolle ermöglicht es Ihnen, das Prinzip der geringsten Rechte zu implementieren, das sicherstellt, dass nur autorisierte Pods miteinander kommunizieren dürfen.

> **Important**
>
> Netzwerkrichtlinien sind notwendig, aber nicht ausreichend. Die Durchsetzung von Netzwerkrichtlinien erfordert eine Policy-Engine wie Calico oder Cilium.

Rollenbasierte Zugriffskontrolle (RBAC)

Rollen und Rollenbindungen sind die Kubernetes-Objekte, die zur Durchsetzung der rollenbasierten Zugriffskontrolle (RBAC) in Kubernetes verwendet werden. Rollen enthalten Listen von Aktionen, die gegen Objekte in Ihrem Cluster ausgeführt werden können. Rollenbindungen geben die Personen oder Gruppen an, für die die Rollen gelten. In den Unternehmens- und KaaS-Einstellungen kann RBAC verwendet werden, um die Verwaltung von Objekten durch ausgewählte Gruppen oder Einzelpersonen zu ermöglichen.

Kontingente

Kontingente werden verwendet, um Grenzwerte für Workloads zu definieren, die in Ihrem Cluster gehostet werden. Mit Kontingenten können Sie die maximale Menge an CPU und Arbeitsspeicher angeben, die ein Pod verbrauchen kann, oder Sie können die Anzahl der Ressourcen einschränken, die einem Cluster oder Namespace zugewiesen werden können. Mit Grenzbereichen können Sie für jedes Limit Mindest-, Höchst- und Standardwerte angeben.

Eine Überbelegung von Ressourcen in einem gemeinsam genutzten Cluster ist oft von Vorteil, da Sie so Ihre Ressourcen maximieren können. Ein unbegrenzter Zugriff auf einen Cluster kann jedoch zu einem Mangel an Ressourcen führen, was zu Leistungseinbußen und zum Verlust der Anwendungsverfügbarkeit führen kann. Wenn die Anforderungen eines Pods zu niedrig angesetzt

sind und die tatsächliche Ressourcenauslastung die Kapazität des Knotens übersteigt, kommt es auf dem Knoten zu einer CPU- oder Speicherauslastung. In diesem Fall können Pods neu gestartet und vom and/or Knoten entfernt werden.

Um dies zu verhindern, sollten Sie planen, in einer Umgebung mit mehreren Mandanten Kontingente für Namespaces festzulegen, um Mandanten zu zwingen, bei der Planung ihrer Pods auf dem Cluster Anforderungen und Beschränkungen festzulegen. Dadurch wird auch ein potenzieller Denial-of-Service eingedämmt, indem die Menge an Ressourcen begrenzt wird, die ein Pod verbrauchen kann.

Sie können Kontingente auch verwenden, um die Ressourcen des Clusters an die Ausgaben eines Mandanten anzupassen. Dies ist besonders im KaaS-Szenario nützlich.

Priorität und Präemption von Pods

Pod-Priorität und Präemption können nützlich sein, wenn Sie einem Pod im Vergleich zu anderen Pods mehr Bedeutung beimessen möchten. Mit Pod-Priorität können Sie beispielsweise Pods von Kunde A so konfigurieren, dass sie mit einer höheren Priorität als Kunde B ausgeführt werden. Wenn nicht genügend Kapazität verfügbar ist, entfernt der Scheduler die Pods mit niedrigerer Priorität von Kunde B, um die Pods mit höherer Priorität von Kunde A aufzunehmen. Dies kann besonders praktisch in einer SaaS-Umgebung sein, in der Kunden, die bereit sind, eine Prämie zu zahlen, eine höhere Priorität erhalten.

> **Important**
>
> Die Priorität von Pods kann sich unerwünscht auf andere Pods mit niedrigerer Priorität auswirken. Beispielsweise werden die Pods des Opfers zwar ordnungsgemäß beendet, dies PodDisruptionBudget ist jedoch nicht garantiert, wodurch eine Anwendung mit niedrigerer Priorität, die auf einem Quorum von Pods basiert, beschädigt werden könnte. Weitere Informationen finden Sie unter Einschränkungen der Präemption.

Mildernde Kontrollen

Ihr Hauptanliegen als Administrator einer Umgebung mit mehreren Mandanten besteht darin, zu verhindern, dass sich ein Angreifer Zugriff auf den zugrundeliegenden Host verschafft. Zur Minderung dieses Risikos sollten die folgenden Maßnahmen in Betracht gezogen werden:

Sandbox-Ausführungsumgebungen für Container

Sandboxing ist eine Technik, bei der jeder Container in einer eigenen isolierten virtuellen Maschine ausgeführt wird. Zu den Technologien, die Pod-Sandboxing durchführen, gehören Firecracker und Firekube von Weave.

Weitere Informationen zu den Bemühungen, Firecracker zu einer unterstützten Runtime für EKS zu machen, finden Sie unter 1238496944684597248.html. https://threadreaderapp.com/thread/

Öffnen Sie Policy Agent (OPA) und Gatekeeper

Gatekeeper ist ein Kubernetes-Zugangscontroller, der mit OPA erstellte Richtlinien durchsetzt. Mit OPA können Sie eine Richtlinie erstellen, die Pods von Mandanten auf separaten Instanzen oder mit einer höheren Priorität als andere Mandanten ausführt. Eine Sammlung gängiger OPA-Richtlinien finden Sie im GitHub Repository für dieses Projekt.

Es gibt auch ein experimentelles OPA-Plugin für CoreDNS, mit dem Sie OPA für die von CoreDNS zurückgegebenen Datensätze verwenden können. filter/control

Kyverno

Kyverno ist eine native Policy-Engine von Kubernetes, die Konfigurationen mit Richtlinien als Kubernetes-Ressourcen validieren, mutieren und generieren kann. Kyverno verwendet zur Validierung Overlays im Kustomize-Stil, unterstützt JSON-Patch und strategischen Merge-Patch für Mutationen und kann Ressourcen auf der Grundlage flexibler Trigger über Namespaces hinweg klonen.

Sie können Kyverno verwenden, um Namespaces zu isolieren, die Pod-Sicherheit und andere bewährte Methoden durchzusetzen und Standardkonfigurationen wie Netzwerkrichtlinien zu generieren. Das Repository für dieses Projekt enthält mehrere Beispiele. GitHub Viele andere sind in der Richtlinienbibliothek auf der Kyverno-Website enthalten.

Isolieren von Mandanten-Workloads auf bestimmte Knoten

Die Beschränkung der Mandanten-Workloads auf bestimmte Knoten kann dazu verwendet werden, die Isolation im Soft-Multi-Tenancy-Modell zu erhöhen. Bei diesem Ansatz werden mandantenspezifische Workloads nur auf Knoten ausgeführt, die für die jeweiligen Mandanten bereitgestellt wurden. Um diese Isolierung zu erreichen, werden native Kubernetes-Eigenschaften (Knotenaffinität sowie Taints and Tolerations) verwendet, um bestimmte Knoten für die Pod-

Planung ins Visier zu nehmen und zu verhindern, dass Pods von anderen Mandanten auf den mandantenspezifischen Knoten geplant werden.

Teil 1 — Knotenaffinität

Die Kubernetes-Knotenaffinität wird verwendet, um Knoten für die Planung auf der Grundlage von Knotenbezeichnungen als Ziel festzulegen. Mit den Regeln für die Knotenaffinität werden die Pods von bestimmten Knoten angezogen, die den Selektorbegriffen entsprechen. In der folgenden Pod-Spezifikation wird die `requiredDuringSchedulingIgnoredDuringExecution` Knotenaffinität auf den jeweiligen Pod angewendet. Das Ergebnis ist, dass der Pod auf Knoten abzielt, die mit dem folgenden Schlüssel/Wert gekennzeichnet sind:. `node-restriction.kubernetes.io/tenant: tenants-x`

```
...
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: node-restriction.kubernetes.io/tenant
            operator: In
            values:
            - tenants-x
...
```

Bei dieser Knotenaffinität ist das Label während der Planung erforderlich, aber nicht während der Ausführung. Wenn sich die Labels der zugrunde liegenden Knoten ändern, werden die Pods nicht allein aufgrund dieser Labeländerung entfernt. Die future Planung könnte jedoch beeinträchtigt werden.

> **Warning**
>
> Das Labelpräfix von `node-restriction.kubernetes.io/` hat in Kubernetes eine besondere Bedeutung. NodeRestrictionwelches für EKS-Cluster aktiviert ist, verhindert die Verwendung `kubelet` von adding/removing/updating Labels mit diesem Präfix. Angreifer können das nicht verwenden, `kubelet's credentials to update the node object or modify the system setup to pass these labels into` `kubelet` da es `kubelet` nicht erlaubt ist, diese Labels zu ändern. Wenn dieses Präfix für die gesamte Pod-zu-Node-Planung verwendet wird, verhindert es Szenarien, in denen ein Angreifer

möglicherweise andere Workloads auf einen Knoten übertragen möchte, indem er die Knotenbezeichnungen ändert.

Example

Statt der Knotenaffinität hätten wir den Node-Selector verwenden können. Die Knotenaffinität ist jedoch aussagekräftiger und ermöglicht es, beim Pod-Scheduling mehr Bedingungen zu berücksichtigen. Weitere Informationen zu den Unterschieden und den erweiterten Planungsmöglichkeiten finden Sie in diesem CNCF-Blogbeitrag zum Thema Advanced Kubernetes Pod-to-Node-Planung.

Teil 2 — Makel und Toleranzen

Das Anlocken von Pods zu Knoten ist nur der erste Teil dieses dreiteiligen Ansatzes. Damit dieser Ansatz funktioniert, müssen wir verhindern, dass Pods auf Knoten eingeplant werden, für die die Pods nicht autorisiert sind. Um unerwünschte oder nicht autorisierte Pods abzuwehren, verwendet Kubernetes Node-Taints. Taints werden verwendet, um Bedingungen auf Knoten festzulegen, die verhindern, dass Pods geplant werden. Der folgende Taint verwendet ein Schlüssel-Wert-Paar von. `tenant: tenants-x`

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

Angesichts des obigen Knotens dürfen nur Pods`taint`, die den Taint tolerieren, auf dem Knoten eingeplant werden. Damit autorisierte Pods auf dem Knoten eingeplant werden können, müssen die jeweiligen Pod-Spezifikationen, wie unten dargestellt, ein `toleration` „A“ für den Taint enthalten.

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

Pods mit den oben genannten Eigenschaften `toleration` werden nicht daran gehindert, auf dem Knoten zu planen, zumindest nicht aufgrund dieses spezifischen Fehlers. Taints werden auch von Kubernetes verwendet, um das Pod-Scheduling unter bestimmten Bedingungen vorübergehend zu unterbrechen, z. B. wenn die Knotenressourcen knapp sind. Mit Knotenaffinität und Taints und Toleranzen können wir die gewünschten Pods effektiv an bestimmte Nodes binden und unerwünschte Pods abwehren.

> **Important**
>
> Bestimmte Kubernetes-Pods müssen auf allen Knoten ausgeführt werden. Beispiele für diese Pods sind solche, die von den Daemonsets Container Network Interface (CNI) und Kube-Proxy gestartet werden. Aus diesem Grund enthalten die Spezifikationen für diese Pods sehr tolerante Toleranzen, um unterschiedliche Taints zu tolerieren. Es sollte darauf geachtet werden, dass diese Toleranzen nicht verändert werden. Eine Änderung dieser Toleranzen kann zu einem fehlerhaften Clusterbetrieb führen. Darüber hinaus können Tools zur Richtlinienverwaltung wie OPA/Gatekeeper und Kyverno verwendet werden, um validierende Richtlinien zu schreiben, die verhindern, dass unbefugte Pods diese toleranten Toleranzen nutzen.

Teil 3 — Policy-basiertes Management für die Knotenauswahl

Es gibt mehrere Tools, mit denen die Knotenaffinität und die Toleranzen von Pod-Spezifikationen verwaltet werden können, einschließlich der Durchsetzung von Regeln in CICD-Pipelines. Die Durchsetzung der Isolation sollte jedoch auch auf Kubernetes-Clusterebene erfolgen. Zu diesem Zweck können Policy-Management-Tools verwendet werden, um eingehende Kubernetes-API-Serveranfragen auf der Grundlage von Anforderungsnutzlasten zu mutieren, sodass die jeweiligen oben genannten Regeln und Toleranzen für die Knotenaffinität angewendet werden.

Beispielsweise können Pods, die für den Tenants-X-Namespace bestimmt sind, mit der richtigen Knotenaffinität und Toleranz versehen werden, um eine Planung auf den Tenants-X-Knoten zu ermöglichen. Mithilfe von Tools zur Richtlinienverwaltung, die mit dem Kubernetes Mutating Admission Webhook konfiguriert wurden, können Richtlinien verwendet werden, um die Spezifikationen für eingehende Pods zu ändern. Die Mutationen fügen die benötigten Elemente hinzu, um die gewünschte Planung zu ermöglichen. Im Folgenden finden Sie ein Beispiel OPA/Gatekeeper für eine Richtlinie, die eine Knotenaffinität hinzufügt.

```
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"
```

Die obige Richtlinie wird auf eine Kubernetes-API-Serveranfrage angewendet, um einen Pod auf den Tenants-x-Namespace anzuwenden. Die Richtlinie fügt die `requiredDuringSchedulingIgnoredDuringExecution` Knotenaffinitätsregel hinzu, sodass Pods von Knoten mit dem Label angezogen werden. `tenant: tenants-x`

Eine zweite Richtlinie (siehe unten) fügt die Toleranz derselben Pod-Spezifikation hinzu und verwendet dabei dieselben Übereinstimmungskriterien für Ziel-Namespace und Gruppen, Typen und Versionen.

```
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"
```

Die oben genannten Richtlinien sind spezifisch für Pods. Dies ist auf die Pfade zu den mutierten Elementen in den Elementen der Richtlinien zurückzuführen. `location` Zusätzliche Richtlinien könnten für den Umgang mit Ressourcen geschrieben werden, die Pods erstellen, z. B. Bereitstellungs- und Jobressourcen. Die aufgelisteten Richtlinien und andere Beispiele finden Sie im GitHubBegleitprojekt zu diesem Leitfaden.

Das Ergebnis dieser beiden Mutationen ist, dass die Hülsen vom gewünschten Knoten angezogen werden, während sie gleichzeitig nicht durch den spezifischen Knotenfleck abgestoßen werden. Um das zu überprüfen, können wir uns die Ausgabeschnipsel von zwei `kubectl` Aufrufen ansehen`tenant=tenants-x`, um die Nodes mit und die Pods im Namespace zu ermitteln. `tenants-x`

```
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...
```

Wie wir den obigen Ausgaben entnehmen können, sind alle Pods auf den mit beschrifteten Knoten geplant. `tenant=tenants-x` Einfach ausgedrückt, werden die Pods nur auf den gewünschten Knoten ausgeführt und die anderen Pods (ohne die erforderliche Affinität und Toleranzen) nicht. Die Workloads der Mandanten sind effektiv isoliert.

Ein Beispiel für eine mutierte Pod-Spezifikation finden Sie unten.

```
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**
>
> Tools zur Richtlinienverwaltung, die in den Anforderungsablauf des Kubernetes-API-Servers integriert sind und Webhooks für mutierende und validierende Zulassungen verwenden, sind so konzipiert, dass sie innerhalb eines bestimmten Zeitraums auf die Anfrage des API-Servers antworten. Dies sind normalerweise 3 Sekunden oder weniger. Wenn der

Webhook-Aufruf innerhalb der konfigurierten Zeit keine Antwort zurückgibt, kann die and/or Mutationsvalidierung der eingehenden API-Serveranfrage erfolgen oder auch nicht. Dieses Verhalten hängt davon ab, ob die Webhook-Konfigurationen für die Zulassung auf „Fail Open“ oder „Fail Close“ gesetzt sind.

In den obigen Beispielen haben wir Richtlinien verwendet, die für OPA/Gatekeeper geschrieben wurden. Es gibt jedoch auch andere Tools zur Richtlinienverwaltung, die unseren Anwendungsfall der Knotenauswahl behandeln. Diese Kyverno-Richtlinie könnte beispielsweise verwendet werden, um die Knotenaffinitätsmutation zu behandeln.

Note

Bei korrekter Funktionsweise wirken sich mutierende Richtlinien auf die gewünschten Änderungen an den Payloads für eingehende API-Serveranfragen aus. Es sollten jedoch auch validierende Richtlinien enthalten sein, um zu überprüfen, ob die gewünschten Änderungen tatsächlich vorgenommen werden, bevor Änderungen beibehalten werden. Dies ist besonders wichtig, wenn diese Richtlinien zur tenant-to-node Isolierung verwendet werden. Es ist auch eine gute Idee, Überwachungsrichtlinien einzubeziehen, um Ihren Cluster routinemäßig auf unerwünschte Konfigurationen zu überprüfen.

Referenzen

- k-rail Entwickelt, um Ihnen bei der Sicherung einer Umgebung mit mehreren Mandanten durch die Durchsetzung bestimmter Richtlinien zu helfen.
- Sicherheitspraktiken für MultiTenant SaaS-Anwendungen mit Amazon EKS

Harte Mehrmandantenfähigkeit

Hard-Multi-Tenancy kann implementiert werden, indem für jeden Mandanten separate Cluster bereitgestellt werden. Dies sorgt zwar für eine sehr starke Isolierung zwischen den Mandanten, hat jedoch mehrere Nachteile.

Erstens kann dieser Ansatz bei vielen Mietern schnell teuer werden. Sie müssen nicht nur die Kosten der Kontrollebene für jeden Cluster bezahlen, Sie können auch keine Rechenressourcen zwischen

Clustern gemeinsam nutzen. Dies führt letztendlich zu einer Fragmentierung, bei der eine Teilmenge Ihrer Cluster nicht ausgelastet ist, während andere überlastet sind.

Zweitens müssen Sie wahrscheinlich spezielle Tools kaufen oder entwickeln, um all diese Cluster zu verwalten. Mit der Zeit kann die Verwaltung von Hunderten oder Tausenden von Clustern einfach zu umständlich werden.

Schließlich wird die Erstellung eines Clusters pro Mandant im Vergleich zur Erstellung eines Namespaces langsam sein. Dennoch kann in stark regulierten Branchen oder in SaaS-Umgebungen, in denen eine starke Isolierung erforderlich ist, ein Hard-Tenancy-Ansatz erforderlich sein.

Künftige Richtungen

Die Kubernetes-Community hat die aktuellen Schwächen von Soft-Multi-Tenancy und die damit verbundenen Herausforderungen erkannt. Die Multi-Tenancy Special Interest Group (SIG) versucht, diese Mängel durch mehrere Inkubationsprojekte zu beheben, darunter Hierarchical Namespace Controller (HNC) und Virtual Cluster.

Der HNC-Vorschlag (KEP) beschreibt eine Möglichkeit, über- und untergeordnete Beziehungen zwischen Namespaces mithilfe der [Richtlinie] Objektvererbung herzustellen und Mandantenadministratoren die Möglichkeit zu bieten, Unter-Namespaces zu erstellen.

Der Vorschlag für einen virtuellen Cluster beschreibt einen Mechanismus zur Erstellung separater Instanzen der Steuerungsebenendienste, einschließlich des API-Servers, des Controller-Managers und des Schedulers, für jeden Mandanten innerhalb des Clusters (auch bekannt als „Kubernetes on Kubernetes“).

Der Vorschlag für Multi-Tenancy Benchmarks enthält Richtlinien für die gemeinsame Nutzung von Clustern mithilfe von Namespaces zur Isolierung und Segmentierung sowie ein Befehlszeilentool kubectl-mtb zur Überprüfung der Einhaltung der Richtlinien.

Verwaltungstools und Ressourcen für mehrere Cluster

- Banzai Cloud
- Kommandant
- Linse
- Nirmata
- Rafay
- Rancher

- Flux weben

Prüfung und Protokollierung

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Das Sammeln und Analysieren von [Audit-] Protokollen ist aus verschiedenen Gründen nützlich. Protokolle können bei der Ursachenanalyse und -zuweisung helfen, d. h. bei der Zuordnung einer Änderung zu einem bestimmten Benutzer. Wenn genügend Protokolle gesammelt wurden, können sie auch zur Erkennung ungewöhnlicher Verhaltensweisen verwendet werden. Auf EKS werden die Audit-Logs an Amazon Cloudwatch Logs gesendet. Die Audit-Richtlinie für EKS lautet wie folgt:

```
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"
```

Empfehlungen

Audit-Logs aktivieren

Die Auditprotokolle sind Teil der von EKS verwalteten Kubernetes-Steuerebenenprotokolle, die von EKS verwaltet werden. Anweisungen für enabling/disabling die Protokolle der Kontrollebene, zu denen die Protokolle für den Kubernetes-API-Server, den Controller-Manager und den Scheduler sowie das Audit-Protokoll gehören, finden Sie hier: -plane-logs.html# -export. https://docs.aws.amazon.com/eks/ latest/userguide/control enabling-control-plane-log

> **Note**
>
> Wenn Sie die Protokollierung auf der Kontrollebene aktivieren, fallen Kosten für das Speichern der Logs an. CloudWatch Dies wirft ein umfassenderes Problem in Bezug auf die laufenden Sicherheitskosten auf. Letztlich müssen Sie diese Kosten gegen die Kosten einer Sicherheitsverletzung abwägen, z. B. finanzielle Verluste, Rufschädigung usw. Möglicherweise stellen Sie fest, dass Sie Ihre Umgebung angemessen schützen können, wenn Sie nur einige der Empfehlungen in diesem Leitfaden umsetzen.

> **Warning**
>
> Die maximale Größe für einen CloudWatch Logs-Eintrag beträgt 1 MB, wohingegen die maximale Größe der Kubernetes-API-Anfrage 1,5 MiB beträgt. Protokolleinträge, die größer als 1 MB sind, werden entweder gekürzt oder enthalten nur die Metadaten der Anfrage.

Verwenden Sie Audit-Metadaten

Die Audit-Logs von Kubernetes enthalten zwei Anmerkungen, die angeben, ob eine Anfrage autorisiert wurde oder nicht, `authorization.k8s.io/decision` und den Grund für die Entscheidung. `authorization.k8s.io/reason` Verwenden Sie diese Attribute, um festzustellen, warum ein bestimmter API-Aufruf zulässig war.

Erstellen Sie Alarme für verdächtige Ereignisse

Erstellen Sie einen Alarm, der Sie automatisch benachrichtigt, wenn die Anzahl der Antworten 403 Verboten und 401 Unbefugt zunimmt, und verwenden Sie dann Attribute wie `hostsourceIPs`,, und, `k8s_user.username` um herauszufinden, woher diese Anfragen kommen.

Analysieren Sie Protokolle mit Log Insights

Verwenden Sie CloudWatch Log Insights, um Änderungen an RBAC-Objekten zu überwachen, z. B. Rollen, RoleBindings ClusterRoles, und. ClusterRoleBindings Nachfolgend finden Sie einige Beispielabfragen:

Listet Aktualisierungen für `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
```

Listet die Erstellung neuer oder Änderungen an Validierungs-Webhooks auf:

```
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
```

Listet Operationen zum Erstellen, Aktualisieren und Löschen von Rollen auf:

```
fields @timestamp, @message
```

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

Listet Operationen zum Erstellen, Aktualisieren und Löschen auf, um RoleBindings:

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

Listet Operationen zum Erstellen, Aktualisieren und Löschen auf für ClusterRoles:

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

Listet Operationen zum Erstellen, Aktualisieren und Löschen auf für ClusterRoleBindings:

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

Plottet unautorisierte Lesevorgänge gegen 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 der fehlgeschlagenen anonymen Anfragen:

```
fields @timestamp, @message, sourceIPs.0
```

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

Überprüfe deine CloudTrail Logs

APIs AWS-Aufrufe durch Pods, die IAM Roles for Service Accounts (IRSA) verwenden, werden automatisch CloudTrail zusammen mit dem Namen des Dienstkontos angemeldet. Wenn der Name eines Dienstkontos, das nicht ausdrücklich zum Aufrufen einer API autorisiert wurde, im Protokoll erscheint, kann dies ein Hinweis darauf sein, dass die Vertrauensrichtlinie der IAM-Rolle falsch konfiguriert war. Im Allgemeinen ist Cloudtrail eine hervorragende Möglichkeit, AWS-API-Aufrufe bestimmten IAM-Prinzipalen zuzuordnen.

Nutzen Sie Insights, um verdächtige Aktivitäten aufzudecken CloudTrail

CloudTrail Insights analysiert automatisch Ereignisse bei der Schreibverwaltung anhand von CloudTrail Pfaden und warnt Sie vor ungewöhnlichen Aktivitäten. Auf diese Weise können Sie feststellen, wann das Anrufvolumen beim Schreiben APIs in Ihrem AWS-Konto zunimmt, auch von Pods, die IRSA verwenden, um eine IAM-Rolle zu übernehmen. Weitere Informationen finden Sie unter Ankündigung von CloudTrail Erkenntnissen: Identifizierung und Reaktion auf ungewöhnliche API-Aktivitäten.

Weitere Ressourcen

Wenn das Volumen der Protokolle zunimmt, kann es unwirksam werden, sie mit Log Insights oder einem anderen Protokollanalysetool zu analysieren und zu filtern. Als Alternative sollten Sie in Betracht ziehen, Sysdig Falco und ekscloudwatch auszuführen. Falco analysiert Auditprotokolle und kennzeichnet Anomalien oder Missbrauch über einen längeren Zeitraum. Das ekscloudwatch-Projekt leitet Auditprotokollereignisse zur Analyse von Falco weiter. CloudWatch Falco bietet eine Reihe von Standard-Auditregeln sowie die Möglichkeit, eigene hinzuzufügen.

Eine weitere Option könnte darin bestehen, die Auditprotokolle in S3 zu speichern und den SageMaker Random Cut Forest-Algorithmus für anomale Verhaltensweisen zu verwenden, die weitere Untersuchungen rechtfertigen.

Tools und Ressourcen

Anhand der folgenden kommerziellen und Open-Source-Projekte können Sie beurteilen, ob Ihr Cluster an etablierten Best Practices ausgerichtet ist:

- Workshop zum Eintauchen in die Sicherheit von Amazon EKS — Detective Controls
- kubeaudit
- kube-scan weist den Workloads, die in Ihrem Cluster ausgeführt werden, eine Risikobewertung gemäß dem Kubernetes Common Configuration Scoring System-Framework zu
- kubesec.io
- Polaris
- Steuerbord
- Snyk
- Kubescape Kubescape ist ein Open-Source-Kubernetes-Sicherheitstool, das Cluster, YAML-Dateien und Helm-Diagramme scannt. Es erkennt Fehlkonfigurationen anhand mehrerer Frameworks (einschließlich NSA-CISA und MITRE ATT&CK®).

Netzwerksicherheit

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Netzwerksicherheit hat mehrere Facetten. Die erste beinhaltet die Anwendung von Regeln, die den Fluss des Netzwerkverkehrs zwischen Diensten einschränken. Die zweite beinhaltet die Verschlüsselung des Datenverkehrs während der Übertragung. Die Mechanismen zur Implementierung dieser Sicherheitsmaßnahmen auf EKS sind unterschiedlich, umfassen jedoch häufig die folgenden Elemente:

Verkehrskontrolle

- Netzwerkrichtlinien
- Sicherheitsgruppen

Netzwerkverschlüsselung

- Dienst Mesh
- Container-Netzwerkschnittstellen (CNIs)

- Ingress-Controller und Load Balancer
- Nitro-Instanzen
- ACM Private CA mit Cert-Manager

Netzwerkrichtlinie

Innerhalb eines Kubernetes-Clusters ist die gesamte Pod-zu-Pod-Kommunikation standardmäßig zulässig. Diese Flexibilität kann zwar dazu beitragen, Experimente zu fördern, wird aber nicht als sicher angesehen. Kubernetes-Netzwerkrichtlinien bieten Ihnen einen Mechanismus, mit dem Sie den Netzwerkverkehr zwischen Pods (oft als Ost-/West-Verkehr bezeichnet) sowie zwischen Pods und externen Diensten einschränken können. Kubernetes-Netzwerkrichtlinien funktionieren auf den Ebenen 3 und 4 des OSI-Modells. Netzwerkrichtlinien verwenden Pods, Namespace-Selektoren und Labels, um Quell- und Ziel-Pods zu identifizieren, können aber auch IP-Adressen, Portnummern, Protokolle oder eine Kombination davon enthalten. Netzwerkrichtlinien können sowohl auf eingehende als auch auf ausgehende Verbindungen zum Pod angewendet werden, was häufig als Eingangs- und Ausgangsregeln bezeichnet wird.

Mit der systemeigenen Netzwerkrichtlinien-Unterstützung des Amazon VPC CNI Plug-ins können Sie Netzwerkrichtlinien implementieren, um den Netzwerkverkehr in Kubernetes-Clustern zu sichern. Dies ist in die vorgelagerte Kubernetes Network Policy API integriert und gewährleistet so die Kompatibilität und Einhaltung der Kubernetes-Standards. Sie können Richtlinien mithilfe verschiedener Identifikatoren definieren, die von der Upstream-API unterstützt werden. Standardmäßig ist der gesamte eingehende und ausgehende Datenverkehr zu einem Pod zulässig. Wenn eine Netzwerkrichtlinie mit einem PolicyType Ingress angegeben ist, sind nur Verbindungen zum Pod zulässig, die vom Pod-Knoten ausgehen und die nach den Eingangsregeln zugelassenen Verbindungen. Das Gleiche gilt für Ausgangsregeln. Wenn mehrere Regeln definiert sind, wird bei der Entscheidung die Vereinigung aller Regeln berücksichtigt. Somit hat die Reihenfolge der Bewertung keinen Einfluss auf das politische Ergebnis.

> **Important**
>
> Wenn Sie einen EKS-Cluster zum ersten Mal bereitstellen, ist die VPC CNI-Netzwerkrichtlinien-Funktionalität standardmäßig nicht aktiviert. Stellen Sie sicher, dass Sie die unterstützte VPC CNI Add-on-Version bereitgestellt haben, und setzen Sie das `ENABLE_NETWORK_POLICY` Flag `true` auf dem vpc-cni-Add-on, um dies zu aktivieren. Ausführliche Anweisungen finden Sie im Amazon EKS-Benutzerhandbuch.

Empfehlungen

Erste Schritte mit Netzwerkrichtlinien — Folgen Sie dem Prinzip der geringsten Rechte

Erstellen Sie eine standardmäßige Ablehnungsrichtlinie

Wie bei RBAC-Richtlinien wird auch bei Netzwerkrichtlinien empfohlen, die Prinzipien des geringsten Zugriffs zu befolgen. Erstellen Sie zunächst eine Deny-All-Richtlinie, die den gesamten eingehenden und ausgehenden Datenverkehr in einem Namespace einschränkt.

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

Standardmäßige Ablehnung

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

> **Note**
>
> Das Bild oben wurde mit dem Network Policy Viewer von Tufin erstellt.

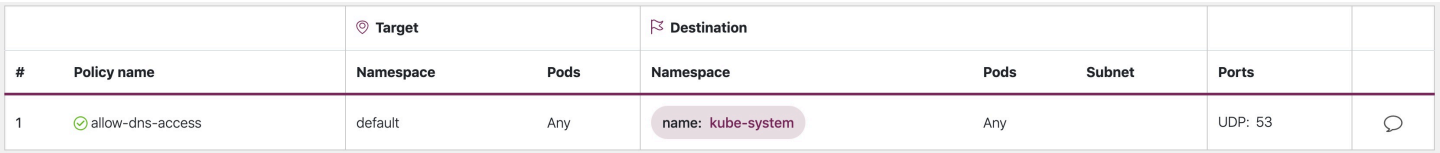
Erstellen Sie eine Regel, um DNS-Abfragen zuzulassen

Sobald Sie die Standardregel „Alle ablehnen“ eingerichtet haben, können Sie zusätzliche Regeln hinzufügen, z. B. eine Regel, die es Pods ermöglicht, CoreDNS zur Namensauflösung abzufragen.

```
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	

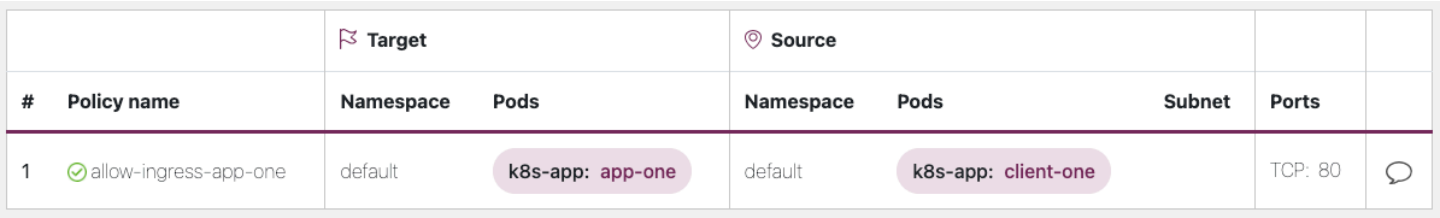
Fügen Sie schrittweise Regeln hinzu, um den Verkehrsfluss zwischen Namespaces/Pods selektiv zuzulassen

Machen Sie sich mit den Anwendungsanforderungen vertraut und erstellen Sie nach Bedarf detaillierte Eingangs- und Ausgangsregeln. Das folgende Beispiel zeigt, wie Sie den eingehenden Verkehr auf Port 80 auf von beschränken können. `app-one` `client-one` Dies trägt dazu bei, die Angriffsfläche zu minimieren und das Risiko eines unbefugten Zugriffs zu verringern.

```
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	

Überwachung der Durchsetzung von Netzwerkrichtlinien

- Verwenden Sie den Netzwerkrichtlinien-Editor
  - Der Netzwerkrichtlinien-Editor hilft bei Visualisierungen, Sicherheitsbewertungen und generiert automatisch aus Netzwerkflussprotokollen
  - Erstellen Sie Netzwerkrichtlinien auf interaktive Weise
- Audit-Protokolle
  - Überprüfen Sie regelmäßig die Auditprotokolle Ihres EKS-Clusters
  - Auditprotokolle enthalten eine Fülle von Informationen darüber, welche Aktionen auf Ihrem Cluster durchgeführt wurden, einschließlich Änderungen der Netzwerkrichtlinien
  - Verwenden Sie diese Informationen, um Änderungen an Ihren Netzwerkrichtlinien im Laufe der Zeit nachzuverfolgen und unbefugte oder unerwartete Änderungen zu erkennen
- Automatisiertes Testen
  - Implementieren Sie automatisierte Tests, indem Sie eine Testumgebung einrichten, die Ihre Produktionsumgebung widerspiegelt, und regelmäßig Workloads bereitstellen, die versuchen, Ihre Netzwerkrichtlinien zu verletzen.
- Metriken überwachen
  - Konfigurieren Sie Ihre Observability-Agents so, dass sie die Prometheus-Metriken von den VPC-CNI-Knotenagenten abrufen, sodass Sie den Zustand der Agenten und SDK-Fehler überwachen können.

- Überprüfen Sie die Netzwerkrichtlinien regelmäßig
  - Überprüfen Sie Ihre Netzwerkrichtlinien regelmäßig, um sicherzustellen, dass sie Ihren aktuellen Anwendungsanforderungen entsprechen. Im Zuge der Weiterentwicklung Ihrer Anwendung haben Sie durch ein Audit die Möglichkeit, redundante Eingangs- und Ausgangsregeln zu entfernen und sicherzustellen, dass Ihre Anwendungen nicht über zu viele Berechtigungen verfügen.
- Stellen Sie mithilfe von Open Policy Agent (OPA) sicher, dass Netzwerkrichtlinien existieren
  - Verwenden Sie die OPA-Richtlinie wie unten gezeigt, um sicherzustellen, dass Netzwerkrichtlinien immer vorhanden sind, bevor Sie Anwendungs-Pods integrieren. Diese Richtlinie verweigert das Onboarding von k8s-Pods mit einem Label, `k8s-app: sample-app` wenn keine entsprechende Netzwerkrichtlinie existiert.

```
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
}
```

Fehlerbehebung

Überwachen Sie die Node-Agent-Protokolle vpc-network-policy-controller

Aktivieren Sie die EKS Control Plane Controller Manager-Protokolle, um die Funktionalität der Netzwerkrichtlinien zu diagnostizieren. Sie können die Protokolle der Kontrollebene in eine CloudWatch Protokollgruppe streamen und mithilfe von CloudWatchLog Insights erweiterte Abfragen durchführen. In den Protokollen können Sie sehen, welche Pod-Endpunktobjekte in eine Netzwerkrichtlinie aufgelöst wurden, den Abgleichstatus der Richtlinien und das Debuggen, ob die Richtlinie erwartungsgemäß funktioniert.

Darüber hinaus können Sie mit Amazon VPC CNI die Erfassung und den Export von Protokollen zur Durchsetzung von Richtlinien aus den EKS-Worker-Knoten nach Amazon Cloudwatch aktivieren. Nach der Aktivierung können Sie CloudWatchContainer Insights nutzen, um Einblicke in Ihre Nutzung im Zusammenhang mit Netzwerkrichtlinien zu gewinnen.

Amazon VPC CNI bietet auch ein SDK, das eine Schnittstelle für die Interaktion mit eBPF-Programmen auf dem Knoten bereitstellt. Das SDK wird installiert, wenn es auf den `aws-node` Knoten bereitgestellt wird. Sie finden die SDK-Binärdatei, die im `/opt/cni/bin` Verzeichnis auf dem Knoten installiert ist. Bei der Markteinführung bietet das SDK Unterstützung für grundlegende Funktionen wie die Inspektion eBPF eBPF-Programmen und -Maps.

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

Protokollieren Sie Metadaten zum Netzwerkverkehr

AWS VPC Flow Logs erfasst Metadaten über den Datenverkehr, der durch eine VPC fließt, wie Quell- und Ziel-IP-Adresse und Port sowie akzeptierte/verworfene Pakete. Diese Informationen könnten analysiert werden, um nach verdächtigen oder ungewöhnlichen Aktivitäten zwischen Ressourcen innerhalb der VPC, einschließlich Pods, zu suchen. Da sich die IP-Adressen von Pods jedoch häufig ändern, wenn sie ausgetauscht werden, reichen Flow Logs allein möglicherweise nicht aus. Calico Enterprise erweitert die Flow Logs um Pod-Labels und andere Metadaten, was es einfacher macht, die Verkehrsflüsse zwischen Pods zu entziffern.

Sicherheitsgruppen

EKS verwendet AWS VPC Security Groups (SGs), um den Verkehr zwischen der Kubernetes-Steuerebene und den Worker-Knoten des Clusters zu kontrollieren. Sicherheitsgruppen werden auch verwendet, um den Verkehr zwischen Worker-Knoten und anderen VPC-Ressourcen sowie externen IP-Adressen zu kontrollieren. Wenn Sie einen EKS-Cluster (mit Kubernetes Version 1.14-eks.3 oder höher) bereitstellen, wird automatisch eine Cluster-Sicherheitsgruppe für Sie erstellt. Diese Sicherheitsgruppe ermöglicht die uneingeschränkte Kommunikation zwischen der EKS-Steuerebene und den Knoten aus verwalteten Knotengruppen. Der Einfachheit halber wird empfohlen, den Cluster-SG zu allen Knotengruppen hinzuzufügen, einschließlich nicht verwalteter Knotengruppen.

Vor der Kubernetes-Version 1.14 und der EKS-Version eks.3 wurden separate Sicherheitsgruppen für die EKS-Steuerungsebene und die Knotengruppen konfiguriert. Die Mindestregeln und die empfohlenen Regeln für die Sicherheitsgruppen der Kontrollebene und der Knotengruppen finden Sie unter -group-reqs.html. https://docs.aws.amazon.com/eks/ latest/userguide/sec Die Mindestregeln für

die Sicherheitsgruppe der Kontrollebene lassen den eingehenden Port 443 vom Worker-Knoten SG zu. Diese Regel ermöglicht es den Kubelets, mit dem Kubernetes-API-Server zu kommunizieren. Sie umfasst auch Port 10250 für ausgehenden Datenverkehr zum Worker-Knoten SG; 10250 ist der Port, auf dem die Kubelets lauschen. In ähnlicher Weise ermöglichen die Mindestregeln für Knotengruppen den eingehenden Port 10250 von der Steuerungsebene SG und den ausgehenden Port 443 zur Steuerungsebene SG. Schließlich gibt es eine Regel, die eine uneingeschränkte Kommunikation zwischen Knoten innerhalb einer Knotengruppe ermöglicht.

Wenn Sie die Kommunikation zwischen Diensten, die innerhalb des Clusters ausgeführt werden, und Diensten, die außerhalb des Clusters ausgeführt werden, wie z. B. einer RDS-Datenbank, kontrollieren müssen, sollten Sie Sicherheitsgruppen für Pods in Betracht ziehen. Mit Sicherheitsgruppen für Pods können Sie einer Sammlung von Pods eine bestehende Sicherheitsgruppe zuweisen.

> **Warning**
>
> Wenn Sie auf eine Sicherheitsgruppe verweisen, die vor der Erstellung der Pods nicht vorhanden war, werden die Pods nicht geplant.

Sie können steuern, welche Pods einer Sicherheitsgruppe zugewiesen werden, indem Sie ein `SecurityGroupPolicy` Objekt erstellen und a `PodSelector` oder a angeben`ServiceAccountSelector`. Wenn Sie die Selektoren auf setzen, `{}` werden die in der SGs referenzierten `SecurityGroupPolicy` Objekte allen Pods in einem Namespace oder allen Dienstkonten in einem Namespace zugewiesen. Stellen Sie sicher, dass Sie sich mit allen Überlegungen vertraut gemacht haben, bevor Sie Sicherheitsgruppen für Pods implementieren.

> **Important**
>
> Wenn Sie Pods verwenden SGs , müssen Sie festlegen, SGs dass Port 53 ausgehende Verbindungen zur Cluster-Sicherheitsgruppe zulassen. Ebenso müssen Sie die Cluster-Sicherheitsgruppe so aktualisieren, dass sie eingehenden Port 53-Datenverkehr von der Pod-Sicherheitsgruppe akzeptiert.

> **Important**
>
> Die Grenzwerte für Sicherheitsgruppen gelten weiterhin, wenn Sicherheitsgruppen für Pods verwendet werden. Verwenden Sie sie daher mit Bedacht.

> **Important**
>
> Sie müssen Regeln für eingehenden Datenverkehr aus der Cluster-Sicherheitsgruppe (Kubelet) für alle für den Pod konfigurierten Tests erstellen.

> **Important**
>
> Sicherheitsgruppen für Pods basieren auf einer als ENI-Trunking bekannten Funktion, die entwickelt wurde, um die ENI-Dichte einer EC2-Instance zu erhöhen. Wenn ein Pod einem SG zugewiesen wird, ordnet ein VPC-Controller dem Pod ein Branch-ENI aus der Knotengruppe zu. Wenn zum Zeitpunkt der Planung des Pods nicht genügend Branches in einer Knotengruppe ENIs verfügbar sind, verbleibt der Pod im Status „Ausstehend“. Die Anzahl der Branches, die ENIs eine Instance unterstützen kann, variiert je nach Instanz type/family. See https://docs.aws.amazon.com/eks/latest/userguide/security — groups-for-pods .html# supported-instance-types für weitere Informationen.

Sicherheitsgruppen für Pods bieten zwar eine AWS-native Möglichkeit, den Netzwerkverkehr innerhalb und außerhalb Ihres Clusters ohne den Aufwand eines Policy-Daemons zu steuern, aber es sind auch andere Optionen verfügbar. Mit der Cilium-Policy-Engine können Sie beispielsweise in einer Netzwerkrichtlinie auf einen DNS-Namen verweisen. Calico Enterprise bietet eine Option für die Zuordnung von Netzwerkrichtlinien zu AWS-Sicherheitsgruppen. Wenn Sie ein Service Mesh wie Istio implementiert haben, können Sie ein Ausgangsgateway verwenden, um den Netzwerkausgang auf bestimmte, voll qualifizierte Domänen oder IP-Adressen zu beschränken. Weitere Informationen zu dieser Option finden Sie in der dreiteiligen Serie zur Steuerung des ausgehenden Datenverkehrs in Istio.

Wann sollten Netzwerkrichtlinien und wann Sicherheitsgruppen für Pods verwendet werden?

Wann sollte die Kubernetes-Netzwerkrichtlinie verwendet werden

- Den Verkehr kontrollieren pod-to-pod
  - Geeignet für die Steuerung des Netzwerkverkehrs zwischen Pods innerhalb eines Clusters (Ost-West-Verkehr)
- Steuern Sie den Verkehr auf IP-Adresse- oder Portebene (OSI-Schicht 3 oder 4)

Wann sollten AWS-Sicherheitsgruppen für Pods (SGP) verwendet werden

- Nutzung vorhandener AWS-Konfigurationen
  - Wenn Sie bereits über einen komplexen Satz von EC2-Sicherheitsgruppen verfügen, die den Zugriff auf AWS-Services verwalten, und Sie Anwendungen von EC2-Instances zu EKS migrieren, SGPs kann dies eine sehr gute Wahl sein, da Sie Sicherheitsgruppenressourcen wiederverwenden und auf Ihre Pods anwenden können.
- Steuern Sie den Zugriff auf AWS-Services
  - Ihre Anwendungen, die in einem EKS-Cluster ausgeführt werden, möchten mit anderen AWS-Services (RDS-Datenbank) kommunizieren und SGPs als effizienten Mechanismus zur Steuerung des Datenverkehrs von den Pods zu AWS-Services verwendet werden.
- Isolierung des Pod- und Node-Datenverkehrs
  - Wenn Sie den Pod-Verkehr vollständig vom restlichen Node-Verkehr trennen möchten, verwenden Sie SGP im `POD_SECURITY_GROUP_ENFORCING_MODE=strict` Modus.

Bewährte Methoden bei der Verwendung von Sicherheitsgruppen für Pods und Netzwerkrichtlinien

- Mehrschichtige Sicherheit
  - Verwenden Sie eine Kombination aus SGP- und Kubernetes-Netzwerkrichtlinien für einen mehrschichtigen Sicherheitsansatz
  - Wird verwendet SGPs , um den Zugriff auf Netzwerkebene auf AWS-Services zu beschränken, die nicht Teil eines Clusters sind, während Kubernetes-Netzwerkrichtlinien den Netzwerkverkehr zwischen Pods innerhalb des Clusters einschränken können

- Prinzip der geringsten Rechte
  - Lassen Sie nur den erforderlichen Verkehr zwischen Pods oder Namespaces zu
- Segmentieren Sie Ihre Anwendungen
  - Segmentieren Sie Anwendungen wo immer möglich nach Netzwerkrichtlinien, um den Explosionsradius zu reduzieren, falls eine Anwendung gefährdet ist
- Halten Sie die Richtlinien einfach und übersichtlich
  - Kubernetes-Netzwerkrichtlinien können sehr detailliert und komplex sein. Es ist am besten, sie so einfach wie möglich zu halten, um das Risiko einer Fehlkonfiguration zu verringern und den Verwaltungsaufwand zu verringern
- Reduzieren Sie die Angriffsfläche
  - Minimieren Sie die Angriffsfläche, indem Sie die Gefährdung Ihrer Anwendungen einschränken

> **Important**
>
> Sicherheitsgruppen für Pods bieten zwei Erzwingungsmodi: `strict` und`standard`. Sie müssen `standard` den Modus verwenden, wenn Sie sowohl Netzwerkrichtlinien- als auch Sicherheitsgruppen für Pods in einem EKS-Cluster verwenden.

Wenn es um Netzwerksicherheit geht, ist ein mehrschichtiger Ansatz oft die effektivste Lösung. Die Kombination von Kubernetes-Netzwerkrichtlinien und SGP kann eine robuste defense-in-depth Strategie für Ihre Anwendungen bieten, die in EKS ausgeführt werden.

Durchsetzung von Service Mesh-Richtlinien oder Kubernetes-Netzwerkrichtlinien

A `service mesh` ist eine dedizierte Infrastrukturebene, die Sie Ihren Anwendungen hinzufügen können. Sie ermöglicht es Ihnen, Funktionen wie Beobachtbarkeit, Verkehrsmanagement und Sicherheit transparent hinzuzufügen, ohne sie Ihrem eigenen Code hinzufügen zu müssen.

Service Mesh setzt Richtlinien auf Ebene 7 (Anwendung) des OSI-Modells durch, wohingegen Kubernetes-Netzwerkrichtlinien auf Ebene 3 (Netzwerk) und Schicht 4 (Transport) angewendet werden. In diesem Bereich gibt es viele Angebote wie AWSAppMesh, Istio, Linkerd usw.

Wann sollte Service Mesh zur Durchsetzung von Richtlinien verwendet werden

- Haben Sie bereits in ein Service Mesh investiert
- Benötigen Sie erweiterte Funktionen wie Verkehrsmanagement, Beobachtbarkeit und Sicherheit
  - Verkehrskontrolle, Lastenausgleich, Stromunterbrechung, Ratenbegrenzung, Timeouts usw.
  - Detaillierte Einblicke in die Leistung Ihrer Dienste (Latenz, Fehlerraten, Anfragen pro Sekunde, Anforderungsvolumen usw.)
  - Sie möchten Service Mesh für Sicherheitsfunktionen wie mTLS implementieren und nutzen

Wählen Sie die Kubernetes-Netzwerkrichtlinie für einfachere Anwendungsfälle

- Beschränken Sie, welche Pods miteinander kommunizieren können
- Netzwerkrichtlinien benötigen weniger Ressourcen als ein Service Mesh und eignen sich daher gut für einfachere Anwendungsfälle oder für kleinere Cluster, bei denen der Aufwand für den Betrieb und die Verwaltung eines Service Mesh möglicherweise nicht gerechtfertigt ist

> **Note**
>
> Netzwerkrichtlinien und Service Mesh können auch zusammen verwendet werden. Verwenden Sie Netzwerkrichtlinien, um ein grundlegendes Maß an Sicherheit und Isolierung zwischen Ihren Pods zu gewährleisten, und verwenden Sie dann ein Service Mesh, um zusätzliche Funktionen wie Verkehrsmanagement, Beobachtbarkeit und Sicherheit hinzuzufügen.

ThirdParty Engines für Netzwerkrichtlinien

Ziehen Sie eine Network Policy Engine eines Drittanbieters in Betracht, wenn Sie erweiterte Richtlinienanforderungen wie globale Netzwerkrichtlinien, Unterstützung für auf DNS-Hostnamen basierende Regeln, Layer-7-Regeln, ServiceAccount basierte Regeln und explizite deny/log Aktionen usw. haben. Calico ist eine Open-Source-Policy-Engine von Tigera, die gut mit EKS zusammenarbeitet. Calico implementiert nicht nur alle Funktionen für Kubernetes-Netzwerkrichtlinien, sondern unterstützt auch erweiterte Netzwerkrichtlinien mit einem umfangreicheren Funktionsumfang, einschließlich der Unterstützung von Layer-7-Regeln, z. B. HTTP, wenn es in Istio integriert ist. Calico-Richtlinien können auf Namespaces, Pods, Dienstkonten oder global beschränkt werden.

Wenn Richtlinien auf ein Dienstkonto beschränkt sind, ordnet es diesem Dienstkonto eine Reihe von Regeln zu. ingress/egress Mit den richtigen RBAC-Regeln können Sie verhindern, dass Teams diese Regeln außer Kraft setzen, sodass IT-Sicherheitsexperten die Verwaltung von Namespaces sicher delegieren können. Isovalent, die Entwickler von Cilium, haben außerdem die Netzwerkrichtlinien erweitert und unterstützen nun teilweise Layer-7-Regeln, z. B. HTTP. Cilium unterstützt auch DNS-Hostnamen, was nützlich sein kann, um den Verkehr zwischen Kubernetes Services/Pods und Ressourcen, die innerhalb oder außerhalb Ihrer VPC laufen, einzuschränken. Im Gegensatz dazu enthält Calico Enterprise eine Funktion, mit der Sie eine Kubernetes-Netzwerkrichtlinie einer AWS-Sicherheitsgruppe sowie DNS-Hostnamen zuordnen können.

Eine Liste gängiger Kubernetes-Netzwerkrichtlinien finden Sie unter. https://github.com/ahmetb/ kubernetes-network-policy-recipes Ein ähnliches Regelwerk für Calico ist unter https://docs.projectcalico verfügbar. org/security/calico-Netzwerkpolitik.

Migration zur Amazon VPC CNI Network Policy Engine

Um die Konsistenz aufrechtzuerhalten und ein unerwartetes Verhalten bei der Pod-Kommunikation zu vermeiden, wird empfohlen, nur eine Network Policy Engine in Ihrem Cluster bereitzustellen. Wenn Sie von 3P zur VPC CNI Network Policy Engine migrieren möchten, empfehlen wir, Ihre vorhandenen 3P-Ressourcen in die NetworkPolicy Kubernetes-Ressourcen NetworkPolicy CRDs zu konvertieren, bevor Sie die VPC CNI-Netzwerkrichtlinienunterstützung aktivieren. Testen Sie außerdem die migrierten Richtlinien in einem separaten Testcluster, bevor Sie sie in Ihrer Produktionsumgebung anwenden. Auf diese Weise können Sie potenzielle Probleme oder Inkonsistenzen im Kommunikationsverhalten von Pods identifizieren und beheben.

Migrationstool

Um Sie bei Ihrem Migrationsprozess zu unterstützen, haben wir ein Tool namens K8s Network Policy Migrator entwickelt, das Ihre bestehenden Netzwerkrichtlinien in native Calico/Cilium Kubernetes-Netzwerkrichtlinien CRDs umwandelt. Nach der Konvertierung können Sie die konvertierten Netzwerkrichtlinien direkt auf Ihren neuen Clustern testen, auf denen der VPC CNI-Netzwerkrichtlinien-Controller ausgeführt wird. Das Tool soll Ihnen helfen, den Migrationsprozess zu rationalisieren und einen reibungslosen Übergang zu gewährleisten.

> **Important**
>
> Das Migrationstool konvertiert nur 3P-Richtlinien, die mit der nativen Kubernetes-Netzwerkrichtlinien-API kompatibel sind. Wenn Sie erweiterte Funktionen für

Netzwerkrichtlinien verwenden, die von 3P-Plugins angeboten werden, überspringt das Migrationstool diese und meldet sie.

Bitte beachten Sie, dass das Migrationstool derzeit nicht vom AWS VPC CNI Network Policy Engineering-Team unterstützt wird. Es wird Kunden nach bestem Wissen zur Verfügung gestellt. Wir empfehlen Ihnen, dieses Tool zu verwenden, um Ihren Migrationsprozess zu vereinfachen. Falls Sie auf Probleme oder Bugs mit dem Tool stoßen, bitten wir Sie, ein GitHubProblem zu erstellen. Ihr Feedback ist für uns von unschätzbarem Wert und trägt zur kontinuierlichen Verbesserung unserer Dienstleistungen bei.

Weitere Ressourcen

- Kubernetes & Tigera: Netzwerkrichtlinien, Sicherheit und Prüfung
- Calico Enterprise
- Cilium
- NetworkPolicyEditor, ein interaktiver Policy-Editor von Cilium
- Inspektor Gadget Advice Netzwerkrichtlinien-Gadget Schlägt Netzwerkrichtlinien auf der Grundlage einer Analyse des Netzwerkverkehrs vor

Verschlüsselung während der Übertragung

Anwendungen, die PCI-, HIPAA- oder anderen Vorschriften entsprechen müssen, müssen möglicherweise Daten während der Übertragung verschlüsseln. Heutzutage ist TLS de facto die erste Wahl für die Verschlüsselung des Datenverkehrs auf der Leitung. TLS bietet, wie sein Vorgänger SSL, sichere Kommunikation über ein Netzwerk mithilfe kryptografischer Protokolle. TLS verwendet eine symmetrische Verschlüsselung, bei der die Schlüssel zur Verschlüsselung der Daten auf der Grundlage eines gemeinsamen Geheimnisses generiert werden, das zu Beginn der Sitzung ausgehandelt wird. Im Folgenden finden Sie einige Möglichkeiten, wie Sie Daten in einer Kubernetes-Umgebung verschlüsseln können.

Nitro-Instanzen

Der Datenverkehr, der zwischen den folgenden Nitro-Instance-Typen, z. B. C5n, G4, I3en, M5dn, M5n, P3dn, R5dn und R5n, ausgetauscht wird, wird standardmäßig automatisch verschlüsselt. Wenn es einen Intermediate-Hop gibt, wie ein Transit-Gateway oder ein Load Balancer, ist der

Datenverkehr nicht verschlüsselt. Weitere Informationen zur Verschlüsselung bei der Übertragung sowie eine vollständige Liste der Instance-Typen, die Netzwerkverschlüsselung standardmäßig unterstützen, finden Sie unter Verschlüsselung bei der Übertragung.

Container-Netzwerkschnittstellen (CNIs)

WeaveNetkann so konfiguriert werden, dass der gesamte Datenverkehr automatisch verschlüsselt wird, wobei NaCl Verschlüsselung für Sleeve-Verkehr und IPsec ESP für schnellen Datenpfadverkehr verwendet wird.

Service Mesh

Die Verschlüsselung bei der Übertragung kann auch mit einem Service Mesh wie App Mesh, Linkerd v2 und Istio implementiert werden. AppMesh unterstützt mTLS mit X.509-Zertifikaten oder den Secret Discovery Service (SDS) von Envoy. Linkerd und Istio unterstützen beide mTLS.

Das aws-app-mesh-examplesGitHub Repository bietet exemplarische Vorgehensweisen für die Konfiguration von mTLS mithilfe von X.509-Zertifikaten und SPIRE als SDS-Anbieter mit Ihrem Envoy-Container:

- Konfiguration von mTLS mithilfe von X.509-Zertifikaten
- Konfiguration von TLS mit SPIRE (SDS)

App Mesh unterstützt auch die TLS-Verschlüsselung mit einem privaten Zertifikat, das von AWS Certificate Manager (ACM) ausgestellt wurde, oder einem Zertifikat, das im lokalen Dateisystem des virtuellen Knotens gespeichert ist.

Das aws-app-mesh-examplesGitHub Repository bietet exemplarische Vorgehensweisen für die Konfiguration von TLS mithilfe von Zertifikaten, die von ACM ausgestellt wurden, und Zertifikaten, die in Ihrem Envoy-Container enthalten sind:

- Konfiguration von TLS mit per Datei bereitgestellten TLS-Zertifikaten
- Konfiguration von TLS mit AWS Certificate Manager

Ingress-Controller und Load Balancer

Ingress-Controller bieten Ihnen die Möglichkeit, die 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 Verarbeitung, die sehr CPU-intensiv ist, intelligent weiterzuleiten. Wenn Sie also die nötige Flexibilität haben, versuchen Sie, den SSL-Offload am Ingress oder am Load Balancer durchzuführen.

Verschlüsselung mit AWS Elastic Load Balancers verwenden

Der AWS Application Load Balancer (ALB) und der Network Load Balancer (NLB) unterstützen beide die Transportverschlüsselung (SSL und TLS). Mit der `alb.ingress.kubernetes.io/certificate-arn` Anmerkung für den ALB können Sie angeben, welche Zertifikate dem ALB hinzugefügt werden sollen. Wenn Sie die Anmerkung weglassen, versucht der Controller, Listenern, die dies benötigen, Zertifikate hinzuzufügen, indem er die verfügbaren AWS Certificate Manager (ACM) -Zertifikate mithilfe des Host-Felds abgleicht. Ab EKS v1.15 können Sie die `service.beta.kubernetes.io/aws-load-balancer-ssl-cert` Anmerkung mit der NLB verwenden, wie im folgenden Beispiel gezeigt.

```
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
```

Im Folgenden finden Sie weitere Beispiele für SSL/TLS eine Kündigung.

- Schützen Sie EKS Ingress mit Contour und lassen Sie uns nebenbei noch verschlüsseln GitOps
- Wie beende ich den HTTPS-Verkehr auf Amazon EKS-Workloads mit ACM?

> **Important**
>
> Einige Ingresses, wie der AWS LB-Controller, implementieren die SSL/TLS Verwendung von Anmerkungen, anstatt sie als Teil der Ingress-Spezifikation zu verwenden.

ACM Private CA mit Cert-Manager

Mithilfe von ACM Private Certificate Authority (CA) und cert-manager, einem beliebten Kubernetes-Add-on zum Verteilen, Erneuern und Widerrufen von Zertifikaten, können Sie TLS und mTLS

aktivieren, um Ihre EKS-Anwendungsworkloads am Eingang, auf dem Pod und zwischen Pods zu sichern. ACM Private CA ist eine hochverfügbare, sichere und verwaltete CA ohne die Vorab- und Wartungskosten, die mit der Verwaltung Ihrer eigenen CA verbunden wären. Wenn Sie die standardmäßige Kubernetes-Zertifizierungsstelle verwenden, besteht die Möglichkeit, Ihre Sicherheit zu verbessern und die Compliance-Anforderungen mit ACM Private CA zu erfüllen. ACM Private CA schützt private Schlüssel in FIPS 140-2 Level 3-Hardwaresicherheitsmodulen (sehr sicher), verglichen mit der Standard-CA, die Schlüssel im Speicher speichert (weniger sicher). Eine zentralisierte CA bietet Ihnen außerdem mehr Kontrolle und verbesserte Überprüfbarkeit für private Zertifikate sowohl innerhalb als auch außerhalb einer Kubernetes-Umgebung.

Kurzlebiger CA-Modus für wechselseitiges TLS zwischen Workloads

Bei der Verwendung von ACM Private CA für mTLS in EKS wird empfohlen, kurzlebige Zertifikate im kurzlebigen CA-Modus zu verwenden. Es ist zwar möglich, kurzlebige Zertifikate im allgemeinen CA-Modus auszustellen, aber der kurzlebige CA-Modus ist kostengünstiger (~ 75% günstiger als der allgemeine Modus) für Anwendungsfälle, in denen häufig neue Zertifikate ausgestellt werden müssen. Darüber hinaus sollten Sie versuchen, die Gültigkeitsdauer der privaten Zertifikate an die Lebensdauer der Pods in Ihrem EKS-Cluster anzupassen. Erfahren Sie hier mehr über ACM Private CA und seine Vorteile.

Anweisungen zur Einrichtung von ACM

Erstellen Sie zunächst eine private CA, indem Sie die in den technischen Dokumenten zu ACM Private CA beschriebenen Verfahren befolgen. Sobald Sie über eine private CA verfügen, installieren Sie cert-manager mithilfe der regulären Installationsanweisungen. Installieren Sie nach der Installation von cert-manager das Private CA Kubernetes cert-manager-Plugin, indem Sie den Installationsanweisungen unter folgen. GitHub Mit dem Plugin kann der Cert-Manager private Zertifikate von ACM Private CA anfordern.

Nachdem Sie nun eine private CA und einen EKS-Cluster mit installiertem Cert-Manager und dem Plugin haben, ist es an der Zeit, die Berechtigungen festzulegen und den Aussteller zu erstellen. Aktualisieren Sie die IAM-Berechtigungen der EKS-Knotenrolle, um den Zugriff auf ACM Private CA zu ermöglichen. Ersetzen Sie das <CA_ARN> durch den Wert Ihrer privaten CA:

```
{
    "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"
        }
    ]
}
```

Servicerollen für IAM-Konten oder IRSA können ebenfalls verwendet werden. Vollständige Beispiele finden Sie im Abschnitt Zusätzliche Ressourcen weiter unten.

Erstellen Sie einen Aussteller in Amazon EKS, indem Sie eine benutzerdefinierte Ressourcendefinitionsdatei mit dem Namen cluster-issuer.yaml mit dem folgenden Text erstellen `<CA_ARN>` und `<Region>` die Informationen durch Ihre private CA ersetzen.

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

Stellen Sie den von Ihnen erstellten Issuer bereit.

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

Ihr EKS-Cluster ist so konfiguriert, dass er Zertifikate von Private CA anfordert. Sie können jetzt die `Certificate` Ressource von cert-manager verwenden, um Zertifikate auszustellen, indem Sie die `issuerRef` Feldwerte auf den Private CA-Aussteller ändern, den Sie oben erstellt haben. Weitere Informationen zur Angabe und Anforderung von Zertifikatsressourcen finden Sie im Certificate Resources Guide von cert-manager. Beispiele finden Sie hier.

ACM Private CA mit Istio und Cert-Manager

Wenn Sie Istio in Ihrem EKS-Cluster ausführen, können Sie die Istio-Steuerebene (insbesondere`istiod`) so deaktivieren, dass sie als Stammzertifizierungsstelle (CA) fungiert, und

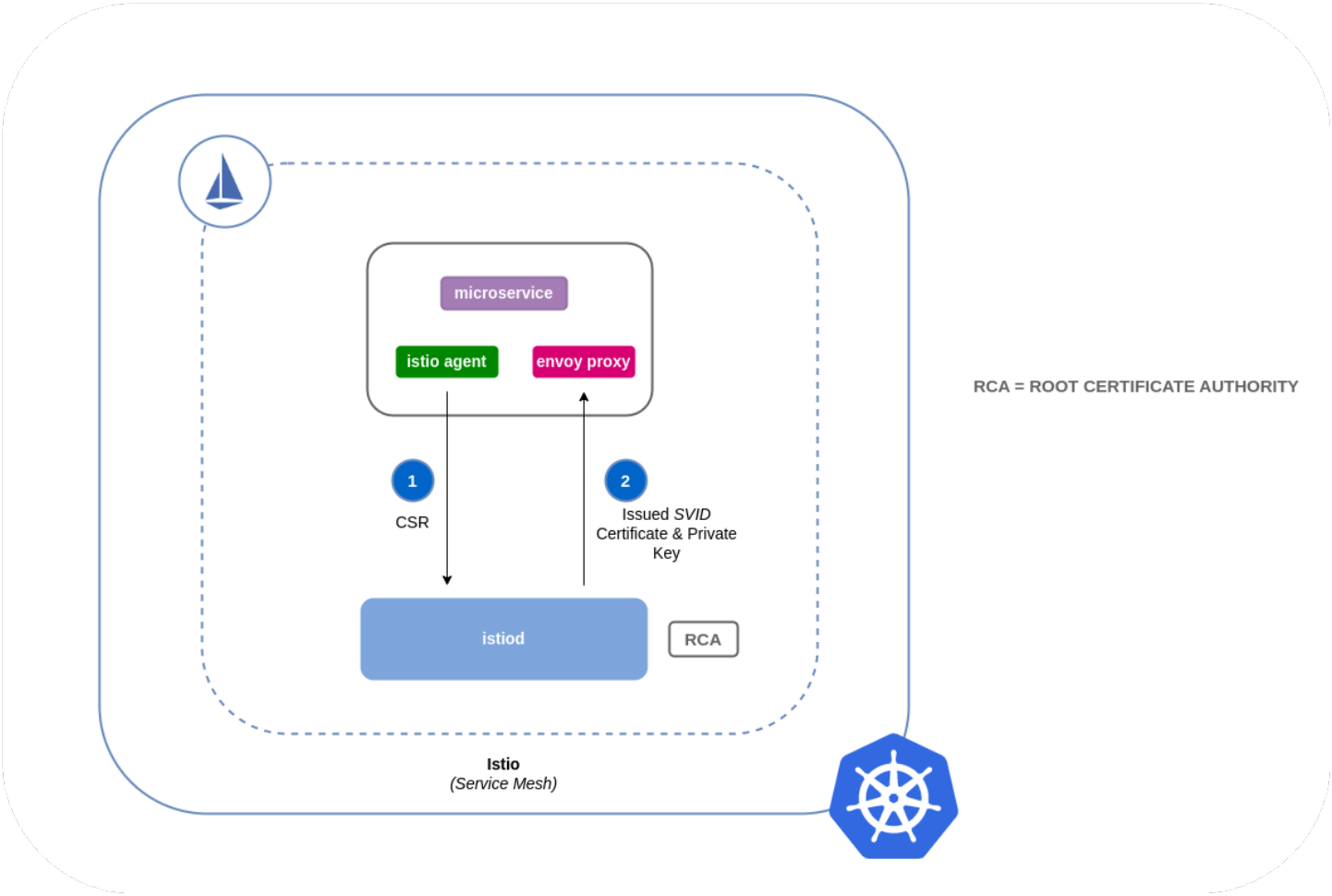
ACM Private CA als Stammzertifizierungsstelle für mTLs zwischen Workloads konfigurieren. Wenn Sie sich für diesen Ansatz entscheiden, sollten Sie die Verwendung des kurzlebigen CA-Modus in ACM Private CA in Betracht ziehen. Weitere Informationen finden Sie im vorherigen Abschnitt und in diesem Blogbeitrag.

So funktioniert das Signieren von Zertifikaten in Istio (Standard)

Workloads in Kubernetes werden mithilfe von Dienstkonten identifiziert. Wenn Sie kein Dienstkonto angeben, weist Kubernetes Ihrem Workload automatisch eines zu. Außerdem hängen Dienstkonten automatisch ein zugeordnetes Token ein. Dieses Token wird vom Dienstkonto für Workloads verwendet, um sich gegenüber der Kubernetes-API zu authentifizieren. Das Dienstkonto kann als Identität für Kubernetes ausreichend sein, aber Istio verfügt über ein eigenes Identitätsverwaltungssystem und eine eigene Zertifizierungsstelle. Wenn ein Workload mit seinem Envoy-Sidecar-Proxy gestartet wird, benötigt er eine von Istio zugewiesene Identität, damit er als vertrauenswürdig eingestuft wird und mit anderen Diensten im Mesh kommunizieren kann.

Um diese Identität von Istio zu erhalten, `istio-agent` sendet Istio eine Anfrage, die als Certificate Signing Request (oder CSR) bezeichnet wird, an die Istio-Steuerebene. Diese CSR enthält das Dienstkonto-Token, sodass die Identität des Workloads vor der Verarbeitung überprüft werden kann. Dieser Überprüfungsprozess wird von `istiod` der Registration Authority (oder RA) und der CA durchgeführt. Die RA dient als Gatekeeper, der sicherstellt, dass nur verifizierte CSR die CA erreicht. Sobald die CSR verifiziert ist, wird sie an die CA weitergeleitet, die dann ein Zertifikat ausstellt, das eine SPIFFE-Identität für das Dienstkonto enthält. Dieses Zertifikat wird als verifizierbares SPIFFE-Ausweisdokument (oder SVID) bezeichnet. Die SVID wird dem anfragenden Dienst zu Identifikationszwecken und zur Verschlüsselung des Transitverkehrs zwischen den kommunizierenden Diensten zugewiesen.

Standardablauf für Istio-Zertifikatsignieranfragen:



So funktioniert das Signieren von Zertifikaten in Istio mit ACM Private CA

Sie können ein Cert-Manager-Add-On namens Istio Certificate Signing Request Agent (istio-csr) verwenden, um Istio in ACM Private CA zu integrieren. Mit diesem Agenten können Istio-Workloads und Komponenten der Kontrollebene mit Cert Manager-Ausstellern, in diesem Fall ACM Private CA, gesichert werden. Der istio-csr-Agent stellt denselben Dienst bereit, den istiod in der Standardkonfiguration für die Validierung eingehender Daten bereitstellt. CSRs Allerdings werden die Anfragen nach der Überprüfung in Ressourcen umgewandelt, die der Zertifizierungsmanager unterstützt (d. h. Integrationen mit externen Zertifizierungsstellen).

Immer wenn eine CSR von einem Workload eingeht, wird sie an istio-csr weitergeleitet, das Zertifikate von ACM Private CA anfordert. Diese Kommunikation zwischen istio-csr und ACM Private CA wird durch das AWS Private CA Issuer-Plugin ermöglicht. Cert Manager verwendet dieses Plugin, um TLS-Zertifikate von ACM Private CA anzufordern. Das Aussteller-Plugin kommuniziert mit dem ACM Private CA-Dienst, um ein signiertes Zertifikat für den Workload anzufordern. Sobald das

Zertifikat signiert wurde, wird es an istio-csr zurückgegeben, das die signierte Anfrage liest und sie an den Workload zurücksendet, der den CSR initiiert hat.

Ablauf für Istio-Zertifikatsignieranfragen mit istio-csr

image:: istio-csr-with-acm -private-ca.png [Ablauf für Istio-Zertifikatsignieranfragen mit istio-csr]

Anweisungen zur Einrichtung von Istio mit privater CA

1. Folgen Sie zunächst den gleichen Einrichtungsanweisungen in diesem Abschnitt, um Folgendes abzuschließen:
2. Erstellen einer privaten CA
3. Installieren Sie cert-manager
4. Installieren Sie das Issuer-Plugin
5. Legen Sie Berechtigungen fest und erstellen Sie einen Emittenten. Der Aussteller repräsentiert die Zertifizierungsstelle und wird verwendet, um Workload-Zertifikate zu signieren `istiod` und miteinander zu verknüpfen. Es wird mit ACM Private CA kommunizieren.
6. Erstellen Sie einen `istio-system` Namespace. Hier werden die `istiod certificate` und andere Istio-Ressourcen bereitgestellt.
7. Installieren Sie Istio CSR, konfiguriert mit dem AWS Private CA Issuer Plugin. Sie können die Zertifikatssignierungsanforderungen für Workloads aufbewahren, um zu überprüfen, ob sie genehmigt und signiert wurden (). `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. Installieren Sie Istio mit benutzerdefinierten Konfigurationen, um es `cert-manager istio-csr` als Zertifikatsanbieter für das Mesh zu ersetzen`istiod`. Dieser Vorgang kann mit dem Istio-Operator ausgeführt werden.

```
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. Stellen Sie die oben angegebene benutzerdefinierte Ressource bereit, die Sie erstellt haben.

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

10. Jetzt können Sie einen Workload für das Mesh in Ihrem EKS-Cluster bereitstellen und mTLS erzwingen.

Anfragen zum Signieren von Istio-Zertifikaten

image: istio-csr-requests .png [Signaturanfragen für Istio-Zertifikate]

Tools und Ressourcen

- Amazon EKS Security Immersion Workshop — Netzwerksicherheit
- So implementieren Sie den Cert-Manager und das ACM Private CA-Plugin, um TLS in EKS zu aktivieren.
- Einrichtung der end-to-end TLS-Verschlüsselung auf Amazon EKS mit dem neuen AWS Load Balancer Controller und ACM Private CA.
- Das Private CA Kubernetes Cert-Manager-Plugin ist aktiviert. GitHub
- Privates CA Kubernetes Cert-Manager-Plug-in-Benutzerhandbuch.
- So verwenden Sie den kurzlebigen Zertifikatsmodus von AWS Private Certificate Authority

- Egress-Operator Ein Operator und ein DNS-Plugin zur Steuerung des ausgehenden Datenverkehrs aus Ihrem Cluster ohne Protokollinspektion
- NeuVector Die Open-Source-Zero-Trust-Container-Sicherheitsplattform von SUSE bietet Policy-Netzwerkregeln, Data Loss Prevention (DLP), Web Application Firewall (WAF) und Signaturen für Netzwerkbedrohungen.

Datenverschlüsselung und Verwaltung von Geheimnissen

Verschlüsselung im Ruhezustand

Es gibt drei verschiedene AWS-native Speicheroptionen, die Sie mit Kubernetes verwenden können: EBS, EFS und for Lustre. FSx Alle drei bieten Verschlüsselung im Ruhezustand mithilfe eines vom Service verwalteten Schlüssels oder eines Customer-Hauptschlüssels (CMK). Für EBS können Sie den In-Tree-Speichertreiber oder den EBS-CSI-Treiber verwenden. Beide enthalten Parameter für die Verschlüsselung von Volumes und die Bereitstellung eines CMK. Für EFS können Sie den EFS-CSI-Treiber verwenden. Im Gegensatz zu EBS unterstützt der EFS-CSI-Treiber jedoch kein dynamisches Provisioning. Wenn Sie EFS mit EKS verwenden möchten, müssen Sie die Verschlüsselung im Ruhezustand für das Dateisystem bereitstellen und konfigurieren, bevor Sie eine PV erstellen. Weitere Informationen zur EFS-Dateiverschlüsselung finden Sie unter Daten im Ruhezustand verschlüsseln. EFS und FSx for Lustre bieten nicht nur Verschlüsselung im Ruhezustand, sondern bieten auch eine Option zur Verschlüsselung von Daten während der Übertragung. FSx for Lustre tut dies standardmäßig. Für EFS können Sie Transportverschlüsselung hinzufügen, indem Sie den `tls` Parameter zu `mountOptions` in Ihrem PV hinzufügen, wie in diesem Beispiel:

```
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>
```

Der FSx CSI-Treiber unterstützt die dynamische Bereitstellung von Lustre-Dateisystemen. Es verschlüsselt Daten standardmäßig mit einem vom Dienst verwalteten Schlüssel, es besteht jedoch die Möglichkeit, Ihr eigenes CMK bereitzustellen, wie in diesem Beispiel:

```
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**
>
> Seit dem 28. Mai 2020 werden alle Daten, die auf das ephemere Volume in EKS Fargate-Pods geschrieben werden, standardmäßig mit einem branchenüblichen kryptografischen AES-256-Algorithmus verschlüsselt. Es sind keine Änderungen an Ihrer Anwendung erforderlich, da die Verschlüsselung und Entschlüsselung vom Service nahtlos übernommen werden.

Verschlüsseln Sie Daten im Ruhezustand

Das Verschlüsseln von Daten im Ruhezustand wird als bewährte Methode angesehen. Wenn Sie sich nicht sicher sind, ob eine Verschlüsselung erforderlich ist, verschlüsseln Sie Ihre Daten.

Rotieren Sie Ihre regelmäßig CMKs

Konfigurieren Sie KMS so, dass Ihr automatisch rotiert wird CMKs. Dadurch werden Ihre Schlüssel einmal im Jahr rotiert, während alte Schlüssel auf unbestimmte Zeit gespeichert werden, sodass Ihre Daten weiterhin entschlüsselt werden können. Weitere Informationen finden Sie unter Rotieren von Kundenhauptschlüsseln

Verwenden Sie EFS-Zugriffspunkte, um den Zugriff auf gemeinsam genutzte Datensätze zu vereinfachen

Wenn Sie gemeinsam genutzte Datensätze mit unterschiedlichen POSIX-Dateiberechtigungen haben oder den Zugriff auf einen Teil des gemeinsam genutzten Dateisystems einschränken möchten, indem Sie verschiedene Einhängepunkte erstellen, sollten Sie die Verwendung von EFS-Zugriffspunkten in Betracht ziehen. Weitere Informationen zum Arbeiten mit Access Points finden Sie unter https://docs.aws.amazon.com/efs/ latest/ug/efs -access-points.html. Wenn Sie heute einen Access Point (AP) verwenden möchten, müssen Sie den AP im `volumeHandle` PV-Parameter referenzieren.

> **Important**
>
> Ab dem 23. März 2021 unterstützt der EFS-CSI-Treiber die dynamische Bereitstellung von EFS Access Points. Access Points sind anwendungsspezifische Einstiegspunkte in ein EFS-Dateisystem, die es einfacher machen, ein Dateisystem zwischen mehreren Pods gemeinsam zu nutzen. Jedes EFS-Dateisystem kann bis zu 120 haben PVs. Weitere Informationen finden Sie unter Einführung in Amazon EFS CSI Dynamic Provisioning.

Verwaltung von Secrets

Kubernetes-Secrets werden verwendet, um vertrauliche Informationen wie Benutzerzertifikate, Passwörter oder API-Schlüssel zu speichern. Sie werden in etcd als Base64-kodierte Zeichenketten gespeichert. Auf EKS sind die EBS-Volumes für etcd-Knoten mit EBS-Verschlüsselung verschlüsselt. Ein Pod kann geheime Objekte von Kubernetes abrufen, indem er auf das Geheimnis in der verweist. `podSpec` Diese Geheimnisse können entweder einer Umgebungsvariablen zugeordnet oder als Volume bereitgestellt werden. Weitere Informationen zum Erstellen von Geheimnissen finden Sie unter https://kubernetes. io/docs/concepts/configuration/secret/.

> **Warning**
>
> Auf Geheimnisse in einem bestimmten Namespace kann von allen Pods im Namespace des Geheimnisses verwiesen werden.

> **Warning**
>
> Der Node Authorizer ermöglicht es dem Kubelet, alle auf dem Knoten montierten Geheimnisse zu lesen.

Verwenden Sie AWS KMS für die Envelope-Verschlüsselung von Kubernetes-Geheimnissen

Auf diese Weise können Sie Ihre Geheimnisse mit einem eindeutigen Datenverschlüsselungsschlüssel (DEK) verschlüsseln. Das DEK wird dann mit einem Key Encryption Key (KEK) von AWS KMS verschlüsselt, der nach einem wiederkehrenden Zeitplan automatisch rotiert werden kann. Mit dem KMS-Plugin für Kubernetes werden alle Kubernetes-Geheimnisse in etcd im Chiffretext statt im Klartext gespeichert und können nur vom Kubernetes-API-Server entschlüsselt werden. Weitere Informationen finden Sie unter Verwenden der Unterstützung von EKS-Verschlüsselungsanbietern für Verteidigungszwecke

Prüfen Sie die Verwendung von Kubernetes Secrets

Aktivieren Sie in EKS die Auditprotokollierung und erstellen Sie einen CloudWatch Metrikfilter und einen Alarm, um Sie zu benachrichtigen, wenn ein Geheimnis verwendet wird (optional). Im Folgenden finden Sie ein Beispiel für einen Metrikfilter für das Kubernetes-Audit-Log. `{($.verb="get") && ($.objectRef.resource="secret")}` Sie können auch die folgenden Abfragen mit CloudWatch Log Insights verwenden:

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

Die obige Abfrage zeigt an, wie oft innerhalb eines bestimmten Zeitraums auf ein Geheimnis zugegriffen wurde.

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| filter verb="get" and objectRef.resource="secrets"
```

```
| display objectRef.namespace, objectRef.name, user.username, responseStatus.code
```

Diese Abfrage zeigt das Geheimnis zusammen mit dem Namespace und dem Benutzernamen des Benutzers, der versucht hat, auf das Geheimnis zuzugreifen, sowie den Antwortcode an.

Wechseln Sie Ihre Geheimnisse regelmäßig

Kubernetes rotiert Geheimnisse nicht automatisch. Wenn Sie geheime Daten rotieren müssen, sollten Sie die Verwendung eines externen geheimen Speichers in Betracht ziehen, z. B. Vault oder AWS Secrets Manager.

Verwenden Sie separate Namespaces, um Geheimnisse aus verschiedenen Anwendungen zu isolieren

Wenn Sie über Geheimnisse verfügen, die nicht von Anwendungen in einem Namespace gemeinsam genutzt werden können, erstellen Sie einen separaten Namespace für diese Anwendungen.

Verwenden Sie Volume-Mounts anstelle von Umgebungsvariablen

Die Werte von Umgebungsvariablen können unbeabsichtigt in Protokollen erscheinen. Als Volumes gemountete Geheimnisse werden als TMPFS-Volumes (ein RAM-gestütztes Dateisystem) instanziiert, die automatisch aus dem Knoten entfernt werden, wenn der Pod gelöscht wird.

Verwenden Sie einen externen Geheimdienstanbieter

Es gibt mehrere praktikable Alternativen zur Verwendung von Kubernetes-Secrets, darunter AWS Secrets Manager und Hashicorp's Vault. Diese Services bieten Funktionen wie detaillierte Zugriffskontrollen, starke Verschlüsselung und automatische Rotation von Geheimnissen, die mit Kubernetes Secrets nicht verfügbar sind. Bitnamis Sealed Secrets ist ein weiterer Ansatz, der asymmetrische Verschlüsselung verwendet, um „versiegelte Geheimnisse“ zu erstellen. Ein öffentlicher Schlüssel wird verwendet, um das Geheimnis zu verschlüsseln, während der private Schlüssel, der zum Entschlüsseln des Geheimnisses verwendet wird, innerhalb des Clusters aufbewahrt wird, sodass Sie versiegelte Geheimnisse sicher in Quellcodeverwaltungssystemen wie Git speichern können. Weitere Informationen finden Sie unter Verwaltung der Bereitstellung von Geheimnissen in Kubernetes mithilfe von Sealed Secrets.

In dem Maße, wie die Verwendung externer Secret-Stores zugenommen hat, ist auch die Notwendigkeit gestiegen, sie in Kubernetes zu integrieren. Der Secret Store CSI Driver ist ein Community-Projekt, das das CSI-Treibermodell verwendet, um Geheimnisse aus externen geheimen Speichern abzurufen. Derzeit unterstützt der Treiber AWS Secrets Manager, Azure, Vault und

GCP. Der AWS-Anbieter unterstützt sowohl AWS Secrets Manager als auch AWS Parameter Store. Es kann auch so konfiguriert werden, dass Secrets nach Ablauf rotiert werden und AWS Secrets Manager Manager-Geheimnisse mit Kubernetes Secrets synchronisiert werden. Die Synchronisation von Geheimnissen kann nützlich sein, wenn Sie auf ein Geheimnis als Umgebungsvariable verweisen müssen, anstatt es aus einem Volume zu lesen.

> **Note**
>
> Wenn der CSI-Treiber für den geheimen Speicher ein Geheimnis abrufen muss, übernimmt er die IRSA-Rolle, die dem Pod zugewiesen ist, der auf ein Geheimnis verweist. Den Code für diesen Vorgang finden Sie hier.

Weitere Informationen zum AWS Secrets & Configuration Provider (ASCP) finden Sie in den folgenden Ressourcen:

- So verwenden Sie den AWS Secrets Configuration Provider mit dem Kubernetes Secret Store CSI-Treiber
- Integration von Secrets Manager Manager-Geheimnissen mit dem Kubernetes Secrets Store CSI-Treiber

external-secrets ist eine weitere Möglichkeit, einen externen geheimen Speicher mit Kubernetes zu verwenden. Wie der CSI-Treiber funktioniert external-Secrets gegen eine Vielzahl verschiedener Backends, einschließlich AWS Secrets Manager. Der Unterschied besteht darin, dass external-secrets keine Geheimnisse aus dem externen Geheimspeicher abruft, sondern Geheimnisse aus diesen Backends als Secrets nach Kubernetes kopiert. Auf diese Weise können Sie Geheimnisse mithilfe Ihres bevorzugten geheimen Speichers verwalten und auf Kubernetes-native Weise mit Geheimnissen interagieren.

Tools und Ressourcen

- Amazon EKS Security Immersion Workshop — Datenverschlüsselung und Geheimnismanagement

Laufzeit-Sicherheit

Runtime Security bietet aktiven Schutz für Ihre Container, während sie laufen. Die Idee besteht darin, zu erkennen und zu and/or verhindern, dass bösartige Aktivitäten innerhalb des Containers

auftreten. Dies kann mit einer Reihe von Mechanismen im Linux-Kernel oder mit Kernelerweiterungen erreicht werden, die in Kubernetes integriert sind, wie z. B. Linux-Funktionen, Secure Computing (Seccomp), oder. AppArmor SELinux Es gibt auch Optionen wie Amazon GuardDuty und Tools von Drittanbietern, die bei der Festlegung von Baselines und der Erkennung anomaler Aktivitäten mit weniger manueller Konfiguration der Linux-Kernel-Mechanismen helfen können.

> **Important**
>
> Kubernetes bietet derzeit keine systemeigenen Mechanismen zum Laden von Seccomp- oder Profilen auf Knoten. AppArmor SELinux Sie müssen entweder manuell geladen oder auf Nodes installiert werden, wenn sie gebootet werden. Dies muss geschehen, bevor Sie in Ihren Pods auf sie verweisen, da der Scheduler nicht weiß, welche Knoten Profile haben. Im Folgenden erfahren Sie, wie Tools wie Security Profiles Operator dazu beitragen können, die Bereitstellung von Profilen auf Knoten zu automatisieren.

Sicherheitskontexte und integrierte Kubernetes-Steuerelemente

Viele Sicherheitsmechanismen für die Linux-Laufzeit sind eng in Kubernetes integriert und können über Kubernetes-Sicherheitskontexte konfiguriert werden. Eine solche Option ist das `privileged` Flag, das `false` standardmäßig aktiviert ist und, falls aktiviert, im Wesentlichen Root auf dem Host entspricht. Es ist fast immer unangemessen, den privilegierten Modus in Produktions-Workloads zu aktivieren, aber es gibt viel mehr Steuerelemente, die Containern je nach Bedarf detailliertere Rechte gewähren können.

Linux-Funktionen

Mithilfe von Linux-Funktionen können Sie einem Pod oder Container bestimmte Funktionen gewähren, ohne alle Fähigkeiten des Root-Benutzers zur Verfügung zu stellen. Beispiele hierfür sind `CAP_NET_ADMIN` die Konfiguration von Netzwerkschnittstellen oder Firewalls oder `CAP_SYS_TIME` die Manipulation der Systemuhr.

Seccomp

Mit Secure Computing (Seccomp) können Sie verhindern, dass eine containerisierte Anwendung bestimmte Syscalls an den Kernel des zugrunde liegenden Host-Betriebssystems sendet. Das Linux-Betriebssystem hat zwar einige hundert Systemaufrufe, aber der Löwenanteil davon ist für den Betrieb von Containern nicht erforderlich. Indem Sie einschränken, welche Syscalls von einem

Container ausgeführt werden können, können Sie die Angriffsfläche Ihrer Anwendung effektiv verringern.

Seccomp funktioniert, indem es Systemaufrufe abfängt und nur diejenigen weiterleitet, die auf der Zulassungsliste stehen. Docker hat ein Standard-Seccomp-Profil, das für die meisten Allzweck-Workloads geeignet ist, und andere Container-Laufzeiten wie containerd bieten vergleichbare Standardwerte. Sie können Ihren Container oder Pod so konfigurieren, dass er das Standard-Seccomp-Profil der Container-Laufzeit verwendet, indem Sie dem Abschnitt der Pod-Spezifikation Folgendes hinzufügen: `securityContext`

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

Ab Version 1.22 (in Alpha, stabil ab 1.27) `RuntimeDefault` können die obigen Angaben für alle Pods auf einem Node verwendet werden, die ein einziges Kubelet-Flag verwenden. `--seccomp-default` Dann wird das unter angegebene Profil nur für `securityContext` andere Profile benötigt.

Es ist auch möglich, eigene Profile für Dinge zu erstellen, für die zusätzliche Rechte erforderlich sind. Dies kann manuell sehr mühsam sein, aber es gibt Tools wie Inspektor Gadget (auch im Abschnitt Netzwerksicherheit für die Generierung von Netzwerkrichtlinien empfohlen) und Security Profiles Operator, die die Verwendung von Tools wie eBPF oder Protokollen unterstützen, um grundlegende Rechteanforderungen als Seccomp-Profile aufzuzeichnen. Der Security Profiles Operator ermöglicht außerdem die Automatisierung der Bereitstellung aufgezeichneter Profile auf Knoten zur Verwendung durch Pods und Container.

AppArmor und SELinux

AppArmor und SELinux werden als obligatorische Zugangskontroll- oder MAC-Systeme bezeichnet. Sie haben ein ähnliches Konzept wie Seccomp, verfügen jedoch über andere Fähigkeiten APIs und ermöglichen die Zugriffskontrolle für z. B. bestimmte Dateisystempfade oder Netzwerkports. Die Unterstützung dieser Tools hängt von der Linux-Distribution ab, wobei Debian/Ubuntu Support AppArmor und RHEL/CentOS/Bottlerocket/Amazon Linux 2023 unterstützt SELinux werden. Weitere Informationen zu finden Sie auch im Abschnitt zur Infrastruktursicherheit SELinux.

Beide AppArmor SELinux sind in Kubernetes integriert, aber seit Kubernetes 1.28 müssen AppArmor Profile über Anmerkungen angegeben werden, während SELinux Beschriftungen direkt über das SELinuxOptionsfeld im Sicherheitskontext festgelegt werden können.

Wie bei Seccomp-Profilen kann der oben erwähnte Security Profiles Operator bei der Bereitstellung von Profilen auf Knoten im Cluster helfen. (In future zielt das Projekt auch darauf ab, Profile für AppArmor und SELinux wie für seccomp zu generieren.)

Empfehlungen

Verwenden Sie Amazon GuardDuty zur Laufzeitüberwachung und Erkennung von Bedrohungen für Ihre EKS-Umgebungen

Wenn Sie derzeit nicht über eine Lösung für die kontinuierliche Überwachung der EKS-Laufzeiten und die Analyse von EKS-Auditprotokollen sowie für die Suche nach Malware und anderen verdächtigen Aktivitäten verfügen, empfiehlt Amazon GuardDuty Kunden, die ihre AWS-Umgebungen mit einem Klick auf einfache, schnelle, sichere, skalierbare und kostengünstige Weise schützen möchten, die Nutzung von Amazon. Amazon GuardDuty ist ein Sicherheitsüberwachungsservice, der grundlegende Datenquellen wie CloudTrail AWS-Managementereignisse, CloudTrail AWS-Ereignisprotokolle, VPC-Flow-Logs (von Amazon EC2 EC2-Instances), Kubernetes-Audit-Logs und DNS-Logs analysiert und verarbeitet. Er beinhaltet auch die EKS-Laufzeitüberwachung. Es verwendet ständig aktualisierte Feeds mit Bedrohungsinformationen, wie z. B. Listen bösartiger IP-Adressen und Domänen, und maschinelles Lernen, um unerwartete, potenziell nicht autorisierte und bösartige Aktivitäten in Ihrer AWS-Umgebung zu identifizieren. Dies kann Probleme wie die Eskalation von Rechten, die Verwendung offengelegter Anmeldeinformationen oder die Kommunikation mit bösartigen IP-Adressen und Domänen, das Vorhandensein von Malware auf Ihren Amazon EC2 EC2-Instances und EKS-Container-Workloads oder die Entdeckung verdächtiger API-Aktivitäten beinhalten. GuardDuty informiert Sie über den Status Ihrer AWS-Umgebung, indem es Sicherheitsergebnisse generiert, die Sie in der GuardDuty Konsole oder über Amazon einsehen können EventBridge. GuardDuty bietet Ihnen auch Unterstützung beim Exportieren Ihrer Ergebnisse in einen Amazon Simple Storage Service (S3) -Bucket und bei der Integration mit anderen Services wie AWS Security Hub und Detective.

Sehen Sie sich diesen AWS Online Tech Talk „Verbesserte Bedrohungserkennung für Amazon EKS with Amazon GuardDuty — AWS Online Tech Talks“ an, um zu erfahren, wie Sie diese zusätzlichen EKS-Sicherheitsfunktionen step-by-step innerhalb von Minuten aktivieren können.

Optional: Verwenden Sie eine Drittanbieterlösung für die Laufzeitüberwachung

Das Erstellen und Verwalten von Seccomp- und Apparmor-Profilen kann schwierig sein, wenn Sie mit der Linux-Sicherheit nicht vertraut sind. Wenn Sie nicht die Zeit haben, sich damit vertraut zu machen, sollten Sie die Verwendung einer kommerziellen Lösung eines Drittanbieters in Betracht

ziehen. Viele von ihnen haben statische Profile wie Apparmor und Seccomp hinter sich gelassen und begonnen, maschinelles Lernen zu verwenden, um verdächtige Aktivitäten zu blockieren oder zu warnen. Eine Handvoll dieser Lösungen finden Sie weiter unten im Bereich Tools. Zusätzliche Optionen finden Sie auf dem AWS Marketplace for Containers.

Ziehen Sie die add/dropping Linux-Funktionen in Betracht, bevor Sie Seccomp-Richtlinien schreiben

Fähigkeiten beinhalten verschiedene Prüfungen von Kernelfunktionen, die über Systemaufrufe erreichbar sind. Schlägt die Prüfung fehl, gibt der Systemaufruf normalerweise einen Fehler zurück. Die Überprüfung kann entweder direkt am Anfang eines bestimmten Systemaufrufs oder tiefer im Kernel in Bereichen durchgeführt werden, die möglicherweise über mehrere verschiedene Systemaufrufe erreichbar sind (z. B. das Schreiben in eine bestimmte privilegierte Datei). Seccomp hingegen ist ein Systemaufruf-Filter, der auf alle Syscalls angewendet wird, bevor sie ausgeführt werden. Ein Prozess kann einen Filter einrichten, der es ihm ermöglicht, ihm sein Recht zur Ausführung bestimmter Systemaufrufe oder bestimmter Argumente für bestimmte Systemaufrufe zu entziehen.

Bevor Sie seccomp verwenden, sollten Sie sich überlegen, ob Sie mit den adding/removing Linux-Funktionen die Kontrolle haben, die Sie benötigen. Weitere Informationen finden Sie unter Funktionen für Container einrichten.

Finden Sie heraus, ob Sie Ihre Ziele mithilfe der Pod-Sicherheitsrichtlinien erreichen können (PSPs)

Pod-Sicherheitsrichtlinien bieten viele verschiedene Möglichkeiten, Ihren Sicherheitsstatus zu verbessern, ohne dass dies zu einer übermäßigen Komplexität führt. Informieren Sie sich über die in verfügbaren Optionen, PSPs bevor Sie sich an die Erstellung von Seccomp- und Apparmor-Profilen wagen.

> **Warning**
>
> Ab Kubernetes 1.25 PSPs wurden sie entfernt und durch den Pod Security Admission Controller ersetzt. Zu den vorhandenen Alternativen von Drittanbietern gehören OPA/Gatekeeper Kyverno. Eine Sammlung von Gatekeeper-Einschränkungen und Einschränkungsvorlagen für die Implementierung von Richtlinien, die häufig in zu finden sind, PSPs kann unter dem Gatekeeper-Bibliotheksarchiv abgerufen werden. GitHub Und viele

Ersatzprodukte für PSPs finden Sie in der Kyverno-Richtlinienbibliothek, einschließlich der vollständigen Sammlung der Pod-Sicherheitsstandards.

Tools und Ressourcen

- 7 Dinge, die Sie wissen sollten, bevor Sie beginnen
- AppArmorLader
- Knoten mit Profilen einrichten
- Security Profiles Operator ist eine Kubernetes-Erweiterung, die Benutzern die Verwendung SELinux in Seccomp und AppArmor in Kubernetes-Clustern erleichtern soll. Es bietet Funktionen sowohl zum Generieren von Profilen aus laufenden Workloads als auch zum Laden von Profilen auf Kubernetes-Knoten zur Verwendung in Pods.
- Inspektor Gadget ermöglicht die Inspektion, Nachverfolgung und Profilierung vieler Aspekte des Laufzeitverhaltens auf Kubernetes, einschließlich der Unterstützung bei der Generierung von Seccomp-Profilen.
- Aqua
- Qualys
- Stackrox
- Sysdig Sicher
- Prisma
- NeuVector Die Open-Source-Zero-Trust-Container-Sicherheitsplattform von SUSE bietet Prozessprofilregeln und Dateizugriffsregeln.

Schutz der Infrastruktur (Hosts)

So wichtig es ist, Ihre Container-Images zu sichern, ist es ebenso wichtig, die Infrastruktur zu schützen, auf der sie ausgeführt werden. In diesem Abschnitt werden verschiedene Möglichkeiten zur Minderung von Risiken untersucht, die von Angriffen ausgehen, die direkt gegen den Host gestartet werden. Diese Richtlinien sollten in Verbindung mit den Richtlinien verwendet werden, die im Abschnitt Runtime Security beschrieben werden.

Empfehlungen

Verwenden Sie ein Betriebssystem, das für die Ausführung von Containern optimiert ist

Erwägen Sie die Verwendung von Flatcar Linux, Project Atomic, RancherOS und Bottlerocket, einem speziellen Betriebssystem von AWS, das für den Betrieb von Linux-Containern entwickelt wurde. Es umfasst eine reduzierte Angriffsfläche, ein Festplatten-Image, das beim Systemstart verifiziert wird, und erzwungene Zugriffsgrenzen mithilfe von. SELinux

Verwenden Sie alternativ das für EKS optimierte AMI für Ihre Kubernetes-Worker-Knoten. Das für EKS optimierte AMI wird regelmäßig veröffentlicht und enthält eine minimale Anzahl von Betriebssystempaketen und Binärdateien, die für die Ausführung Ihrer containerisierten Workloads erforderlich sind.

In der Amazon EKS AMI RHEL Build Specification finden Sie ein Beispielkonfigurationsskript, mit dem Sie mithilfe von Hashicorp Packer ein benutzerdefiniertes Amazon EKS-AMI erstellen können, das auf Red Hat Enterprise Linux ausgeführt wird. Dieses Skript kann weiter genutzt werden, um benutzerdefinierte STIG-konforme EKS-Dateien zu erstellen. AMIs

Halten Sie das Betriebssystem Ihres Worker-Nodes auf dem neuesten Stand

Unabhängig davon, ob Sie ein containeroptimiertes Host-Betriebssystem wie Bottlerocket oder ein größeres, aber dennoch minimalistisches Amazon Machine Image wie das EKS-optimierte verwenden, empfiehlt es sich AMIs, diese Host-Betriebssystem-Images mit den neuesten Sicherheitspatches auf dem neuesten Stand zu halten.

Für die EKS-optimierte AMIs Version sollten Sie regelmäßig den CHANGELOG - und/oder Versionshinweise-Channel überprüfen und den Rollout aktualisierter Worker-Node-Images in Ihrem Cluster automatisieren.

Behandeln Sie Ihre Infrastruktur als unveränderlich und automatisieren Sie den Austausch Ihrer Worker-Knoten

Anstatt Upgrades vor Ort durchzuführen, sollten Sie Ihre Mitarbeiter austauschen, wenn ein neuer Patch oder ein neues Update verfügbar ist. Dies kann auf verschiedene Arten angegangen werden. Sie können entweder Instances zu einer vorhandenen Autoscaling-Gruppe hinzufügen, indem Sie das neueste AMI verwenden, während Sie Knoten sequenziell sperren und entleeren, bis alle Knoten in der Gruppe durch das neueste AMI ersetzt wurden. Alternativ können Sie Instances zu einer neuen Knotengruppe hinzufügen, während Sie nacheinander Knoten aus der alten Knotengruppe sperren und entfernen, bis alle Knoten ersetzt wurden. Die von EKS verwalteten Knotengruppen verwenden

den ersten Ansatz und zeigen in der Konsole eine Meldung an, dass Sie Ihre Worker aktualisieren müssen, sobald ein neues AMI verfügbar ist. `eksctl`verfügt auch über einen Mechanismus zum Erstellen von Knotengruppen mit dem neuesten AMI und zum ordnungsgemäßen Sperren und Entleeren von Pods aus Knotengruppen, bevor die Instances beendet werden. Wenn Sie sich für eine andere Methode zum Austausch Ihrer Worker-Knoten entscheiden, wird dringend empfohlen, den Prozess zu automatisieren, um menschliche Aufsicht zu minimieren, da Sie wahrscheinlich regelmäßig Worker austauschen müssen, sobald neue veröffentlicht updates/patches werden und wenn die Steuerungsebene aktualisiert wird.

Mit EKS Fargate aktualisiert AWS die zugrunde liegende Infrastruktur automatisch, sobald Updates verfügbar sind. Oft kann dies problemlos erfolgen, aber es kann vorkommen, dass ein Update dazu führt, dass Ihr Pod verschoben wird. Daher empfehlen wir, Bereitstellungen mit mehreren Replikaten zu erstellen, wenn Sie Ihre Anwendung als Fargate-Pod ausführen.

Führen Sie kube-bench regelmäßig aus, um die Einhaltung der CIS-Benchmarks für Kubernetes zu überprüfen

kube-bench ist ein Open-Source-Projekt von Aqua, das Ihren Cluster anhand der CIS-Benchmarks für Kubernetes bewertet. Der Benchmark beschreibt die besten Methoden zur Sicherung nicht verwalteter Kubernetes-Cluster. Der CIS Kubernetes Benchmark umfasst die Steuerungsebene und die Datenebene. Da Amazon EKS eine vollständig verwaltete Kontrollebene bietet, sind nicht alle Empfehlungen des CIS Kubernetes Benchmark zutreffend. Um sicherzustellen, dass dieser Umfang die Implementierung von Amazon EKS widerspiegelt, hat AWS den CIS Amazon EKS Benchmark erstellt. Der EKS-Benchmark basiert auf CIS Kubernetes Benchmark und enthält zusätzliche Beiträge aus der Community mit spezifischen Überlegungen zur Konfiguration von EKS-Clustern.

Wenn Sie Kube-Bench auf einem EKS-Cluster ausführen, folgen Sie diesen Anweisungen von Aqua Security. Weitere Informationen finden Sie unter Einführung in den CIS Amazon EKS Benchmark.

Minimiert den Zugriff auf Worker-Knoten

Anstatt den SSH-Zugriff zu aktivieren, verwenden Sie SSM Session Manager, wenn Sie remote auf einen Host zugreifen müssen. Im Gegensatz zu SSH-Schlüsseln, die verloren gehen, kopiert oder gemeinsam genutzt werden können, können Sie mit Session Manager den Zugriff auf EC2-Instances mithilfe von IAM steuern. Darüber hinaus bietet er einen Audit-Trail und ein Protokoll der Befehle, die auf der Instance ausgeführt wurden.

Seit dem 19. August 2020 unterstützen Managed Node Groups benutzerdefinierte AMIs und EC2-Startvorlagen. Auf diese Weise können Sie den SSM-Agenten in das AMI einbetten oder

ihn installieren, während der Worker-Node gebootet wird. Wenn Sie das optimierte AMI oder die Startvorlage der ASG nicht ändern möchten, können Sie den SSM-Agenten DaemonSet wie in diesem Beispiel mit einem installieren.

Minimale IAM-Richtlinie für SSM-basierten SSH-Zugriff

Die von `AmazonSSMManagedInstanceCore` AWS verwaltete Richtlinie enthält eine Reihe von Berechtigungen, die für SSM Session Manager/SSM nicht erforderlich sind, RunCommand wenn Sie nur den SSH-Zugriff vermeiden möchten. Besonders besorgniserregend sind die * Berechtigungen`ssm:GetParameter(s)`, für die die Rolle den Zugriff auf alle Parameter im Parameter Store ermöglichen würde (auch SecureStrings wenn der von AWS verwaltete KMS-Schlüssel konfiguriert ist).

Die folgende IAM-Richtlinie enthält die Mindestberechtigungen, um den Knotenzugriff über SSM Systems Manager zu ermöglichen.

```
{
  "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": "*"
    }
  ]
}
```

Wenn diese Richtlinie eingerichtet und das Session Manager-Plug-In installiert ist, können Sie dann Folgendes ausführen

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

um auf den Knoten zuzugreifen.

> **Note**
>
> Möglicherweise möchten Sie auch Berechtigungen hinzufügen, um die Session Manager-Protokollierung zu aktivieren.

Stellen Sie Worker in privaten Subnetzen bereit

Durch den Einsatz von Mitarbeitern in privaten Subnetzen minimieren Sie deren Gefährdung durch das Internet, wo Angriffe häufig ihren Ursprung haben. Ab dem 22. April 2020 wird die Zuweisung öffentlicher IP-Adressen zu Knoten in verwalteten Knotengruppen durch das Subnetz gesteuert, in dem sie bereitgestellt werden. Zuvor wurde Knoten in einer verwalteten Knotengruppe automatisch eine öffentliche IP zugewiesen. Wenn Sie sich dafür entscheiden, Ihre Worker-Knoten in öffentlichen Subnetzen bereitzustellen, implementieren Sie restriktive AWS-Sicherheitsgruppenregeln, um deren Gefährdung zu begrenzen.

Führen Sie Amazon Inspector aus, um Hosts auf Risiken, Sicherheitslücken und Abweichungen von bewährten Methoden zu untersuchen

Sie können Amazon Inspector verwenden, um nach unbeabsichtigten Netzwerkzugriffen auf Ihre Knoten und nach Sicherheitslücken in den zugrunde liegenden Amazon EC2 EC2-Instances zu suchen.

Amazon Inspector kann CVE-Daten (Common Vulnerabilities and Exposures) für Ihre Amazon EC2-Instances nur bereitstellen, wenn der Amazon EC2 Systems Manager (SSM) -Agent installiert und aktiviert ist. Dieser Agent ist auf mehreren Amazon Machine Images (AMIs) vorinstalliert, einschließlich EKS-optimiertem Amazon Linux AMIs. Unabhängig vom Status des SSM-Agenten

werden alle Ihre Amazon EC2 EC2-Instances auf Probleme mit der Netzwerkerreichbarkeit überprüft. Weitere Informationen zur Konfiguration von Scans für Amazon EC2 finden Sie unter Scannen von Amazon EC2 EC2-Instances.

> **Important**
>
> Inspector kann nicht auf der Infrastruktur ausgeführt werden, die für den Betrieb von Fargate-Pods verwendet wird.

Alternativen

Lauf SELinux

> **Note**
>
> Verfügbar auf Red Hat Enterprise Linux (RHEL), CentOS, Bottlerocket und Amazon Linux 2023

SELinux bietet eine zusätzliche Sicherheitsebene, um Container voneinander und vom Host zu isolieren. SELinux ermöglicht es Administratoren, verbindliche Zugriffskontrollen (MAC) für jeden Benutzer, jede Anwendung, jeden Prozess und jede Datei durchzusetzen. Stellen Sie sich das als Backstop vor, der die Operationen, die anhand einer Reihe von Labels ausgeführt werden können, auf bestimmte Ressourcen beschränkt. SELinux Kann auf EKS verwendet werden, um zu verhindern, dass Container gegenseitig auf die Ressourcen zugreifen.

SELinux Container-Richtlinien sind im Paket container-selinux definiert. Docker CE benötigt dieses Paket (zusammen mit seinen Abhängigkeiten), damit die von Docker (oder anderen Container-Laufzeiten) erstellten Prozesse und Dateien mit eingeschränktem Systemzugriff ausgeführt werden können. Container nutzen das `container_t` Label, das ein Alias für ist. `svirt_lxc_net_t` Diese Richtlinien verhindern effektiv, dass Container auf bestimmte Funktionen des Hosts zugreifen.

Wenn Sie SELinux für Docker konfigurieren, kennzeichnet Docker Workloads automatisch `container_t` als Typ und weist jedem Container eine eindeutige MCS-Ebene zu. Dadurch werden Container voneinander isoliert. Wenn Sie lockerere Einschränkungen benötigen, können Sie Ihr eigenes Profil erstellen SElinux , das einem Container Berechtigungen für bestimmte Bereiche des Dateisystems gewährt. Dies ist PSPs insofern vergleichbar, als Sie unterschiedliche Profile für verschiedene Container/Pods erstellen können. Sie können beispielsweise ein Profil für allgemeine

Workloads mit einer Reihe von restriktiven Kontrollen und ein anderes für Dinge haben, die privilegierten Zugriff erfordern.

SELinux for Containers verfügt über eine Reihe von Optionen, die konfiguriert werden können, um die Standardeinschränkungen zu ändern. Die folgenden SELinux Booleschen Werte können je nach Bedarf aktiviert oder deaktiviert werden:

Boolesch	Standard	Description
`container_connect_any`	`off`	Erlauben Sie Containern den Zugriff auf privilegierte Ports auf dem Host. Zum Beispiel, wenn Sie einen Container haben, der Ports 443 oder 80 auf dem Host zuordnen muss.
`container_manage_cgroup`	`off`	Erlauben Sie Container n, die Cgroup-Konfigurati on zu verwalten. Für einen Container, auf dem systemd ausgeführt wird, muss dies beispielsweise aktiviert sein.
`container_use_cephfs`	`off`	Erlaubt Containern, ein Ceph-Dateisystem zu verwenden.

Standardmäßig ist es Containern erlaubt, die meisten Inhalte zu read/execute `/usr` unterschreiben und daraus `/etc` zu lesen. Die Dateien unter `/var/lib/docker` und `/var/lib/containers` haben das Label`container_var_lib_t`. Eine vollständige Liste der Standard-Labels finden Sie in der Datei 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
```

Dateien, die mit gekennzeichnet sind, `container_file_t` sind die einzigen Dateien, die von Containern beschreibbar sind. Wenn Sie möchten, dass ein Volume-Mount beschreibbar ist, müssen Sie `:Z` am Ende den `:z` Wert oder angeben.

- `:z`benennt die Dateien neu, sodass der Container lesen/schreiben kann
- `:Z`benennt die Dateien neu, sodass nur der Container lesen/schreiben kann

```
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
```

In Kubernetes ist das Umetikettieren etwas anders. Anstatt Docker die Dateien automatisch umbenennen zu lassen, können Sie ein benutzerdefiniertes MCS-Label angeben, um den Pod auszuführen. Volumes, die das Umbenennen unterstützen, werden automatisch neu beschriftet, sodass auf sie zugegriffen werden kann. Pods mit einem passenden MCS-Label können auf das Volume zugreifen. Wenn Sie eine strikte Isolierung benötigen, legen Sie für jeden Pod ein anderes MCS-Label fest.

```
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
```

In diesem Beispiel `s0:c144:c154` entspricht dies einem MCS-Label, das einer Datei zugewiesen ist, auf die der Container zugreifen darf.

Auf EKS könnten Sie Richtlinien erstellen, die die Ausführung von privilegierten Containern ermöglichen, wie FluentD, und eine SELinux Richtlinie erstellen, die es ermöglicht, aus /var/log auf dem Host zu lesen, ohne das Hostverzeichnis neu benennen zu müssen. Pods mit derselben Bezeichnung können auf dieselben Host-Volumes zugreifen.

Wir haben ein Beispiel AMIs für Amazon EKS implementiert, das auf CentOS 7 und RHEL 7 SELinux konfiguriert wurde. Diese AMIs wurden entwickelt, um Beispielimplementierungen zu demonstrieren, die den Anforderungen stark regulierter Kunden entsprechen.

> **Warning**
>
> SELinux ignoriert Container, bei denen der Typ nicht begrenzt ist.

Tools und Ressourcen

- SELinuxKubernetes RBAC- und Versandsicherheitsrichtlinien für lokale Anwendungen
- Iteratives Härten von Kubernetes
- Audit2Allow
- SEAlert
- Generate SELinux Policies for Containers with Udica beschreibt ein Tool, das Container-Spezifikationsdateien für Linux-Funktionen, -Ports und Mount-Points untersucht und eine Reihe von SELinux Regeln generiert, die den ordnungsgemäßen Betrieb des Containers ermöglichen
- AMI Hardening-Playbooks zum Härten des Betriebssystems zur Erfüllung verschiedener regulatorischer Anforderungen
- Keiko Upgrade Manager ist ein Open-Source-Projekt von Intuit, das die Rotation von Worker-Knoten orchestriert.
- Sysdig Secure
- eksctl

Compliance

Die Einhaltung der Vorschriften ist eine gemeinsame Verantwortung von AWS und den Nutzern seiner Services. Im Allgemeinen ist AWS für die „Sicherheit der Cloud“ verantwortlich, während seine Benutzer für die „Sicherheit in der Cloud“ verantwortlich sind. Die Zeile, die beschreibt, wofür AWS und seine Benutzer verantwortlich sind, variiert je nach Service. Bei Fargate ist AWS beispielsweise für die Verwaltung der physischen Sicherheit seiner Rechenzentren, der Hardware, der virtuellen Infrastruktur (Amazon EC2) und der Container-Laufzeit (Docker) verantwortlich. Benutzer von Fargate sind für die Sicherung des Container-Images und ihrer Anwendung verantwortlich. Bei der Ausführung von Workloads, die den Compliance-Standards entsprechen müssen, ist es wichtig zu wissen, wer wofür verantwortlich ist.

Die folgende Tabelle zeigt die Compliance-Programme, denen die verschiedenen Container-Services entsprechen.

Compliance-Programm	Amazon ECS Orchestrator	Amazon EKS-Orchestrator	ECS Fargate	Amazon ECR
PCI DSS Stufe 1	1	1	1	1
HIPAA-konform	1	1	1	1
SOCK IST	1	1	1	1
SOC II	1	1	1	1
SOC III	1	1	1	1
ISO 27001:2013	1	1	1	1
ISO9001:2015	1	1	1	1
ISO 27017:2015	1	1	1	1
ISO 27018:2019	1	1	1	1
IRAP	1	1	1	1

Compliance-Programm	Amazon ECS Orchestrator	Amazon EKS-Orchestrator	ECS Fargate	Amazon ECR
FedRAMP Moderate (Osten/West)	1	1	0	1
FedRAMP Hoch () GovCloud	1	1	0	1
DOD CC SRG	1	DISA-Rezension () IL5	0	1
HIPAA BAA	1	1	1	1
MTCS	1	1	0	1
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

Der Compliance-Status ändert sich im Laufe der Zeit. Den aktuellen Status finden Sie immer unter https://aws.amazon.com/compliance/ services-in-scope /.

Weitere Informationen zu Cloud-Akkreditierungsmodellen und bewährten Verfahren finden Sie im AWS-Whitepaper Accreditation Models for Secure Cloud Adoption

Verlagerung nach links

Das Konzept der Linksverlagerung beinhaltet das Aufdecken von Richtlinienverstößen und Fehlern zu einem früheren Zeitpunkt im Lebenszyklus der Softwareentwicklung. Aus Sicht der Sicherheit kann dies sehr vorteilhaft sein. Ein Entwickler kann beispielsweise Probleme mit seiner Konfiguration beheben, bevor seine Anwendung im Cluster bereitgestellt wird. Wenn Sie solche Fehler früher

erkennen, können Sie verhindern, dass Konfigurationen bereitgestellt werden, die gegen Ihre Richtlinien verstoßen.

Richtlinie als Code

Politik kann als eine Reihe von Regeln zur Regelung von Verhaltensweisen betrachtet werden, d. h. Verhaltensweisen, die erlaubt oder verboten sind. Möglicherweise haben Sie eine Richtlinie, die besagt, dass alle Dockerfiles eine USER-Direktive enthalten sollten, die bewirkt, dass der Container als Nicht-Root-Benutzer ausgeführt wird. Als Dokument kann es schwierig sein, eine solche Richtlinie zu entdecken und durchzusetzen. Sie kann auch veraltet sein, wenn sich Ihre Anforderungen ändern. Mit Policy-as-Code (PaC) -Lösungen können Sie Sicherheits-, Compliance- und Datenschutzkontrollen automatisieren, um bekannte und anhaltende Bedrohungen zu erkennen, zu verhindern, zu reduzieren und ihnen entgegenzuwirken. Darüber hinaus bieten sie Ihnen die Möglichkeit, Ihre Richtlinien zu kodifizieren und sie wie andere Code-Artefakte zu verwalten. Der Vorteil dieses Ansatzes besteht darin, dass Sie Ihre DevOps GitOps Strategien wiederverwenden können, um Richtlinien für Flotten von Kubernetes-Clustern zu verwalten und konsistent anzuwenden. Informationen zu den PaC-Optionen und der future von finden Sie unter Pod Security PSPs.

Verwenden Sie policy-as-code Tools in Pipelines, um Verstöße vor der Bereitstellung zu erkennen

- OPA ist eine Open-Source-Policy-Engine, die Teil der CNCF ist. Es wird für politische Entscheidungen verwendet und kann auf verschiedene Arten betrieben werden, z. B. als Sprachbibliothek oder als Dienst. OPA-Richtlinien werden in einer domänenspezifischen Sprache (DSL) namens Rego geschrieben. OPA wird zwar häufig als Teil eines Kubernetes Dynamic Admission Controllers wie das Gatekeeper-Projekt ausgeführt, kann aber auch in Ihre Pipeline integriert werden. CI/CD Auf diese Weise können Entwickler zu einem früheren Zeitpunkt im Veröffentlichungszyklus Feedback zu ihrer Konfiguration erhalten, was ihnen anschließend helfen kann, Probleme zu lösen, bevor sie mit der Produktion beginnen. Eine Sammlung gängiger OPA-Richtlinien finden Sie im GitHub Repository für dieses Projekt.
- Conftest baut auf OPA auf und bietet eine auf Entwickler ausgerichtete Erfahrung zum Testen der Kubernetes-Konfiguration.
- Kyverno ist eine Policy-Engine, die für Kubernetes entwickelt wurde. Mit Kyverno werden Richtlinien als Kubernetes-Ressourcen verwaltet, und es ist keine neue Sprache erforderlich, um Richtlinien zu schreiben. Dies ermöglicht die Verwendung vertrauter Tools wie kubectl, git und kustomize zur Verwaltung von Richtlinien. Kyverno-Richtlinien können Kubernetes-Ressourcen

validieren, mutieren und generieren und gleichzeitig die Sicherheit der OCI-Image-Lieferkette gewährleisten. Die Kyverno CLI kann verwendet werden, um Richtlinien zu testen und Ressourcen als Teil einer CI/CD Pipeline zu validieren. Alle Richtlinien der Kyverno-Community finden Sie auf der Kyverno-Website. Beispiele für die Verwendung der Kyverno-CLI zum Schreiben von Tests in Pipelines finden Sie im Polices-Repository.

Tools und Ressourcen

- Amazon EKS Security Immersion Workshop — Einhaltung gesetzlicher Vorschriften
- Kube-Bench
- docker-bench-security
- AWS-Inspektor
- Kubernetes Security Review Eine Sicherheitsbewertung von Kubernetes 1.13.4 (2019) durch Dritte
- NeuVector Die Open-Source-Zero-Trust-Container-Sicherheitsplattform von SUSE bietet Compliance-Berichte und benutzerdefinierte Compliance-Prüfungen

Reaktion auf Vorfälle und Forensik

Ihre Fähigkeit, schnell auf einen Vorfall zu reagieren, kann dazu beitragen, den durch eine Sicherheitsverletzung verursachten Schaden zu minimieren. Ein zuverlässiges Warnsystem, das Sie vor verdächtigem Verhalten warnen kann, ist der erste Schritt zu einem guten Plan zur Reaktion auf Vorfälle. Wenn es doch einmal zu einem Zwischenfall kommt, müssen Sie schnell entscheiden, ob Sie den betroffenen Container vernichten und austauschen oder den Container isolieren und inspizieren möchten. Wenn Sie den Container im Rahmen einer forensischen Untersuchung und Ursachenanalyse isolieren möchten, sollten Sie die folgenden Maßnahmen ergreifen:

Beispiel für einen Plan zur Reaktion auf Vorfälle

Identifizieren Sie den Pod und den Worker-Knoten, die das Problem betreffen

Ihre erste Maßnahme sollte darin bestehen, den Schaden zu isolieren. Identifizieren Sie zunächst, wo der Verstoß aufgetreten ist, und isolieren Sie den Pod und seinen Knoten vom Rest der Infrastruktur.

Identifizieren Sie die betroffenen Pods und Worker-Knoten anhand des Workload-Namens

Wenn Sie den Namen und den Namespace des betreffenden Pods kennen, können Sie den Worker-Knoten, auf dem der Pod ausgeführt wird, wie folgt identifizieren:

```
kubectl get pods <name> --namespace <namespace> -o=jsonpath='{.spec.nodeName}{"\n"}'
```

Wenn eine Workload-Ressource wie ein Deployment kompromittiert wurde, sind wahrscheinlich alle Pods, die Teil der Workload-Ressource sind, gefährdet. Verwenden Sie den folgenden Befehl, um alle Pods der Workload-Ressource und die Knoten, auf denen sie ausgeführt werden, aufzulisten:

```
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)"'
```

Der obige Befehl ist für Bereitstellungen vorgesehen. Sie können denselben Befehl für andere Workload-Ressourcen wie Replicasets, Statefulsets usw. ausführen.

Identifizieren Sie die fehlerhaften Pods und Worker-Knoten anhand des Dienstkontonamens

In einigen Fällen stellen Sie möglicherweise fest, dass ein Dienstkonto gefährdet ist. Es ist wahrscheinlich, dass Pods, die das identifizierte Dienstkonto verwenden, kompromittiert wurden. Mit dem folgenden Befehl können Sie alle Pods anhand des Dienstkontos und der Knoten, auf denen sie ausgeführt werden, identifizieren:

```
kubectl get pods -o json --namespace <namespace> | \
    jq -r '.items[] |
    select(.spec.serviceAccount == "<service account name>") |
    "\(.metadata.name) \(.spec.nodeName)"'
```

Identifizieren Sie Pods mit anfälligen oder gefährdeten Images und Worker-Knoten

In einigen Fällen stellen Sie möglicherweise fest, dass ein Container-Image, das in Pods auf Ihrem Cluster verwendet wird, bösartig oder gefährdet ist. Ein Container-Image ist bösartig oder

kompromittiert, wenn festgestellt wurde, dass es Malware enthält, ein bekanntermaßen schlechtes Image ist oder ein CVE enthält, das ausgenutzt wurde. Sie sollten davon ausgehen, dass alle Pods, die das Container-Image verwenden, kompromittiert sind. Sie können die Pods anhand des Images und der Knoten, auf denen sie ausgeführt werden, mit dem folgenden Befehl identifizieren:

```
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)"'
```

Isolieren Sie den Pod, indem Sie eine Netzwerkrichtlinie erstellen, die den gesamten eingehenden und ausgehenden Datenverkehr zum Pod verweigert

Eine Regel „Gesamten Datenverkehr verweigern“ kann dazu beitragen, einen bereits laufenden Angriff zu stoppen, indem alle Verbindungen zum Pod unterbrochen werden. Die folgende Netzwerkrichtlinie gilt für einen Pod mit der Bezeichnung`app=web`.

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

> **Important**
>
> Eine Netzwerkrichtlinie kann sich als unwirksam erweisen, wenn sich ein Angreifer Zugriff auf den zugrundeliegenden Host verschafft hat. Wenn Sie vermuten, dass dies passiert ist, können Sie AWS-Sicherheitsgruppen verwenden, um einen gefährdeten Host von anderen Hosts zu isolieren. Wenn Sie die Sicherheitsgruppe eines Hosts ändern, sollten Sie sich bewusst sein, dass sich dies auf alle Container auswirkt, die auf diesem Host ausgeführt werden.

Widerrufen Sie gegebenenfalls die temporären Sicherheitsanmeldedaten, die dem Pod oder Worker-Knoten zugewiesen wurden

Wenn dem Worker-Knoten eine IAM-Rolle zugewiesen wurde, die es Pods ermöglicht, auf andere AWS-Ressourcen zuzugreifen, entfernen Sie diese Rollen aus der Instance, um weiteren Schaden durch den Angriff zu verhindern. Wenn dem Pod eine IAM-Rolle zugewiesen wurde, sollten Sie ebenfalls prüfen, ob Sie die IAM-Richtlinien sicher aus der Rolle entfernen können, ohne andere Workloads zu beeinträchtigen.

Sperren Sie den Worker-Knoten ab

Indem Sie den betroffenen Worker-Knoten sperren, informieren Sie den Scheduler, damit er verhindern kann, dass Pods auf dem betroffenen Knoten eingeplant werden. Auf diese Weise können Sie den Knoten für forensische Untersuchungen entfernen, ohne andere Workloads zu stören.

> **Note**
>
> Diese Anleitung gilt nicht für Fargate, wo jeder Fargate-Pod in seiner eigenen Sandbox-Umgebung ausgeführt wird. Anstatt sie abzusperren, können Sie die betroffenen Fargate-Pods absperren, indem Sie eine Netzwerkrichtlinie anwenden, die jeglichen eingehenden und ausgehenden Datenverkehr verweigert.

Aktivieren Sie den Kündigungsschutz für den betroffenen Worker-Knoten

Ein Angreifer kann versuchen, seine Missetaten auszulöschen, indem er einen betroffenen Knoten beendet. Durch die Aktivierung des Kündigungsschutzes kann dies verhindert werden. Der Instanz-Scale-In-Schutz schützt den Knoten vor einem Scale-In-Ereignis.

> **Warning**
>
> Sie können den Kündigungsschutz für eine Spot-Instance nicht aktivieren.

Kennzeichnen Sie den Verstoß Pod/Node mit einer Kennzeichnung, die darauf hinweist, dass er Teil einer aktiven Untersuchung ist

Dies dient als Warnung an die Clusteradministratoren, die betroffenen Objekte nicht zu manipulieren, Pods/Nodes bis die Untersuchung abgeschlossen ist.

Erfassen Sie flüchtige Artefakte auf dem Worker-Knoten

- Erfassen Sie den Arbeitsspeicher des Betriebssystems. Dadurch werden der Docker-Daemon (oder eine andere Container-Laufzeit) und seine Unterprozesse pro Container erfasst. Dies kann mithilfe von Tools wie LiME und Volatility oder durch darauf aufbauende Tools auf höherer Ebene wie Automated Forensics Orchestrator für Amazon EC2 erreicht werden.
- Führen Sie einen Netstat-Tree-Dump der laufenden Prozesse und der offenen Ports durch. Dadurch werden der Docker-Daemon und sein Unterprozess pro Container erfasst.
- Führen Sie Befehle aus, um den Status auf Containerebene zu speichern, bevor Beweise geändert werden. Sie können die Funktionen der Container-Laufzeit verwenden, um Informationen über aktuell ausgeführte Container zu erfassen. Mit Containerd könnten Sie beispielsweise Folgendes tun:
  - `crictl ps`für laufende Prozesse.
  - `crictl logs CONTAINER`für auf Daemon-Ebene gespeicherte Protokolle.

    Das Gleiche könnte mit containerd erreicht werden, indem die nerdctl-CLI anstelle von `docker` (z. B.) verwendet wird. `nerdctl inspect` Je nach Laufzeit des Containers sind einige zusätzliche Befehle verfügbar. Docker muss `docker diff` beispielsweise Änderungen am Container-Dateisystem erkennen oder `docker checkpoint` den gesamten Container-Status einschließlich des flüchtigen Speichers (RAM) speichern. In diesem Kubernetes-Blogbeitrag finden Sie Informationen zu ähnlichen Funktionen mit Containerd- oder CRI-O-Laufzeiten.
- Halten Sie den Container für die forensische Erfassung an.
- Erstellen Sie einen Snapshot der EBS-Volumes der Instance.

Stellen Sie die gefährdete Pod- oder Workload-Ressource erneut bereit

Sobald Sie Daten für die forensische Analyse gesammelt haben, können Sie die kompromittierte Pod- oder Workload-Ressource erneut bereitstellen.

Führen Sie zunächst das Update für die Sicherheitslücke aus, die gefährdet wurde, und starten Sie neue Ersatz-Pods. Löschen Sie anschließend die anfälligen Pods.

Wenn die anfälligen Pods von einer übergeordneten Kubernetes-Workload-Ressource (z. B. einem Deployment oder DaemonSet) verwaltet werden, werden beim Löschen der Pods neue geplant. Daher werden anfällige Pods erneut gestartet. In diesem Fall sollten Sie nach der Behebung der Sicherheitsanfälligkeit eine neue Ersatz-Workload-Ressource bereitstellen. Anschließend sollten Sie den anfälligen Workload löschen.

Empfehlungen

Lesen Sie das AWS-Whitepaper zur Reaktion auf Sicherheitsvorfälle

Dieser Abschnitt bietet zwar einen kurzen Überblick und einige Empfehlungen für den Umgang mit vermuteten Sicherheitsverletzungen, das Thema wird jedoch im Whitepaper AWS Security Incident Response ausführlich behandelt.

Üben Sie Spieltage zum Thema Sicherheit

Teilen Sie Ihre Sicherheitsexperten in zwei Teams ein: Rot und Blau. Das rote Team wird sich darauf konzentrieren, verschiedene Systeme auf Sicherheitslücken zu untersuchen, während das blaue Team für die Abwehr dieser Schwachstellen verantwortlich ist. Wenn Sie nicht über genügend Sicherheitsexperten verfügen, um separate Teams zusammenzustellen, sollten Sie erwägen, eine externe Stelle einzustellen, die sich mit den Exploits von Kubernetes auskennt.

Kubesploit ist ein Framework für Penetrationstests CyberArk , mit dem Sie Spieltage durchführen können. Im Gegensatz zu anderen Tools, die Ihren Cluster nach Schwachstellen durchsuchen, simuliert Kubesploit einen realen Angriff. Auf diese Weise hat Ihr blaues Team die Möglichkeit, seine Reaktion auf einen Angriff zu üben und seine Effektivität zu beurteilen.

Führen Sie Penetrationstests für Ihren Cluster durch

Regelmäßige Angriffe auf Ihren eigenen Cluster können Ihnen dabei helfen, Sicherheitslücken und Fehlkonfigurationen aufzudecken. Bevor Sie beginnen, sollten Sie die Richtlinien für Penetrationstests befolgen, bevor Sie einen Test mit Ihrem Cluster durchführen.

Tools und Ressourcen

- kube-hunter, ein Penetrationstest-Tool für Kubernetes.
- Gremlin, ein Chaos-Engineering-Toolkit, mit dem Sie Angriffe auf Ihre Anwendungen und Infrastruktur simulieren können.
- Kubernetes-Installationen angreifen und verteidigen
- kubesploit
- NeuVector Die Open-Source-Zero-Trust-Container-Sicherheitsplattform von SUSE bietet Berichte über Sicherheitslücken und Risiken sowie Benachrichtigungen über Sicherheitsereignisse
- Fortgeschrittene persistente Bedrohungen
- Praktischer Angriff und Verteidigung von Kubernetes

- Kompromittierung des Kubernetes-Clusters durch Ausnutzung von RBAC-Berechtigungen

Image-Sicherheit

Sie sollten das Container-Image als erste Verteidigungslinie gegen einen Angriff betrachten. Ein unsicheres, schlecht aufgebautes Image kann es einem Angreifer ermöglichen, die Grenzen des Containers zu umgehen und sich Zugriff auf den Host zu verschaffen. Sobald sich ein Angreifer auf dem Host befindet, kann er Zugriff auf vertrauliche Informationen erhalten oder sich innerhalb des Clusters oder mit Ihrem AWS-Konto seitlich bewegen. Die folgenden bewährten Methoden tragen dazu bei, das Risiko eines solchen Vorfalls zu verringern.

Empfehlungen

Erstellen Sie ein Minimum an Bildern

Entfernen Sie zunächst alle überflüssigen Binärdateien aus dem Container-Image. Wenn Sie ein unbekanntes Bild von Dockerhub verwenden, überprüfen Sie das Bild mit einer Anwendung wie Dive, die Ihnen den Inhalt der einzelnen Ebenen des Containers anzeigen kann. Entfernen Sie alle Binärdateien mit den Bits SETUID und SETGID, da sie zur Erweiterung von Rechten verwendet werden können, und erwägen Sie, alle Shells und Hilfsprogramme wie nc und curl zu entfernen, die für schändliche Zwecke verwendet werden können. Sie können die Dateien mit den Bits SETUID und SETGID mit dem folgenden Befehl finden:

```
find / -perm /6000 -type f -exec ls -ld {} \;
```

Um die Sonderberechtigungen aus diesen Dateien zu entfernen, fügen Sie Ihrem Container-Image die folgende Direktive hinzu:

```
RUN find / -xdev -perm /6000 -type f -exec chmod a-s {} \; || true
```

Umgangssprachlich wird dies als De-Fanging Ihres Images bezeichnet.

Verwenden Sie mehrstufige Builds

Die Verwendung mehrstufiger Builds ist eine Möglichkeit, minimale Images zu erstellen. Oft werden mehrstufige Builds verwendet, um Teile des Continuous Integration-Zyklus zu automatisieren. Mehrstufige Builds können beispielsweise verwendet werden, um Ihren Quellcode zu verfälschen

oder eine statische Codeanalyse durchzuführen. Dies bietet Entwicklern die Möglichkeit, nahezu sofortiges Feedback zu erhalten, anstatt auf die Ausführung einer Pipeline zu warten. Mehrstufige Builds sind aus Sicherheitsgründen attraktiv, da Sie damit die Größe des endgültigen Images minimieren können, das in Ihre Container-Registry übertragen wird. Container-Images ohne Build-Tools und andere überflüssige Binärdateien verbessern Ihre Sicherheitslage, indem sie die Angriffsfläche des Images reduzieren. Weitere Informationen zu mehrstufigen Builds finden Sie in der Dokumentation zu mehrstufigen Builds von Docker.

Erstellen Sie eine Softwareliste (SBOMs) für Ihr Container-Image

Eine „Softwareliste“ (SBOM) ist ein verschachteltes Inventar der Software-Artefakte, aus denen Ihr Container-Image besteht. SBOM ist ein wichtiger Baustein für Softwaresicherheit und Risikomanagement in der Software-Lieferkette. Das Generieren, Speichern von SBOMS in einem zentralen Repository und das Scannen SBOMs nach Sicherheitslücken trägt dazu bei, die folgenden Probleme auszuräumen:

- Sichtbarkeit: Verstehen Sie, aus welchen Komponenten Ihr Container-Image besteht. Durch die Speicherung in einem zentralen Repository können SBOMs Sie jederzeit, auch nach der Implementierung, geprüft und gescannt werden, um neue Sicherheitslücken wie Zero-Day-Sicherheitslücken zu erkennen und darauf zu reagieren.
- Überprüfung der Herkunft: Gewissheit, dass bestehende Annahmen darüber, woher und wie ein Artefakt stammt, wahr sind und dass das Artefakt oder die zugehörigen Metadaten während des Erstellungs- oder Bereitstellungsprozesses nicht manipuliert wurden.
- Vertrauenswürdigkeit: Zusicherung, dass einem bestimmten Artefakt und seinem Inhalt das anvertraut werden kann, wofür es vorgegeben ist, d. h. dass es für einen bestimmten Zweck geeignet ist. Dazu gehört die Beurteilung, ob der Code sicher ausgeführt werden kann, und fundierte Entscheidungen über die mit der Ausführung des Codes verbundenen Risiken zu treffen. Die Vertrauenswürdigkeit wird durch die Erstellung eines bestätigten Pipeline-Ausführungsberichts zusammen mit einem bestätigten SBOM und einem beglaubigten CVE-Scanbericht gewährleistet, um den Benutzern des Images zu versichern, dass dieses Image tatsächlich auf sichere Weise (Pipeline) mit sicheren Komponenten erstellt wurde.
- Vertrauensverifizierung durch Abhängigkeit: rekursive Überprüfung des Abhängigkeitsbaums eines Artefakts auf Vertrauenswürdigkeit und Herkunft der verwendeten Artefakte. Drift in SBOMs kann dabei helfen, böswillige Aktivitäten wie unbefugte, nicht vertrauenswürdige Abhängigkeiten und Infiltrationsversuche zu erkennen.

Die folgenden Tools können zur Generierung von SBOM verwendet werden:

- Amazon Inspector kann zum Erstellen und Exportieren verwendet werden SBOMs.
- Syft von Anchore kann auch für die SBOM-Generierung verwendet werden. Für schnellere Schwachstellenscans kann die für ein Container-Image generierte SBOM als Eingabe für den Scan verwendet werden. Die SBOM und der Scanbericht werden dann bestätigt und an das Bild angehängt, bevor das Bild zu Überprüfungs- und Auditzwecken an ein zentrales OCI-Repository wie Amazon ECR übertragen wird.

Weitere Informationen zur Sicherung Ihrer Software-Lieferkette finden Sie im CNCF-Leitfaden zu Best Practices für die Softwarelieferkette.

Scannen Sie Bilder regelmäßig auf Sicherheitslücken

Wie ihre Gegenstücke zu virtuellen Maschinen können Container-Images Binärdateien und Anwendungsbibliotheken mit Sicherheitslücken enthalten oder im Laufe der Zeit Sicherheitslücken entwickeln. Am besten schützen Sie sich vor Angriffen, indem Sie Ihre Images regelmäßig mit einem Image-Scanner überprüfen. Bilder, die in Amazon ECR gespeichert sind, können auf Push- oder On-Demand-Modus (einmal innerhalb von 24 Stunden) gescannt werden. ECR unterstützt derzeit zwei Arten des Scannens: Basic und Enhanced. Basic Scanning nutzt Clair, eine kostenlose Open-Source-Lösung zum Scannen von Bildern. Enhanced Scanning verwendet Amazon Inspector, um gegen Aufpreis automatische kontinuierliche Scans bereitzustellen. Nachdem ein Bild gescannt wurde, werden die Ergebnisse im EventBridge Event-Stream für ECR protokolliert. Sie können die Ergebnisse eines Scans auch von der ECR-Konsole aus sehen. Bilder mit der Sicherheitslücke HOCH oder KRITISCH sollten gelöscht oder neu erstellt werden. Wenn ein bereitgestelltes Image eine Schwachstelle aufweist, sollte es so schnell wie möglich ersetzt werden.

Für die Sicherheit Ihrer Umgebung ist es wichtig zu wissen, wo Images mit Sicherheitslücken bereitgestellt wurden. Es ist zwar denkbar, eine Image-Tracking-Lösung selbst zu entwickeln, aber es gibt bereits mehrere kommerzielle Angebote, die diese und andere erweiterte Funktionen sofort bereitstellen, darunter:

- Grype
- Palo Alto - Prismawolke (twistcli)
- Aquamarin
- Kubei
- Trivy
- Snyk

Ein Kubernetes-Validierungs-Webhook könnte auch verwendet werden, um zu überprüfen, ob Images frei von kritischen Sicherheitslücken sind. Validierungs-Webhooks werden vor der Kubernetes-API aufgerufen. Sie werden in der Regel verwendet, um Anfragen abzulehnen, die nicht den im Webhook definierten Validierungskriterien entsprechen. Dies ist ein Beispiel für einen serverlosen Webhook, der die ECR describeImageScan Findings API aufruft, um festzustellen, ob ein Pod ein Bild mit kritischen Sicherheitslücken abruft. Wenn Sicherheitslücken gefunden werden, wird der Pod zurückgewiesen und eine Nachricht mit einer Liste von CVEs wird als Ereignis zurückgegeben.

Verwenden Sie Bescheinigungen, um die Integrität von Artefakten zu überprüfen

Eine Bescheinigung ist eine kryptografisch signierte „Aussage“, die behauptet, dass etwas — ein „Prädikat“, z. B. ein Pipeline-Lauf, die SBOM oder der Schwachstellenscan-Bericht — wahr ist, und zwar über ein „Subjekt“, d. h. das Container-Image.

Mithilfe von Bescheinigungen können Benutzer überprüfen, ob ein Artefakt aus einer vertrauenswürdigen Quelle in der Software-Lieferkette stammt. Beispielsweise können wir ein Container-Image verwenden, ohne alle Softwarekomponenten oder Abhängigkeiten zu kennen, die in diesem Image enthalten sind. Wenn wir jedoch den Aussagen des Herstellers des Container-Images über die vorhandene Software vertrauen, können wir die Bescheinigung des Herstellers verwenden, um uns auf dieses Artefakt zu verlassen. Das bedeutet, dass wir das Artefakt sicher in unserem Arbeitsablauf verwenden können, anstatt die Analyse selbst durchgeführt zu haben.

- Bescheinigungen können mit AWS Signer oder Sigstore Cosign erstellt werden.
- Kubernetes-Zugangscontroller wie Kyverno können zur Überprüfung von Bescheinigungen verwendet werden.
- In diesem Workshop erfahren Sie mehr über bewährte Methoden für das Software-Supply-Chain-Management auf AWS mithilfe von Open-Source-Tools. Themen wie das Erstellen und Anhängen von Bescheinigungen an ein Container-Image sind Themen wie das Erstellen und Anhängen von Bescheinigungen an ein Container-Image.

Erstellen Sie IAM-Richtlinien für ECR-Repositorys

Heutzutage ist es nicht ungewöhnlich, dass ein Unternehmen mehrere Entwicklungsteams hat, die unabhängig voneinander innerhalb eines gemeinsamen AWS-Kontos arbeiten. Wenn diese Teams keine Ressourcen gemeinsam nutzen müssen, sollten Sie eine Reihe von IAM-Richtlinien erstellen, die den Zugriff auf die Repositorys einschränken, mit denen jedes Team interagieren kann. Eine gute Möglichkeit, dies zu implementieren, ist die Verwendung von ECR-Namespaces. Namespaces sind

eine Möglichkeit, ähnliche Repositorys zu gruppieren. Beispielsweise kann allen Registern für Team A das Präfix team-a/ vorangestellt werden, während denen für Team B das Präfix team-b/ verwendet werden kann. Die Richtlinie zur Zugriffsbeschränkung könnte wie folgt aussehen:

```
{
  "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/*"
      ]
    }
  ]
}
```

Erwägen Sie die Verwendung von privaten ECR-Endpunkten

Die ECR-API hat einen öffentlichen Endpunkt. Folglich kann über das Internet auf ECR-Register zugegriffen werden, sofern die Anfrage von IAM authentifiziert und autorisiert wurde. Für Benutzer, die in einer Sandbox-Umgebung arbeiten müssen, in der die Cluster-VPC kein Internet Gateway (IGW) hat, können Sie einen privaten Endpunkt für ECR konfigurieren. Wenn Sie einen privaten Endpunkt erstellen, können Sie über eine private IP-Adresse privat auf die ECR-API zugreifen, anstatt den Datenverkehr über das Internet weiterzuleiten. Weitere Informationen zu diesem Thema finden Sie unter VPC-Endpoints mit Amazon ECR Interface.

Implementieren Sie Endpunktrichtlinien für ECR

Die Standard-Endpunktrichtlinie für ermöglicht den Zugriff auf alle ECR-Repositorys innerhalb einer Region. Dies könnte es einem ermöglichen, Daten attacker/insider zu exfiltrieren, indem sie als Container-Image verpackt und in eine Registrierung in einem anderen AWS-Konto übertragen

werden. Um dieses Risiko zu minimieren, müssen Sie eine Endpunktrichtlinie erstellen, die den API-Zugriff auf ECR-Repositorys einschränkt. Die folgende Richtlinie ermöglicht es beispielsweise allen AWS-Prinzipien in Ihrem Konto, alle Aktionen gegen Ihre und nur gegen Ihre ECR-Repositorys durchzuführen:

```
{
  "Statement": [
    {
      "Sid": "LimitECRAccess",
      "Principal": "*",
      "Action": "*",
      "Effect": "Allow",
      "Resource": "arn:aws:ecr:<region>:<account_id>:repository/*"
    }
  ]
}
```

Sie können dies weiter verbessern, indem Sie eine Bedingung festlegen, die das neue `PrincipalOrgID` Attribut verwendet, wodurch verhindert pushing/pulling wird, dass Bilder nach einem IAM-Prinzip, das nicht Teil Ihrer AWS-Organisation ist, erstellt werden. Weitere Informationen finden Sie unter aws: PrincipalOrg ID. Wir haben empfohlen, dieselbe Richtlinie sowohl auf die `com.amazonaws.<region>.ecr.dkr`- als auch die `com.amazonaws.<region>.ecr.api`-Endpunkte anzuwenden. Da EKS Images für kube-proxy, coredns und aws-node aus ECR bezieht, müssen Sie die Konto-ID der Registrierung hinzufügen, z. B. zur Liste der Ressourcen in der Endpunktrichtlinie, oder die Richtlinie ändern, `602401143452.dkr.ecr.us-west-2.amazonaws.com/` um Pulls von Ihrer Konto-ID zuzulassen und Pushs auf diese zu beschränken. Die folgende Tabelle zeigt die Zuordnung zwischen den AWS-Konten, von denen EKS-Images verkauft werden, und der Cluster-Region.

Account Number (Kontonummer)	Region
602401143452	Alle Handelsregionen mit Ausnahme der unten aufgeführten
—	—
800184023465	ap-east-1 - Asien-Pazifik (Hongkong)
558608220178	me-south-1 - Naher Osten (Bahrain)

Account Number (Kontonummer)	Region
918309763551	cn-north-1 - China (Peking)
961992271922	cn-northwest-1 - China (Ningxia)

Weitere Informationen zur Verwendung von Endpunktrichtlinien finden Sie unter Verwenden von VPC-Endpunktrichtlinien zur Steuerung des Amazon ECR-Zugriffs.

Implementieren Sie Lebenszyklusrichtlinien für ECR

Der NIST Application Container Security Guide warnt vor dem Risiko „veralteter Images in Registern“ und weist darauf hin, dass alte Images mit anfälligen out-of-date Softwarepaketen im Laufe der Zeit entfernt werden sollten, um eine versehentliche Bereitstellung und Offenlegung zu verhindern. Jedes ECR-Repository kann über eine Lebenszyklusrichtlinie verfügen, die Regeln dafür festlegt, wann Images ablaufen. In der offiziellen AWS-Dokumentation wird beschrieben, wie Testregeln eingerichtet, bewertet und anschließend angewendet werden. In den offiziellen Dokumenten finden Sie mehrere Beispiele für Lebenszyklusrichtlinien, die verschiedene Möglichkeiten zum Filtern der Bilder in einem Repository zeigen:

- Filterung nach Alter oder Anzahl der Bilder
- Filterung nach markierten oder nicht markierten Bildern
- Filterung nach Bild-Tags, entweder in mehreren Regeln oder in einer einzigen Regel

??? + Warnung Wenn das Image einer Anwendung mit langer Laufzeit aus ECR gelöscht wird, kann dies zu Fehlern beim Abrufen des Images führen, wenn die Anwendung erneut bereitgestellt oder horizontal skaliert wird. Achten Sie bei der Verwendung von Image-Lebenszyklusrichtlinien darauf, dass Sie über CI/CD bewährte Verfahren verfügen, um Bereitstellungen und die Images, auf die sie verweisen, auf dem neuesten Stand zu halten, und immer Ablaufregeln für [Image] zu erstellen, die berücksichtigen, wie oft Sie Releases/Bereitstellungen durchführen.

Eine Reihe von kuratierten Images erstellen

Anstatt es Entwicklern zu ermöglichen, ihre eigenen Images zu erstellen, sollten Sie erwägen, eine Reihe von geprüften Images für die verschiedenen Anwendungsstapel in Ihrem Unternehmen zu erstellen. Auf diese Weise können Entwickler darauf verzichten, zu lernen, wie man Dockerfiles erstellt, und sich auf das Schreiben von Code konzentrieren. Wenn Änderungen in Master

zusammengeführt werden, kann eine CI/CD Pipeline das Asset automatisch kompilieren, in einem Artefakt-Repository speichern und das Artefakt in das entsprechende Image kopieren, bevor es in eine Docker-Registry wie ECR übertragen wird. Zumindest sollten Sie eine Reihe von Basis-Images erstellen, aus denen Entwickler ihre eigenen Dockerfiles erstellen können. Im Idealfall sollten Sie vermeiden, Bilder aus Dockerhub abzurufen, da Sie zu einem Teil nicht immer wissen, was im Image enthalten ist und zum anderen etwa ein Fünftel der 1000 besten Images Sicherheitslücken aufweist. Eine Liste dieser Images und ihrer Sicherheitslücken finden Sie hier.

Fügen Sie die USER-Direktive zu Ihren Dockerfiles hinzu, um sie als Nicht-Root-Benutzer auszuführen

Wie im Abschnitt zur Pod-Sicherheit erwähnt, sollten Sie es vermeiden, den Container als Root auszuführen. Sie können dies zwar als Teil der PodSpec konfigurieren, es ist jedoch eine gute Angewohnheit, die `USER` Direktive für Ihre Dockerfiles zu verwenden. Die `USER` Direktive legt die UID fest, die beim Ausführen verwendet werden soll, oder die `CMD` Anweisung `RUNENTRYPOINT`, die nach der USER-Direktive erscheint.

Linten Sie Ihre Dockerfiles

Linting kann verwendet werden, um zu überprüfen, ob Ihre Dockerfiles einer Reihe von vordefinierten Richtlinien entsprechen, z. B. der Aufnahme der `USER` Direktive oder der Anforderung, dass alle Bilder markiert werden müssen. dockerfile_lint ist ein Open-Source-Projekt von, RedHat das gängige Best Practices verifiziert und eine Regel-Engine enthält, mit der Sie Ihre eigenen Regeln für das Linting von Dockerfiles erstellen können. Es kann in eine CI-Pipeline integriert werden, sodass Builds mit Dockerfiles, die gegen eine Regel verstoßen, automatisch fehlschlagen.

Erstellen Sie Bilder von Grund auf neu

Die Reduzierung der Angriffsfläche Ihrer Container-Images sollte das Hauptziel bei der Erstellung von Images sein. Der ideale Weg, dies zu tun, besteht darin, möglichst wenige Images ohne Binärdateien zu erstellen, mit denen Sicherheitslücken ausgenutzt werden können. Zum Glück verfügt Docker über einen Mechanismus, aus dem Bilder erstellt werden können. `scratch` Mit Sprachen wie Go können Sie eine statische verknüpfte Binärdatei erstellen und in Ihrem Dockerfile darauf verweisen, wie in diesem Beispiel:

```
############################
# 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"]
```

Dadurch wird ein Container-Image erstellt, das aus Ihrer Anwendung und nichts anderem besteht, was es extrem sicher macht.

Verwenden Sie unveränderliche Tags mit ECR

Unveränderliche Tags zwingen Sie, das Image-Tag bei jedem Push in das Image-Repository zu aktualisieren. Dadurch kann verhindert werden, dass ein Angreifer ein Bild mit einer bösartigen Version überschreibt, ohne die Bild-Tags zu ändern. Darüber hinaus bietet es Ihnen die Möglichkeit, ein Bild einfach und eindeutig zu identifizieren.

Signieren Sie Ihre Bilder SBOMs, Pipeline-Läufe und Schwachstellenberichte

Als Docker zum ersten Mal eingeführt wurde, gab es kein kryptografisches Modell zur Überprüfung von Container-Images. Mit Version 2 fügte Docker dem Image-Manifest Digests hinzu. Dadurch konnte die Konfiguration eines Images gehasht und der Hash verwendet werden, um eine ID für das Image zu generieren. Wenn die Image-Signierung aktiviert ist, verifiziert die Docker-Engine die Signatur des Manifests und stellt so sicher, dass der Inhalt aus einer vertrauenswürdigen Quelle stammt und keine Manipulation stattgefunden hat. Nach dem Herunterladen jeder Ebene überprüft die Engine den Digest der Ebene und stellt sicher, dass der Inhalt mit dem im Manifest angegebenen Inhalt übereinstimmt. Mit der Bildsignierung können Sie effektiv eine sichere Lieferkette einrichten, indem die mit dem Bild verknüpften digitalen Signaturen überprüft werden.

Wir können AWS Signer oder Sigstore Cosign verwenden, um Container-Images zu signieren, Bescheinigungen zu erstellen, Schwachstellen-Scan-Berichte und SBOMs Pipeline-Run-Berichte zu erstellen. Diese Bescheinigungen gewährleisten die Vertrauenswürdigkeit und Integrität des Images, d. h., dass es tatsächlich von der vertrauenswürdigen Pipeline ohne jegliche Eingriffe oder

Manipulationen erstellt wurde und dass es nur die Softwarekomponenten enthält, die dokumentiert sind (in der SBOM), die vom Herausgeber des Images verifiziert und als vertrauenswürdig eingestuft wurden. Diese Bescheinigungen können an das Container-Image angehängt und in das Repository übertragen werden.

Im nächsten Abschnitt werden wir sehen, wie die beglaubigten Artefakte für Audits und die Überprüfung durch die Zulassungsbehörden verwendet werden können.

Überprüfung der Bildintegrität mithilfe des Kubernetes Admission Controllers

Wir können Bildsignaturen und bestätigte Artefakte auf automatisierte Weise überprüfen, bevor wir das Image mithilfe des Dynamic Admission Controllers auf dem Ziel-Kubernetes-Cluster bereitstellen. Bereitstellungen können nur zugelassen werden, wenn die Sicherheitsmetadaten der Artefakte den Richtlinien des Zugangscontrollers entsprechen.

Wir können beispielsweise eine Richtlinie schreiben, die die Signatur eines Images kryptografisch verifiziert, eine bescheinigte SBOM, einen bescheinigten Pipeline-Run-Bericht oder einen beglaubigten CVE-Scanbericht. Wir können Bedingungen in die Richtlinie schreiben, um Daten im Bericht zu überprüfen, z. B. sollte ein CVE-Scan keine kritischen Daten enthalten. CVEs Die Bereitstellung ist nur für Bilder zulässig, die diese Bedingungen erfüllen. Alle anderen Bereitstellungen werden vom Zulassungskontrolleur abgelehnt.

Zu den Beispielen für den Zugangscontroller gehören:

- Kyverno
- OPA Torwächter
- Portier ist
- Ratifizieren
- Kritis
- Grafeas Anleitung
- Gutschein

Aktualisieren Sie die Pakete in Ihren Container-Images

Sie sollten RUN `apt-get update && apt-get upgrade` in Ihre Dockerfiles aufnehmen, um die Pakete in Ihren Images zu aktualisieren. Für das Upgrade müssen Sie zwar als Root ausführen, dies geschieht jedoch während der Image-Erstellungsphase. Die Anwendung muss nicht als Root-

Benutzer ausgeführt werden. Sie können die Updates installieren und dann mit der USER-Direktive zu einem anderen Benutzer wechseln. Wenn Ihr Basis-Image nicht als Root-Benutzer ausgeführt wird, wechseln Sie zu Root und zurück. Verlassen Sie sich bei der Installation der neuesten Sicherheitsupdates nicht ausschließlich auf die Betreuer des Basis-Images.

Führen Sie `apt-get clean` den Befehl aus, um die Installationsdateien von zu löschen. `/var/cache/apt/archives/` Sie können das Programm auch `rm -rf /var/lib/apt/lists/*` nach der Installation von Paketen ausführen. Dadurch werden die Indexdateien oder die Listen der Pakete entfernt, die zur Installation verfügbar sind. Beachten Sie, dass diese Befehle für jeden Paketmanager unterschiedlich sein können. Beispiel:

```
RUN apt-get update && apt-get install -y \
    curl \
    git \
    libsqlite3-dev \
    && apt-get clean && rm -rf /var/lib/apt/lists/*
```

Tools und Ressourcen

- Workshop zum Eintauchen in die Sicherheit von Amazon EKS — Bildsicherheit
- docker-slim Erstellen Sie sichere Minimal-Images
- dockle Überprüft, ob Ihr Dockerfile den Best Practices für die Erstellung sicherer Images entspricht
- dockerfile-lint Regelbasierter Linter für Dockerfiles
- hadolint Ein intelligenter Dockerfile-Linter
- Gatekeeper und OPA Ein richtlinienbasierter Zugangscontroller
- Kyverno Eine native Policy-Engine für Kubernetes
- in-toto Ermöglicht es dem Benutzer zu überprüfen, ob ein Schritt in der Lieferkette ausgeführt werden sollte und ob der Schritt vom richtigen Akteur ausgeführt wurde
- Notar Ein Projekt zum Signieren von Container-Images
- Notar v2
- Grafeas Eine offene API für Artefakt-Metadaten zur Prüfung und Steuerung Ihrer Software-Lieferkette
- NeuVector von SUSE Open Source, einer Zero-Trust-Container-Sicherheitsplattform, die Container-, Image- und Registry-Scans auf Sicherheitslücken, geheime Daten und Konformität ermöglicht.

Strategie für mehrere Konten

AWS empfiehlt, eine Strategie für mehrere Konten und AWS-Organisationen zu verwenden, um Ihre Geschäftsanwendungen und -daten zu isolieren und zu verwalten. Die Verwendung einer Strategie für mehrere Konten bietet viele Vorteile:

- Höhere AWS-API-Servicekontingente. Kontingente werden auf AWS-Konten angewendet, und wenn Sie mehrere Konten für Ihre Workloads verwenden, erhöht sich das für Ihre Workloads verfügbare Gesamtkontingent.
- Einfachere Richtlinien für Identity and Access Management (IAM). Wenn Sie Workloads und den Betreibern, die sie unterstützen, nur Zugriff auf ihre eigenen AWS-Konten gewähren, bedeutet dies weniger Zeit für die Erstellung detaillierter IAM-Richtlinien, um das Prinzip der geringsten Rechte zu erreichen.
- Verbesserte Isolierung von AWS-Ressourcen. Standardmäßig sind alle innerhalb eines Kontos bereitgestellten Ressourcen logisch von Ressourcen isoliert, die in anderen Konten bereitgestellt werden. Diese Isolationsgrenze bietet Ihnen die Möglichkeit, das Risiko anwendungsbezogener Probleme, Fehlkonfigurationen oder böswilliger Aktionen zu begrenzen. Wenn ein Problem innerhalb eines Kontos auftritt, können die Auswirkungen auf die Workloads anderer Konten entweder reduziert oder vermieden werden.
- Weitere Vorteile, wie im Whitepaper AWS Multi Account Strategy beschrieben

In den folgenden Abschnitten wird erklärt, wie Sie eine Strategie mit mehreren Konten für Ihre EKS-Workloads mithilfe eines zentralisierten oder dezentralen EKS-Cluster-Ansatzes implementieren.

Planung einer Kontenstrategie für mehrere Workloads für Multi-Tenant-Cluster

In einer AWS-Strategie mit mehreren Konten werden Ressourcen, die zu einem bestimmten Workload gehören, wie S3-Buckets, ElastiCache Cluster und DynamoDB-Tabellen, alle in einem AWS-Konto erstellt, das alle Ressourcen für diesen Workload enthält. Diese werden als Workload-Konto bezeichnet, und der EKS-Cluster wird in einem Konto bereitgestellt, das als Cluster-Konto bezeichnet wird. Clusterkonten werden im nächsten Abschnitt untersucht. Die Bereitstellung von Ressourcen in einem dedizierten Workload-Konto ähnelt der Bereitstellung von Kubernetes-Ressourcen in einem dedizierten Namespace.

Workload-Konten können dann gegebenenfalls weiter nach dem Lebenszyklus der Softwareentwicklung oder nach anderen Anforderungen unterteilt werden. Beispielsweise kann ein

bestimmter Workload über ein Produktionskonto, ein Entwicklungskonto oder Konten für das Hosten von Instanzen dieses Workloads in einer bestimmten Region verfügen. Weitere Informationen finden Sie in diesem AWS-Whitepaper.

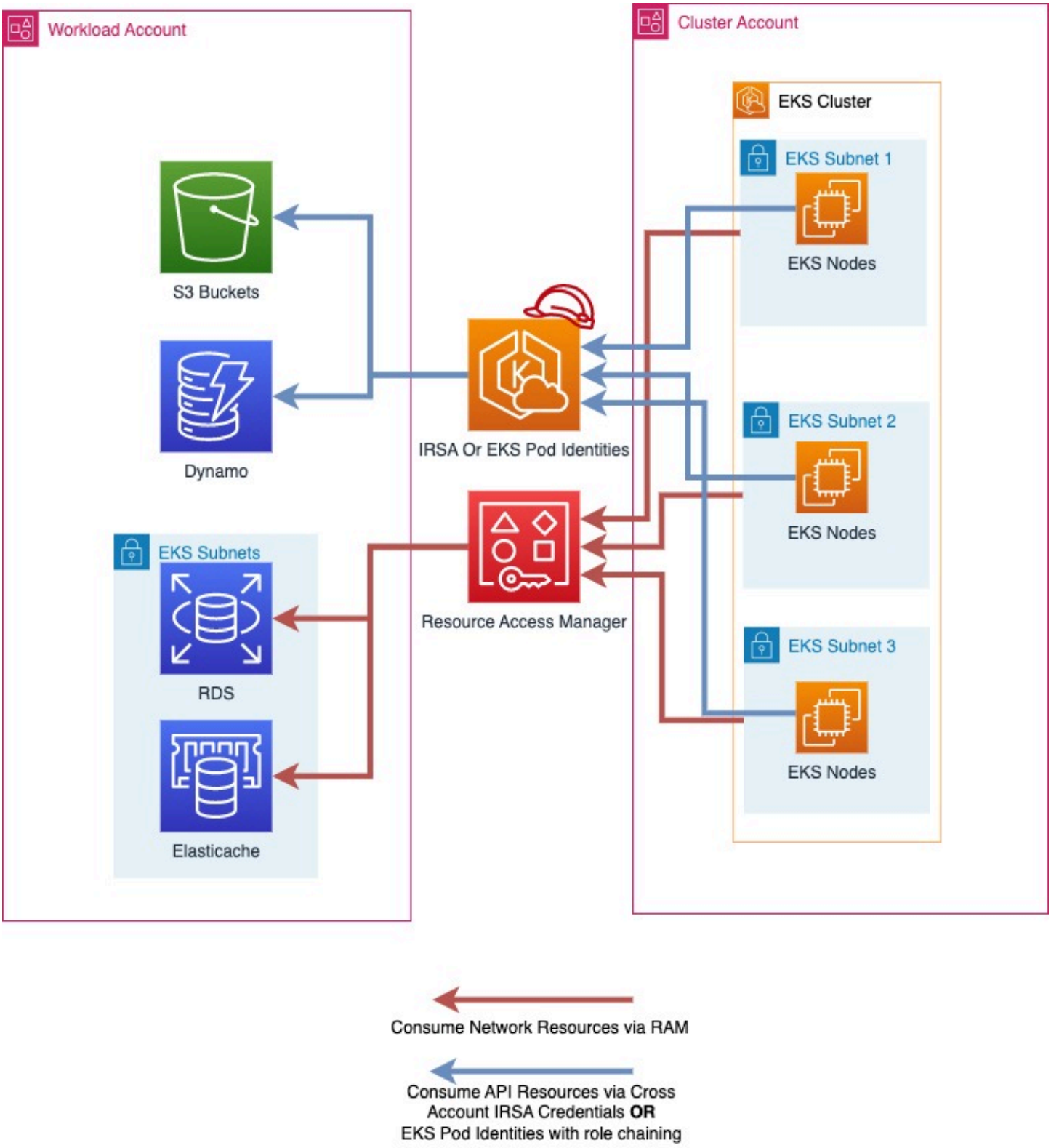
Bei der Implementierung der EKS-Strategie für mehrere Konten können Sie die folgenden Ansätze anwenden:

Zentralisierter EKS-Cluster

Bei diesem Ansatz wird Ihr EKS-Cluster in einem einzigen AWS-Konto namens bereitgestelltCluster Account. Mithilfe von IAM-Rollen für Service Accounts (IRSA) oder EKS-Pod-Identitäten zur Bereitstellung temporärer AWS-Anmeldeinformationen und AWS Resource Access Manager (RAM) zur Vereinfachung des Netzwerkzugriffs können Sie eine Strategie für mehrere Konten für Ihren EKS-Cluster mit mehreren Mandanten verfolgen. Das Clusterkonto enthält die VPC, die Subnetze, den EKS-Cluster, die EC2/Fargate-Rechenressourcen (Worker-Knoten) und alle zusätzlichen Netzwerkkonfigurationen, die für den Betrieb Ihres EKS-Clusters erforderlich sind.

In einer Multi-Workload-Kontostrategie für Multi-Tenant-Cluster orientieren sich AWS-Konten in der Regel an Kubernetes-Namespaces als Mechanismus zur Isolierung von Ressourcengruppen. Bei der Implementierung einer Strategie für mehrere Konten für EKS-Cluster mit mehreren Mandanten sollten weiterhin die bewährten Methoden für die Mandantenisolierung innerhalb eines EKS-Clusters befolgt werden.

Es ist möglich, mehrere Cluster Accounts in Ihrer AWS-Organisation zu haben, und es ist eine bewährte Methode, mehrere zu habenCluster Accounts, die Ihren Anforderungen im Lebenszyklus der Softwareentwicklung entsprechen. Für Workloads, die in sehr großem Umfang ausgeführt werden, benötigen Sie möglicherweise mehrere, Cluster Accounts um sicherzustellen, dass für alle Ihre Workloads genügend Kubernetes- und AWS-Servicekontingente verfügbar sind.



|Im obigen Diagramm wird AWS-RAM verwendet, um Subnetze von einem Cluster-Konto für ein Workload-Konto gemeinsam zu nutzen. Dann verwenden Workloads, die in EKS-Pods ausgeführt

werden, IRSA- oder EKS-Pod-Identitäten und Rollenverkettung, um eine Rolle in ihrem Workload-Konto zu übernehmen und auf ihre AWS-Ressourcen zuzugreifen.

Implementierung einer Multi-Workload-Kontostrategie für Multi-Tenant-Cluster

Teilen von Subnetzen mit AWS Resource Access Manager

Mit AWS Resource Access Manager (RAM) können Sie Ressourcen für mehrere AWS-Konten gemeinsam nutzen.

Wenn RAM für Ihre AWS-Organisation aktiviert ist, können Sie die VPC-Subnetze aus dem Cluster-Konto für Ihre Workload-Konten gemeinsam nutzen. Dadurch können AWS-Ressourcen, die Ihren Workload-Konten gehören, wie Amazon ElastiCache Clusters oder Amazon Relational Database Service (RDS) -Datenbanken, in derselben VPC wie Ihr EKS-Cluster bereitgestellt und von den Workloads genutzt werden, die auf Ihrem EKS-Cluster ausgeführt werden.

Um eine Ressource über RAM gemeinsam zu nutzen, öffnen Sie RAM in der AWS-Konsole des Cluster-Kontos und wählen Sie „Resource Shares“ und „Create Resource Share“ aus. Geben Sie Ihrem Resource Share einen Namen und wählen Sie die Subnetze aus, die Sie teilen möchten. Wählen Sie erneut Weiter aus und geben Sie das 12-stellige Konto IDs für die Workload-Konten ein, mit denen Sie die Subnetze teilen möchten. Wählen Sie erneut Weiter aus und klicken Sie auf Resource Share erstellen, um den Vorgang abzuschließen. Nach diesem Schritt kann das Workload-Konto Ressourcen in diesen Subnetzen bereitstellen.

RAM-Shares können auch programmgesteuert oder mit Infrastruktur als Code erstellt werden.

Wählen Sie zwischen EKS Pod Identities und IRSA

Auf der re:Invent 2023 hat AWS EKS Pod Identities als einfachere Möglichkeit eingeführt, temporäre AWS-Anmeldeinformationen für Ihre Pods auf EKS bereitzustellen. Sowohl IRSA- als auch EKS-Pod-Identitäten sind gültige Methoden für die Bereitstellung temporärer AWS-Anmeldeinformationen für Ihre EKS-Pods und werden weiterhin unterstützt. Sie sollten überlegen, welche Methode der Bereitstellung Ihren Anforderungen am besten entspricht.

Bei der Arbeit mit einem EKS-Cluster und mehreren AWS-Konten kann IRSA direkt Rollen in anderen AWS-Konten als dem Konto übernehmen, in dem der EKS-Cluster direkt gehostet wird, während für EKS-Pod-Identitäten die Konfiguration der Rollenverkettung erforderlich ist. Einen ausführlichen Vergleich finden Sie in der EKS-Dokumentation.

Zugreifen auf AWS-API-Ressourcen mit IAM-Rollen für Servicekonten

Mit IAM Roles for Service Accounts (IRSA) können Sie temporäre AWS-Anmeldeinformationen für Ihre Workloads bereitstellen, die auf EKS ausgeführt werden. IRSA kann verwendet werden, um temporäre Anmeldeinformationen für IAM-Rollen in den Workload-Konten vom Cluster-Konto abzurufen. Auf diese Weise können Ihre Workloads, die auf Ihren EKS-Clustern im Cluster-Konto ausgeführt werden, AWS-API-Ressourcen wie S3-Buckets, die im Workload-Konto gehostet werden, problemlos nutzen und die IAM-Authentifizierung für Ressourcen wie Amazon RDS-Datenbanken oder Amazon EFS verwenden. FileSystems

Auf AWS-API-Ressourcen und andere Ressourcen, die die IAM-Authentifizierung in einem Workload-Konto verwenden, kann nur über Anmeldeinformationen für IAM-Rollen in demselben Workload-Konto zugegriffen werden, es sei denn, der kontoübergreifende Zugriff ist möglich und wurde ausdrücklich aktiviert.

IRSA für kontoübergreifenden Zugriff aktivieren

Um IRSA für Workloads in Ihrem Cluster-Konto für den Zugriff auf Ressourcen in Ihren Workload-Konten zu aktivieren, müssen Sie zunächst einen IAM-OIDC-Identitätsanbieter in Ihrem Workload-Konto erstellen. Dies kann mit dem gleichen Verfahren für die Einrichtung von IRSA erfolgen, mit der Ausnahme, dass der Identity Provider im Workload-Konto erstellt wird.

Wenn Sie dann IRSA für Ihre Workloads auf EKS konfigurieren, können Sie dieselben Schritte wie in der Dokumentation befolgen, aber die 12-stellige Konto-ID des Workload-Kontos verwenden, wie im Abschnitt „Beispiel: Erstellen Sie einen Identitätsanbieter aus dem Cluster eines anderen Kontos“ beschrieben.

Nach der Konfiguration kann Ihre in EKS ausgeführte Anwendung ihr Dienstkonto direkt verwenden, um eine Rolle im Workload-Konto zu übernehmen und die darin enthaltenen Ressourcen zu nutzen.

Zugreifen auf AWS-API-Ressourcen mit EKS-Pod-Identitäten

EKS Pod Identities ist eine neue Methode zur Bereitstellung von AWS-Anmeldeinformationen für Ihre Workloads, die auf EKS ausgeführt werden. EKS-Pod-Identitäten vereinfachen die Konfiguration von AWS-Ressourcen, da Sie keine OIDC-Konfigurationen mehr verwalten müssen, um AWS-Anmeldeinformationen für Ihre Pods auf EKS bereitzustellen.

Aktivierung von EKS-Pod-Identitäten für kontoübergreifenden Zugriff

Im Gegensatz zu IRSA können EKS-Pod-Identitäten nur verwendet werden, um direkten Zugriff auf eine Rolle in demselben Konto wie der EKS-Cluster zu gewähren. Um auf eine Rolle in einem

anderen AWS-Konto zuzugreifen, müssen Pods, die EKS-Pod-Identitäten verwenden, eine Rollenverkettung durchführen.

Die Rollenverkettung kann in einem Anwendungsprofil mit der zugehörigen AWS-Konfigurationsdatei mithilfe des Process Credentials Provider konfiguriert werden, der in verschiedenen AWS SDKs verfügbar ist. `credential_process`kann bei der Konfiguration eines Profils als Quelle für Anmeldeinformationen verwendet werden, z. B.:

```
# 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
```

Die Quelle des von credentials al_process aufgerufenen Skripts:

```
#!/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}'
```

Sie können wie oben gezeigt eine AWS-Konfigurationsdatei mit den Rollen Konto A und B erstellen und die Variablen AWS_CONFIG_FILE und AWS_PROFILE env in Ihrer Pod-Spezifikation angeben. Der EKS Pod Identity-Webhook überschreibt nicht, ob die Umgebungsvariablen bereits in der Pod-Spezifikation vorhanden sind.

```
# 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
```

Bei der Konfiguration von Richtlinien zur Rollenvertrauensstellung für die Rollenverkettung mit EKS-Pod-Identitäten können Sie EKS-spezifische Attribute als Sitzungs-Tags referenzieren und mithilfe der attributebasierten Zugriffskontrolle (ABAC) den Zugriff auf Ihre IAM-Rollen auf bestimmte EKS-Pod-Identitätssitzungen beschränken, z. B. auf das Kubernetes-Dienstkonto, zu dem ein Pod gehört.

Bitte beachten Sie, dass einige dieser Attribute möglicherweise nicht allgemein eindeutig sind. Beispielsweise können zwei EKS-Cluster identische Namespaces haben, und ein Cluster kann identische Dienstkonten in allen Namespaces haben. Wenn Sie also Zugriff über EKS Pod Identities und ABAC gewähren, empfiehlt es sich, bei der Gewährung des Zugriffs auf ein Dienstkonto immer den Cluster-ARN und den Namespace zu berücksichtigen.

ABAC- und EKS-Pod-Identitäten für kontoübergreifenden Zugriff

Wenn Sie EKS Pod Identities verwenden, um im Rahmen einer Strategie für mehrere Konten Rollen (Rollenverkettung) in anderen Konten zu übernehmen, haben Sie die Möglichkeit, jedem Dienstkonto, das auf ein anderes Konto zugreifen muss, eine eigene IAM-Rolle zuzuweisen oder eine gemeinsame IAM-Rolle für mehrere Dienstkonten zu verwenden und mit ABAC zu steuern, auf welche Konten es zugreifen kann.

Um mit ABAC zu steuern, welche Dienstkonten eine Rolle in ein anderes Konto mit Rollenverkettung übernehmen können, erstellen Sie eine Rollenvertrauensrichtlinie, die es nur erlaubt, dass eine Rolle von einer Rollensitzung übernommen wird, wenn die erwarteten Werte vorliegen. Mit der folgenden Richtlinie zur Rollenvertrauensstellung kann eine Rolle aus dem EKS-Clusterkonto (Konto-ID 111122223333) nur dann eine Rolle übernehmen, wenn die `kubernetes-namespace` Tags`kubernetes-service-account`, `eks-cluster-arn` und alle den erwarteten Wert haben.

```
{
    "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",
```

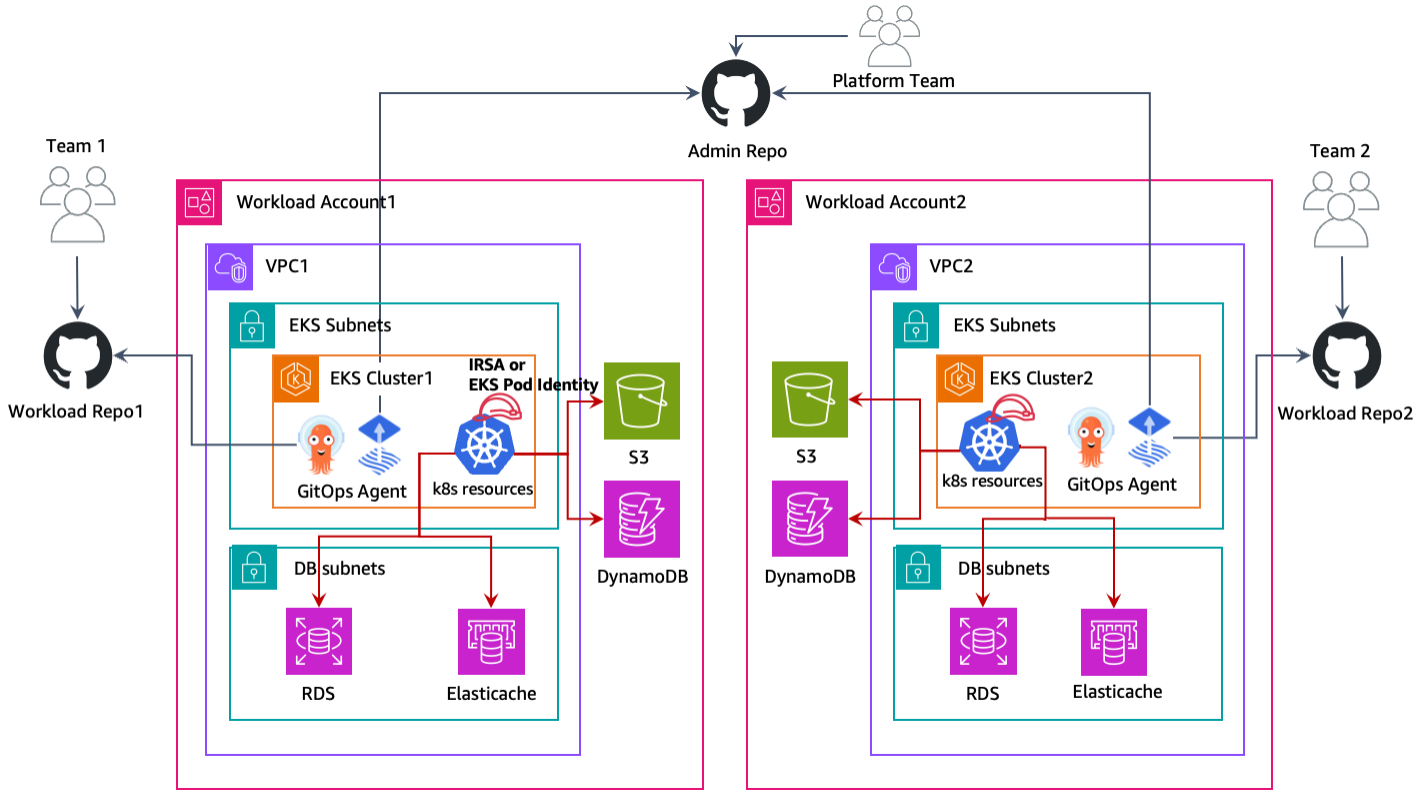
```
                    "aws:PrincipalTag/kubernetes-namespace": "PayrollNamespace"
                }
            }
        }
    ]
}
```

Bei der Verwendung dieser Strategie ist es eine bewährte Methode, sicherzustellen, dass die gemeinsame IAM-Rolle nur über `sts:AssumeRole` Berechtigungen und keinen anderen AWS-Zugriff verfügt.

Bei der Verwendung von ABAC ist es wichtig, dass Sie kontrollieren, wer IAM-Rollen und -Benutzer kennzeichnen darf, und zwar nur für diejenigen, die dies unbedingt benötigen. Jemand, der in der Lage ist, eine IAM-Rolle oder einen IAM-Benutzer zu taggen, könnte Tags auf roles/users identische Tags setzen, wie sie von EKS Pod Identities gesetzt würden, und möglicherweise seine Rechte erweitern. Mithilfe der IAM-Richtlinie oder der Service Control Policy (SCP) können Sie einschränken, wer Zugriff darauf hat, `eks-` Tags `kubernetes-` und Tags für IAM-Rollen und Benutzer festzulegen.

Dezentrale EKS-Cluster

Bei diesem Ansatz werden EKS-Cluster für die jeweiligen Workload-AWS-Konten bereitgestellt und zusammen mit anderen AWS-Ressourcen wie Amazon S3 S3-Buckets VPCs, Amazon DynamoDB-Tabellen usw. betrieben. Jedes Workload-Konto ist unabhängig, autark und wird vom jeweiligen Unit/Application teams. This model allows the creation of reusuable blueprints for various cluster capabilities — AI/ML Business-Cluster, Batch-Verarbeitung, Allzweck usw. betrieben — und verkauft die Cluster auf der Grundlage der Anforderungen des Anwendungsteams. Sowohl Anwendungs- als auch Plattformteams arbeiten von ihren jeweiligen GitOpsRepositorys aus, um die Bereitstellungen in den Workload-Clustern zu verwalten.



Im obigen Diagramm werden Amazon EKS-Cluster und andere AWS-Ressourcen für die jeweiligen Workload-Konten bereitgestellt. Dann verwenden Workloads, die in EKS-Pods ausgeführt werden, IRSA- oder EKS-Pod-Identitäten, um auf ihre AWS-Ressourcen zuzugreifen.

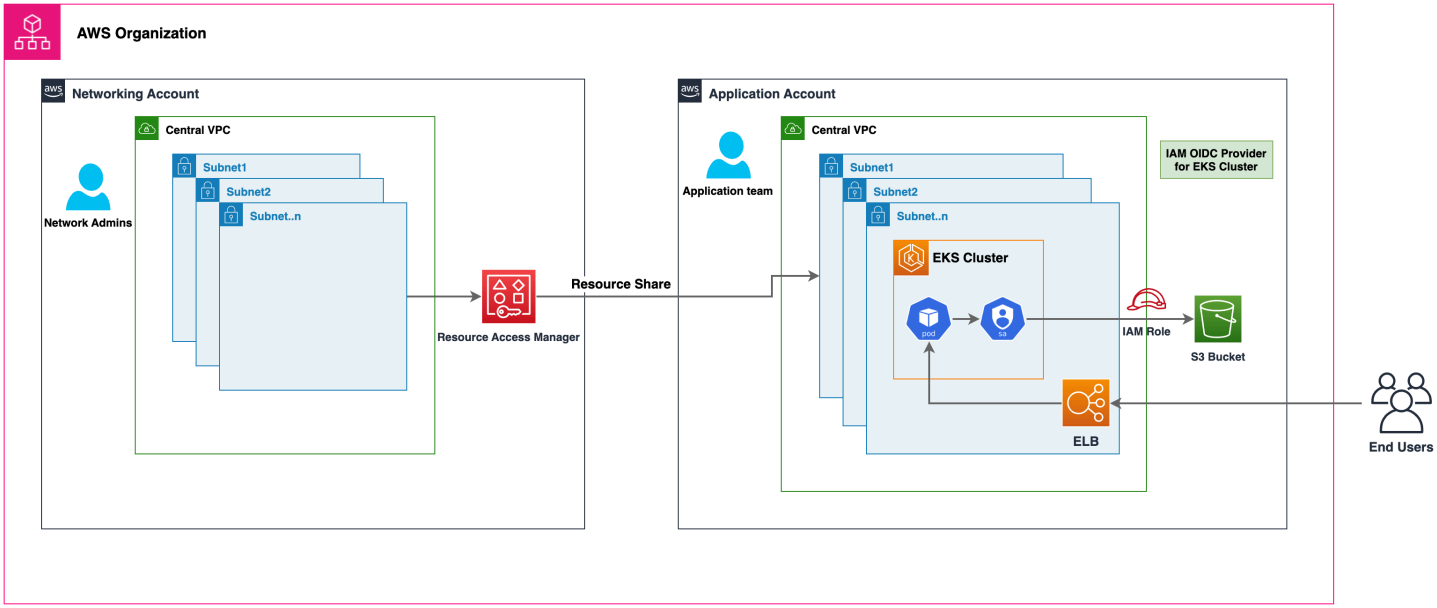
GitOps ist eine Methode zur Verwaltung der Anwendungs- und Infrastrukturbereitstellung, sodass das gesamte System deklarativ in einem Git-Repository beschrieben wird. Es handelt sich um ein Betriebsmodell, das Ihnen die Möglichkeit bietet, den Status mehrerer Kubernetes-Cluster mithilfe der bewährten Methoden der Versionskontrolle, unveränderlicher Artefakte und Automatisierung zu verwalten. In diesem Multi-Cluster-Modell ist jeder Workload-Cluster mit mehreren Git-Repos ausgestattet, sodass jedes Team (Anwendung, Plattform, Sicherheit usw.) seine jeweiligen Änderungen auf dem Cluster implementieren kann.

Sie würden in jedem Konto IAM-Rollen für Service Accounts (IRSA) oder EKS-Pod-Identitäten verwenden, damit Ihre EKS-Workloads temporäre AWS-Anmeldeinformationen für den sicheren Zugriff auf andere AWS-Ressourcen erhalten können. IAM-Rollen werden in den jeweiligen Workload-AWS-Konten erstellt und ihnen k8s-Servicekonten zugeordnet, um temporären IAM-Zugriff zu ermöglichen. Bei diesem Ansatz ist also kein kontoübergreifender Zugriff erforderlich. Informationen zur Einrichtung der einzelnen Workloads für IRSA finden Sie in der Dokumentation zu

IAM-Rollen für Dienstkonten und in der Dokumentation zu EKS Pod-Identitäten, wie Sie EKS-Pod-Identitäten in jedem Konto einrichten.

Zentralisiertes Netzwerk

Sie können AWS-RAM auch verwenden, um die VPC-Subnetze für Workload-Konten gemeinsam zu nutzen und Amazon EKS-Cluster und andere AWS-Ressourcen darin zu starten. Dies ermöglicht eine zentrale Netzwerkverwaltung/-verwaltung, eine vereinfachte Netzwerkkonnektivität und dezentrale EKS-Cluster. In diesem AWS-Blog finden Sie eine detaillierte Anleitung und Überlegungen zu diesem Ansatz.



Im obigen Diagramm wird AWS-RAM verwendet, um Subnetze von einem zentralen Netzwerkkonto für ein Workload-Konto gemeinsam zu nutzen. Dann werden EKS-Cluster und andere AWS-Ressourcen in diesen Subnetzen in den jeweiligen Workload-Konten gestartet. EKS-Pods verwenden IRSA- oder EKS-Pod-Identitäten, um auf ihre AWS-Ressourcen zuzugreifen.

Zentralisierte und dezentrale EKS-Cluster

Die Entscheidung, mit einem zentralisierten oder dezentralen System zu arbeiten, hängt von Ihren Anforderungen ab. Diese Tabelle zeigt die wichtigsten Unterschiede zwischen den einzelnen Strategien.

#	Zentralisierter EKS-Cluster	Dezentralisierte EKS-Cluster
Clusterverwaltung:	Die Verwaltung eines einzelnen EKS-Clusters ist einfacher als die Verwaltung mehrerer Cluster	Eine effiziente Automatisierung der Clusterverwaltung ist erforderlich, um den betrieblichen Aufwand für die Verwaltung mehrerer EKS-Cluster zu reduzieren
Kosteneffizienz:	Ermöglicht die Wiederverwendung von EKS-Cluster- und Netzwerkressourcen, was die Kosteneffizienz fördert	Erfordert Netzwerk- und Cluster-Setups pro Workload, was zusätzliche Ressourcen erfordert
Belastbarkeit:	Mehrere Workloads auf dem zentralisierten Cluster können beeinträchtigt werden, wenn ein Cluster beeinträchtigt wird	Wenn ein Cluster beeinträchtigt wird, beschränkt sich der Schaden nur auf die Workloads, die auf diesem Cluster ausgeführt werden. Alle anderen Workloads sind davon nicht betroffen
Isolierung und Sicherheit:	Isolation/Soft-Mehrmandantenfähigkeit wird durch die Verwendung nativer k8s-Konstrukte wie erreicht. `Namespaces` Workloads können die zugrunde liegenden Ressourcen wie CPU, Arbeitsspeicher usw. gemeinsam nutzen. AWS-Ressourcen sind in ihren eigenen Workload-Konten isoliert, auf die standardmäßig von anderen AWS-Konten aus nicht zugegriffen werden kann.	Stärkere Isolierung der Rechenressourcen, da die Workloads in einzelnen Clustern und Knoten ausgeführt werden, die sich keine Ressourcen teilen. AWS-Ressourcen sind in ihren eigenen Workload-Konten isoliert, auf die standardmäßig von anderen AWS-Konten aus nicht zugegriffen werden kann.

#	Zentralisierter EKS-Cluster	Dezentralisierte EKS-Cluster
Leistung und Skalierbarkeit:	Wenn Workloads sehr groß werden, kann es sein, dass Kubernetes- und AWS-Servicequotas im Cluster-K onto auftreten. Sie können zusätzliche Cluster-Konten bereitstellen, um noch weiter zu skalieren	Je mehr Cluster vorhanden VPCs sind, desto mehr verfügbare K8s und AWS-Servicekontingenten stehen für jeden Workload zur Verfügung
Netzwerkbetrieb:	Pro Cluster wird eine einzige VPC verwendet, was eine einfachere Konnektivität für Anwendungen auf diesem Cluster ermöglicht	Das Routing muss zwischen dem dezentralen EKS-Cluster eingerichtet werden VPCs
Kubernetes-Zugriffsverwaltung:	Sie müssen viele verschied ene Rollen und Benutzer im Cluster verwalten, um allen Workload-Teams Zugriff zu gewähren und sicherzustellen, dass die Kubernetes-Ressour cen ordnungsgemäß getrennt sind	Vereinfachtes Zugriffsm anagement, da jeder Cluster einer Workload/einem Team zugewiesen ist
AWS-Zugriffsmanagement:	AWS-Ressourcen werden in einem eigenen Konto bereitgestellt, auf das standardmäßig nur mit IAM-Rollen im Workload-Konto zugegriffen werden kann. IAM-Rollen in den Workload-Konten werden kontenübe rgreifend entweder mit IRSA- oder EKS-Pod-Identitäten übernommen.	AWS-Ressourcen werden in einem eigenen Konto bereitgestellt, auf das standardmäßig nur mit IAM-Rollen im Workload-Konto zugegriffen werden kann. IAM-Rollen in den Workload-Konten werden direkt an Pods mit IRSA- oder EKS-Pod-I dentitäten bereitgestellt

Verwaltung des Cluster-Zugriffs

Effektives Zugriffsmanagement ist entscheidend für die Aufrechterhaltung der Sicherheit und Integrität Ihrer Amazon EKS-Cluster. In diesem Handbuch werden verschiedene Optionen für das EKS-Zugriffsmanagement untersucht, wobei der Schwerpunkt auf der Verwendung von AWS IAM Identity Center (früher AWS SSO) liegt. Wir werden verschiedene Ansätze vergleichen, ihre Kompromisse erörtern und bekannte Einschränkungen und Überlegungen hervorheben.

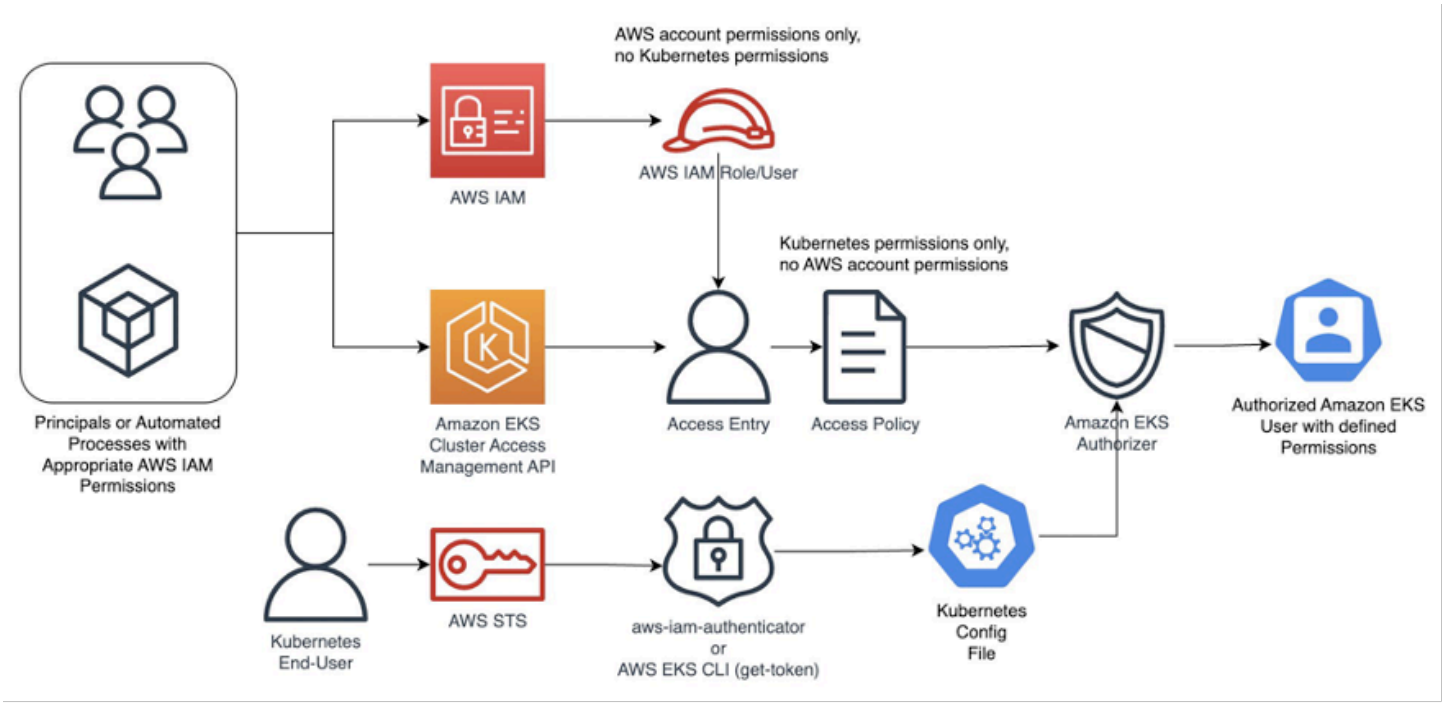
EKS-Zugriffsverwaltungsoptionen

> **Note**
>
> ConfigMapDie basierte Zugriffsverwaltung (aws-auth ConfigMap) ist veraltet und wurde durch die Cluster Access Management (CAM) -API ersetzt. Implementieren Sie für neue EKS-Cluster die CAM-API zur Verwaltung des Clusterzugriffs. Migrieren Sie für bestehende Cluster, die aws-auth verwenden ConfigMap, auf die CAM-API.

Option 1: AWS IAM Identity Center mit Cluster Access Management (CAM) -API

- Zentralisierte Benutzer- und Berechtigungsverwaltung
- Integration mit bestehenden Identitätsanbietern (z. B. Microsoft AD, Okta PingId und mehr)
- Die CAM-API verwendet Access Entries, um AWS IAM-Prinzipale (Benutzer oder Rollen) mit dem EKS-Cluster zu verknüpfen. Diese Einträge funktionieren mit den verwalteten Identitäten von IAM Identity Center und ermöglichen es Administratoren, den Clusterzugriff für in Identity Center definierte Benutzer und Gruppen zu kontrollieren.

Ablauf der EKS-Cluster-Authentifizierung:



1. Principals (menschliche Benutzer) oder automatisierte Prozesse authentifizieren sich über AWS IAM, indem sie die entsprechenden AWS-Kontoberechtigungen vorlegen. In diesem Schritt werden sie dem entsprechenden AWS IAM-Prinzipal (Rolle oder Benutzer) zugeordnet.
2. Als Nächstes ordnet ein EKS-Zugriffseintrag diesen IAM-Prinzipal einem Kubernetes-RBAC-Prinzipal (Benutzer oder Gruppe) zu, indem eine entsprechende Zugriffsrichtlinie definiert wird, die nur Kubernetes-Berechtigungen enthält.
3. Wenn ein Kubernetes-Endbenutzer versucht, auf einen Cluster zuzugreifen, wird seine Authentifizierungsanfrage von aws-iam-authenticator oder der AWS EKS-CLI verarbeitet und anhand des Clusterkontextes in der kubeconfig-Datei validiert.
4. Schließlich überprüft der EKS-Autorisierer die mit dem Zugriffseintrag des authentifizierten Benutzers verbundenen Berechtigungen und gewährt oder verweigert den Zugriff entsprechend.
    - Die API verwendet Amazon EKS-spezifische Zugriffsrichtlinien, um die Autorisierungsebene für jeden Access-Eintrag zu definieren. Diese Richtlinien können Rollen und Berechtigungen zugeordnet werden, die in IAM Identity Center eingerichtet wurden, wodurch eine konsistente Zugriffskontrolle für alle AWS-Services und EKS-Cluster gewährleistet wird.

Vorteile gegenüber einer ConfigMap basierten Zugriffsverwaltung:

1. Geringeres Risiko von Fehlkonfigurationen: Direktes API-basiertes Management verhindert häufige Fehler im Zusammenhang mit manueller Bearbeitung. ConfigMap Dies trägt dazu bei, versehentliche Löschungen oder Syntaxfehler zu verhindern, die Benutzer vom Cluster ausschließen könnten.
2. Verbessertes Prinzip der geringsten Rechte: Macht die Cluster-Admin-Rechte für die Identität des Clustererstellers überflüssig und ermöglicht eine detailliertere und angemessenere Zuweisung von Berechtigungen. Sie können sich dafür entscheiden, diese Berechtigung für Break-Glass-Anwendungsfälle hinzuzufügen.
3. Verbessertes Sicherheitsmodell: Bietet eine integrierte Überprüfung von Zugriffseinträgen, bevor sie angewendet werden. Darüber hinaus bietet es eine engere Integration mit AWS IAM für die Authentifizierung.
4. Optimierter Betrieb: Bietet eine intuitivere Möglichkeit, Berechtigungen mithilfe von AWS-nativen Tools zu verwalten.

Bewährte Methoden:

1. Verwenden Sie AWS Organizations, um mehrere Konten zu verwalten und Richtlinien zur Servicekontrolle anzuwenden (SCPs).
2. Implementieren Sie das Prinzip der geringsten Rechte, indem Sie spezifische Berechtigungssätze für verschiedene EKS-Rollen (z. B. Administrator, Entwickler, schreibgeschützt) erstellen.
3. Verwenden Sie die attributebasierte Zugriffskontrolle (ABAC), um Pods basierend auf Benutzerattributen dynamisch Berechtigungen zuzuweisen.
4. Prüfen und überprüfen Sie die Zugriffsberechtigungen regelmäßig.

Überlegungen/Einschränkungen:

1. Von Identity Center ARNs generierte Rollen haben zufällige Suffixe, was ihre Verwendung in statischen Konfigurationen erschwert.
2. Eingeschränkte Unterstützung für detaillierte Berechtigungen auf Kubernetes-Ressourcenebene. Für benutzerdefinierte Kubernetes-RBAC-Rollen ist eine zusätzliche Konfiguration erforderlich. Erwägen Sie, neben Kubernetes-nativem RBAC auch Kyverno für die erweiterte Rechteverwaltung in EKS-Clustern zu verwenden.

Option 2: AWS IAM ist Users/Roles Kubernetes-Gruppen zugeordnet

Vorteile:

1. Präzise Kontrolle über IAM-Berechtigungen.
2. Vorhersehbare und statische Rolle ARNs

Nachteile:

1. Erhöhter Verwaltungsaufwand für Benutzerkonten
2. Mangel an zentralisiertem Identitätsmanagement
3. Potenzielle Verbreitung von IAM-Entitäten

Bewährte Verfahren:

1. Verwenden Sie IAM-Rollen anstelle von IAM-Benutzern, um die Sicherheit und Verwaltbarkeit zu verbessern
2. Implementieren Sie eine Namenskonvention für Rollen, um Konsistenz und einfache Verwaltung zu gewährleisten
3. Verwenden Sie die IAM-Richtlinienbedingungen, um den Zugriff anhand von Tags oder anderen Attributen einzuschränken.
4. Wechseln Sie die Zugriffsschlüssel regelmäßig und überprüfen Sie die Berechtigungen.

Überlegungen/Einschränkungen:

1. Probleme mit der Skalierbarkeit bei der Verwaltung einer großen Anzahl von Benutzern oder Rollen
2. Keine integrierten Single-Sign-On-Funktionen

Option 3: OIDC-Anbieter

Vorteile:

1. Integration mit bestehenden Identitätsmanagementsystemen

2. Reduzierter Verwaltungsaufwand für Benutzerkonten

Nachteile:

1. Zusätzliche Komplexität der Konfiguration
2. Potenziell erhöhte Latenz bei der Authentifizierung
3. Abhängigkeit von einem externen Identitätsanbieter

Bewährte Verfahren:

1. Konfigurieren Sie den OIDC-Anbieter sorgfältig, um eine sichere Token-Validierung zu gewährleisten.
2. Verwenden Sie kurzlebige Token und implementieren Sie Mechanismen zur Token-Aktualisierung.
3. Prüfen und aktualisieren Sie regelmäßig die OIDC-Konfigurationen.

In diesem Leitfaden finden Sie eine Referenzimplementierung für die Integration externer Single Sign-On-Anbieter mit Amazon EKS

Überlegungen/Einschränkungen:

1. Eingeschränkte native Integration mit AWS-Services im Vergleich zu IAM.
2. Die Aussteller-URL des OIDC-Anbieters muss öffentlich zugänglich sein, damit EKS die Signaturschlüssel ermitteln kann.

AWS EKS Pod Identity im Vergleich zu IRSA für Workloads

Amazon EKS bietet zwei Möglichkeiten, AWS IAM-Berechtigungen für Workloads zu gewähren, die in Amazon EKS-Clustern ausgeführt werden: IAM-Rollen für Dienstkonten (IRSA) und EKS-Pod-Identitäten.

Sowohl IRSA- als auch EKS-Pod-Identitäten bieten die Vorteile des Zugriffs mit den geringsten Rechten, der Isolierung von Anmeldeinformationen und der Überprüfbarkeit, aber EKS Pod Identity ist die empfohlene Methode, um Berechtigungen für Workloads zu erteilen.

Ausführliche Anleitungen zu Identität und Anmeldeinformationen für EKS-Pods finden Sie im Abschnitt Identitäten und Anmeldeinformationen unter Bewährte Sicherheitsmethoden.

Empfehlung

Kombinieren Sie IAM Identity Center mit der CAM-API

- Vereinfachtes Management: Durch die Verwendung der Cluster Access Management API in Verbindung mit IAM Identity Center können Administratoren den EKS-Clusterzugriff zusammen mit anderen AWS-Services verwalten, sodass weniger zwischen verschiedenen Schnittstellen gewechselt oder ConfigMaps manuell bearbeitet werden muss.
- Verwenden Sie Zugriffseinträge, um die Kubernetes-Berechtigungen von IAM-Prinzipalen außerhalb des Clusters zu verwalten. Sie können den Zugriff auf den Cluster mithilfe der EKS-API, der AWS-Befehlszeilenschnittstelle, AWS SDKs CloudFormation, AWS und der AWS-Managementkonsole hinzufügen und verwalten. Für die Benutzerverwaltung können also die gleichen Tools verwendet werden wie für die Clustererstellung.
- Granulare Kubernetes-Berechtigungen können angewendet werden, indem Kubernetes-Benutzern oder -Gruppen IAM-Prinzipalen zugeordnet werden, die mit SSO-Identitäten über Zugriffseinträge und Zugriffsrichtlinien verknüpft sind.
- Folgen Sie zunächst den Anweisungen Authentifizierungsmodus zur Verwendung von Zugriffseinträgen ändern und anschließend Vorhandene aws-auth-Einträge zu Zugriffseinträgen migrieren. ConfigMap

Bewährte Methoden für Cluster-Autoscaling

Dieses Handbuch enthält Hinweise zur Cluster-Autoskalierung, einschließlich Anleitungen für Auto Mode, Karpenter und Kubernetes Cluster Autoscaler.

Themen

- EKS Auto Mode
- Karpenter
- Cluster Autoscaler

EKS Auto Mode

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Amazon EKS Auto Mode stellt eine bedeutende Weiterentwicklung im Kubernetes-Infrastrukturmanagement dar und kombiniert eine sichere und skalierbare Cluster-Infrastruktur mit integrierten Kubernetes-Funktionen, die von AWS verwaltet werden. Der Service bietet vollständig verwaltete Worker-Node-Operationen, sodass Kunden keine verwalteten Knotengruppen oder -gruppen einrichten müssen. AutoScaling

Der entscheidende architektonische Unterschied besteht darin, dass EKS Auto Mode ein auf Karpenter basierendes System verwendet, das automatisch EC2-Instances als Antwort auf Pod-Anfragen bereitstellt. Diese Instances laufen auf Bottlerocket AMIs mit vorinstallierten Add-Ons wie EBS-CSI-Treibern, sodass die Infrastruktur wirklich von AWS verwaltet wird. Im Gegensatz zu herkömmlichen Skalierungsmethoden:

- Herkömmlicher Cluster Autoscaler (CAS) erfordert manuelles Knotengruppenmanagement und kann nur Knoten mit einem einzigen Instanztyp pro Knotengruppe erstellen
- Das selbstverwaltete Karpenter bietet mehr Flexibilität, da es mit der EC2 Fleet API arbeitet. Es kann verschiedene Instance-Typen bereitstellen, erfordert jedoch eine Kundenverwaltung
- Der automatische Modus von EKS wickelt alle Skalierungsvorgänge automatisch über verwaltete und NodePools NodeClasses

Das neue System bietet mehrere betriebliche Verbesserungen:

- Automatische pod-gesteuerte Skalierung ohne manuelle Konfiguration der Knotengruppen
- Integrierte verwaltete Load Balancer-Controller, die automatisch auf der ALB/NLB Grundlage von Ingress-Ressourcen Daten erstellen
- Integrierte Sicherheitsfunktionen mit vorkonfigurierter Pod-Identität
- Maximale Knotenlaufzeit von 21 Tagen mit automatischem Austausch

Aus Kostengründen behält EKS Auto Mode die EC2-Standardpreise bei, wobei jedoch nur für Knoten, die im Auto Mode verwaltet werden, eine Verwaltungsgebühr erhoben wird. Wichtig ist, dass Kunden weiterhin verwaltete Knoten im automatischen Modus mit selbstverwalteten Knoten im selben Cluster kombinieren können.

Während AWS die meisten betrieblichen Aspekte abwickelt, behalten die Kunden die Verantwortung für die Cluster-Versionsverwaltung und können kontrollierte Upgrades durchführen, die fortlaufende Updates der Worker-Knoten auslösen.

Gründe für die Verwendung des automatischen Modus

Auto Mode richtet sich an Benutzer, die die Vorteile von Kubernetes und EKS nutzen möchten, aber die Betriebsbelastung im Zusammenhang mit Kubernetes wie Upgrades und installation/maintenance kritischer Plattformkomponenten wie auto-scaling, Lastenausgleich und Speicher minimieren müssen. Mit dem Auto Mode geht EKS noch einen Schritt weiter bei der Minimierung der undifferenzierten Schwerarbeit, die mit der Wartung von Kubernetes einhergeht

Häufig gestellte Fragen

Was ist der Unterschied zwischen EKS Auto Mode und Open Source Karpenter?

EKS Auto Mode ist eine große Suite von Funktionen, die den Betrieb von Kubernetes in Produktionsqualität einfach machen. Eine dieser Funktionen sind die Vorteile der auto-scaling von Karpenter, die vollständig verwaltet werden. Aus betrieblicher Sicht besteht der einzige Unterschied darin, dass Sie im automatischen EKS-Modus die Bereitstellung, Skalierung und Aktualisierung der Karpenter Pods nicht selbst verwalten müssen. Alle anderen Operationen, z. B. verwaltet NodeClasses und NodePools funktionieren genauso wie bei der Open-Source-Version von Karpenter.

Kann ich verwaltete Knotengruppen zusammen mit Knoten ausführen, die im automatischen Modus verwaltet werden?

Ja, Sie können statische Knoten über verwaltete Knotengruppen zusammen mit Ihren Autoscaling-Knoten, die mit dem automatischen Modus ausgestattet sind, ausführen

Kann ich einen Cluster vom Standard-EKS in den EKS-Automatikmodus migrieren?

Ja, Anweisungen zur Aktivierung des automatischen EKS-Modus auf einem vorhandenen Cluster finden Sie in der offiziellen AWS-Dokumentation

Dinge, die Sie beachten sollten: 1. Nachdem Sie den automatischen Modus aktiviert haben, sollten Sie alle Komponenten deinstallieren, die Sie installiert haben und die jetzt vom Auto Mode verwaltet werden, wie Karpenter oder den AWS Load Balancer Controller 2. Sie müssen sicherstellen, dass Ihre installierten Add-Ons installiert sind. up-to-date Siehe Dokumentation.

Wie konfiguriere ich NodePools im EKS-Automatikmodus?

Ein neuer Cluster wird mit zwei vorkonfigurierten Clustern geliefert NodePools

für allgemeine Zwecke

```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
```

Dadurch wird Karpenter NodePool angewiesen, Knoten mit den folgenden Eigenschaften zu starten:

1. Kapazitätstyp „Auf Abruf“
2. Instanztypen von C, M oder R
3. Instanzgenerierung von 4
4. AMD-Architektur
5. Linux-Betriebssystem

Sie definiert auch die Logik des Herunterskalierens, indem sie festlegt, dass sich nur 10% aller Knoten zu einem bestimmten Zeitpunkt in einem unterbrochenen Zustand befinden dürfen und dass eine Konsolidierung nur erfolgen sollte, wenn die Knoten leer oder nicht ausgelastet sind.

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
```

NodePool Es ähnelt dem „Allzweckmodell“, mit Ausnahme der folgenden Unterschiede:

1. Es ermöglicht Knoten sowohl mit der ARM-Architektur als auch mit der AMD-Architektur
2. Es versieht diese Knoten mit einem, NoSchedule sofern es keine Toleranz für „“ CriticalAddonsOnly gibt. Dies ist für den internen Gebrauch durch EKS-Add-Ons

benutzerdefiniert

Sie können NodePools je nach Ihren Bedürfnissen Ihr eigenes benutzerdefiniertes erstellen. Um mehr darüber zu erfahren, schauen NodePools Sie bitte in der Karpenter Dokumentation nach.

Kann ich das vom Auto Mode verwendete AMI anpassen, wenn neue Knoten gestartet werden?

Nein, derzeit AMIs werden nur von Amazon bereitgestellte Bottlerocket unterstützt

Wie kann ich benutzerdefinierte Tools oder Agenten auf meinen Kubernetes-Hosts installieren?

Da die AMI-Anpassung nicht unterstützt wird, sollten Sie den Workload als Kubernetes bereitstellen, wenn Sie Software auf Host-Ebene für Dinge wie Sicherheitsscans benötigen. DaemonSet

Welche Komponenten werden in meiner Cluster-Datenebene ausgeführt, wenn ich einen neuen EKS-Auto-Mode-Cluster bereitstelle?

Wenn Sie einen Cluster mit eksctl oder der AWS-Konsole erstellen, sind die einzigen Pods, die in einem EKS-Auto-Mode-Cluster ausgeführt werden, Kubernetes Metrics Server-Pods. Die anderen Komponenten von EKS Auto Mode wie Karpenter, der AWS Load Balancer Controller und der EBS CSI-Treiber werden alle außerhalb des Clusters ausgeführt und verwaltet.

Welche verwalteten Komponenten werden ausgeführt, um meinen neuen EKS-Auto-Mode-Cluster zu unterstützen?

Der EKS Auto Mode automatisiert die Bereitstellung der meisten Teile einer Datenebene, die für Kubernetes in Produktionsqualität benötigt werden, vollständig. Dies umfasst:

- Karpenter, für die auto-scaling der Rechenleistung Ihres Clusters
- AWS Load Balancer Controller, mit dem Sie Kubernetes-Services einfach über die automatisierte Elastic Load Balancer Balancer-Integration bereitstellen können

- EBS CSI
- VPC CNI
- EKS Pod Identity Agent

Wie behebe ich Fehler bei den Komponenten des automatischen Modus, die früher als Pods in meinem Cluster ausgeführt wurden?

Mit EKS Auto Mode werden viele Komponenten wie der AWS Load Balancer Controller und Karpenter für Sie außerhalb Ihres Clusters verwaltet, sodass Sie nicht den gleichen Einblick in die Protokolle haben, den Sie von der Selbstverwaltung gewohnt sind. Wenn Sie sich in einer Situation befinden, in der Sie Probleme mit der Funktionalität einer Funktion im automatischen Modus beheben müssen, erstellen Sie ein AWS-Support-Ticket.

Karpenter

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Karpenter ist ein Open-Source-Projekt zur Verbesserung des Knotenlebenszyklusmanagements innerhalb von Kubernetes-Clustern. Es automatisiert die Bereitstellung und Deprovisionierung von Knoten auf der Grundlage der spezifischen Planungsanforderungen von Pods und ermöglicht so eine effiziente Skalierung und Kostenoptimierung. Ihre Hauptfunktionen sind:

- Überwachen Sie Pods, die der Kubernetes-Scheduler aufgrund von Ressourcenbeschränkungen nicht planen kann.
- Evaluieren Sie die Planungsanforderungen (Ressourcenanfragen, Knotenselektoren, Affinitäten, Toleranzen usw.) der Pods, die nicht planbar sind.
- Stellen Sie neue Knoten bereit, die die Anforderungen dieser Pods erfüllen.
- Entfernen Sie Knoten, wenn sie nicht mehr benötigt werden.

Mit Karpenter können Sie Einschränkungen für die Knotenbereitstellung wie Taints, Labels, Anforderungen (Instanztypen, Zonen usw.) und Beschränkungen für die Gesamtzahl der bereitgestellten Ressourcen definieren NodePools . Bei der Bereitstellung von Workloads können

Sie in den Pod-Spezifikationen verschiedene Planungsbeschränkungen wie requests/limits, node selectors, node/pod Ressourcenaffinitäten, Toleranzen und Einschränkungen der Topologieverteilung angeben. Karpenter stellt dann auf der Grundlage dieser Spezifikationen Knoten in der richtigen Größe bereit.

Gründe für die Nutzung von Karpenter

Vor der Einführung von Karpenter verließen sich Kubernetes-Benutzer hauptsächlich auf Amazon EC2 Auto Scaling Scaling-Gruppen und den Kubernetes Cluster Autoscaler (CAS), um die Rechenkapazität ihrer Cluster dynamisch anzupassen. Mit Karpenter müssen Sie nicht Dutzende von Knotengruppen erstellen, um die Flexibilität und Vielfalt zu erreichen, die Sie mit Karpenter erhalten. Im Gegensatz zu CAS ist Karpenter nicht so eng mit Kubernetes-Versionen verbunden und erfordert nicht, dass Sie zwischen AWS und Kubernetes wechseln. APIs

Karpenter konsolidiert die Zuständigkeiten für die Instanzorchestrierung in einem einzigen System, das einfacher, stabiler und clusterfähiger ist. Karpenter wurde entwickelt, um einige der Herausforderungen zu bewältigen, die Cluster Autoscaler mit sich bringt. Es bietet vereinfachte Möglichkeiten für:

- Stellen Sie Knoten auf der Grundlage der Workload-Anforderungen bereit.
- Erstellen Sie mithilfe flexibler NodePool Optionen unterschiedliche Knotenkonfigurationen nach Instanztyp. Anstatt viele spezifische benutzerdefinierte Knotengruppen zu verwalten, können Sie mit Karpenter verschiedene Workload-Kapazitäten mit einer einzigen, flexiblen Lösung verwalten. NodePool
- Erzielen Sie eine verbesserte Pod-Planung im großen Maßstab, indem Sie Knoten schnell starten und Pods planen.

Informationen und Dokumentation zur Verwendung von Karpenter finden Sie auf der Website karpenter.sh.

Empfehlungen

Die bewährten Methoden sind in Abschnitte zu Karpenter selbst und zur Planung von Pods NodePools unterteilt.

Bewährte Methoden von Karpenter

Die folgenden bewährten Methoden behandeln Themen, die sich auf Karpenter selbst beziehen.

Lockdown AMIs in Produktionsclustern

Wir empfehlen dringend, dass Sie bekannte Amazon Machine Images (AMIs), die von Karpenter für Produktionscluster verwendet werden, anheften. Wenn Sie einen Aliasnamen verwenden`amiSelector`, der auf gesetzt ist`@latest`, oder eine andere Methode verwenden, die dazu führt, dass die Bereitstellung nicht getestet wird, AMIs sobald sie veröffentlicht werden, besteht das Risiko von Workload-Ausfällen und Ausfallzeiten in Ihren Produktionsclustern. Daher empfehlen wir dringend, getestete Arbeitsversionen von AMIs für Ihre Produktionscluster zu verwenden, während Sie neuere Versionen in Clustern testen, die nicht zur Produktion gehören. Sie könnten in Ihrem beispielsweise NodeClass wie folgt einen Alias einrichten:

```
amiSelectorTerms
  - alias: al2023@v20240807
```

Informationen zur Verwaltung und Fixierung AMIs in Karpenter finden Sie in der Karpenter-Dokumentation unter Verwalten AMIs.

Verwenden Sie Karpenter für Workloads mit wechselnden Kapazitätsanforderungen

Karpenter bringt das Skalierungsmanagement näher an die native Version von Kubernetes heran APIs als Autoscaling Groups (ASGs) und Managed Node Groups (MNGs). ASGs und MNGs sind AWS-native Abstraktionen, bei denen die Skalierung auf AWS-Ebene, wie z. B. der EC2-CPU-Last, ausgelöst wird. Cluster Autoscaler verbindet die Kubernetes-Abstraktionen mit AWS-Abstraktionen, verliert dadurch jedoch an Flexibilität, z. B. bei der Planung für eine bestimmte Availability Zone.

Karpenter entfernt eine AWS-Abstraktionsebene, um einen Teil der Flexibilität direkt in Kubernetes zu bringen. Karpenter eignet sich am besten für Cluster mit Workloads, die auf Zeiten hoher, hoher Nachfrage stoßen oder unterschiedliche Rechenanforderungen haben. MNGs und ASGs eignen sich gut für Cluster, auf denen Workloads ausgeführt werden, die tendenziell statischer und konsistenter sind. Sie können je nach Ihren Anforderungen eine Mischung aus dynamisch und statisch verwalteten Knoten verwenden.

Ziehen Sie andere Autoscaling-Projekte in Betracht, wenn...

Sie benötigen Funktionen, die in Karpenter noch entwickelt werden. Da Karpenter ein relativ neues Projekt ist, sollten Sie vorerst andere Autoscaling-Projekte in Betracht ziehen, wenn Sie Funktionen benötigen, die noch nicht Teil von Karpenter sind.

Führen Sie den Karpenter-Controller auf EKS Fargate oder auf einem Worker-Knoten aus, der zu einer Knotengruppe gehört

Karpenter wird mithilfe eines Helm-Charts installiert. Das Helm-Diagramm installiert den Karpenter-Controller und einen Webhook-Pod als Deployment, das ausgeführt werden muss, bevor der Controller für die Skalierung Ihres Clusters verwendet werden kann. Wir empfehlen mindestens eine kleine Knotengruppe mit mindestens einem Worker-Knoten. Als Alternative können Sie diese Pods auf EKS Fargate ausführen, indem Sie ein Fargate-Profil für den `karpenter` Namespace erstellen. Dadurch werden alle in diesem Namespace bereitgestellten Pods auf EKS Fargate ausgeführt. Führen Sie Karpenter nicht auf einem Knoten aus, der von Karpenter verwaltet wird.

Karpenter unterstützt keine benutzerdefinierten Startvorlagen

In Version 1 gibt es keine Unterstützung für benutzerdefinierte Startvorlagen. APIs Sie können benutzerdefinierte Benutzerdaten and/or direkt verwenden, indem Sie benutzerdefinierte Daten AMIs in der angeben EC2NodeClass. Weitere Informationen dazu finden Sie unter NodeClasses.

Schließen Sie Instance-Typen aus, die nicht zu Ihrer Arbeitslast passen

Erwägen Sie, bestimmte Instanztypen mit dem `node.kubernetes.io/instance-type` Schlüssel auszuschließen, wenn sie für Workloads, die in Ihrem Cluster ausgeführt werden, nicht benötigt werden.

Das folgende Beispiel zeigt, wie die Bereitstellung großer Graviton-Instanzen vermieden werden kann.

```
- key: node.kubernetes.io/instance-type
  operator: NotIn
  values:
  - m6g.16xlarge
  - m6gd.16xlarge
  - r6g.16xlarge
  - r6gd.16xlarge
  - c6g.16xlarge
```

Aktivieren Sie die Behandlung von Unterbrechungen, wenn Sie Spot verwenden

Karpenter unterstützt die systemeigene Behandlung von Unterbrechungen und kann unfreiwillige Unterbrechungsereignisse wie Spot-Instance-Unterbrechungen, geplante Wartungsereignisse und Ereignisse zum Beenden/Stoppen von Instanzen verarbeiten, die Ihre Workloads stören

könnten. Wenn Karpenter solche Ereignisse für Knoten erkennt, werden die betroffenen Knoten automatisch verdorben, entleert und im Voraus beendet, um mit der ordnungsgemäßen Bereinigung der Workloads zu beginnen, bevor sie unterbrochen werden. Bei Spot-Unterbrechungen mit einer Frist von 2 Minuten startet Karpenter schnell einen neuen Node, sodass Pods verschoben werden können, bevor die Instance zurückgeholt wird. Um die Behandlung von Unterbrechungen zu aktivieren, konfigurieren Sie das `--interruption-queue` CLI-Argument mit dem Namen der SQS-Warteschlange, die für diesen Zweck bereitgestellt wurde. Es wird nicht empfohlen, Karpenter Disruption Handling zusammen mit Node Termination Handler zu verwenden, wie hier erklärt.

Pods, für die Checkpoints oder andere Formen der ordnungsgemäßen Entleerung erforderlich sind und die 2 Minuten vor dem Herunterfahren benötigt werden, sollten die Karpenter-Interruptionsbehandlung in ihren Clustern aktivieren.

Privater Amazon EKS-Cluster ohne ausgehenden Internetzugang

Wenn Sie einen EKS-Cluster in einer VPC ohne Route zum Internet bereitstellen, müssen Sie sicherstellen, dass Sie Ihre Umgebung gemäß den privaten Cluster-Anforderungen konfiguriert haben, die in der EKS-Dokumentation aufgeführt sind. Darüber hinaus müssen Sie sicherstellen, dass Sie in Ihrer VPC einen regionalen STS-VPC-Endpunkt erstellt haben. Wenn nicht, werden Ihnen ähnliche Fehler wie die unten aufgeführten angezeigt.

```
{"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"}
```

Diese Änderungen sind in einem privaten Cluster erforderlich, da der Karpenter Controller IAM-Rollen für Dienstkonten (IRSA) verwendet. Mit IRSA konfigurierte Pods erhalten Anmeldeinformationen, indem sie die AWS Security Token Service (AWS STS) -API aufrufen. Wenn es keinen ausgehenden Internetzugang gibt, müssen Sie einen AWS STS STS-VPC-Endpunkt in Ihrer VPC erstellen und verwenden.

Bei privaten Clustern müssen Sie außerdem einen VPC-Endpunkt für SSM erstellen. Wenn Karpenter versucht, einen neuen Knoten bereitzustellen, fragt es die Launch-Vorlagenkonfigurationen und einen SSM-Parameter ab. Wenn Sie in Ihrer VPC keinen SSM-VPC-Endpunkt haben, führt dies zu dem folgenden Fehler:

```
{"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"}
```

Es gibt keinen VPC-Endpunkt für die Preislistenabfrage-API. Infolgedessen werden die Preisdaten im Laufe der Zeit veralten. Karpenter umgeht dies, indem es On-Demand-Preisdaten in seine Binärdatei aufnimmt, diese Daten jedoch nur aktualisiert, wenn Karpenter aktualisiert wird. Fehlgeschlagene Anfragen nach Preisdaten führen zu den folgenden Fehlermeldungen:

```
{"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"}
```

In dieser Dokumentation erfahren Sie, wie Sie Karpenter in einem vollständig privaten EKS-Cluster verwenden und wissen, welche VPC-Endpoints erstellt werden müssen.

Erstellen NodePools

Die folgenden bewährten Methoden behandeln Themen rund um das Erstellen NodePools.

Erstellen Sie mehrere NodePools , wenn...

Wenn sich verschiedene Teams einen Cluster teilen und ihre Workloads auf unterschiedlichen Worker-Knoten ausführen müssen oder unterschiedliche Anforderungen an das Betriebssystem oder den Instanztyp haben, erstellen Sie mehrere NodePools. Beispielsweise möchte ein Team möglicherweise Bottlerocket verwenden, während ein anderes möglicherweise Amazon Linux verwenden möchte. Ebenso könnte ein Team Zugriff auf teure GPU-Hardware haben, die von einem anderen Team nicht benötigt würde. Durch die Verwendung mehrerer Ressourcen NodePools wird sichergestellt, dass jedem Team die am besten geeigneten Ressourcen zur Verfügung stehen.

Erstellen Sie NodePools , die sich gegenseitig ausschließen oder gewichtet sind

Es wird empfohlen, solche zu erstellen NodePools , die sich entweder gegenseitig ausschließen oder gewichtet sind, um ein einheitliches Planungsverhalten zu gewährleisten. Wenn dies nicht der Fall ist und mehrere Treffer NodePools zutreffen, wählt Karpenter nach dem Zufallsprinzip aus, was zu unerwarteten Ergebnissen führt. Zu den nützlichen Beispielen für die Erstellung von mehreren NodePools gehören die folgenden:

Eine NodePool mit GPU erstellen und nur spezielle Workloads auf diesen (teuren) Knoten ausführen lassen:

```
# 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"
```

Bereitstellung mit Toleranz für den Makel:

```
# 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"
```

Bei einer allgemeinen Bereitstellung für ein anderes Team könnte die NodePool Spezifikation NodeAffinity enthalten. Ein entsprechendes Deployment könnte dann verwendet werden. nodeSelectorTerms `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
```

Bereitstellung mit 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"]
```

Verwenden Sie Timer (TTL), um Knoten automatisch aus dem Cluster zu löschen

Sie können Timer auf bereitgestellten Knoten verwenden, um festzulegen, wann Knoten gelöscht werden sollen, die keine Workload-Pods haben oder die eine Ablaufzeit erreicht haben. Der Ablauf von Knoten kann als Mittel zur Aktualisierung verwendet werden, sodass Knoten ausgemustert und durch aktualisierte Versionen ersetzt werden. Informationen zur Konfiguration des Ablaufs von Knoten finden Sie in der Karpenter-Dokumentation unter `spec.template.spec` Ablauf.

Vermeiden Sie es, die Instance-Typen, die Karpenter bereitstellen kann, zu stark einzuschränken, insbesondere wenn Sie Spot verwenden

Bei der Verwendung von Spot verwendet Karpenter die Zuweisungsstrategie Price Capacity Optimized zur Bereitstellung von EC2-Instances. Diese Strategie weist EC2 an, Instances aus den tiefsten Pools für die Anzahl der Instances bereitzustellen, die Sie starten und bei denen das Risiko einer Unterbrechung am geringsten ist. EC2 Fleet fordert dann Spot-Instances aus den Pools mit dem niedrigsten Preis an. Je mehr Instance-Typen Sie Karpenter verwenden dürfen, desto besser kann EC2 die Laufzeit Ihrer Spot-Instance optimieren. Standardmäßig verwendet Karpenter alle Instance-Typen, die EC2 in der Region und den Availability Zones anbietet, in denen Ihr Cluster bereitgestellt wird. Karpenter wählt auf intelligente Weise aus dem Satz aller Instance-Typen auf der Grundlage ausstehender Pods aus, um sicherzustellen, dass Ihre Pods auf Instances mit entsprechender Größe und Ausstattung eingeplant werden. Wenn Ihr Pod beispielsweise keine GPU benötigt, ordnet Karpenter Ihrem Pod keinen EC2-Instanztyp zu, der eine GPU unterstützt. Wenn Sie sich nicht sicher sind, welche Instance-Typen Sie verwenden sollen, können Sie den Amazon ec2-instance-selector ausführen, um eine Liste von Instance-Typen zu generieren, die Ihren Rechenanforderungen entsprechen. Die CLI verwendet beispielsweise Speicher, vCPU, Architektur und Region als Eingabeparameter und stellt Ihnen eine Liste von EC2-Instances zur Verfügung, die diese Einschränkungen erfüllen.

```
$ 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
```

Sie sollten Karpenter nicht zu viele Einschränkungen auferlegen, wenn Sie Spot-Instances verwenden, da dies die Verfügbarkeit Ihrer Anwendungen beeinträchtigen kann. Nehmen wir zum Beispiel an, dass alle Instanzen eines bestimmten Typs zurückgefordert werden und es keine geeigneten Alternativen gibt, um sie zu ersetzen. Ihre Pods bleiben so lange im Status „Ausstehend“, bis die Spot-Kapazität für die konfigurierten Instance-Typen wieder aufgefüllt ist. Sie können das Risiko von Fehlern aufgrund unzureichender Kapazität verringern, indem Sie Ihre Instances auf verschiedene Availability Zones verteilen, da Spot-Pools von Land zu Land AZs unterschiedlich sind. Die allgemeine bewährte Methode besteht jedoch darin, Karpenter bei der Verwendung von Spot die Verwendung verschiedener Instance-Typen zu gestatten.

Pods planen

Die folgenden bewährten Methoden beziehen sich auf die Bereitstellung von Pods in einem Cluster unter Verwendung von Karpenter für die Knotenbereitstellung.

Folgen Sie den Best Practices von EKS für Hochverfügbarkeit

Wenn Sie hochverfügbare Anwendungen ausführen müssen, befolgen Sie die allgemeinen EKS-Best-Practice-Empfehlungen. Einzelheiten zur Verteilung von Pods auf Knoten und Zonen finden Sie in der Dokumentation zu Topology Spread in Karpenter. Verwenden Sie Disruption Budgets, um die Mindestanzahl verfügbarer Pods festzulegen, die verwaltet werden müssen, falls versucht wird, Pods zu räumen oder zu löschen.

Verwenden Sie mehrschichtige Einschränkungen, um die von Ihrem Cloud-Anbieter verfügbaren Rechenfunktionen einzuschränken

Das Modell der mehrschichtigen Einschränkungen von Karpenter ermöglicht es Ihnen, einen komplexen Satz von NodePool Einschränkungen für die Pod-Bereitstellung zu erstellen, um die bestmöglichen Übereinstimmungen für die Pod-Planung zu erzielen. Zu den Einschränkungen, die eine Pod-Spezifikation anfordern kann, gehören unter anderem die folgenden:

- Muss in Verfügbarkeitszonen ausgeführt werden, in denen nur bestimmte Anwendungen verfügbar sind. Angenommen, Sie haben einen Pod, der mit einer anderen Anwendung kommunizieren muss, die auf einer EC2-Instance läuft, die sich in einer bestimmten Availability Zone befindet. Wenn Sie den AZ-übergreifenden Verkehr in Ihrer VPC reduzieren möchten, sollten Sie die Pods in der AZ zusammenlegen, in der sich die EC2-Instance befindet. Diese Art von Targeting wird häufig mithilfe von Knotenselektoren erreicht. Weitere Informationen zu Node-Selektoren finden Sie in der Kubernetes-Dokumentation.

- Bestimmte Arten von Prozessoren oder anderer Hardware sind erforderlich. Im Abschnitt Accelerators der Karpenter-Dokumentation finden Sie ein Beispiel für eine Pod-Spezifikation, bei der der Pod auf einer GPU ausgeführt werden muss.

Erstellen Sie Abrechnungsalarme, um Ihre Rechenausgaben zu überwachen

Wenn Sie Ihren Cluster für die automatische Skalierung konfigurieren, sollten Sie Abrechnungsalarme einrichten, die Sie warnen, wenn Ihre Ausgaben einen Schwellenwert überschreiten, und Ihrer Karpenter-Konfiguration Ressourcenlimits hinzufügen. Das Festlegen von Ressourcenlimits mit Karpenter ähnelt dem Festlegen der maximalen Kapazität einer AWS-Autoscaling-Gruppe, da es die maximale Menge an Rechenressourcen darstellt, die von einem Karpenter instanziiert werden können. NodePool

> **Note**
>
> Es ist nicht möglich, ein globales Limit für den gesamten Cluster festzulegen. Grenzwerte gelten für bestimmte NodePools.

Der folgende Ausschnitt weist Karpenter an, nur maximal 1000 CPU-Kerne und 1000 Gi Arbeitsspeicher bereitzustellen. Karpenter beendet das Hinzufügen von Kapazität erst, wenn das Limit erreicht oder überschritten wird. Wenn ein Limit überschritten wird, schreibt der Karpenter-Controller eine Meldung `memory resource usage of 1001 exceeds limit of 1000` oder eine ähnlich aussehende Meldung in die Logs des Controllers. Wenn Sie Ihre Container-Logs an CloudWatch Logs weiterleiten, können Sie einen Metrikfilter erstellen, der nach bestimmten Mustern oder Begriffen in Ihren Logs sucht, und dann einen CloudWatchAlarm einrichten, der Sie warnt, wenn Ihr konfigurierter Metrik-Schwellenwert überschritten wird.

Weitere Informationen zur Verwendung von Limits mit Karpenter finden Sie in der Karpenter-Dokumentation unter Festlegen von Ressourcenlimits.

```
spec:
  limits:
    cpu: 1000
    memory: 1000Gi
```

Wenn Sie keine Limits verwenden oder die Instanztypen einschränken, die Karpenter bereitstellen kann, fügt Karpenter Ihrem Cluster nach Bedarf weiterhin Rechenkapazität hinzu. Durch die

Konfiguration von Karpenter auf diese Weise kann Ihr Cluster zwar frei skaliert werden, dies kann jedoch auch erhebliche Kostenauswirkungen haben. Aus diesem Grund empfehlen wir die Konfiguration von Abrechnungsalarmen. Mit Abrechnungsalarmen können Sie benachrichtigt und proaktiv benachrichtigt werden, wenn die berechneten geschätzten Gebühren auf Ihren Konten einen definierten Schwellenwert überschreiten. Weitere Informationen finden Sie unter Einrichtung eines CloudWatch Amazon-Abrechnungsalarms zur proaktiven Überwachung geschätzter Gebühren.

Möglicherweise möchten Sie auch die Erkennung von Kostenanomalien aktivieren, eine AWS-Kostenmanagement-Funktion, die maschinelles Lernen verwendet, um Ihre Kosten und Nutzung kontinuierlich zu überwachen und ungewöhnliche Ausgaben zu erkennen. Weitere Informationen finden Sie im AWS Cost Anomaly Detection Getting Started Guide. Wenn Sie so weit gegangen sind, in AWS Budgets ein Budget zu erstellen, können Sie auch eine Aktion konfigurieren, die Sie benachrichtigt, wenn ein bestimmter Schwellenwert überschritten wurde. Mit Budgetaktionen können Sie eine E-Mail senden, eine Nachricht zu einem SNS-Thema posten oder eine Nachricht an einen Chatbot wie Slack senden. Weitere Informationen finden Sie unter Konfiguration von AWS-Budgets-Aktionen.

Verwenden Sie die do-not-disrupt Anmerkung karpenter.sh/, um zu verhindern, dass Karpenter einen Knoten deprovisioniert

Wenn Sie eine kritische Anwendung auf einem von Karpenter bereitgestellten Knoten ausführen, z. B. einen Batch-Job mit langer Laufzeit oder eine statusbehaftete Anwendung, und die TTL des Knotens abgelaufen ist, wird die Anwendung unterbrochen, wenn die Instanz beendet wird. Indem Sie dem Pod eine `karpenter.sh/do-not-disrupt` Anmerkung hinzufügen, weisen Sie Karpenter an, den Knoten beizubehalten, bis der Pod beendet oder die Anmerkung entfernt wird. `karpenter.sh/do-not-disrupt` Weitere Informationen finden Sie in der Dokumentation zu Distruption.

Wenn die einzigen Pods auf einem Knoten, die nicht auf Daemon gesetzt sind, diejenigen sind, die mit Jobs verknüpft sind, kann Karpenter diese Knoten gezielt ansprechen und beenden, solange der Jobstatus erfolgreich oder fehlgeschlagen ist.

Konfigurieren Sie requests=limits für alle Nicht-CPU-Ressourcen, wenn Sie die Konsolidierung verwenden

Konsolidierung und Planung funktionieren im Allgemeinen, indem die Ressourcenanforderungen des Pods mit der Menge der zuweisbaren Ressourcen auf einem Knoten verglichen werden. Die Ressourcengrenzen werden nicht berücksichtigt. Beispielsweise können Pods, deren Speicherlimit größer als die Speicheranforderung ist, bei Überschreitung der Anforderung zu einem Burst-Wert

führen. Wenn mehrere Pods auf demselben Knoten gleichzeitig platzen, kann dies dazu führen, dass einige der Pods aufgrund eines Speicherausfalls (OOM) beendet werden. Durch Konsolidierung kann dies wahrscheinlicher werden, da es funktioniert, Pods auf Knoten nur unter Berücksichtigung ihrer Anfragen zu packen.

Wird verwendet LimitRanges , um Standardwerte für Ressourcenanfragen und Grenzwerte zu konfigurieren

Da Kubernetes keine Standardanforderungen oder Grenzwerte festlegt, ist der Ressourcenverbrauch eines Containers vom zugrunde liegenden Host, der CPU und dem Speicher unbegrenzt. Der Kubernetes-Scheduler untersucht die Gesamtzahl der Anfragen eines Pods (je höher die Gesamtzahl der Anfragen aus den Containern des Pods oder die Gesamtressourcen der Init-Container des Pods), um zu ermitteln, auf welchem Worker-Knoten der Pod eingeplant werden soll. In ähnlicher Weise berücksichtigt Karpenter die Anfragen eines Pods, um zu bestimmen, welchen Instanztyp er bereitstellt. Sie können einen Grenzbereich verwenden, um einen sinnvollen Standard für einen Namespace anzuwenden, falls Ressourcenanforderungen von einigen Pods nicht spezifiziert werden.

Siehe Standardspeicheranforderungen und Grenzwerte für einen Namespace konfigurieren

Wenden Sie korrekte Ressourcenanforderungen auf alle Workloads an

Karpenter ist in der Lage, Knoten zu starten, die am besten zu Ihren Workloads passen, wenn die Informationen über Ihre Workload-Anforderungen korrekt sind. Dies ist besonders wichtig, wenn Sie die Konsolidierungsfunktion von Karpenter verwenden.

Weitere Informationen finden Sie unter Ressourcen Requests/Limits für alle Workloads konfigurieren und dimensionieren

CoreDNS-Empfehlungen

Aktualisieren Sie die Konfiguration von CoreDNS, um die Zuverlässigkeit aufrechtzuerhalten

Bei der Bereitstellung von CoreDNS-Pods auf von Karpenter verwalteten Knoten ist es angesichts des dynamischen Charakters von Karpenter bei schnell terminating/creating neuen Knoten zur Anpassung an die Nachfrage ratsam, die folgenden Best Practices einzuhalten:

Dauer CoreDNS CoreDNS-Lameduck

CoreDNS-Bereitschaftstest

Dadurch wird sichergestellt, dass DNS-Abfragen nicht an einen CoreDNS-Pod weitergeleitet werden, der noch nicht bereit ist oder beendet wurde.

Baupläne von Karpenter

Da Karpenter bei der Bereitstellung von Rechenkapazität für die Kubernetes-Datenebene einen anwendungsorientierten Ansatz verfolgt, gibt es häufig auftretende Workload-Szenarien, bei denen Sie sich vielleicht fragen, wie sie richtig konfiguriert werden sollen. Karpenter Blueprints ist ein Repository, das eine Liste gängiger Workload-Szenarien gemäß den hier beschriebenen Best Practices enthält. Sie verfügen über alle Ressourcen, die Sie benötigen, um sogar einen EKS-Cluster mit konfiguriertem Karpenter zu erstellen und jeden der im Repository enthaltenen Blueprints zu testen. Sie können verschiedene Blueprints kombinieren, um schließlich den zu erstellen, den Sie für Ihre Arbeitslast (en) benötigen.

Weitere Ressourcen

- Workshop zum Karpenter Immersion Day
- Workshop zur Kostenoptimierung in Karpenter
- EKS-Werkstatt - Karpenter
- Karpenter gegen Cluster Autoscaler
- Karpenter Session auf der re:Invent 2023
- Tutorial: Kubernetes-Cluster mit Amazon EC2 Spot und Karpenter günstiger ausführen

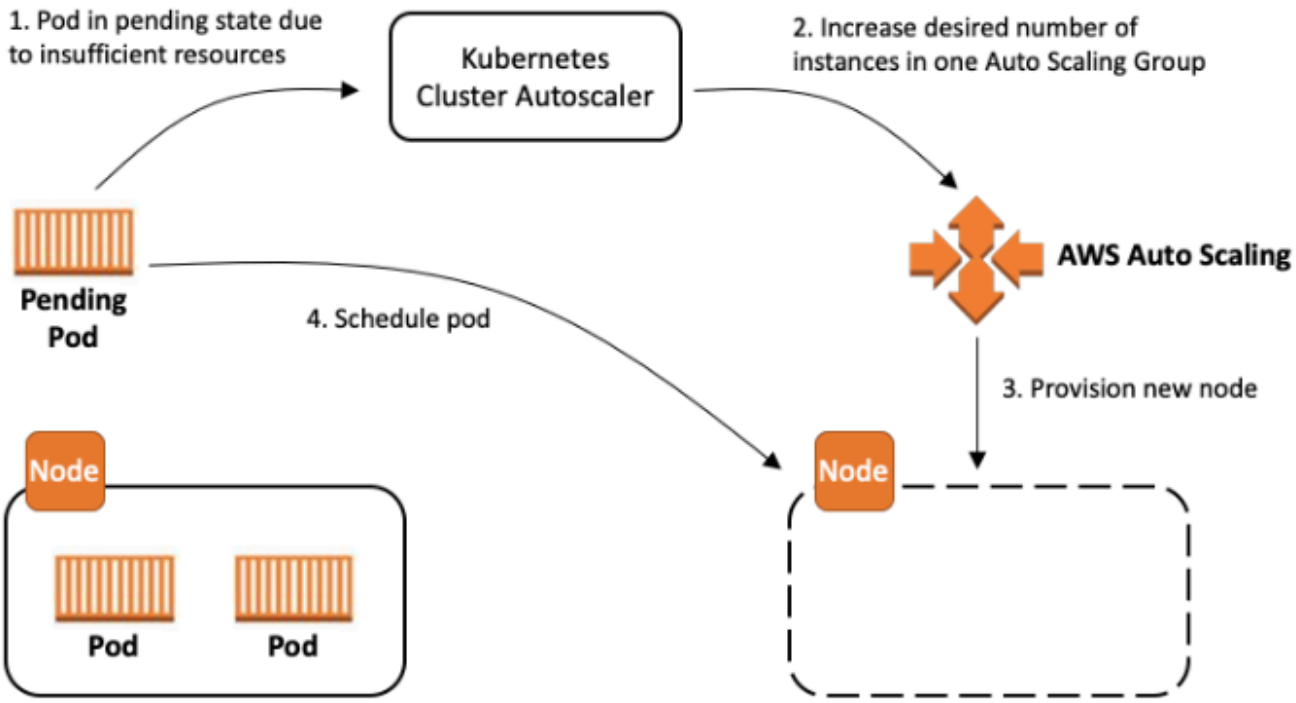
Cluster Autoscaler

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

-Übersicht

Der Kubernetes Cluster Autoscaler ist eine beliebte Cluster-Autoscaling-Lösung, die von SIG Autoscaling verwaltet wird. Er ist dafür verantwortlich, dass Ihr Cluster über genügend Knoten

verfügt, um Ihre Pods zu planen, ohne Ressourcen zu verschwenden. Es sucht nach Pods, deren Zeitplan nicht eingehalten wird, und nach Knoten, die nicht ausgelastet sind. Anschließend wird das Hinzufügen oder Entfernen von Knoten simuliert, bevor die Änderung auf Ihren Cluster angewendet wird. Die AWS Cloud Provider-Implementierung innerhalb von Cluster Autoscaler steuert das `.DesiredReplicas` Feld Ihrer EC2 Auto Scaling Scaling-Gruppen.



Dieser Leitfaden bietet ein mentales Modell für die Konfiguration des Cluster Autoscaler und die Auswahl der besten Kompromisse, um die Anforderungen Ihres Unternehmens zu erfüllen. Es gibt zwar keine einzige optimale Konfiguration, aber es gibt eine Reihe von Konfigurationsoptionen, mit denen Sie Kompromisse zwischen Leistung, Skalierbarkeit, Kosten und Verfügbarkeit eingehen können. Darüber hinaus enthält dieser Leitfaden Tipps und bewährte Methoden zur Optimierung Ihrer Konfiguration für AWS.

Glossar

Die folgende Terminologie wird in diesem Dokument häufig verwendet. Diese Begriffe können eine weit gefasste Bedeutung haben, sind jedoch für die Zwecke dieses Dokuments auf die folgenden Definitionen beschränkt.

Skalierbarkeit bezieht sich darauf, wie gut der Cluster Autoscaler funktioniert, wenn die Anzahl der Pods und Knoten Ihres Kubernetes-Clusters zunimmt. Wenn die Skalierbarkeitsgrenzen erreicht werden, verschlechtern sich die Leistung und Funktionalität des Cluster Autoscalers. Wenn der

Cluster Autoscaler seine Skalierbarkeitsgrenzen überschreitet, kann er Ihrem Cluster keine Knoten mehr hinzufügen oder entfernen.

Leistung bezieht sich darauf, wie schnell der Cluster Autoscaler Skalierungsentscheidungen treffen und ausführen kann. Ein Cluster-Autoscaler mit perfekter Leistung würde sofort eine Entscheidung treffen und eine Skalierungsaktion als Reaktion auf Stimuli auslösen, z. B. wenn ein Pod nicht mehr planbar ist.

Verfügbarkeit bedeutet, dass Pods schnell und ohne Unterbrechung geplant werden können. Dazu gehört auch, wann neu erstellte Pods geplant werden müssen und wann ein herunterskalierter Knoten alle verbleibenden Pods beendet, die für ihn geplant sind.

Die Kosten hängen von der Entscheidung ab, die hinter der Skalierung und Skalierung von Ereignissen steht. Ressourcen werden verschwendet, wenn ein vorhandener Knoten nicht ausgelastet ist oder ein neuer Knoten hinzugefügt wird, der zu groß für eingehende Pods ist. Je nach Anwendungsfall kann es zu Kosten kommen, wenn Pods aufgrund einer aggressiven Skalierungsentscheidung vorzeitig beendet werden.

Knotengruppen sind ein abstraktes Kubernetes-Konzept für eine Gruppe von Knoten innerhalb eines Clusters. Es ist keine echte Kubernetes-Ressource, sondern existiert als Abstraktion im Cluster Autoscaler, der Cluster-API und anderen Komponenten. Knoten innerhalb einer Knotengruppe haben Eigenschaften wie Labels und Taints gemeinsam, können aber aus mehreren Availability Zones oder Instance-Typen bestehen.

EC2 Auto Scaling Groups können als Implementierung von Node Groups auf EC2 verwendet werden. EC2 Auto Scaling Groups sind so konfiguriert, dass sie Instances starten, die ihren Kubernetes-Clustern automatisch beitreten, und Labels und Taints auf ihre entsprechende Node-Ressource in der Kubernetes-API anwenden.

EC2 Managed Node Groups sind eine weitere Implementierung von Node Groups auf EC2. Sie reduzieren die Komplexität der manuellen Konfiguration von EC2 Autoscaling Scaling Groups und bieten zusätzliche Verwaltungsfunktionen wie ein Upgrade der Knotenversion und eine ordnungsgemäße Knotenbeendigung.

Betrieb des Cluster-Autoscalers

Der Cluster Autoscaler wird normalerweise als Bereitstellung in Ihrem Cluster installiert. Um eine hohe Verfügbarkeit zu gewährleisten, nutzt er die Wahl eines Leaders, aber die Arbeit wird jeweils von einem einzelnen Replikat erledigt. Es ist nicht horizontal skalierbar. Bei Basis-Setups sollte es

standardmäßig mit den mitgelieferten Installationsanweisungen sofort einsatzbereit sein, aber es gibt ein paar Dinge zu beachten.

Stellen Sie Folgendes sicher:

- Die Version des Cluster-Autoscalers entspricht der Version des Clusters. Die versionsübergreifende Kompatibilität wurde nicht getestet oder unterstützt.
- Auto Discovery ist aktiviert, es sei denn, Sie haben spezielle erweiterte Anwendungsfälle, die die Verwendung dieses Modus verhindern.

Verwenden Sie den Zugriff mit den geringsten Rechten auf die IAM-Rolle

Wenn Auto Discovery verwendet wird, empfehlen wir dringend, den Zugriff mit den geringsten Rechten `autoscaling:TerminateInstanceInAutoScalingGroup` zu verwenden, indem Sie Aktionen `autoscaling:SetDesiredCapacity` auf die Auto Scaling-Gruppen beschränken, die auf den aktuellen Cluster beschränkt sind.

Dadurch wird verhindert, dass ein Cluster-Autoscaler, der in einem Cluster ausgeführt wird, Knotengruppen in einem anderen Cluster ändert, auch wenn das `--node-group-auto-discovery` Argument nicht mithilfe von Tags auf die Knotengruppen des Clusters beschränkt wurde (z. B.). `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": "*"
        }
    ]
}
```

Konfiguration Ihrer Knotengruppen

Effektives Autoscaling beginnt mit der korrekten Konfiguration einer Reihe von Knotengruppen für Ihren Cluster. Die Auswahl der richtigen Gruppe von Knotengruppen ist entscheidend, um die Verfügbarkeit zu maximieren und die Kosten für Ihre Workloads zu senken. AWS implementiert Node Groups mithilfe von EC2 Auto Scaling Groups, die für eine Vielzahl von Anwendungsfällen flexibel sind. Der Cluster Autoscaler geht jedoch von einigen Annahmen über Ihre Node Groups aus. Wenn Sie Ihre EC2 Auto Scaling Group-Konfigurationen mit diesen Annahmen konsistent halten, wird unerwünschtes Verhalten minimiert.

Stellen Sie Folgendes sicher:

- Jeder Knoten in einer Knotengruppe hat identische Scheduling-Eigenschaften wie Labels, Taints und Resources.
  - Denn MixedInstancePolicies die Instanztypen müssen für CPU, Arbeitsspeicher und GPU dieselbe Form haben
  - Der erste in der Richtlinie angegebene Instance-Typ wird verwendet, um die Planung zu simulieren.
  - Wenn Ihre Richtlinie zusätzliche Instance-Typen mit mehr Ressourcen vorsieht, können Ressourcen nach dem Skalieren verschwendet werden.
  - Wenn Ihre Richtlinie zusätzliche Instance-Typen mit weniger Ressourcen vorsieht, können Pods die Instances möglicherweise nicht einplanen.

- Knotengruppen mit vielen Knoten werden vielen Knotengruppen mit weniger Knoten vorgezogen. Dies wird den größten Einfluss auf die Skalierbarkeit haben.
- Bevorzugen Sie nach Möglichkeit EC2-Funktionen, wenn beide Systeme Unterstützung bieten (z. B. Regionen, MixedInstancePolicy)

> **Note**
>
> Wir empfehlen die Verwendung von EKS Managed Node Groups. Verwaltete Knotengruppen verfügen über leistungsstarke Verwaltungsfunktionen, darunter Funktionen für Cluster Autoscaler wie die automatische Erkennung von EC2 Auto Scaling Scaling-Gruppen und die ordnungsgemäße Knotenbeendigung.

Optimierung im Hinblick auf Leistung und Skalierbarkeit

Wenn Sie die Komplexität der Laufzeit des Autoscaling-Algorithmus verstehen, können Sie den Cluster Autoscaler so optimieren, dass er auch in großen Clustern mit mehr als 1.000 Knoten reibungslos funktioniert.

Die wichtigsten Faktoren für die Optimierung der Skalierbarkeit des Cluster-Autoscalers sind die für den Prozess bereitgestellten Ressourcen, das Scanintervall des Algorithmus und die Anzahl der Knotengruppen im Cluster. Die tatsächliche Laufzeitkomplexität dieses Algorithmus hängt von anderen Faktoren ab, wie z. B. die Komplexität des Plug-ins für die Terminplanung und die Anzahl der Pods. Diese Parameter werden als nicht konfigurierbare Parameter betrachtet, da sie von der Arbeitslast des Clusters abhängen und nicht einfach angepasst werden können.

Der Cluster Autoscaler lädt den Status des gesamten Clusters in den Arbeitsspeicher, einschließlich Pods, Knoten und Knotengruppen. Bei jedem Scanintervall identifiziert der Algorithmus Pods, die nicht geplant werden können, und simuliert die Planung für jede Knotengruppe. Die Optimierung dieser Faktoren ist mit verschiedenen Kompromissen verbunden, die für Ihren Anwendungsfall sorgfältig abgewogen werden sollten.

Vertikale automatische Skalierung des Cluster-Autoscalers

Die einfachste Möglichkeit, den Cluster-Autoscaler auf größere Cluster zu skalieren, besteht darin, die Ressourcenanforderungen für seine Bereitstellung zu erhöhen. Sowohl Arbeitsspeicher als auch CPU sollten für große Cluster erhöht werden, obwohl dies je nach Clustergröße erheblich variiert. Der Autoscaling-Algorithmus speichert alle Pods und Knoten im Arbeitsspeicher, was in einigen Fällen

zu einem Speicherbedarf von mehr als einem Gigabyte führen kann. Das Erhöhen der Ressourcen erfolgt in der Regel manuell. Wenn Sie feststellen, dass die ständige Optimierung der Ressourcen zu einer Betriebsbelastung führt, sollten Sie die Verwendung von Addon Resizer oder Vertical Pod Autoscaler in Betracht ziehen.

Reduzierung der Anzahl der Knotengruppen

Die Minimierung der Anzahl der Knotengruppen ist eine Möglichkeit, sicherzustellen, dass der Cluster Autoscaler auch in großen Clustern eine gute Leistung erbringt. Dies kann für einige Organisationen, die ihre Knotengruppen pro Team oder pro Anwendung strukturieren, eine Herausforderung darstellen. Dies wird zwar vollständig von der Kubernetes-API unterstützt, es wird jedoch als Cluster-Autoscaler-Anti-Pattern mit Auswirkungen auf die Skalierbarkeit angesehen. Es gibt viele Gründe, mehrere Knotengruppen zu verwenden (z. B. Spot oder GPUs), aber in vielen Fällen gibt es alternative Designs, die bei Verwendung einer kleinen Anzahl von Gruppen den gleichen Effekt erzielen.

Stellen Sie Folgendes sicher:

- Die Pod-Isolierung erfolgt mithilfe von Namespaces und nicht mithilfe von Knotengruppen.
  - Dies ist in Clustern mit mehreren Mandanten mit geringem Vertrauensgrad möglicherweise nicht möglich.
  - Pod ResourceRequests und ResourceLimits sind richtig eingestellt, um Ressourcenkonflikte zu vermeiden.
  - Größere Instance-Typen führen zu einer optimaleren Bin-Packung und zu einem geringeren System-Pod-Overhead.
- NodeTaints oder NodeSelectors werden verwendet, um Pods als Ausnahme und nicht als Regel einzuplanen.
- Regionale Ressourcen sind als eine einzelne EC2 Auto Scaling Scaling-Gruppe mit mehreren Availability Zones definiert.

Verkürzung des Scan-Intervalls

Ein niedriges Scanintervall (z. B. 10 Sekunden) stellt sicher, dass der Cluster-Autoscaler so schnell wie möglich reagiert, wenn Pods nicht mehr planbar sind. Jeder Scan führt jedoch zu vielen API-Aufrufen an die Kubernetes-API und die EC2 Auto Scaling Group oder die EKS Managed Node Group. APIs Diese API-Aufrufe können zu einer Ratenbegrenzung oder sogar zur Nichtverfügbarkeit des Dienstes für Ihre Kubernetes Control Plane führen.

Das Standard-Scan-Intervall beträgt 10 Sekunden, aber auf AWS dauert das Starten eines Knotens erheblich länger, um eine neue Instance zu starten. Dies bedeutet, dass es möglich ist, das Intervall zu erhöhen, ohne die Gesamtzeit für die Hochskalierung zu erhöhen. Wenn es beispielsweise 2 Minuten dauert, einen Knoten zu starten, führt eine Änderung des Intervalls auf 1 Minute zu einem Kompromiss zwischen 6-fach reduzierten API-Aufrufen und einer um 38% langsameren Skalierung.

Sharding zwischen Knotengruppen

Der Cluster Autoscaler kann so konfiguriert werden, dass er auf einer bestimmten Gruppe von Knotengruppen ausgeführt wird. Mithilfe dieser Funktionalität ist es möglich, mehrere Instanzen des Cluster Autoscaler bereitzustellen, von denen jede für den Betrieb auf einem anderen Satz von Knotengruppen konfiguriert ist. Mit dieser Strategie können Sie eine beliebig große Anzahl von Knotengruppen verwenden und dabei Kosten gegen Skalierbarkeit eintauschen. Wir empfehlen, dies nur als letzten Ausweg zur Leistungsverbesserung zu verwenden.

Der Cluster Autoscaler wurde ursprünglich nicht für diese Konfiguration entwickelt, daher gibt es einige Nebenwirkungen. Da die Shards nicht miteinander kommunizieren, ist es möglich, dass mehrere Autoscaler versuchen, einen Pod zu planen, der nicht geplant werden kann. Dies kann zu einer unnötigen Skalierung mehrerer Knotengruppen führen. Diese zusätzlichen Knoten werden nach dem wieder skaliert`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
```

Stellen Sie Folgendes sicher:

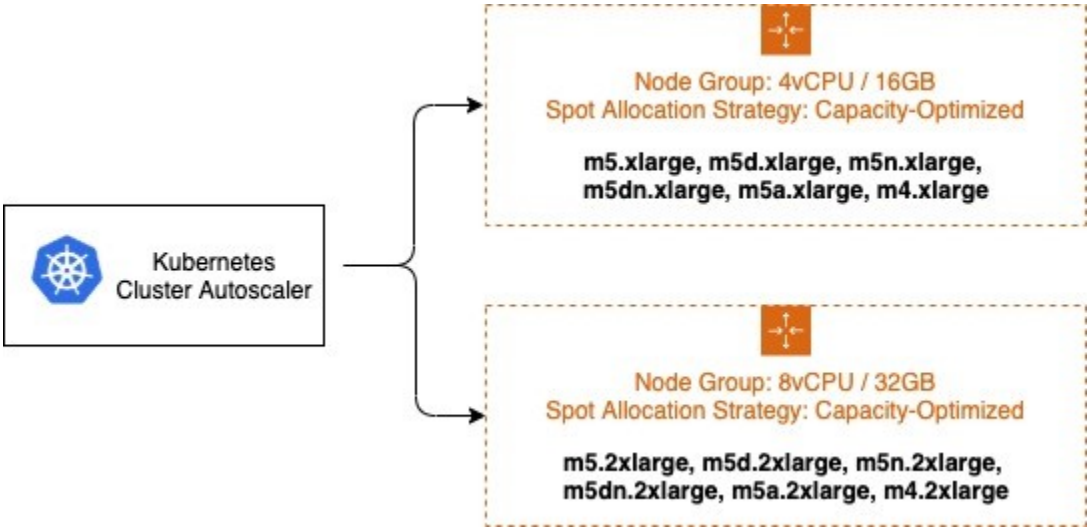
- Jeder Shard ist so konfiguriert, dass er auf einen eindeutigen Satz von EC2 Auto Scaling Scaling-Gruppen verweist
- Jeder Shard wird in einem separaten Namespace bereitgestellt, um Konflikte bei der Wahl von Führungskräften zu vermeiden

Optimierung im Hinblick auf Kosten und Verfügbarkeit

Spot Instances

Sie können Spot-Instances in Ihren Knotengruppen verwenden und bis zu 90% gegenüber dem On-Demand-Preis sparen. Im Gegenzug können Spot-Instances jederzeit unterbrochen werden, wenn EC2 die Kapazität wieder benötigt. Fehler mit unzureichender Kapazität treten auf, wenn Ihre EC2 Auto Scaling Scaling-Gruppe aufgrund mangelnder verfügbarer Kapazität nicht hochskaliert werden kann. Wenn Sie die Vielfalt maximieren, indem Sie viele Instance-Familien auswählen, können Sie Ihre Chance erhöhen, die gewünschte Skalierung zu erreichen, indem Sie viele Spot-Kapazitätspools nutzen, und die Auswirkungen von Spot-Instance-Unterbrechungen auf die Verfügbarkeit Ihres Clusters verringern. Gemischte Instance-Richtlinien mit Spot-Instances sind eine großartige Möglichkeit, die Vielfalt zu erhöhen, ohne die Anzahl der Knotengruppen zu erhöhen. Denken Sie daran, dass Sie On-Demand-Instances anstelle von Spot-Instances verwenden sollten, wenn Sie garantierte Ressourcen benötigen.

Bei der Konfiguration gemischter Instance-Richtlinien ist es wichtig, dass alle Instance-Typen über eine ähnliche Ressourcenkapazität verfügen. Der Scheduling-Simulator des Autoscalers verwendet den ersten InstanceType in der. MixedInstancePolicy Wenn nachfolgende Instance-Typen größer sind, können Ressourcen nach einer Skalierung verschwendet werden. Wenn sie kleiner sind, können Ihre Pods aufgrund unzureichender Kapazität möglicherweise nicht im Zeitplan für die neuen Instances eingesetzt werden. Beispielsweise verfügen M4-, M5-, M5a- und M5n-Instances alle über eine ähnliche Menge an CPU und Arbeitsspeicher und eignen sich hervorragend für eine. MixedInstancePolicy Das EC2 Instance Selector Tool kann Ihnen helfen, ähnliche Instance-Typen zu identifizieren.



Es wird empfohlen, On-Demand- und Spot-Kapazitäten in separate EC2 Auto Scaling Scaling-Gruppen zu isolieren. Dies wird der Verwendung einer Basiskapazitätsstrategie vorgezogen, da sich die Scheduling-Eigenschaften grundlegend unterscheiden. Da Spot-Instances jederzeit unterbrochen werden können (wenn EC2 die Kapazität wieder benötigt), belasten Benutzer häufig ihre präemptiven Knoten, sodass eine ausdrückliche Pod-Toleranz gegenüber dem Preemption-Verhalten erforderlich ist. Diese Fehler führen zu unterschiedlichen Scheduling-Eigenschaften für die Knoten, weshalb sie in mehrere EC2 Auto Scaling Scaling-Gruppen aufgeteilt werden sollten.

Der Cluster Autoscaler basiert auf einem Konzept von Expandern, die unterschiedliche Strategien für die Auswahl der zu skalierenden Knotengruppe bieten. Diese Strategie `--expander=least-waste` ist eine gute Standardstrategie für allgemeine Zwecke. Wenn Sie mehrere Knotengruppen für die Diversifizierung von Spot-Instances verwenden möchten (wie in der Abbildung oben beschrieben), kann sie zur weiteren Kostenoptimierung der Knotengruppen beitragen, indem die Gruppe skaliert wird, die nach der Skalierung am besten genutzt werden kann.

Priorisierung einer Knotengruppe//ASG

Sie können die prioritätsbasierte automatische Skalierung auch mithilfe des Priority Expanders konfigurieren. `--expander=priority`ermöglicht es Ihrem Cluster, einer Knotengruppe /ASG Priorität einzuräumen. Wenn er aus irgendeinem Grund nicht skalieren kann, wählt er die nächste Knotengruppe in der Prioritätenliste aus. Dies ist nützlich in Situationen, in denen Sie beispielsweise P3-Instance-Typen verwenden möchten, da deren GPU eine optimale Leistung für Ihren Workload bietet. Als zweite Option können Sie aber auch P2-Instance-Typen verwenden.

```
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 versucht, die EC2 Auto Scaling-Gruppe entsprechend dem Namen p3-node-group zu skalieren. Wenn dieser Vorgang innerhalb nicht erfolgreich ist`--max-node-provision-time`, wird versucht, eine EC2 Auto Scaling-Gruppe zu skalieren, die dem Namen p2-node-group entspricht. Dieser Wert ist standardmäßig auf 15 Minuten voreingestellt und kann für eine reaktionsschnellere Knotengruppenauswahl reduziert werden. Wenn der Wert jedoch zu niedrig ist, kann dies zu unnötigen Skalierungen führen.

Überbereitstellung

Der Cluster Autoscaler minimiert die Kosten, indem er sicherstellt, dass Knoten dem Cluster nur hinzugefügt werden, wenn sie benötigt werden, und entfernt werden, wenn sie nicht verwendet werden. Dies wirkt sich erheblich auf die Bereitstellungslatenz aus, da viele Pods gezwungen sein werden, auf die Skalierung eines Knotens zu warten, bevor sie geplant werden können. Es kann mehrere Minuten dauern, bis Knoten verfügbar sind, was die Pod-Planungslatenz um eine Größenordnung erhöhen kann.

Dies kann durch Overprovisioning gemildert werden, bei dem die Kosten für die Planungslatenz eingetauscht werden. Overprovisioning wird mithilfe temporärer Pods mit negativer Priorität implementiert, die Speicherplatz im Cluster belegen. Wenn neu erstellte Pods nicht planbar sind und eine höhere Priorität haben, werden die temporären Pods gesperrt, um Platz zu schaffen. Die temporären Pods können dann nicht mehr geplant werden, was den Cluster Autoscaler veranlasst, neue Knoten mit überdimensionierter Kapazität zu skalieren.

Überprovisionierung bietet weitere, weniger offensichtliche Vorteile. Ohne Überprovisionierung besteht einer der Nebeneffekte eines stark ausgelasteten Clusters darin, dass Pods mithilfe der `preferredDuringSchedulingIgnoredDuringExecution` Regel der Pod- oder Node-Affinität weniger optimale Planungsentscheidungen treffen. Ein häufiger Anwendungsfall hierfür ist die Trennung von Pods für eine hochverfügbare Anwendung über mehrere Verfügbarkeitszonen hinweg mithilfe von. AntiAffinity Eine Überprovisionierung kann die Wahrscheinlichkeit, dass ein Knoten der richtigen Zone verfügbar ist, erheblich erhöhen.

Die Höhe der überbereitgestellten Kapazität ist eine sorgfältige Geschäftsentscheidung für Ihr Unternehmen. Im Kern handelt es sich um einen Kompromiss zwischen Leistung und Kosten. Eine Möglichkeit, diese Entscheidung zu treffen, besteht darin, Ihre durchschnittliche Skalierungshäufigkeit zu ermitteln und diese durch die Zeit zu dividieren, die für die Skalierung eines neuen Knotens benötigt wird. Wenn Sie beispielsweise im Durchschnitt alle 30 Sekunden einen neuen Knoten benötigen und EC2 30 Sekunden benötigt, um einen neuen Knoten bereitzustellen, sorgt ein einziger Overprovisioning-Node dafür, dass immer ein zusätzlicher Knoten verfügbar ist, wodurch die Planungslatenz um 30 Sekunden reduziert wird — auf Kosten einer einzigen zusätzlichen EC2-Instance. Um die Entscheidungen zur zonalen Planung zu verbessern, sollten Sie eine Anzahl von Knoten, die der Anzahl der Availability Zones in Ihrer EC2 Auto Scaling Scaling-Gruppe entspricht, zu viel bereitstellen, um sicherzustellen, dass der Scheduler die beste Zone für eingehende Pods auswählen kann.

Vermeiden Sie die Räumung von Scale Down

Die Bereinigung einiger Workloads ist teuer. Big-Data-Analysen, Aufgaben für maschinelles Lernen und Testläufe werden irgendwann abgeschlossen, müssen aber neu gestartet werden, wenn sie unterbrochen werden. Der Cluster Autoscaler versucht, jeden Knoten unter dem herunterzuskalieren scale-down-utilization-threshold, wodurch alle verbleibenden Pods auf dem Knoten unterbrochen werden. Dies kann verhindert werden, indem sichergestellt wird, dass Pods, deren Entfernung teuer ist, durch ein Etikett geschützt werden, das vom Cluster Autoscaler erkannt wird.

Stellen Sie Folgendes sicher:

- Bei Pods, die teuer zu entfernen sind, gibt es die Anmerkung `cluster-autoscaler.kubernetes.io/safe-to-evict=false`

Fortschrittliche Anwendungsfälle

EBS-Volumes

Persistenter Speicher ist entscheidend für die Erstellung von zustandsbehafteten Anwendungen wie Datenbanken oder verteilten Caches. EBS-Volumes ermöglichen diesen Anwendungsfall auf Kubernetes, sind jedoch auf eine bestimmte Zone beschränkt. Diese Anwendungen können hochverfügbar sein, wenn sie auf mehrere AZs verteilt werden, wobei für jede AZ ein separates EBS-Volume verwendet wird. Der Cluster Autoscaler kann dann die Skalierung der EC2-Autoscaling-Gruppen ausgleichen.

Stellen Sie Folgendes sicher:

- Der Knotengruppenausgleich wird durch Setzen von `balance-similar-node-groups=true` aktiviert.
- Knotengruppen werden mit identischen Einstellungen konfiguriert, mit Ausnahme verschiedener Availability Zones und EBS-Volumes.

Gemeinsame Terminplanung

Verteilte Trainingsjobs für Machine Learning profitieren erheblich von der minimierten Latenz von Knotenkonfigurationen in derselben Zone. Diese Workloads stellen mehrere Pods in einer bestimmten Zone bereit. Dies kann erreicht werden, indem die Pod-Affinität für alle gemeinsam geplanten Pods oder die Verwendung von Node Affinity festgelegt wird. `topologyKey: failure-domain.beta.kubernetes.io/zone` Der Cluster Autoscaler skaliert dann eine bestimmte Zone entsprechend den Anforderungen. Möglicherweise möchten Sie mehrere EC2 Auto Scaling Scaling-Gruppen zuweisen, eine pro Availability Zone, um ein Failover für den gesamten gemeinsam geplanten Workload zu ermöglichen.

Stellen Sie Folgendes sicher:

- Der Knotengruppenausgleich wird durch folgende Einstellung aktiviert `balance-similar-node-groups=false`
- Node Affinity und/oder Pod Preemption werden verwendet, wenn Cluster sowohl regionale als auch zonale Knotengruppen enthalten.
  - Verwenden Sie Node Affinity, um regionale Pods zu zwingen oder zu ermutigen, zonale Knotengruppen zu meiden und umgekehrt.
  - Wenn zonale Pods regionalen Knotengruppen zugewiesen werden, führt dies zu einer unausgewogenen Kapazität Ihrer regionalen Pods.
  - Wenn Ihre zonalen Workloads Störungen und Verlagerungen vertragen, konfigurieren Sie Pod Preemption so, dass regional skalierte Pods aktiviert werden, um Preemption und Neuplanung in einer weniger umkämpften Zone zu erzwingen.

Accelerators

Einige Cluster nutzen spezielle Hardwarebeschleuniger wie GPU. Beim Skalieren kann es mehrere Minuten dauern, bis das Accelerator Device Plug-in die Ressource dem Cluster bekannt gibt. Der Cluster Autoscaler hat simuliert, dass dieser Knoten über den Accelerator verfügen wird, aber solange der Accelerator nicht bereit ist und die verfügbaren Ressourcen des Knotens aktualisiert,

können ausstehende Pods auf dem Knoten nicht geplant werden. Dies kann zu wiederholtem unnötigem Aufskalieren führen.

Darüber hinaus werden Knoten mit Beschleunigern und hoher CPU- oder Speicherauslastung bei der Herunterskalierung nicht berücksichtigt, selbst wenn der Beschleuniger nicht verwendet wird. Dieses Verhalten kann aufgrund der relativ hohen Kosten von Beschleunigern kostspielig sein. Stattdessen kann der Cluster Autoscaler spezielle Regeln anwenden, um Knoten beim Herunterfahren in Betracht zu ziehen, wenn sie über ungenutzte Beschleuniger verfügen.

Um in diesen Fällen das richtige Verhalten sicherzustellen, können Sie das Kubelet auf Ihren Accelerator-Knoten so konfigurieren, dass der Knoten beschriftet wird, bevor er dem Cluster beitritt. Der Cluster Autoscaler verwendet diesen Labelselektor, um das für den Beschleuniger optimierte Verhalten auszulösen.

Stellen Sie Folgendes sicher:

- Das Kubelet für GPU-Knoten ist konfiguriert mit `--node-labels k8s.amazonaws.com/accelerator=$ACCELERATOR_TYPE`
- Knoten mit Beschleunigern halten sich an die oben genannte Regel für identische Scheduling-Eigenschaften.

Skalierung ab 0

Cluster Autoscaler ist in der Lage, Knotengruppen auf und von Null zu skalieren, was zu erheblichen Kosteneinsparungen führen kann. Es erkennt die CPU-, Speicher- und GPU-Ressourcen einer Auto Scaling Scaling-Gruppe, indem es die in seiner LaunchConfiguration oder InstanceType LaunchTemplate angegebenen Werte überprüft. Einige Pods benötigen zusätzliche Ressourcen wie `WindowsENI` oder oder spezifische `PrivateIPv4Address` NodeSelectors oder Taints, die anhand von nicht erkannt werden können. LaunchConfiguration Der Cluster Autoscaler kann diese Faktoren berücksichtigen, indem er sie anhand von Tags in der EC2 Auto Scaling Group erkennt. Beispiel:

```
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**
>
> Denken Sie daran, dass bei einer Skalierung auf Null Ihre Kapazität an EC2 zurückgegeben wird und möglicherweise in future nicht mehr verfügbar ist.

Zusätzliche Parameter

Es gibt viele Konfigurationsoptionen, mit denen das Verhalten und die Leistung des Cluster Autoscaler optimiert werden können. Eine vollständige Liste der Parameter ist verfügbar unter GitHub.

Parameter	Description	Standard
Scan-Intervall	Wie oft wird ein Cluster im Hinblick auf Skalierung nach oben oder unten neu bewertet	10 Sekunden
max-empty-bulk-delete	Maximale Anzahl leerer Knoten, die gleichzeitig gelöscht werden können.	10
scale-down-delay-after-hinzufügen	Wie lange dauert es, bis die Evaluierung nach oben mit dem Herunterskalieren wieder aufgenommen wird	10 Minuten
scale-down-delay-after-löschen	Wie lange nach dem Löschen des Knotens wird die Scale-Down-Evaluierung wieder aufgenommen (standard mäßig scan-interval)	Scan-Intervall
scale-down-delay-after-Ausfall	Wie lange nach einem Ausfall des Herunterskalierens wird die Down-Evaluierung wieder aufgenommen	3 Minuten

Parameter	Description	Standard
scale-down-unneeded-time	Wie lange sollte ein Knoten nicht benötigt werden, bevor er herunterskaliert werden kann	10 Minuten
scale-down-unready-time	Wie lange sollte ein nicht einsatzbereiter Knoten nicht benötigt werden, bevor er herunterskaliert werden kann	20 Minuten
scale-down-utilization-threshold	Auslastungsgrad des Knotens, definiert als Summe der angeforderten Ressourcen geteilt durch die Kapazität. Unterschreitet dieser Wert, kann ein Knoten herunterskaliert werden	0.5

Parameter	Description	Standard
scale-down-non-empty-Anzahl der Kandidaten	Maximale Anzahl nicht leerer Knoten, die in einer Iteration als Kandidaten für eine Herunterskalierung mit Drain betrachtet werden. Ein niedrigerer Wert bedeutet eine bessere Reaktionsfähigkeit der CA, aber möglicherweise eine langsamere Scale-Dow n-Latenz. Ein höherer Wert kann sich auf die Leistung der CA bei großen Clustern (Hunderte von Knoten) auswirken. Setzen Sie diesen Wert auf einen nicht positiven Wert, um diese Heuristik auszuschalten. CA begrenzt die Anzahl der berücksic htigten Knoten nicht. “	30

Parameter	Description	Standard
scale-down-candidates-pool-verhältnis	Ein Verhältnis von Knoten, die als zusätzliche, nicht leere Kandidaten für eine Verkleine rung betrachtet werden, wenn einige Kandidaten aus einer vorherigen Iteration nicht mehr gültig sind. Ein niedriger er Wert bedeutet eine bessere Reaktionsfähigkeit der CA, aber möglicherweise eine langsamere Reduzierung der Latenz. Ein höherer Wert kann sich auf die Leistung der CA bei großen Clustern (Hunderte von Knoten) auswirken. Setzen Sie den Wert auf 1,0, um diese Heuristik auszuschalten. CA nimmt alle Knoten als zusätzliche Kandidaten.	0.1

Parameter	Description	Standard
scale-down-candidates-pool-min-Anzahl	Mindestanzahl von Knoten, die als zusätzliche, nicht leere Kandidaten für das Herunters kalieren betrachtet werden, wenn einige Kandidaten aus früheren Iterationen nicht mehr gültig sind. Bei der Berechnung der Poolgröße für zusätzliche Kandidaten nehmen wir `max(#nodes * scale-down-candidates-pool-ratio, scale-down-candidates-pool-min-count)`	50

Weitere Ressourcen

Diese Seite enthält eine Liste der Präsentationen und Demos von Cluster Autoscaler. Wenn Sie hier eine Präsentation oder Demo hinzufügen möchten, senden Sie bitte eine Pull-Anfrage.

Präsentation/Demo	Moderatoren
Autoscaling und Kostenoptimierung auf Kubernetes: Von 0 auf 100	Guy Templeton, Skyscanner und Jiaxin Shan, Amazon
Tiefer Einblick in SIG-Autoscaling	Maciek Pytel und Marcin Wielgus

Referenzen

- 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/Amazon-EC2-Instanzenauswahl

- https://github.com/aws/aws-node-termination-handler

Bewährte Methoden für Zuverlässigkeit

Dieser Abschnitt enthält Anleitungen dazu, wie Sie Workloads, die auf EKS ausgeführt werden, robust und hochverfügbar machen

Verwendung dieses Leitfadens

Dieser Leitfaden richtet sich an Entwickler und Architekten, die hochverfügbare und fehlertolerante Dienste in EKS entwickeln und betreiben möchten. Der Leitfaden ist zur leichteren Nutzung in verschiedene Themenbereiche unterteilt. Jedes Thema beginnt mit einem kurzen Überblick, gefolgt von einer Liste mit Empfehlungen und bewährten Methoden für die Zuverlässigkeit Ihrer EKS-Cluster.

Einführung

Die bewährten Methoden zur Zuverlässigkeit von EKS wurden unter den folgenden Themen zusammengefasst:

- Anwendungen
- Steuerebene
- Datenebene

Was macht ein System zuverlässig? Wenn ein System trotz Veränderungen in seiner Umgebung im Laufe der Zeit konsistent funktioniert und die Anforderungen erfüllt, kann es als zuverlässig bezeichnet werden. Um dies zu erreichen, muss das System Fehler erkennen, sich automatisch selbst reparieren und in der Lage sein, je nach Bedarf zu skalieren.

Kunden können Kubernetes als Grundlage für den zuverlässigen Betrieb unternehmenskritischer Anwendungen und Dienste nutzen. Aber abgesehen von den Prinzipien des containerbasierten Anwendungsdesigns erfordert die zuverlässige Ausführung von Workloads auch eine zuverlässige Infrastruktur. In Kubernetes umfasst die Infrastruktur die Steuerungsebene und die Datenebene.

EKS bietet eine Kubernetes-Steuerebene in Produktionsqualität, die so konzipiert ist, dass sie hochverfügbar und fehlertolerant ist.

In EKS ist AWS für die Zuverlässigkeit der Kubernetes-Steuerebene verantwortlich. EKS führt die Kubernetes-Steuerebene in drei Availability Zones in einer AWS-Region aus. Es verwaltet automatisch die Verfügbarkeit und Skalierbarkeit der Kubernetes-API-Server und des etcd-Clusters.

Sie, der Kunde und AWS tragen gemeinsam die Verantwortung für die Zuverlässigkeit der Datenebene. EKS bietet vier Worker-Node-Optionen für die Bereitstellung der Kubernetes-Datenebene.

Der automatische Modus von EKS, die am häufigsten verwaltete Option, übernimmt die Bereitstellung, Skalierung und Aktualisierung der Datenebene sowie die Bereitstellung verwalteter Rechen-, Netzwerk- und Speicherfunktionen. Der Auto-Modus AMIs wird häufig veröffentlicht und Cluster werden automatisch auf das neueste AMI aktualisiert, um CVE-Fixes und Sicherheitspatches bereitzustellen. Sie haben die Möglichkeit, zu kontrollieren, wann dies geschieht, indem Sie in Ihrem Automodus NodePools die Störungskontrolle konfigurieren.

Fargate kümmert sich um die Bereitstellung und Skalierung der Datenebene, indem es einen Pod pro Knoten ausführt. Die dritte Option, verwaltete Knotengruppen, kümmert sich um die Bereitstellung und Aktualisierung der Datenebene. Und schließlich sind selbstverwaltete Knoten die am wenigsten verwaltete Option für die Datenebene. Je mehr AWS-verwaltete Datenebene Sie verwenden, desto weniger Verantwortung haben Sie.

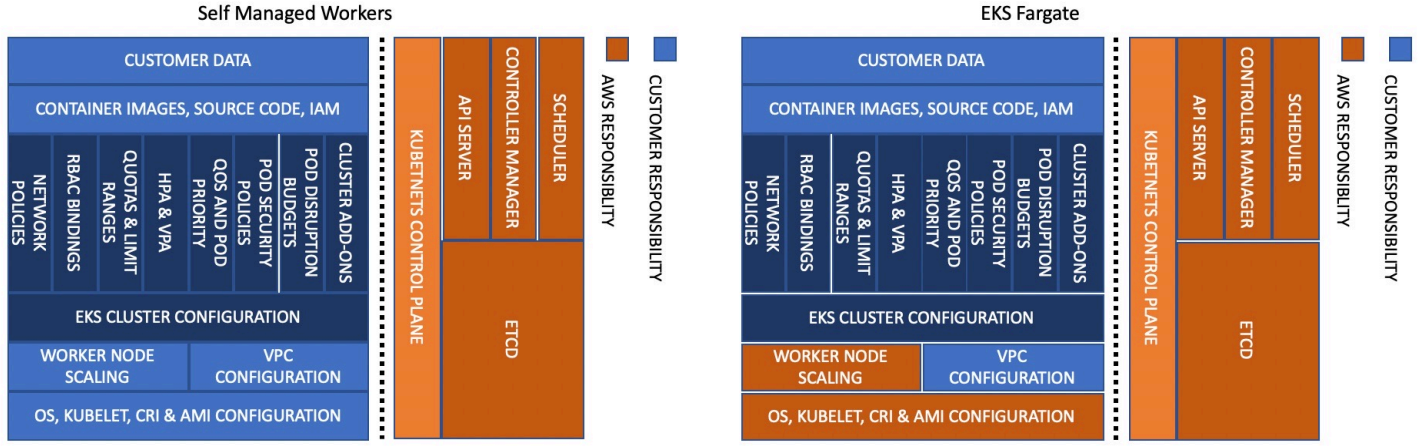
Verwaltete Knotengruppen automatisieren die Bereitstellung und das Lebenszyklusmanagement von EC2-Knoten. Sie können die EKS-API (mithilfe der EKS-Konsole, AWS-API, AWS-CLICloudFormation, Terraform oder`eksctl`) verwenden, um verwaltete Knoten zu erstellen, zu skalieren und zu aktualisieren. Auf verwalteten Knoten werden EKS-optimierte Amazon Linux 2 EC2-Instances in Ihrem Konto ausgeführt, und Sie können benutzerdefinierte Softwarepakete installieren, indem Sie den SSH-Zugriff aktivieren. Wenn Sie verwaltete Knoten bereitstellen, werden sie als Teil einer von EKS verwalteten Auto Scaling-Gruppe ausgeführt, die sich über mehrere Availability Zones erstrecken kann. Sie steuern dies über die Subnetze, die Sie bei der Erstellung verwalteter Knoten angeben. EKS kennzeichnet verwaltete Knoten außerdem automatisch, sodass sie mit Cluster Autoscaler verwendet werden können.

Amazon EKS folgt dem Modell der gemeinsamen Verantwortung für CVEs und Sicherheitspatches auf verwalteten Knotengruppen. Da auf verwalteten Knoten die für Amazon EKS optimierte Version ausgeführt wird AMIs, ist Amazon EKS dafür verantwortlich, gepatchte Versionen davon zu erstellen, AMIs wenn Fehler behoben werden. Sie sind jedoch dafür verantwortlich, diese gepatchten AMI-Versionen für Ihre verwalteten Knotengruppen bereitzustellen.

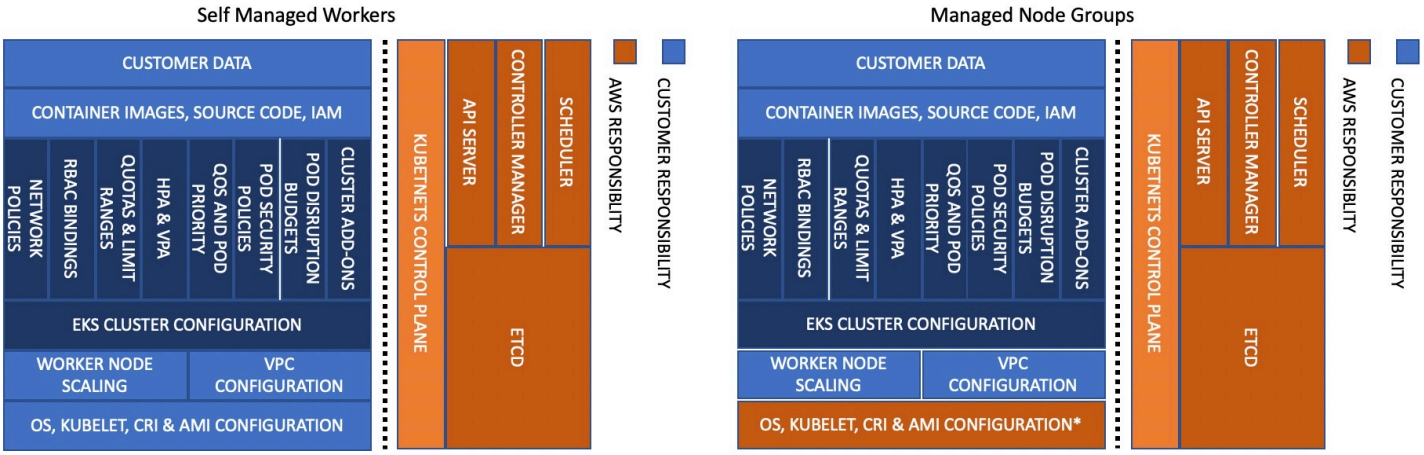
EKS verwaltet auch die Aktualisierung der Knoten, obwohl Sie den Aktualisierungsvorgang einleiten müssen. Der Prozess der Aktualisierung des verwalteten Knotens wird in der EKS-Dokumentation erklärt.

Wenn Sie selbstverwaltete Knoten ausführen, können Sie das Amazon EKS-optimierte Linux-AMI verwenden, um Worker-Knoten zu erstellen. Sie sind für das Patchen und Aktualisieren des AMI und der Knoten verantwortlich. Es hat sich bewährt `eksctl` CloudFormation, Infrastruktur als Codetools für die Bereitstellung von selbstverwalteten Knoten zu verwenden, da Sie auf diese Weise das Upgrade von selbstverwalteten Knoten problemlos durchführen können. Erwägen Sie bei der Aktualisierung von Worker-Knoten die Migration auf neue Knoten, da der Migrationsprozess die alte Knotengruppe unverändert lässt **`NoSchedule`** und die Knoten überlastet, sobald ein neuer Stack bereit ist, die bestehende Pod-Arbeitslast zu akzeptieren. Sie können jedoch auch ein direktes Upgrade von selbstverwalteten Knoten durchführen.

Modell der geteilten Verantwortung - Fargate



Modell der geteilten Verantwortung — MNG



Dieser Leitfaden enthält eine Reihe von Empfehlungen, mit denen Sie die Zuverlässigkeit Ihrer EKS-Datenebene, der Kubernetes-Kernkomponenten und Ihrer Anwendungen verbessern können.

Feedback

Dieser Leitfaden wird am veröffentlicht, GitHub um direktes Feedback und Vorschläge von der breiteren EKS/Kubernetes Community zu sammeln. Wenn Sie eine bewährte Methode haben, die wir Ihrer Meinung nach in den Leitfaden aufnehmen sollten, reichen Sie bitte ein Problem ein oder reichen Sie eine PR im GitHub Repository ein. Wir beabsichtigen, den Leitfaden regelmäßig zu aktualisieren, sobald der Service um neue Funktionen erweitert wird oder wenn sich eine neue bewährte Methode herausstellt.

Ausführung hochverfügbarer Anwendungen

Ihre Kunden erwarten, dass Ihre Anwendung immer verfügbar ist, auch wenn Sie Änderungen vornehmen, und insbesondere bei Verkehrsspitzen. Eine skalierbare und belastbare Architektur sorgt dafür, dass Ihre Anwendungen und Dienste unterbrechungsfrei laufen, sodass Ihre Benutzer zufrieden sind. Eine skalierbare Infrastruktur wächst und schrumpft je nach den Anforderungen des Unternehmens. Die Eliminierung einzelner Fehlerquellen ist ein entscheidender Schritt, um die Verfügbarkeit einer Anwendung zu verbessern und sie widerstandsfähiger zu machen.

Mit Kubernetes können Sie Ihre Anwendungen betreiben und sie hochverfügbar und belastbar ausführen. Die deklarative Verwaltung stellt sicher, dass Kubernetes, sobald Sie die Anwendung eingerichtet haben, kontinuierlich versucht, den aktuellen Status mit dem gewünschten Status abzugleichen.

Empfehlungen

Konfigurieren Sie Pod-Disruption-Budgets

Budgets für Pod-Unterbrechungen werden verwendet, um die Anzahl der gleichzeitigen Unterbrechungen zu begrenzen, denen eine Anwendung ausgesetzt sein wird. Sie sollten für Workloads konfiguriert werden, wenn es wichtig ist, dass immer ein Teil dieser Workloads verfügbar ist. EKS Auto Mode, Karpenter und Cluster Autoscaler kennen die konfigurierten Pod-Disruption-Budgets und halten sich bei der Herunterskalierung daran. EKS Auto Mode, Karpenter und Managed Node Groups halten sich bei der Aktualisierung von Nodes auch an die Pod-Disruption-Budgets

Vermeiden Sie es, Singleton-Pods auszuführen

Wenn Ihre gesamte Anwendung in einem einzigen Pod ausgeführt wird, ist Ihre Anwendung nicht verfügbar, wenn dieser Pod beendet wird. Anstatt Anwendungen mithilfe einzelner Pods bereitzustellen, erstellen Sie Bereitstellungen. Wenn ein durch eine Bereitstellung erstellter Pod

ausfällt oder beendet wird, startet der Deployment Controller einen neuen Pod, um sicherzustellen, dass die angegebene Anzahl von Replikat-Pods immer ausgeführt wird.

Führen Sie mehrere Replikate aus

Wenn Sie mehrere Replikate ausführen (Pods) einer App mithilfe eines Deployment, kann diese auf hochverfügbare Weise ausgeführt werden. Wenn ein Replikat ausfällt, funktionieren die verbleibenden Replikate weiterhin, wenn auch mit reduzierter Kapazität, bis Kubernetes einen weiteren Pod erstellt, um den Verlust auszugleichen. Darüber hinaus können Sie den Horizontal Pod Autoscaler verwenden, um Replikate automatisch auf der Grundlage der Workload-Nachfrage zu skalieren.

Planen Sie Replikate knotenübergreifend

Das Ausführen mehrerer Replikate ist nicht sehr nützlich, wenn alle Replikate auf demselben Knoten ausgeführt werden und der Knoten nicht mehr verfügbar ist. Erwägen Sie die Verwendung von Pod-Anti-Affinitäts- oder Pod-Topologie-Verbreitungsbeschränkungen, um Replikate einer Bereitstellung auf mehrere Worker-Knoten zu verteilen.

Sie können die Zuverlässigkeit einer typischen Anwendung weiter verbessern, indem Sie sie auf mehreren ausführen. AZs

Verwendung von Pod-Anti-Affinitätsregeln

Das folgende Manifest weist den Kubernetes-Scheduler an, Pods lieber auf separaten Knoten und zu platzieren. AZs Es sind keine eigenen Knoten oder AZ erforderlich, denn wenn dies der Fall wäre, kann Kubernetes keine Pods planen, sobald in jeder AZ ein Pod läuft. Wenn Ihre Anwendung nur drei Replikate benötigt, können Sie `requiredDuringSchedulingIgnoredDuringExecution` for verwenden`topologyKey: topology.kubernetes.io/zone`, und der Kubernetes-Scheduler plant nicht zwei Pods in derselben AZ.

```
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
```

Verbreitungsbeschränkungen mithilfe der Pod-Topologie

Ähnlich wie bei Pod-Anti-Affinitätsregeln können Sie mit Einschränkungen für die Verteilung der Pod-Topologie Ihre Anwendung für verschiedene Fehler- (oder Topologie-) Domänen wie Hosts oder verfügbar machen. AZs Dieser Ansatz funktioniert sehr gut, wenn Sie versuchen, sowohl Fehlertoleranz als auch Verfügbarkeit sicherzustellen, indem Sie mehrere Replikate in jeder der verschiedenen Topologiedomänen verwenden. Pod-Anti-Affinitätsregeln können dagegen leicht zu Ergebnissen führen, wenn Sie ein einzelnes Replikat in einer Topologiedomäne haben, da die Pods mit einer Anti-Affinität zueinander eine abstoßende Wirkung haben. In solchen Fällen ist ein einzelnes Replikat auf einem dedizierten Knoten weder ideal für die Fehlertoleranz noch für eine sinnvolle Nutzung von Ressourcen. Mit Einschränkungen der Topologieverteilung haben Sie mehr Kontrolle

über die Verteilung oder Verteilung, die der Scheduler auf die Topologiedomänen anwenden sollte. Hier sind einige wichtige Eigenschaften, die bei diesem Ansatz verwendet werden sollten:

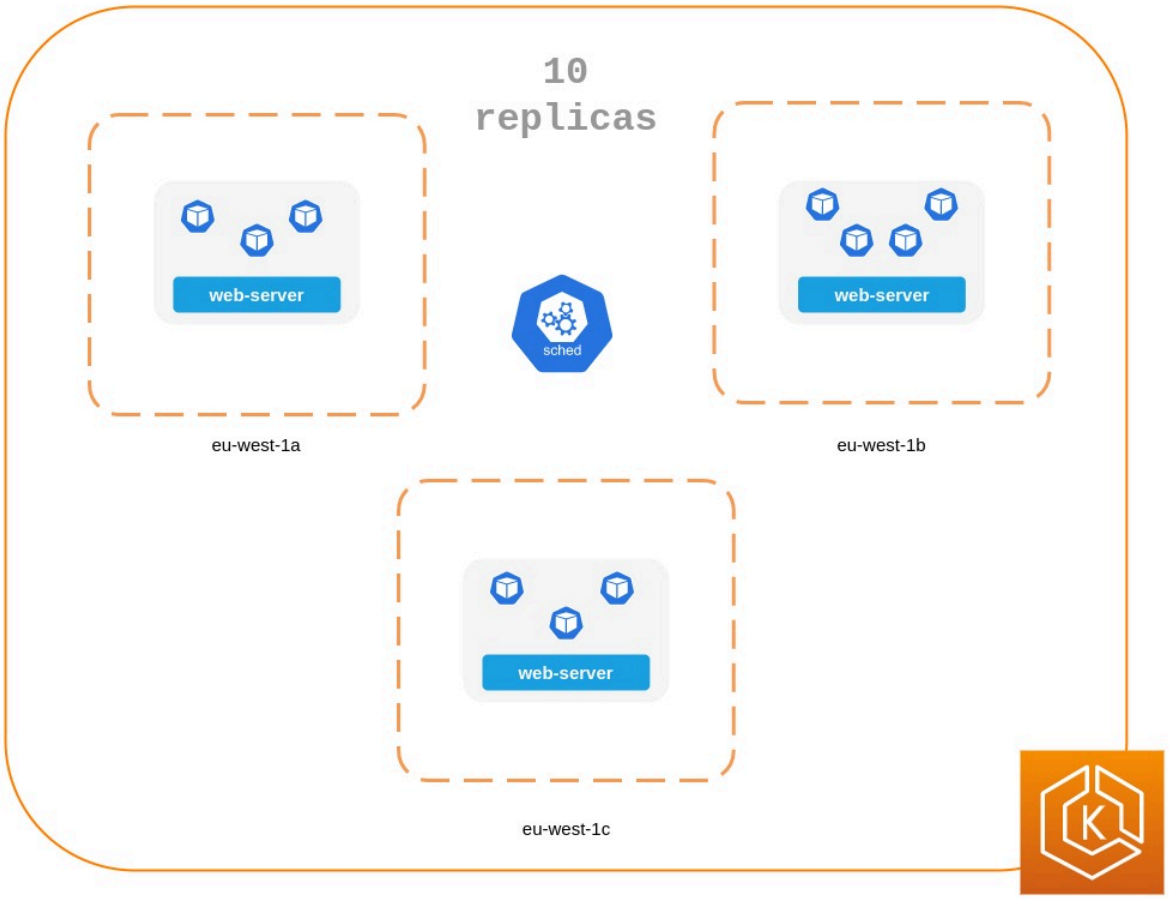
1. Der `maxSkew` wird verwendet, um den maximalen Punkt zu steuern oder zu bestimmen, bis zu dem die Dinge in den Topologiedomänen ungleichmäßig sein können. Wenn eine Anwendung beispielsweise über 10 Replikate verfügt und auf 3 verteilt ist AZs, können Sie zwar keine gleichmäßige Verteilung erzielen, aber Sie können beeinflussen, wie ungleichmäßig die Verteilung sein wird. In diesem Fall `maxSkew` kann es sich um einen Wert zwischen 1 und 10 handeln. Ein Wert von 1 bedeutet, dass Sie am Ende möglicherweise eine Streuung wie `4,3,3` `3,4,3` oder `3,3,4` über die 3 erhalten können AZs. Im Gegensatz dazu bedeutet ein Wert von 10, dass Sie möglicherweise eine Streuung von etwa `10,0,0` `0,10,0` oder `0,0,10` über 3 erhalten können AZs.
2. Das `topologyKey` ist ein Schlüssel für eine der Knotenbezeichnungen und definiert den Typ der Topologiedomäne, die für die Pod-Verteilung verwendet werden soll. Ein zonaler Spread hätte beispielsweise das folgende Schlüssel-Wert-Paar:

   ```
   topologyKey: "topology.kubernetes.io/zone"
   ```

3. Die `whenUnsatisfiable` Eigenschaft wird verwendet, um zu bestimmen, wie der Scheduler reagieren soll, wenn die gewünschten Einschränkungen nicht erfüllt werden können.
4. Die `labelSelector` wird verwendet, um passende Pods zu finden, sodass der Scheduler sie erkennen kann, wenn er entscheidet, wo Pods gemäß den von Ihnen angegebenen Einschränkungen platziert werden sollen.

Zusätzlich zu den oben genannten gibt es weitere Felder, über die Sie in der Kubernetes-Dokumentation weiter lesen können.

Die Pod-Topologie verteilt sich auf 3 Einschränkungen 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
```

Führen Sie den Kubernetes Metrics Server aus

Installieren Sie den Kubernetes Metrics Server, um Ihre Anwendungen zu skalieren. Kubernetes-Autoscaler-Add-ons wie HPA und VPA müssen die Metriken von Anwendungen verfolgen, um sie skalieren zu können. Der Metrics-Server sammelt Ressourcenmetriken, die verwendet werden können, um Skalierungsentscheidungen zu treffen. Die Metriken werden von Kubelets gesammelt und im Metrics-API-Format bereitgestellt.

Der Metrik-Server speichert keine Daten und ist keine Überwachungslösung. Sein Zweck besteht darin, Metriken zur CPU- und Speichernutzung anderen Systemen zugänglich zu machen. Wenn Sie den Status Ihrer Anwendung im Laufe der Zeit verfolgen möchten, benötigen Sie ein Überwachungstool wie Prometheus oder Amazon. CloudWatch

Folgen Sie der EKS-Dokumentation, um den Metrics-Server in Ihrem EKS-Cluster zu installieren.

Horizontaler Pod-Autoscaler (HPA)

HPA kann Ihre Anwendung je nach Bedarf automatisch skalieren und Ihnen helfen, zu vermeiden, dass Ihre Kunden zu Spitzenzeiten beeinträchtigt werden. Es ist als Regelkreis in Kubernetes implementiert, der in regelmäßigen Abständen Metriken abfragt, die Ressourcenmetriken APIs bereitstellen.

HPA kann Metriken aus folgenden APIs Quellen abrufen: 1. `metrics.k8s.io`auch bekannt als Resource Metrics API — Stellt die CPU- und Speicherauslastung für Pods 2 bereit. `custom.metrics.k8s.io` — Stellt Metriken von anderen Metriksammlern wie Prometheus bereit. Diese Metriken sind intern in Ihrem Kubernetes-Cluster. 3. `external.metrics.k8s.io` — Stellt Metriken bereit, die sich außerhalb Ihres Kubernetes-Clusters befinden (z. B. SQS Queue Depth, ELB-Latenz).

Sie müssen eine dieser drei Optionen verwenden, APIs um die Metrik für die Skalierung Ihrer Anwendung bereitzustellen.

Skalierung von Anwendungen auf der Grundlage benutzerdefinierter oder externer Metriken

Sie können benutzerdefinierte oder externe Metriken verwenden, um Ihre Anwendung anhand anderer Messwerte als der CPU- oder Speicherauslastung zu skalieren. API-Server für benutzerdefinierte Metriken stellen die `custom-metrics.k8s.io` API bereit, mit der HPA Anwendungen automatisch skalieren kann.

Sie können den Prometheus-Adapter für Kubernetes-Metriken verwenden, APIs um Metriken von Prometheus zu sammeln und mit der HPA zu verwenden. In diesem Fall stellt der Prometheus-Adapter Prometheus-Metriken im Metrics-API-Format zur Verfügung.

Sobald Sie den Prometheus-Adapter bereitgestellt haben, können Sie benutzerdefinierte Metriken mit kubectl abfragen. `kubectl get –raw /apis/custom.metrics.k8s.io/v1beta1/`

Externe Metriken bieten, wie der Name schon sagt, dem Horizontal Pod Autoscaler die Möglichkeit, Bereitstellungen mithilfe von Metriken zu skalieren, die sich außerhalb des Kubernetes-Clusters befinden. Bei Workloads zur Stapelverarbeitung ist es beispielsweise üblich, die Anzahl der Replikate auf der Grundlage der Anzahl der laufenden Jobs in einer SQS-Warteschlange zu skalieren.

Um Kubernetes-Workloads automatisch zu skalieren, können Sie KEDA (Kubernetes Event-Driven Autoscaling) verwenden, ein Open-Source-Projekt, das die Container-Skalierung auf der Grundlage einer Reihe von benutzerdefinierten Ereignissen vorantreiben kann. In diesem AWS-Blog wird beschrieben, wie Amazon Managed Service for Prometheus für die auto-scaling von Kubernetes-Workloads verwendet wird.

Vertikaler Pod-Autoscaler (VPA)

VPA passt die CPU- und Speicherreservierung für Ihre Pods automatisch an, damit Sie Ihre Anwendungen „richtig dimensionieren“ können. Für Anwendungen, die vertikal skaliert werden müssen — was durch Erhöhung der Ressourcenzuweisung erreicht wird — können Sie VPA verwenden, um Pod-Replikate automatisch zu skalieren oder Skalierungsempfehlungen zu geben.

Ihre Anwendung ist möglicherweise vorübergehend nicht verfügbar, wenn VPA sie skalieren muss, da die aktuelle Implementierung von VPA keine direkten Anpassungen an Pods vornimmt. Stattdessen wird der Pod, der skaliert werden muss, neu erstellt.

Die EKS-Dokumentation enthält eine Anleitung zur Einrichtung von VPA.

Das Fairwinds Goldilocks-Projekt bietet ein Dashboard zur Visualisierung von VPA-Empfehlungen für CPU- und Speicheranforderungen und Grenzwerte. Im VPA-Aktualisierungsmodus können Sie Pods auf der Grundlage von VPA-Empfehlungen automatisch skalieren.

Anwendungen werden aktualisiert

Moderne Anwendungen erfordern schnelle Innovationen mit einem hohen Maß an Stabilität und Verfügbarkeit. Kubernetes bietet Ihnen die Tools, mit denen Sie Ihre Anwendungen kontinuierlich aktualisieren können, ohne Ihre Kunden zu stören.

Schauen wir uns einige der Best Practices an, die es ermöglichen, Änderungen schnell umzusetzen, ohne die Verfügbarkeit zu beeinträchtigen.

Haben Sie einen Mechanismus zur Durchführung von Rollbacks

Mit einer Schaltfläche zum Rückgängigmachen können Katastrophen vermieden werden. Es hat sich bewährt, Bereitstellungen in einer separaten unteren Umgebung (Test- oder Entwicklungsumgebung) zu testen, bevor der Produktionscluster aktualisiert wird. Mithilfe einer CI/CD-Pipeline können Sie Bereitstellungen automatisieren und testen. Mit einer kontinuierlichen Bereitstellungspipeline können Sie schnell zur älteren Version zurückkehren, falls das Upgrade defekt sein sollte.

Sie können Deployments verwenden, um eine laufende Anwendung zu aktualisieren. Dies erfolgt in der Regel durch die Aktualisierung des Container-Images. Sie können es verwendenkubectl, um ein Deployment wie folgt zu aktualisieren:

```
kubectl --record deployment.apps/nginx-deployment set image nginx-deployment
 nginx=nginx:1.16.1
```

Das `--record` Argument zeichnet die Änderungen am Deployment auf und hilft Ihnen, wenn Sie ein Rollback durchführen müssen. `kubectl rollout history deployment`zeigt Ihnen die aufgezeichneten Änderungen an den Bereitstellungen in Ihrem Cluster. Sie können eine Änderung rückgängig machen mit. `kubectl rollout undo deployment <DEPLOYMENT_NAME>`

Wenn Sie ein Deployment aktualisieren, das eine Neuerstellung der Pods erfordert, führt Deployment standardmäßig ein fortlaufendes Update durch. Mit anderen Worten, Kubernetes aktualisiert nur einen Teil der laufenden Pods in einem Deployment und nicht alle Pods gleichzeitig. Sie können mithilfe von Property steuern, wie Kubernetes fortlaufende Updates durchführt. `RollingUpdateStrategy`

Wenn Sie ein fortlaufendes Update eines Deployments durchführen, können Sie die `Max Unavailable`Eigenschaft verwenden, um die maximale Anzahl von Pods anzugeben, die während des Updates nicht verfügbar sein können. Mit der `Max Surge` Eigenschaft Deployment können Sie die maximale Anzahl von Pods festlegen, die über die gewünschte Anzahl von Pods hinweg erstellt werden können.

Erwägen Sie Anpassungen, `max unavailable` um sicherzustellen, dass ein Rollout Ihre Kunden nicht stört. Kubernetes legt beispielsweise `max unavailable` standardmäßig 25% fest, was bedeutet, dass bei 100 Pods möglicherweise nur 75 Pods während eines Rollouts aktiv sind. Wenn Ihre Anwendung mindestens 80 Pods benötigt, kann dieser Rollout störend sein. Stattdessen können Sie einen Wert von 20% festlegen`max unavailable`, um sicherzustellen, dass während des gesamten Rollouts mindestens 80 funktionsfähige Pods vorhanden sind.

Verwenden Sie Bereitstellungen blue/green

Änderungen sind von Natur aus riskant, aber Änderungen, die nicht rückgängig gemacht werden können, können potenziell katastrophal sein. Änderungsverfahren, mit denen Sie die Zeit durch ein Rollback effektiv zurückdrehen können, machen Verbesserungen und Experimente sicherer. Blue/green Bereitstellungen bieten Ihnen eine Methode, um die Änderungen schnell rückgängig zu machen, falls etwas schief geht. In dieser Bereitstellungsstrategie erstellen Sie eine Umgebung für die neue Version. Diese Umgebung ist identisch mit der aktuellen Version der Anwendung, die aktualisiert wird. Sobald die neue Umgebung bereitgestellt ist, wird der Datenverkehr an die neue Umgebung weitergeleitet. Wenn die neue Version die gewünschten Ergebnisse liefert, ohne Fehler zu generieren, wird die alte Umgebung beendet. Andernfalls wird der Verkehr auf die alte Version zurückgesetzt.

Sie können blue/green Bereitstellungen in Kubernetes durchführen, indem Sie ein neues Deployment erstellen, das mit dem Deployment der vorhandenen Version identisch ist. Sobald Sie sichergestellt haben, dass die Pods im neuen Deployment fehlerfrei ausgeführt werden, können Sie damit beginnen, Traffic an das neue Deployment zu senden, indem Sie die `selector` Spezifikation im Service ändern, der den Datenverkehr an die Pods Ihrer Anwendung weiterleitet.

Mit vielen Tools für die kontinuierliche Integration wie Flux, Jenkins und Spinnaker können Sie Bereitstellungen automatisieren. blue/green Der AWS-Container-Blog enthält eine Anleitung zur Verwendung von AWS Load Balancer Controller: Verwenden von AWS Load Balancer Controller für blue/green Bereitstellung, Bereitstellung und Testen auf Canary A/B

Verwenden Sie kanarische Bereitstellungen

Bei Bereitstellungen auf Kanaren handelt es sich um eine Variante von blue/green Bereitstellungen, mit denen das Risiko von Änderungen erheblich verringert werden kann. Bei dieser Bereitstellungsstrategie erstellen Sie ein neues Deployment mit weniger Pods neben Ihrem alten Deployment und leiten einen kleinen Teil des Traffics auf das neue Deployment um. Wenn die Messwerte darauf hindeuten, dass die neue Version genauso gut oder besser funktioniert als die bestehende Version, erhöhen Sie den Datenverkehr zur neuen Bereitstellung schrittweise und skalieren sie gleichzeitig, bis der gesamte Datenverkehr auf die neue Bereitstellung umgeleitet wird. Wenn ein Problem auftritt, können Sie den gesamten Datenverkehr an das alte Deployment weiterleiten und das Senden von Traffic an das neue Deployment beenden.

Kubernetes bietet zwar keine native Möglichkeit, kanarische Bereitstellungen durchzuführen, aber Sie können Tools wie Flagger mit Istio verwenden.

Gesundheitschecks und Selbstheilung

Keine Software ist fehlerfrei, aber Kubernetes kann Ihnen helfen, die Auswirkungen von Softwareausfällen zu minimieren. In der Vergangenheit musste jemand, wenn eine Anwendung abstürzte, die Situation beheben, indem er die Anwendung manuell neu startete. Kubernetes bietet Ihnen die Möglichkeit, Softwarefehler in Ihren Pods zu erkennen und sie automatisch durch neue Replikate zu ersetzen. Mit Kubernetes können Sie den Zustand Ihrer Anwendungen überwachen und fehlerhafte Instanzen automatisch ersetzen.

Kubernetes unterstützt drei Arten von Zustandsprüfungen:

1. Lebendigkeitstest
2. Startprobe (unterstützt in Kubernetes Version 1.16+)
3. Bereitschaftstest

Kubelet, der Kubernetes-Agent, ist für die Ausführung aller oben genannten Prüfungen verantwortlich. Kubelet kann den Zustand eines Pods auf drei Arten überprüfen: Kubelet kann entweder einen Shell-Befehl im Container eines Pods ausführen, eine HTTP-GET-Anfrage an seinen Container senden oder einen TCP-Socket an einem bestimmten Port öffnen.

Wenn Sie sich für eine *exec* basierte Probe entscheiden, die ein Shell-Skript in einem Container ausführt, stellen Sie sicher, dass der Shell-Befehl beendet wird, bevor der Wert abläuft.

`timeoutSeconds` Andernfalls wird Ihr Knoten über `<defunct>` Prozesse verfügen, was zu einem Ausfall des Knotens führt.

Empfehlungen

Verwenden Sie Liveness Probe, um ungesunde Pods zu entfernen

Der Liveness Test kann Deadlock-Bedingungen erkennen, bei denen der Prozess zwar weiter ausgeführt wird, die Anwendung aber nicht mehr reagiert. Wenn Sie beispielsweise einen Webdienst ausführen, der Port 80 überwacht, können Sie einen Liveness-Test so konfigurieren, dass er eine HTTP-GET-Anforderung an den Port 80 des Pods sendet. Kubelet sendet regelmäßig eine GET-Anfrage an den Pod und erwartet eine Antwort. Wenn der Pod zwischen 200 und 399 antwortet, geht das Kubelet davon aus, dass der Pod fehlerfrei ist. Andernfalls wird der Pod als fehlerhaft markiert. Wenn ein Pod die Integritätsprüfungen kontinuierlich nicht besteht, beendet ihn das Kubelet.

Sie können es verwenden, `initialDelaySeconds` um die erste Sonde zu verzögern.

Wenn Sie die Liveness Probe verwenden, stellen Sie sicher, dass Ihre Anwendung nicht in eine Situation gerät, in der alle Pods gleichzeitig die Liveness Probe nicht bestehen, da Kubernetes versucht, alle Ihre Pods zu ersetzen, wodurch Ihre Anwendung offline geschaltet wird. Darüber hinaus wird Kubernetes weiterhin neue Pods erstellen, bei denen auch Liveness Probes nicht funktionieren, was die Kontrollebene unnötig belastet. Vermeiden Sie es, die Liveness Probe so zu konfigurieren, dass sie von einem Faktor abhängt, der außerhalb Ihres Pods liegt, z. B. einer externen Datenbank. Mit anderen Worten: Eine external-to-your-Pod Datenbank, die nicht reagiert, sollte nicht dazu führen, dass Ihre Pods ihre Liveness Probes nicht bestehen.

Der Beitrag LIVENESS PROBES ARE DANGEROUS von Sandor Szücs beschreibt Probleme, die durch falsch konfigurierte Sonden verursacht werden können.

Verwenden Sie Startup Probe für Anwendungen, deren Start länger dauert

Wenn Ihre App zusätzliche Zeit zum Starten benötigt, können Sie die Startup Probe verwenden, um den Liveness and Readiness Test zu verzögern. Beispielsweise kann eine Java-App, die den Cache aus einer Datenbank hydratisieren muss, bis zu zwei Minuten dauern, bis sie voll funktionsfähig ist. Jede Liveness- oder Readiness Probe kann fehlschlagen, bis sie voll funktionsfähig ist. Durch die Konfiguration einer Startup Probe kann die Java-App wieder funktionsfähig sein, bevor Liveness oder Readiness Probe ausgeführt werden.

Bis die Startup Probe erfolgreich ist, sind alle anderen Probes deaktiviert. Sie können festlegen, wie lange Kubernetes maximal auf den Start der Anwendung warten soll. Wenn der Pod nach der

konfigurierten Höchstzeit die Startup Probes immer noch nicht bestanden hat, wird er beendet und ein neuer Pod wird erstellt.

Die Startup Probe ähnelt der Liveness Probe — wenn sie fehlschlagen, wird der Pod neu erstellt. Wie Ricardo A. in seinem Beitrag Fantastic Probes And How To Configure Them erklärt, sollten Startup Probes verwendet werden, wenn die Startzeit einer Anwendung nicht vorhersehbar ist. Wenn Sie wissen, dass Ihre Anwendung zehn Sekunden benötigt, um zu starten, sollten Sie stattdessen Liveness/Readiness Probe with verwenden. `initialDelaySeconds`

Verwenden Sie Readiness Probe, um eine teilweise Nichtverfügbarkeit festzustellen

Während der Liveness Test Fehler in einer App erkennt, die durch das Beenden des Pods (also durch einen Neustart der App) behoben werden, erkennt Readiness Probe Situationen, in denen die App möglicherweise vorübergehend nicht verfügbar ist. In diesen Situationen reagiert die App möglicherweise vorübergehend nicht mehr. Es wird jedoch davon ausgegangen, dass sie wieder fehlerfrei ist, sobald dieser Vorgang abgeschlossen ist.

Beispielsweise können Anwendungen während intensiver I/O Festplattenoperationen vorübergehend nicht zur Bearbeitung von Anfragen zur Verfügung stehen. In diesem Fall ist das Beenden des Pods der Anwendung keine Lösung. Gleichzeitig können zusätzliche Anfragen, die an den Pod gesendet werden, fehlschlagen.

Sie können die Readiness Probe verwenden, um eine vorübergehende Nichtverfügbarkeit in Ihrer App zu erkennen und das Senden von Anfragen an den zugehörigen Pod zu beenden, bis dieser wieder funktionsfähig ist. Im Gegensatz zu Liveness Probe, bei der ein Ausfall zur Neuerstellung des Pods führen würde, würde ein fehlgeschlagener Readiness Probe bedeuten, dass der Pod keinen Traffic vom Kubernetes-Service empfängt. Wenn die Readiness Probe erfolgreich ist, empfängt der Pod wieder Datenverkehr vom Service.

Vermeiden Sie es, genau wie bei der Liveness Probe, Readiness Probes zu konfigurieren, die von einer externen Ressource des Pods abhängen (z. B. einer Datenbank). Hier ist ein Szenario, in dem eine schlecht konfigurierte Readiness dazu führen kann, dass die Anwendung nicht mehr funktioniert. Wenn die Readiness Probe eines Pods fehlschlägt, obwohl die Datenbank der App nicht erreichbar ist, fallen auch andere Pod-Replikate gleichzeitig aus, da sie dieselben Integritätsprüfkriterien verwenden. Wenn Sie den Test auf diese Weise einrichten, wird sichergestellt, dass die Readiness Probes des Pods immer dann fehlschlagen, wenn die Datenbank nicht verfügbar ist, und Kubernetes den Datenverkehr an alle Pods nicht mehr sendet.

Ein Nebeneffekt der Verwendung von Readiness Probes besteht darin, dass sie die für die Aktualisierung von Bereitstellungen benötigte Zeit verlängern können. Neue Replikate empfangen

keinen Datenverkehr, es sei denn, die Readiness Probes sind erfolgreich. Bis dahin empfangen alte Replikate weiterhin Datenverkehr.

Umgang mit Störungen

Pods haben eine begrenzte Lebensdauer. Selbst wenn Sie Pods mit langer Laufzeit haben, sollten Sie sicherstellen, dass die Pods zu gegebener Zeit korrekt beendet werden. Abhängig von Ihrer Upgrade-Strategie müssen Sie bei Kubernetes-Cluster-Upgrades möglicherweise neue Worker-Knoten erstellen. Dazu müssen alle Pods auf neueren Knoten neu erstellt werden. Durch die richtige Bearbeitung von Terminierungen und Budgets für Pod-Unterbrechungen können Sie Serviceunterbrechungen vermeiden, da Pods aus älteren Knoten entfernt und auf neueren Knoten neu erstellt werden.

Die bevorzugte Methode zur Aktualisierung von Worker-Knoten besteht darin, neue Worker-Knoten zu erstellen und alte zu beenden. Bevor Sie Worker-Knoten beenden, sollten `drain` Sie dies tun. Wenn ein Worker-Knoten leer ist, werden alle seine Pods sicher entfernt. Sicher ist hier ein Schlüsselwort. Wenn Pods auf einem Worker geräumt werden, wird ihnen nicht einfach ein Signal gesendet. `SIGKILL` Stattdessen wird ein `SIGTERM` Signal an den Hauptprozess (PID 1) jedes Containers in den Pods gesendet, der geräumt wird. Nachdem das `SIGTERM` Signal gesendet wurde, gibt Kubernetes dem Prozess einige Zeit (Gnadenfrist), bevor ein `SIGKILL` Signal gesendet wird. Diese Übergangszeit beträgt standardmäßig 30 Sekunden. Sie können die Standardeinstellung überschreiben, indem Sie `grace-period` flag in kubectl verwenden oder in Ihrer Podspec deklarieren`terminationGracePeriodSeconds`.

```
kubectl delete pod <pod name> –grace-period=<seconds>
```

Es ist üblich, Container zu haben, in denen der Hauptprozess nicht die PID 1 hat. Stellen Sie sich diesen Python-basierten Probenbehälter vor:

```
$ 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
```

In diesem Beispiel empfängt das Shell-Skript`SIGTERM`, der Hauptprozess, der in diesem Beispiel zufällig eine Python-Anwendung ist, erhält kein `SIGTERM` Signal. Wenn der Pod beendet wird, wird die Python-Anwendung abrupt beendet. Dies kann behoben werden, indem der `ENTRYPOINT`Container geändert wird, um die Python-Anwendung zu starten. Alternativ können Sie

ein Tool wie dumb-init verwenden, um sicherzustellen, dass Ihre Anwendung Signale verarbeiten kann.

Sie können Container-Hooks auch verwenden, um ein Skript oder eine HTTP-Anfrage beim Start oder Stopp eines Containers auszuführen. Die `PreStop` Hook-Aktion wird ausgeführt, bevor der Container ein `SIGTERM` Signal empfängt, und muss abgeschlossen sein, bevor dieses Signal gesendet wird. Der `terminationGracePeriodSeconds` Wert gilt ab dem Zeitpunkt, zu dem die `PreStop` Hook-Aktion ausgeführt wird, nicht ab dem Zeitpunkt, zu dem das `SIGTERM` Signal gesendet wird.

Empfehlungen

Schützen Sie kritische Workloads mit Pod Disruption Budgets

Pod Disruption Budget oder PDB können den Räumungsprozess vorübergehend unterbrechen, wenn die Anzahl der Replikate einer Anwendung unter den angegebenen Schwellenwert fällt. Der Räumungsprozess wird fortgesetzt, sobald die Anzahl der verfügbaren Replikate den Schwellenwert überschreitet. Sie können PDB verwenden, um die Anzahl `minAvailable` und `maxUnavailable` Anzahl der Replikate zu deklarieren. Wenn Sie beispielsweise möchten, dass mindestens drei Kopien Ihrer App verfügbar sind, können Sie eine PDB erstellen.

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: my-svc-pdb
spec:
  minAvailable: 3
  selector:
    matchLabels:
      app: my-svc
```

Die obige PDB-Richtlinie weist Kubernetes an, den Räumungsprozess anzuhalten, bis drei oder mehr Replikate verfügbar sind. Das `PodDisruptionBudgets` Entleeren von Knoten wird respektiert. Während eines Upgrades einer verwalteten EKS-Knotengruppe werden die Knoten mit einem Zeitlimit von fünfzehn Minuten entlastet. Wenn das Update nach fünfzehn Minuten nicht erzwungen wird (die Option wird in der EKS-Konsole als fortlaufendes Update bezeichnet), schlägt das Update fehl. Wenn das Update erzwungen wird, werden die Pods gelöscht.

Für selbstverwaltete Knoten können Sie auch Tools wie AWS Node Termination Handler verwenden, der sicherstellt, dass die Kubernetes-Steuerebene angemessen auf Ereignisse reagiert, die dazu

führen können, dass Ihre EC2-Instance nicht verfügbar ist, wie z. B. EC2-Wartungsereignisse und EC2-Spot-Unterbrechungen. Es verwendet die Kubernetes-API, um den Knoten zu sperren, um sicherzustellen, dass keine neuen Pods geplant werden. Anschließend wird er entleert und alle laufenden Pods beendet.

Sie können Pod-Anti-Affinität verwenden, um die Pods eines Deployments auf verschiedenen Knoten zu planen und PDB-bedingte Verzögerungen bei Knoten-Upgrades zu vermeiden.

Üben Sie sich in Chaos Engineering

Chaos Engineering ist die Disziplin des Experimentierens an einem verteilten System, um Vertrauen in die Fähigkeit des Systems aufzubauen, turbulenten Produktionsbedingungen standzuhalten.

In seinem Blog erklärt Dominik Tornow, dass Kubernetes ein deklaratives System ist, bei dem "der Benutzer dem System eine Darstellung des gewünschten Systemzustands zur Verfügung stellt. Das System berücksichtigt dann den aktuellen Status und den gewünschten Zustand, um die Reihenfolge der Befehle für den Übergang vom aktuellen Zustand in den gewünschten Zustand zu bestimmen. “ Das bedeutet, dass Kubernetes immer den gewünschten Status speichert und wenn das System davon abweicht, ergreift Kubernetes Maßnahmen, um den Status wiederherzustellen. Wenn beispielsweise ein Worker-Node nicht mehr verfügbar ist, verschiebt Kubernetes die Pods auf einen anderen Worker-Knoten. In ähnlicher Weise erstellt der Deployment Controller bei einem `replica` Absturz eine neue. `replica` Auf diese Weise beheben Kubernetes-Controller Fehler automatisch.

Chaos-Engineering-Tools wie Gremlin helfen Ihnen dabei, die Resilienz Ihres Kubernetes-Clusters zu testen und einzelne Fehlerquellen zu identifizieren. Tools, die für künstliches Chaos in Ihrem Cluster (und darüber hinaus) sorgen, können systemische Schwächen aufdecken, bieten die Möglichkeit, Engpässe und Fehlkonfigurationen zu identifizieren und Probleme in einer kontrollierten Umgebung zu beheben. Die Philosophie von Chaos Engineering besteht darin, Dinge absichtlich zu beschädigen und die Infrastruktur einem Stresstest zu unterziehen, um unerwartete Ausfallzeiten zu minimieren.

Verwenden Sie ein Service Mesh

Sie können ein Service Mesh verwenden, um die Ausfallsicherheit Ihrer Anwendung zu verbessern. Service Meshes ermöglichen die service-to-service Kommunikation und erhöhen die Beobachtbarkeit Ihres Microservices-Netzwerks. Die meisten Service Mesh-Produkte funktionieren so, dass neben jedem Dienst ein kleiner Netzwerk-Proxy läuft, der den Netzwerkverkehr der Anwendung abfängt und überprüft. Sie können Ihre Anwendung in einem Mesh platzieren, ohne Ihre Anwendung zu ändern. Mithilfe der integrierten Funktionen des Service-Proxys können Sie Netzwerkstatistiken

generieren, Zugriffsprotokolle erstellen und ausgehenden Anfragen HTTP-Header für die verteilte Ablaufverfolgung hinzufügen.

Ein Service Mesh kann Ihnen dabei helfen, Ihre Microservices mit Funktionen wie automatischen Wiederholungen von Anfragen, Timeouts, Unterbrechungen und Ratenbegrenzung widerstandsfähiger zu machen.

Wenn Sie mehrere Cluster betreiben, können Sie ein Service Mesh verwenden, um die clusterübergreifende Kommunikation zu ermöglichen. service-to-service

Service Meshes

- Istio
- Verlinkt D
- Konsul

Beobachtbarkeit

Observability ist ein Überbegriff, der Überwachung, Protokollierung und Rückverfolgung umfasst. Anwendungen, die auf Microservices basieren, werden von Natur aus vertrieben. Im Gegensatz zu monolithischen Anwendungen, bei denen die Überwachung eines einzelnen Systems ausreicht, müssen Sie in einer verteilten Anwendungsarchitektur die Leistung der einzelnen Komponenten überwachen. Sie können Überwachungs-, Protokollierungs- und verteilte Ablaufverfolgungssysteme auf Clusterebene verwenden, um Probleme in Ihrem Cluster zu identifizieren, bevor sie Ihre Kunden stören.

Die in Kubernetes integrierten Tools zur Fehlerbehebung und Überwachung sind begrenzt. Der Metrics-Server sammelt Ressourcenmetriken und speichert sie im Speicher, speichert sie aber nicht dauerhaft. Sie können die Logs eines Pods mit kubectl einsehen, aber Kubernetes speichert Logs nicht automatisch. Und die Implementierung der verteilten Ablaufverfolgung erfolgt entweder auf der Ebene des Anwendungscodes oder mithilfe von Services-Meshes.

Die Erweiterbarkeit von Kubernetes kommt hier besonders gut zur Geltung. Mit Kubernetes können Sie Ihre bevorzugte zentralisierte Überwachungs-, Protokollierungs- und Tracing-Lösung einsetzen.

Empfehlungen

Überwachen Sie Ihre Anwendungen

Die Anzahl der Metriken, die Sie in modernen Anwendungen überwachen müssen, wächst kontinuierlich. Es ist hilfreich, wenn Sie Ihre Anwendungen automatisiert verfolgen können, sodass Sie sich darauf konzentrieren können, die Herausforderungen Ihrer Kunden zu lösen. Clusterweite Überwachungstools wie Prometheus oder CloudWatchContainer Insights können Ihren Cluster und Ihre Arbeitslast überwachen und Ihnen Signale geben, wann oder am besten, bevor etwas schief geht.

Mit Überwachungstools können Sie Warnmeldungen erstellen, die Ihr Betriebsteam abonnieren kann. Ziehen Sie Regeln zur Aktivierung von Alarmen für Ereignisse in Betracht, die, wenn sie sich verschärfen, zu einem Ausfall führen oder die Anwendungsleistung beeinträchtigen können.

Wenn Sie sich nicht sicher sind, welche Kennzahlen Sie überwachen sollten, können Sie sich von diesen Methoden inspirieren lassen:

- RED-Methode. Steht für Anfragen, Fehler und Dauer.
- USE-Methode. Steht für Auslastung, Sättigung und Fehler.

Der Beitrag Best Practices for Alerting on Kubernetes von Sysdig enthält eine umfassende Liste von Komponenten, die sich auf die Verfügbarkeit Ihrer Anwendungen auswirken können.

Verwenden Sie die Prometheus-Clientbibliothek, um Anwendungsmetriken bereitzustellen

Neben der Überwachung des Status der Anwendung und der Aggregation von Standardmetriken können Sie auch die Prometheus-Clientbibliothek verwenden, um anwendungsspezifische benutzerdefinierte Metriken bereitzustellen, um die Beobachtbarkeit der Anwendung zu verbessern.

Verwenden Sie zentralisierte Protokollierungstools, um Protokolle zu sammeln und zu speichern

Die Protokollierung in EKS lässt sich in zwei Kategorien einteilen: Protokolle der Steuerungsebene und Anwendungsprotokolle. Die Protokollierung auf der EKS-Kontrollebene stellt Prüf- und Diagnoseprotokolle direkt von der Steuerungsebene in die CloudWatch Protokolle in Ihrem Konto bereit. Anwendungsprotokolle sind Protokolle, die von Pods erstellt werden, die in Ihrem Cluster

ausgeführt werden. Zu den Anwendungsprotokollen gehören Protokolle, die von Pods erstellt wurden, auf denen die Geschäftslogikanwendungen und Kubernetes-Systemkomponenten wie CoreDNS, Cluster Autoscaler, Prometheus usw. ausgeführt werden.

EKS bietet fünf Arten von Protokollen auf der Kontrollebene:

1. Protokolle der Kubernetes-API-Serverkomponenten
2. Audit
3. Authentifikator
4. Leiter des Controllers
5. Scheduler

Die Controller-Manager- und Scheduler-Protokolle können bei der Diagnose von Problemen mit der Steuerungsebene wie Engpässen und Fehlern helfen. Standardmäßig werden die Protokolle der EKS-Steuerungsebene nicht an Logs gesendet. CloudWatch Sie können die Protokollierung der Kontrollebene aktivieren und die Arten von Protokollen der EKS-Kontrollebene auswählen, die Sie für jeden Cluster in Ihrem Konto erfassen möchten

Das Sammeln von Anwendungsprotokollen erfordert die Installation eines Protokollaggregator-Tools wie Fluent Bit, Fluentd oder CloudWatchContainer Insights in Ihrem Cluster.

Die Log-Aggregator-Tools von Kubernetes werden als DaemonSets Knoten ausgeführt und entfernen Container-Logs von Knoten. Anwendungsprotokolle werden dann zur Speicherung an ein zentrales Ziel gesendet. CloudWatch Container Insights kann beispielsweise entweder Fluent Bit oder Fluentd verwenden, um Protokolle zu sammeln und sie zur Speicherung an CloudWatch Logs zu senden. Fluent Bit und Fluentd unterstützen viele beliebte Protokollanalysesysteme wie Elasticsearch und InfluxDB, sodass Sie das Speicher-Backend für Ihre Logs ändern können, indem Sie Fluent Bit oder die Log-Konfiguration von Fluentd ändern.

Verwenden Sie ein verteiltes Tracing-System, um Engpässe zu identifizieren

Eine typische moderne Anwendung besteht aus Komponenten, die über das Netzwerk verteilt sind, und ihre Zuverlässigkeit hängt vom ordnungsgemäßen Funktionieren der einzelnen Komponenten ab, aus denen die Anwendung besteht. Sie können eine verteilte Ablaufverfolgungslösung verwenden, um zu verstehen, wie Anfragen fließen und wie Systeme kommunizieren. Mithilfe von Ablaufverfolgungen können Sie erkennen, wo in Ihrem Anwendungsnetzwerk Engpässe bestehen, und Probleme verhindern, die zu kaskadierenden Ausfällen führen können.

Sie haben zwei Möglichkeiten, die Ablaufverfolgung in Ihren Anwendungen zu implementieren: Sie können entweder die verteilte Ablaufverfolgung auf Codeebene mithilfe gemeinsam genutzter Bibliotheken implementieren oder ein Service Mesh verwenden.

Die Implementierung von Tracing auf Codeebene kann nachteilig sein. Bei dieser Methode müssen Sie Änderungen an Ihrem Code vornehmen. Dies ist noch komplizierter, wenn Sie mehrsprachige Anwendungen haben. Sie sind auch für die Verwaltung einer weiteren Bibliothek in all Ihren Diensten verantwortlich.

Service Meshes wie LinkerD und Istio können verwendet werden, um verteiltes Tracing in Ihrer Anwendung mit minimalen Änderungen am Anwendungscode zu implementieren. Sie können Service Mesh verwenden, um die Generierung, Protokollierung und Ablaufverfolgung von Metriken zu standardisieren.

Tracing-Tools wie AWS X-Ray und Jaeger unterstützen sowohl Shared Library- als auch Service Mesh-Implementierungen.

Erwägen Sie, ein Tracing-Tool wie AWS X-Ray oder Jaeger zu verwenden, das beide Implementierungen (Shared Library und Service Mesh) unterstützt, damit Sie nicht zwischen Tools wechseln müssen, wenn Sie später Service Mesh einführen.

EKS-Steuerebene

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Amazon Elastic Kubernetes Service (EKS) ist ein verwalteter Kubernetes-Service, mit dem Sie Kubernetes auf AWS ganz einfach ausführen können, ohne Ihre eigene Kubernetes-Steuerebene oder Ihre eigenen Worker-Knoten installieren, betreiben und warten zu müssen. Er läuft auf Upstream-Kubernetes und ist als Kubernetes-konform zertifiziert. Diese Konformität stellt sicher, dass EKS die Kubernetes-APIs unterstützt, genau wie die Open-Source-Community-Version, die Sie auf EC2 oder vor Ort installieren können. Bestehende Anwendungen, die auf Upstream-Kubernetes ausgeführt werden, sind mit Amazon EKS kompatibel.

EKS verwaltet automatisch die Verfügbarkeit und Skalierbarkeit der Kubernetes-Steuerebenenknoten und ersetzt automatisch fehlerhafte Knoten der Kontrollebene.

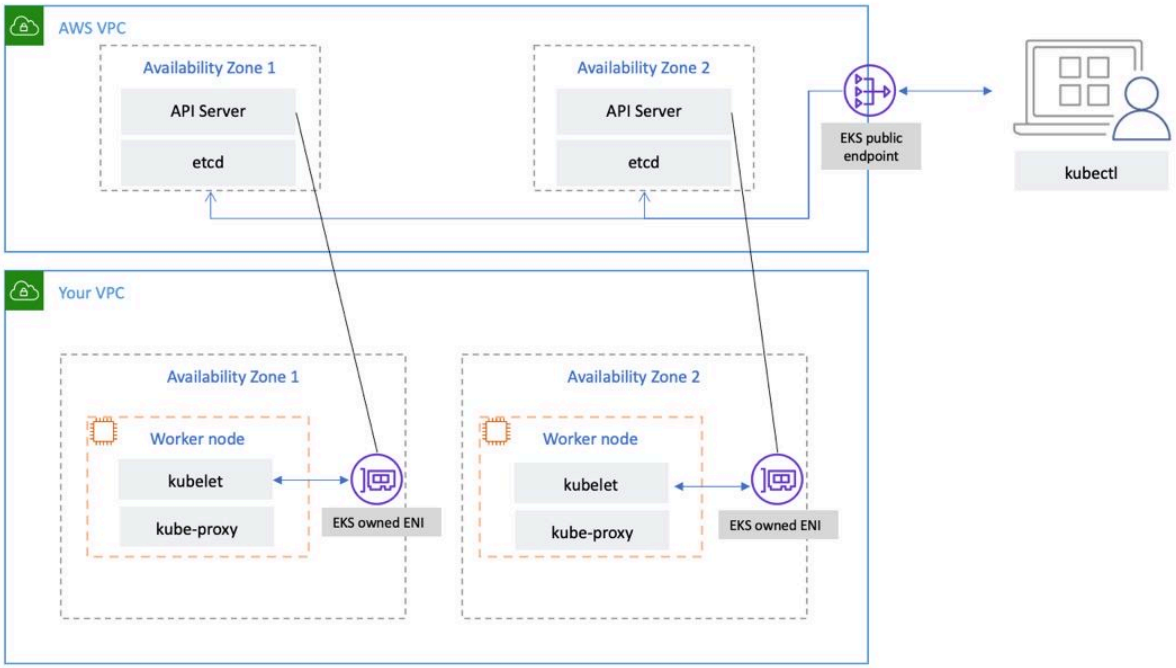
EKS-Architektur

Die EKS-Architektur wurde entwickelt, um einzelne Fehlerquellen zu eliminieren, die die Verfügbarkeit und Beständigkeit der Kubernetes-Steuerebene beeinträchtigen könnten.

Die von EKS verwaltete Kubernetes-Steuerebene wird in einer von EKS verwalteten VPC ausgeführt. Die EKS-Steuerebene umfasst die Kubernetes-API-Serverknoten usw. Kubernetes-API-Serverknoten, die Komponenten wie den API-Server und den Scheduler `kube-controller-manager` ausführen und in einer Gruppe mit auto-scaling ausgeführt werden. EKS betreibt mindestens zwei API-Serverknoten in unterschiedlichen Availability Zones (AZs) innerhalb der AWS-Region. Aus Gründen der Haltbarkeit laufen die etcd-Serverknoten ebenfalls in einer Auto-Scaling-Gruppe, die sich über drei erstreckt. AZs EKS betreibt in jeder AZ ein NAT-Gateway, und API-Server und etcd-Server laufen in einem privaten Subnetz. Diese Architektur stellt sicher, dass ein Ereignis in einer einzelnen AZ die Verfügbarkeit des EKS-Clusters nicht beeinträchtigt.

Wenn Sie einen neuen Cluster erstellen, erstellt Amazon EKS einen hochverfügbaren Endpunkt für den verwalteten Kubernetes-API-Server, den Sie für die Kommunikation mit Ihrem Cluster verwenden (mithilfe von Tools wie). `kubectl` Der verwaltete Endpunkt verwendet NLB für den Lastenausgleich der Kubernetes-API-Server. EKS stellt außerdem zwei unterschiedliche ENIs bereit, AZs um die Kommunikation mit Ihren Worker-Knoten zu erleichtern.

EKS-Netzwerkkonnektivität auf der Datenebene



Sie können konfigurieren, ob der API-Server Ihres Kubernetes-Clusters vom öffentlichen Internet (über den öffentlichen Endpunkt) oder über Ihre VPC (mit dem ENIs EKS-verwalteten) oder von beiden aus erreichbar ist.

Unabhängig davon, ob Benutzer und Worker-Knoten über den öffentlichen Endpunkt oder das von EKS verwaltete ENI eine Verbindung zum API-Server herstellen, gibt es redundante Verbindungspfade.

Empfehlungen

Lesen Sie sich die folgenden Empfehlungen durch.

Überwachen Sie die Kennzahlen der Kontrollebene

Durch die Überwachung der Kubernetes-API-Metriken erhalten Sie Einblicke in die Leistung der Kontrollebene und können Probleme identifizieren. Eine fehlerhafte Kontrollebene kann die Verfügbarkeit der Workloads beeinträchtigen, die innerhalb des Clusters ausgeführt werden. Beispielsweise können schlecht geschriebene Controller die API-Server überlasten und die Verfügbarkeit Ihrer Anwendung beeinträchtigen.

Kubernetes stellt Metriken der Kontrollebene am Endpunkt zur Verfügung. `/metrics`

Sie können die bereitgestellten Metriken wie folgt anzeigen: `kubectl`

```
kubectl get --raw /metrics
```

Diese Metriken werden in einem Prometheus-Textformat dargestellt.

Sie können Prometheus verwenden, um diese Metriken zu sammeln und zu speichern. Im Mai 2020 CloudWatch wurde Unterstützung für die Überwachung von Prometheus-Metriken in CloudWatch Container Insights hinzugefügt. Sie können Amazon also auch CloudWatch zur Überwachung der EKS-Steuerebene verwenden. Sie können Tutorial for Adding a New Prometheus Scrape Target: Prometheus KPI Server Metrics verwenden, um Metriken zu sammeln und ein CloudWatch Dashboard zur Überwachung der Kontrollebene Ihres Clusters zu erstellen.

Kennzahlen zum Kubernetes-API-Server finden Sie hier.
`apiserver_request_duration_seconds`Kann beispielsweise angeben, wie lange die Ausführung von API-Anfragen dauert.

Erwägen Sie die Überwachung dieser Metriken auf der Kontrollebene:

API-Server

Metrik	Description
`apiserver_request_total`	Zähler der Apiserver-Anfragen, aufgeschlüsselt nach Verb, Probelaufwert, Gruppe, Version, Ressource, Bereich, Komponente und HTTP-Antwortcode.
`apiserver_request_duration_seconds*`	Histogramm der Antwortlatenz in Sekunden für jedes Verb, jeden Probelaufwert, jede Gruppe, Version, Ressource, Unterressource, jeden Bereich und jede Komponente.
`apiserver_admission_controller_admission_duration_seconds*`	Latenzhistogramm für den Zugangscontroller in Sekunden, identifiziert anhand des Namens und aufgeschlüsselt nach Vorgang, API-Ressource und Typ (validieren oder zulassen).

Metrik	Description
apiserver_admission_webhook_rejection_count	Anzahl der Webhook-Ablehnungen für die Zulassung. Identifiziert nach Name, Vorgang, Ablehnungscode, Typ (validieren oder zulassen), error_type (calling_webhook_error, apiserver_internal_error, no_error)
rest_client_request_duration_seconds*	Latenz-Histogramm in Sekunden anfordern. Aufgeschlüsselt nach Verb und URL.
rest_client_requests_total	Anzahl der HTTP-Anfragen, partitioniert nach Statuscode, Methode und Host.

- Zu den Histogramm-Metriken gehören die Suffixe _bucket, _sum und _count.

usw

Metrik	Descri on
etcd_request_duration_seconds*	Etcd-Anfo rderun atenzh ogram in Sekun für jeden Vorga und Objekt .

Metrik	Descri on
`apiserver_storage_db_total_size_in_bytes` oder `apiserver_storage_size_bytes` (ab EKS v1.28)	Größe der Etcd-Date nbank

- Zu den Histogramm-Metriken gehören die Suffixe _bucket, _sum und _count.

Erwägen Sie, das Kubernetes Monitoring Overview Dashboard zu verwenden, um Kubernetes-API-Serveranfragen sowie Latenz- und Latenzmetriken usw. zu visualisieren und zu überwachen.

> **Important**
>
> Wenn die Größenbeschränkung für die Datenbank überschritten wird, gibt etcd einen Alarm ohne Speicherplatz aus und nimmt keine weiteren Schreibanfragen mehr entgegen. Mit anderen Worten, der Cluster wird schreibgeschützt, und alle Anfragen zur Änderung von Objekten wie das Erstellen neuer Pods, das Skalieren von Bereitstellungen usw. werden vom API-Server des Clusters abgelehnt.

Cluster-Authentifizierung

EKS unterstützt derzeit zwei Arten der Authentifizierung: Inhaber-/Dienstkonto-Tokens und IAM-Authentifizierung, bei der die Webhook-Token-Authentifizierung verwendet wird. Wenn Benutzer die Kubernetes-API aufrufen, übergibt ein Webhook ein in der Anfrage enthaltenes Authentifizierungstoken an IAM. Das Token, eine signierte Base-64-URL, wird von der AWS-Befehlszeilenschnittstelle (AWS CLI) generiert.

Der IAM-Benutzer oder die IAM-Rolle, die den EKS-Cluster erstellt, erhält automatisch vollen Zugriff auf den Cluster. Sie können den Zugriff auf den EKS-Cluster verwalten, indem Sie die aws-auth-Configmap bearbeiten.

Wenn Sie die `aws-auth` Configmap falsch konfigurieren und den Zugriff auf den Cluster verlieren, können Sie trotzdem den Benutzer oder die Rolle des Clustererstellers verwenden, um auf Ihren EKS-Cluster zuzugreifen.

In dem unwahrscheinlichen Fall, dass Sie den IAM-Service in der AWS-Region nicht nutzen können, können Sie den Cluster auch mit dem Bearer-Token des Kubernetes-Servicekontos verwalten.

Erstellen Sie ein `super-admin` Konto, das alle Aktionen im Cluster ausführen darf:

```
kubectl -n kube-system create serviceaccount super-admin
```

Erstellen Sie eine Rollenbindung, die der Super-Admin-Cluster-Admin-Rolle zuweist:

```
kubectl create clusterrolebinding super-admin-rb --clusterrole=cluster-admin --
serviceaccount=kube-system:super-admin
```

Holen Sie sich das Geheimnis des Dienstkontos:

```
SECRET_NAME=`kubectl -n kube-system get serviceaccount/super-admin -o
 jsonpath='{.secrets[0].name}'`
```

Ruft das mit dem Secret verknüpfte Token ab:

```
TOKEN=`kubectl -n kube-system get secret $SECRET_NAME -o jsonpath='{.data.token}'|
 base64 --decode`
```

Dienstkonto und Token hinzufügen zu`kubeconfig`:

```
kubectl config set-credentials super-admin --token=$TOKEN
```

Stellen Sie den aktuellen Kontext ein, um das `kubeconfig` Super-Admin-Konto zu verwenden:

```
kubectl config set-context --current --user=super-admin
```

Final `kubeconfig` sollte so aussehen:

```
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>>
```

Zulassung: Webhooks

Kubernetes hat zwei Arten von Zulassungs-Webhooks: Validierungs-Webhooks und mutierende Zulassungs-Webhooks. Diese ermöglichen es einem Benutzer, die Kubernetes-API zu erweitern und Objekte zu validieren oder zu mutieren, bevor sie von der API akzeptiert werden. Schlechte Konfigurationen dieser Webhooks können die EKS-Steuerebene destabilisieren, indem sie kritische Clustervorgänge blockieren.

Um eine Beeinträchtigung kritischer Clusteroperationen zu vermeiden, sollten Sie entweder das Setzen von „Catch-All“ -Webhooks wie den folgenden vermeiden:

```
- name: "pod-policy.example.com"
```

```
  rules:
  - apiGroups:   ["*"]
    apiVersions: ["*"]
    operations:  ["*"]
    resources:   ["*"]
    scope: "*"
```

Oder stellen Sie sicher, dass der Webhook über eine Fail-Open-Richtlinie mit einem Timeout von weniger als 30 Sekunden verfügt, um sicherzustellen, dass kritische Cluster-Workloads nicht beeinträchtigt werden, wenn Ihr Webhook nicht verfügbar ist.

Blockieren Sie Pods mit unsicheren **`sysctls`**

`Sysctl`ist ein Linux-Hilfsprogramm, mit dem Benutzer Kernelparameter während der Laufzeit ändern können. Diese Kernelparameter steuern verschiedene Aspekte des Verhaltens des Betriebssystems, z. B. das Netzwerk, das Dateisystem, den virtuellen Speicher und die Prozessverwaltung.

Kubernetes ermöglicht das Zuweisen von `sysctl` Profilen für Pods. Kubernetes wird als sicher und unsicher eingestuft`systcls`. Sicher `sysctls` haben im Container oder Pod Namespaces, und ihre Einstellung hat keine Auswirkungen auf andere Pods auf dem Knoten oder den Knoten selbst. Im Gegensatz dazu sind unsichere Sysctls standardmäßig deaktiviert, da sie möglicherweise andere Pods stören oder den Knoten instabil machen können.

Da unsicher standardmäßig deaktiviert `sysctls` sind, erstellt das Kubelet keinen Pod mit einem unsicheren Profil. `sysctl` Wenn Sie einen solchen Pod erstellen, weist der Scheduler solche Pods wiederholt Knoten zu, während der Knoten ihn nicht starten kann. Diese Endlosschleife belastet letztendlich die Cluster-Steuerebene und macht den Cluster instabil.

Erwägen Sie, OPA Gatekeeper oder Kyverno zu verwenden, um Pods mit unsicheren Daten zurückzuweisen. `sysctls`

Umgang mit Cluster-Upgrades

Seit April 2021 wurde der Kubernetes-Veröffentlichungszyklus von vier Releases pro Jahr (einmal pro Quartal) auf drei Releases pro Jahr geändert. Eine neue Nebenversion (wie 1. 21 oder 1. 22) wird ungefähr alle fünfzehn Wochen veröffentlicht. Ab Kubernetes 1.19 wird jede Nebenversion nach ihrer ersten Veröffentlichung etwa zwölf Monate lang unterstützt. Mit dem Aufkommen von Kubernetes v1.28 hat sich der Kompatibilitätsunterschied zwischen der Steuerungsebene und den Worker-Knoten von n-2 auf n-3 Nebenversionen ausgeweitet. Weitere Informationen finden Sie unter Bewährte Methoden für Cluster-Upgrades.

Konnektivität von Cluster-Endpunkten

Bei der Arbeit mit Amazon EKS (Elastic Kubernetes Service) können Verbindungstimeouts oder Fehler bei Ereignissen wie der Skalierung oder dem Patchen der Kubernetes-Steuerebene auftreten. Diese Ereignisse können dazu führen, dass die Kube-Apiserver-Instanzen ersetzt werden, was möglicherweise dazu führt, dass bei der Auflösung des FQDN unterschiedliche IP-Adressen zurückgegeben werden. In diesem Dokument werden bewährte Methoden für Kubernetes-API-Nutzer zur Aufrechterhaltung einer zuverlässigen Konnektivität beschrieben.

> **Note**
>
> Die Implementierung dieser bewährten Methoden erfordert möglicherweise Aktualisierungen der Client-Konfigurationen oder der Skripte, um neue Strategien zur erneuten DNS-Auflösung und zum erneuten Versuch effektiv handhaben zu können.

Das Hauptproblem ist das clientseitige DNS-Caching und das Potenzial für veraltete IP-Adressen von EKS-Endpunkten — öffentliche NLB für öffentliche Endgeräte oder X-ENI für private Endgeräte. Wenn die Kube-Apiserver-Instanzen ersetzt werden, wird der vollqualifizierte Domainname (FQDN) möglicherweise zu neuen IP-Adressen aufgelöst. Aufgrund der DNS Time to Live (TTL) - Einstellungen, die in der von AWS verwalteten Route 53 53-Zone auf 60 Sekunden festgelegt sind, können Clients jedoch für einen kurzen Zeitraum weiterhin veraltete IP-Adressen verwenden.

Um diese Probleme zu beheben, sollten Kubernetes-API-Nutzer (wie Kubectl, CI/CD Pipelines und benutzerdefinierte Anwendungen) die folgenden bewährten Methoden implementieren:

- Implementieren Sie die DNS-Neuauflösung
- Implementieren Sie Wiederholungen mit Backoff und Jitter. Lesen Sie zum Beispiel diesen Artikel mit dem Titel Fehler passieren
- Implementieren Sie Client-Timeouts. Legen Sie geeignete Timeouts fest, um zu verhindern, dass Anfragen mit langer Laufzeit Ihre Anwendung blockieren. Beachten Sie, dass einige Kubernetes-Clientbibliotheken, insbesondere solche, die von OpenAPI-Generatoren generiert werden, das Einstellen benutzerdefinierter Timeouts möglicherweise nicht einfach zulassen.
  - Beispiel 1 mit kubectl:

```
kubectl get pods --request-timeout 10s # default: no timeout
```

  - Beispiel 2 mit Python: Der Kubernetes-Client stellt einen _request_timeout-Parameter bereit

Durch die Implementierung dieser Best Practices können Sie die Zuverlässigkeit und Belastbarkeit Ihrer Anwendungen bei der Interaktion mit der Kubernetes-API erheblich verbessern. Denken Sie daran, diese Implementierungen gründlich zu testen, insbesondere unter simulierten Ausfallbedingungen, um sicherzustellen, dass sie sich bei tatsächlichen Skalierungs- oder Patching-Ereignissen erwartungsgemäß verhalten.

Große Cluster ausführen

EKS überwacht aktiv die Auslastung der Instanzen auf der Kontrollebene und skaliert sie automatisch, um eine hohe Leistung zu gewährleisten. Sie sollten jedoch potenzielle Leistungsprobleme und -beschränkungen innerhalb von Kubernetes und Kontingente in AWS-Services berücksichtigen, wenn Sie große Cluster ausführen.

- Bei Clustern mit mehr als 1000 Services kann es laut den vom Team durchgeführten Tests zu Netzwerklatenz kommen, wenn sie `kube-proxy` im `iptables` Modus verwendet werden. ProjectCalico Die Lösung besteht darin, in den `ipvs`Modus „`kube-proxy`Im Betrieb“ zu wechseln.
- Möglicherweise kommt es auch zu einer Drosselung von EC2-API-Anfragen, wenn das CNI IP-Adressen für Pods anfordern muss oder wenn Sie häufig neue EC2-Instances erstellen müssen. Sie können die Anzahl der EC2-API-Aufrufe reduzieren, indem Sie das CNI so konfigurieren, dass IP-Adressen zwischengespeichert werden. Sie können größere EC2-Instance-Typen verwenden, um EC2-Skalierungsereignisse zu reduzieren.

Zusätzliche Ressourcen:

- Entmystifizierung von Cluster-Netzwerken für Amazon EKS-Worker-Knoten
- Zugriffskontrolle für Amazon EKS-Cluster-Endpunkte
- AWS re:Invent 2019: Amazon EKS unter der Haube (-R1) CON421

EKS-Datenebene

Um hochverfügbare und ausfallsichere Anwendungen zu betreiben, benötigen Sie eine hochverfügbare und belastbare Datenebene. Eine elastische Datenebene stellt sicher, dass Kubernetes Ihre Anwendungen automatisch skalieren und reparieren kann. Eine robuste Datenebene besteht aus zwei oder mehr Worker-Knoten, kann mit der Arbeitslast wachsen und schrumpfen und wird nach Ausfällen automatisch wiederhergestellt.

Bei EKS stehen Ihnen mehrere Optionen für Worker-Knoten zur Verfügung: verwaltete EKS Auto Mode Nodes, EC2-Instances und Fargate.

Der automatische Modus von EKS bietet den einfachsten Weg zu einer ausfallsicheren Datenebene. Der automatische Modus erweitert das AWS-Management von Kubernetes-Clustern über den Cluster selbst hinaus, sodass AWS auch die Infrastruktur einrichten und verwalten kann, die den reibungslosen Betrieb Ihrer Workloads ermöglicht. Der automatische Modus skaliert die Datenebene automatisch nach oben oder unten, während Kubernetes die Pods skaliert, und stellt kontinuierlich sicher, dass die Nodes in Ihrem Cluster für die aktuell ausgeführten Workloads angemessen und kostengünstig dimensioniert sind.

Wenn Sie sich für EC2-Instances entscheiden, können Sie die Worker-Knoten selbst verwalten oder von EKS verwaltete Knotengruppen verwenden. Sie können einen Cluster mit einer Mischung aus Auto Mode, verwalteten, selbstverwalteten Worker-Knoten und Fargate haben.

Fargate führt jeden Pod in einer isolierten Rechenumgebung aus. Jeder Pod, der auf Fargate läuft, erhält seinen eigenen Worker-Knoten. Fargate skaliert die Datenebene automatisch, während Kubernetes Pods skaliert. Sie können sowohl die Datenebene als auch Ihre Arbeitslast skalieren, indem Sie den horizontalen Pod-Autoscaler verwenden.

Die bevorzugte Methode zur Skalierung von EC2-Worker-Knoten (sofern nicht der EKS-Auto-Modus verwendet wird, in dem dies automatisch von AWS ausgeführt wird), ist die Verwendung von Karpenter -, Kubernetes Cluster Autoscaler- oder EC2 Auto Scaling Scaling-Gruppen.

Empfehlungen

Verteilen Sie Worker-Knoten und Workloads auf mehrere AZs

Sie können Ihre Workloads vor Ausfällen in einer einzelnen AZ schützen, indem Sie Worker-Knoten und Pods in mehreren ausführen. AZs Sie können die AZ, in der die Worker-Knoten erstellt werden, mithilfe der Subnetze steuern, in denen Sie die Knoten erstellen.

Die empfohlene Methode zur Verteilung von Pods auf mehrere Pods AZs ist die Verwendung von Topology Spread Constraints für Pods. Funktionen zur automatischen Skalierung wie EKS Auto Mode und Karpenter sind sich der Einschränkungen der Topologieverteilung bewusst und starten Knoten automatisch in der richtigen Reihenfolge, AZs damit Ihre Einschränkungen eingehalten werden können.

Bei der folgenden Bereitstellung werden Pods nach Möglichkeit verteilt, sodass diese Pods trotzdem ausgeführt werden können, AZs wenn nicht:

```
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-scheduler`kennt Topologiedomänen nur über Knoten, die mit diesen Bezeichnungen existieren. Wenn die oben genannte Bereitstellung in einem Cluster mit Knoten nur in einer einzigen Zone bereitgestellt wird, planen alle Pods auf diesen Knoten, da sie die anderen Zonen `kube-scheduler` nicht kennt. Damit diese Topologieverteilung mit dem Scheduler erwartungsgemäß funktioniert, müssen in allen Zonen bereits Knoten vorhanden sein. Die `minDomains` Eigenschaft einer Topologie-Spread-Beschränkung wird verwendet, um den Scheduler über die Anzahl der geeigneten Domänen zu informieren, auch wenn dort ein Knoten läuft, um dieses Problem zu vermeiden.

> **Warning**
>
> Die Einstellung `whenUnsatisfiable` auf `DoNotSchedule` führt dazu, dass Pods nicht geplant werden können, wenn die Topologie-Spread-Beschränkung nicht erfüllt werden kann. Sie sollte nur gesetzt werden, wenn es besser ist, dass Pods nicht ausgeführt werden, anstatt gegen die Topologie-Spread-Beschränkung zu verstoßen.

In älteren Versionen von Kubernetes können Sie Pod-Anti-Affinitätsregeln verwenden, um Pods für mehrere Pods zu planen. AZs Das folgende Manifest informiert den Kubernetes-Scheduler darüber, Pods lieber einzeln zu planen. 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**
>
> Erfordern Sie nicht, dass Pods unterschiedlich geplant werden, AZs andernfalls wird die Anzahl der Pods in einer Bereitstellung niemals die Anzahl von überschreiten. AZs

Stellen Sie sicher, dass Knoten in jeder AZ gestartet werden können, wenn Sie EBS-Volumes verwenden

Wenn Sie Amazon EBS zur Bereitstellung persistenter Volumes verwenden, müssen Sie sicherstellen, dass sich die Pods und das zugehörige EBS-Volume in derselben AZ befinden. Ein Pod kann nicht auf EBS-gestützte persistente Volumes zugreifen, die sich in einer anderen AZ befinden. Der Kubernetes-Scheduler weiß anhand der Labels auf dem Knoten, in welcher AZ sich ein Worker-Knoten befindet, und plant immer einen Pod, für den ein EBS-Volume in derselben AZ wie das Volume erforderlich ist. Wenn jedoch in der AZ, in der sich das Volume befindet, keine Worker-Knoten verfügbar sind, kann der Pod nicht geplant werden.

Wenn Sie EKS Auto Mode oder Karpenter verwenden, müssen Sie sicherstellen, dass Sie Subnetze in jeder AZ NodeClass auswählen. Wenn Sie verwaltete Knotengruppen verwenden, müssen Sie sicherstellen, dass Sie in jeder AZ über eine Knotengruppe verfügen.

In den automatischen Modus von EKS ist eine EBS-Speicherfunktion integriert. Wenn Sie jedoch Karpenter oder Managed Node Groups verwenden, muss die EBS CSI ebenfalls installiert werden.

Verwenden Sie den EKS-Automatikmodus, um Worker-Knoten zu verwalten

Der automatische Modus von EKS optimiert das EKS-Management, indem er produktionsbereite Cluster mit minimalem Betriebsaufwand bereitstellt. Der automatische Modus ist dafür verantwortlich, die Anzahl der Knoten je nach den Pods, die im Cluster ausgeführt werden, nach oben oder unten zu skalieren. Die Knoten werden automatisch mit Software-Patches und Fixes auf dem neuesten Stand gehalten, wobei die Updates gemäß den konfigurierten NodePoolUnterbrechungseinstellungen und Pod-Disruption-Budgets durchgeführt werden.

Führen Sie den Node Monitoring Agent aus

Der Node Monitoring Agent überwacht Node-Integritätsprobleme und reagiert darauf, indem er Kubernetes-Ereignisse veröffentlicht und den Statuszustand auf Nodes aktualisiert. Der Node

Monitoring Agent ist in EKS Auto Mode Nodes enthalten und kann als EKS-Addon für Knoten installiert werden, die nicht im Auto Mode verwaltet werden.

EKS Auto Mode, Managed Node Groups und Karpenter sind alle in der Lage, vom Node Monitoring Agent gemeldete schwerwiegende Knotenprobleme zu erkennen und diese Knoten automatisch zu reparieren, wenn diese Bedingungen eintreten.

Implementieren Sie QoS

Für kritische Anwendungen sollten Sie erwägen, `requests` = `limits` für den Container im Pod zu definieren. Dadurch wird sichergestellt, dass der Container nicht beendet wird, wenn ein anderer Pod Ressourcen anfordert.

Es hat sich bewährt, CPU- und Speicherbegrenzungen für alle Container zu implementieren, um zu verhindern, dass ein Container versehentlich Systemressourcen verbraucht und dadurch die Verfügbarkeit anderer Prozesse am selben Standort beeinträchtigt.

Konfigurieren und dimensionieren Sie die Ressourcen für alle Workloads Requests/Limits

Bei der Dimensionierung von Ressourcenanforderungen und der Grenzwerte für Workloads können einige allgemeine Richtlinien angewendet werden:

- Geben Sie keine Ressourcenlimits für die CPU an. In Ermangelung von Grenzwerten bestimmt die Anfrage, wie viel relative CPU-Zeit Container erhalten. Auf diese Weise können Ihre Workloads die gesamte CPU nutzen, ohne dass es zu einer künstlichen Begrenzung oder zu einem Verlust kommt.
- Bei Ressourcen, die nicht zur CPU gehören, `limits` bietet die Konfiguration von `requests` = das vorhersehbarste Verhalten. Wenn! `requests` `=limits`, außerdem wurde die QOS des Containers von Guaranteed auf Burstable herabgesetzt, wodurch die Wahrscheinlichkeit, dass er entfernt wird, wenn der Knoten überlastet wird, größer ist.
- Geben Sie für Ressourcen, die nicht zur CPU gehören, kein Limit an, das viel größer ist als die Anforderung. Je größer `limits` die Konfiguration im Verhältnis zu ist`requests`, desto wahrscheinlicher ist es, dass Knoten überlastet werden, was zu einer hohen Wahrscheinlichkeit einer Unterbrechung der Arbeitslast führt.
- Anfragen mit der richtigen Größe sind besonders wichtig, wenn Sie eine Node-Auto-Scaling-Lösung wie Karpenter oder Cluster verwenden. AutoScaler Diese Tools untersuchen Ihre Workload-Anfragen, um die Anzahl und Größe der bereitzustellenden Knoten zu ermitteln. Wenn

Ihre Anfragen zu klein sind und höhere Grenzwerte aufweisen, kann es sein, dass Ihre Workloads entfernt oder OOM beendet werden, wenn sie auf einem Knoten eng zusammengepackt wurden.

Das Ermitteln von Ressourcenanforderungen kann schwierig sein, aber Tools wie der Vertical Pod Autoscaler können Ihnen helfen, die richtige Größe der Anfragen zu finden, indem Sie die Nutzung der Container-Ressourcen zur Laufzeit beobachten. Zu den weiteren Tools, die für die Bestimmung der Anforderungsgrößen nützlich sein können, gehören:

- Goldlöckchen
- Parca
- Profiler
- rsg

Konfigurieren Sie Ressourcenkontingente für Namespaces

Namespaces sind für die Verwendung in Umgebungen mit vielen Benutzern vorgesehen, die auf mehrere Teams oder Projekte aufgeteilt sind. Sie bieten einen Gültigkeitsbereich für Namen und bieten eine Möglichkeit, Clusterressourcen auf mehrere Teams, Projekte und Workloads aufzuteilen. Sie können den aggregierten Ressourcenverbrauch in einem Namespace einschränken. Das `ResourceQuota`Objekt kann die Anzahl der Objekte, die in einem Namespace erstellt werden können, nach Typ sowie die Gesamtmenge der Rechenressourcen einschränken, die von Ressourcen in diesem Projekt verbraucht werden können. Sie können die Gesamtsumme der and/or Speicher-Rechenressourcen (CPU und Arbeitsspeicher) begrenzen, die in einem bestimmten Namespace angefordert werden können.

Wenn das Ressourcenkontingent für einen Namespace für Rechenressourcen wie CPU und Arbeitsspeicher aktiviert ist, müssen Benutzer Anforderungen oder Grenzwerte für jeden Container in diesem Namespace angeben.

Erwägen Sie, Kontingente für jeden Namespace zu konfigurieren. Erwägen Sie`LimitRanges`, diese Option zu verwenden, um automatisch vorkonfigurierte Grenzwerte auf Container innerhalb eines Namespaces anzuwenden.

Beschränken Sie die Nutzung von Container-Ressourcen innerhalb eines Namespace

Ressourcenkontingente helfen dabei, die Menge an Ressourcen zu begrenzen, die ein Namespace verwenden kann. Das `LimitRange`Objekt kann Ihnen dabei helfen, die minimalen und maximalen

Ressourcen zu implementieren, die ein Container anfordern kann. Mithilfe von `LimitRange` Ihnen können Sie eine Standardanforderung und Grenzwerte für Container festlegen. Dies ist hilfreich, wenn die Festlegung von Limits für Rechenressourcen in Ihrer Organisation nicht üblich ist. Wie der Name schon sagt, `LimitRange` kann die minimale und maximale Nutzung von Rechenressourcen pro Pod oder Container in einem Namespace durchgesetzt werden. Außerdem können Sie die minimale und maximale Speicheranforderung pro PersistentVolumeClaim Namespace durchsetzen.

Erwägen Sie die Verwendung `LimitRange` in Verbindung mit`ResourceQuota`, um Grenzwerte sowohl auf Container- als auch auf Namespace-Ebene durchzusetzen. Durch die Festlegung dieser Grenzwerte wird sichergestellt, dass sich ein Container oder ein Namespace nicht auf Ressourcen auswirkt, die von anderen Mandanten im Cluster verwendet werden.

Verwenden NodeLocal DNSCache

Sie können die Cluster-DNS-Leistung verbessern, indem Sie Folgendes NodeLocalDNSCacheausführen: Diese Funktion führt einen DNS-Caching-Agenten auf Clusterknoten als DaemonSet aus. Alle Pods verwenden den DNS-Caching-Agenten, der auf dem Knoten ausgeführt wird, für die Namensauflösung, anstatt Service zu verwenden`kube-dns`. Diese Funktion ist automatisch im automatischen Modus von EKS enthalten.

Konfigurieren Sie die auto-scaling von CoreDNS

Eine weitere Methode zur Verbesserung der Cluster-DNS-Leistung besteht darin, die integrierte auto-scaling von CoreDNS-Pods zu aktivieren.

Diese Funktion überwacht kontinuierlich den Clusterstatus, einschließlich der Anzahl der Knoten und CPU-Kerne. Anhand dieser Informationen passt der Controller die Anzahl der Replikate der CoreDNS-Bereitstellung in einem EKS-Cluster dynamisch an.

Bewährte Methoden für Netzwerke

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Es ist wichtig, Kubernetes-Netzwerke zu verstehen, um Ihren Cluster und Ihre Anwendungen effizient zu betreiben. Pod-Networking, auch Cluster-Networking genannt, ist das Zentrum von Kubernetes-Netzwerken. Kubernetes unterstützt Container Network Interface (CNI) -Plugins für Cluster-Netzwerke.

Amazon EKS unterstützt offiziell das Amazon Virtual Private Cloud (VPC) CNI-Plugin zur Implementierung von Kubernetes-Pod-Netzwerken. Das VPC CNI bietet eine native Integration mit AWS VPC und arbeitet im Underlay-Modus. Im Underlay-Modus befinden sich Pods und Hosts auf derselben Netzwerkschicht und teilen sich den Netzwerk-Namespace. Die IP-Adresse des Pods ist aus Cluster- und VPC-Perspektive konsistent.

In diesem Handbuch wird das Amazon VPC Container Network Interface (VPC CNI) im Kontext von Kubernetes-Cluster-Netzwerken vorgestellt. Das VPC CNI ist das von EKS unterstützte Standard-Netzwerk-Plugin und steht daher im Mittelpunkt des Handbuchs. Das VPC CNI ist hochgradig konfigurierbar, um verschiedene Anwendungsfälle zu unterstützen. Dieses Handbuch enthält außerdem spezielle Abschnitte zu verschiedenen VPC-CNI-Anwendungsfällen, Betriebsmodi und Unterkomponenten, gefolgt von den Empfehlungen.

Amazon EKS führt Upstream-Kubernetes aus und ist als Kubernetes-konform zertifiziert. Sie können zwar alternative CNI-Plugins verwenden, diese Anleitung enthält jedoch keine Empfehlungen für die Verwaltung alternativer Plug-ins. CNIs In der Dokumentation zu EKS Alternate CNI finden Sie eine Liste der Partner und Ressourcen für die effektive Verwaltung von Alternate CNIs .

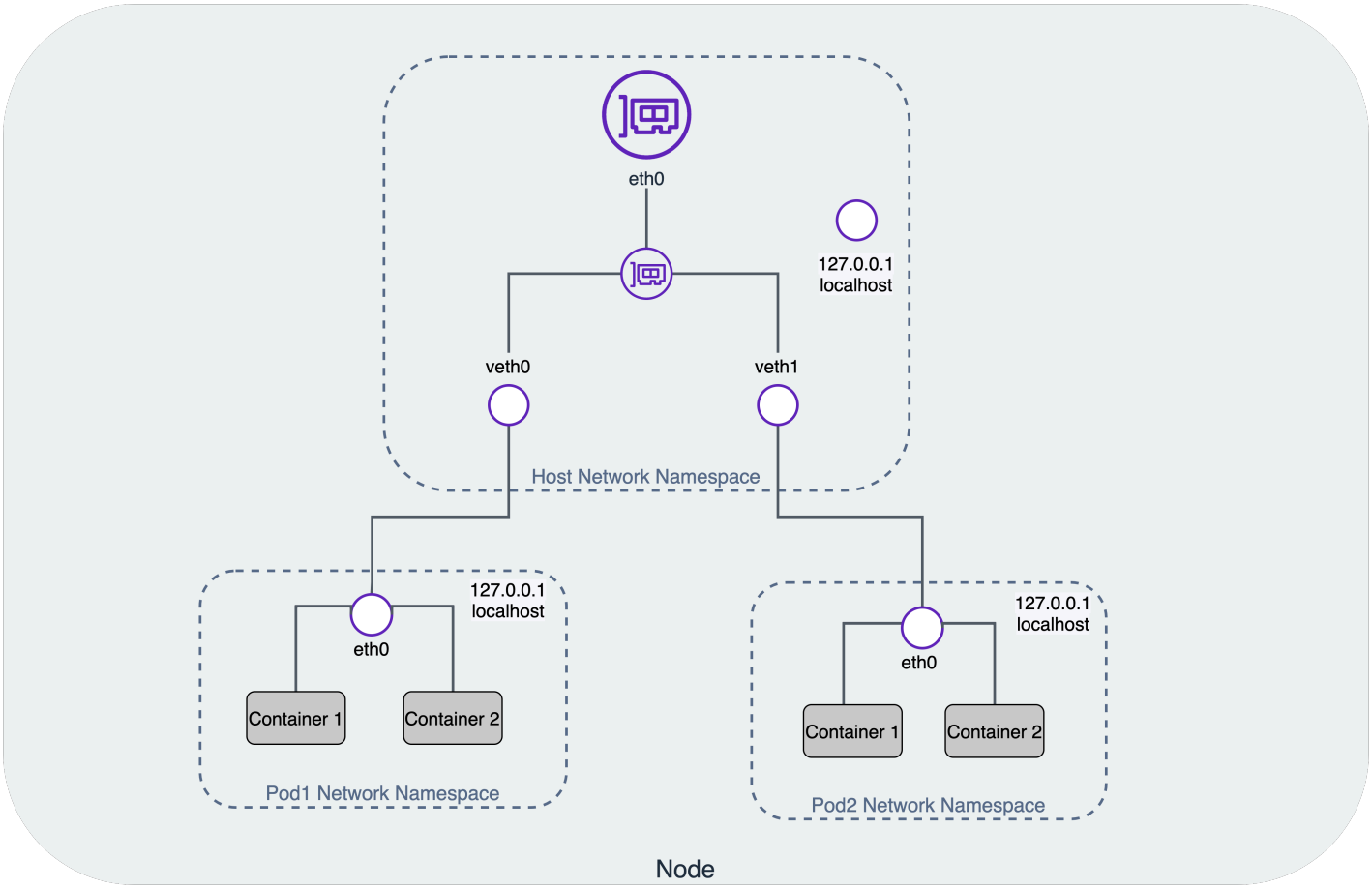
Kubernetes-Netzwerkmodell

Kubernetes stellt die folgenden Anforderungen an Cluster-Netzwerke:

- Pods, die auf demselben Knoten geplant sind, müssen in der Lage sein, mit anderen Pods zu kommunizieren, ohne NAT (Network Address Translation) zu verwenden.

- Alle System-Daemons (Hintergrundprozesse, z. B. Kubelet), die auf einem bestimmten Knoten ausgeführt werden, können mit den Pods kommunizieren, die auf demselben Knoten ausgeführt werden.
- Pods, die das Host-Netzwerk verwenden, müssen in der Lage sein, alle anderen Pods auf allen anderen Knoten zu kontaktieren, ohne NAT zu verwenden.

Einzelheiten darüber, was Kubernetes von kompatiblen Netzwerkimplementierungen erwartet, finden Sie im Kubernetes-Netzwerkmodell. Die folgende Abbildung veranschaulicht die Beziehung zwischen Pod-Netzwerk-Namespaces und dem Host-Netzwerk-Namespace.



Container-Netzwerkschnittstelle (CNI)

Kubernetes unterstützt CNI-Spezifikationen und Plugins zur Implementierung des Kubernetes-Netzwerkmodells. Ein CNI besteht aus einer Spezifikation (aktuelle Version 1.0.0) und Bibliotheken zum Schreiben von Plugins zur Konfiguration von Netzwerkschnittstellen in Containern sowie einer

Reihe unterstützter Plugins. CNI befasst sich ausschließlich mit der Netzwerkkonnektivität von Containern und dem Entfernen zugewiesener Ressourcen, wenn der Container gelöscht wird.

Das CNI-Plugin wird aktiviert, indem Kubelet die Befehlszeilenoption übergeben wird. `--network-plugin=cni` Kubelet liest eine Datei aus `--cni-conf-dir` (Standard/etc/cni/net.d) und verwendet die CNI-Konfiguration aus dieser Datei, um das Netzwerk jedes Pods einzurichten. Die CNI-Konfigurationsdatei muss der CNI-Spezifikation entsprechen (mindestens v0.4.0) und alle erforderlichen CNI-Plugins, auf die in der Konfiguration verwiesen wird, müssen im Verzeichnis vorhanden sein (Standard/). `--cni-bin-dir` opt/cni/bin Wenn das Verzeichnis mehrere CNI-Konfigurationsdateien enthält, verwendet das Kubelet die Konfigurationsdatei, deren Name in lexikografischer Reihenfolge an erster Stelle steht.

Amazon Virtual Private Cloud (VPC) CNI

Das von AWS bereitgestellte VPC CNI ist das Standard-Netzwerk-Add-On für EKS-Cluster. Das VPC CNI-Add-on wird standardmäßig installiert, wenn Sie EKS-Cluster bereitstellen. VPC CNI läuft auf Kubernetes-Worker-Knoten. Das VPC CNI-Add-on besteht aus der CNI-Binärdatei und dem IP-Adressverwaltungs-Plugin (ipamd). Das CNI weist einem Pod eine IP-Adresse aus dem VPC-Netzwerk zu. Das ipamd verwaltet AWS Elastic Networking Interfaces (ENIs) zu jedem Kubernetes-Knoten und verwaltet den warmen Pool von. IPs Das VPC CNI bietet Konfigurationsoptionen für die Vorzuweisung von ENIs und IP-Adressen für schnelle Pod-Startzeiten. Empfohlene Best Practices für die Plugin-Verwaltung finden Sie auf Amazon VPC CNI.

Amazon EKS empfiehlt, bei der Erstellung eines Clusters Subnetze in mindestens zwei Availability Zones anzugeben. Amazon VPC CNI weist Pods IP-Adressen aus den Knotensubnetzen zu. Wir empfehlen dringend, die Subnetze auf verfügbare IP-Adressen zu überprüfen. Bitte beachten Sie die VPC- und Subnetzempfehlungen, bevor Sie EKS-Cluster bereitstellen.

Amazon VPC CNI weist einen warmen Pool von ENIs und sekundären IP-Adressen aus dem Subnetz zu, das mit der primären ENI des Knotens verbunden ist. Dieser Modus von VPC CNI wird als sekundärer IP-Modus bezeichnet. Die Anzahl der IP-Adressen und damit die Anzahl der Pods (Pod-Dichte) wird durch die Anzahl ENIs und die IP-Adresse pro ENI (Grenzwerte) definiert, die durch den Instance-Typ definiert sind. Der sekundäre Modus ist der Standard und eignet sich gut für kleine Cluster mit kleineren Instance-Typen. Bitte erwägen Sie, den Präfixmodus zu verwenden, wenn Sie Probleme mit der Pod-Dichte haben. Sie können auch die verfügbaren IP-Adressen auf dem Knoten für Pods erhöhen, indem Sie Präfixe für zuweisen. ENIs

Amazon VPC CNI lässt sich nativ in AWS VPC integrieren und ermöglicht es Benutzern, bestehende bewährte AWS-VPC-Netzwerk- und Sicherheitsmethoden für den Aufbau von Kubernetes-Clustern anzuwenden. Dazu gehört die Möglichkeit, VPC-Flussprotokolle, VPC-Routing-Richtlinien und Sicherheitsgruppen für die Isolierung des Netzwerkverkehrs zu verwenden. Standardmäßig wendet das Amazon VPC CNI die Sicherheitsgruppe, die der primären ENI auf dem Knoten zugeordnet ist, auf die Pods an. Erwägen Sie die Aktivierung von Sicherheitsgruppen für Pods, wenn Sie einem Pod unterschiedliche Netzwerkregeln zuweisen möchten.

Standardmäßig weist VPC CNI Pods IP-Adressen aus dem Subnetz zu, das der primären ENI eines Knotens zugewiesen ist. Beim Betrieb großer Cluster mit Tausenden von Workloads kommt es häufig zu IPv4 Adressknappheit. Mit AWS VPC können Sie die Verfügbarkeit erweitern, IPs indem Sie eine Sekundärseite zuweisen, um die Erschöpfung der IPv4 CIDR-Blöcke CIDRs zu umgehen. Mit AWS VPC CNI können Sie einen anderen Subnetz-CIDR-Bereich für Pods verwenden. Diese Funktion von VPC CNI wird als benutzerdefiniertes Netzwerk bezeichnet. Sie könnten erwägen, benutzerdefinierte Netzwerke zu verwenden, um `100.64.0.0/10` und `198.19.0.0/16` CIDRs (CG-NAT) mit EKS zu verwenden. Auf diese Weise können Sie effektiv eine Umgebung schaffen, in der Pods keine RFC1918 IP-Adressen mehr von Ihrer VPC verbrauchen.

Benutzerdefinierte Netzwerke sind eine Option, um das Problem der IPv4 Adressüberflutung zu lösen, aber sie erfordern betrieblichen Mehraufwand. Wir empfehlen IPv6 Cluster anstelle von benutzerdefinierten Netzwerken, um dieses Problem zu lösen. Insbesondere empfehlen wir die Migration zu IPv6 Clustern, wenn Sie den gesamten verfügbaren IPv4 Adressraum für Ihre VPC vollständig ausgeschöpft haben. Prüfen Sie, welche Pläne Ihr Unternehmen zu unterstützen hat IPv6, und überlegen Sie, ob sich Investitionen in diese Produkte auf lange Sicht lohnen IPv6 könnten.

Die Unterstützung von EKS für konzentriert IPv6 sich auf die Lösung des Problems der IP-Erschöpfung, das durch einen begrenzten IPv4 Adressraum verursacht wird. Als Reaktion auf Kundenprobleme mit Überlastung hat EKS ausschließlich IPv4 Pods mit zwei Stacks Vorrang vor Pods mit IPv6 nur einem Stack eingeräumt. Das heißt, Pods können möglicherweise auf IPv4 Ressourcen zugreifen, ihnen wird jedoch keine IPv4 Adresse aus dem VPC-CIDR-Bereich zugewiesen. Die VPC-CNI weist Pods IPv6 Adressen aus dem von AWS verwalteten IPv6 VPC-CIDR-Block zu.

Subnetz-Rechner

Dieses Projekt enthält ein Excel-Dokument für den Subnetzrechner. Dieses Rechendokument simuliert den IP-Adressverbrauch eines bestimmten Workloads unter verschiedenen ENI-Konfigurationsoptionen wie `WARM_IP_TARGET` und. `WARM_ENI_TARGET` Das Dokument umfasst

zwei Blätter, ein erstes für den Warm-ENI-Modus und ein zweites für den Warm-IP-Modus. Weitere Informationen zu diesen Modi finden Sie in den VPC CNI-Leitlinien.

Eingänge:

- CIDR-Größe des Subnetzes
- Warmes ENI-Ziel oder warmes IP-Ziel
- Liste der Instanzen
  - Typ, Anzahl und Anzahl der pro Instanz geplanten Workload-Pods

Ausgaben:

- Gesamtzahl der gehosteten Pods
- Anzahl der genutzten Subnetze IPs
- Anzahl der verbleibenden Subnetze IPs
- Details auf Instanzebene
  - Anzahl der Warms IPs/ENIs pro Instanz
  - Anzahl der Aktiven IPs/ENIs pro Instanz

Überlegungen zu VPC und Subnetzen

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Der Betrieb eines EKS-Clusters erfordert neben Kubernetes-Netzwerken auch Kenntnisse über AWS-VPC-Netzwerke.

Wir empfehlen Ihnen, sich mit den Kommunikationsmechanismen der EKS-Kontrollebene vertraut zu machen, bevor Sie mit dem Entwurf Ihrer VPC oder der Bereitstellung von Clustern in bestehenden VPCs Systemen beginnen.

Weitere Informationen finden Sie unter Überlegungen zur Cluster-VPC und Überlegungen zu Amazon EKS-Sicherheitsgruppen bei der Architektur einer VPC und von Subnetzen, die mit EKS verwendet werden sollen.

-Übersicht

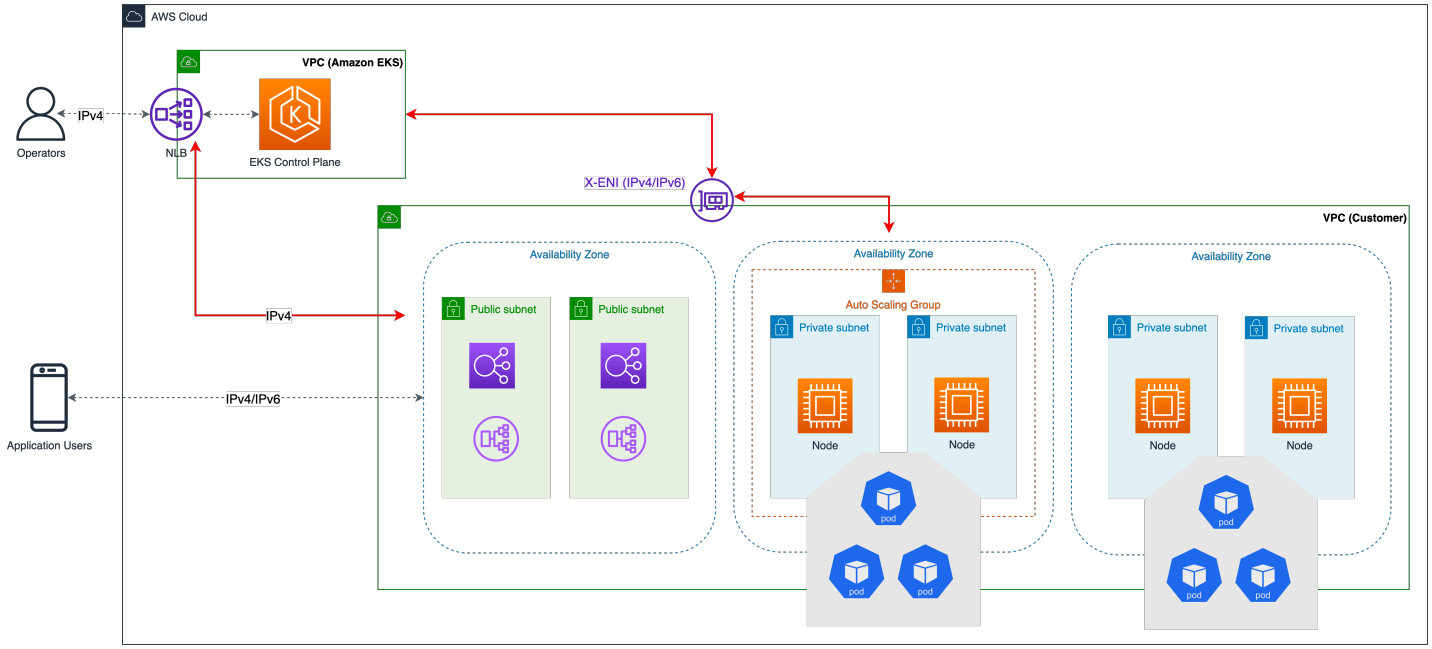
EKS-Cluster-Architektur

Ein EKS-Cluster besteht aus zwei VPCs:

- Eine von AWS verwaltete VPC, die die Kubernetes-Steuerebene hostet. Diese VPC erscheint nicht im Kundenkonto.
- Eine vom Kunden verwaltete VPC, die die Kubernetes-Knoten hostet. Hier laufen Container sowie andere vom Kunden verwaltete AWS-Infrastrukturen wie Load Balancer, die vom Cluster verwendet werden. Diese VPC erscheint im Kundenkonto. Sie müssen eine vom Kunden verwaltete VPC erstellen, bevor Sie einen Cluster erstellen. Das eksctl erstellt eine VPC, wenn Sie keine angeben.

Die Knoten in der Kunden-VPC müssen in der Lage sein, eine Verbindung zum verwalteten API-Serverendpunkt in der AWS-VPC herzustellen. Auf diese Weise können sich die Knoten bei der Kubernetes-Steuerebene registrieren und Anfragen zur Ausführung von Anwendungs-Pods empfangen.

Die Knoten stellen über (a) einen öffentlichen EKS-Endpunkt oder (b) ein von EKS verwaltetes Cross-Account-Elastic Network Interfaces (X-ENI) eine Verbindung zur EKS-Steuerebene her. Wenn ein Cluster erstellt wird, müssen Sie mindestens zwei VPC-Subnetze angeben. EKS platziert eine X-ENI in jedem Subnetz, das bei der Clustererstellung angegeben wurde (auch Cluster-Subnetze genannt). Der Kubernetes-API-Server verwendet diese Cros-Accounts, um mit Knoten ENIs zu kommunizieren, die in den vom Kunden verwalteten Cluster-VPC-Subnetzen bereitgestellt werden.



Beim Start des Knotens wird das EKS-Bootstrap-Skript ausgeführt und die Kubernetes-Knotenkonfigurationsdateien werden installiert. Als Teil des Startvorgangs auf jeder Instanz werden die Container-Runtime-Agents, Kubelet- und Kubernetes-Node-Agents gestartet.

Um einen Knoten zu registrieren, kontaktiert Kubelet den Kubernetes-Cluster-Endpunkt. Es stellt eine Verbindung entweder mit dem öffentlichen Endpunkt außerhalb der VPC oder dem privaten Endpunkt innerhalb der VPC her. Kubelet erhält API-Anweisungen und sendet regelmäßig Statusaktualisierungen und Heartbeats an den Endpunkt.

Kommunikation auf der EKS-Kontrollebene

EKS bietet zwei Möglichkeiten, den Zugriff auf den Cluster-Endpunkt zu kontrollieren. Mit der Endpoint Access Control können Sie wählen, ob der Endpunkt über das öffentliche Internet oder nur über Ihre VPC erreicht werden kann. Sie können den öffentlichen Endpunkt (was die Standardeinstellung ist), den privaten Endpunkt oder beide gleichzeitig aktivieren.

Die Konfiguration des Cluster-API-Endpunkts bestimmt den Pfad, den Knoten nehmen, um mit der Steuerungsebene zu kommunizieren. Beachten Sie, dass diese Endpunkteinstellungen jederzeit über die EKS-Konsole oder API geändert werden können.

Öffentlicher Endpunkt

Dies ist das Standardverhalten für neue Amazon-EKS-Cluster. Wenn nur der öffentliche Endpunkt für den Cluster aktiviert ist, verlassen Kubernetes-API-Anfragen, die aus der VPC Ihres Clusters

stammen (z. B. der Worker-Node zur Steuerung der Ebenenkommunikation), die VPC, nicht jedoch das Amazon-Netzwerk. Damit Knoten eine Verbindung zur Steuerungsebene herstellen können, benötigen sie eine öffentliche IP-Adresse und eine Route zu einem Internet-Gateway oder eine Route zu einem NAT-Gateway, wo sie die öffentliche IP-Adresse des NAT-Gateways verwenden können.

Öffentlicher und privater Endpunkt

Wenn sowohl der öffentliche als auch der private Endpunkt aktiviert sind, kommunizieren Kubernetes-API-Anfragen innerhalb der VPC über das X- ENIs innerhalb Ihrer VPC mit der Steuerungsebene. Ihr Cluster-API-Server ist über das Internet erreichbar.

Privater Endpunkt

Es gibt keinen öffentlichen Zugriff auf Ihren API-Server über das Internet, wenn nur der private Endpunkt aktiviert ist. Der gesamte Datenverkehr zu Ihrem Cluster-API-Server muss aus der VPC des Clusters oder einem verbundenen Netzwerk stammen. Die Knoten kommunizieren ENIs innerhalb Ihrer VPC über X- mit dem API-Server. Beachten Sie, dass Clusterverwaltungstools Zugriff auf den privaten Endpunkt haben müssen. Erfahren Sie mehr darüber, wie Sie von außerhalb der Amazon VPC eine Verbindung zu einem privaten Amazon EKS-Cluster-Endpunkt herstellen können.

Beachten Sie, dass der API-Serverendpunkt des Clusters von öffentlichen DNS-Servern in eine private IP-Adresse von der VPC aufgelöst wird. In der Vergangenheit konnte der Endpunkt nur innerhalb der VPC aufgelöst werden.

VPC-Konfigurationen

Amazon VPC unterstützt IPv4 und IPv6 adressiert. Amazon EKS unterstützt IPv4 standardmäßig. Einer VPC muss ein IPv4 CIDR-Block zugeordnet sein. Sie können Ihrer VPC optional mehrere IPv4 Classless Inter-Domain Routing (CIDR) -Blöcke und mehrere IPv6 CIDR-Blöcke zuordnen. Wenn Sie eine VPC erstellen, müssen Sie einen IPv4 CIDR-Block für die VPC aus den privaten IPv4 Adressbereichen angeben, wie in RFC 1918 angegeben. Die zulässige Blockgröße liegt zwischen einem `/16` Präfix (65.536 IP-Adressen) und `/28` einem Präfix (16 IP-Adressen).

Wenn Sie eine neue VPC erstellen, können Sie einen einzelnen IPv6 CIDR-Block anhängen und bis zu fünf, wenn Sie eine bestehende VPC ändern. Die Präfixlänge der IPv6 CIDR-Blockgröße kann zwischen /44 und /60 und für die IPv6 Subnetze zwischen /44/ und /64 liegen. Sie können einen IPv6 CIDR-Block aus dem von Amazon verwalteten IPv6 Adresspool anfordern. Weitere Informationen finden Sie im Abschnitt VPC CIDR-Blöcke des VPC-Benutzerhandbuchs.

Amazon EKS-Cluster unterstützen IPv4 sowohl als auch IPv6. Standardmäßig verwenden EKS-Cluster IPv4 IP. Wenn Sie dies IPv6 bei der Clustererstellung angeben, können Sie IPv6 Cluster verwenden. IPv6 Cluster benötigen Dual-Stack VPCs und Subnetze.

Amazon EKS empfiehlt, bei der Clustererstellung mindestens zwei Subnetze zu verwenden, die sich in unterschiedlichen Availability Zones befinden. Die Subnetze, die Sie bei der Clustererstellung übergeben, werden als Cluster-Subnetze bezeichnet. Wenn Sie einen Cluster erstellen, erstellt Amazon EKS bis zu 4 kontoübergreifende Konten (X-Konto oder X-ENIs) ENIs in den von Ihnen angegebenen Subnetzen. Die X- ENIs werden immer bereitgestellt und für den Cluster-Administrationsdatenverkehr wie Protokollzustellung, Exec und Proxy verwendet. Vollständige Informationen zu den VPC- und Subnetzanforderungen finden Sie im EKS-Benutzerhandbuch.

Kubernetes-Worker-Knoten können in den Cluster-Subnetzen ausgeführt werden, dies wird jedoch nicht empfohlen. Während Cluster-Upgrades stellt Amazon EKS zusätzliche Subnetze ENIs in den Cluster-Subnetzen bereit. Wenn Ihr Cluster horizontal skaliert wird, verbrauchen Worker-Knoten und Pods möglicherweise die IPs im Cluster-Subnetz verfügbaren Ressourcen. Um sicherzustellen, dass genügend verfügbar sind, sollten IPs Sie daher die Verwendung von dedizierten Cluster-Subnetzen mit der Netzmaske /28 in Betracht ziehen.

Kubernetes-Worker-Knoten können entweder in einem öffentlichen oder einem privaten Subnetz ausgeführt werden. Ob ein Subnetz öffentlich oder privat ist, bezieht sich darauf, ob der Verkehr innerhalb des Subnetzes über ein Internet-Gateway geleitet wird. Öffentliche Subnetze haben einen Routing-Tabelleneintrag zum Internet über das Internet-Gateway, private Subnetze jedoch nicht.

Der Verkehr, der woanders entsteht und Ihre Knoten erreicht, wird als Ingress bezeichnet. Der Verkehr, der von den Knoten ausgeht und das Netzwerk verlässt, wird als Egress bezeichnet. Knoten mit öffentlichen oder elastischen IP-Adressen (EIPs) innerhalb eines Subnetzes, das mit einem Internet-Gateway konfiguriert ist, ermöglichen den Zugriff von außerhalb der VPC. Private Subnetze verfügen in der Regel über ein Routing zu einem NAT-Gateway, das eingehenden Datenverkehr zu den Knoten in den Subnetzen von außerhalb der VPC nicht zulässt, während der Datenverkehr von den Knoten die VPC verlassen kann (Egress).

IPv6 Weltweit ist jede Adresse über das Internet routingfähig. Die IPv6 Adressen, die den Knoten und Pods zugeordnet sind, sind öffentlich. Private Subnetze werden durch die Implementierung von nur ausgehenden Internet-Gateways (EIGW) in einer VPC unterstützt, die ausgehenden Datenverkehr zulassen und gleichzeitig den gesamten eingehenden Verkehr blockieren. Bewährte Methoden für die Implementierung von IPv6 Subnetzen finden Sie im VPC-Benutzerhandbuch.

Sie können VPC und Subnetze auf drei verschiedene Arten konfigurieren:

Nur öffentliche Subnetze verwenden

In denselben öffentlichen Subnetzen werden sowohl Knoten als auch Eingangsressourcen (wie Load Balancer) erstellt. Markieren Sie das öffentliche Subnetz mit, um Load Balancer `kubernetes.io/role/elb`zu erstellen, die mit dem Internet verbunden sind. In dieser Konfiguration kann der Cluster-Endpunkt als öffentlich, privat oder beides (öffentlich und privat) konfiguriert werden.

Verwendung von privaten und öffentlichen Subnetzen

Knoten werden in privaten Subnetzen erstellt, wohingegen Ingress-Ressourcen in öffentlichen Subnetzen instanziiert werden. Sie können öffentlichen, privaten oder beides (öffentlich und privat) Zugriff auf den Cluster-Endpunkt aktivieren. Abhängig von der Konfiguration des Cluster-Endpunkts wird der Knotenverkehr über das NAT-Gateway oder die ENI eingegeben.

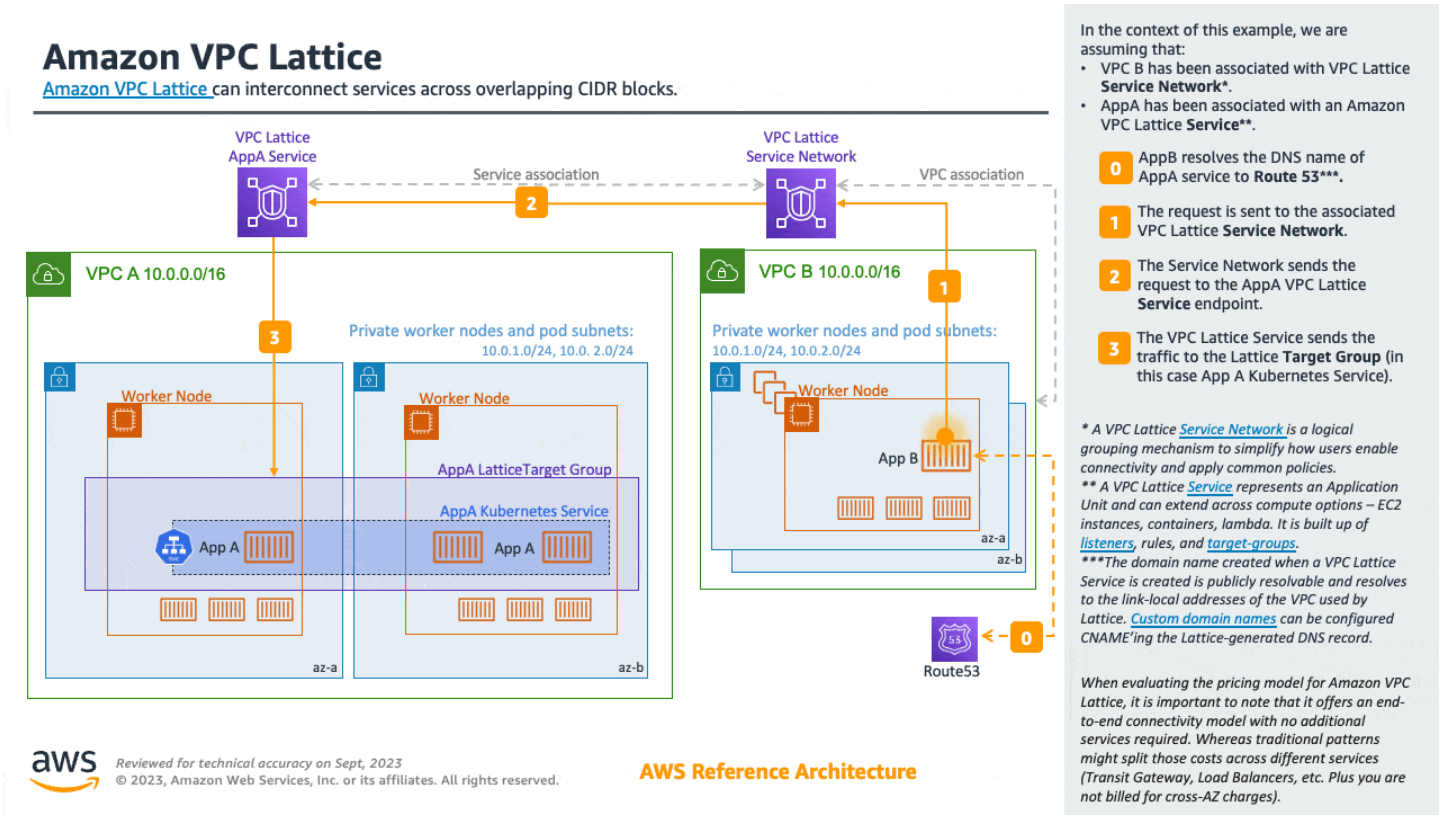
Es werden nur private Subnetze verwendet

Sowohl Knoten als auch Ingress werden in privaten Subnetzen erstellt. Verwendung des `kubernetes.io/role/internal-elb`Subnetz-Tags zum Aufbau interner Load Balancer. Für den Zugriff auf den Endpunkt Ihres Clusters ist eine VPN-Verbindung erforderlich. Sie müssen AWS PrivateLink für EC2 und alle Amazon ECR- und S3-Repositorys aktivieren. Nur der private Endpunkt des Clusters sollte aktiviert werden. Wir empfehlen, die Anforderungen für private EKS-Cluster durchzugehen, bevor Sie private Cluster bereitstellen.

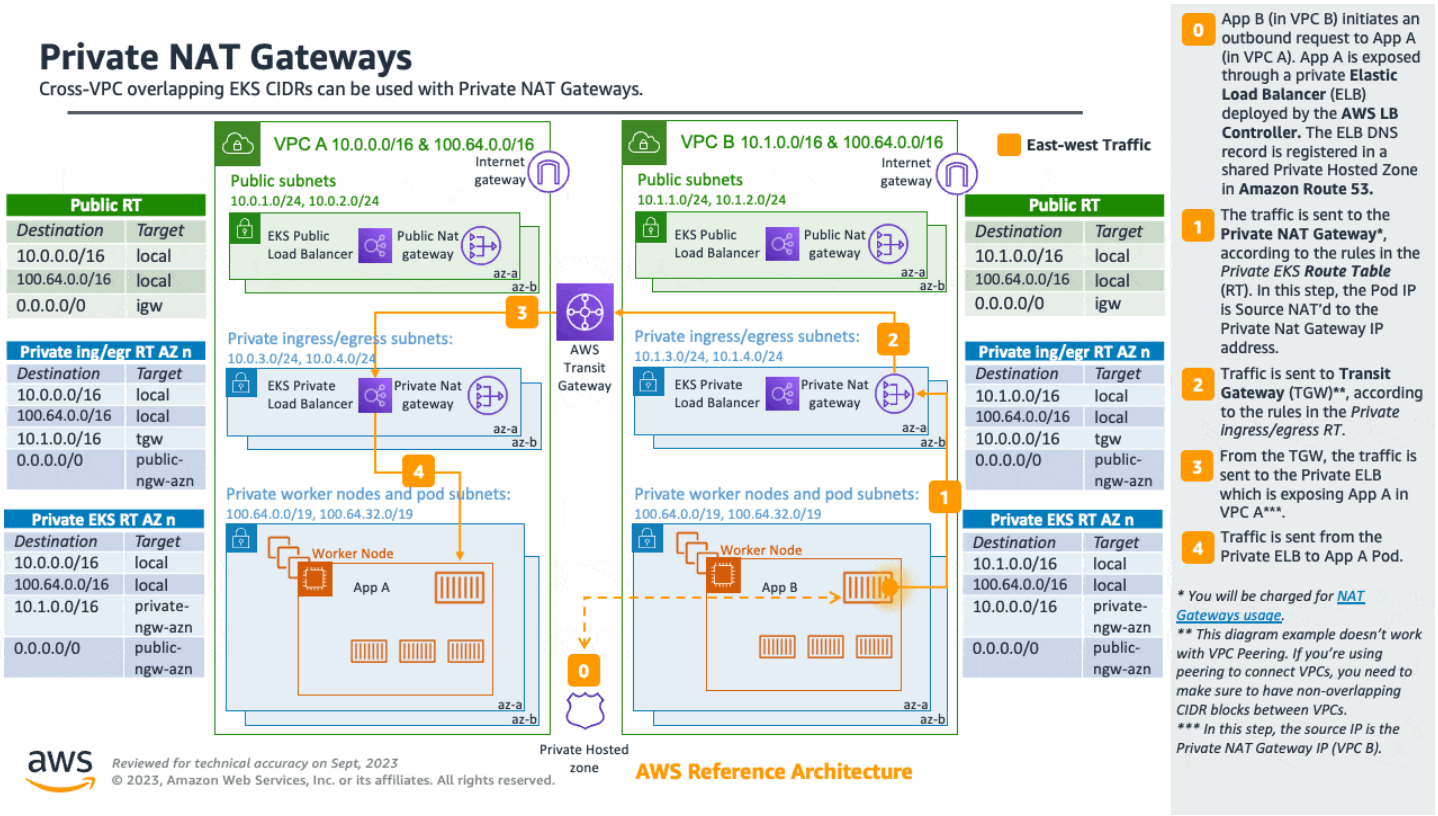
Kommunikation zwischen VPCs

Es gibt viele Szenarien, in denen mehrere VPCs und separate EKS-Cluster für diese bereitgestellt werden müssen VPCs.

Sie können Amazon VPC Lattice verwenden, um Dienste konsistent und sicher über mehrere VPCs Konten hinweg zu verbinden (ohne dass zusätzliche Konnektivität durch Dienste wie VPC Peering, AWS PrivateLink oder AWS Transit Gateway bereitgestellt werden muss). Erfahren Sie hier mehr.



Amazon VPC Lattice arbeitet im Link-Local-Adressraum in IPv4 und und bietet Konnektivität zwischen Diensten IPv6, die sich möglicherweise überschneiden. IPv4 Aus Gründen der betrieblichen Effizienz empfehlen wir dringend, EKS-Cluster und -Knoten in IP-Bereichen bereitzustellen, die sich nicht überschneiden. Falls Ihre Infrastruktur VPCs überlappende IP-Bereiche umfasst, müssen Sie Ihr Netzwerk entsprechend gestalten. Wir empfehlen Private NAT Gateway oder VPC CNI im benutzerdefinierten Netzwerkmodus in Verbindung mit einem Transit-Gateway, um Workloads auf EKS zu integrieren, um überlappende CIDR-Herausforderungen zu lösen und gleichzeitig routingfähige IP-Adressen beizubehalten. RFC1918

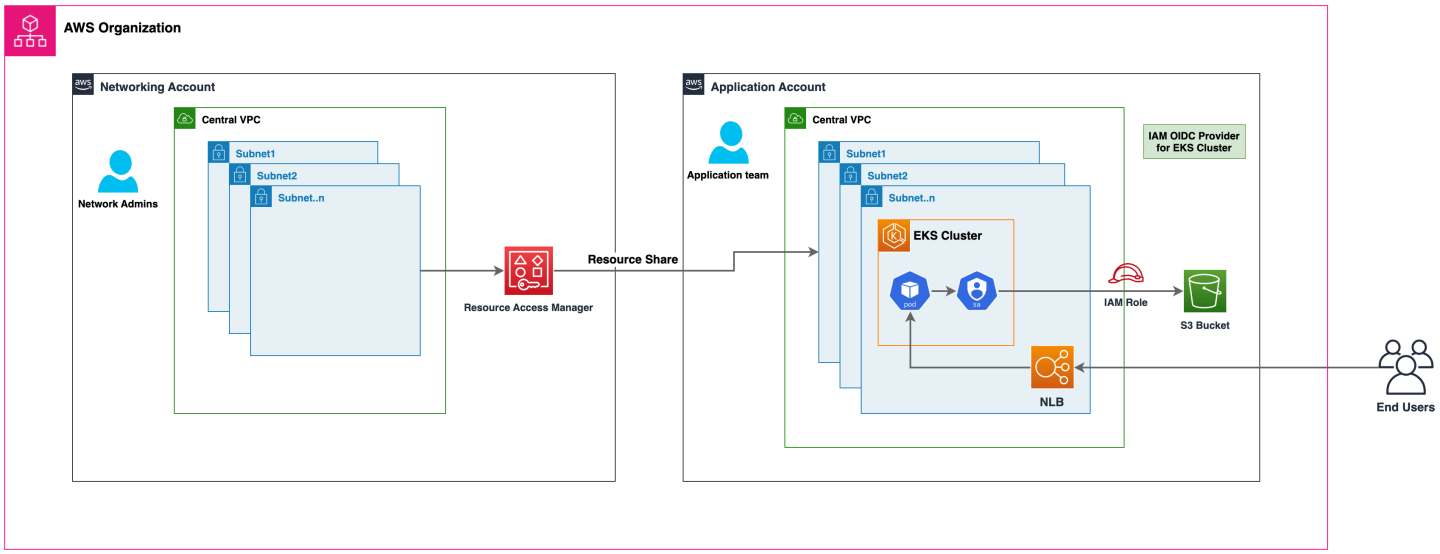


Erwägen Sie die Nutzung von AWS PrivateLink, auch bekannt als Endpunkt-Service, wenn Sie der Service Provider sind und Ihren Kubernetes-Service und Ihren Ingress (entweder ALB oder NLB) in separaten Konten mit Ihrer Kunden-VPC teilen möchten.

Gemeinsame Nutzung von VPC für mehrere Konten

Viele Unternehmen haben Amazon VPCs gemeinsam genutzt, um die Netzwerkadministration zu optimieren, Kosten zu senken und die Sicherheit mehrerer AWS-Konten in einer AWS-Organisation zu verbessern. Sie nutzen AWS Resource Access Manager (RAM), um unterstützte AWS-Ressourcen sicher mit einzelnen AWS-Konten, Organisationseinheiten (OUs) oder der gesamten AWS-Organisation zu teilen.

Sie können Amazon EKS-Cluster, verwaltete Knotengruppen und andere unterstützende AWS-Ressourcen (wie LoadBalancers Sicherheitsgruppen, Endpunkte usw.) in gemeinsam genutzten VPC-Subnetzen von einem anderen AWS-Konto aus mithilfe von AWS-RAM bereitstellen. Die folgende Abbildung zeigt ein Beispiel für eine High-Level-Architektur. Dies ermöglicht zentralen Netzwerkteams die Kontrolle über die Netzwerkkonstrukte wie VPCs Subnetze usw., während Anwendungs- oder Plattformteams gleichzeitig Amazon EKS-Cluster in ihren jeweiligen AWS-Konten bereitstellen können. Eine vollständige Anleitung dieses Szenarios ist in diesem Github-Repository verfügbar.



Überlegungen bei der Verwendung gemeinsam genutzter Subnetze

- Amazon EKS-Cluster und Worker-Knoten können in gemeinsam genutzten Subnetzen erstellt werden, die alle Teil derselben VPC sind. Amazon EKS unterstützt nicht die Erstellung von Clustern über mehrere VPCs.
- Amazon EKS verwendet AWS VPC Security Groups (SGs), um den Verkehr zwischen der Kubernetes-Steuerebene und den Worker-Knoten des Clusters zu kontrollieren. Sicherheitsgruppen werden auch verwendet, um den Verkehr zwischen Worker-Knoten und anderen VPC-Ressourcen sowie externen IP-Adressen zu kontrollieren. Sie müssen diese Sicherheitsgruppen im application/participant Konto erstellen. Stellen Sie sicher, dass sich die Sicherheitsgruppen, die Sie für Ihre Pods verwenden möchten, auch im Teilnehmerkonto befinden. Sie können die Regeln für eingehenden und ausgehenden Datenverkehr innerhalb Ihrer Sicherheitsgruppen so konfigurieren, dass der erforderliche Datenverkehr zu und von Sicherheitsgruppen im zentralen VPC-Konto zugelassen wird.
- Erstellen Sie IAM-Rollen und zugehörige Richtlinien innerhalb des Teilnehmerkontos, in dem sich Ihr Amazon EKS-Cluster befindet. Diese IAM-Rollen und -Richtlinien sind unerlässlich, um den von Amazon EKS verwalteten Kubernetes-Clustern sowie den auf Fargate laufenden Knoten und Pods die erforderlichen Berechtigungen zu gewähren. Die Berechtigungen ermöglichen es Amazon EKS, in Ihrem Namen Anrufe an andere AWS-Services zu tätigen.
- Sie können die folgenden Ansätze verfolgen, um den kontoübergreifenden Zugriff auf AWS-Ressourcen wie Amazon S3 S3-Buckets, Dynamodb-Tabellen usw. von k8s-Pods aus zu ermöglichen:

  - Ressourcenbasierter Richtlinienansatz: Wenn der AWS-Service Ressourcenrichtlinien unterstützt, können Sie entsprechende ressourcenbasierte Richtlinien hinzufügen, um kontoübergreifenden Zugriff auf IAM-Rollen zu ermöglichen, die den Kubernetes-Pods zugewiesen sind. In diesem Szenario sind OIDC-Anbieter, IAM-Rollen und Berechtigungsrichtlinien im Anwendungskonto vorhanden. AWS-Services, die ressourcenbasierte Richtlinien unterstützen, finden Sie unter AWS-Services, die mit IAM funktionieren, und suchen Sie in der Spalte Ressourcenbasiert nach den Services, für die Ja steht.
  - OIDC-Anbieter-Ansatz: IAM-Ressourcen wie OIDC-Anbieter, IAM-Rollen, Genehmigungs- und Vertrauensrichtlinien werden in den AWS-Konten anderer Teilnehmer erstellt, in denen die Ressourcen vorhanden sind. Diese Rollen werden den Kubernetes-Pods im Anwendungskonto zugewiesen, sodass sie auf kontoübergreifende Ressourcen zugreifen können. Eine vollständige Anleitung zu diesem Ansatz finden Sie im Blog Kontenübergreifende IAM-Rollen für Kubernetes-Dienstkonten.
- Sie können die Amazon Elastic Loadbalancer (ELB) -Ressourcen (ALB oder NLB) bereitstellen, um den Datenverkehr entweder in Anwendungs- oder zentralen Netzwerkkonten an k8s-Pods weiterzuleiten. Ausführliche Anweisungen zur Bereitstellung der ELB-Ressourcen im zentralen Netzwerkkonto finden Sie unter Expose Amazon EKS Pods Through Account Load Balancer Walkthrough Expose Amazon EKS Pods Through Account Load Balancer. Diese Option bietet mehr Flexibilität, da sie dem Central Networking-Konto die volle Kontrolle über die Sicherheitskonfiguration der Load Balancer-Ressourcen gewährt.
- Wenn Sie Amazon VPC CNI verwendencustom networking feature, müssen Sie die ID-Zuordnungen der Availability Zone (AZ) verwenden, die im zentralen Netzwerkkonto aufgeführt sind, um jede Zuordnung zu erstellen. ENIConfig Dies ist auf die zufällige Zuordnung von physischen AZs und AZ-Namen in jedem AWS-Konto zurückzuführen.

Sicherheitsgruppen

Eine Sicherheitsgruppe steuert den Datenverkehr, der die Ressourcen erreichen und verlassen darf, mit denen er verknüpft ist. Amazon EKS verwendet Sicherheitsgruppen, um die Kommunikation zwischen der Kontrollebene und den Knoten zu verwalten. Wenn Sie einen Cluster erstellen, erstellt Amazon EKS eine Sicherheitsgruppe mit dem Namen `eks-cluster-sg-my-cluster-uniqueID`. EKS ordnet diese Sicherheitsgruppen den verwalteten Knoten ENIs und den Knoten zu. Die Standardregeln ermöglichen es, dass der gesamte Datenverkehr frei zwischen Ihrem Cluster und Ihren Knoten fließt. Außerdem lassen sie den gesamten ausgehenden Datenverkehr zu jedem Ziel zu.

Wenn Sie einen Cluster erstellen, können Sie Ihre eigenen Sicherheitsgruppen angeben. Bitte beachten Sie die Empfehlung für Sicherheitsgruppen, wenn Sie eigene Sicherheitsgruppen angeben.

Empfehlungen

Ziehen Sie eine Multi-AZ-Bereitstellung in Betracht

AWS-Regionen bieten mehrere physisch getrennte und isolierte Availability Zones (AZ), die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind. Mit Availability Zones können Sie Anwendungen entwerfen und betreiben, die automatisch und ohne Unterbrechung ein Failover zwischen Availability Zones durchführen. Amazon EKS empfiehlt dringend, EKS-Cluster in mehreren Availability Zones bereitzustellen. Bitte erwägen Sie, bei der Erstellung des Clusters Subnetze in mindestens zwei Availability Zones anzugeben.

Kubelet, das auf Knoten ausgeführt wird, fügt dem Knotenobjekt automatisch Labels hinzu, wie z. `topology.kubernetes.io/region=us-west-2` Wir empfehlen, Node-Labels in Verbindung mit den Beschränkungen der Pod-Topologie zu verwenden, um zu kontrollieren, wie Pods über Zonen verteilt werden. Diese Hinweise ermöglichen es dem Kubernetes-Scheduler, Pods so zu platzieren, dass die erwartete Verfügbarkeit besser ist. Dadurch wird das Risiko verringert, dass sich ein korrelierter Ausfall auf Ihren gesamten Workload auswirkt. Beispiele für Node-Selector- und AZ-Spread-Beschränkungen finden Sie unter Zuweisen von Knoten zu Pods.

Sie können die Subnetze oder Verfügbarkeitszonen definieren, wenn Sie Knoten erstellen. Die Knoten werden in Cluster-Subnetzen platziert, wenn keine Subnetze konfiguriert sind. Die EKS-Unterstützung für verwaltete Knotengruppen verteilt die Knoten je nach verfügbarer Kapazität automatisch auf mehrere Availability Zones. Karpenter berücksichtigt die AZ-Spread-Platzierung, indem die Knoten auf die angegebenen Werte skaliert werden, AZs sofern Workloads die Topologie-Spread-Grenzen definieren.

AWS Elastic Load Balancers werden vom AWS Load Balancer Controller für einen Kubernetes-Cluster verwaltet. Es stellt einen Application Load Balancer (ALB) für Kubernetes-Eingangsressourcen und einen Network Load Balancer (NLB) für Kubernetes-Dienste vom Typ Loadbalancer bereit. Der Elastic Load Balancer Balancer-Controller verwendet Tags, um die Subnetze zu erkennen. Der ELB-Controller benötigt mindestens zwei Availability Zones (AZs), um eingehende Ressourcen erfolgreich bereitzustellen. Erwägen Sie die Einrichtung von Subnetzen in mindestens zwei AZs , um die Sicherheit und Zuverlässigkeit geografischer Redundanz zu nutzen.

Stellen Sie Knoten in privaten Subnetzen bereit

Eine VPC, die sowohl private als auch öffentliche Subnetze umfasst, ist die ideale Methode für die Bereitstellung von Kubernetes-Workloads auf EKS. Erwägen Sie, mindestens zwei öffentliche Subnetze und zwei private Subnetze in zwei unterschiedlichen Verfügbarkeitszonen einzurichten. Die zugehörige Routentabelle eines öffentlichen Subnetzes enthält eine Route zu einem Internet-Gateway. Pods können über ein NAT-Gateway mit dem Internet interagieren. Private Subnetze werden von Internet-Gateways in der Umgebung (EIGW) unterstützt, die IPv6 nur für ausgehenden Datenverkehr bestimmt sind.

Die Instanziierung von Knoten in privaten Subnetzen bietet maximale Kontrolle über den Datenverkehr zu den Knoten und ist für die überwiegende Mehrheit der Kubernetes-Anwendungen wirksam. Eingangsressourcen (wie Load Balancer) werden in öffentlichen Subnetzen instanziiert und leiten den Datenverkehr an Pods weiter, die in privaten Subnetzen betrieben werden.

Ziehen Sie den Modus „Nur privat“ in Betracht, wenn Sie strenge Sicherheitsvorkehrungen und Netzwerkisolierung benötigen. In dieser Konfiguration werden drei private Subnetze in unterschiedlichen Availability Zones innerhalb der VPC der AWS-Region bereitgestellt. Die in den Subnetzen bereitgestellten Ressourcen können nicht auf das Internet zugreifen, und das Internet kann auch nicht auf die Ressourcen in den Subnetzen zugreifen. Damit Ihre Kubernetes-Anwendung auf andere AWS-Services zugreifen kann, müssen Sie PrivateLink Schnittstellen, and/or Gateway-Endpunkte konfigurieren. Sie können interne Load Balancer einrichten, um den Datenverkehr mithilfe des AWS Load Balancer Controllers zu Pods umzuleiten. Die privaten Subnetze müssen mit (`kubernetes.io/role/internal-elb: 1`) gekennzeichnet sein, damit der Controller Load Balancer bereitstellen kann. Damit sich Knoten beim Cluster registrieren können, muss der Cluster-Endpunkt auf den privaten Modus gesetzt werden. Vollständige Anforderungen und Überlegungen finden Sie im Leitfaden für private Cluster.

Ziehen Sie den öffentlichen und privaten Modus für Cluster-Endpunkte in Betracht

Amazon EKS bietet Cluster-Endpunktmodi public-and-private nur öffentlich und nur privat. Der Standardmodus ist nur öffentlich. Wir empfehlen jedoch, den Cluster-Endpunkt im öffentlichen und privaten Modus zu konfigurieren. Mit dieser Option können Kubernetes-API-Aufrufe innerhalb der VPC Ihres Clusters (z. B. node-to-control-plane Kommunikation) den privaten VPC-Endpunkt nutzen und der Datenverkehr bleibt innerhalb der VPC Ihres Clusters. Ihr Cluster-API-Server kann dagegen über das Internet erreicht werden. Wir empfehlen jedoch dringend, die CIDR-Blöcke einzuschränken, die den öffentlichen Endpunkt verwenden können. Erfahren Sie, wie Sie den Zugriff auf öffentliche und private Endgeräte konfigurieren, einschließlich der Begrenzung von CIDR-Blöcken.

Wir empfehlen einen ausschließlich privaten Endpunkt, wenn Sie Sicherheit und Netzwerkisolierung benötigen. Wir empfehlen, eine der im EKS-Benutzerhandbuch aufgeführten Optionen zu verwenden, um eine private Verbindung zu einem API-Server herzustellen.

Konfigurieren Sie Sicherheitsgruppen sorgfältig

Amazon EKS unterstützt die Verwendung benutzerdefinierter Sicherheitsgruppen. Alle benutzerdefinierten Sicherheitsgruppen müssen die Kommunikation zwischen Knoten und der Kubernetes-Steuerebene ermöglichen. Bitte überprüfen Sie die Portanforderungen und konfigurieren Sie die Regeln manuell, wenn Ihre Organisation keine offene Kommunikation zulässt.

EKS wendet die benutzerdefinierten Sicherheitsgruppen, die Sie bei der Clustererstellung angeben, auf die verwalteten Schnittstellen (X-ENIs) an. Es ordnet sie jedoch nicht sofort Knoten zu. Beim Erstellen von Knotengruppen wird dringend empfohlen, benutzerdefinierte Sicherheitsgruppen manuell zuzuordnen. Bitte erwägen Sie, securityGroupSelectorTerms zu aktivieren, um die Erkennung benutzerdefinierter Sicherheitsgruppen durch Karpenter bei der automatischen Skalierung von Knoten zu ermöglichen.

Wir empfehlen dringend, eine Sicherheitsgruppe zu erstellen, um den gesamten Kommunikationsverkehr zwischen den Knoten zuzulassen. Während des Bootstrap-Vorgangs benötigen Knoten eine ausgehende Internetverbindung, um auf den Cluster-Endpunkt zugreifen zu können. Bewerten Sie die Anforderungen für den ausgehenden Zugriff, z. B. die Verbindung vor Ort und den Zugriff auf die Container-Registry, und legen Sie die Regeln entsprechend fest. Bevor Sie Änderungen in der Produktionsumgebung vornehmen, empfehlen wir Ihnen dringend, die Verbindungen in Ihrer Entwicklungsumgebung sorgfältig zu überprüfen.

Stellen Sie NAT-Gateways in jeder Availability Zone bereit

Wenn Sie Knoten in privaten Subnetzen (IPv4 und IPv6) bereitstellen, sollten Sie erwägen, in jeder Availability Zone (AZ) ein NAT-Gateway einzurichten, um eine zonenunabhängige Architektur zu gewährleisten und die AZ-übergreifenden Ausgaben zu reduzieren. Jedes NAT-Gateway in einer AZ ist redundant implementiert.

Amazon VPC CNI

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

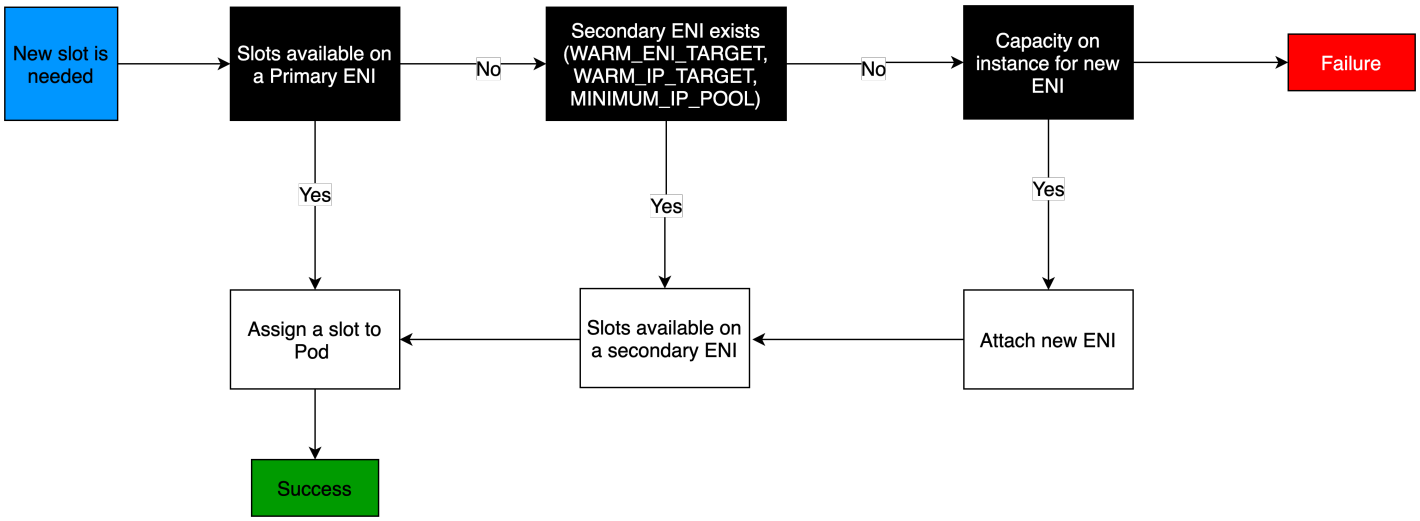
Amazon EKS implementiert Cluster-Netzwerke über das Amazon VPC Container Network Interface-Plugin, auch bekannt als VPC CNI. Das CNI-Plugin ermöglicht es Kubernetes-Pods, dieselbe IP-Adresse wie im VPC-Netzwerk zu haben. Genauer gesagt teilen sich alle Container innerhalb des Pods einen Netzwerk-Namespace und können über lokale Ports miteinander kommunizieren.

Amazon VPC CNI besteht aus zwei Komponenten:

- CNI Binary, das das Pod-Netzwerk einrichtet, um die Kommunikation zu ermöglichen. Pod-to-Pod Die CNI-Binärdatei läuft auf einem Knoten-Root-Dateisystem und wird vom Kubelet aufgerufen, wenn ein neuer Pod hinzugefügt oder ein vorhandener Pod vom Knoten entfernt wird.
- ipamd, ein seit langem laufender Node-Local IP Address Management (IPAM) -Daemon, der für Folgendes verantwortlich ist:
  - Verwaltung auf einem Knoten und ENIs
  - Aufrechterhaltung eines warmen Pools verfügbarer IP-Adressen oder Präfixe

Wenn eine Instanz erstellt wird, erstellt EC2 eine primäre ENI, die einem primären Subnetz zugeordnet ist, und fügt sie an. Das primäre Subnetz kann öffentlich oder privat sein. Die Pods, die im HostNetwork-Modus ausgeführt werden, verwenden die primäre IP-Adresse, die dem primären Knoten ENI zugewiesen ist, und verwenden denselben Netzwerk-Namespace wie der Host.

Das CNI-Plugin verwaltet Elastic Network Interfaces (ENI) auf dem Knoten. Wenn ein Knoten bereitgestellt wird, weist das CNI-Plugin der primären ENI automatisch einen Pool von Steckplätzen (IPs oder Präfixen) aus dem Subnetz des Knotens zu. Dieser Pool wird als Warmpool bezeichnet, und seine Größe wird durch den Instanztyp des Knotens bestimmt. Je nach den CNI-Einstellungen kann es sich bei einem Steckplatz um eine IP-Adresse oder ein Präfix handeln. Wenn ein Steckplatz auf einem ENI zugewiesen wurde, kann das CNI weitere Steckplätze ENIs mit einem warmen Pool an Steckplätzen an die Knoten anschließen. Diese zusätzlichen ENIs werden als sekundär ENIs bezeichnet. Jede ENI kann je nach Instance-Typ nur eine bestimmte Anzahl von Steckplätzen unterstützen. Das CNI fügt je ENIs nach Anzahl der benötigten Steckplätze, was in der Regel der Anzahl der Pods entspricht, mehr Instances hinzu. Dieser Vorgang wird fortgesetzt, bis der Knoten kein zusätzliches ENI mehr unterstützt. Das CNI weist außerdem „warm“ ENIs und Steckplätze für einen schnelleren Pod-Start vorab zu. Beachten Sie, dass für jeden Instanztyp eine maximale Anzahl ENIs angehängt werden kann. Dies ist zusätzlich zu den Rechenressourcen eine Einschränkung der Pod-Dichte (Anzahl der Pods pro Knoten).



Die maximale Anzahl von Netzwerkschnittstellen und die maximale Anzahl von Steckplätzen, die Sie verwenden können, hängen vom Typ der EC2-Instance ab. Da jeder Pod eine IP-Adresse an einem Steckplatz verbraucht, hängt die Anzahl der Pods, die Sie auf einer bestimmten EC2-Instance ausführen können, davon ab, wie viele Pods an ihn angeschlossen werden ENIs können und wie viele Steckplätze jede ENI unterstützt. Wir empfehlen, die maximale Anzahl an Pods pro EKS-Benutzerhandbuch festzulegen, um eine Erschöpfung der CPU- und Speicherressourcen der Instance zu vermeiden. Pods, `hostNetwork` die verwendet werden, sind von dieser Berechnung ausgenommen. Sie könnten erwägen, ein Skript namens max-pod-calculator.sh zu verwenden, um die von EKS empfohlene maximale Anzahl an Pods für einen bestimmten Instance-Typ zu berechnen.

-Übersicht

Der sekundäre IP-Modus ist der Standardmodus für VPC CNI. Dieses Handbuch bietet einen allgemeinen Überblick über das Verhalten von VPC CNI, wenn der sekundäre IP-Modus aktiviert ist. Die Funktionalität von ipamd (Zuweisung von IP-Adressen) kann je nach den Konfigurationseinstellungen für VPC CNI variieren, z. B.the section called “Präfixmodus für Linux”, undthe section called “Sicherheitsgruppen pro Pod”. the section called “Benutzerdefiniertes Netzwerk”

Das Amazon VPC CNI wird als Kubernetes-Daemonset namens aws-node auf Worker-Knoten bereitgestellt. Wenn ein Worker-Knoten bereitgestellt wird, ist ihm eine Standard-ENI, die so genannte primäre ENI, zugeordnet. Das CNI weist einen warmen Pool mit ENIs und sekundären IP-Adressen aus dem Subnetz zu, das mit der primären ENI des Knotens verbunden ist. Standardmäßig versucht ipamd, dem Knoten eine zusätzliche ENI zuzuweisen. Die IPAMD weist zusätzliche ENI zu, wenn ein einzelner Pod geplant und ihm eine sekundäre IP-Adresse von der primären ENI

zugewiesen wird. Diese „warme“ ENI ermöglicht ein schnelleres Pod-Networking. Wenn der Pool an sekundären IP-Adressen aufgebraucht ist, fügt das CNI eine weitere ENI hinzu, um weitere zuzuweisen.

Die Anzahl ENIs und IP-Adressen in einem Pool werden über Umgebungsvariablen namens WARM_ENI_TARGET, WARM_IP_TARGET, MINIMUM_IP_TARGET konfiguriert. Der Daemonset überprüft regelmäßig, ob eine ausreichende Anzahl von angehängt ist. `aws-node` ENIs Eine ausreichende Anzahl von wird ENIs angehängt, wenn alle `MINIMUM_IP_TARGET` BedingungenWARM_ENI_TARGET, oder `WARM_IP_TARGET` und erfüllt sind. Wenn nicht genügend ENIs angehängt sind, sendet das CNI einen API-Aufruf an EC2, um weitere ENI anzuhängen, bis das `MAX_ENI` Limit erreicht ist.

- `WARM_ENI_TARGET`- Ganzzahl, Werte über 0 bedeuten, dass die Anforderung aktiviert ist
  - Die Anzahl der Warmwerte ENIs , die beibehalten werden sollen. Ein ENI ist „warm“, wenn es als sekundäres ENI an einen Knoten angeschlossen ist, aber von keinem Pod verwendet wird. Genauer gesagt wurden keine IP-Adressen des ENI einem Pod zugeordnet.
  - Beispiel: Stellen Sie sich eine Instanz mit 2 vor ENIs, wobei jede ENI 5 IP-Adressen unterstützt. WARM_ENI_TARGET ist auf 1 gesetzt. Wenn der Instanz genau 5 IP-Adressen zugeordnet sind, verwaltet das CNI zwei ENIs , die an die Instanz angehängt sind. Die erste ENI wird verwendet, und alle 5 möglichen IP-Adressen dieser ENI werden verwendet. Die zweite ENI ist „warm“, da sich alle 5 IP-Adressen im Pool befinden. Wenn ein weiterer Pod auf der Instance gestartet wird, wird eine sechste IP-Adresse benötigt. Das CNI weist diesem sechsten Pod eine IP-Adresse von der zweiten ENI und von 5 IPs aus dem Pool zu. Die zweite ENI wird jetzt verwendet und befindet sich nicht mehr im Status „warm“. Das CNI wird ein drittes ENI zuweisen, um mindestens ein warmes ENI aufrechtzuerhalten.

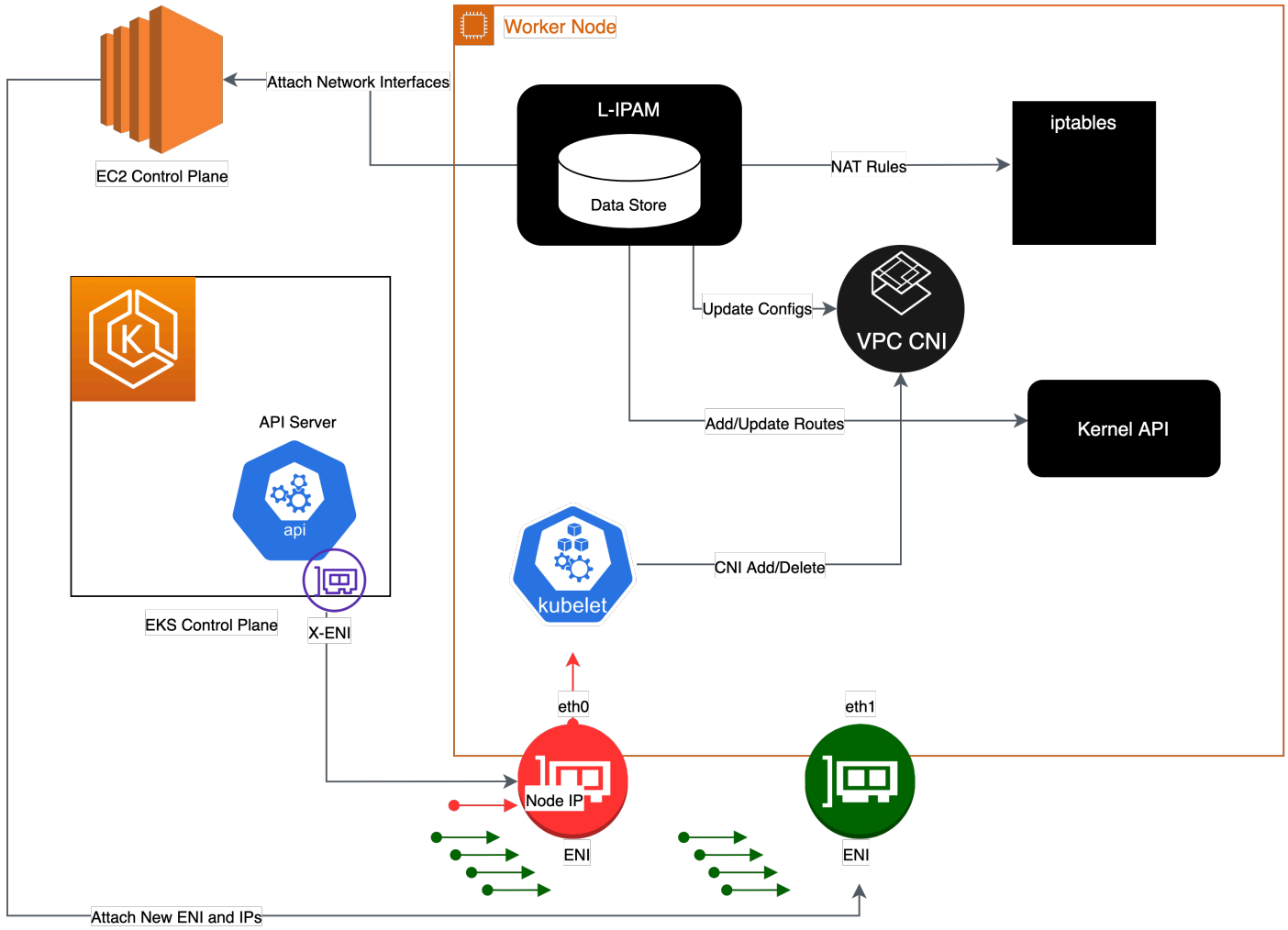
> **Note**
>
> Die warmen verwenden ENIs immer noch IP-Adressen aus dem CIDR Ihrer VPC. IP-Adressen sind „unbenutzt“ oder „warm“, bis sie einem Workload zugeordnet werden, z. B. einem Pod.

- `WARM_IP_TARGET`, Ganzzahl, Werte über 0 geben an, dass die Anforderung aktiviert ist
  - Die Anzahl der Warm-IP-Adressen, die verwaltet werden sollen. Eine Warm-IP ist auf einer aktiv angeschlossenen ENI verfügbar, wurde aber keinem Pod zugewiesen. Mit anderen Worten,

die Anzahl der IPs verfügbaren Warm ist die Anzahl der verfügbaren Warms, die einem Pod zugewiesen werden können, ohne IPs dass eine zusätzliche ENI erforderlich ist.

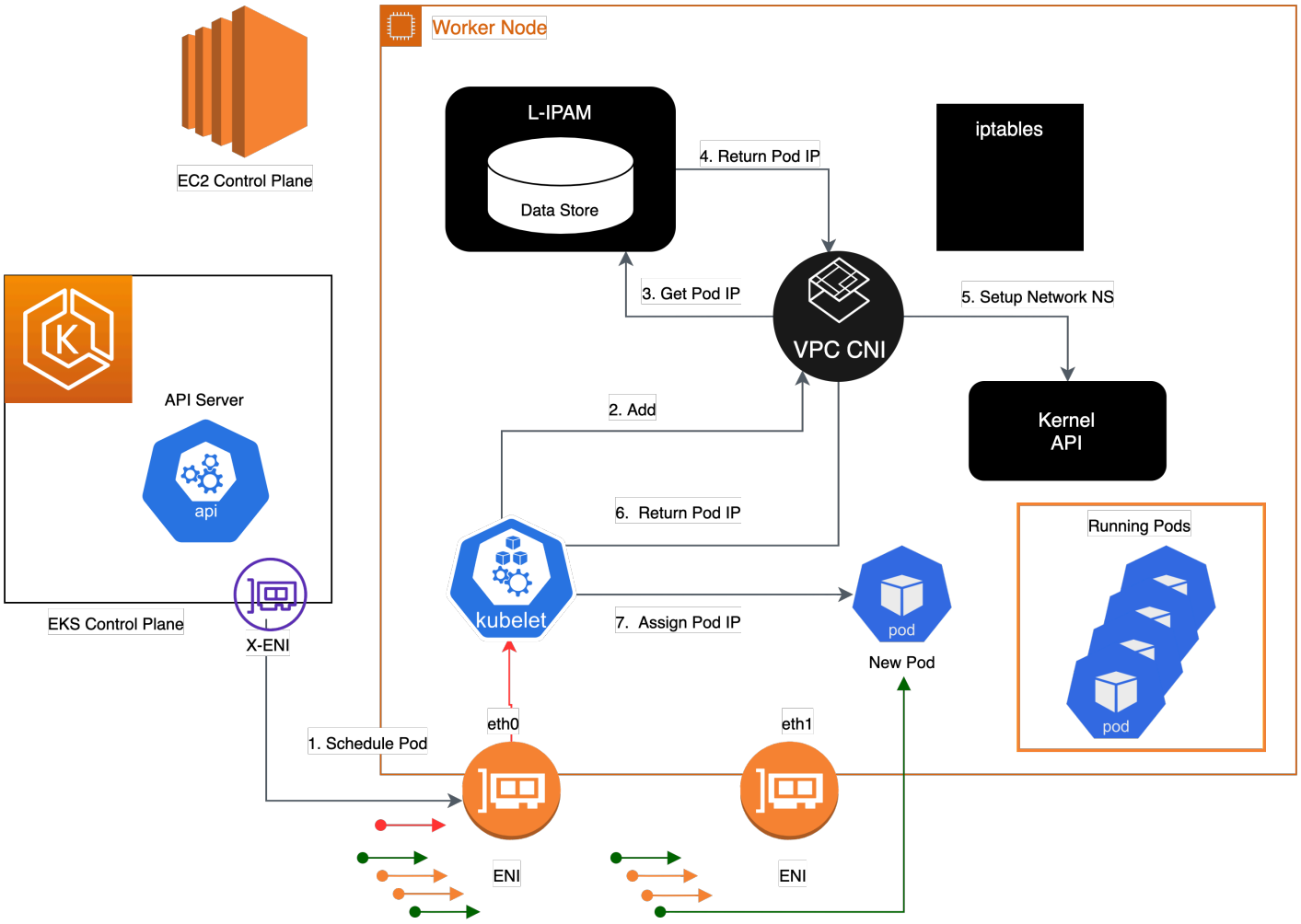
- Beispiel: Stellen Sie sich eine Instance mit 1 ENI vor, wobei jede ENI 20 IP-Adressen unterstützt. WARM_IP_TARGET ist auf 5 gesetzt. WARM_ENI_TARGET ist auf 0 gesetzt. Es wird nur 1 ENI angehängt, bis eine 16. IP-Adresse benötigt wird. Dann fügt das CNI ein zweites ENI an und verwendet dabei 20 mögliche Adressen aus dem Subnetz CIDR.
- `MINIMUM_IP_TARGET`, Ganzzahl, Werte über 0 geben an, dass die Anforderung aktiviert ist
  - Die Mindestanzahl von IP-Adressen, die zu einem beliebigen Zeitpunkt zugewiesen werden können. Dies wird häufig verwendet, um die Zuweisung mehrerer Instanzen beim Start von Instances ENIs im Voraus zu laden.
  - Beispiel: Stellen Sie sich eine neu gestartete Instance vor. Sie hat 1 ENI und jede ENI unterstützt 10 IP-Adressen. MINIMUM_IP_TARGET ist auf 100 gesetzt. Die ENI fügt sofort 9 weitere ENIs hinzu, was insgesamt 100 Adressen ergibt. Dies geschieht unabhängig von den Werten WARM_IP_TARGET oder WARM_ENI_TARGET.

Dieses Projekt beinhaltet ein Excel-Dokument für Subnet Calculator. Dieses Rechendokument simuliert den IP-Adressverbrauch eines bestimmten Workloads unter verschiedenen ENI-Konfigurationsoptionen wie `WARM_IP_TARGET` und. `WARM_ENI_TARGET`

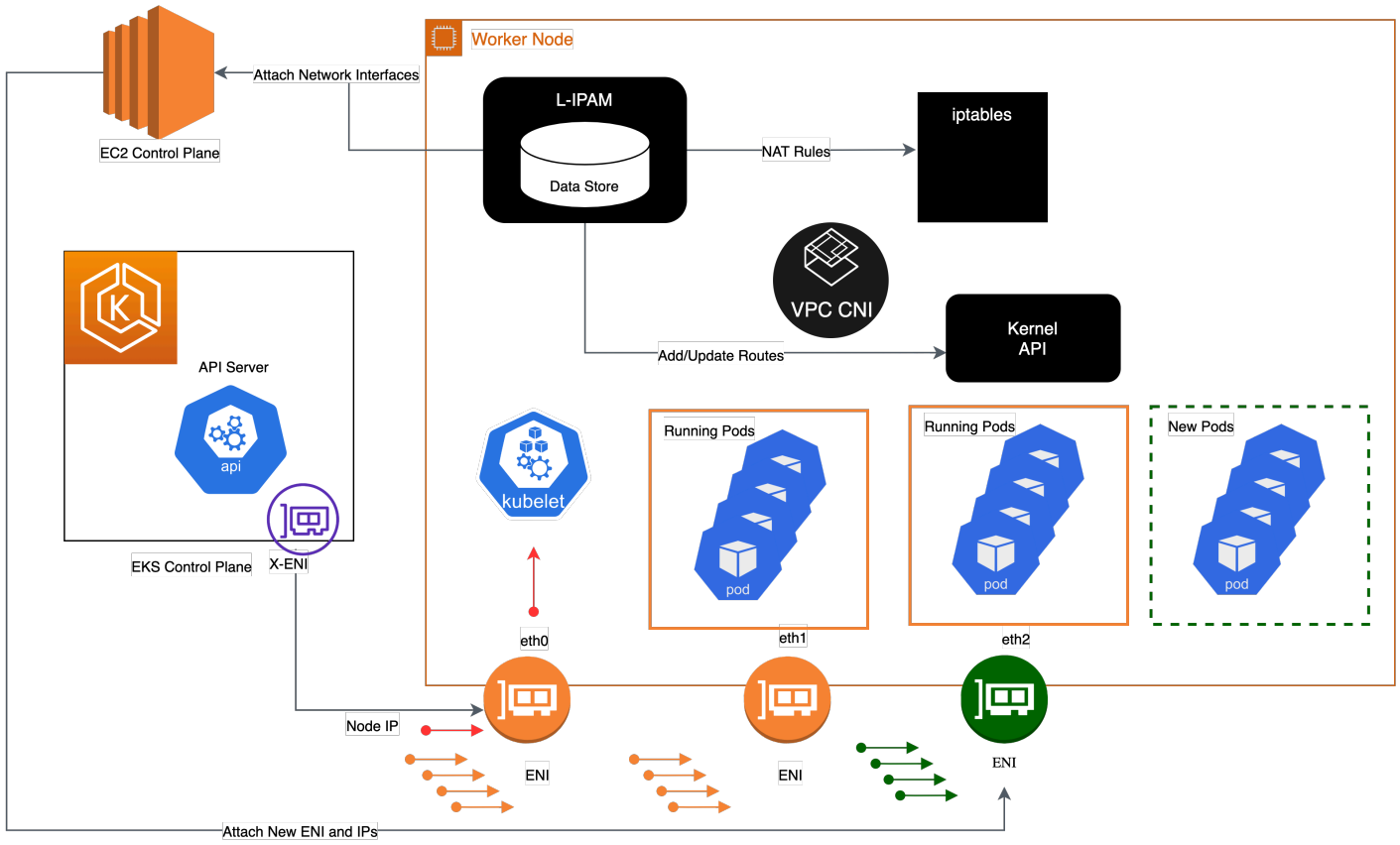


Wenn Kubelet eine Anfrage zum Hinzufügen eines Pods erhält, fragt die CNI-Binärdatei ipamd nach einer verfügbaren IP-Adresse ab, die ipamd dann dem Pod zur Verfügung stellt. Die CNI-Binärdatei verkabelt das Host- und Pod-Netzwerk.

Pods, die auf einem Knoten bereitgestellt werden, werden standardmäßig denselben Sicherheitsgruppen zugewiesen wie die primäre ENI. Alternativ können Pods mit unterschiedlichen Sicherheitsgruppen konfiguriert werden.



Wenn der Pool von IP-Adressen aufgebraucht ist, fügt das Plugin automatisch eine weitere Elastic Network-Schnittstelle an die Instance an und weist dieser Schnittstelle einen weiteren Satz sekundärer IP-Adressen zu. Dieser Vorgang wird fortgesetzt, bis der Knoten keine zusätzlichen Elastic Network-Schnittstellen mehr unterstützen kann.



Wenn ein Pod gelöscht wird, platziert VPC CNI die IP-Adresse des Pods in einem 30-Sekunden-Cooldown-Cache. Die IPs in einem Cooldown-Cache befindlichen Dateien werden keinen neuen Pods zugewiesen. Wenn die Abkühlphase vorbei ist, verschiebt VPC CNI die Pod-IP zurück in den warmen Pool. Die Bedenkzeit verhindert, dass Pod-IP-Adressen vorzeitig recycelt werden, und ermöglicht es dem Kube-Proxy auf allen Clusterknoten, die Aktualisierung der iptables-Regeln abzuschließen. Wenn die Anzahl der Warm-Pool-Einstellungen IPs oder diese ENIs übersteigt, kehrt das ipamd-Plugin zurück IPs und ENIs zur VPC.

Wie oben im Modus Sekundäre IP beschrieben, erhält jeder Pod eine sekundäre private IP-Adresse von einer der ENIs, die mit einer Instance verbunden sind. Da jeder Pod eine IP-Adresse verwendet, hängt die Anzahl der Pods, die Sie auf einer bestimmten EC2-Instance ausführen können, davon ab, wie viele Pods an ihn angeschlossen werden ENIs können und wie viele IP-Adressen er unterstützt. Die VPC-CNI überprüft die Limits-Datei, um herauszufinden, wie viele ENIs IP-Adressen für jeden Instance-Typ zulässig sind.

Sie können die folgende Formel verwenden, um die maximale Anzahl von Pods zu bestimmen, die Sie auf einem Knoten bereitstellen können.

```
(Number of network interfaces for the instance type * (the number of IP addresses per
 network interface - 1)) + 2
```

+2 steht für Pods, für die Host-Netzwerke erforderlich sind, z. B. Kube-Proxy und VPC CNI. Amazon EKS setzt voraus, dass Kube-Proxy und VPC CNI auf jedem Knoten laufen, und diese Anforderungen werden im Max-Pods-Wert berücksichtigt. Wenn Sie zusätzliche Host-Netzwerk-Pods ausführen möchten, sollten Sie den Max-Pods-Wert aktualisieren. Sie können in `--kubelet-extra-args "–max-pods=110"` der Startvorlage Benutzerdaten angeben.

Beispiel: Bei einem Cluster mit 3 c5.large-Knoten (3 ENIs und max. 10 IPs pro ENI) verbraucht das CNI beim Start des Clusters mit 2 CoreDNS-Pods 50 IP-Adressen und behält 43 im Warmpool. IPs Der warme Pool ermöglicht schnellere Pod-Starts, wenn die Anwendung bereitgestellt wird.

Knoten 1 (mit CoreDNS-Pod): 2 ENIs, 20 zugewiesen IPs

Knoten 2 (mit CoreDNS-Pod): 2 ENIs, 20 zugewiesen IPs

Knoten 3 (kein Pod): 1 ENI. 10 IPs zugewiesen.

Für Knoten 1 und Knoten 2 (identische Konfiguration):

- 2 ENIs × 10 IPs pro ENI = 20 IPs insgesamt
- Subtrahieren Sie 2 IPs Primärdaten (1 pro ENI) = 18 IPs
- 1 IP für CoreDNS-Pod abziehen = 17 verfügbar IPs
- Jeder dieser Knoten hat also 17 IPs im warmen Pool

Für Knoten 3:

- 1 ENI × 10 IPs = 10 IPs insgesamt
- Subtrahieren Sie 1 primäre IP = 9, die im warmen Pool IPs verfügbar ist

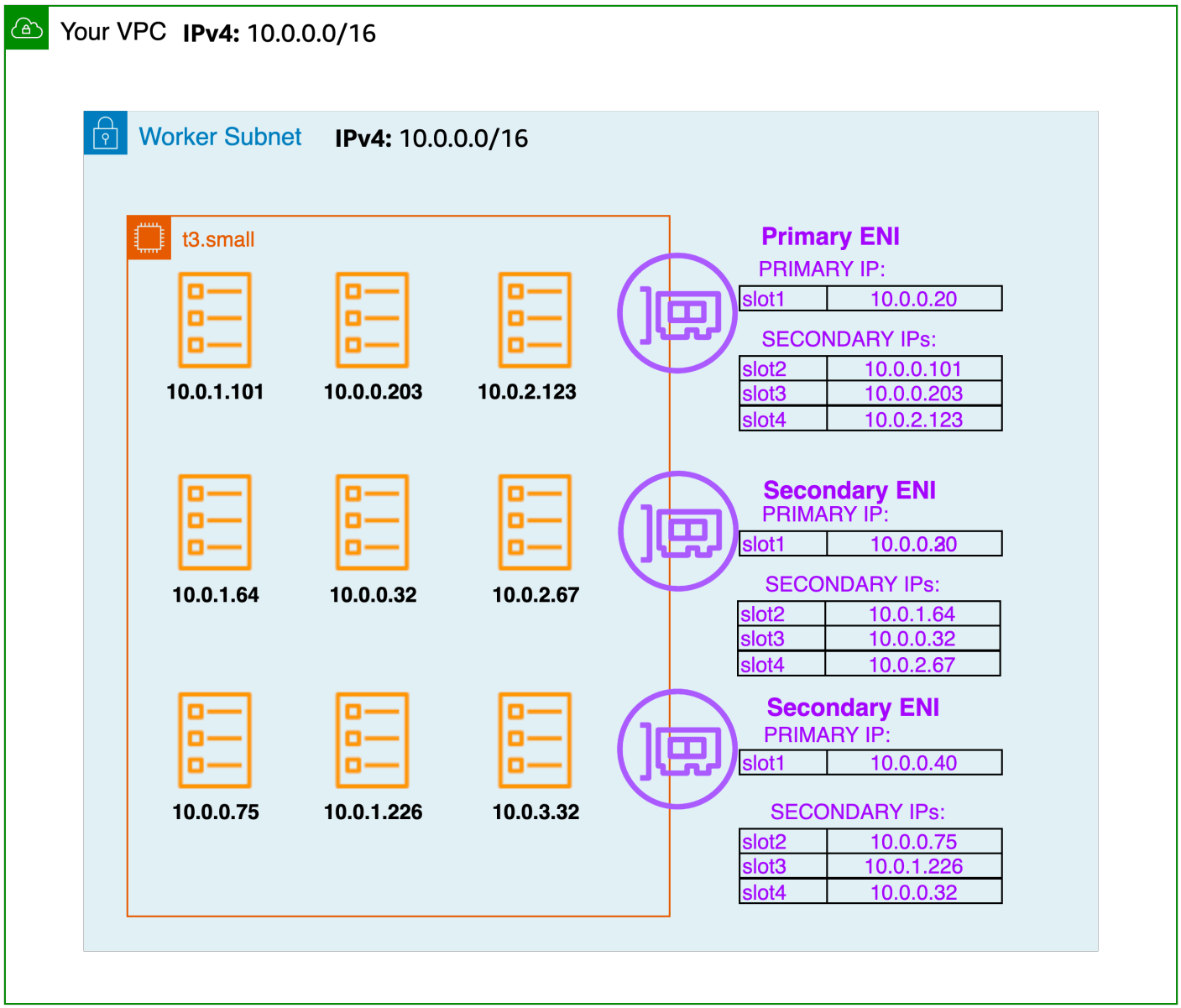
Berechnung des gesamten warmen Pools: - 17 (Knoten 1) + 17 (Knoten 2) + 9 (Knoten 3) = 43 IPs

Denken Sie daran, dass Infrastruktur-Pods, die oft als Daemon-Sets ausgeführt werden, jeweils zur maximalen Pod-Anzahl beitragen. Diese können Folgendes beinhalten:

- CoreDNS
- Amazon Elastic LoadBalancer

- Betriebsbereite Pods für Metrics-Server

Wir empfehlen Ihnen, Ihre Infrastruktur so zu planen, dass Sie die Kapazitäten dieser Pods kombinieren. Eine Liste der maximalen Anzahl von Pods, die von jedem Instance-Typ unterstützt werden, finden Sie unter eni-max-pods.txt on. GitHub



Empfehlungen

Stellen Sie den EKS-Cluster im automatischen Modus bereit

Wenn Sie EKS Auto Mode verwenden, um einen Cluster zu erstellen, verwaltet AWS die VPC Container Network Interface (CNI) -Konfiguration für Ihren Cluster. Mit Amazon EKS Auto Mode ist es nicht erforderlich, Netzwerk-Add-Ons zu installieren oder zu aktualisieren. Stellen Sie jedoch sicher, dass Ihre Workloads mit der verwalteten VPC-CNI-Konfiguration kompatibel sind.

VPC CNI Managed Add-On bereitstellen

Wenn Sie einen Cluster bereitstellen, installiert Amazon EKS VPC CNI automatisch. Amazon EKS unterstützt dennoch verwaltete Add-Ons, die es dem Cluster ermöglichen, mit den zugrunde liegenden AWS-Ressourcen wie Datenverarbeitung, Speicher und Netzwerk zu interagieren. Wir empfehlen dringend, Cluster mit verwalteten Add-Ons wie VPC CNI bereitzustellen.

Das verwaltete Amazon EKS-Add-on bietet die Installation und Verwaltung von VPC-CNI für Amazon EKS-Cluster. Amazon EKS-Add-Ons enthalten die neuesten Sicherheitspatches und Bugfixes und wurden von AWS für die Verwendung mit Amazon EKS validiert. Mit dem VPC CNI-Add-on können Sie die Sicherheit und Stabilität Ihrer Amazon EKS-Cluster kontinuierlich gewährleisten und den Aufwand für die Installation, Konfiguration und Aktualisierung von Add-Ons verringern. Darüber hinaus kann ein verwaltetes Add-on über die Amazon EKS-API, die AWS-Managementkonsole, die AWS-CLI und eksctl hinzugefügt, aktualisiert oder gelöscht werden.

Sie können die verwalteten Felder von VPC CNI mithilfe von `--show-managed-fields` flag mit dem `kubectl get` Befehl finden.

```
kubectl get daemonset aws-node --show-managed-fields -n kube-system -o yaml
```

Verwaltete Add-Ons verhindern Konfigurationsabweichungen, indem sie Konfigurationen automatisch alle 15 Minuten überschreiben. Das bedeutet, dass alle Änderungen an verwalteten Add-Ons, die nach der Erstellung des Add-ons über die Kubernetes-API vorgenommen wurden, durch den automatisierten Drift-Prevention-Prozess überschrieben und auch während des Add-On-Updates auf Standardwerte gesetzt werden.

Die von EKS verwalteten Felder sind unter ManagedFields aufgeführt, wobei der Manager als EKS angegeben ist. Zu den von EKS verwalteten Feldern gehören Dienstkonto, Bild, Bild-URL, Liveness Probe, Readiness Probe, Labels, Volumes und Volume Mounts.

> **Note**
>
> Die am häufigsten verwendeten Felder wie WARM_ENI_TARGET, WARM_IP_TARGET und MINIMUM_IP_TARGET werden nicht verwaltet und nicht abgeglichen. Die Änderungen an diesen Feldern werden bei der Aktualisierung des Add-ons beibehalten.

Wir empfehlen, das Verhalten des Add-ons in Ihren Nicht-Produktionsclustern für eine bestimmte Konfiguration zu testen, bevor Sie Produktionscluster aktualisieren. Folgen Sie außerdem den Schritten im EKS-Benutzerhandbuch für Add-On-Konfigurationen.

Migrieren Sie zum verwalteten Add-On

Sie verwalten die Versionskompatibilität und aktualisieren die Sicherheitspatches der selbstverwalteten VPC CNI. Um ein selbstverwaltetes Add-on zu aktualisieren, müssen Sie Kubernetes APIs und die Anweisungen im EKS-Benutzerhandbuch verwenden. Wir empfehlen die Migration zu einem verwalteten Add-on für bestehende EKS-Cluster und empfehlen dringend, vor der Migration eine Sicherungskopie Ihrer aktuellen CNI-Einstellungen zu erstellen. Um verwaltete Add-Ons zu konfigurieren, können Sie die Amazon EKS-API, die AWS-Managementkonsole oder die AWS-Befehlszeilenschnittstelle verwenden.

```
kubectl apply view-last-applied daemonset aws-node -n kube-system  aws-k8s-cni-old.yaml
```

Amazon EKS ersetzt die CNI-Konfigurationseinstellungen, wenn das Feld als mit Standardeinstellungen verwaltet aufgeführt ist. Wir warnen davor, die verwalteten Felder zu ändern. Das Add-on gleicht Konfigurationsfelder wie die warmen Umgebungsvariablen und die CNI-Modi nicht ab. Die Pods und Anwendungen werden weiterhin ausgeführt, während Sie zu einem verwalteten CNI migrieren.

CNI-Einstellungen vor dem Update Backup

VPC CNI wird auf der Kundendatenebene (Knoten) ausgeführt. Daher aktualisiert Amazon EKS das Add-on (verwaltet und selbstverwaltet) nicht automatisch, wenn neue Versionen veröffentlicht werden oder nachdem Sie Ihren Cluster auf eine neue Kubernetes-Nebenversion aktualisiert haben. Um das Add-on für einen vorhandenen Cluster zu aktualisieren, müssen Sie ein Update über die Update-Addon-API auslösen oder in der EKS-Konsole für Add-Ons auf den Link Jetzt aktualisieren klicken. Wenn Sie ein selbstverwaltetes Add-on bereitgestellt haben, befolgen Sie die unter Aktualisieren des selbstverwalteten VPC CNI-Add-ons genannten Schritte.

Wir empfehlen dringend, jeweils eine Nebenversion zu aktualisieren. Wenn Ihre aktuelle Nebenversion beispielsweise `1.9` ist und Sie auf `1.11` aktualisieren möchten, sollten Sie zuerst auf die neueste Patch-Version von `1.10` aktualisieren und dann auf die neueste Patch-Version von `1.11` aktualisieren.

Führen Sie eine Inspektion des aws-node Daemonset durch, bevor Sie Amazon VPC CNI aktualisieren. Erstellen Sie eine Sicherungskopie der vorhandenen Einstellungen. Wenn Sie ein verwaltetes Add-on verwenden, stellen Sie sicher, dass Sie keine Einstellungen aktualisiert haben, die Amazon EKS möglicherweise überschreibt. Wir empfehlen einen Hook nach dem Update in Ihrem Automatisierungs-Workflow oder einen manuellen Anwendungsschritt nach einem Add-On-Update.

```
kubectl apply view-last-applied daemonset aws-node -n kube-system  aws-k8s-cni-old.yaml
```

Vergleichen Sie bei einem selbstverwalteten Add-on das Backup mit `releases` on, GitHub um die verfügbaren Versionen zu sehen und sich mit den Änderungen in der Version vertraut zu machen, auf die Sie aktualisieren möchten. Wir empfehlen, Helm zu verwenden, um selbstverwaltete Add-Ons zu verwalten und Wertedateien zu nutzen, um Einstellungen anzuwenden. Alle Aktualisierungsvorgänge, bei denen Daemonset Delete beinhaltet, führen zu Ausfallzeiten der Anwendung und müssen vermieden werden.

Verstehen Sie den Sicherheitskontext

Wir empfehlen Ihnen dringend, die Sicherheitskontexte zu verstehen, die für die effiziente Verwaltung von VPC CNI konfiguriert sind. Amazon VPC CNI besteht aus zwei Komponenten: CNI binary und ipamd (aws-node) Daemonset. Das CNI wird als Binärdatei auf einem Knoten ausgeführt und hat Zugriff auf das Root-Dateisystem des Knotens. Außerdem hat es privilegierten Zugriff, da es sich mit iptables auf Knotenebene befasst. Die CNI-Binärdatei wird vom Kubelet aufgerufen, wenn Pods hinzugefügt oder entfernt werden.

Der aws-node Daemonset ist ein lang andauernder Prozess, der für die IP-Adressverwaltung auf Knotenebene verantwortlich ist. Der aws-Node läuft im `hostNetwork` Modus und ermöglicht den Zugriff auf das Loopback-Gerät und die Netzwerkaktivität anderer Pods auf demselben Knoten. Der aws-node-Init-Container läuft im privilegierten Modus und mountet den CRI-Socket, sodass der Daemonset die IP-Nutzung durch die Pods überwachen kann, die auf dem Knoten laufen. Amazon EKS arbeitet daran, die privilegierte Anforderung des AWS-Node-Init-Containers zu entfernen. Darüber hinaus muss der aws-node NAT-Einträge aktualisieren und die iptables-Module laden und wird daher mit NET_ADMIN-Rechten ausgeführt.

Amazon EKS empfiehlt die Bereitstellung der Sicherheitsrichtlinien, wie sie im aws-node-Manifest für die IP-Verwaltung für die Pods und Netzwerkeinstellungen definiert sind. Bitte erwägen Sie, auf die neueste Version von VPC CNI zu aktualisieren. Bitte erwägen Sie außerdem, ein GitHub Problem zu eröffnen, wenn Sie eine bestimmte Sicherheitsanforderung haben.

Verwenden Sie eine separate IAM-Rolle für CNI

Das AWS VPC CNI benötigt AWS Identity and Access Management (IAM) -Berechtigungen. Die CNI-Richtlinie muss eingerichtet werden, bevor die IAM-Rolle verwendet werden kann. Sie können `AmazonEKS_CNI_Policy`eine von AWS verwaltete Richtlinie für IPv4 Cluster verwenden. Die von AmazonEKS verwaltete CNI Richtlinie verfügt nur über Berechtigungen für IPv4 Cluster. Sie müssen eine separate IAM-Richtlinie für IPv6 Cluster mit den hier aufgeführten Berechtigungen erstellen.

Standardmäßig erbt VPC CNI die IAM-Rolle des Amazon EKS-Knotens (sowohl verwaltete als auch selbstverwaltete Knotengruppen).

Es wird dringend empfohlen, eine separate IAM-Rolle mit den entsprechenden Richtlinien für Amazon VPC CNI zu konfigurieren. Wenn nicht, erhalten die Pods von Amazon VPC CNI die der Node-IAM-Rolle zugewiesene Berechtigung und haben Zugriff auf das Instance-Profil, das dem Knoten zugewiesen ist.

Das VPC CNI-Plugin erstellt und konfiguriert ein Dienstkonto namens aws-node. Standardmäßig ist das Dienstkonto an die IAM-Rolle des Amazon EKS-Knotens gebunden, wobei die Amazon EKS-CNI-Richtlinie angehängt ist. Um die separate IAM-Rolle zu verwenden, empfehlen wir Ihnen, ein neues Servicekonto mit angehängter Amazon EKS-CNI-Richtlinie zu erstellen. Um ein neues Servicekonto zu verwenden, müssen Sie die CNI-Pods erneut bereitstellen. Erwägen Sie die Angabe eines `--service-account-role-arn` für VPC CNI verwalteten Add-Ons, wenn Sie neue Cluster erstellen. Stellen Sie sicher, dass Sie die Amazon EKS-CNI-Richtlinie IPv4 sowohl IPv6 für als auch für die Amazon EKS-Knotenrolle entfernen.

Es wird empfohlen, die Metadaten der Access-Instance zu blockieren, um den Explosionsradius von Sicherheitsverletzungen zu minimieren.

Behandeln Sie Liveness/Readiness Testausfälle

Wir empfehlen, die Timeout-Werte für Liveness und Readiness Probe (Standard`timeoutSeconds: 10`) für Cluster mit EKS 1.20 und höher zu erhöhen, um zu verhindern, dass der Pod Ihrer Anwendung aufgrund von Testausfällen im Zustand ContainerCreating hängen bleibt. Dieses Problem wurde bei datenintensiven Clustern und Clustern mit Batchverarbeitung beobachtet. Eine hohe CPU-Auslastung führt zu Integritätsfehlern bei der AWS-Node-Sonde, was zu nicht erfüllten

Pod-CPU-Anfragen führt. Stellen Sie zusätzlich zur Änderung des Test-Timeouts sicher, dass die CPU-Ressourcenanforderungen (Standard`CPU: 25m`) für aws-node korrekt konfiguriert sind. Wir empfehlen nicht, die Einstellungen zu aktualisieren, es sei denn, Ihr Knoten hat Probleme.

Wir empfehlen Ihnen dringend, sudo `bash /opt/cni/bin/aws-cni-support.sh` auf einem Knoten auszuführen, während Sie den Amazon EKS-Support in Anspruch nehmen. Das Skript hilft bei der Auswertung von Kubelet-Protokollen und der Speicherauslastung auf dem Knoten. Bitte erwägen Sie, SSM Agent auf Amazon EKS-Worker-Knoten zu installieren, um das Skript auszuführen.

Konfiguration der IPTables Forward-Richtlinie für AMI-Instances, die nicht für EKS optimiert sind

Wenn Sie ein benutzerdefiniertes AMI verwenden, stellen Sie sicher, dass die IPTABLES-Forward-Richtlinie unter kubelet.service auf ACCEPT gesetzt ist. Viele Systeme setzen die iptables-Forward-Richtlinie auf DROP. Sie können ein benutzerdefiniertes AMI mit HashiCorp Packer und einer Build-Spezifikation mit Ressourcen und Konfigurationsskripts aus dem Amazon EKS AMI-Repository auf AWS GitHub erstellen. Sie können den kubelet.service aktualisieren und den hier angegebenen Anweisungen folgen, um ein benutzerdefiniertes AMI zu erstellen.

Aktualisieren Sie die CNI-Version routinemäßig

Das VPC CNI ist abwärtskompatibel. Die neueste Version funktioniert mit allen von Amazon EKS unterstützten Kubernetes-Versionen. Darüber hinaus wird das VPC CNI als EKS-Add-On angeboten (siehe „Deploy VPC CNI Managed Add-On“ oben). EKS-Add-Ons orchestrieren zwar Upgrades von Add-Ons, aber Add-Ons wie das CNI werden nicht automatisch aktualisiert, da sie auf der Datenebene ausgeführt werden. Sie sind dafür verantwortlich, das VPC CNI-Add-on nach Upgrades von verwalteten und selbstverwalteten Worker-Nodes zu aktualisieren.

Optimierung der Nutzung von IP-Adressen

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Containerisierte Umgebungen nehmen dank der Modernisierung von Anwendungen rasant an Umfang zu. Das bedeutet, dass immer mehr Worker-Knoten und Pods bereitgestellt werden.

Das Amazon VPC CNI-Plugin weist jedem Pod eine IP-Adresse aus den CIDR (s) der VPC zu. Dieser Ansatz bietet mit Tools wie VPC Flow Logs und anderen Überwachungslösungen eine vollständige Sichtbarkeit der Pod-Adressen. Je nach Workload-Typ kann dies dazu führen, dass eine beträchtliche Anzahl von IP-Adressen von den Pods verbraucht wird.

Beim Entwerfen Ihrer AWS-Netzwerkarchitektur ist es wichtig, den IP-Verbrauch von Amazon EKS auf VPC- und Knotenebene zu optimieren. Auf diese Weise können Sie Probleme mit der IP-Erschöpfung verringern und die Pod-Dichte pro Knoten erhöhen.

In diesem Abschnitt werden wir Techniken besprechen, die Ihnen helfen können, diese Ziele zu erreichen.

Optimieren Sie den IP-Verbrauch auf Knotenebene

Die Präfix-Delegierung ist eine Funktion von Amazon Virtual Private Cloud (Amazon VPC), mit der Sie Ihren Amazon Elastic Compute Cloud (Amazon EC2) -Instances IPv4- oder IPv6-Präfixe zuweisen können. Sie erhöht die IP-Adressen pro Netzwerkschnittstelle (ENI), was die Pod-Dichte pro Knoten erhöht und Ihre Recheneffizienz verbessert. Die Präfix-Delegierung wird auch bei Custom Networking unterstützt.

Ausführliche Informationen finden Sie in den Abschnitten Präfix-Delegierung mit Linux-Knoten und Präfix-Delegierung mit Windows-Knoten.

Reduzieren Sie die IP-Erschöpfung

Um zu verhindern, dass Ihre Cluster alle verfügbaren IP-Adressen verbrauchen, empfehlen wir dringend, bei der Dimensionierung Ihrer Cluster VPCs und der Subnetze das Wachstum zu berücksichtigen.

Adoption IPv6ist eine hervorragende Möglichkeit, diese Probleme von Anfang an zu vermeiden. Für Unternehmen, deren Skalierbarkeitsanforderungen über die ursprüngliche Planung hinausgehen und diese nicht übernehmen können IPv6, ist die Verbesserung des VPC-Designs jedoch die empfohlene Antwort auf die Erschöpfung der IP-Adressen. Die von Amazon EKS-Kunden am häufigsten verwendete Technik besteht darin, der VPC nicht routingfähige sekundäre Daten hinzuzufügen und die VPC-CNI so CIDRs zu konfigurieren, dass sie diesen zusätzlichen IP-Bereich bei der Zuweisung von IP-Adressen zu Pods verwendet. Dies wird allgemein als benutzerdefiniertes Netzwerk bezeichnet.

Wir werden uns damit befassen, welche Variablen des Amazon VPC CNI Sie verwenden können, um den Ihren Knoten IPs zugewiesenen Warmpool zu optimieren. Wir werden diesen Abschnitt mit

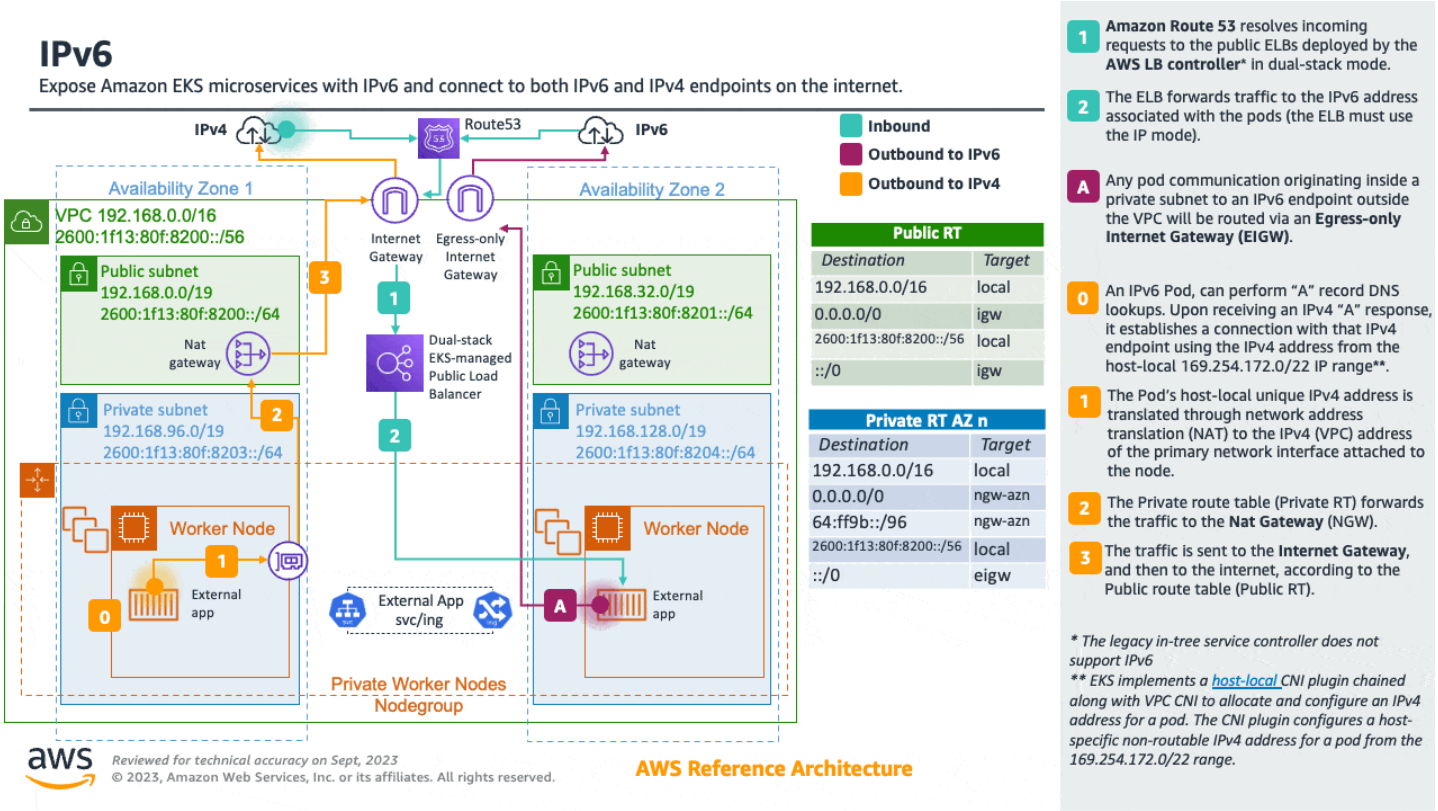
einigen anderen Architekturmustern schließen, die zwar nicht Teil von Amazon EKS sind, aber dazu beitragen können, die IP-Erschöpfung zu verringern.

Verwendung (empfohlen) IPv6

Die Einführung IPv6 ist die einfachste Methode, um die RFC1918 Einschränkungen zu umgehen. Wir empfehlen Ihnen dringend, die Einführung IPv6 als erste Option bei der Auswahl einer Netzwerkarchitektur in Betracht zu ziehen. IPv6 bietet einen deutlich größeren gesamten IP-Adressraum, und Clusteradministratoren können sich auf die Migration und Skalierung von Anwendungen konzentrieren, ohne sich mit der IPv4 Umgehung von Beschränkungen befassen zu müssen.

Amazon EKS-Cluster unterstützen IPv4 sowohl als auch IPv6. Standardmäßig verwenden EKS-Cluster den IPv4 Adressraum. Die Angabe eines IPv6 basierten Adressraums bei der Clustererstellung ermöglicht die Verwendung von IPv6. In einem IPv6 EKS-Cluster erhalten Pods und Dienste IPv6 Adressen, während ältere IPv4 Endgeräte weiterhin eine Verbindung zu Diensten herstellen können, die auf IPv6 Clustern ausgeführt werden, und umgekehrt. Die gesamte pod-to-pod Kommunikation innerhalb eines Clusters erfolgt immer über IPv6. Innerhalb einer VPC (/56) ist die IPv6 CIDR-Blockgröße für IPv6 Subnetze auf /64 festgelegt. Dies bietet 2^64 (ungefähr 18 Trillionen) Adressen, sodass Sie Ihre Bereitstellungen auf EKS skalieren können. IPv6

Ausführliche Informationen finden Sie im Abschnitt Ausführen von IPv6 EKS-Clustern und praktische Erfahrungen finden Sie im Abschnitt Grundlegendes IPv6 zu Amazon EKS des Workshops Get hands-on with IPv6 .



Optimieren Sie den IP-Verbrauch in IPv4 Clustern

Dieser Abschnitt richtet sich an Kunden, die ältere Anwendungen ausführen und noch nicht bereit and/or sind, zu ihnen zu migrieren IPv6. Wir empfehlen zwar allen Unternehmen, IPv6 so schnell wie möglich auf diese Version umzusteigen, sind uns jedoch bewusst, dass einige Unternehmen möglicherweise noch nach alternativen Ansätzen suchen müssen, um ihre Container-Workloads zu skalieren. IPv4 Aus diesem Grund führen wir Sie auch durch die Architekturmuster zur Optimierung IPv4 (RFC1918) der Adressspeicherbelegung mit Amazon EKS-Clustern.

Planen Sie für Wachstum

Als erste Verteidigungslinie gegen IP-Erschöpfung empfehlen wir dringend, Ihre Größe IPv4 VPCs und die Subnetze unter Berücksichtigung des Wachstums zu dimensionieren, um zu verhindern, dass Ihre Cluster alle verfügbaren IP-Adressen verbrauchen. Sie können keine neuen Pods oder Knoten erstellen, wenn die Subnetze nicht über genügend verfügbare IP-Adressen verfügen.

Vor dem Aufbau von VPC und Subnetzen wird empfohlen, von der erforderlichen Workload-Skala aus rückwärts zu arbeiten. Wenn Cluster beispielsweise mit eksctl (einem einfachen CLI-Tool zum Erstellen und Verwalten von Clustern auf EKS) erstellt werden, werden standardmäßig /19-Subnetze

erstellt. Eine Netzmaske von /19 ist für die meisten Workload-Typen geeignet und ermöglicht die Zuweisung von mehr als 8000 Adressen.

> **Important**
>
> Wenn Sie Größe VPCs und Subnetze festlegen, gibt es möglicherweise eine Reihe von Elementen (außer Pods und Knoten), die IP-Adressen verbrauchen können, z. B. Load Balancer, RDS-Datenbanken und andere In-VPC-Dienste.

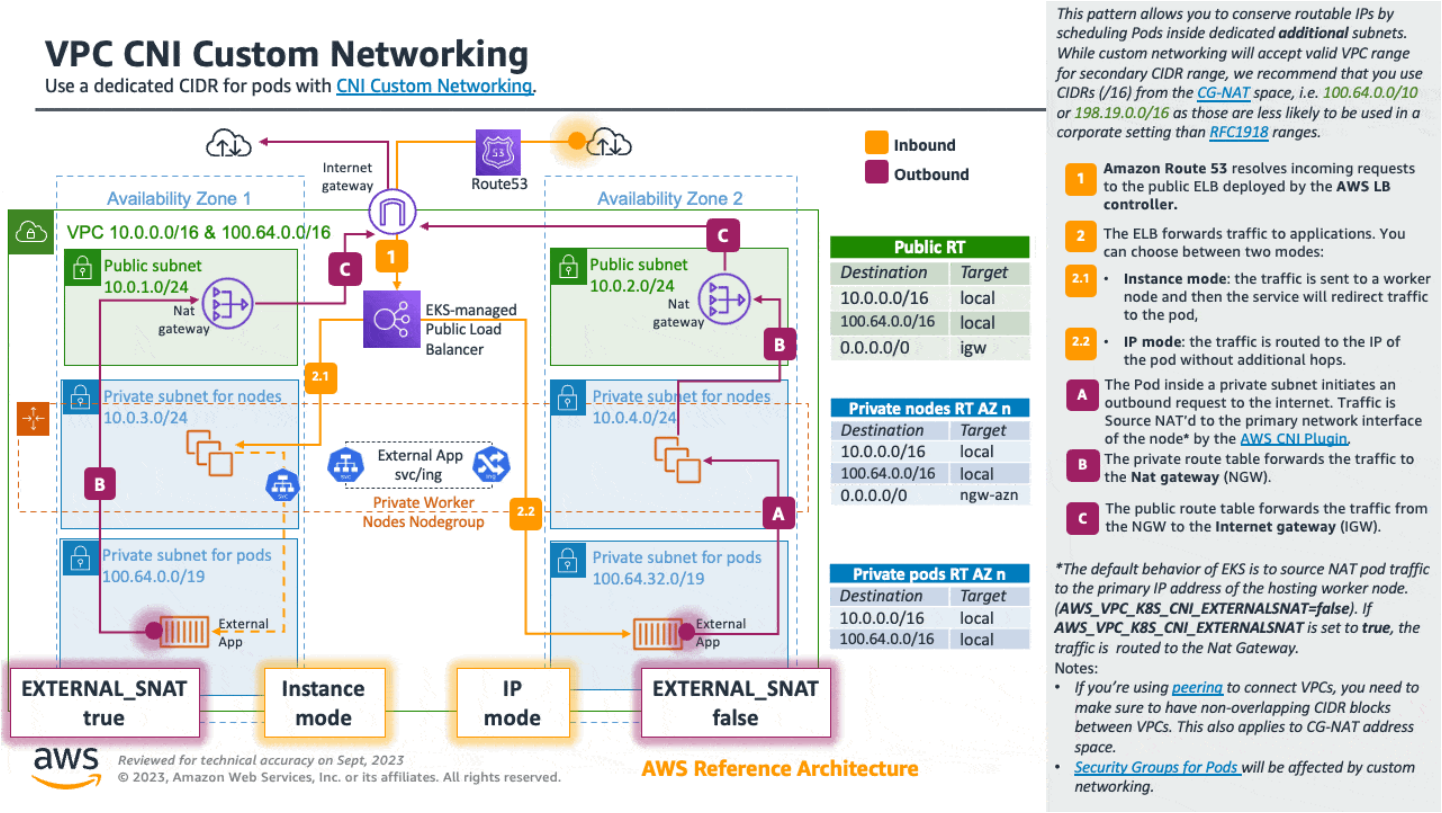
Darüber hinaus kann Amazon EKS bis zu 4 elastische Netzwerkschnittstellen (X-ENI) erstellen, die für die Kommunikation mit der Steuerungsebene erforderlich sind (weitere Informationen hier). Bei Cluster-Upgrades erstellt Amazon EKS neue X- ENIs und löscht die alten, wenn das Upgrade erfolgreich ist. Aus diesem Grund empfehlen wir eine Netzmaske von mindestens /28 (16 IP-Adressen) für Subnetze, die einem EKS-Cluster zugeordnet sind.

Sie können die Beispieltabelle EKS Subnet Calculator verwenden, um Ihr Netzwerk zu planen. Die Tabelle berechnet die IP-Nutzung auf der Grundlage von Workloads und der VPC-ENI-Konfiguration. Die IP-Nutzung wird mit einem IPv4 Subnetz verglichen, um festzustellen, ob die Konfiguration und die Subnetzgröße für Ihre Arbeitslast ausreichend sind. Denken Sie daran, dass wir empfehlen, ein neues Subnetz mit den ursprünglichen CIDR-Blöcken der VPC zu erstellen, wenn Subnetze in Ihrer VPC keine verfügbaren IP-Adressen mehr haben. Beachten Sie, dass Amazon EKS jetzt die Änderung von Cluster-Subnetzen und Sicherheitsgruppen ermöglicht.

Benutzerdefiniertes Netzwerk

Wenn Sie im Begriff sind, den RFC1918 IP-Speicherplatz zu erschöpfen, können Sie das benutzerdefinierte Netzwerkmuster verwenden, um Routing-fähig zu halten, IPs indem Sie Pods in dedizierten zusätzlichen Subnetzen einplanen. Benutzerdefinierte Netzwerke akzeptieren zwar einen gültigen VPC-Bereich für den sekundären CIDR-Bereich, wir empfehlen jedoch, CIDRs (/16) aus dem CG-NAT-Bereich zu verwenden, d. h. `100.64.0.0/10` oder, `198.19.0.0/16` da diese in einer Unternehmensumgebung mit geringerer Wahrscheinlichkeit verwendet werden als Bereiche. RFC1918

Ausführliche Informationen finden Sie im speziellen Abschnitt für benutzerdefinierte Netzwerke.



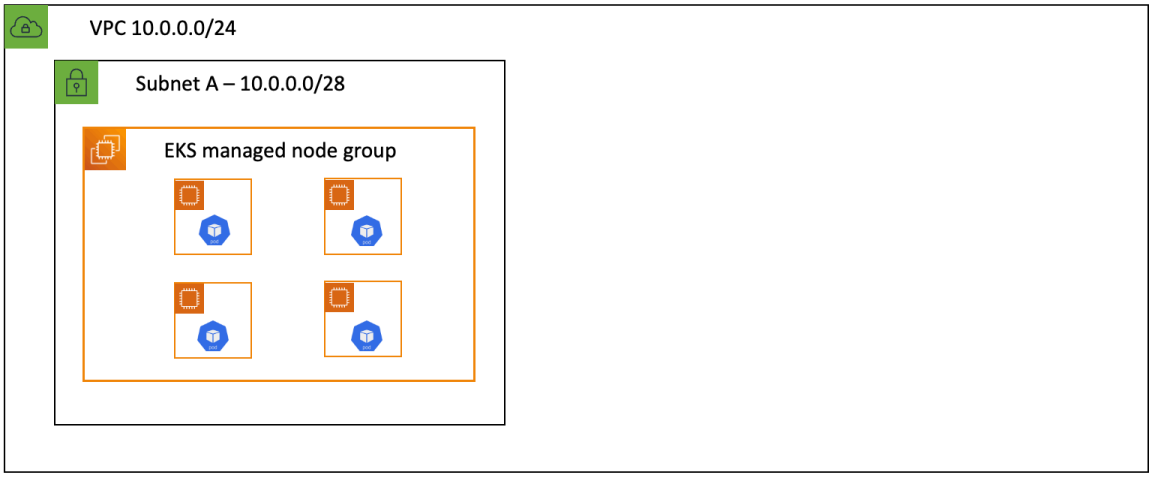
Verbesserte Subnetzerkennung

Enhanced Subnet Discovery bietet eine optimierte Alternative zur Netzwerkkonfiguration bei IP-Erschöpfung, indem neue Subnetze markiert werden, sodass sie vom Amazon VPC CNI erkannt werden können. Mit Enhanced Subnet Discovery können die aktuellen Workloads weiterhin in denselben Subnetzen ausgeführt werden, und Amazon Elastic Kubernetes Service (Amazon EKS) kann jetzt zusätzliche Pods in den neuen „nutzbaren Subnetzen“ planen.

Wenn die aktuellen Subnetze Ihres Clusters keine IP-Adressen mehr haben, können Sie Ihrem Amazon EKS-Cluster einfach wie folgt zusätzliche Subnetze hinzufügen:

1. Ordnen Sie Ihrer VPC einen neuen CIDR-Block zu.
2. Erstellen Sie ein neues Subnetz im neuen CIDR-Block und kennzeichnen Sie es mit „Kubernetes“. io/role/cni“ = „1“.
3. Aktivieren Sie die ENABLE_SUBNET_DISCOVERY-Konfiguration des Amazon VPC CNI-Add-ons auf „true“ (Standard seit Version 1.18.0).

Sobald Enhanced Subnet Discovery auf Ihren VPC- und Amazon EKS-Clustern aktiviert ist, werden neue Elastic Network Interfaces (ENIs) an Ihre Amazon EKS-Knoten angehängt, wie in der folgenden Abbildung beschrieben:



Weitere Informationen finden Sie unter Amazon VPC CNI provides Enhanced Subnet Discovery im AWS-Container-Blog.

Optimieren Sie den warmen Pool IPs

Mit der Standardkonfiguration behält das VPC-CNI eine gesamte ENI (und die zugehörige IPs) im warmen Pool. Dies kann eine große Anzahl von Daten in Anspruch nehmen IPs, insbesondere bei größeren Instance-Typen.

Wenn in Ihrem Cluster-Subnetz nur eine begrenzte Anzahl von IP-Adressen verfügbar ist, überprüfen Sie diese Umgebungsvariablen für die VPC-CNI-Konfiguration:

- `WARM_IP_TARGET`
- `MINIMUM_IP_TARGET`
- `WARM_ENI_TARGET`

Sie können den Wert von so konfigurieren`MINIMUM_IP_TARGET`, dass er der Anzahl der Pods, die Sie voraussichtlich auf Ihren Knoten ausführen werden, genau entspricht. Dadurch wird sichergestellt, dass das CNI bei der Erstellung von Pods IP-Adressen aus dem warmen Pool zuweisen kann, ohne die EC2-API aufrufen zu müssen.

Bitte beachten Sie, dass ein `WARM_IP_TARGET` zu niedriger Wert zu zusätzlichen Aufrufen der EC2-API führen kann, was zu einer Drosselung der Anfragen führen kann. Verwenden Sie bei großen Clustern zusammen mit, um eine Drosselung `MINIMUM_IP_TARGET` der Anfragen zu vermeiden.

Um diese Optionen zu konfigurieren, können Sie das `aws-k8s-cni.yaml` Manifest herunterladen und die Umgebungsvariablen festlegen. Zum Zeitpunkt der Erstellung dieses Artikels befindet sich die neueste Version hier. Überprüfen Sie, ob die Version des Konfigurationswerts mit der installierten VPC-CNI-Version übereinstimmt.

> **Warning**
>
> Diese Einstellungen werden auf die Standardeinstellungen zurückgesetzt, wenn Sie das CNI aktualisieren. Bitte erstellen Sie eine Sicherungskopie des CNI, bevor Sie es aktualisieren. Überprüfen Sie die Konfigurationseinstellungen, um festzustellen, ob Sie sie nach einem erfolgreichen Update erneut anwenden müssen.

Sie können die CNI-Parameter für Ihre vorhandenen Anwendungen im laufenden Betrieb ohne Ausfallzeiten anpassen. Sie sollten jedoch Werte wählen, die Ihren Skalierbarkeitsanforderungen entsprechen. Wenn Sie beispielsweise mit Batch-Workloads arbeiten, empfehlen wir, den Standard so zu aktualisieren, dass er den Anforderungen `WARM_ENI_TARGET` an die Pod-Skalierung entspricht. Durch `WARM_ENI_TARGET` die Einstellung auf einen hohen Wert bleibt immer der warme IP-Pool erhalten, der für die Ausführung großer Batch-Workloads erforderlich ist, und somit werden Verzögerungen bei der Datenverarbeitung vermieden.

> **Warning**
>
> Die Verbesserung Ihres VPC-Designs ist die empfohlene Reaktion auf die Erschöpfung von IP-Adressen. Ziehen Sie Lösungen wie IPv6 und Secondary in Betracht. CIDRs Das Anpassen dieser Werte zur Minimierung der Anzahl von Warmen IPs sollte eine vorübergehende Lösung sein, nachdem andere Optionen ausgeschlossen wurden. Eine Fehlkonfiguration dieser Werte kann den Clusterbetrieb beeinträchtigen. Bevor

Sie Änderungen an einem Produktionssystem vornehmen, sollten Sie unbedingt die Überlegungen auf dieser Seite lesen.

Überwachen Sie den IP-Adressbestand

Zusätzlich zu den oben beschriebenen Lösungen ist es auch wichtig, einen Überblick über die IP-Auslastung zu haben. Sie können den IP-Adressbestand von Subnetzen mit CNI Metrics Helper überwachen. Einige der verfügbaren Metriken sind:

- maximale Anzahl, ENIs die der Cluster unterstützen kann
- Anzahl der ENIs bereits zugewiesenen
- Anzahl der IP-Adressen, die derzeit Pods zugewiesen sind
- Gesamtzahl und maximale Anzahl verfügbarer IP-Adressen

Sie können auch CloudWatch Alarme einrichten, um benachrichtigt zu werden, wenn einem Subnetz die IP-Adressen ausgehen.

Warning

Stellen Sie sicher, dass die `DISABLE_METRICS` Variable für VPC CNI auf false gesetzt ist.

Weitere Überlegungen

Es gibt noch andere Architekturmuster, die Amazon EKS nicht eigen sind und die bei IP-Erschöpfung helfen können. Sie können beispielsweise die Kommunikation zwischen Konten optimieren VPCs oder eine VPC für mehrere Konten gemeinsam nutzen, um die IPv4 Adresszuweisung einzuschränken.

Erfahren Sie hier mehr über diese Muster:

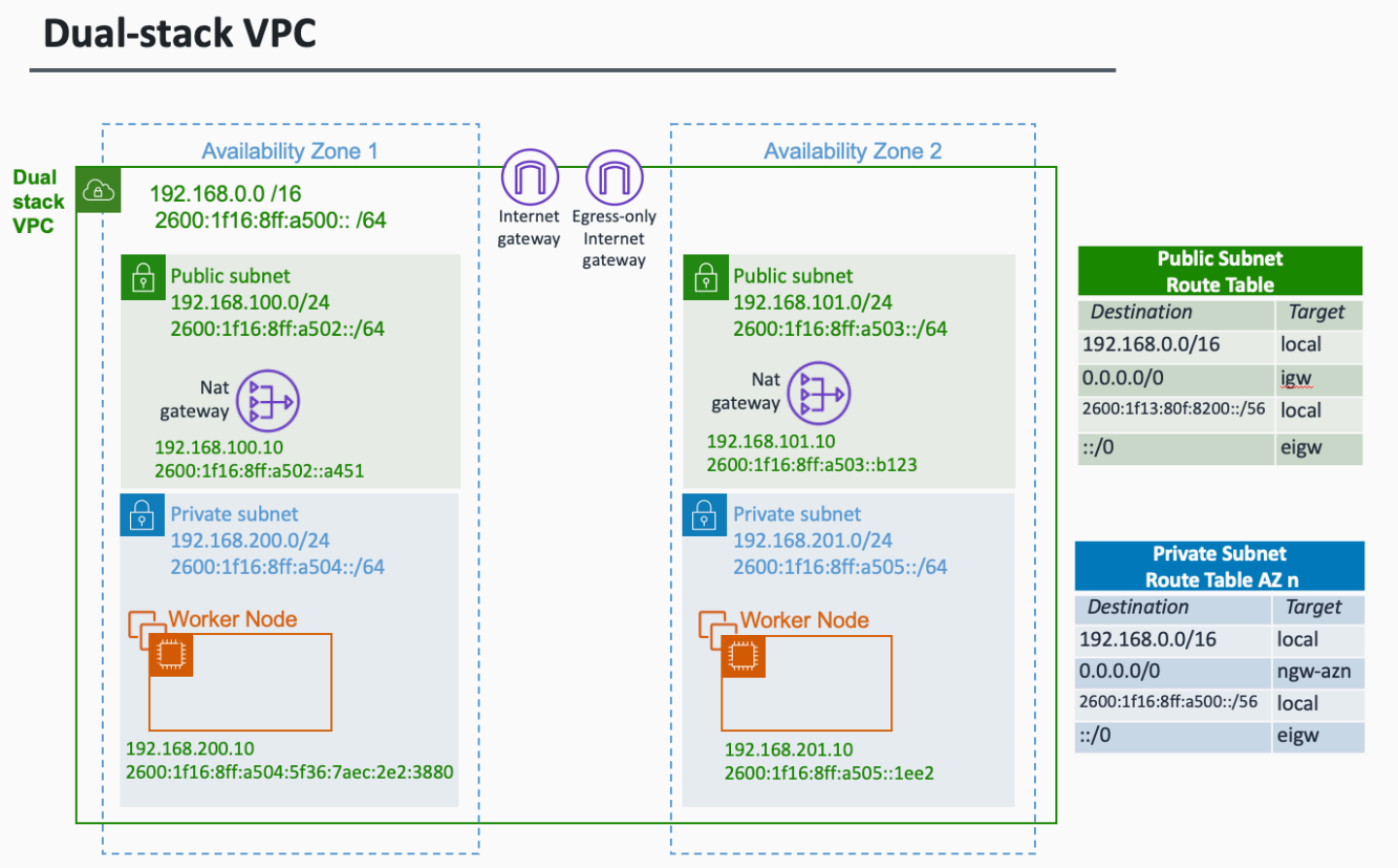
- Entwurf von Hyperscale-Amazon-VPC-Netzwerken,
- Bauen Sie mit Amazon VPC Lattice eine sichere Multi-VPC-Konnektivität für mehrere Konten auf.

IPv6 EKS-Cluster ausführen

EKS im IPv6 Modus löst das Problem der IPv4 Erschöpfung, das häufig bei großen EKS-Clustern auftritt. Die Unterstützung von EKS für konzentriert IPv6 sich auf die Lösung des IPv4 Überlastungsproblems, das auf die begrenzte Größe des Adressraums zurückzuführen ist. IPv4 Dies ist ein erhebliches Problem, das von einer Reihe unserer Kunden geäußert wurde und sich von der IPv4 Kubernetes/Dual-Stack-Funktion unterscheidet. IPv6 EKS/ IPv6 wird auch die Flexibilität bieten, Netzwerkgrenzen miteinander zu verbinden und so die Wahrscheinlichkeit von CIDR-Überschneidungen IPv6 CIDRs zu minimieren, wodurch ein zweifaches Problem gelöst wird (In-Cluster, Cross-Cluster). Bei der Bereitstellung von EKS-Clustern im IPv6 Modus (--ip-family ipv6) kann die Aktion nicht rückgängig gemacht werden. In einfachen Worten, die IPv6 EKS-Unterstützung ist für die gesamte Lebensdauer Ihres Clusters aktiviert.

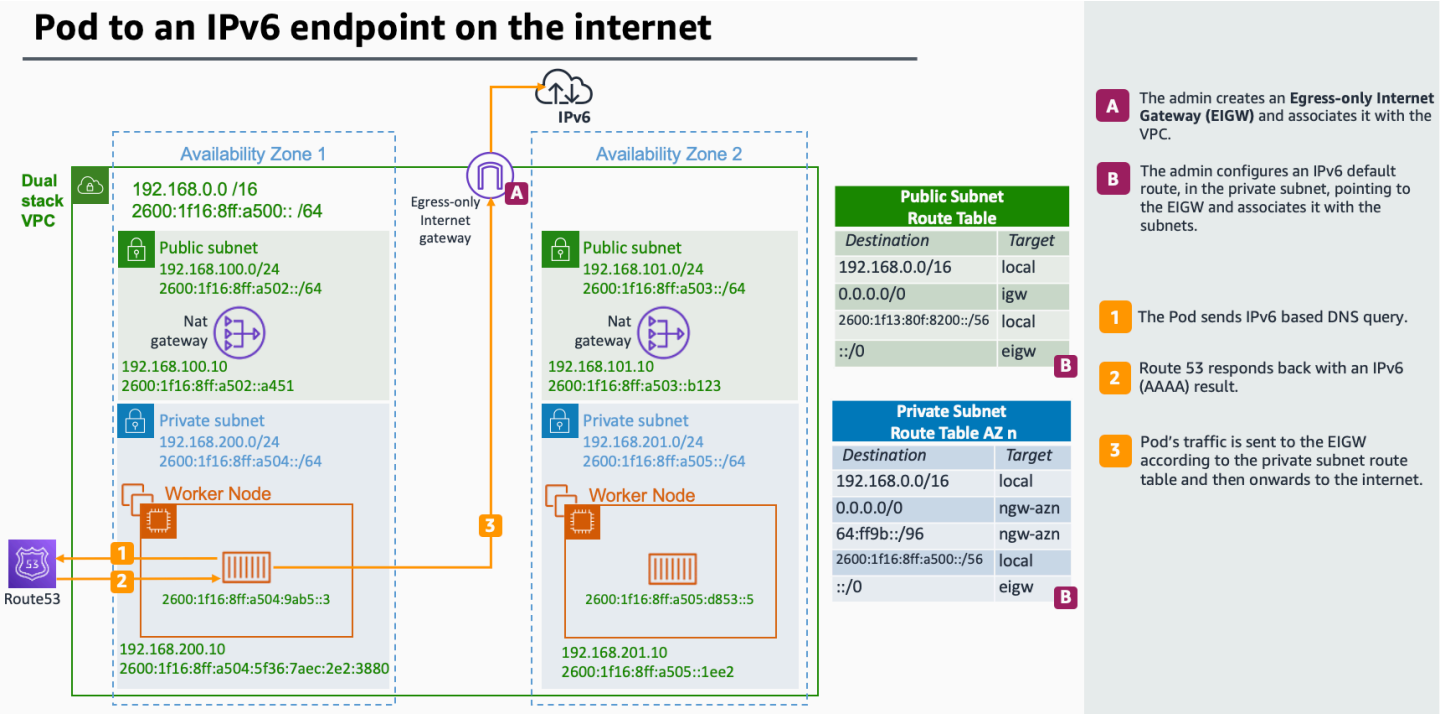
In einem IPv6 EKS-Cluster erhalten Pods und Services IPv6 Adressen und behalten gleichzeitig die Kompatibilität mit älteren IPv4 Endpunkten bei. Dazu gehört die Möglichkeit, dass externe IPv4 Endpunkte auf Dienste innerhalb des Clusters zugreifen können, und Pods auf externe Endpunkte zugreifen können. IPv4

Die Amazon IPv6 EKS-Unterstützung nutzt die nativen VPC-Funktionen IPv6 . Jeder VPC wird mit einem IPv4 Adresspräfix (die CIDR-Blockgröße kann zwischen /16 und /28 liegen) und einem eindeutigen IPv6 /56-Adresspräfix (fest) aus Amazons GUA (Global Unicast Address) zugewiesen. Sie können jedem Subnetz in Ihrer VPC ein /64-Adresspräfix zuweisen. IPv4 Funktionen wie Routentabellen, Network Access Control Lists, Peering und DNS-Auflösung funktionieren in einer IPv6 aktivierten VPC auf die gleiche Weise. Die VPC wird dann als Dual-Stack-VPC bezeichnet. Nach Dual-Stack-Subnetzen zeigt das folgende Diagramm das IPV4 IPv6 VPC-Grundmuster, das EKS/-basierte Cluster unterstützt: IPv6



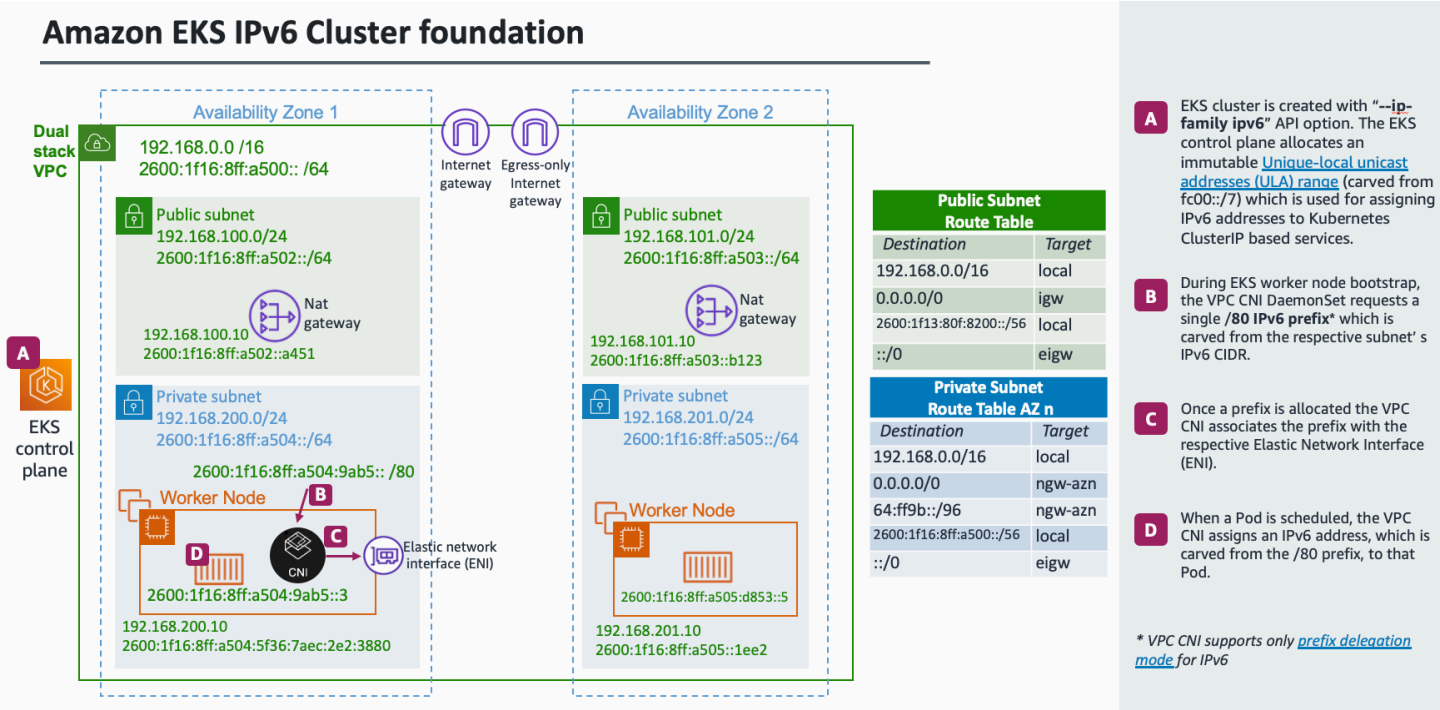
Auf der IPv6 Welt ist jede Adresse über das Internet routingfähig. Standardmäßig weist VPC IPv6 CIDR aus dem öffentlichen GUA-Bereich zu. Seit August 2024 können Sie jedoch auch private IPv6 Adressierung für VPCs und Subnetze mit Amazon VPC IP Address Manager (IPAM) verwenden. Weitere Informationen finden Sie in diesem Blogbeitrag zu AWS Networking und in der VPC-Dokumentation.

Das folgende Diagramm zeigt einen IPv6 Pod-Internetausgangsfluss innerhalb eines EKS/-Clusters: IPv6



Bewährte Methoden für die Implementierung von IPv6 Subnetzen finden Sie im VPC-Benutzerhandbuch.

In einem IPv6 EKS-Cluster erhalten Knoten und Pods öffentliche IPv6 Adressen. EKS weist Diensten IPv6 Adressen zu, die auf Unique Local IPv6 Unicast Addresses (ULA) basieren. Der ULA Service CIDR für einen IPv6 Cluster wird während der Clustererstellungsphase automatisch zugewiesen und kann im Gegensatz dazu nicht angegeben werden. IPv4 Das folgende Diagramm zeigt ein auf EKS/C IPv6 basierendes Grundmuster für einen Datenplan auf der Cluster-Steuerungsebene:



-Übersicht

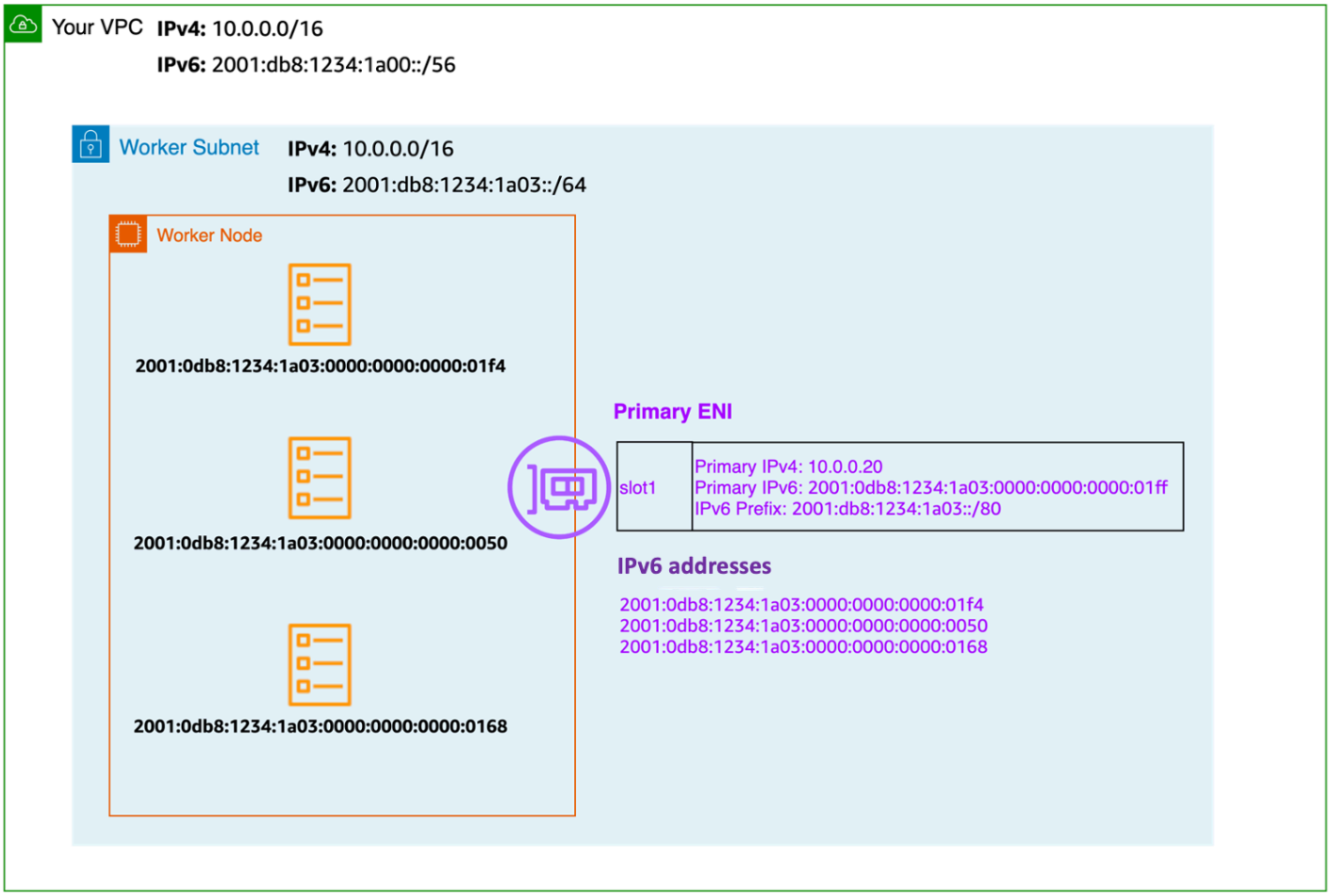
EKS/ IPv6 wird nur im Präfixmodus unterstützt (VPC-CNI Plug-in, ENI IP-Zuweisungsmodus). Erfahren Sie mehr über den Präfix-Modus.

Die Präfixzuweisung funktioniert nur auf Nitro-basierten EC2-Instances. Daher wird EKS/IPv6 nur unterstützt, wenn die Cluster-Datenebene EC2-Nitro-basierte Instances verwendet.

In einfachen Worten, ein IPv6 Präfix von /80 (pro Worker-Node) ergibt ~10^14 IPv6 Adressen. Der limitierende Faktor wird nicht mehr die Pod-Dichte sein (was die Ressourcen angeht). IPs

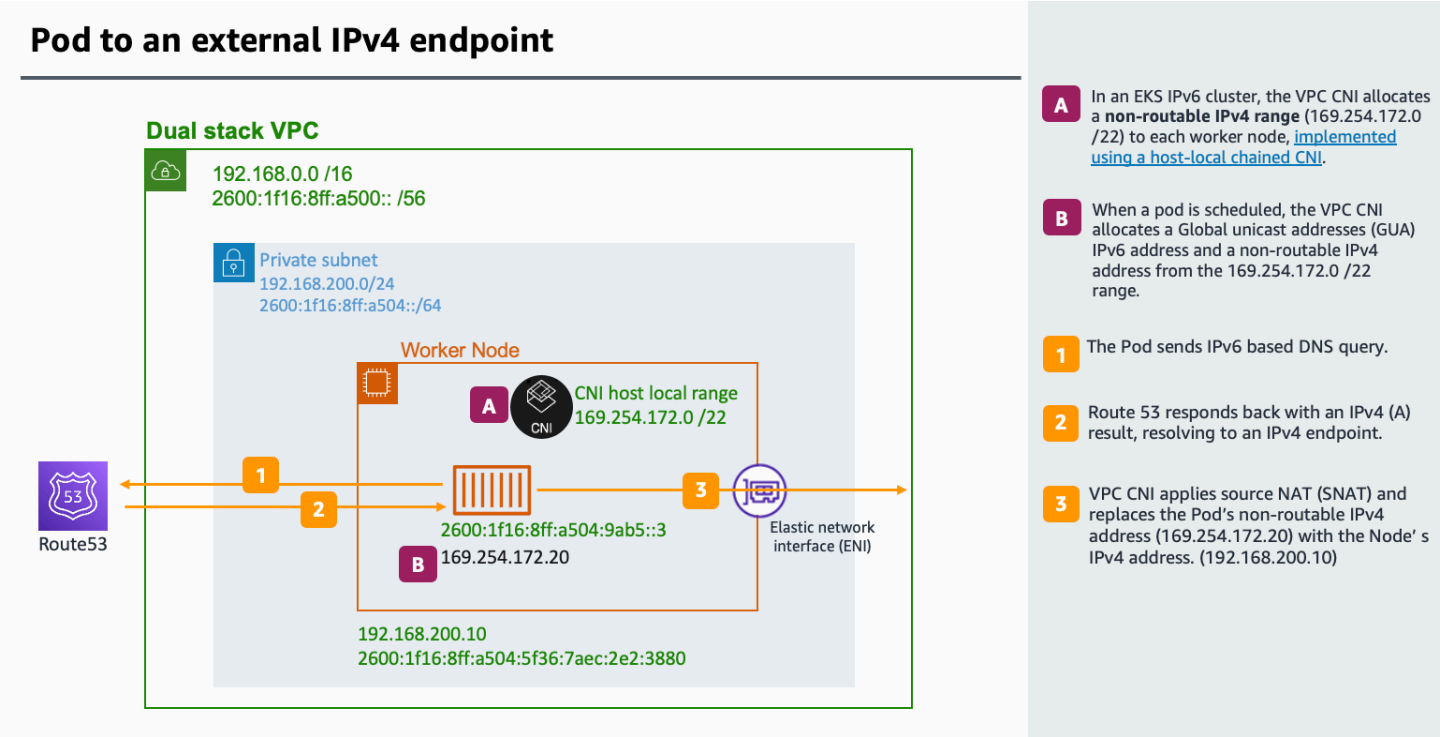
IPv6 Die Präfixzuweisung erfolgt nur beim Bootstrap-Zeitpunkt des EKS-Worker-Nodes. Es ist bekannt, dass dieses Verhalten Szenarien abmildert, in denen IPv4 EKS/Cluster mit hoher Pod-Abwanderung bei der Pod-Planung häufig verzögert werden, da vom VPC CNI-Plug-In (ipamd) gedrosselte API-Aufrufe generiert werden, um private Adressen rechtzeitig zuzuweisen. IPv4
Es ist auch bekannt, dass die fortschrittlichen Regler des VPC-CNI-Plug-ins WARM_IP/ENI und MINIMUM_IP unnötig einstellen.

Das folgende Diagramm vergrößert die Ansicht eines Elastic Network Interface (ENI) für Worker-Nodes: IPv6



Jedem EKS-Worker-Knoten werden IPv6 Adressen IPv4 und entsprechende DNS-Einträge zugewiesen. Für einen bestimmten Worker-Node wird nur eine einzige IPv4 Adresse aus dem Dual-Stack-Subnetz verwendet. Die EKS-Unterstützung für IPv6 ermöglicht Ihnen die Kommunikation mit IPv4 Endgeräten (AWS, vor Ort, Internet) über ein äußerst eigenwilliges Modell, das nur für ausgehenden Datenverkehr konzipiert ist. IPv4 EKS implementiert ein hostlokales CNI-Plugin, das dem VPC-CNI-Plugin zweitrangig ist und eine Adresse für einen Pod zuweist und konfiguriert. IPv4 Das CNI-Plugin konfiguriert eine hostspezifische, nicht routbare Adresse für einen Pod aus dem Bereich 169.254.172.0/22. IPv4 Die dem Pod zugewiesene IPv4 Adresse ist für den Worker-Node eindeutig und wird nicht außerhalb des Worker-Nodes bekannt gegeben. 169.254.172.0/22 bietet bis zu 1024 eindeutige Adressen, die große Instance-Typen unterstützen können. IPv4

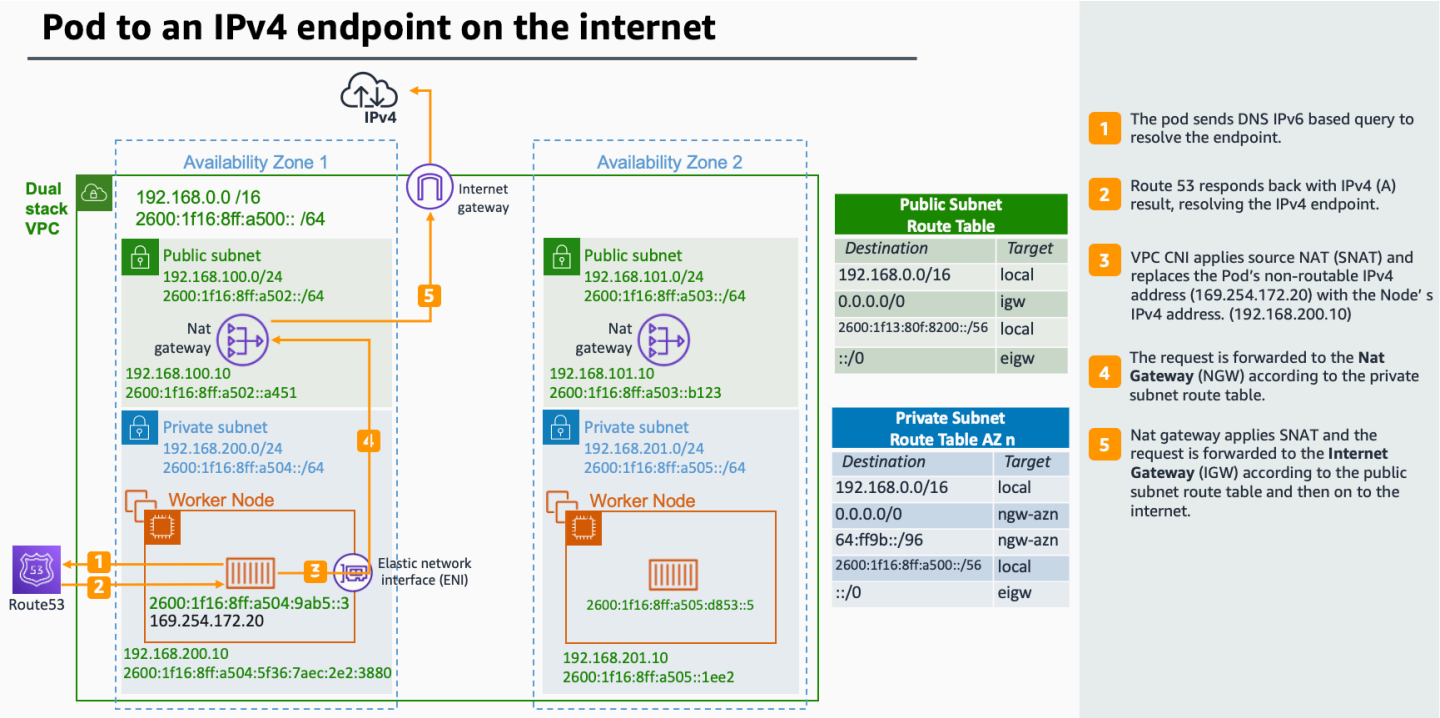
Das folgende Diagramm zeigt den Ablauf einer Verbindung eines Pods mit einem IPv6 Endpunkt außerhalb der Cluster-Grenze (kein Internet): IPv4



Im obigen Diagramm führen Pods eine DNS-Suche nach dem Endpunkt durch. Nach Erhalt einer IPv4-Antwort „A“ wird die eindeutige IPv4-Adresse des Pods, die nur für den Knoten bestimmt ist, durch Source Network Address Translation (SNAT) in die private IPv4-Adresse (VPC) der primären Netzwerkschnittstelle übersetzt, die an den EC2-Worker-Knoten angeschlossen ist.

> **Note**
>
> Das obige Muster muss DNS64 in Subnetzen, in denen EKS/Pods ausgeführt werden, deaktiviert werden. IPv6 Wenn diese Option aktiviert DNS64 ist, gibt der DNS-Resolver eine synthetisierte IPv6 Adresse für IPv4 reine Endpunkte zusammen mit einer Adresse zurück. IPv4 Infolgedessen wird der Datenverkehr über die NAT64 Funktionalität des NAT-Gateways (sofern in der Architektur enthalten) geleitet, anstatt innerhalb der VPC zu bleiben, wie im obigen Muster dargestellt. Dies kann zu einer unerwarteten Nutzung des NAT-Gateways und den damit verbundenen Kosten führen.

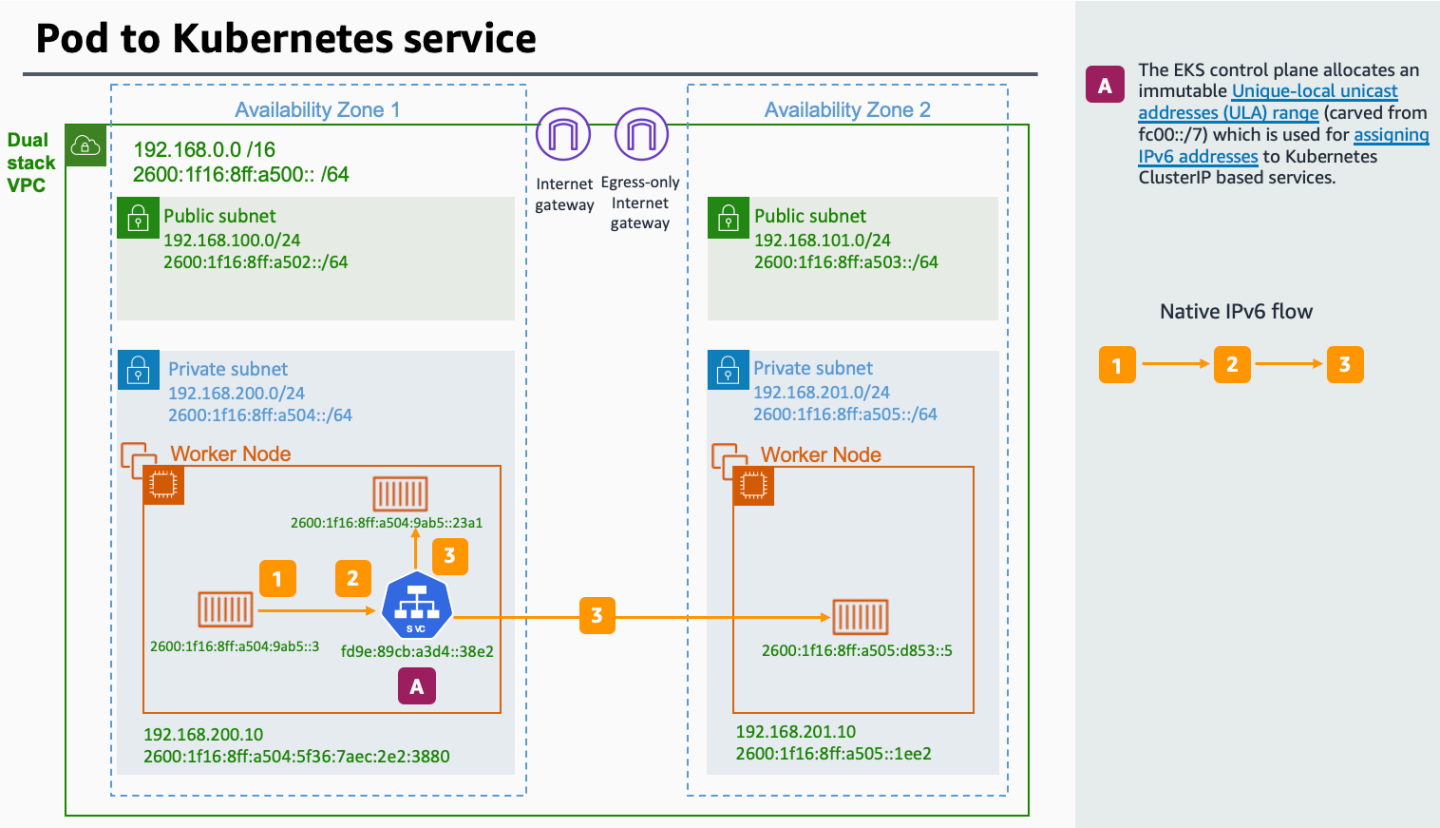
IPv6 EKS/Pods müssen außerdem über öffentliche IPv4 Adressen eine Verbindung zu IPv4 Endpunkten über das Internet herstellen, um einen ähnlichen Datenfluss sicherzustellen. Das folgende Diagramm zeigt den Ablauf eines IPv6 Pods, der sich mit einem IPv4 Endpunkt außerhalb der Clustergrenze verbindet (über das Internet routingfähig):



Im obigen Diagramm führen Pods eine DNS-Suche nach dem Endpunkt durch. Nach Erhalt einer IPv4-Antwort „A“ wird die eindeutige IPv4-Adresse des Pods, die nur für den Knoten bestimmt ist, durch Source Network Address Translation (SNAT) in die private IPv4-Adresse (VPC) der primären Netzwerkschnittstelle übersetzt, die an den EC2-Worker-Knoten angeschlossen ist. Die Pod-IPv4-Adresse (Quell-IPv4: EC2 Primary IP) wird dann an das IPv4-NAT-Gateway weitergeleitet, wo die primäre EC2-IP (SNAT) in eine gültige routbare IPv4 öffentliche Internet-IP-Adresse (NAT Gateway Assigned Public IP) übersetzt wird.

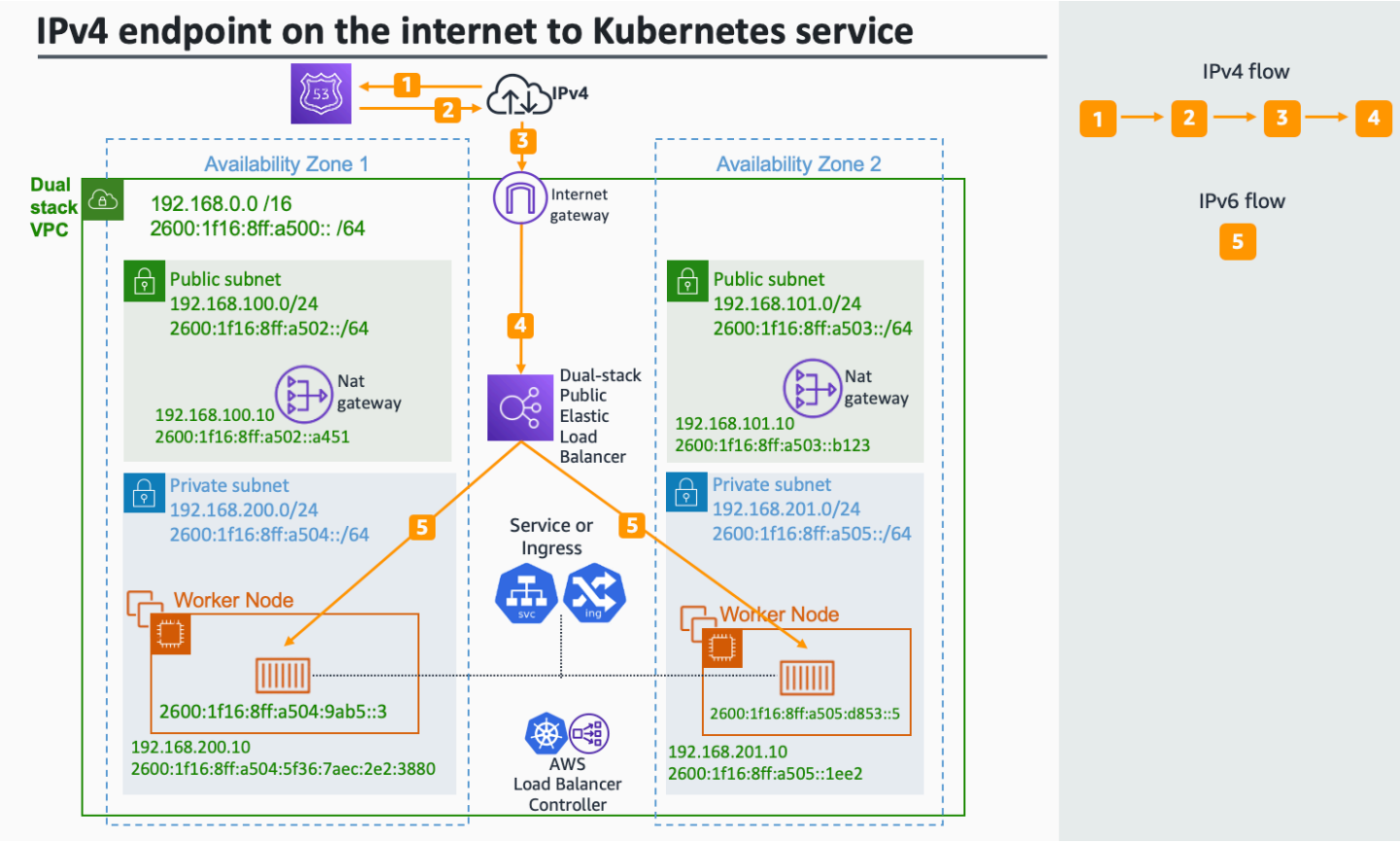
Jegliche Pod-to-Pod Kommunikation zwischen den Knoten verwendet immer eine Adresse. IPv6 VPC CNI konfiguriert iptables so, dass es Verbindungen verarbeitet IPv6 und gleichzeitig blockiert. IPv4

Kubernetes-Dienste erhalten nur IPv6 Adressen (ClusterIP) von Unique Local IPv6 Unicast Addresses (ULA). Der ULA Service CIDR für einen IPv6 Cluster wird bei der Erstellung des EKS-Clusters automatisch zugewiesen und kann nicht geändert werden. Das folgende Diagramm zeigt den Dienstablauf zwischen Pod und Kubernetes:



Dienste werden über einen AWS-Load Balancer dem Internet zugänglich gemacht. Der Load Balancer empfängt öffentliche IPv6 Adressen IPv4 und wird auch als Dual-Stack-Loadbalancer bezeichnet. Bei IPv4 Clients, die auf IPv6 Cluster-Kubernetes-Dienste zugreifen, übernimmt der Load Balancer die Übersetzung. IPv4 IPv6

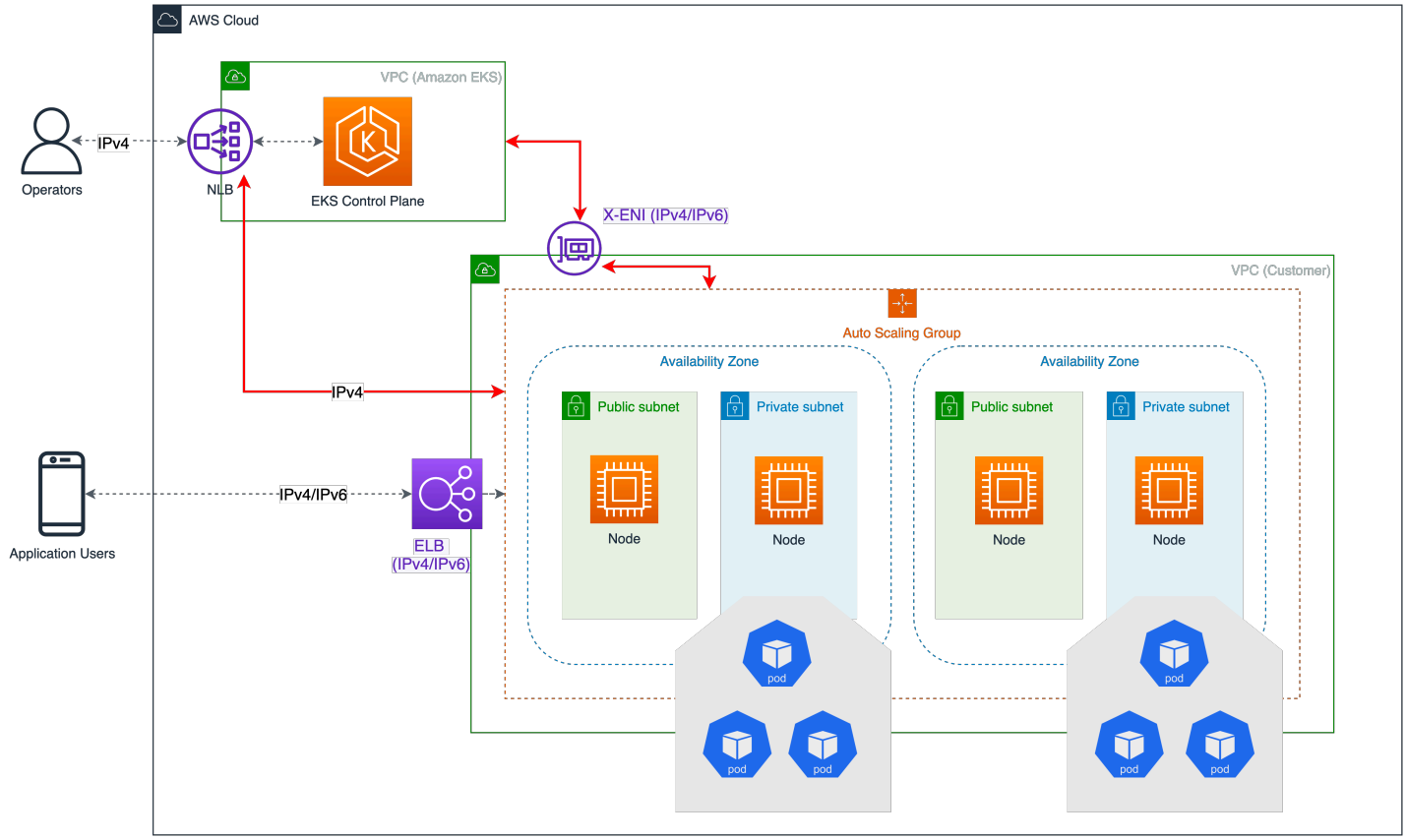
Amazon EKS empfiehlt, Worker-Knoten und Pods in privaten Subnetzen auszuführen. Sie können in den öffentlichen Subnetzen öffentliche Load Balancer einrichten, die den Datenverkehr auf Pods ausgleichen, die auf Knoten in privaten Subnetzen ausgeführt werden. Das folgende Diagramm zeigt einen IPv4 Internetbenutzer, der auf einen IPv6 EKS/Ingress-basierten Dienst zugreift:



> **Note**
>
> Das obige Muster erfordert die Bereitstellung der neuesten Version des AWS Load Balancer-Controllers.

Kommunikation auf der EKS-Kontrollebene auf der Datenebene

EKS stellt Cross-account ENIs (X-ENIs) im Dual-Stack-Modus (IPv4/IPv6) bereit. Kubernetes-Knotenkomponenten wie Kubelet und Kube-Proxy sind so konfiguriert, dass sie Dual-Stack unterstützen. Kubelet und Kube-Proxy werden in einem HostNetwork-Modus ausgeführt und binden sich an beide IPv4 und IPv6 an Adressen, die an die primäre Netzwerkschnittstelle eines Knotens angeschlossen sind. Der Kubernetes-API-Server kommuniziert mit Pods und Knotenkomponenten über die X-basierte Schnittstelle. ENIs IPv6 Pods kommunizieren mit den API-Servern über das X-ENIs, und die Pod-zu-API-Server-Kommunikation verwendet immer den Modus. IPv6



Empfehlungen

Auf Rechenressourcen basierender Zeitplan

Ein einzelnes IPv6 Präfix reicht aus, um viele Pods auf einem einzelnen Knoten auszuführen. Dadurch werden auch die ENI- und IP-Beschränkungen für die maximale Anzahl von Pods auf einem Knoten effektiv aufgehoben. Die direkte IPv6 Abhängigkeit von MAX-Pods wird zwar aufgehoben, aber wenn Sie Präfixanhänge mit kleineren Instance-Typen wie m5.large verwenden, werden Sie wahrscheinlich die CPU- und Speicherressourcen der Instance erschöpfen, lange bevor Sie ihre IP-Adressen erschöpfen. Sie müssen den von EKS empfohlenen maximalen Pod-Wert manuell festlegen, wenn Sie selbstverwaltete Knotengruppen oder eine verwaltete Knotengruppe mit einer benutzerdefinierten AMI-ID verwenden.

Sie können die folgende Formel verwenden, um die maximale Anzahl von Pods zu bestimmen, die Sie auf einem Knoten für einen IPv6 EKS-Cluster bereitstellen können.

```
((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)
```

Verwaltete Knotengruppen berechnen automatisch die maximale Anzahl von Pods für Sie. Vermeiden Sie es, den von EKS empfohlenen Wert für die maximale Anzahl von Pods zu ändern, um Fehler bei der Pod-Planung aufgrund von Ressourcenbeschränkungen zu vermeiden.

Evaluieren Sie den Zweck vorhandener benutzerdefinierter Netzwerke

Wenn benutzerdefinierte Netzwerke derzeit aktiviert sind, empfiehlt Amazon EKS, Ihren Bedarf daran mit IPv6 neu zu bewerten. Wenn Sie sich dafür entschieden haben, benutzerdefinierte Netzwerke zu verwenden, um das Problem der IPv4 Erschöpfung zu lösen, ist dies mit nicht mehr erforderlich. IPv6 Wenn Sie benutzerdefinierte Netzwerke verwenden, um eine Sicherheitsanforderung zu erfüllen, z. B. ein separates Netzwerk für Knoten und Pods, sollten Sie eine EKS-Roadmap-Anfrage einreichen.

Fargate-Pods im EKS/Cluster IPv6

EKS unterstützt IPv6 Pods, die auf Fargate laufen. Pods, die auf Fargate laufen, verbrauchen IPv6 und routbare private IPv4 VPC-Adressen, die aus den VPC-CIDR-Bereichen () stammen. IPv4 IPv6 In einfachen Worten, die Clusterdichte Ihrer EKS/Fargate Pods wird auf die verfügbaren Adressen beschränkt. IPv4 IPv6 Es wird empfohlen, Ihren Dual-Stack subnets/VPC CIDRs für future Wachstum zu dimensionieren. Sie können keine neuen Fargate-Pods planen, wenn das zugrunde liegende Subnetz keine verfügbare IPv4 Adresse enthält, unabhängig von den IPv6 verfügbaren Adressen.

Stellen Sie den AWS Load Balancer Controller (LBC) bereit

Der Upstream-In-Tree-Kubernetes-Servicekontroller unterstützt nicht. IPv6 Wir empfehlen die Verwendung der neuesten Version des AWS Load Balancer Controller-Add-ons. Das LBC stellt nur dann eine Dual-Stack-NLB oder eine Dual-Stack-ALB bereit, wenn sie die entsprechende Kubernetes-Definition verwendet, die wie folgt annotiert ist: und service/ingress `"alb.ingress.kubernetes.io/ip-address-type: dualstack"` `"alb.ingress.kubernetes.io/target-type: ip"`

Benutzerdefiniertes Netzwerk

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Standardmäßig weist Amazon VPC CNI Pods eine IP-Adresse zu, die aus dem primären Subnetz ausgewählt wurde. Das primäre Subnetz ist das Subnetz-CIDR, an das die primäre ENI angeschlossen ist, normalerweise das Subnetz des Knoten/Hosts.

Wenn das Subnetz-CIDR zu klein ist, kann das CNI möglicherweise nicht genügend sekundäre IP-Adressen abrufen, um sie Ihren Pods zuzuweisen. Dies ist eine häufige Herausforderung für EKS-Cluster. IPv4

Benutzerdefinierte Netzwerke sind eine Lösung für dieses Problem.

Benutzerdefiniertes Netzwerk behebt das Problem der IP-Erschöpfung, indem der Knoten und der Pod IPs aus sekundären VPC-Adressräumen (CIDR) zugewiesen werden. Die Unterstützung benutzerdefinierter Netzwerke unterstützt benutzerdefinierte Ressourcen. ENIConfig ENIConfig Dazu gehören ein alternativer Subnetz-CIDR-Bereich (aus einem sekundären VPC-CIDR) sowie die Sicherheitsgruppe (n), zu der die Pods gehören werden. Wenn das benutzerdefinierte Netzwerk aktiviert ist, erstellt das VPC-CNI sekundäre Netzwerke ENIs in dem unter definierten Subnetz. ENIConfig Das CNI weist Pods IP-Adressen aus einem in einer CRD definierten CIDR-Bereich zu. ENIConfig

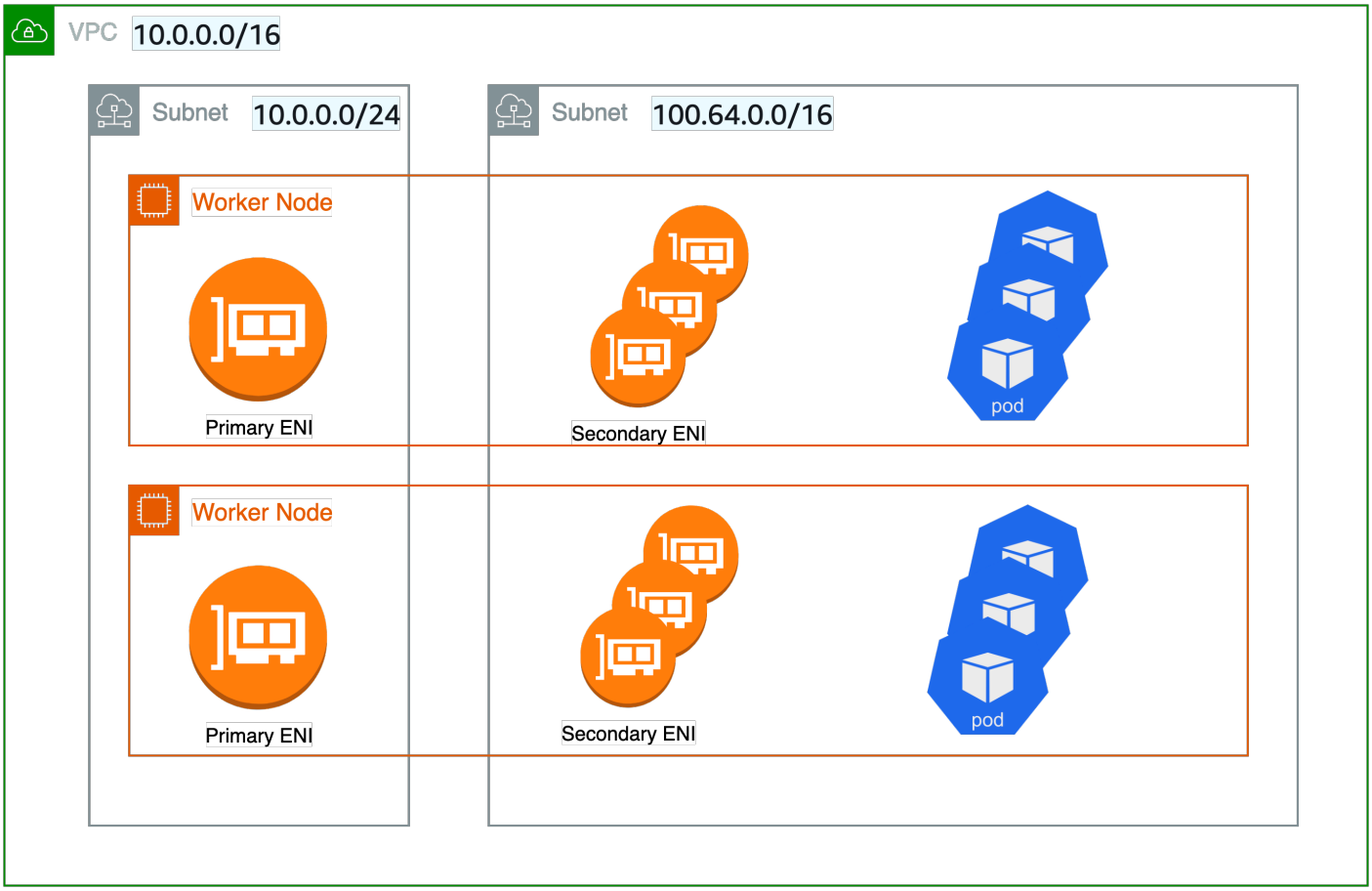
Da die primäre ENI nicht von benutzerdefinierten Netzwerken verwendet wird, ist die maximale Anzahl von Pods, die Sie auf einem Knoten ausführen können, niedriger. Die Host-Netzwerk-Pods verwenden weiterhin die IP-Adresse, die der primären ENI zugewiesen wurde. Darüber hinaus wird das primäre ENI verwendet, um die Übersetzung des Quellnetzwerks zu verarbeiten und den Pods-Verkehr außerhalb des Knotens weiterzuleiten.

Beispielkonfiguration

Benutzerdefinierte Netzwerke akzeptieren zwar einen gültigen VPC-Bereich für den sekundären CIDR-Bereich, wir empfehlen jedoch, CIDRs (/16) aus dem CG-NAT-Bereich zu verwenden, d. h. 100.64.0.0/10 oder 198.19.0.0/16, da diese Bereiche in einer Unternehmensumgebung mit geringerer Wahrscheinlichkeit verwendet werden als andere Bereiche. RFC1918 Weitere Informationen zu den zulässigen und eingeschränkten CIDR-Blockzuordnungen, die Sie mit Ihrer VPC verwenden können, finden Sie unter Einschränkungen der IPv4 CIDR-Blockzuweisung im Abschnitt VPC und Subnetzgröße der VPC-Dokumentation.

Wie in der Abbildung unten dargestellt, verwendet das primäre Elastic Network Interface (ENI) des Worker-Knotens immer noch den primären VPC-CIDR-Bereich (in diesem Fall 10.0.0.0/16), das sekundäre ENIs VPC-CIDR-Bereich (in diesem Fall 100.64.0.0/16). Damit die Pods nun den

CIDR-Bereich 100.64.0.0/16 verwenden können, müssen Sie das CNI-Plugin für die Verwendung benutzerdefinierter Netzwerke konfigurieren. Sie können die hier dokumentierten Schritte ausführen.



Wenn Sie möchten, dass das CNI benutzerdefinierte Netzwerke verwendet, setzen Sie die `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG` Umgebungsvariable auf. `true`

```
kubectl set env daemonset aws-node -n kube-system
 AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

Wann weist `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true` das CNI die Pod-IP-Adresse aus einem Subnetz zu, das in definiert ist. `ENIConfig` Die `ENIConfig` benutzerdefinierte Ressource wird verwendet, um das Subnetz zu definieren, in dem Pods geplant werden.

```
apiVersion : crd.k8s.amazonaws.com/v1alpha1
kind : ENIConfig
metadata:
  name: us-west-2a
spec:
```

```
  securityGroups:
    - sg-0dff111a1d11c1c11
  subnet: subnet-011b111c1f11fdf11
```

Nach der Erstellung der `ENIconfig` benutzerdefinierten Ressourcen müssen Sie neue Worker-Knoten erstellen und die vorhandenen Knoten entleeren. Die vorhandenen Worker-Knoten und Pods bleiben davon unberührt.

Empfehlungen

Verwenden Sie Custom Networking, wenn

Wir empfehlen Ihnen, benutzerdefinierte Netzwerke in Betracht zu ziehen, wenn Sie mit IPv4 Erschöpfung zu kämpfen haben und diese IPv6 noch nicht nutzen können. Die Amazon EKS-Unterstützung für RFC6598Speicherplatz ermöglicht es Ihnen, Pods so zu skalieren, dass sie nicht nur Probleme mit Erschöpfung RFC1918bewältigen. Bitte erwägen Sie, die Präfix-Delegierung mit benutzerdefinierten Netzwerken zu verwenden, um die Pod-Dichte auf einem Knoten zu erhöhen.

Sie könnten benutzerdefinierte Netzwerke in Betracht ziehen, wenn Sie eine Sicherheitsanforderung haben, Pods in einem anderen Netzwerk mit unterschiedlichen Sicherheitsgruppenanforderungen auszuführen. Wenn benutzerdefinierte Netzwerke aktiviert sind, verwenden die Pods andere Subnetze oder Sicherheitsgruppen, wie sie in der ENIConfig primären Netzwerkschnittstelle des Knotens definiert sind.

Benutzerdefinierte Netzwerke sind in der Tat eine ideale Option für die Bereitstellung mehrerer EKS-Cluster und -Anwendungen zur Verbindung von Rechenzentrumsdiensten vor Ort. Sie können die Anzahl der privaten Adressen (RFC1918) erhöhen, auf die EKS in Ihrer VPC für Dienste wie Amazon Elastic Load Balancing und NAT-GW zugreifen kann, und gleichzeitig nicht routbaren CG-NAT-Speicherplatz für Ihre Pods in mehreren Clustern verwenden. Durch benutzerdefinierte Netzwerke mit dem Transit-Gateway und einer Shared Services-VPC (einschließlich NAT-Gateways in mehreren Availability Zones für hohe Verfügbarkeit) können Sie skalierbare und vorhersehbare Datenverkehrsflüsse bereitstellen. In diesem Blogbeitrag wird ein Architekturmuster beschrieben, das eine der am meisten empfohlenen Methoden ist, EKS-Pods mithilfe benutzerdefinierter Netzwerke mit einem Rechenzentrumsnetzwerk zu verbinden.

Vermeiden Sie benutzerdefinierte Netzwerke, wenn

Bereit zur Implementierung IPv6

Durch benutzerdefinierte Netzwerke können Probleme mit der IP-Erschöpfung gemildert werden, sie erfordern jedoch zusätzlichen betrieblichen Aufwand. Wenn Sie derzeit eine Dual-Stack-VPC (IPv4/IPv6) bereitstellen oder Ihr Plan IPv6 Support beinhaltet, empfehlen wir, stattdessen IPv6 Cluster zu implementieren. Sie können IPv6 EKS-Cluster einrichten und Ihre Apps migrieren. In einem IPv6 EKS-Cluster erhalten sowohl Kubernetes als auch Pods eine IPv6 Adresse und können sowohl mit Endpunkten ein IPv4 - IPv6 als auch ausgehend kommunizieren. Bitte lesen Sie sich die bewährten Methoden für das Ausführen IPv6 von EKS-Clustern durch.

Erschöpfter CG-NAT-Speicherplatz

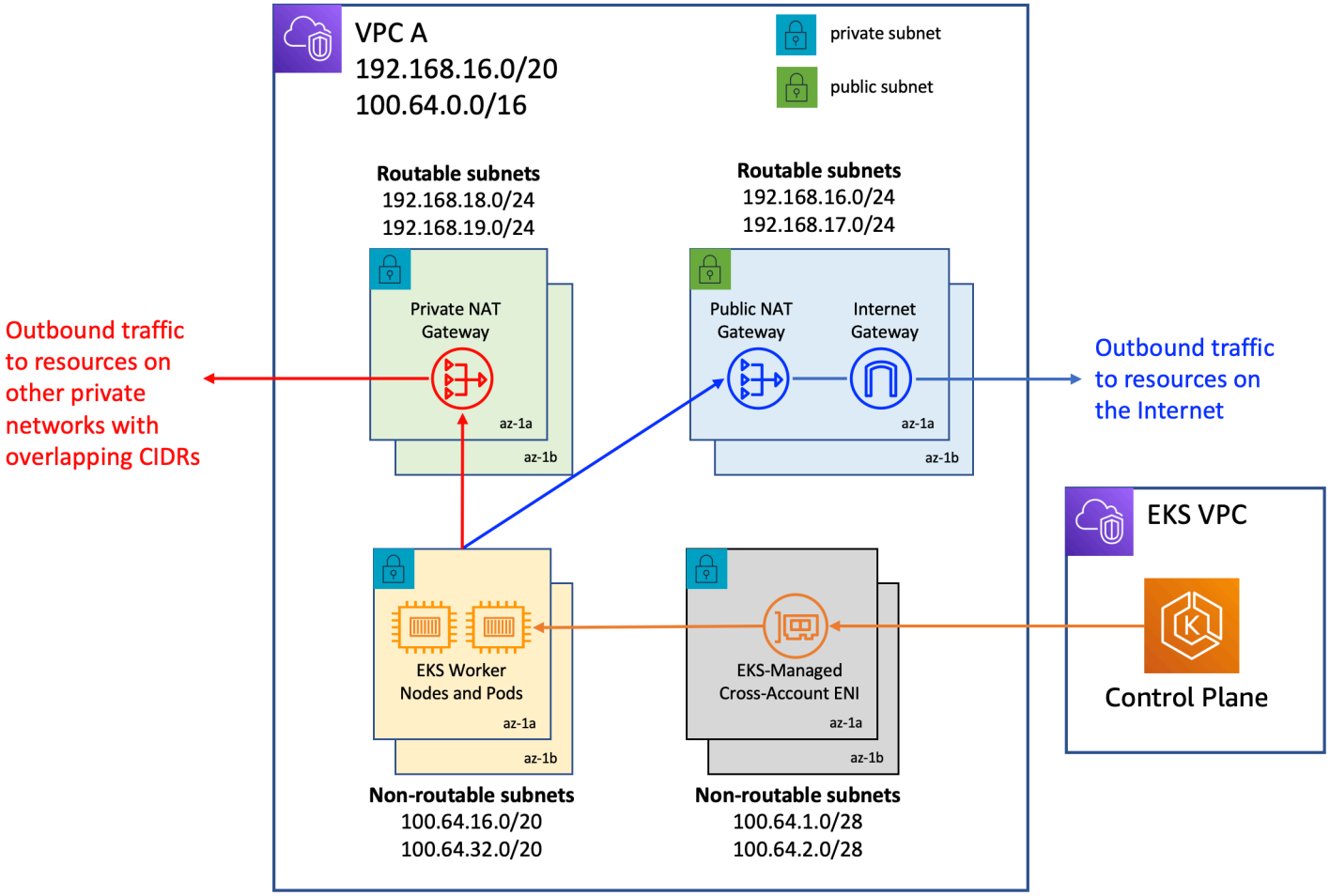
Wenn Sie derzeit den CG-NAT-Bereich nutzen CIDRs oder kein sekundäres CIDR mit Ihrer Cluster-VPC verknüpfen können, müssen Sie möglicherweise andere Optionen prüfen, z. B. die Verwendung eines alternativen CNI. Wir empfehlen Ihnen dringend, entweder kommerziellen Support in Anspruch zu nehmen oder über die erforderlichen internen Kenntnisse zum Debuggen und Einreichen von Patches für das Open-Source-CNI-Plug-in-Projekt zu verfügen. Weitere Informationen finden Sie im Benutzerhandbuch für alternative CNI-Plugins.

Verwenden Sie das private NAT-Gateway

Amazon VPC bietet jetzt private NAT-Gateway-Funktionen. Das private NAT-Gateway von Amazon ermöglicht es Instances in privaten Subnetzen, sich mit anderen VPCs und lokalen Netzwerken zu verbinden, die sich überschneiden. CIDRs Erwägen Sie, die in diesem Blogbeitrag beschriebene Methode zu verwenden, um ein privates NAT-Gateway einzusetzen, um Kommunikationsprobleme bei den EKS-Workloads zu lösen, die durch Überschneidungen verursacht wurden — eine erhebliche Beschwerde CIDRs, die von unseren Kunden geäußert wurde. Benutzerdefinierte Netzwerke können die sich überschneidenden CIDR-Probleme nicht alleine lösen, und das macht die Konfiguration noch schwieriger.

Die in der Implementierung dieses Blogposts verwendete Netzwerkarchitektur folgt den Empfehlungen unter Kommunikation zwischen überlappenden Netzwerken aktivieren in der Amazon VPC-Dokumentation. Wie in diesem Blogbeitrag gezeigt, können Sie die Nutzung von privaten NAT-Gateways in Verbindung mit RFC6598 Adressen ausweiten, um die Erschöpfung der privaten IP-Adressen von Kunden zu bewältigen. Die EKS-Cluster und Worker-Knoten werden im nicht routbaren sekundären VPC-CIDR-Bereich 100.64.0.0/16 bereitgestellt, wohingegen das private NAT-Gateway und das NAT-Gateway in den routbaren CIDR-Bereichen bereitgestellt werden. RFC1918 In diesem

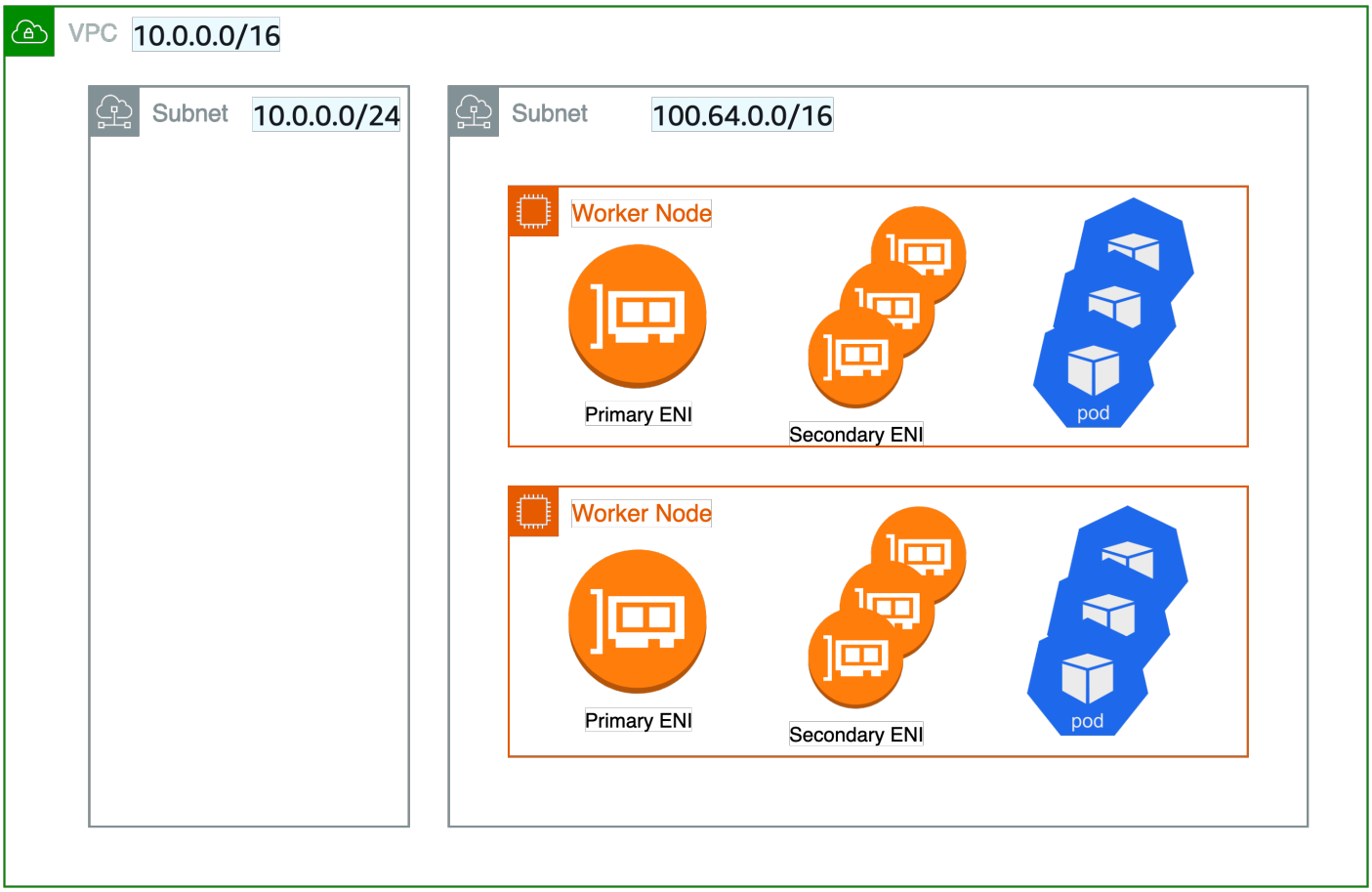
Blog wird erklärt, wie ein Transit-Gateway für Verbindungen verwendet wird, um die Kommunikation zwischen sich überschneidenden, nicht routbaren VPCs CIDR-Bereichen zu erleichtern. VPCs Für Anwendungsfälle, in denen EKS-Ressourcen im nicht routbaren Adressbereich einer VPC mit anderen kommunizieren müssen VPCs , die keine überlappenden Adressbereiche haben, haben Kunden die Möglichkeit, VPC-Peering zu verwenden, um solche Verbindungen herzustellen. VPCs Diese Methode könnte potenzielle Kosteneinsparungen bieten, da der gesamte Datentransfer innerhalb einer Availability Zone über eine VPC-Peering-Verbindung jetzt kostenlos ist.



Einzigartiges Netzwerk für Knoten und Pods

Wenn Sie Ihre Knoten und Pods aus Sicherheitsgründen für ein bestimmtes Netzwerk isolieren müssen, empfehlen wir, Knoten und Pods in einem Subnetz von einem größeren sekundären CIDR-Block aus bereitzustellen (z. B. 100.64.0.0/8). Nach der Installation des neuen CIDR in Ihrer VPC können Sie mithilfe des sekundären CIDR eine weitere Knotengruppe bereitstellen und die ursprünglichen Knoten entleeren, um die Pods automatisch erneut auf den neuen Worker-Knoten bereitzustellen. Weitere Informationen zur Implementierung finden Sie in diesem Blogbeitrag.

Benutzerdefiniertes Netzwerk wird in der Konfiguration, die in der folgenden Abbildung dargestellt ist, nicht verwendet. Stattdessen werden Kubernetes-Worker-Knoten in Subnetzen aus dem sekundären VPC-CIDR-Bereich Ihrer VPC bereitgestellt, z. B. 100.64.0.0/10. Sie können den EKS-Cluster weiterlaufen lassen (die Steuerungsebene bleibt auf dem Original). subnet/s), but the nodes and Pods will be moved to a secondary subnet/s Dies ist eine weitere, wenn auch unkonventionelle Technik, um die Gefahr der IP-Erschöpfung in einer VPC zu verringern. Wir schlagen vor, die alten Knoten zu entleeren, bevor die Pods auf den neuen Worker-Knoten neu bereitgestellt werden.



Automatisieren Sie die Konfiguration mit Availability Zone Labels

Sie können Kubernetes so einrichten, dass die entsprechende Availability Zone (AZ) ENIConfig für den Worker-Knoten automatisch angewendet wird.

Kubernetes fügt das Tag automatisch `topology.kubernetes.io/zone`zu Ihren Worker-Knoten hinzu. Amazon EKS empfiehlt, die Availability Zone als Ihren ENI-Konfigurationsnamen zu verwenden, wenn Sie nur ein sekundäres Subnetz (alternatives CIDR) pro AZ haben. Sie können dann die Bezeichnung, die zur Erkennung des ENI-Konfigurationsnamens verwendet wird, auf festlegen. `topology.kubernetes.io/zone` Beachten Sie, dass das Tag veraltet

`failure-domain.beta.kubernetes.io/zone` ist und durch das Tag ersetzt wurde. `topology.kubernetes.io/zone`

1. Stellen Sie `name` das Feld auf die Availability Zone Ihrer VPC ein.
2. Aktivieren Sie die automatische Konfiguration mit dem folgenden Befehl
3. Stellen Sie das Konfigurationslabel mit dem folgenden Befehl ein

```
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"
```

Wenn Sie mehrere sekundäre Subnetze pro Availability Zone haben, müssen Sie ein bestimmtes `ENI_CONFIG_LABEL_DEF` erstellen. Sie könnten erwägen, Knoten mit benutzerdefinierten EniConfig-Namen zu konfigurieren `ENI_CONFIG_LABEL_DEF` `k8s.amazonaws.com/eniConfig`und sie mit benutzerdefinierten EniConfig-Namen wie und zu kennzeichnen. `k8s.amazonaws.com/eniConfig=us-west-2a-subnet-1``k8s.amazonaws.com/eniConfig=us-west-2a-subnet-2`

Ersetzen Sie Pods bei der Konfiguration des sekundären Netzwerks

Durch die Aktivierung benutzerdefinierter Netzwerke werden vorhandene Knoten nicht geändert. Benutzerdefiniertes Networking ist eine störende Maßnahme. Anstatt nach der Aktivierung des benutzerdefinierten Netzwerks alle Worker-Knoten in Ihrem Cluster fortlaufend zu ersetzen, empfehlen wir, die CloudFormation AWS-Vorlage im EKS-Handbuch „Erste Schritte“ mit einer benutzerdefinierten Ressource zu aktualisieren, die eine Lambda-Funktion aufruft, um das `aws-node` Daemonset mit der Umgebungsvariablen zu aktualisieren, um benutzerdefinierte Netzwerke zu aktivieren, bevor die Worker-Knoten bereitgestellt werden.

Wenn Sie vor der Umstellung auf die benutzerdefinierte CNI-Netzwerkfunktion Knoten in Ihrem Cluster hatten, auf denen Pods ausgeführt wurden, sollten Sie die Knoten sperren und entleeren, um die Pods ordnungsgemäß herunterzufahren und dann die Knoten zu beenden. Nur neue Knoten, die dem ENIConfig Label oder den Anmerkungen entsprechen, verwenden benutzerdefinierte Netzwerke. Daher kann den auf diesen neuen Knoten geplanten Pods eine IP aus dem sekundären CIDR zugewiesen werden.

Berechnen Sie die maximale Anzahl an Pods pro Knoten

Da die primäre ENI des Knotens nicht mehr für die Zuweisung von Pod-IP-Adressen verwendet wird, sinkt die Anzahl der Pods, die Sie auf einem bestimmten EC2-Instance-Typ ausführen können. Um diese Einschränkung zu umgehen, können Sie die Präfixzuweisung mit benutzerdefinierten Netzwerken verwenden. Bei der Präfixzuweisung wird jede sekundäre IP auf der ENIs sekundären durch ein /28-Präfix ersetzt.

Beachten Sie die maximale Anzahl von Pods für eine m5.large-Instance mit benutzerdefiniertem Netzwerk.

Die maximale Anzahl von Pods, die Sie ohne Präfixzuweisung ausführen können, ist 29.

- `3 ENIs - 1) * (10 secondary IPs per ENI - 1 + 2 = 20`

Durch die Aktivierung von Präfixanhängen wird die Anzahl der Pods auf 290 erhöht.

- `(3 ENIs - 1) * ((10 secondary IPs per ENI - 1) * 16 + 2 = 290`

Wir empfehlen jedoch, Max-Pods auf 110 statt 290 zu setzen, da die Instanz eine relativ geringe Anzahl virtueller Pods hat. CPUs Für größere Instances empfiehlt EKS einen maximalen Pod-Wert von 250. Wenn Sie Präfixanhänge mit kleineren Instance-Typen (z. B. m5.large) verwenden, ist es möglich, dass Sie die CPU- und Speicherressourcen der Instance deutlich vor ihren IP-Adressen erschöpfen.

> **Note**
>
> Wenn das CNI-Präfix einer ENI ein /28-Präfix zuweist, muss es sich um einen zusammenhängenden Block von IP-Adressen handeln. Wenn das Subnetz, aus dem das Präfix generiert wird, stark fragmentiert ist, schlägt die Präfixverknüpfung möglicherweise fehl. Sie können dies verhindern, indem Sie eine neue dedizierte VPC für den Cluster erstellen oder indem Sie dem Subnetz einen Satz von CIDR ausschließlich für Präfixanhänge reservieren. Weitere Informationen zu diesem Thema finden Sie unter Subnetz-CIDR-Reservierungen.

Identifizieren Sie die bestehende Nutzung von CG-NAT-Speicherplatz

Mit benutzerdefinierten Netzwerken können Sie das Problem der IP-Erschöpfung verringern, es können jedoch nicht alle Probleme gelöst werden. Wenn Sie bereits CG-NAT-Speicherplatz für Ihren Cluster verwenden oder einfach nicht in der Lage sind, Ihrer Cluster-VPC ein sekundäres CIDR zuzuordnen, empfehlen wir Ihnen, andere Optionen zu prüfen, z. B. die Verwendung eines alternativen CNI oder die Umstellung auf Cluster. IPv6

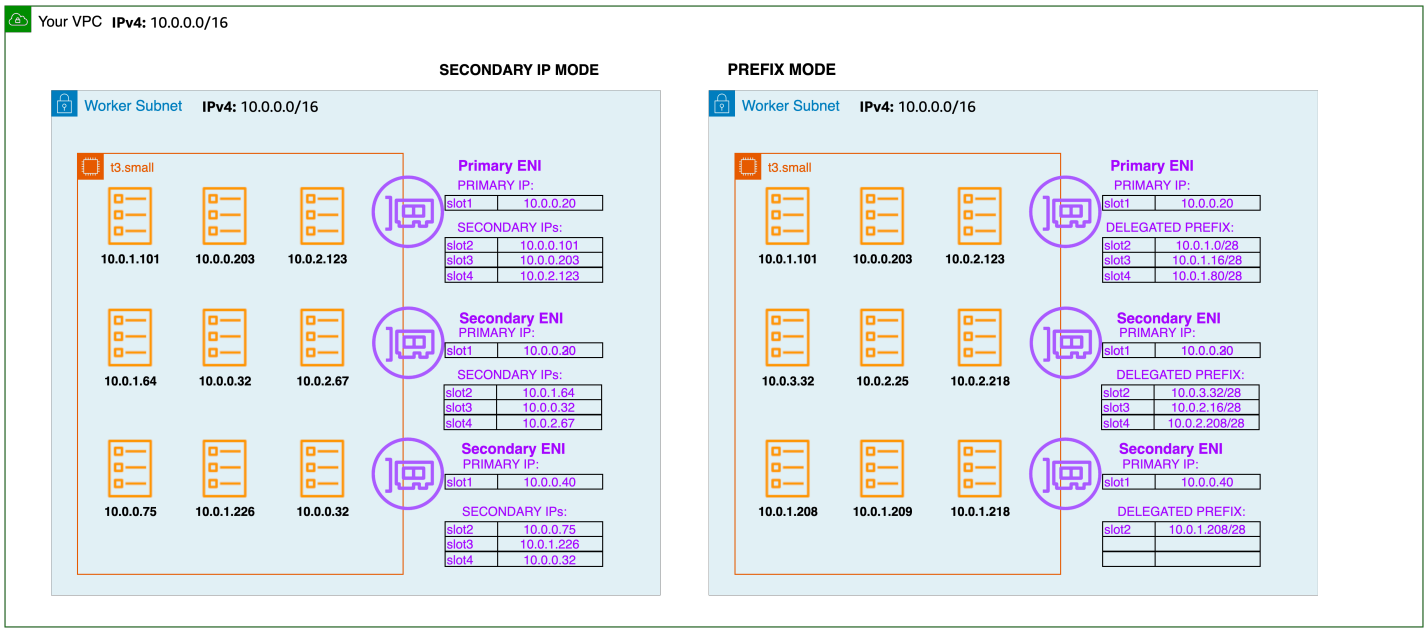
Präfix-Modus für Linux

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Amazon VPC CNI weist Amazon EC2 EC2-Netzwerkschnittstellen Netzwerkpräfixe zu, um die Anzahl der für Knoten verfügbaren IP-Adressen zu erhöhen und die Pod-Dichte pro Knoten zu erhöhen. Sie können Version 1.9.0 oder höher des Amazon VPC CNI-Add-ons so konfigurieren, dass Netzwerkschnittstellen einzelne sekundäre IP-Adressen zugewiesen werden IPv4 und IPv6 CIDRs nicht.

Der Präfix-Modus ist standardmäßig auf IPv6 Clustern aktiviert und ist die einzige unterstützte Option. Die VPC CNI weist einem Steckplatz auf einer ENI ein IPv6 /80-Präfix zu. Weitere Informationen finden Sie im IPv6 Abschnitt dieses Handbuchs.

Im Präfixzuweisungsmodus bleibt die maximale Anzahl von Elastic Network-Schnittstellen pro Instance-Typ gleich, aber Sie können Amazon VPC CNI jetzt so konfigurieren, dass /28 (16 IP-Adressen) IPv4 Adresspräfixe zugewiesen werden, anstatt den Steckplätzen auf Netzwerkschnittstellen einzelne IPv4 Adressen zuzuweisen. Wenn auf true gesetzt `ENABLE_PREFIX_DELEGATION` ist, weist VPC CNI einem Pod eine IP-Adresse aus dem Präfix zu, das einem ENI zugewiesen wurde. Bitte folgen Sie den Anweisungen im EKS-Benutzerhandbuch, um den Prefix-IP-Modus zu aktivieren.

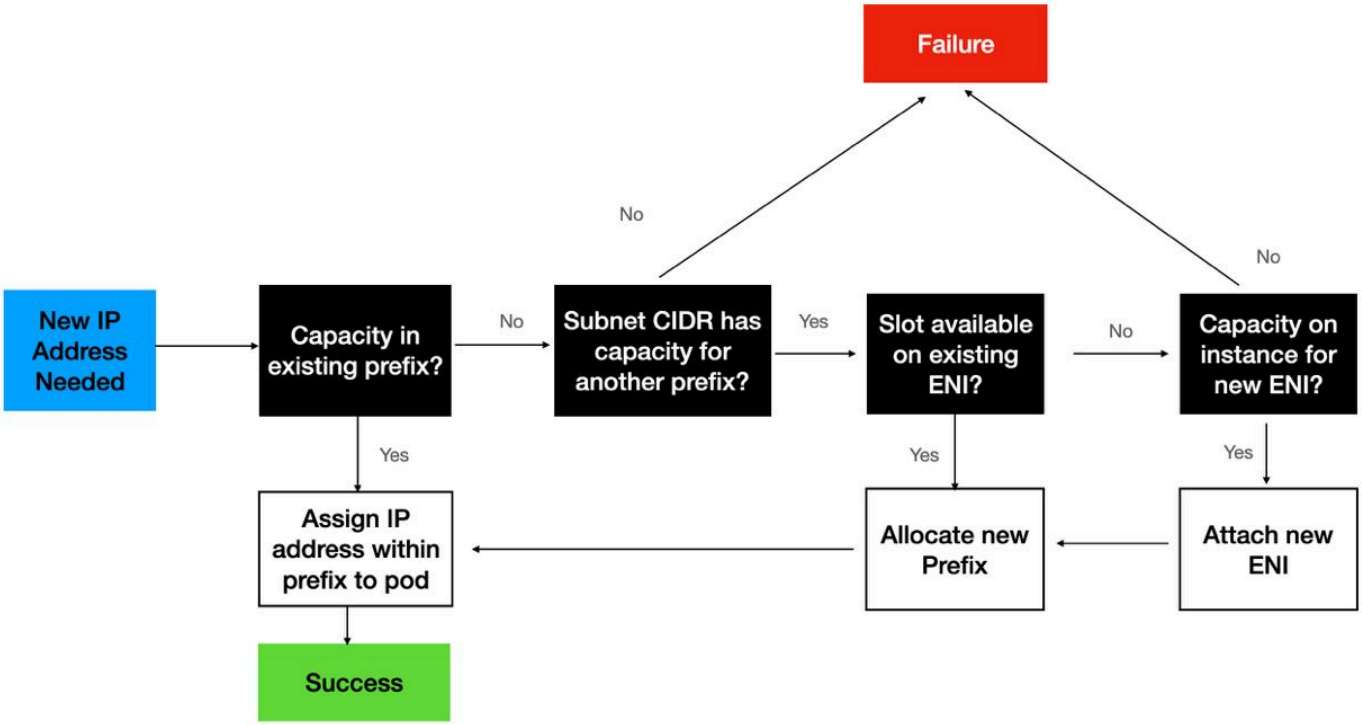


Die maximale Anzahl der IP-Adressen, die Sie einer Netzwerkschnittstelle zuweisen können, hängt vom Instance-Typ ab. Jedes Präfix, das Sie einer Netzwerkschnittstelle zuweisen, zählt als eine IP-Adresse. Zum Beispiel hat eine `c5.large`-Instance ein Limit von `10` IPv4-Adressen pro Netzwerkschnittstelle. Jede Netzwerkschnittstelle für diese Instance hat eine primäre IPv4-Adresse. Wenn eine Netzwerkschnittstelle keine sekundären IPv4-Adressen hat, können Sie der Netzwerkschnittstelle bis zu 9 Präfixe zuweisen. Für jede weitere IPv4-Adresse, die Sie einer Netzwerkschnittstelle zuweisen, können Sie der Netzwerkschnittstelle ein Präfix weniger zuweisen. Lesen Sie die AWS EC2-Dokumentation zu IP-Adressen pro Netzwerkschnittstelle pro Instance-Typ und zur Zuweisung von Präfixen zu Netzwerkschnittstellen.

Während der Initialisierung des Worker-Knotens weist die VPC-CNI der primären ENI ein oder mehrere Präfixe zu. Das CNI weist vorab ein Präfix für einen schnelleren Pod-Start zu, indem ein warmer Pool aufrechterhalten wird. Die Anzahl der Präfixe, die im warmen Pool gespeichert werden sollen, kann durch die Einstellung von Umgebungsvariablen gesteuert werden.

- `WARM_PREFIX_TARGET`, die Anzahl der Präfixe, die über den aktuellen Bedarf hinaus zugewiesen werden sollen.
- `WARM_IP_TARGET`, die Anzahl der IP-Adressen, die über den aktuellen Bedarf hinaus zugewiesen werden sollen.
- `MINIMUM_IP_TARGET`, die Mindestanzahl von IP-Adressen, die jederzeit verfügbar sein müssen.
- `WARM_IP_TARGET`und `MINIMUM_IP_TARGET` falls gesetzt, wird es überschrieben`WARM_PREFIX_TARGET`.

Da mehr Pods geplant sind, werden zusätzliche Präfixe für das bestehende ENI angefordert. Zunächst versucht die VPC CNI, einer vorhandenen ENI ein neues Präfix zuzuweisen. Wenn die ENI ausgelastet ist, versucht die VPC CNI, dem Knoten eine neue ENI zuzuweisen. Neue ENIs werden hinzugefügt, bis das maximale ENI-Limit (definiert durch den Instance-Typ) erreicht ist. Wenn eine neue ENI angehängt wird, weist ipamd ein oder mehrere Präfixe zu, die zur Beibehaltung der Einstellung `WARM_PREFIX_TARGETWARM_IP_TARGET`, und erforderlich sind. `MINIMUM_IP_TARGET`



Empfehlungen

Verwenden Sie den Präfixmodus, wenn

Verwenden Sie den Präfixmodus, wenn auf den Worker-Knoten ein Problem mit der Pod-Dichte auftritt. Um VPC-CNI-Fehler zu vermeiden, empfehlen wir, die Subnetze auf zusammenhängende Adressblöcke für das Präfix /28 zu untersuchen, bevor Sie in den Präfixmodus migrieren. Einzelheiten zur Subnetzreservierung finden Sie im Abschnitt „Subnetzreservierungen verwenden, um Subnetzfragmentierung zu vermeiden“ (). IPv4

Aus Gründen der Abwärtskompatibilität ist das Max-Pods-Limit so eingestellt, dass es den sekundären IP-Modus unterstützt. Um die Pod-Dichte zu erhöhen, geben Sie bitte den `max-pods` Wert für Kubelet und `--use-max-pods=false` als Benutzerdaten für die Knoten an. Sie könnten erwägen, das max-pod-calculator.sh-Skript zu verwenden, um die von EKS empfohlene maximale

Anzahl von Pods für einen bestimmten Instance-Typ zu berechnen. Beispiele für Benutzerdaten finden Sie im EKS-Benutzerhandbuch.

```
./max-pods-calculator.sh --instance-type m5.large --cni-version ``1.9``.0 --cni-prefix-delegation-enabled
```

Der Präfixzuweisungsmodus ist besonders für Benutzer von benutzerdefinierten CNI-Netzwerken relevant, bei denen das primäre ENI nicht für Pods verwendet wird. Mit der Präfixzuweisung können Sie immer noch mehr IPs an fast jeden Nitro-Instance-Typ anhängen, auch ohne die primäre ENI, die für Pods verwendet wird.

Vermeiden Sie den Präfix-Modus, wenn

Wenn Ihr Subnetz stark fragmentiert ist und nicht genügend IP-Adressen verfügbar sind, um /28-Präfixe zu erstellen, vermeiden Sie den Präfixmodus. Die Präfixzuweisung kann fehlschlagen, wenn das Subnetz, aus dem das Präfix generiert wird, fragmentiert ist (ein stark frequentiertes Subnetz mit verstreuten sekundären IP-Adressen). Dieses Problem kann vermieden werden, indem ein neues Subnetz erstellt und ein Präfix reserviert wird.

Im Präfixmodus wird die den Worker-Knoten zugewiesene Sicherheitsgruppe von den Pods gemeinsam genutzt. Erwägen Sie die Verwendung von Sicherheitsgruppen für Pods, wenn Sie Sicherheitsanforderungen haben, um die Einhaltung von Vorschriften zu erreichen, indem Sie Anwendungen mit unterschiedlichen Netzwerksicherheitsanforderungen auf gemeinsam genutzten Rechenressourcen ausführen.

Verwenden Sie ähnliche Instanztypen in derselben Knotengruppe

Ihre Knotengruppe kann Instanzen vieler Typen enthalten. Wenn eine Instanz eine niedrige maximale Pod-Anzahl hat, wird dieser Wert auf alle Knoten in der Knotengruppe angewendet. Erwägen Sie, ähnliche Instanztypen in einer Knotengruppe zu verwenden, um die Knotennutzung zu maximieren. Wir empfehlen, node.kubernetes.io/instance-type im Anforderungsteil der Provisioner-API zu konfigurieren, wenn Sie Karpenter für die automatisierte Knotenskalierung verwenden.

> **Warning**
>
> Die maximale Pod-Anzahl für alle Knoten in einer bestimmten Knotengruppe wird durch die niedrigste maximale Pod-Anzahl aller einzelnen Instance-Typen in der Knotengruppe definiert.

So konfigurieren**WARM_PREFIX_TARGET**, dass Adressen gespeichert werden IPv4

Der Standardwert für des Installationsmanifests `WARM_PREFIX_TARGET` ist 1. In den meisten Fällen bietet der empfohlene Wert 1 für `WARM_PREFIX_TARGET` eine gute Mischung aus schnellen Pod-Startzeiten und der Minimierung ungenutzter IP-Adressen, die der Instance zugewiesen sind.

Wenn Sie IPv4 Adressen pro Node-Nutzung `WARM_IP_TARGET` und `MINIMUM_IP_TARGET` Einstellungen, die `WARM_PREFIX_TARGET` bei der Konfiguration außer Kraft gesetzt werden, weiter speichern müssen. Durch `WARM_IP_TARGET` die Einstellung auf einen Wert unter 16 können Sie verhindern, dass das CNI ein ganzes überschüssiges Präfix beibehält.

Ziehen Sie es vor, neue Präfixe zuzuweisen, anstatt ein neues ENI anzuhängen

Das Zuweisen eines zusätzlichen Präfixes zu einer vorhandenen ENI ist ein schnellerer EC2-API-Vorgang als das Erstellen und Anhängen eines neuen ENI an die Instance. Die Verwendung von Präfixen verbessert die Leistung und ist gleichzeitig sparsam bei der Adresszuweisung. IPv4 Das Anhängen eines Präfixes ist in der Regel in weniger als einer Sekunde abgeschlossen, wohingegen das Anhängen einer neuen ENI bis zu 10 Sekunden dauern kann. In den meisten Anwendungsfällen benötigt das CNI nur eine einzige ENI pro Worker-Knoten, wenn es im Präfixmodus ausgeführt wird. Wenn Sie sich (im schlimmsten Fall) bis zu 15 ungenutzte IPs Knoten leisten können, empfehlen wir dringend, den neueren Netzwerkmodus mit Präfixzuweisung zu verwenden und die damit verbundenen Leistungs- und Effizienzsteigerungen zu nutzen.

Verwenden Sie Subnetzreservierungen, um Subnetzfragmentierung zu vermeiden () IPv4

Wenn EC2 einer ENI ein IPv4 /28-Präfix zuweist, muss es sich um einen zusammenhängenden Block von IP-Adressen aus Ihrem Subnetz handeln. Wenn das Subnetz, aus dem das Präfix generiert wird, fragmentiert ist (ein häufig verwendetes Subnetz mit verstreuten sekundären IP-Adressen), schlägt die Präfixzuweisung möglicherweise fehl, und in den VPC-CNI-Protokollen wird die folgende Fehlermeldung angezeigt:

```
failed to allocate a private IP/Prefix address: InsufficientCidrBlocks: There are not
 enough free cidr blocks in the specified subnet to satisfy the request.
```

Um Fragmentierung zu vermeiden und über ausreichend zusammenhängenden Speicherplatz für die Erstellung von Präfixen zu verfügen, können Sie VPC-Subnetz-CIDR-Reservierungen verwenden, um IP-Speicherplatz innerhalb eines Subnetzes für die ausschließliche Verwendung durch Präfixe

zu reservieren. Sobald Sie eine Reservierung erstellt haben, ruft das VPC-CNI-Plugin EC2 auf, um Präfixe APIs zuzuweisen, die automatisch aus dem reservierten Speicherplatz zugewiesen werden.

Es wird empfohlen, ein neues Subnetz zu erstellen, Speicherplatz für Präfixe zu reservieren und die Präfixzuweisung mit VPC CNI für Worker-Knoten zu aktivieren, die in diesem Subnetz laufen. Wenn das neue Subnetz nur für Pods reserviert ist, die in Ihrem EKS-Cluster mit aktivierter VPC-CNI-Präfixzuweisung ausgeführt werden, können Sie den Schritt zur Präfixreservierung überspringen.

Vermeiden Sie ein Downgrade von VPC CNI

Der Präfixmodus funktioniert mit VPC CNI Version 1.9.0 und höher. Ein Downgrade des Amazon VPC CNI-Add-ons auf eine niedrigere Version als 1.9.0 muss vermieden werden, sobald der Präfixmodus aktiviert und Präfixe zugewiesen wurden. ENIs Sie müssen Knoten löschen und neu erstellen, wenn Sie sich für ein Downgrade des VPC-CNI entscheiden.

Ersetzen Sie alle Knoten während des Übergangs zur Präfix-Delegierung

Es wird dringend empfohlen, neue Knotengruppen zu erstellen, um die Anzahl der verfügbaren IP-Adressen zu erhöhen, anstatt bestehende Worker-Knoten fortlaufend zu ersetzen. Sperren und entleeren Sie alle vorhandenen Knoten, um all Ihre vorhandenen Pods sicher zu entfernen. Um Serviceunterbrechungen zu vermeiden, empfehlen wir, Pod-Disruption-Budgets für Ihre Produktionscluster für kritische Workloads zu implementieren. Pods auf neuen Knoten erhalten eine IP aus einem Präfix, das einer ENI zugewiesen ist. Nachdem Sie bestätigt haben, dass die Pods ausgeführt werden, können Sie die alten Knoten und Knotengruppen löschen. Wenn Sie verwaltete Knotengruppen verwenden, folgen Sie bitte den hier genannten Schritten, um eine Knotengruppe sicher zu löschen.

Präfix-Modus für Windows

In Amazon EKS wird jedem Pod, der auf einem Windows-Host ausgeführt wird, standardmäßig vom VPC-Ressourcencontroller eine sekundäre IP-Adresse zugewiesen. Diese IP-Adresse ist eine VPC-routbare Adresse, die vom Subnetz des Hosts zugewiesen wird. Unter Linux verfügt jedes mit der Instance verbundene ENI über mehrere Steckplätze, die mit einer sekundären IP-Adresse oder einem /28-CIDR (einem Präfix) belegt werden können. Windows-Hosts unterstützen jedoch nur eine einzelne ENI und ihre verfügbaren Steckplätze. Wenn Sie nur sekundäre IP-Adressen verwenden, kann die Anzahl der Pods, die Sie auf einem Windows-Host ausführen können, künstlich begrenzt werden, selbst wenn eine Fülle von IP-Adressen für die Zuweisung verfügbar ist.

Um die Pod-Dichte auf Windows-Hosts zu erhöhen, insbesondere bei Verwendung kleinerer Instance-Typen, können Sie die Präfix-Delegierung für Windows-Knoten aktivieren. Wenn die Präfix-Delegierung aktiviert ist, werden IPv4 /28-Präfixe ENI-Steckplätzen und nicht sekundären IP-Adressen zugewiesen. Die Präfix-Delegierung kann aktiviert werden, indem der `enable-windows-prefix-delegation: "true"` Eintrag zur `amazon-vpc-cni` Konfigurationsübersicht hinzugefügt wird. Dies ist dieselbe Konfigurationsübersicht, in der Sie einen `enable-windows-ipam: "true"` Eintrag für die Aktivierung der Windows-Unterstützung festlegen müssen.

Bitte folgen Sie den Anweisungen im EKS-Benutzerhandbuch, um den Präfix-Delegierungsmodus für Windows-Knoten zu aktivieren.

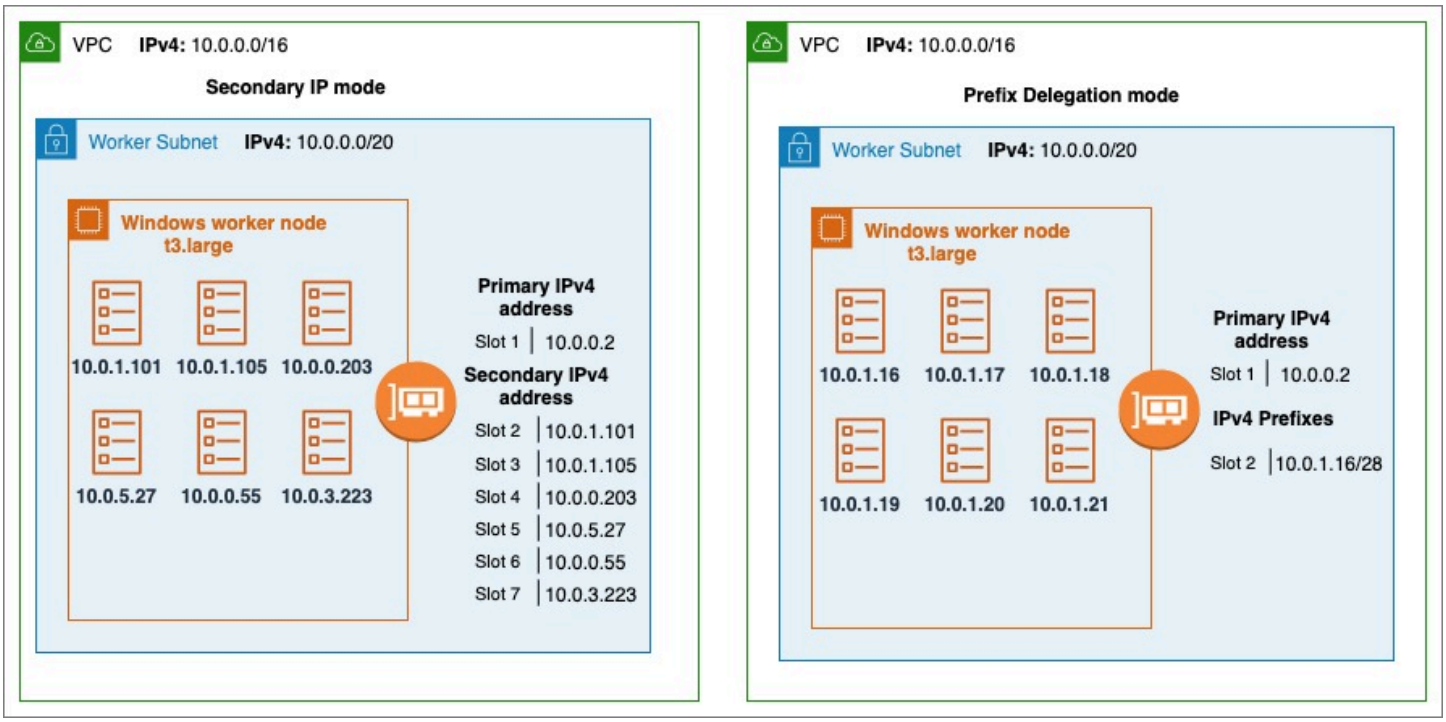


Abbildung: Vergleich des sekundären IP-Modus mit dem Präfix-Delegierungsmodus

Die maximale Anzahl von IP-Adressen, die Sie einer Netzwerkschnittstelle zuweisen können, hängt vom Instanztyp und dessen Größe ab. Jedes einer Netzwerkschnittstelle zugewiesene Präfix belegt einen verfügbaren Steckplatz. Zum Beispiel hat eine `c5.large` Instanz ein Limit an `10` Steckplätzen pro Netzwerkschnittstelle. Der erste Steckplatz einer Netzwerkschnittstelle wird immer von der primären IP-Adresse der Schnittstelle belegt, sodass Ihnen 9 Steckplätze für and/or sekundäre Präfixe zur Verfügung stehen. Wenn diesen Steckplätzen Präfixe zugewiesen sind, kann der Knoten (9 * 16) 144 IP-Adressen unterstützen, während er, wenn ihnen sekundäre IP-Adressen zugewiesen sind, nur 9 IP-Adressen unterstützen kann. Weitere Informationen finden Sie in

der Dokumentation zu IP-Adressen pro Netzwerkschnittstelle pro Instanztyp und zur Zuweisung von Präfixen zu Netzwerkschnittstellen.

Während der Initialisierung des Worker-Knotens weist der VPC Resource Controller der primären ENI ein oder mehrere Präfixe zu, um den Pod-Start zu beschleunigen, indem ein warmer Pool mit IP-Adressen aufrechterhalten wird. Die Anzahl der Präfixe, die im warmen Pool gespeichert werden sollen, kann gesteuert werden, indem die folgenden Konfigurationsparameter in der Konfigurationsübersicht festgelegt werden. `amazon-vpc-cni`

- `warm-prefix-target`, die Anzahl der Präfixe, die über den aktuellen Bedarf hinaus zugewiesen werden sollen.
- `warm-ip-target`, die Anzahl der IP-Adressen, die über den aktuellen Bedarf hinaus zugewiesen werden sollen.
- `minimum-ip-target`, die Mindestanzahl von IP-Adressen, die jederzeit verfügbar sein müssen.
- `warm-ip-target` and/or `minimum-ip-target`wenn gesetzt, wird es überschriebenwarm-prefix-target.

Wenn mehr Pods auf dem Knoten geplant sind, werden zusätzliche Präfixe für das bestehende ENI angefordert. Wenn ein Pod auf dem Knoten geplant ist, versucht VPC Resource Controller zunächst, eine IPv4 Adresse aus den vorhandenen Präfixen auf dem Knoten zuzuweisen. Wenn das nicht möglich ist, wird ein neues IPv4 Präfix angefordert, sofern das Subnetz über die erforderliche Kapazität verfügt.

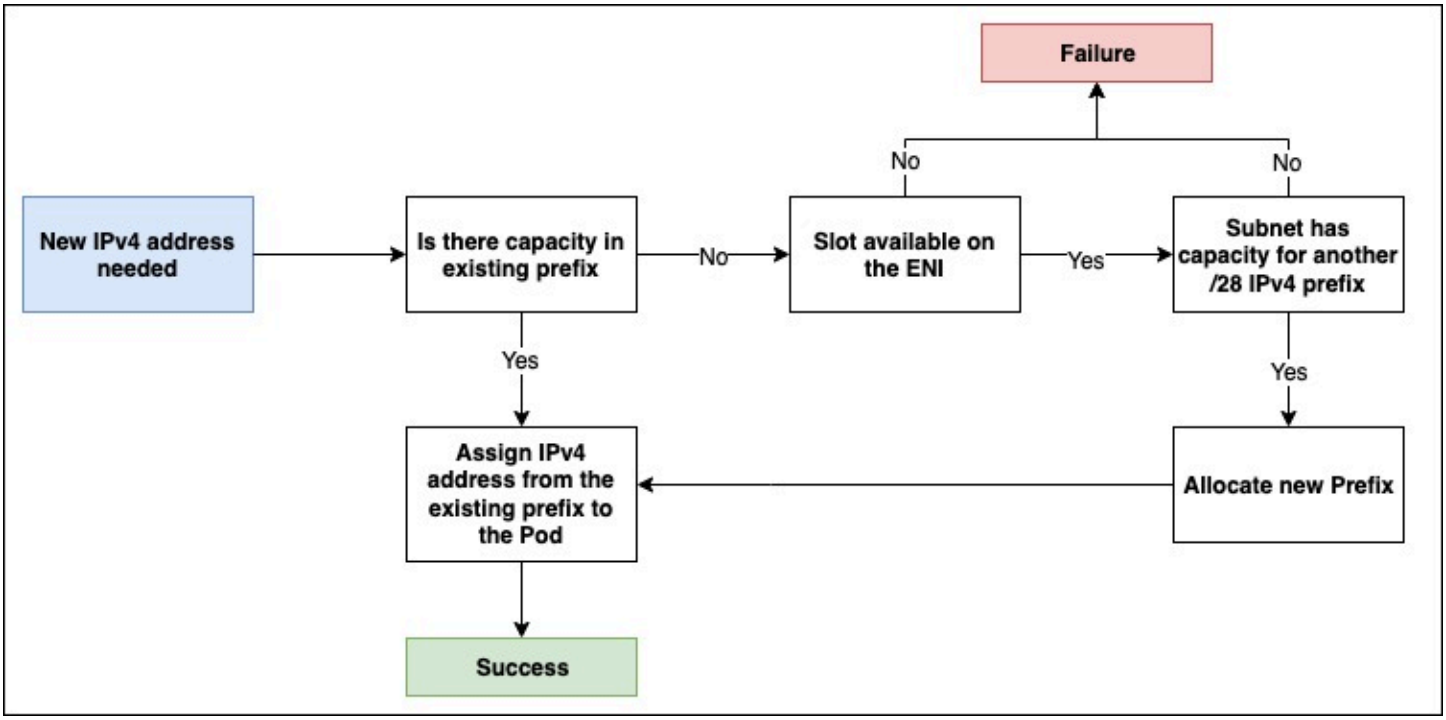


Abbildung: Arbeitsablauf bei der IPv4 Adresszuweisung an den Pod

Empfehlungen

Verwenden Sie die Präfix-Delegierung wenn

Verwenden Sie die Präfix-Delegierung, wenn Sie Probleme mit der Pod-Dichte auf den Worker-Knoten haben. Um Fehler zu vermeiden, empfehlen wir, die Subnetze auf zusammenhängende Adressblöcke für das Präfix /28 zu untersuchen, bevor Sie in den Präfixmodus migrieren. Einzelheiten zur Subnetzreservierung finden Sie im Abschnitt „Subnetzreservierungen verwenden, um Subnetzfragmentierung zu vermeiden () IPv4“.

Standardmäßig ist der On-Windows-Knoten `max-pods` auf eingestellt. `110` Für die überwiegende Mehrheit der Instance-Typen sollte dies ausreichend sein. Wenn Sie dieses Limit erhöhen oder verringern möchten, fügen Sie dem Bootstrap-Befehl in Ihren Benutzerdaten Folgendes hinzu:

```
-KubeletExtraArgs '--max-pods=example-value'
```

Weitere Informationen zu den Bootstrap-Konfigurationsparametern für Windows-Knoten finden Sie in der Dokumentation hier.

Vermeiden Sie Präfix-Delegierung, wenn

Wenn Ihr Subnetz stark fragmentiert ist und nicht genügend IP-Adressen verfügbar sind, um /28-Präfixe zu erstellen, vermeiden Sie den Präfixmodus. Die Präfixzuweisung kann fehlschlagen, wenn das Subnetz, aus dem das Präfix generiert wird, fragmentiert ist (ein stark frequentiertes Subnetz mit verstreuten sekundären IP-Adressen). Dieses Problem kann vermieden werden, indem ein neues Subnetz erstellt und ein Präfix reserviert wird.

Konfigurieren Sie Parameter für die Präfix-Delegierung, um Adressen zu erhalten IPv4

`warm-prefix-targetwarm-ip-target`, und `minimum-ip-target` kann zur Feinabstimmung des Verhaltens der Vorskalierung und der dynamischen Skalierung mit Präfixen verwendet werden. Standardmäßig werden die folgenden Werte verwendet:

```
warm-ip-target: "1"
minimum-ip-target: "3"
```

Durch die Feinabstimmung dieser Konfigurationsparameter können Sie ein optimales Gleichgewicht zwischen der Beibehaltung der IP-Adressen und der Verringerung der Pod-Latenz aufgrund der Zuweisung von IP-Adressen erreichen. Weitere Informationen zu diesen Konfigurationsparametern finden Sie in der Dokumentation hier.

Verwenden Sie Subnetzreservierungen, um Subnetzfragmentierung zu vermeiden () IPv4

Wenn EC2 einer ENI ein IPv4 /28-Präfix zuweist, muss es sich um einen zusammenhängenden Block von IP-Adressen aus Ihrem Subnetz handeln. Wenn das Subnetz, aus dem das Präfix generiert wird, fragmentiert ist (ein stark frequentiertes Subnetz mit verstreuten sekundären IP-Adressen), schlägt die Präfixzuweisung möglicherweise fehl und es wird das folgende Knotenereignis angezeigt:

```
InsufficientCidrBlocks: The specified subnet does not have enough free cidr blocks to
 satisfy the request
```

Um Fragmentierung zu vermeiden und über ausreichend zusammenhängenden Speicherplatz für die Erstellung von Präfixen zu verfügen, verwenden Sie VPC-Subnetz-CIDR-Reservierungen, um IP-Speicherplatz innerhalb eines Subnetzes für die ausschließliche Verwendung durch Präfixe zu reservieren. Sobald Sie eine Reservierung erstellt haben, werden die IP-Adressen aus den reservierten Blöcken keinen anderen Ressourcen zugewiesen. Auf diese Weise kann VPC Resource Controller während des Zuweisungsaufrufs an den Knoten ENI verfügbare Präfixe abrufen.

Es wird empfohlen, ein neues Subnetz zu erstellen, Speicherplatz für Präfixe zu reservieren und die Präfixzuweisung für Worker-Knoten zu aktivieren, die in diesem Subnetz laufen. Wenn das neue Subnetz nur für Pods reserviert ist, die in Ihrem EKS-Cluster mit aktivierter Präfix-Delegierung ausgeführt werden, können Sie den Schritt zur Präfixreservierung überspringen.

Ersetzen Sie alle Knoten, wenn Sie vom sekundären IP-Modus in den Präfix-Delegierungsmodus oder umgekehrt migrieren

Es wird dringend empfohlen, neue Knotengruppen zu erstellen, um die Anzahl der verfügbaren IP-Adressen zu erhöhen, anstatt bestehende Worker-Knoten fortlaufend zu ersetzen.

Bei der Verwendung von selbstverwalteten Knotengruppen wären die Schritte für den Übergang wie folgt:

- Erhöhen Sie die Kapazität in Ihrem Cluster, sodass die neuen Knoten Ihre Workloads aufnehmen können
- Aktivieren/Deaktivieren Sie die Funktion zur Präfix-Delegierung für Windows
- Sperren und entleeren Sie alle vorhandenen Knoten, um all Ihre vorhandenen Pods sicher zu entfernen. Um Serviceunterbrechungen zu vermeiden, empfehlen wir, Pod-Disruption-Budgets für Ihre Produktionscluster für kritische Workloads zu implementieren.
- Nachdem Sie bestätigt haben, dass die Pods ausgeführt werden, können Sie die alten Knoten und Knotengruppen löschen. Pods auf neuen Knoten wird eine IPv4 Adresse aus einem Präfix zugewiesen, das dem Knoten ENI zugewiesen ist.

Bei der Verwendung verwalteter Knotengruppen wären die Schritte für den Übergang wie folgt:

- Aktivieren/Deaktivieren Sie die Funktion zur Präfix-Delegierung für Windows
- Aktualisieren Sie die Knotengruppe mithilfe der hier genannten Schritte. Dabei werden ähnliche Schritte wie oben ausgeführt, die jedoch von EKS verwaltet werden.

> **Warning**
>
> Führen Sie alle Pods auf einem Knoten im selben Modus aus

Für Windows empfehlen wir, Pods nicht gleichzeitig im sekundären IP-Modus und im Präfix-Delegierungsmodus auszuführen. Eine solche Situation kann auftreten, wenn Sie bei laufenden

Windows-Workloads vom sekundären IP-Modus in den Präfix-Delegierungsmodus oder umgekehrt migrieren.

Dies hat zwar keine Auswirkungen auf Ihre laufenden Pods, es kann jedoch zu Inkonsistenzen in Bezug auf die IP-Adresskapazität des Knotens kommen. Stellen Sie sich zum Beispiel einen t3.xlarge-Knoten vor, der über 14 Steckplätze für sekundäre Adressen verfügt. IPv4 Wenn Sie 10 Pods betreiben, werden 10 Steckplätze auf der ENI von sekundären IP-Adressen belegt. Nachdem Sie die Präfix-Delegierung aktiviert haben, würde die dem Kube-API-Server angekündigte Kapazität (14 Steckplätze * 16 IP-Adressen pro Präfix) 244 betragen, aber die tatsächliche Kapazität wäre zu diesem Zeitpunkt (4 verbleibende Steckplätze * 16 Adressen pro Präfix) 64. Diese Inkonsistenz zwischen der angekündigten Kapazität und der tatsächlichen Kapazität (verbleibende Steckplätze) kann zu Problemen führen, wenn Sie mehr Pods ausführen, als IP-Adressen für die Zuweisung verfügbar sind.

Davon abgesehen können Sie die oben beschriebene Migrationsstrategie verwenden, um Ihre Pods sicher von einer sekundären IP-Adresse auf Adressen umzustellen, die über Präfixe abgerufen wurden. Wenn Sie zwischen den Modi wechseln, laufen die Pods normal weiter und:

- Beim Umschalten vom sekundären IP-Modus in den Präfix-Delegierungsmodus werden die den laufenden Pods zugewiesenen sekundären IP-Adressen nicht freigegeben. Den freien Steckplätzen werden Präfixe zugewiesen. Sobald ein Pod beendet ist, werden die sekundäre IP und der Steckplatz, den er verwendet hat, freigegeben.
- Beim Umschalten vom Präfix-Delegierungsmodus in den sekundären IP-Modus wird ein Präfix freigegeben, wenn nicht mehr alle IPs in seinem Bereich befindlichen Pods zugewiesen sind. Wenn einem Pod eine IP aus dem Präfix zugewiesen wird, wird dieses Präfix beibehalten, bis die Pods beendet werden.

Debugging-Probleme mit der Präfix-Delegierung

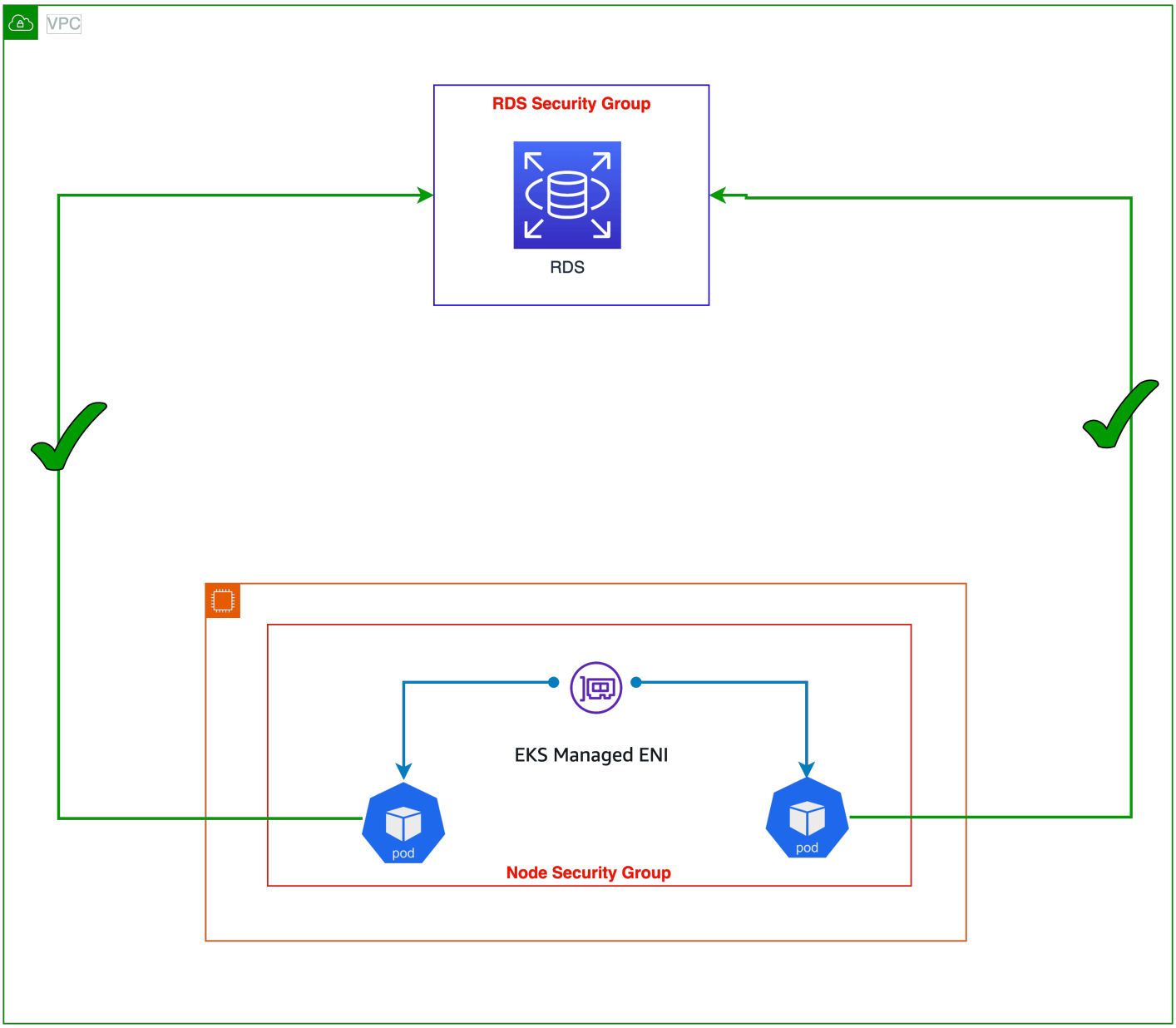
Sie können unseren Debugging-Leitfaden hier verwenden, um sich eingehend mit dem Problem zu befassen, mit dem Sie bei der Präfix-Delegierung unter Windows konfrontiert sind.

Sicherheitsgruppen pro Pod

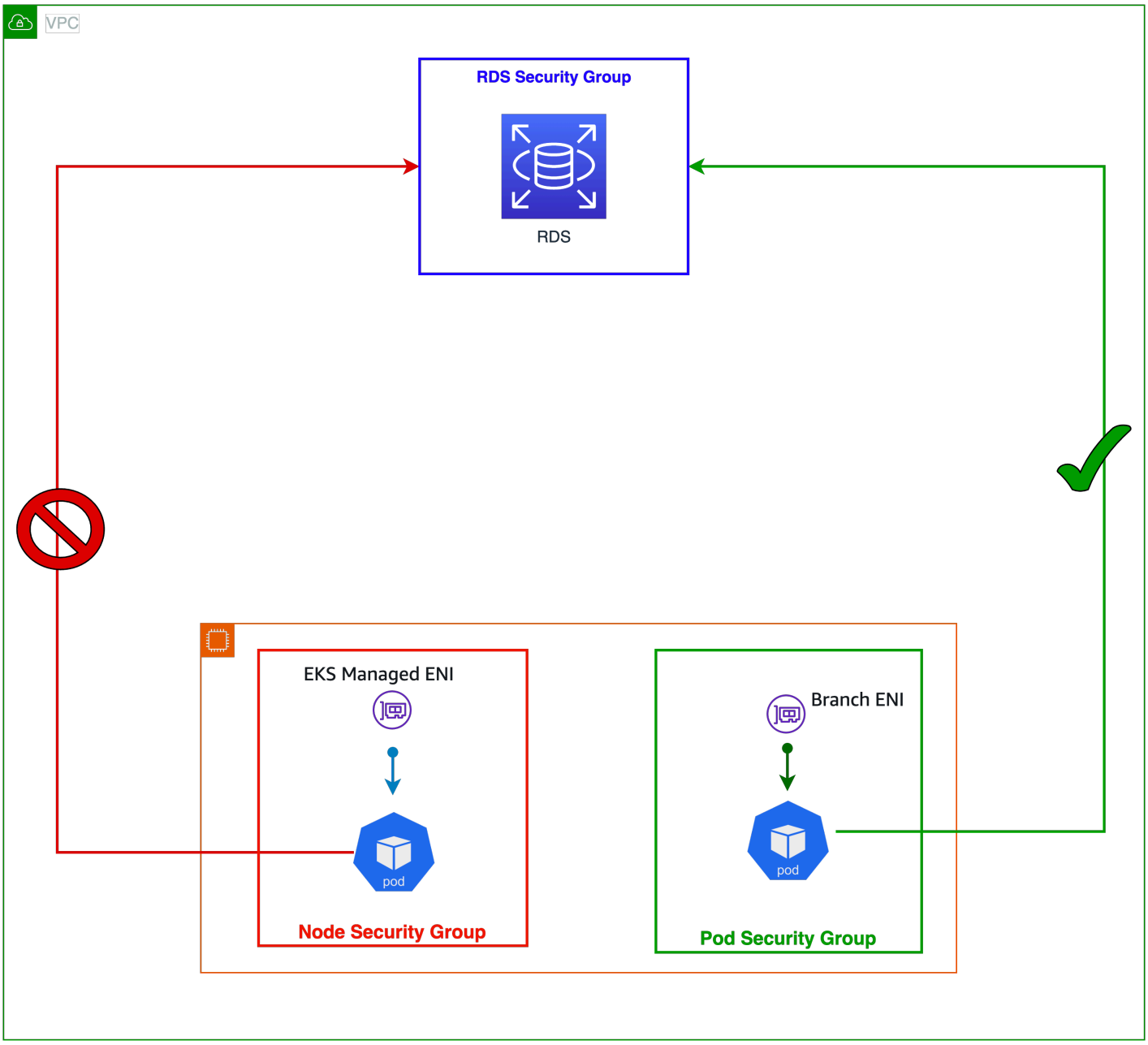
> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Eine AWS-Sicherheitsgruppe fungiert als virtuelle Firewall für EC2-Instances, um eingehenden und ausgehenden Datenverkehr zu kontrollieren. Standardmäßig verwendet das Amazon VPC CNI Sicherheitsgruppen, die der primären ENI auf dem Knoten zugeordnet sind. Genauer gesagt hat jedes der Instance zugeordnete ENI dieselben EC2-Sicherheitsgruppen. Somit verwendet jeder Pod auf einem Knoten dieselben Sicherheitsgruppen wie der Knoten, auf dem er ausgeführt wird.

Wie in der Abbildung unten zu sehen ist, haben alle Anwendungs-Pods, die auf Worker-Knoten ausgeführt werden, Zugriff auf den RDS-Datenbankdienst (wobei berücksichtigt wird, dass RDS Inbound die Knotensicherheitsgruppe zulässt). Sicherheitsgruppen sind zu grob gefasst, da sie für alle Pods gelten, die auf einem Knoten ausgeführt werden. Sicherheitsgruppen für Pods bieten Netzwerksegmentierung für Workloads, was ein wesentlicher Bestandteil einer guten Verteidigungsstrategie ist.

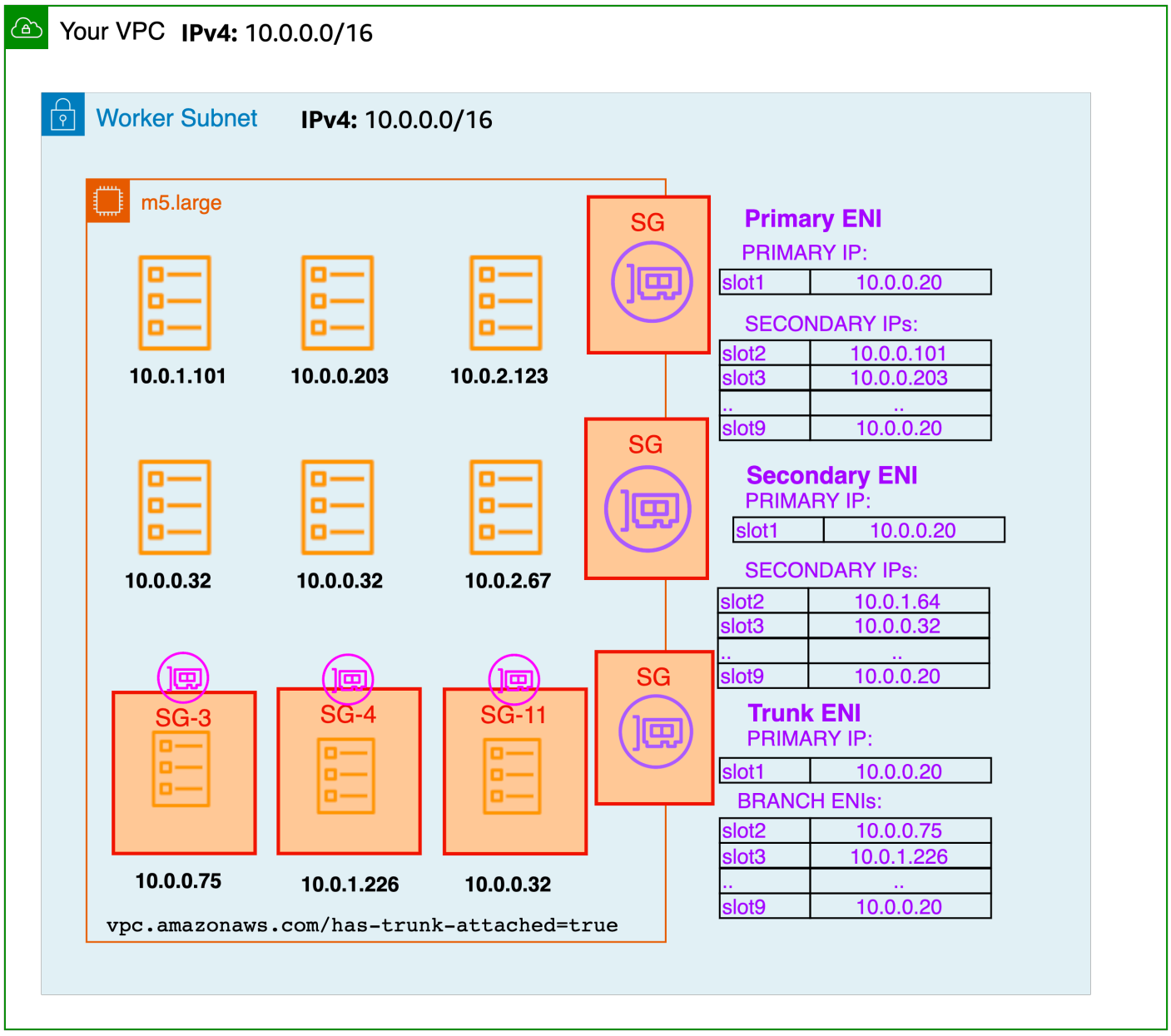


Mit Sicherheitsgruppen für Pods können Sie die Recheneffizienz verbessern, indem Sie Anwendungen mit unterschiedlichen Netzwerksicherheitsanforderungen auf gemeinsam genutzten Rechenressourcen ausführen. Mehrere Arten von Sicherheitsregeln, z. B. Pod-to-Pod Pod-to-External AWS-Services, können mit EC2-Sicherheitsgruppen an einem einzigen Ort definiert und auf Workloads mit nativem Kubernetes-Betriebssystem angewendet werden. APIs Die Abbildung unten zeigt Sicherheitsgruppen, die auf Pod-Ebene angewendet werden, und zeigt, wie sie Ihre Anwendungsbereitstellung und Ihre Knotenarchitektur vereinfachen. Der Pod kann jetzt auf die Amazon RDS-Datenbank zugreifen.



Sie können Sicherheitsgruppen für Pods aktivieren, indem Sie VPC CNI einstellen`ENABLE_POD_ENI=true`. Nach der Aktivierung erstellt der VPC Resource Controller, der auf der Steuerungsebene läuft (von EKS verwaltet), eine Trunk-Schnittstelle namens „`aws-k8 „`s-trunk-eniund fügt sie dem Knoten hinzu. Die Trunk-Schnittstelle fungiert als Standard-Netzwerkschnittstelle, die an die Instanz angeschlossen ist. Um Trunk-Schnittstellen zu verwalten, müssen Sie die `AmazonEKSVPCResourceController` verwaltete Richtlinie der Cluster-Rolle hinzufügen, die zu Ihrem Amazon EKS-Cluster gehört.

Der Controller erstellt auch Zweigschnittstellen mit dem Namen „aws-k8s-branch-eni“ und ordnet sie der Trunk-Schnittstelle zu. Den Pods wird mithilfe der SecurityGroupPolicybenutzerdefinierten Ressource eine Sicherheitsgruppe zugewiesen und sie sind einer Branch-Schnittstelle zugeordnet. Da Sicherheitsgruppen mit Netzwerkschnittstellen spezifiziert werden, können wir jetzt Pods planen, für die bestimmte Sicherheitsgruppen auf diesen zusätzlichen Netzwerkschnittstellen erforderlich sind. Lesen Sie den Abschnitt Sicherheitsgruppen für Pods im EKS-Benutzerhandbuch, einschließlich der Voraussetzungen für die Bereitstellung.



Die Kapazität der Zweigstellenschnittstelle wird zu den bestehenden Beschränkungen für Instance-Typen für sekundäre IP-Adressen addiert. Pods, die Sicherheitsgruppen verwenden, werden in

der Max-Pods-Formel nicht berücksichtigt. Wenn Sie eine Sicherheitsgruppe für Pods verwenden, müssen Sie eine Erhöhung des Max-Pods-Werts in Betracht ziehen oder damit einverstanden sein, weniger Pods auszuführen, als der Knoten tatsächlich unterstützen kann.

Einem m5.large können bis zu 9 Zweignetzwerkschnittstellen und bis zu 27 sekundäre IP-Adressen den Standardnetzwerkschnittstellen zugewiesen werden. Wie im Beispiel unten gezeigt, sind die standardmäßigen Max-Pods für eine m5.large 29, und EKS zählt die Pods, die Sicherheitsgruppen verwenden, auf die maximale Anzahl an Pods. Anweisungen zum Ändern der Max-Pods für Knoten finden Sie im EKS-Benutzerhandbuch.

Wenn Sicherheitsgruppen für Pods in Kombination mit benutzerdefinierten Netzwerken verwendet werden, wird die Sicherheitsgruppe verwendet, die in den Sicherheitsgruppen für Pods definiert ist, und nicht die in der angegebene Sicherheitsgruppe. ENIConfig Wenn benutzerdefinierte Netzwerke aktiviert sind, sollten Sie daher die Reihenfolge der Sicherheitsgruppen sorgfältig prüfen und dabei Sicherheitsgruppen pro Pod verwenden.

Empfehlungen

Deaktivieren Sie TCP Early Demux für Liveness Probe

Wenn Sie Liveness- oder Readiness Probes verwenden, müssen Sie auch TCP-Early-Demux deaktivieren, damit das Kubelet über TCP eine Verbindung zu Pods auf Zweignetzwerkschnittstellen herstellen kann. Dies ist nur im strikten Modus erforderlich. Führen Sie dazu den folgenden Befehl aus:

```
kubectl edit daemonset aws-node -n kube-system
```

Ändern Sie unter dem `initContainer` Abschnitt den Wert für `DISABLE_TCP_EARLY_DEMUX` zu `true`.

Verwenden Sie Security Group For Pods, um bestehende Investitionen in die AWS-Konfiguration zu nutzen.

Sicherheitsgruppen erleichtern die Beschränkung des Netzwerkzugriffs auf VPC-Ressourcen wie RDS-Datenbanken oder EC2-Instances. Ein klarer Vorteil von Sicherheitsgruppen pro Pod ist die Möglichkeit, vorhandene AWS-Sicherheitsgruppenressourcen wiederzuverwenden. Wenn Sie Sicherheitsgruppen als Netzwerk-Firewall verwenden, um den Zugriff auf Ihre AWS-Services zu beschränken, empfehlen wir, Sicherheitsgruppen mithilfe von Branch-ENIs auf Pods anzuwenden.

Erwägen Sie die Verwendung von Sicherheitsgruppen für Pods, wenn Sie Apps von EC2-Instances auf EKS übertragen und den Zugriff auf andere AWS-Services mit Sicherheitsgruppen einschränken.

Konfigurieren Sie den Modus zur Durchsetzung von Pod-Sicherheitsgruppen

Mit Version 1.11 des Amazon VPC CNI-Plug-ins wurde eine neue Einstellung mit dem Namen `POD_SECURITY_GROUP_ENFORCING_MODE` („Enforcing Mode“) hinzugefügt. Der Erzwingungsmodus steuert sowohl, welche Sicherheitsgruppen für den Pod gelten, als auch, ob die Quell-NAT aktiviert ist. Sie können den Erzwingungsmodus entweder als strikt oder als Standard angeben. Strict ist die Standardeinstellung, die das vorherige Verhalten der VPC-CNI mit der `ENABLE_POD_ENI` Einstellung auf widerspiegelt. `true`

Im Strict-Modus werden nur die ENI-Sicherheitsgruppen der Zweigstelle durchgesetzt. Das Quell-NAT ist ebenfalls deaktiviert.

Im Standardmodus werden die Sicherheitsgruppen angewendet, die sowohl der primären ENI als auch der Zweig-ENI (dem Pod zugeordnet) zugeordnet sind. Der Netzwerkverkehr muss beiden Sicherheitsgruppen entsprechen.

> **Warning**
>
> Jede Modusänderung wirkt sich nur auf neu gestartete Pods aus. Bestehende Pods verwenden den Modus, der bei der Erstellung des Pods konfiguriert wurde. Kunden müssen bestehende Pods mit Sicherheitsgruppen recyceln, wenn sie das Verkehrsverhalten ändern möchten.

Erzwungener Modus: Verwenden Sie den Strict-Modus, um den Pod- und Node-Verkehr zu isolieren:

Standardmäßig ist für Sicherheitsgruppen für Pods der „strikte Modus“ festgelegt. Verwenden Sie diese Einstellung, wenn Sie den Pod-Verkehr vollständig vom restlichen Datenverkehr des Knotens trennen müssen. Im strikten Modus ist das Quell-NAT ausgeschaltet, sodass die ausgehenden Branch-ENI-Sicherheitsgruppen verwendet werden können.

> **Warning**
>
> Wenn der strikte Modus aktiviert ist, verlässt der gesamte ausgehende Datenverkehr von einem Pod den Knoten und gelangt in das VPC-Netzwerk. Der Verkehr zwischen Pods

auf demselben Knoten wird über die VPC abgewickelt. Dies erhöht den VPC-Verkehr und schränkt knotenbasierte Funktionen ein. Das NodeLocal DNSCache wird im strikten Modus nicht unterstützt.

Erzwungener Modus: Verwenden Sie den Standardmodus in den folgenden Situationen

Die Client-Quell-IP ist für die Container im Pod sichtbar

Wenn Sie sicherstellen möchten, dass die Client-Quell-IP für die Container im Pod sichtbar bleibt, sollten Sie die Einstellung `POD_SECURITY_GROUP_ENFORCING_MODE` auf in Erwägung ziehen`standard`. Kubernetes-Dienste unterstützen externalTrafficPolicy =local, um die Beibehaltung der Client-Quell-IP zu unterstützen (Standardtyp Cluster). Sie können jetzt Kubernetes-Dienste des Typs NodePort und der LoadBalancer Verwendung von Instanzzielen mit einer externalTrafficPolicy Einstellung auf Lokal im Standardmodus ausführen. `Local`behält die Quell-IP des Clients bei und vermeidet einen zweiten Hop für Services LoadBalancer und NodePort den Typ Services.

Bereitstellen NodeLocal DNSCache

Wenn Sie Sicherheitsgruppen für Pods verwenden, konfigurieren Sie den Standardmodus so, dass er Pods unterstützt, die NodeLocal DNSCache NodeLocal DNSCache verbessert die Cluster-DNS-Leistung, indem ein DNS-Caching-Agent auf Clusterknoten als DaemonSet ausgeführt wird. Dies hilft den Pods mit den höchsten DNS-QPS-Anforderungen, lokale Kube-DNS/CoreDNS mit einem lokalen Cache abzufragen, was die Latenz verbessert.

NodeLocal DNSCache wird im strikten Modus nicht unterstützt, da der gesamte Netzwerkverkehr, auch der zum Knoten, in die VPC gelangt.

Unterstützung der Kubernetes-Netzwerkrichtlinie

Wir empfehlen, den Standard-Erzwingungsmodus zu verwenden, wenn Netzwerkrichtlinien mit Pods verwendet werden, denen Sicherheitsgruppen zugeordnet sind.

Wir empfehlen dringend, Sicherheitsgruppen für Pods zu verwenden, um den Zugriff auf Netzwerkebene auf AWS-Services zu beschränken, die nicht Teil eines Clusters sind. Ziehen Sie Netzwerkrichtlinien in Betracht, um den Netzwerkverkehr zwischen Pods innerhalb eines Clusters einzuschränken, der häufig als East/West Verkehr bezeichnet wird.

Identifizieren Sie Inkompatibilitäten mit Sicherheitsgruppen pro Pod

Windows-basierte und Nicht-Nitro-Instances unterstützen keine Sicherheitsgruppen für Pods. Um Sicherheitsgruppen mit Pods verwenden zu können, müssen die Instanzen mit gekennzeichnet werden. isTrunkingEnabled Verwenden Sie Netzwerkrichtlinien, um den Zugriff zwischen Pods statt Sicherheitsgruppen zu verwalten, wenn Ihre Pods nicht von AWS-Services innerhalb oder außerhalb Ihrer VPC abhängig sind.

Verwenden Sie Sicherheitsgruppen pro Pod, um den Datenverkehr zu AWS-Services effizient zu steuern

Wenn eine Anwendung, die innerhalb des EKS-Clusters ausgeführt wird, mit einer anderen Ressource innerhalb der VPC kommunizieren muss, z. B. einer RDS-Datenbank, sollten Sie die Verwendung SGs für Pods in Betracht ziehen. Zwar gibt es Policy-Engines, mit denen Sie einen CIDR- oder einen DNS-Namen angeben können, aber sie sind eine weniger optimale Wahl, wenn Sie mit AWS-Services kommunizieren, deren Endpunkte sich in einer VPC befinden.

Im Gegensatz dazu bieten Kubernetes-Netzwerkrichtlinien einen Mechanismus zur Steuerung des ein- und ausgehenden Datenverkehrs sowohl innerhalb als auch außerhalb des Clusters. Kubernetes-Netzwerkrichtlinien sollten in Betracht gezogen werden, wenn Ihre Anwendung nur begrenzte Abhängigkeiten von anderen AWS-Services aufweist. Sie können Netzwerkrichtlinien konfigurieren, die Ausgangsregeln auf der Grundlage von CIDR-Bereichen festlegen, um den Zugriff auf AWS-Services zu beschränken, im Gegensatz zu systemeigener AWS-Semantik wie. SGs Sie können Kubernetes-Netzwerkrichtlinien verwenden, um den Netzwerkverkehr zwischen Pods (oft als East/West Verkehr bezeichnet) und zwischen Pods und externen Diensten zu kontrollieren. Kubernetes-Netzwerkrichtlinien werden auf den OSI-Stufen 3 und 4 implementiert.

Amazon EKS ermöglicht Ihnen die Verwendung von Netzwerkrichtlinien-Engines wie Calico und Cilium. Standardmäßig sind die Netzwerkrichtlinien-Engines nicht installiert. Anweisungen zur Einrichtung finden Sie in den jeweiligen Installationsanleitungen. Weitere Informationen zur Verwendung von Netzwerkrichtlinien finden Sie unter Bewährte Methoden für EKS Security. Die Funktion DNS-Hostnamen ist in den Unternehmensversionen von Network Policy Engines verfügbar. Sie könnte nützlich sein, um den Verkehr zwischen Kubernetes Services/Pods und Ressourcen zu kontrollieren, die außerhalb von AWS laufen. Sie können auch die Unterstützung von DNS-Hostnamen für AWS-Services in Betracht ziehen, die standardmäßig keine Sicherheitsgruppen unterstützen.

Kennzeichnen Sie eine einzelne Sicherheitsgruppe zur Verwendung von AWS Loadbalancer Controller

Wenn einem Pod viele Sicherheitsgruppen zugewiesen sind, empfiehlt Amazon EKS, eine einzelne Sicherheitsgruppe mit `kubernetes.io/cluster/$name`Shared oder Owned zu kennzeichnen. Das Tag ermöglicht es dem AWS Loadbalancer Controller, die Regeln von Sicherheitsgruppen zu aktualisieren, um den Datenverkehr an die Pods weiterzuleiten. Wenn einem Pod nur eine Sicherheitsgruppe zugewiesen wird, ist die Zuweisung eines Tags optional. In einer Sicherheitsgruppe festgelegte Berechtigungen sind additiv, daher reicht es aus, eine einzelne Sicherheitsgruppe zu kennzeichnen, damit der Loadbalancer-Controller die Regeln finden und abgleichen kann. Es hilft auch, die von Sicherheitsgruppen definierten Standardkontingente einzuhalten.

Konfigurieren Sie NAT für ausgehenden Datenverkehr

Quell-NAT ist für ausgehenden Datenverkehr von Pods deaktiviert, denen Sicherheitsgruppen zugewiesen sind. Für Pods, die Sicherheitsgruppen verwenden, die Zugriff auf das Internet benötigen, starten Sie Worker-Knoten in privaten Subnetzen, die mit einem NAT-Gateway oder einer NAT-Instance konfiguriert sind, und aktivieren Sie externes SNAT im CNI.

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

Stellen Sie Pods mit Sicherheitsgruppen in privaten Subnetzen bereit

Pods, denen Sicherheitsgruppen zugewiesen sind, müssen auf Knoten ausgeführt werden, die in privaten Subnetzen bereitgestellt werden. Beachten Sie, dass Pods mit zugewiesenen Sicherheitsgruppen, die in öffentlichen Subnetzen bereitgestellt werden, nicht auf das Internet zugreifen können.

Überprüfen Sie die terminationGracePeriodSekunden in der Pod-Spezifikationsdatei

Stellen Sie sicher, `terminationGracePeriodSeconds` dass dieser Wert in Ihrer Pod-Spezifikationsdatei ungleich Null ist (Standard 30 Sekunden). Dies ist wichtig, damit Amazon VPC CNI das Pod-Netzwerk vom Worker-Knoten löschen kann. Wenn der Wert auf Null gesetzt ist, entfernt das CNI-Plugin das Pod-Netzwerk nicht vom Host, und der Branch-ENI wird nicht effektiv bereinigt.

Verwenden von Sicherheitsgruppen für Pods mit Fargate

Sicherheitsgruppen für Pods, die auf Fargate ausgeführt werden, funktionieren sehr ähnlich wie Pods, die auf EC2-Worker-Knoten ausgeführt werden. Beispielsweise müssen Sie die Sicherheitsgruppe erstellen, bevor Sie sie in dem, den SecurityGroupPolicy Sie Ihrem Fargate-Pod zuordnen, referenzieren. Standardmäßig wird die Cluster-Sicherheitsgruppe allen Fargate-Pods zugewiesen, wenn Sie einem Fargate-Pod nicht explizit a SecurityGroupPolicy zuweisen. Der Einfachheit halber möchten Sie möglicherweise die Cluster-Sicherheitsgruppe zu der eines Fagate-Pods hinzufügen, SecurityGroupPolicy andernfalls müssen Sie Ihrer Sicherheitsgruppe die Mindestsicherheitsgruppenregeln hinzufügen. Sie können die Cluster-Sicherheitsgruppe mithilfe der Describe-Cluster-API finden.

```
 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
```

Die Mindestregeln für Sicherheitsgruppen sind hier aufgeführt. Diese Regeln ermöglichen es Fargate Pods, mit Clusterdiensten wie kube-apiserver, kubelet und CoreDNS zu kommunizieren. Sie müssen auch Regeln hinzufügen, um eingehende und ausgehende Verbindungen zu und von Ihrem Fargate-Pod zuzulassen. Dadurch kann Ihr Pod mit anderen Pods oder Ressourcen in Ihrer VPC kommunizieren. Darüber hinaus müssen Sie Regeln für Fargate angeben, um Container-Images aus Amazon ECR oder anderen Container-Registern abzurufen, z. DockerHub Weitere Informationen finden Sie unter AWS-IP-Adressbereiche in der Allgemeinen AWS-Referenz.

Sie können die folgenden Befehle verwenden, um die Sicherheitsgruppen zu finden, die auf einen Fargate-Pod angewendet wurden.

```
kubectl get pod FARGATE_POD -o jsonpath='{.metadata.annotations.fargate\.amazonaws
\.com/pod-sg}{"\n"}'
```

Notieren Sie sich den Befehl enIId aus dem obigen Befehl.

```
aws ec2 describe-network-interfaces --network-interface-ids ENI_ID --query
 'NetworkInterfaces[*].Groups[*]'
```

Bestehende Fargate-Pods müssen gelöscht und neu erstellt werden, damit neue Sicherheitsgruppen angewendet werden können. Der folgende Befehl initiiert beispielsweise die Bereitstellung der Beispiel-App. Um bestimmte Pods zu aktualisieren, können Sie den Namespace und den Bereitstellungsnamen im folgenden Befehl ändern.

```
kubectl rollout restart -n example-ns deployment example-pod
```

Lastausgleich

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Load Balancer empfangen eingehenden Datenverkehr und verteilen ihn auf die Ziele der vorgesehenen Anwendung, die in einem EKS-Cluster gehostet wird. Dies verbessert die Widerstandsfähigkeit der Anwendung. Bei der Bereitstellung in einem EKS-Cluster erstellt und verwaltet der AWS Load Balancer-Controller AWS Elastic Load Balancer für diesen Cluster. Wenn ein Kubernetes-Service dieses Typs erstellt LoadBalancer wird, erstellt der AWS Load Balancer Controller einen Network Load Balancer (NLB), der den empfangenen Verkehr auf Layer 4 des OSI-Modells ausgleicht. Wenn ein Kubernetes Ingress-Objekt erstellt wird, erstellt der AWS Load Balancer Controller dagegen einen Application Load Balancer (ALB), der den Datenverkehr auf Layer 7 des OSI-Modells ausgleicht.

Auswahl des Load Balancer Balancer-Typs

Das AWS Elastic Load Balancing (ELB) -Portfolio unterstützt die folgenden Load Balancer: Application Load Balancers (ALB), Network Load Balancers (NLB), Gateway Load Balancers (GWLB) und Classic Load Balancers (CLB). Dieser Abschnitt mit bewährten Methoden konzentriert sich auf ALB und NLB, die beiden, die für EKS-Cluster am relevantesten sind.

Bei der Auswahl des Load Balancer-Typs stehen vor allem die Workload-Anforderungen im Vordergrund.

Ausführlichere Informationen und als Referenz für alle AWS Load Balancer finden Sie unter Produktvergleiche

Wählen Sie den Application Load Balancer (ALB), wenn Ihre Arbeitslast HTTP/HTTPS ist

Wenn für Workloads ein Lastenausgleich auf Layer 7 des OSI-Modells erforderlich ist, kann der AWS Load Balancer Controller zur Bereitstellung eines ALB verwendet werden. Die Bereitstellung wird im folgenden Abschnitt behandelt. Der ALB wird durch die bereits erwähnte Ingress-Ressource gesteuert und konfiguriert und leitet HTTP- oder HTTPS-Verkehr an verschiedene Pods innerhalb des Clusters weiter. Das ALB bietet Kunden die Flexibilität, den Routing-Algorithmus für den Anwendungsdatenverkehr zu ändern. Der Standard-Routing-Algorithmus ist Round-Robin-Verfahren, wobei der Routing-Algorithmus für die wenigsten ausstehenden Anfragen ebenfalls eine Alternative darstellt.

Wählen Sie den Network Load Balancer (NLB), wenn es sich bei Ihrer Arbeitslast um TCP handelt oder wenn für Ihre Arbeitslast die Quell-IP-Erhaltung von Clients erforderlich ist

Ein Network Load Balancer funktioniert auf der vierten Ebene (Transport) des Open Systems Interconnection (OSI) -Modells. Er ist für TCP- und UDP-basierte Workloads geeignet. Network Load Balancer behält standardmäßig auch die Quell-IP-Adresse der Clients bei, wenn der Datenverkehr dem Pod präsentiert wird.

Wählen Sie den Network Load Balancer (NLB), wenn Ihr Workload DNS nicht nutzen kann

Ein weiterer wichtiger Grund für die Verwendung von NLB ist, dass Ihre Clients DNS nicht nutzen können. In diesem Fall ist der NLB möglicherweise besser für Ihre Arbeitslast geeignet, da sie IPs auf

einem Network Load Balancer statisch sind. Clients IPs wird zwar empfohlen, DNS bei der Auflösung von Domainnamen in IP-Adressen zu verwenden, wenn sie eine Verbindung zu Load Balancers herstellen, aber wenn die Anwendung eines Clients die DNS-Auflösung nicht unterstützt und nur hartcodierte Dateien akzeptiert, ist ein NLB besser geeignet, da sie statisch IPs sind und für die gesamte Lebensdauer des NLB unverändert bleiben.

Bereitstellung von Load Balancern

Nachdem Sie den Load Balancer ermittelt haben, der für Ihre Workloads am besten geeignet ist, haben Kunden eine Reihe von Optionen für die Bereitstellung eines Load Balancers.

Bereitstellung von Load Balancers durch die Bereitstellung des AWS Load Balancer Controllers

Es gibt zwei Hauptmethoden für die Bereitstellung von Load Balancern innerhalb eines EKS-Clusters.

- Nutzung des Service Controllers im AWS Cloud Provider (Legacy)
- Nutzung des AWS Load Balancer Controllers (empfohlen)

Standardmäßig gleicht der Kubernetes Service Controller, auch bekannt als In-Tree-Controller, den Typ der Kubernetes-Service-Ressource ab. LoadBalancer Dieser Controller ist in die AWS Cloud Provider-Komponente integriert, die als Kubernetes Cloud Controller Manager fungiert.

Die Konfiguration des bereitgestellten Elastic Load Balancer wird durch Anmerkungen gesteuert, die dem Kubernetes Service-Manifest hinzugefügt werden müssen. Die Anmerkungen, die vom Service Controller und dem AWS Load Balancer Controller verwendet werden, sind unterschiedlich.

Der Service Controller ist veraltet und erhält derzeit nur kritische Bugfixes. Wenn Sie einen Kubernetes-Service vom Typ erstellen LoadBalancer, erstellt der Service Controller standardmäßig eine AWS-CLB, kann aber auch AWS-NLB erstellen, wenn Sie die richtige Anmerkung verwenden. Es ist erwähnenswert, dass Service Controller keine Kubernetes Ingress-Ressourcen unterstützt und auch nicht unterstützt. IPv6

Wir empfehlen, den AWS Load Balancer Controller in Ihren EKS-Clustern zu verwenden, um Kubernetes Service- und Ingress-Ressourcen abzustimmen. Sie müssen die richtigen Anmerkungen in Ihrem Kubernetes Service- oder Ingress-Manifest verwenden, damit der AWS Load Balancer Controller für den Abstimmungsprozess verantwortlich ist. (anstelle von Service Controller)

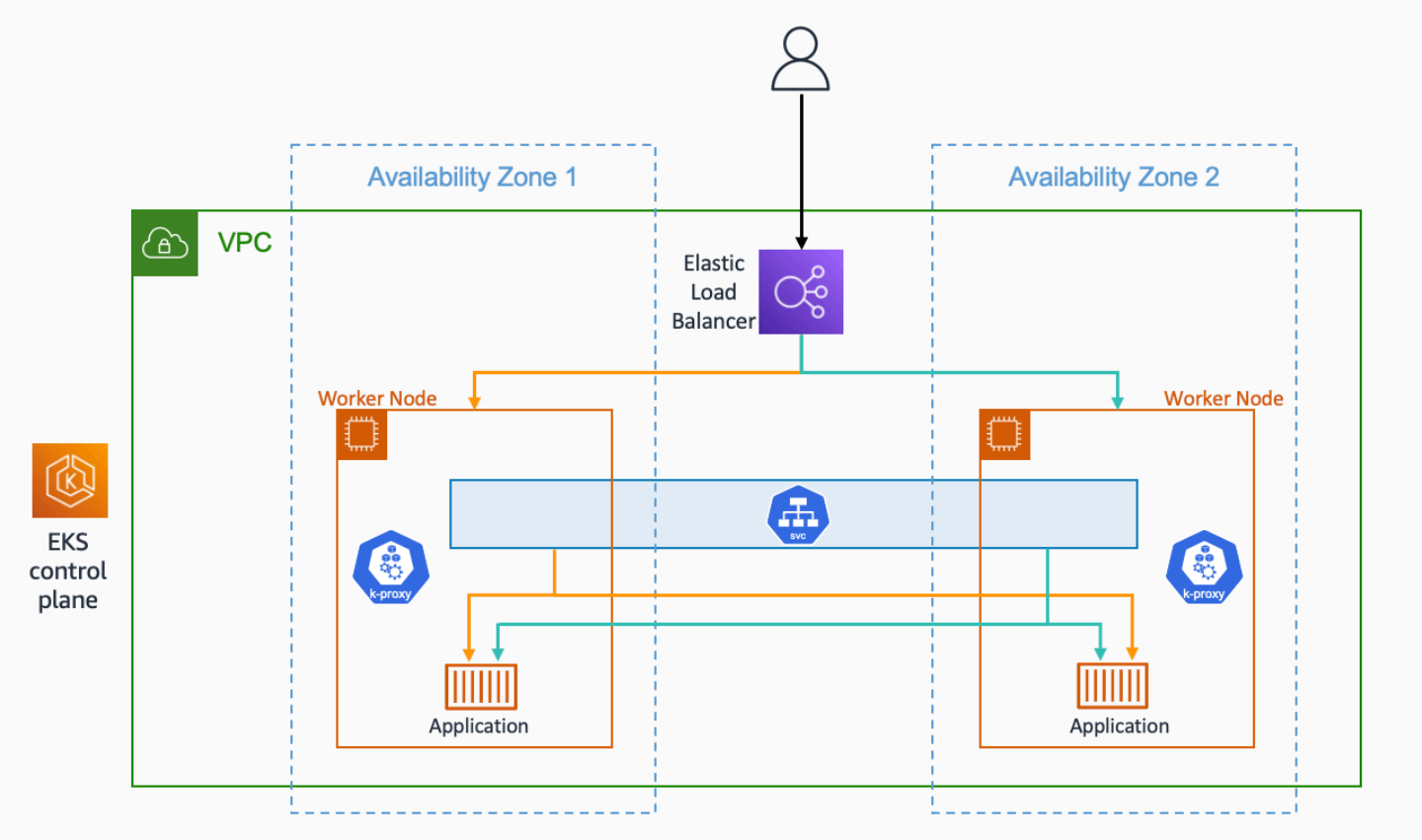
Wenn Sie den EKS Auto Mode verwenden, wird Ihnen der AWS Load Balancer Controller automatisch zur Verfügung gestellt. Es ist keine Installation erforderlich.

Wählen Sie den Load Balancer-Zieltyp

Registrieren Sie Pods mithilfe von IP Target-Type als Ziele

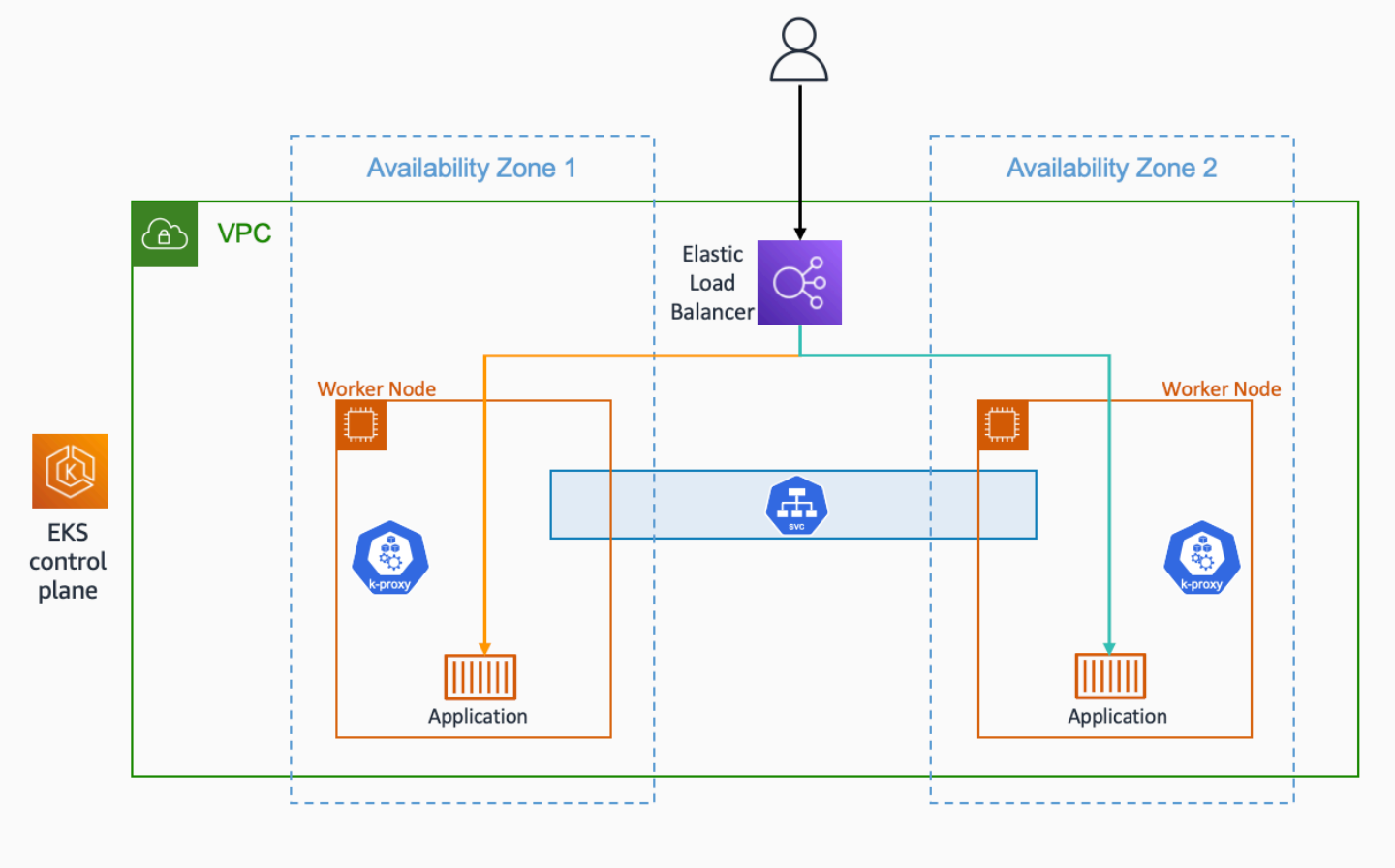
Ein AWS Elastic Load Balancer: Network & Application sendet empfangenen Datenverkehr an registrierte Ziele in einer Zielgruppe. Für einen EKS-Cluster gibt es zwei Arten von Zielen, die Sie in der Zielgruppe registrieren können: Instance und IP. Welcher Zieltyp verwendet wird, hat Auswirkungen darauf, was registriert wird und wie der Datenverkehr vom Load Balancer zum Pod weitergeleitet wird. Standardmäßig registriert der AWS Load Balancer-Controller Ziele mit dem Typ „Instance“. Dieses Ziel ist die IP des Worker Nodes. Dies hat folgende Auswirkungen: NodePort

- Der Datenverkehr vom Load Balancer wird an den Worker Node auf dem weitergeleitet NodePort, dieser wird durch iptables-Regeln (konfiguriert durch den auf dem Knoten laufenden Kube-Proxy) verarbeitet und an den Service auf seiner ClusterIP (immer noch auf dem Knoten) weitergeleitet. Schließlich wählt der Service nach dem Zufallsprinzip einen für ihn registrierten Pod aus und leitet den Datenverkehr an ihn weiter. Dieser Ablauf beinhaltet mehrere Hops und es kann zu zusätzlicher Latenz kommen, insbesondere weil der Service manchmal einen Pod auswählt, der auf einem anderen Worker-Knoten ausgeführt wird, der sich möglicherweise auch in einer anderen AZ befindet.
- Da der Load Balancer den Worker Node als sein Ziel registriert, bedeutet dies, dass seine an das Ziel gesendete Zustandsprüfung nicht direkt vom Pod, sondern vom Worker Node auf dem Pod empfangen wird NodePort und der Health Check-Verkehr dem oben beschriebenen Pfad folgt.
- Die Überwachung und Fehlerbehebung ist komplexer, da der vom Load Balancer weitergeleitete Datenverkehr nicht direkt an die Pods gesendet wird und Sie das auf dem Worker Node empfangene Paket sorgfältig mit der Service ClusterIP und eventuell dem Pod korrelieren müssen, um für eine korrekte Fehlerbehebung einen vollständigen end-to-end Überblick über den Pfad des Pakets zu haben.



Wenn Sie den Zieltyp dagegen wie von uns empfohlen als „IP“ konfigurieren, hat dies folgende Auswirkungen:

- Der Datenverkehr vom Load Balancer wird direkt an den Pod weitergeleitet. Dies vereinfacht den Netzwerkpfad, da die vorherigen zusätzlichen Hops von Worker Nodes und Service Cluster-IP umgangen werden, es reduziert die Latenz, die sonst entstanden wäre, wenn der Service den Datenverkehr an einen Pod in einer anderen AZ weitergeleitet hätte, und schließlich wird die Overhead-Verarbeitung der iptables-Regeln auf den Worker-Knoten entfernt.
- Die Zustandsprüfung des Load Balancers wird direkt vom Pod empfangen und beantwortet. Das bedeutet, dass der Zielstatus „gesund“ oder „ungesund“ eine direkte Darstellung des Zustands des Pods ist.
- Die Überwachung und Fehlerbehebung ist einfacher, und jedes verwendete Tool, das die Paket-IP-Adresse erfasst, zeigt den bidirektionalen Verkehr zwischen dem Load Balancer und dem Pod direkt in seinen Quell- und Zielfeldern an.



Um ein AWS Elastic Load Balancing zu erstellen, das IP-Ziele verwendet, fügen Sie Folgendes hinzu:

- `alb.ingress.kubernetes.io/target-type: ip`Anmerkung zum Manifest Ihres Ingress bei der Konfiguration Ihres Kubernetes Ingress (Application Load Balancer)
- `service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip`Anmerkung zum Manifest Ihres Dienstes bei der Konfiguration Ihres Kubernetes-Dienstes vom Typ LoadBalancer (Network Load Balancer).

Konfiguration von Load Balancer Balancer-Zustandsprüfungen

Kubernetes bietet zwar eigene Zustandsprüfungsmechanismen (im nächsten Abschnitt näher beschrieben), wir empfehlen jedoch, ELB-Zustandsprüfungen als ergänzenden Schutz zu implementieren, der außerhalb der Kubernetes-Kontrollebene funktioniert. Diese unabhängige Ebene setzt die Überwachung Ihrer Anwendung fort, auch wenn:

- Störungen der Kubernetes-Kontrollebene
- Verzögerungen bei der Ausführung der Sonde

- Netzwerkpartitionen zwischen Kubelet und Pod

Für kritische Workloads, die in den oben genannten Szenarien maximale Verfügbarkeit und beschleunigte Wiederherstellung erfordern, bieten ELB-Integritätsprüfungen ein unverzichtbares Sicherheitsnetz, das neben den systemeigenen Mechanismen von Kubernetes funktioniert und diese nicht ersetzt.

Um Integritätsprüfungen auf Ihrem ELB zu konfigurieren und zu optimieren, müssen Sie Anmerkungen in Ihrem Kubernetes Service- oder Ingress-Manifest verwenden, die von Service Controller oder AWS Load Balancer Controller abgeglichen würden.

Verfügbarkeit und Pod-Lebenszyklus

Während eines Anwendungsupgrades müssen Sie sicherstellen, dass Ihre Anwendung immer für die Bearbeitung von Anfragen verfügbar ist, damit es bei Benutzern nicht zu Ausfallzeiten kommt. Eine häufige Herausforderung in diesem Szenario besteht darin, den Verfügbarkeitsstatus Ihrer Workloads zwischen der Kubernetes-Schicht und der Infrastruktur, z. B. externen Load Balancers, zu synchronisieren. In den nächsten Abschnitten werden die bewährten Methoden zur Bewältigung solcher Szenarien beschrieben.

> **Note**
>
> Die folgenden Erläuterungen basieren auf dem, EndpointSlicesda es sich um den empfohlenen Ersatz für die Endpoints in Kubernetes handelt. Die Unterschiede zwischen den beiden sind im Zusammenhang mit den unten behandelten Szenarien vernachlässigbar. Der AWS Load Balancer Controller verwendet standardmäßig Endpoints. Sie können ihn aktivieren, EndpointSlices indem Sie den enable-endpoint-sliceflagauf dem Controller aktivieren.

Verwenden Sie Integritätsprüfungen

Kubernetes führt standardmäßig die Prozessintegritätsprüfung durch, bei der der Kubelet-Prozess auf dem Knoten überprüft, ob der Hauptprozess des Containers läuft oder nicht. Wenn nicht, wird dieser Container standardmäßig neu gestartet. Sie können Kubernetes-Sonden jedoch auch so konfigurieren, dass sie erkennen, wann ein Container-Prozess ausgeführt wird, sich aber in einem Deadlock-Zustand befindet, oder ob eine Anwendung erfolgreich gestartet wurde oder nicht. Die

Sonden können auf den Mechanismen Exec, Grpc, HttpGet und TcpSocket basieren. Je nach Typ und Ergebnis der Sonde kann der Container neu gestartet werden.

Die Reihenfolge der Ereignisse im Prozess der Pod-Erstellung finden Sie im Abschnitt „Pod-Erstellung“ im Anhang weiter unten.

Verwenden Sie Bereitschaftstests

Wenn alle Container innerhalb eines Pods ausgeführt werden, gilt der Pod-Zustand standardmäßig als „Bereit“. Die Anwendung ist jedoch möglicherweise immer noch nicht in der Lage, Kundenanfragen zu verarbeiten. Beispielsweise muss die Anwendung möglicherweise einige Daten oder Konfigurationen von einer externen Ressource abrufen, um Anfragen verarbeiten zu können. In einem solchen Zustand möchten Sie die Anwendung weder beenden noch Anfragen an sie weiterleiten. Mit der Bereitschaftsprüfung können Sie sicherstellen, dass der Pod nicht als „Bereit“ betrachtet wird, d. h., dass er dem EndpointSlice Objekt erst hinzugefügt wird, wenn das Ergebnis der Prüfung vorliegt`success`. Fällt die Sonde dagegen im weiteren Verlauf der Linie aus, wird der Pod vom EndpointSlice Objekt entfernt. Sie können im Pod-Manifest für jeden Container einen Bereitschaftstest konfigurieren. `kubelet`Ein Prozess auf jedem Knoten führt den Bereitschaftstest für die Container auf diesem Knoten aus.

Verwenden Sie Pod Readiness Gates

Ein Aspekt der Readiness Probe ist die Tatsache, dass sie keinen externen Feedback-/Einflussmechanismus enthält. Der Kubelet-Prozess auf dem Knoten führt die Probe aus und definiert den Status der Probe. Dies hat keine Auswirkungen auf die Anfragen zwischen den Microservices selbst in der Kubernetes-Schicht (Ost-West-Verkehr), da der EndpointSlice Controller die Liste der Endpunkte (Pods) immer auf dem neuesten Stand hält. Warum und wann würden Sie dann einen externen Mechanismus benötigen?

Wenn Sie Ihre Anwendungen mithilfe des Kubernetes Service-Typs Load Balancer oder Kubernetes Ingress (für Nord-Süd-Verkehr) verfügbar machen, muss die Pod-Liste IPs für den jeweiligen Kubernetes-Dienst an den externen Infrastruktur-Loadbalancer weitergegeben werden, damit der Load Balancer auch über eine aktuelle Zielliste verfügt. AWS Load Balancer Controller schließt hier die Lücke. Wenn Sie AWS Load Balancer Controller und Leverage verwenden`target group: IP`, erhält auch `kube-proxy` der AWS Load Balancer Controller ein Update (via`watch`) und kommuniziert dann mit der ELB-API, um die Pod-IP zu konfigurieren und mit der Registrierung der Pod-IP als Ziel auf dem ELB zu beginnen.

Wenn Sie ein fortlaufendes Update einer Bereitstellung durchführen, werden neue Pods erstellt, und sobald der Zustand eines neuen Pods „Bereit“ ist, wird ein old/existing Pod beendet. Während dieses Vorgangs wird das EndpointSlice Kubernetes-Objekt schneller aktualisiert als die Zeit, die der ELB benötigt, um die neuen Pods als Ziele zu registrieren, siehe Zielregistrierung. Für kurze Zeit könnte es zu einem Zustandskonflikt zwischen der Kubernetes-Schicht und der Infrastrukturebene kommen, wodurch Client-Anfragen verworfen werden könnten. Während dieser Zeit wären innerhalb der Kubernetes-Schicht neue Pods bereit, Anfragen zu bearbeiten, aber aus ELB-Sicht sind sie es nicht.

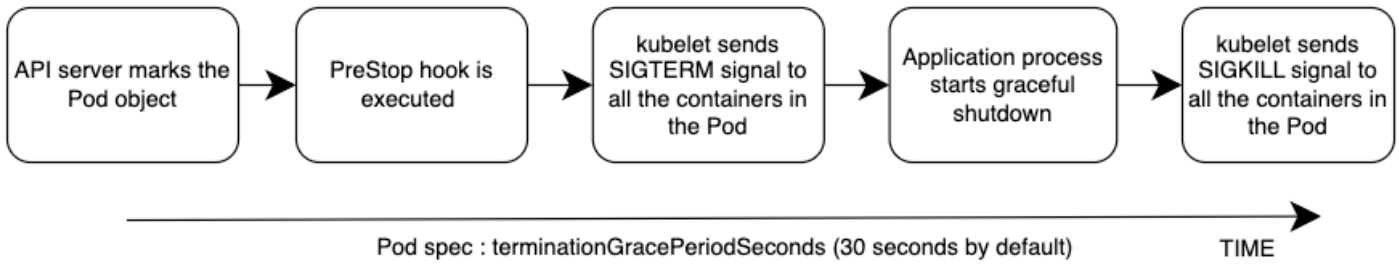
Mit Pod Readiness Gates können Sie zusätzliche Anforderungen definieren, die erfüllt sein müssen, bevor der Pod-Zustand als „Bereit“ angesehen wird. Im Fall von AWS ELB überwacht der AWS Load Balancer Controller den Status des Ziels (des Pods) auf dem AWS-ELB. Sobald die Zielregistrierung abgeschlossen ist und der Status „Healthy“ wird, aktualisiert der Controller den Zustand des Pods auf „Bereit“. Mit diesem Ansatz beeinflussen Sie den Pod-Zustand auf der Grundlage des Status des externen Netzwerks, der dem Zielstatus auf dem AWS-ELB entspricht. Pod Readiness Gates ist in Szenarien mit fortlaufenden Updates von entscheidender Bedeutung, da Sie damit verhindern können, dass das rollierende Update einer Bereitstellung alte Pods beendet, bis der Zielstatus der neu erstellten Pods auf dem AWS-ELB auf „Fehlerfrei“ wechselt.

Anwendungen ordnungsgemäß herunterfahren

Ihre Anwendung sollte auf ein SIGTERM-Signal reagieren, indem sie ihr ordnungsgemäßes Herunterfahren startet, damit es bei den Clients nicht zu Ausfallzeiten kommt. Das bedeutet, dass Ihre Anwendung Säuberungsprozeduren wie das Speichern von Daten, das Schließen von Dateideskriptoren, das Schließen von Datenbankverbindungen, das ordnungsgemäße Abschließen von Anfragen während der Übertragung und das Beenden rechtzeitig ausführen sollte, um die Pod-Terminierungsanforderung zu erfüllen. Sie sollten den Kulanzzeitraum so lang einstellen, dass die Bereinigung abgeschlossen werden kann. Informationen zur Reaktion auf das SIGTERM-Signal finden Sie in den Ressourcen der jeweiligen Programmiersprache, die Sie für Ihre Anwendung verwenden.

Wenn Ihre Anwendung nach Empfang eines SIGTERM-Signals nicht ordnungsgemäß heruntergefahren werden kann oder wenn sie das Signal ignoriert/nicht empfängt, können Sie stattdessen PreStopHook verwenden, um ein ordnungsgemäßes Herunterfahren der Anwendung einzuleiten. Der Prestop-Hook wird unmittelbar vor dem Senden des SIGTERM-Signals ausgeführt und kann beliebige Operationen ausführen, ohne dass diese Operationen im Anwendungscode selbst implementiert werden müssen.

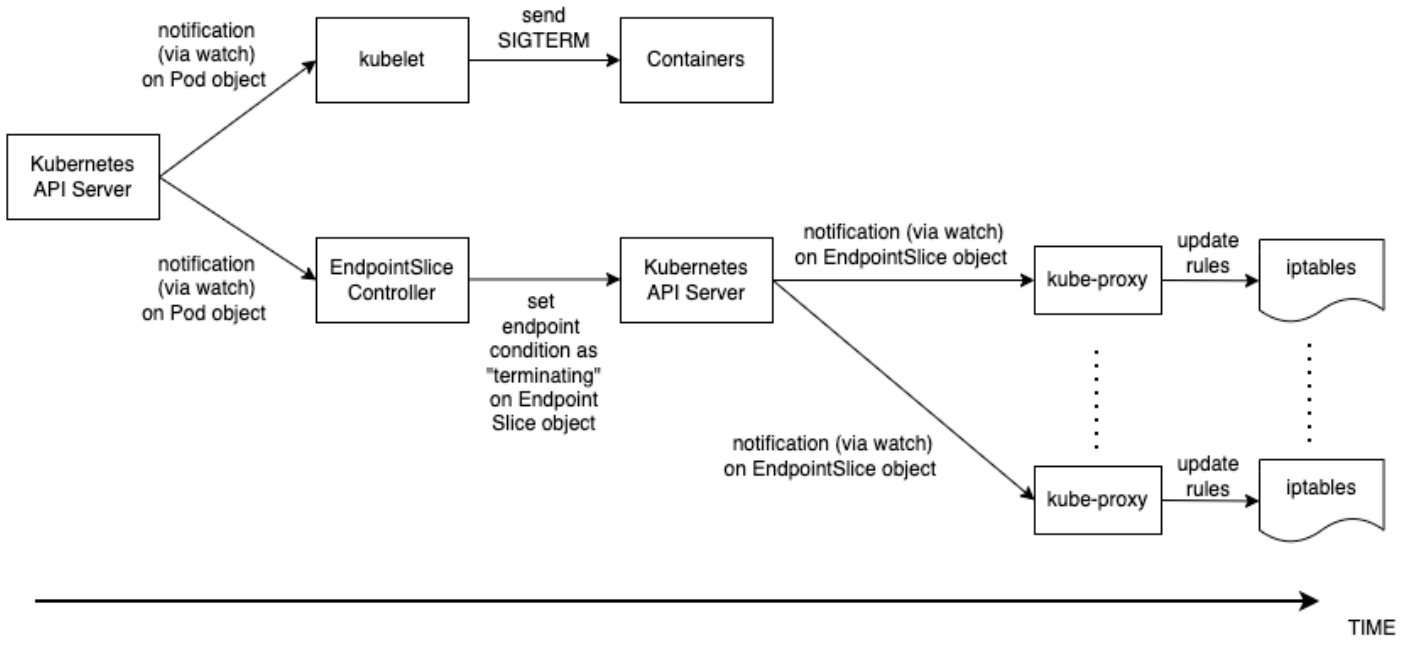
Die gesamte Abfolge der Ereignisse ist in der folgenden Abbildung dargestellt. Hinweis: Unabhängig vom Ergebnis des ordnungsgemäßen Herunterfahrens der Anwendung oder dem Ergebnis des PreStop Hooks werden die Anwendungscontainer am Ende des Kulanzzeitraums mit SIGKILL beendet.



Die Reihenfolge der Ereignisse beim Löschen von Pods finden Sie im Abschnitt zum Löschen von Pods im Anhang weiter unten.

Behandeln Sie die Kundenanfragen mit Bedacht

Die Reihenfolge der Ereignisse beim Löschen von Pods unterscheidet sich von der bei der Pod-Erstellung. Wenn ein Pod erstellt wird, wird die Pod-IP in der Kubernetes-API `kubelet` aktualisiert, und erst dann wird das EndpointSlice Objekt aktualisiert. Wenn andererseits ein Pod beendet wird, benachrichtigt die Kubernetes-API sowohl das Kubelet als auch den Controller gleichzeitig. EndpointSlice Schauen Sie sich das folgende Diagramm, das die Abfolge der Ereignisse zeigt, sorgfältig an.



Die Art und Weise, wie sich der Status vom API-Server bis hin zu den oben erläuterten iptables-Regeln auf den Knoten verbreitet, schafft eine interessante Rennbedingung. Weil die Wahrscheinlichkeit hoch ist, dass der Container das SIGKILL-Signal viel früher empfängt, als der Kube-Proxy auf jedem Knoten die lokalen iptables-Regeln aktualisiert. In einem solchen Fall sind zwei erwähnenswerte Szenarien zu nennen:

- Wenn Ihre Anwendung die laufenden Anfragen und Verbindungen nach Erhalt von SIGTERM sofort und unverblümt verwirft, bedeutet das, dass den Kunden überall 50-fache Fehler angezeigt werden.
- Selbst wenn Ihre Anwendung sicherstellt, dass alle laufenden Anfragen und Verbindungen nach Erhalt von SIGTERM vollständig bearbeitet werden, würden während der Übergangszeit immer noch neue Client-Anfragen an den Anwendungscontainer gesendet werden, da die iptables-Regeln möglicherweise noch nicht aktualisiert wurden. Bis die Bereinigungsprozedur den Server-Socket auf dem Container schließt, führen diese neuen Anfragen zu neuen Verbindungen. Wenn die Karenzzeit endet, werden die Verbindungen, die nach dem SIGTERM hergestellt wurden, zu diesem Zeitpunkt bedingungslos unterbrochen, da SIGKILL gesendet wurde.

Wenn Sie den Kulanzzeitraum in der Pod-Spezifikation lang genug festlegen, kann dieses Problem zwar behoben werden, aber abhängig von der Übertragungsverzögerung und der Anzahl der tatsächlichen Client-Anfragen ist es schwierig, abzuschätzen, wie lange die Anwendung benötigt, um die Verbindungen ordnungsgemäß zu schließen. Daher ist der nicht so perfekte, aber praktikabelste Ansatz hier, einen PreStop Hook zu verwenden, um das SIGTERM-Signal zu verzögern, bis die iptables-Regeln aktualisiert sind, um sicherzustellen, dass keine neuen Client-Anfragen an die Anwendung gesendet werden, sondern nur bestehende Verbindungen weitergeführt werden. PreStop Hook kann ein einfacher Exec-Handler sein, wie z. `sleep 10`

Das Verhalten und die oben erwähnte Empfehlung gelten gleichermaßen, wenn Sie Ihre Anwendungen mit dem Load Balancer Typ Kubernetes Service oder Kubernetes Ingress (für Nord-Süd-Verkehr) mithilfe von AWS Load Balancer Controller und Leverage verfügbar machen. `target group: IP` Denn genau wie `kube-proxy` der AWS Load Balancer Controller erhält auch er ein Update (per Watch) für das EndpointSlice Objekt und kommuniziert dann mit der ELB-API, um mit der Abmeldung der Pod-IP vom ELB zu beginnen. Abhängig von der Auslastung der Kubernetes-API oder der ELB-API kann dies jedoch auch einige Zeit dauern, und der SIGTERM wurde möglicherweise bereits vor langer Zeit an die Anwendung gesendet. Sobald der ELB mit der Deregistrierung des Ziels beginnt, sendet er keine Anfragen mehr an dieses Ziel, sodass die Anwendung keine neuen Anfragen erhält. Außerdem startet der ELB eine Abmeldeverzögerung, die standardmäßig 300 Sekunden beträgt. Während des Abmeldevorgangs wartet der ELB im Grunde

daraufdraining, dass die Verbindungen zu diesem Ziel nicht mehr verfügbar sind. requests/existing Sobald die Verzögerung bei der Abmeldung abgelaufen ist, ist das Ziel ungenutzt und alle Anfragen während der Übertragung an dieses Ziel werden gewaltsam gelöscht.

Verwenden Sie das Budget für Pod-Unterbrechungen

Konfigurieren Sie ein Pod Disruption Budget (PDB) für Ihre Anwendungen. PDBlimits die Anzahl der Pods einer replizierten Anwendung, die aufgrund freiwilliger Unterbrechungen gleichzeitig ausgefallen sind. Es stellt sicher, dass eine Mindestanzahl oder ein Mindestprozentsatz von Pods in einer StatefulSet Oder-Bereitstellung verfügbar bleibt. Eine quorumbasierte Anwendung muss beispielsweise sicherstellen, dass die Anzahl der ausgeführten Replikate niemals unter die für ein Quorum erforderliche Anzahl sinkt. Oder ein Web-Frontend könnte dafür sorgen, dass die Anzahl der Replikate, die ausgelastet sind, niemals unter einen bestimmten Prozentsatz der Gesamtzahl fällt. PDB schützt die Anwendung vor Aktionen wie der Entleerung von Knoten oder der Einführung neuer Versionen von Bereitstellungen. Beachten Sie, dass PDBs die Anwendung nicht vor ungewollten Störungen wie einem Ausfall des Knotenbetriebssystems oder dem Verlust der Netzwerkkonnektivität schützen. Weitere Informationen finden Sie in der Dokumentation zur Angabe eines Unterbrechungsbudgets für Ihre Anwendung in Kubernetes.

Referenzen

- KubeCon Sitzung Europa 2019 — Bereit? Ein tiefer Einblick in Pod Readiness Gates for Service Health
- Buch — Kubernetes in Aktion
- AWS-Blog — So skalieren Sie Ihre Anwendung mit ALB auf EKS schnell (ohne Datenverkehrsverlust)

Anhang

Pod-Erstellung

Es ist unerlässlich, die Reihenfolge der Ereignisse in einem Szenario zu verstehen, in dem ein Pod bereitgestellt wird healthy/ready und dann Kundenanfragen entgegennimmt und verarbeitet. Lassen Sie uns über die Reihenfolge der Ereignisse sprechen.

1. Ein Pod wird auf der Kubernetes-Steuerebene erstellt (d. h. durch einen kubectl-Befehl, ein Deployment-Update oder eine Skalierungsaktion).
2. kube-schedulerweist den Pod einem Knoten im Cluster zu.

3. Der Kubelet-Prozess, der auf dem zugewiesenen Knoten ausgeführt wird, empfängt das Update (viawatch) und kommuniziert mit der Container-Laufzeit, um die in der Pod-Spezifikation definierten Container zu starten.
4. Wenn die Container ausgeführt werden, aktualisiert das Kubelet den Pod-Zustand wie Ready im Pod-Objekt in der Kubernetes-API.
5. Der EndpointSliceController empfängt das Pod-Zustandsupdate (viawatch) und fügt den Pod IP/Port als neuen Endpunkt zum EndpointSliceObjekt (Pod-Liste IPs) des jeweiligen Kubernetes-Dienstes hinzu.
6. Der Kube-Proxy-Prozess auf jedem Knoten empfängt das Update (viawatch) für das EndpointSlice Objekt und aktualisiert dann die iptables-Regeln auf jedem Knoten mit der neuen Pod-IP/dem neuen Pod-Port.

Pod-Löschen

Genau wie bei der Pod-Erstellung ist es unerlässlich, die Reihenfolge der Ereignisse beim Pod-Löschen zu verstehen. Lassen Sie uns über die Reihenfolge der Ereignisse sprechen.

1. Eine Anfrage zum Löschen eines Pods wird an den Kubernetes-API-Server gesendet (d. h. durch einen kubectl Befehl, ein Deployment-Update oder eine Skalierungsaktion).
2. Der Kubernetes-API-Server startet eine Übergangszeit, die standardmäßig 30 Sekunden beträgt, indem er das Feld DeletionTimeStamp im Pod-Objekt festlegt. (Die Nachfrist kann in der Pod-Spezifikation über konfiguriert werden.) terminationGracePeriodSeconds
3. Der auf dem Knoten ausgeführte kubelet Prozess empfängt das Update (über Watch) für das Pod-Objekt und sendet ein SIGTERM-Signal an die Prozess-ID 1 (PID 1) in jedem Container in diesem Pod. Es beobachtet dann dieterminationGracePeriodSeconds.
4. Der EndpointSliceController empfängt außerdem das Update (viawatch) aus Schritt 2 und setzt die Endpunktbedingung im EndpointSliceObjekt (Pod-Liste IPs) des jeweiligen Kubernetes-Dienstes auf „beendend“.
5. Der Kube-Proxy-Prozess auf jedem Knoten empfängt das Update (viawatch) für das EndpointSlice Objekt, dann werden die iptables-Regeln auf jedem Knoten vom Kube-Proxy aktualisiert, sodass keine Kundenanfragen mehr an den Pod weitergeleitet werden.
6. Wenn der terminationGracePeriodSeconds abläuft, kubelet sendet er ein SIGKILL-Signal an den übergeordneten Prozess jedes Containers im Pod und beendet ihn gewaltsam.
7. TheEndpointSliceDer Controller entfernt den Endpunkt aus dem Objekt. EndpointSlice
8. Der API-Server löscht das Pod-Objekt.

Überwachung von EKS-Workloads im Hinblick auf Netzwerkleistungsprobleme

Überwachung des CoreDNS-Datenverkehrs auf DNS-Throttling-Probleme

Beim Ausführen von DNS-intensiven Workloads kann es manchmal aufgrund von DNS-Drosselung zu zeitweiligen CoreDNS-Ausfällen kommen. Dies kann sich auf Anwendungen auswirken, bei denen gelegentlich Fehler auftreten können. UnknownHostException

Das Deployment for CoreDNS verfügt über eine Anti-Affinitätsrichtlinie, die den Kubernetes-Scheduler anweist, Instanzen von CoreDNS auf separaten Worker-Knoten im Cluster auszuführen, d. h. es sollte vermieden werden, Replikate auf demselben Worker-Knoten gemeinsam zu platzieren. Dadurch wird die Anzahl der DNS-Abfragen pro Netzwerkschnittstelle effektiv reduziert, da der Datenverkehr von jedem Replikat über eine andere ENI geleitet wird. Wenn Sie feststellen, dass DNS-Abfragen aufgrund der Obergrenze von 1024 Paketen pro Sekunde gedrosselt werden, können Sie 1) versuchen, die Anzahl der CoreDNS-Replikate zu erhöhen oder 2) implementieren. NodeLocal DNSCache Weitere Informationen finden Sie unter CoreDNS-Metriken überwachen.

Challenge

- Der Paketverlust erfolgt innerhalb von Sekunden, und es kann für uns schwierig sein, diese Muster ordnungsgemäß zu überwachen, um festzustellen, ob eine DNS-Drosselung tatsächlich stattfindet.
- DNS-Abfragen werden auf der Ebene der elastic network interface gedrosselt. Daher werden gedrosselte Abfragen nicht in der Abfrageprotokollierung angezeigt.
- Flow-Protokolle erfassen nicht alle Arten von IP-Datenverkehr. z. B. Datenverkehr von Instances, die den Amazon-DNS-Server kontaktieren. Wenn Sie Ihren eigenen DNS-Server verwenden, wird der gesamte Datenverkehr zu diesem DNS-Server protokolliert

Lösung

Eine einfache Möglichkeit, Probleme mit der DNS-Drosselung in Worker-Knoten zu identifizieren, ist die Erfassung `linklocal_allowance_exceeded` von Metriken. Der Wert linklocal_allowance_exceeded gibt die Anzahl der Pakete an, die verworfen wurden, weil der PPS-Wert des Datenverkehrs zu lokalen Proxydiensten den Höchstwert für die Netzwerkschnittstelle überschritten hat. Dies wirkt sich auf den Datenverkehr zum DNS-Dienst, zum Instance Metadata Service und zum Amazon Time Sync Service aus. Anstatt dieses Ereignis in Echtzeit zu verfolgen,

können wir diese Metrik auch an Amazon Managed Service for Prometheus streamen und sie in Amazon Managed Grafana visualisieren lassen

Überwachung von Verzögerungen bei DNS-Abfragen mithilfe von Conntrack-Metriken

Eine weitere Metrik, die bei der Überwachung der CoreDNS-Drosselung/Abfrageverzögerung helfen kann, sind und. `conntrack_allowance_available` `conntrack_allowance_exceeded` Verbindungsfehler, die durch die Überschreitung der erlaubten Verbindungen verursacht werden, können größere Auswirkungen haben als solche, die auf die Überschreitung anderer Zulagen zurückzuführen sind. Wenn man sich bei der Übertragung von Daten auf TCP verlässt, werden Pakete, die aufgrund der Überschreitung der EC2-Instance-Netzwerkzugriffsrechte, wie Bandbreite, PPS usw., in die Warteschlange gestellt oder verworfen werden, dank der TCP-Funktionen zur Überlastungskontrolle in der Regel problemlos verarbeitet. Betroffene Datenflüsse werden verlangsamt und verlorene Pakete werden erneut übertragen. Überschreitet eine Instance jedoch die zulässige Anzahl der verfolgten Verbindungen, können keine neuen Verbindungen hergestellt werden, bis einige der vorhandenen geschlossen wurden, um Platz für neue Verbindungen zu schaffen.

`conntrack_allowance_available`und `conntrack_allowance_exceeded` unterstützt Kunden bei der Überwachung der Anzahl der verfolgten Verbindungen, die von Instanz zu Instanz unterschiedlich ist. Diese Netzwerkleistungskennzahlen geben Kunden Einblick in die Anzahl der Pakete, die in die Warteschlange gestellt oder verworfen wurden, wenn die Netzwerkberechtigungen einer Instance wie Netzwerkbandbreite Packets-Per-Second (PPS), verfolgte Verbindungen und Link-Local-Servicezugriff (Amazon DNS, Instance Meta Data Service, Amazon Time Sync) überschritten werden.

`conntrack_allowance_available`ist die Anzahl der verfolgten Verbindungen, die von der Instance hergestellt werden können, bevor die zulässige Anzahl von verfolgten Verbindungen für diesen Instance-Typ erreicht wird (wird nur für Nitro-Instances unterstützt). `conntrack_allowance_exceeded`ist die Anzahl der Pakete, die verworfen wurden, weil die Verbindungsverfolgung das Maximum für die Instanz überschritten hat und keine neuen Verbindungen hergestellt werden konnten.

Andere wichtige Kennzahlen zur Netzwerkleistung

Zu den weiteren wichtigen Kennzahlen zur Netzwerkleistung gehören:

`bw_in_allowance_exceeded`(der ideale Wert der Metrik sollte Null sein) ist die Anzahl der Pakete, die in der Warteschlange and/or verworfen wurden, weil die Gesamtbandbreite für eingehende Nachrichten das Maximum für die Instanz überschritten hat

`bw_out_allowance_exceeded`(der ideale Wert der Metrik sollte Null sein) ist die Anzahl der Pakete, die in der Warteschlange and/or verworfen wurden, weil die Gesamtbandbreite für ausgehende Nachrichten das Maximum für die Instance überschritten hat

`pps_allowance_exceeded`(der ideale Wert der Metrik sollte Null sein) ist die Anzahl der Pakete, die in der Warteschlange and/or verworfen wurden, weil das bidirektionale PPS das Maximum für die Instanz überschritten hat

Erfassung der Metriken zur Überwachung von Workloads im Hinblick auf Netzwerkleistungsprobleme

Der Elastic Network Adapter (ENA) -Treiber veröffentlicht die oben erläuterten Netzwerkleistungsmetriken aus den Instances, in denen sie aktiviert sind. Alle Netzwerkleistungsmetriken können CloudWatch mithilfe des CloudWatch Agenten veröffentlicht werden. Weitere Informationen finden Sie im Blog.

Lassen Sie uns nun die oben besprochenen Kennzahlen erfassen, sie in Amazon Managed Service for Prometheus speichern und mit Amazon Managed Grafana visualisieren.

Voraussetzungen

- ethtool — Stellen Sie sicher, dass auf den Worker-Knoten Ethtool installiert ist
- Ein in Ihrem AWS-Konto konfigurierter AMP-Workspace. Anweisungen finden Sie im AMP-Benutzerhandbuch unter Einen Workspace erstellen.
- Von Amazon verwalteter Grafana-Arbeitsbereich

Bereitstellung von Prometheus Node Exporter

Um die Ethtool-Metriken für Worker-Knoten zu sammeln, müssen wir einen Collector einsetzen, der die Metriken sammelt und sie im Prometheus-Format bereitstellt. Die folgenden Befehle fügen das `prometheus-community` Helm-Repository hinzu und installieren das `prometheus-node-exporter` Diagramm bei aktiviertem Collector. `ethtool`

```
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'"
```

Stellen Sie den ADOT-Collector bereit, um die Ethtool-Metriken zu erfassen und im Amazon Managed Service for Prometheus Workspace zu speichern

Jeder Cluster, für den Sie AWS Distro for OpenTelemetry (ADOT) installieren, muss über diese Rolle verfügen, um Ihrem AWS-Servicekonto Berechtigungen zum Speichern von Metriken in Amazon Managed Service for Prometheus zu gewähren. Befolgen Sie diese Schritte, um Ihre IAM-Rolle mit Ihrem Amazon-EKS-Servicekonto mit IRSA zu erstellen und zuzuordnen:

```
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
```

Lassen Sie uns den ADOT-Collector einsetzen, um die Metriken aus dem Prometheus-Ethtool-Exporter zu extrahieren und sie in Amazon Managed Service for Prometheus zu speichern

Das folgende Verfahren verwendet eine YAML-Beispieldatei mit Deployment als Moduswert. Dies ist der Standardmodus und stellt den ADOT Collector wie eine eigenständige Anwendung bereit. Diese Konfiguration empfängt OTLP-Metriken aus der Beispielanwendung und Amazon Managed Service for Prometheus-Metriken, die aus Pods auf dem Cluster gescrapt wurden.

```
curl -o collector-config-amp.yaml https://raw.githubusercontent.com/aws-observability/
aws-otel-community/master/sample-configs/operator/collector-config-amp.yaml
```

Ersetzen Sie in collector-config-amp .yaml Folgendes durch Ihre eigenen Werte:

- Modus: Bereitstellung
- ServiceAccount: ein Punktesammler
- Endpunkt: <YOUR_REMOTE_WRITE_ENDPOINT>

- Region: <YOUR_ >AWS_REGION
- Name: adot-Collector

```
kubectl apply -f collector-config-amp.yaml
```

Sobald der Adot Collector bereitgestellt ist, werden die Metriken erfolgreich in Amazon Prometheus gespeichert

Konfigurieren Sie den Alert Manager in Amazon Managed Service für Prometheus, um Benachrichtigungen zu senden

Sie können den Alert Manager in Amazon Managed Service for Prometheus verwenden, um Warnungsregeln für kritische Alerts einzurichten. Anschließend können Sie Benachrichtigungen an ein Amazon SNS SNS-Thema senden. Lassen Sie uns die Aufzeichnungs- und Alarmregeln konfigurieren, um zu überprüfen, ob die bisher besprochenen Metriken vorliegen.

Wir werden den ACK Controller für Amazon Managed Service for Prometheus verwenden, um die Warn- und Aufzeichnungsregeln bereitzustellen.

Lassen Sie uns den ACL-Controller für den Service Amazon Managed Service for Prometheus bereitstellen:

```
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
```

Führen Sie den Befehl aus und nach ein paar Augenblicken sollten Sie die folgende Meldung sehen:

```
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.
```

Lassen Sie uns nun eine Yaml-Datei für die Bereitstellung der Alert Manager-Definition und der Regelgruppen erstellen. Speichern Sie die folgende Datei als `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"
```

Ersetzen Sie Ihre WORKSPACE-ID durch die Workspace-ID des Workspace, den Sie verwenden.

Lassen Sie uns nun die Alert Manager-Definition konfigurieren. Speichern Sie die folgende Datei als `alertmanager.yaml`

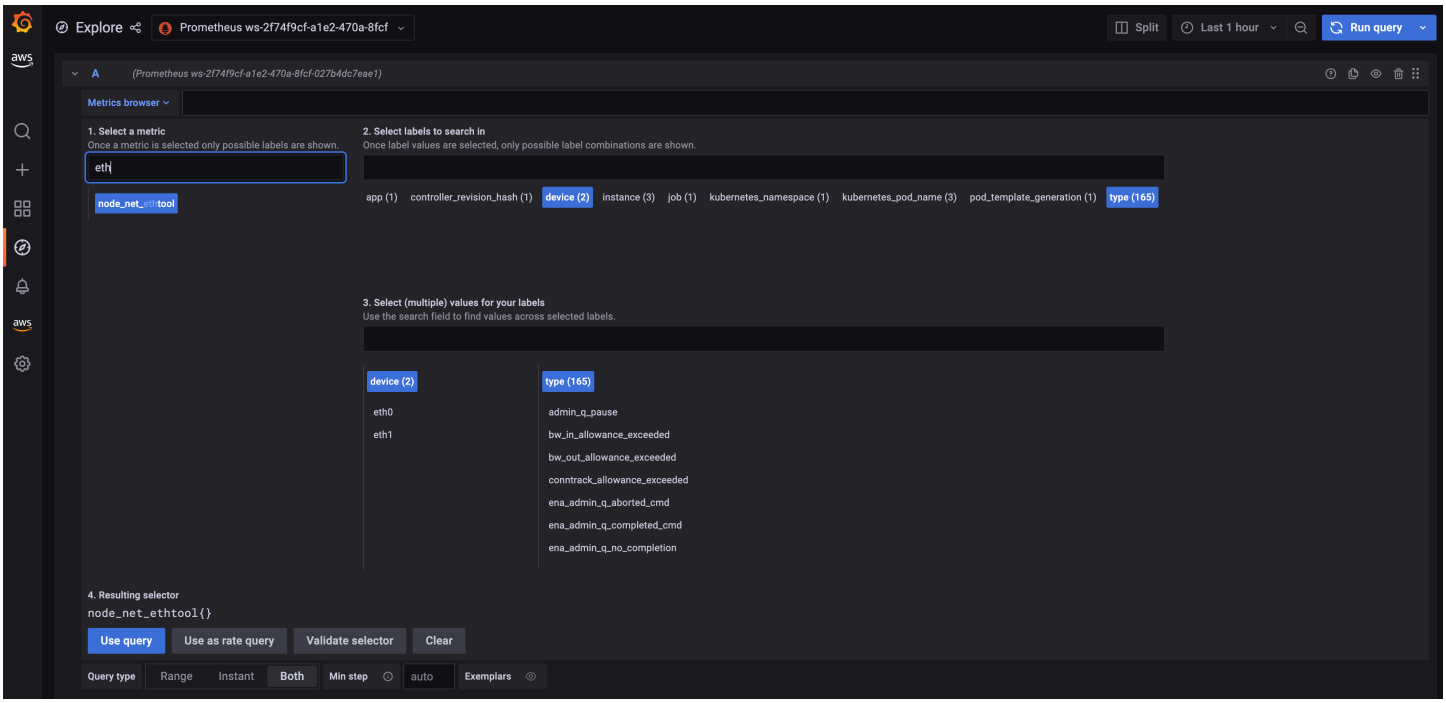
```
apiVersion: prometheusservice.services.k8s.aws/v1alpha1
kind: AlertManagerDefinition
metadata:
  name: alert-manager
spec:
  workspaceID: <Your WORKSPACE-ID >
  configuration: |
    alertmanager_config: |
      route:
         receiver: default_receiver
       receivers:
       - name: default_receiver
          sns_configs:
          - topic_arn: TOPIC-ARN
            sigv4:
              region: REGION
            message: |
              alert_type: {{ .CommonLabels.alertname }}
              event_type: {{ .CommonLabels.event_type }}
```

Ersetzen Sie Ihre WORKSPACE-ID durch die Workspace-ID des neuen Workspace, TOPIC-ARN durch den ARN eines Amazon Simple Notification Service-Themas, an das Sie die Benachrichtigungen senden möchten, und REGION durch die aktuelle Region des Workloads. Vergewissern Sie sich, dass Ihr Workspace berechtigt ist, Nachrichten an Amazon SNS zu senden.

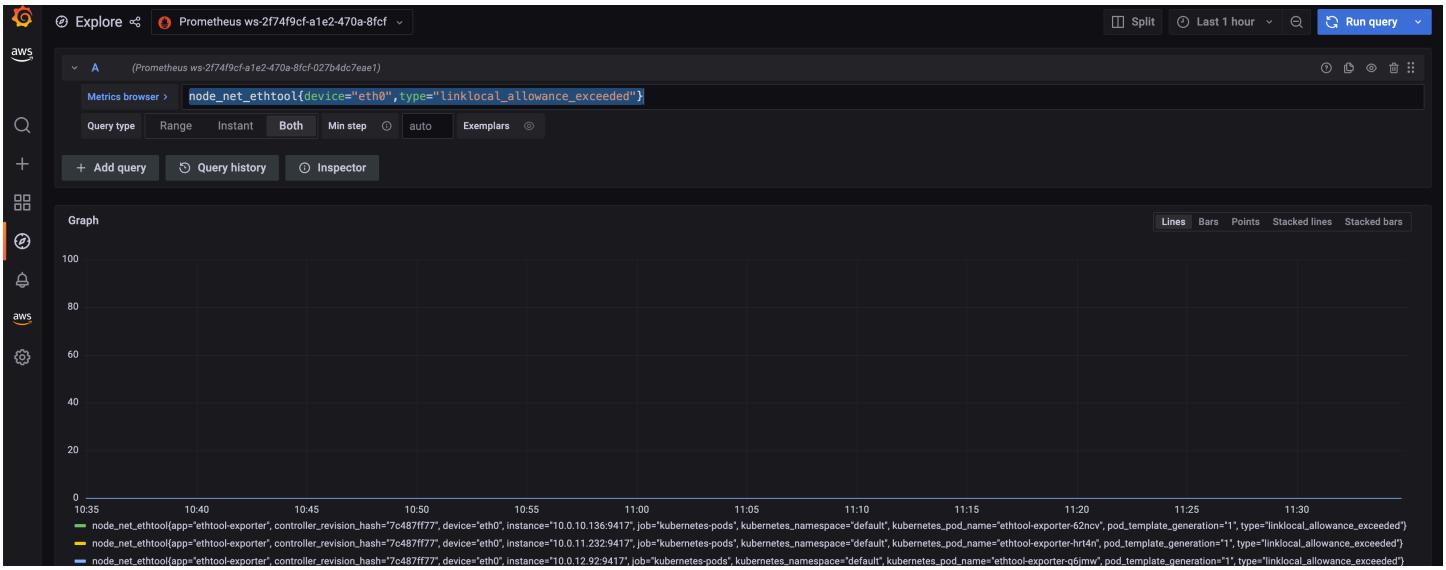
Visualisieren Sie Ethtool-Metriken in Amazon Managed Grafana

Lassen Sie uns die Metriken in Amazon Managed Grafana visualisieren und ein Dashboard erstellen. Konfigurieren Sie den Amazon Managed Service für Prometheus als Datenquelle in der Amazon Managed Grafana-Konsole. Anweisungen finden Sie unter Amazon Prometheus als Datenquelle hinzufügen

Lassen Sie uns jetzt die Metriken in Amazon Managed Grafana untersuchen: Klicken Sie auf die Schaltfläche „Erkunden“ und suchen Sie nach ethtool:



Lassen Sie uns mithilfe der Abfrage ein Dashboard für die Metrik linklocal_allowance_exceeded erstellen.
`rate(node_net_ethtool{device="eth0",type="linklocal_allowance_exceeded"}[30s])` Es wird zum folgenden Dashboard führen.

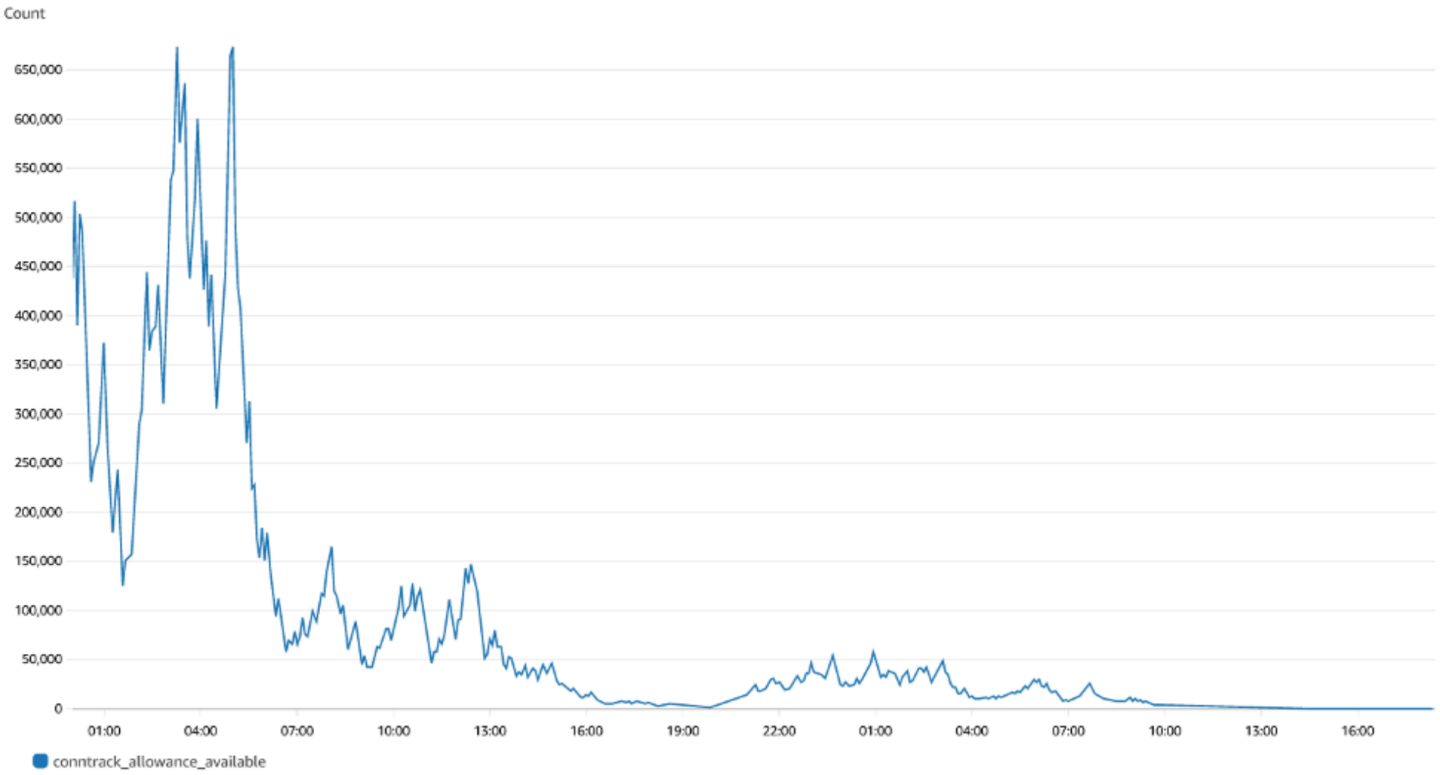


Wir können deutlich erkennen, dass keine Pakete verworfen wurden, da der Wert Null ist.

Lassen Sie uns mithilfe der Abfrage ein Dashboard für die Metrik conntrack_allowance_exceeded erstellen. `rate(node_net_ethtool{device="eth0",type="conntrack_allowance_exceeded"}[30s])` Es wird das folgende Dashboard ergeben.



Die Metrik `conntrack_allowance_exceeded` kann visualisiert werden CloudWatch, vorausgesetzt, Sie führen einen Cloudwatch-Agenten aus, wie hier beschrieben. Das resultierende Dashboard CloudWatch wird wie folgt aussehen:



Wir können deutlich erkennen, dass keine Pakete verworfen wurden, da der Wert Null ist. Wenn Sie Nitro-basierte Instances verwenden, können Sie ein ähnliches Dashboard für Ihre EC2-Instance erstellen `conntrack_allowance_available` und die Verbindungen in Ihrer EC2-Instance proaktiv überwachen. Sie können dies weiter ausbauen, indem Sie Benachrichtigungen in Amazon Managed Grafana so konfigurieren, dass Benachrichtigungen an Slack, SNS, Pagerduty usw. gesendet werden.

Kube-Proxy im IPVS-Modus ausführen

EKS im IP Virtual Server (IPVS) -Modus behebt das Problem der Netzwerklatenz, das häufig auftritt, wenn große Cluster mit über 1.000 Diensten ausgeführt werden, wobei Kube-Proxy im Legacy-Iptables-Modus ausgeführt wird. Dieses Leistungsproblem ist das Ergebnis der sequentiellen Verarbeitung der iptables-Paketfilterregeln für jedes Paket. Dieses Latenzproblem wurde in nftables, dem Nachfolger von iptables, behoben. Zum Zeitpunkt der Erstellung dieses Artikels befindet sich Kube-Proxy jedoch noch in der Entwicklung, um Nftables zu nutzen. Um dieses Problem zu umgehen, können Sie Ihren Cluster so konfigurieren, dass er im IPVS-Modus ausgeführt wird`kube-proxy`.

-Übersicht

IPVS, seit Kubernetes Version 1.11 allgemein verfügbar, verwendet Hashtabellen statt linearer Suche, um Pakete zu verarbeiten, was Clustern mit Tausenden von Knoten und Diensten Effizienz bietet. IPVS wurde für den Lastenausgleich entwickelt und ist damit eine geeignete Lösung für Leistungsprobleme bei Kubernetes-Netzwerken.

IPVS bietet mehrere Optionen für die Verteilung des Datenverkehrs auf Backend-Pods. Detaillierte Informationen zu den einzelnen Optionen finden Sie in der offiziellen Kubernetes-Dokumentation. Eine einfache Liste finden Sie unten. Round Robin und Least Connections gehören zu den beliebtesten Optionen für IPVS-Lastenausgleichsoptionen in 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)
```

Implementierung

Es sind nur wenige Schritte erforderlich, um IPVS in Ihrem EKS-Cluster zu aktivieren. Als Erstes müssen Sie sicherstellen, dass auf Ihren EKS-Worker-Knoten-Images das Linux Virtual `ipvsadm` Server-Verwaltungspaket installiert ist. Um dieses Paket auf einem Fedora-basierten Image wie Amazon Linux 2023 zu installieren, können Sie den folgenden Befehl auf der Worker-Node-Instance ausführen.

```
sudo dnf install -y ipvsadm
```

Auf einem Debian-basierten Image wie Ubuntu würde der Installationsbefehl so aussehen.

```
sudo apt-get install ipvsadm
```

Als Nächstes müssen Sie die Kernelmodule für die oben aufgeführten IPVS-Konfigurationsoptionen laden. Wir empfehlen, diese Module in eine Datei innerhalb des `/etc/modules-load.d/` Verzeichnisses zu schreiben, damit sie einen Neustart überstehen.

```
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'
```

Sie können den folgenden Befehl ausführen, um diese Module auf einen Computer zu laden, der bereits läuft.

```
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**
>
> Es wird dringend empfohlen, diese Worker-Node-Schritte als Teil des Bootstrapping-Prozesses Ihres Worker-Knotens per Benutzerdatenskript oder in beliebigen Build-Skripten auszuführen, die zum Erstellen eines benutzerdefinierten Worker-Knoten-AMI ausgeführt werden.

Als Nächstes konfigurieren Sie Ihre Cluster so, dass sie im `kube-proxy` DaemonSet IPVS-Modus ausgeführt werden. Dazu setzen Sie `kube-proxy` `mode` auf eine der oben aufgeführten Lastenausgleichsoptionen auf `ipvs` und auf, zum Beispiel: `rr` für Round Robin. `ipvs scheduler`

> **Warning**
>
> Dabei handelt es sich um eine störende Änderung, die außerhalb der Geschäftszeiten durchgeführt werden sollte. Wir empfehlen, diese Änderungen bei der ersten Erstellung des EKS-Clusters vorzunehmen, um die Auswirkungen zu minimieren.

Sie können einen AWS-CLI-Befehl ausgeben, um IPVS zu aktivieren, indem Sie das `kube-proxy` EKS-Add-on aktualisieren.

```
aws eks update-addon --cluster-name $CLUSTER_NAME --addon-name kube-proxy \
  --configuration-values '{"ipvs": {"scheduler": "rr"}, "mode": "ipvs"}' \
  --resolve-conflicts OVERWRITE
```

Oder Sie können dies tun, indem Sie die `kube-proxy-config` ConfigMap in Ihrem Cluster ändern.

```
kubectl -n kube-system edit cm kube-proxy-config
```

Suchen Sie die `scheduler` Einstellung unter `ipvs` und legen Sie den Wert auf eine der oben aufgeführten IPVS-Lastenausgleichsoptionen fest, zum Beispiel: `rr` für Round Robin. Suchen Sie die `mode` Einstellung, die standardmäßig ist`iptables`, und ändern Sie den Wert auf. `ipvs` Das Ergebnis einer der beiden Optionen sollte der folgenden Konfiguration ähneln.

```
  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
```

Wenn Ihre Worker-Knoten Ihrem Cluster hinzugefügt wurden, bevor Sie diese Änderungen vorgenommen haben, müssen Sie den DaemonSet Kube-Proxy neu starten.

```
kubectl -n kube-system rollout restart ds kube-proxy
```

Validierung

Sie können überprüfen, ob Ihr Cluster und Ihre Worker-Knoten im IPVS-Modus ausgeführt werden, indem Sie den folgenden Befehl auf einem Ihrer Worker-Knoten ausführen.

```
sudo ipvsadm -L
```

Sie sollten mindestens ein Ergebnis wie das unten stehende sehen, das Einträge für den Kubernetes API Server-Dienst unter `10.100.0.1` und den CoreDNS-Dienst unter enthält. `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**
>
> Diese Beispielausgabe stammt aus einem EKS-Cluster mit einem Service-IP-Adressbereich von. `10.100.0.0/16`

Bewährte Methoden für EKS-Skalierbarkeit

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Dieser Leitfaden enthält Ratschläge zur Skalierung von EKS-Clustern. Das Ziel der Skalierung eines EKS-Clusters besteht darin, den Arbeitsaufwand, den ein einzelner Cluster ausführen kann, zu maximieren. Die Verwendung eines einzelnen, großen EKS-Clusters kann die Betriebslast im Vergleich zur Verwendung mehrerer Cluster reduzieren, bringt jedoch Kompromisse bei Bereitstellungen in mehreren Regionen, Mandantenisolierung und Cluster-Upgrades mit sich. In diesem Dokument werden wir uns darauf konzentrieren, wie mit einem einzigen Cluster maximale Skalierbarkeit erreicht werden kann.

Verwendung dieses Leitfadens

Dieses Handbuch richtet sich an Entwickler und Administratoren, die für die Erstellung und Verwaltung von EKS-Clustern in AWS verantwortlich sind. Es konzentriert sich auf einige generische Kubernetes-Skalierungspraktiken, enthält jedoch keine Besonderheiten für selbstverwaltete Kubernetes-Cluster oder Cluster, die außerhalb einer AWS-Region mit EKS Anywhere ausgeführt werden.

Jedes Thema bietet einen kurzen Überblick, gefolgt von Empfehlungen und bewährten Methoden für den skalierbaren Betrieb von EKS-Clustern. Die Themen müssen nicht in einer bestimmten Reihenfolge gelesen werden, und Empfehlungen sollten nicht angewendet werden, ohne sie zu testen und zu überprüfen, ob sie in Ihren Clustern funktionieren.

Die Dimensionen der Skalierung verstehen

Skalierbarkeit unterscheidet sich von Leistung und Zuverlässigkeit, und alle drei sollten bei der Planung Ihrer Cluster- und Workload-Anforderungen berücksichtigt werden. Wenn Cluster skalieren, müssen sie überwacht werden. In diesem Leitfaden werden jedoch keine bewährten Methoden für die Überwachung behandelt. EKS kann auf große Größen skaliert werden, aber Sie müssen planen, wie Sie einen Cluster auf mehr als 300 Knoten oder 5.000 Pods skalieren wollen. Dies sind keine

absoluten Zahlen, sondern sie stammen aus der Zusammenarbeit dieses Handbuchs mit mehreren Benutzern, Technikern und Support-Experten.

Die Skalierung in Kubernetes ist mehrdimensional und es gibt keine spezifischen Einstellungen oder Empfehlungen, die in jeder Situation funktionieren. Zu den wichtigsten Bereichen, in denen wir Hinweise zur Skalierung geben können, gehören:

Kubernetes Control Plane in einem EKS-Cluster umfasst alle Services, die AWS automatisch für Sie ausführt und skaliert (z. B. Kubernetes API-Server). Die Skalierung der Kontrollebene liegt in der Verantwortung von AWS, aber die verantwortungsvolle Nutzung der Kontrollebene liegt in Ihrer Verantwortung.

Die Skalierung von Kubernetes Data Plane befasst sich mit AWS-Ressourcen, die für Ihren Cluster und Ihre Workloads erforderlich sind, sich jedoch außerhalb der EKS-Kontrollebene befinden. Ressourcen wie EC2-Instances, Kubelet und Speicher müssen alle mit der Skalierung Ihres Clusters skaliert werden.

Clusterdienste sind Kubernetes-Controller und -Anwendungen, die innerhalb des Clusters ausgeführt werden und Funktionen für Ihren Cluster und Ihre Workloads bereitstellen. Dies können EKS-Add-Ons und auch andere Dienste oder Helm-Charts sein, die Sie aus Compliance-Gründen und Integrationen installieren. Diese Dienste hängen oft von Workloads ab, und wenn Ihre Workloads wachsen, müssen Ihre Cluster-Services mit ihnen skalieren.

Workloads sind der Grund dafür, dass Sie einen Cluster haben und diese horizontal mit dem Cluster skalieren sollten. Es gibt Integrationen und Einstellungen, die Workloads in Kubernetes haben und die die Skalierung des Clusters unterstützen können. Bei Kubernetes-Abstraktionen wie Namespaces und Diensten gibt es auch architektonische Überlegungen.

Extra große Skalierung

Wenn Sie einen einzelnen Cluster auf mehr als 1.000 Knoten oder 50.000 Pods skalieren, würden wir uns freuen, mit Ihnen zu sprechen. Wir empfehlen Ihnen, sich an Ihr Support-Team oder Ihren technischen Kundenbetreuer zu wenden, um mit Experten in Kontakt zu treten, die Ihnen bei der Planung und Skalierung helfen können, und zwar über die Informationen in diesem Leitfaden hinaus. Amazon EKS kann bis zu 100.000 Knoten in einem einzigen Cluster unterstützen, wenn Sie für das Onboarding ausgewählt werden.

Kubernetes-Steuerebene

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Die Kubernetes-Steuerebene besteht aus dem Kubernetes API-Server, dem Kubernetes Controller Manager, dem Scheduler und anderen Komponenten, die für das Funktionieren von Kubernetes erforderlich sind. Die Skalierbarkeitsgrenzen dieser Komponenten sind je nachdem, was Sie im Cluster ausführen, unterschiedlich. Zu den Bereichen mit den größten Auswirkungen auf die Skalierung gehören jedoch die Kubernetes-Version, die Auslastung und die Skalierung einzelner Knoten.

Begrenzen Sie Arbeitslast und Node-Bursting

> **Important**
>
> Um zu vermeiden, dass API-Grenzwerte auf der Steuerungsebene erreicht werden, sollten Sie Skalierungsspitzen begrenzen, die die Clustergröße jeweils um zweistellige Prozentsätze erhöhen (z. B. 1000 Knoten auf 1100 Knoten oder 4000 bis 4500 Pods gleichzeitig).

Die EKS-Steuerungsebene wird automatisch skaliert, wenn Ihr Cluster wächst, aber es gibt Grenzen, wie schnell sie skaliert werden kann. Wenn Sie zum ersten Mal einen EKS-Cluster erstellen, kann die Steuerungsebene nicht sofort auf Hunderte von Knoten oder Tausende von Pods skaliert werden. Weitere Informationen darüber, wie EKS die Skalierung verbessert hat, finden Sie in diesem Blogbeitrag.

Für die Skalierung großer Anwendungen muss sich die Infrastruktur anpassen, damit sie vollständig einsatzbereit ist (z. B. Load Balancer, die sich erwärmen). Um die Geschwindigkeit der Skalierung zu kontrollieren, stellen Sie sicher, dass Sie auf der Grundlage der richtigen Metriken für Ihre Anwendung skalieren. Die CPU- und Speicherskalierung kann Ihre Anwendungsbeschränkungen möglicherweise nicht genau vorhersagen, und die Verwendung benutzerdefinierter Metriken (z. B. Anfragen pro Sekunde) in Kubernetes Horizontal Pod Autoscaler (HPA) ist möglicherweise eine bessere Skalierungsoption.

Informationen zur Verwendung einer benutzerdefinierten Metrik finden Sie in den Beispielen in der Kubernetes-Dokumentation. Wenn Sie erweiterte Skalierungsanforderungen haben oder auf der Grundlage externer Quellen (z. B. AWS SQS-Warteschlange) skalieren müssen, verwenden Sie KEDA für die ereignisbasierte Workload-Skalierung.

Skalieren Sie Knoten und Pods sicher herunter

Ersetzen Sie Instanzen mit langer Laufzeit

Durch den regelmäßigen Austausch von Knoten bleibt Ihr Cluster funktionsfähig, da Konfigurationsabweichungen und Probleme vermieden werden, die erst nach längerer Verfügbarkeit auftreten (z. B. langsame Speicherlecks). Durch den automatisierten Austausch erhalten Sie bewährte Verfahren und Verfahren für Knoten-Upgrades und Sicherheitspatches. Wenn jeder Knoten in Ihrem Cluster regelmäßig ausgetauscht wird, ist weniger Aufwand erforderlich, um separate Prozesse für die laufende Wartung aufrechtzuerhalten.

Verwenden Sie die TTL-Einstellungen (Time to Live) von Karpenter, um Instanzen zu ersetzen, nachdem sie eine bestimmte Zeit lang ausgeführt wurden. Selbstverwaltete Knotengruppen können die `max-instance-lifetime` Einstellung verwenden, um Knoten automatisch zu wechseln. Verwaltete Knotengruppen verfügen derzeit nicht über diese Funktion, aber Sie können die Anfrage hier verfolgen GitHub.

Nicht ausgelastete Knoten entfernen

Sie können Knoten entfernen, wenn sie keine laufenden Workloads haben, indem Sie den Schwellenwert für das Herunterskalieren im Kubernetes Cluster Autoscaler mit `--scale-down-utilization-threshold`oder in Karpenter können Sie die Provisioner-Einstellung verwenden. `ttlSecondsAfterEmpty`

Verwenden Sie Budgets für Pod-Unterbrechungen und das sichere Herunterfahren von Knoten

Um Pods und Knoten aus einem Kubernetes-Cluster zu entfernen, müssen Controller Aktualisierungen an mehreren Ressourcen vornehmen (z. B. EndpointSlices). Wenn Sie dies häufig oder zu schnell tun, kann dies zu einer Drosselung des API-Servers und zu Anwendungsausfällen führen, wenn Änderungen auf die Controller übertragen werden. Budgets für Pod-Unterbrechungen sind eine bewährte Methode, um die Fluktuation zu verlangsamen und so die Verfügbarkeit von Workloads zu schützen, wenn Knoten in einem Cluster entfernt oder neu geplant werden.

Verwenden Sie den clientseitigen Cache, wenn Sie Kubectl ausführen

Wenn Sie den Befehl kubectl ineffizient verwenden, kann dies zu zusätzlicher Belastung des Kubernetes-API-Servers führen. Sie sollten vermeiden, Skripts oder Automatisierungen auszuführen, die kubectl wiederholt verwenden (z. B. in einer For-Schleife), oder Befehle ohne lokalen Cache auszuführen.

`kubectl`hat einen clientseitigen Cache, der Erkennungsinformationen aus dem Cluster zwischenspeichert, um die Anzahl der erforderlichen API-Aufrufe zu reduzieren. Der Cache ist standardmäßig aktiviert und wird alle 10 Minuten aktualisiert.

Wenn Sie kubectl aus einem Container oder ohne clientseitigen Cache ausführen, können Probleme mit der API-Drosselung auftreten. Es wird empfohlen, den Cluster-Cache beizubehalten, indem Sie ihn mounten, um unnötige API-Aufrufe `--cache-dir` zu vermeiden.

Deaktivieren Sie die Kubectl-Komprimierung

Das Deaktivieren der kubectl-Komprimierung in Ihrer kubeconfig-Datei kann die API- und Client-CPU-Auslastung reduzieren. Standardmäßig komprimiert der Server die an den Client gesendeten Daten, um die Netzwerkbandbreite zu optimieren. Dadurch erhöht sich die CPU-Last auf dem Client und dem Server bei jeder Anfrage, und wenn Sie über ausreichend Bandbreite verfügen, können durch Deaktivieren der Komprimierung der Overhead und die Latenz reduziert werden. Um die Komprimierung zu deaktivieren, können Sie das `--disable-compression=true` Flag verwenden oder es `disable-compression: true` in Ihrer kubeconfig-Datei festlegen.

```
apiVersion: v1
clusters:
- cluster:
    server: serverURL
    disable-compression: true
  name: cluster
```

Shard Cluster Autoscaler

Der Kubernetes Cluster Autoscaler wurde für die Skalierung auf bis zu 1000 Knoten getestet. Auf einem großen Cluster mit mehr als 1000 Knoten wird empfohlen, mehrere Instanzen des Cluster Autoscaler im Shard-Modus auszuführen. Jede Cluster Autoscaler-Instanz ist so konfiguriert, dass sie eine Reihe von Knotengruppen skaliert. Das folgende Beispiel zeigt zwei Cluster-Autoscaling-Konfigurationen, die für jeweils 4 Knotengruppen konfiguriert sind.

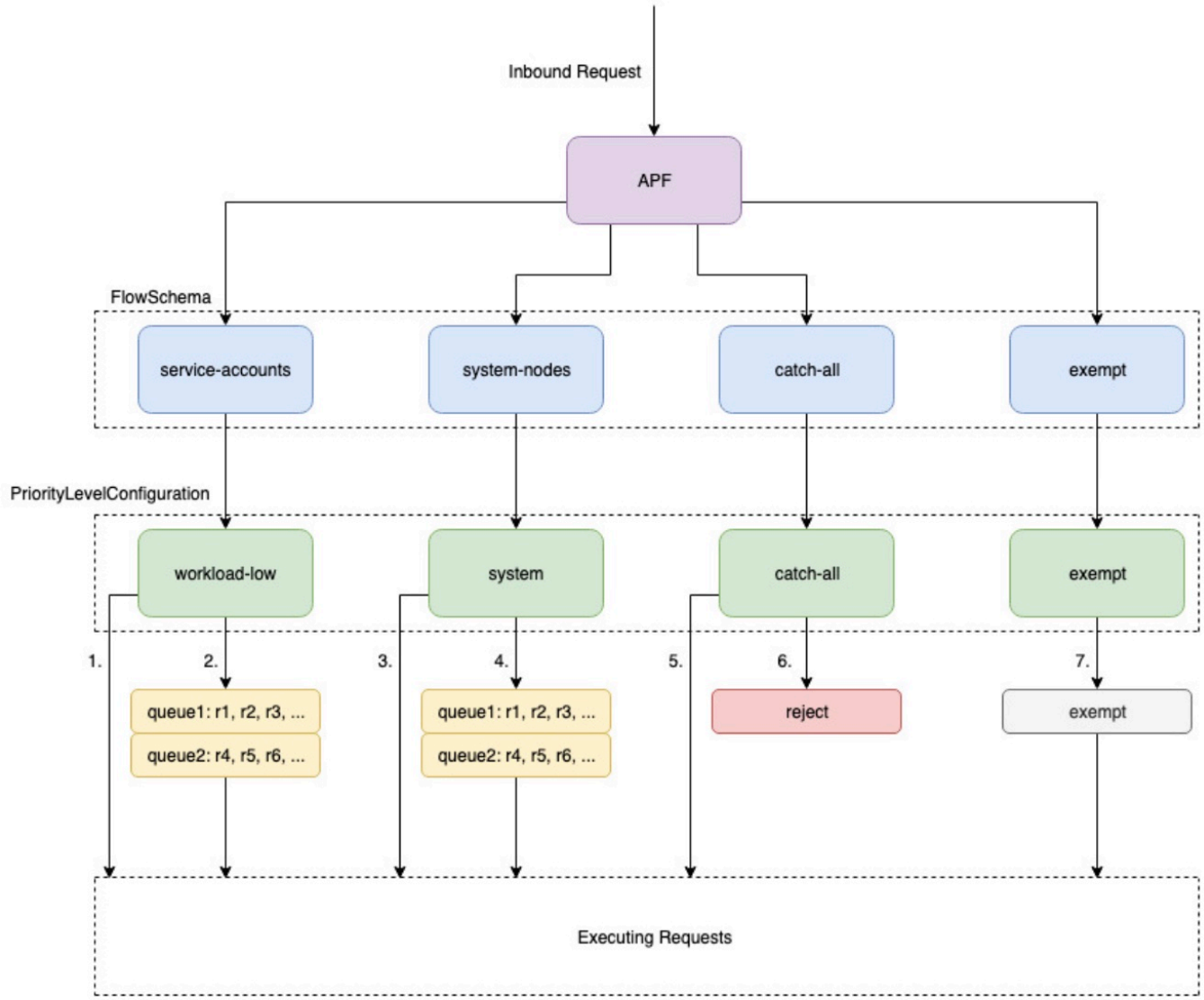
ClusterAutoscaler-1

```
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
```

API-Priorität und Fairness



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.

-Übersicht

Um sich bei häufigen Anfragen vor Überlastung zu schützen, begrenzt der API-Server die Anzahl der Inflight-Anfragen, die zu einem bestimmten Zeitpunkt noch ausstehen können. Sobald dieses

Limit überschritten wird, beginnt der API-Server, Anfragen abzulehnen und gibt einen 429-HTTP-Antwortcode für „Zu viele Anfragen“ an die Clients zurück. Es ist vorzuziehen, dass der Server Anfragen verwirft und die Clients es später erneut versuchen, als keine serverseitigen Beschränkungen für die Anzahl der Anfragen festzulegen und die Steuerungsebene zu überlasten, was zu Leistungseinbußen oder Nichtverfügbarkeit führen könnte.

Der Mechanismus, mit dem Kubernetes konfiguriert, wie diese Inflight-Anfragen auf verschiedene Anfragetypen aufgeteilt werden, wird API-Priorität und Fairness genannt. Der API-Server konfiguriert die Gesamtzahl der Anfragen an Bord, die er annehmen kann, indem er die durch die Flags und angegebenen Werte summiert. `--max-requests-inflight` `--max-mutating-requests-inflight` EKS verwendet die Standardwerte von 400 und 200 Anfragen für diese Flags, sodass insgesamt 600 Anfragen zu einem bestimmten Zeitpunkt versendet werden können. Da die Steuerungsebene jedoch als Reaktion auf die zunehmende Auslastung und die Fluktuation der Arbeitslast auf größere Größen skaliert wird, wird die Quote für Anfragen während der Übertragung bis 2000 entsprechend erhöht (Änderungen vorbehalten). APF gibt an, wie diese Quote für Anfragen während des Fluges weiter auf verschiedene Anforderungstypen aufgeteilt wird. Beachten Sie, dass EKS-Kontrollebenen hochverfügbar sind und mindestens 2 API-Server für jeden Cluster registriert sind. Das bedeutet, dass die Gesamtzahl der Inflight-Anfragen, die Ihr Cluster verarbeiten kann, doppelt so hoch ist (oder höher, wenn die horizontale Skalierung weiter ausgebaut wird) wie die pro Kube-Apiserver festgelegte Inflight-Quote. Dies entspricht mehreren Tausend auf den größten EKS-Clustern. requests/second

Zwei Arten von Kubernetes-Objekten, die als PriorityLevelConfigurations und bezeichnet werden FlowSchemas, konfigurieren, wie die Gesamtzahl der Anfragen auf verschiedene Anforderungstypen aufgeteilt wird. Diese Objekte werden vom API-Server automatisch verwaltet und EKS verwendet die Standardkonfiguration dieser Objekte für die jeweilige Kubernetes-Nebenversion. PriorityLevelConfigurations stellen einen Bruchteil der Gesamtzahl der zulässigen Anfragen dar. Beispielsweise werden dem Workload-High PriorityLevelConfiguration 98 der insgesamt 600 Anfragen zugewiesen. Die Summe der allen PriorityLevelConfigurations zugewiesenen Anfragen entspricht 600 (oder etwas mehr als 600, da der API-Server aufrundet, wenn einer bestimmten Stufe nur ein Bruchteil einer Anfrage gewährt wird). Um den Wert PriorityLevelConfigurations in Ihrem Cluster und die Anzahl der einzelnen Anfragen zu überprüfen, können Sie den folgenden Befehl ausführen. Dies sind die Standardeinstellungen für 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
```

Der zweite Objekttyp ist. FlowSchemas API-Serveranfragen mit einem bestimmten Satz von Eigenschaften werden denselben Kategorien zugeordnet FlowSchema. Zu diesen Eigenschaften gehören entweder der authentifizierte Benutzer oder Attribute der Anfrage, z. B. die API-Gruppe, der Namespace oder die Ressource. A gibt FlowSchema auch an, welcher Art von Anfrage PriorityLevelConfiguration dieser Anfragetyp zugeordnet werden soll. Die beiden Objekte zusammen sagen: „Ich möchte, dass diese Art von Anfrage auf diesen Anteil der Anfragen während des Fluges angerechnet wird.“ Wenn eine Anfrage den API-Server erreicht, überprüft dieser jede Anfrage, FlowSchemas bis eine gefunden wird, die allen erforderlichen Eigenschaften entspricht. Wenn mehrere mit einer Anfrage FlowSchemas übereinstimmen, wählt der API-Server die FlowSchema mit der kleinsten übereinstimmenden Priorität aus, die als Eigenschaft im Objekt angegeben ist.

Die Zuordnung von FlowSchemas to PriorityLevelConfigurations kann mit diesem Befehl eingesehen werden:

```
$ 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 kann den Typ Queue, Reject oder Exempt haben. Bei den Typen Queue und Reject gilt für diese Prioritätsstufe ein Limit für die maximale Anzahl von Inflight-Anfragen. Das Verhalten ist jedoch unterschiedlich, wenn dieses Limit erreicht wird. Der Workload-High PriorityLevelConfiguration verwendet beispielsweise den Typ Queue und verfügt über 98 Anfragen, die vom Controller-Manager, Endpoint-Controller, Scheduler, EKS-bezogenen Controllern und von Pods, die im Kube-System-Namespace ausgeführt werden, verwendet werden können. Da der Typ Queue verwendet wird, versucht der API-Server, Anfragen im Speicher zu belassen und hofft, dass die Anzahl der Inflight-Anfragen unter 98 fällt, bevor diese Anfragen das Timeout überschreiten. Wenn bei einer bestimmten Anfrage ein Timeout in der Warteschlange auftritt oder wenn zu viele Anfragen bereits in der Warteschlange stehen, hat der API-Server keine andere Wahl, als die Anfrage zu verwerfen und dem Client eine 429 zurückzugeben. Beachten Sie, dass die Warteschlange zwar verhindern kann, dass eine Anfrage eine 429-Nummer erhält, dass sie jedoch den Nachteil einer erhöhten end-to-end Latenz bei der Anfrage mit sich bringt.

Betrachten Sie nun den Catch-All FlowSchema , der dem Catch-All mit dem Typ Reject zugeordnet PriorityLevelConfiguration ist. Wenn Clients das Limit von 13 Inflight-Anfragen erreichen, führt der API-Server keine Warteschlangen durch und verwirft die Anfragen sofort mit dem Antwortcode 429. Und schließlich erhalten Anfragen, die einer PriorityLevelConfiguration vom Typ Exempt zugeordnet sind, niemals eine 429 und werden immer sofort versendet. Dies wird für Anfragen mit hoher Priorität verwendet, wie z. B. Healthz-Anfragen oder Anfragen, die aus der Gruppe system:masters kommen.

Überwachung von APF und verworfenen Anfragen

Um zu überprüfen, ob Anfragen aufgrund von APF gelöscht wurden, `apiserver_flowcontrol_rejected_requests_total` können die API-Server-Metriken überwacht werden, um die betroffenen FlowSchemas und zu überprüfen. PriorityLevelConfigurations

Diese Metrik zeigt beispielsweise, dass 100 Anfragen von den Service-Accounts aufgrund eines Timeouts bei Anfragen in Warteschlangen mit niedriger Auslastung verworfen FlowSchema wurden:

```
% 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
```

Um zu überprüfen, wie nah eine bestimmte Person daran PriorityLevelConfiguration ist, 429 Sekunden zu empfangen oder aufgrund von Warteschlangen eine erhöhte Latenz zu erleben, können Sie den Unterschied zwischen dem Parallelitätslimit und der verwendeten Parallelität vergleichen. In diesem Beispiel haben wir einen Puffer von 100 Anfragen.

```
% 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
```

Um zu überprüfen, ob bei einer bestimmten PriorityLevelConfiguration Anfrage Warteschlangen, aber nicht unbedingt verworfene Anfragen auftreten, `apiserver_flowcontrol_current_inqueue_requests` kann auf die Metrik für verwiesen werden:

```
% 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
```

Zu den weiteren nützlichen Prometheus-Metriken gehören:

- apiserver_flowcontrol_dispatched_requests_total
- apiserver_flowcontrol_request_execution_seconds
- apiserver_flowcontrol_request_wait_duration_seconds

Eine vollständige Liste der APF-Metriken finden Sie in der Upstream-Dokumentation.

Verhinderung verworfener Anfragen

Vermeiden Sie 429s, indem Sie Ihren Workload ändern

Wenn APF Anfragen aufgrund einer bestimmten PriorityLevelConfiguration Überschreitung der maximal zulässigen Anzahl von Inflight-Anfragen verwirft, FlowSchemas können die betroffenen Clients die Anzahl der Anfragen, die zu einem bestimmten Zeitpunkt ausgeführt werden, verringern. Dies kann erreicht werden, indem die Gesamtzahl der Anfragen reduziert wird, die in dem Zeitraum gestellt wurden, in dem 429 Anfragen gestellt wurden. Beachten Sie, dass Anfragen mit langer Laufzeit, wie z. B. teure Listenanrufe, besonders problematisch sind, da sie für die gesamte Dauer ihrer Ausführung als Inflight-Anfragen gelten. Wenn Sie die Anzahl dieser teuren Anfragen reduzieren oder die Latenz dieser Listenaufrufen optimieren (z. B. indem Sie die Anzahl der pro Anfrage abgerufenen Objekte reduzieren oder zur Verwendung einer Überwachungsanforderung wechseln), können Sie dazu beitragen, die Gesamtparallelität zu reduzieren, die für die jeweilige Arbeitslast erforderlich ist.

Vermeiden Sie 429s, indem Sie Ihre APF-Einstellungen ändern

> **Warning**
>
> Ändern Sie die APF-Standardeinstellungen nur, wenn Sie wissen, was Sie tun. Falsch konfigurierte APF-Einstellungen können zu verworfenen API-Serveranfragen und erheblichen Störungen der Arbeitslast führen.

Ein weiterer Ansatz zur Verhinderung verworfener Anfragen besteht darin, die Standardeinstellung FlowSchemas oder die PriorityLevelConfigurations Installation auf EKS-Clustern zu ändern. EKS installiert die Upstream-Standardeinstellungen für FlowSchemas und PriorityLevelConfigurations für die angegebene Kubernetes-Nebenversion. Der API-Server gleicht diese Objekte automatisch mit ihren Standardwerten ab, sofern sie nicht geändert werden, es sei denn, die folgende Anmerkung zu den Objekten ist auf „Falsch“ gesetzt:

```
metadata:
  annotations:
    apf.kubernetes.io/autoupdate-spec: "false"
```

Auf hoher Ebene können die APF-Einstellungen wie folgt geändert werden:

- Weisen Sie Anfragen, die Ihnen wichtig sind, mehr Kapazität an Bord zu.

- Isolieren Sie unwichtige oder teure Anfragen, die zu Kapazitätsverlusten für andere Anforderungstypen führen können.

Dies kann erreicht werden, indem entweder die Standardeinstellung FlowSchemas geändert PriorityLevelConfigurations oder neue Objekte dieser Typen erstellt werden. Betreiber können die Werte assuredConcurrencyShares für die entsprechenden PriorityLevelConfigurations Objekte erhöhen, um den Anteil der ihnen zugewiesenen Anfragen während des Fluges zu erhöhen. Darüber hinaus kann die Anzahl der Anfragen, die zu einem bestimmten Zeitpunkt in die Warteschlange gestellt werden können, erhöht werden, wenn die Anwendung die zusätzliche Latenz bewältigen kann, die dadurch entsteht, dass Anfragen vor dem Versand in die Warteschlange gestellt werden.

Alternativ können neue FlowSchema PriorityLevelConfigurations Objekte erstellt werden, die speziell auf die Arbeitslast des Kunden zugeschnitten sind. Beachten Sie, dass die Anzahl der Anfragen, die von anderen Buckets bearbeitet PriorityLevelConfigurations werden können, reduziert wird, wenn Sie entweder vorhandenen PriorityLevelConfigurations oder neuen mehr assuredConcurrencyShares Anfragen zuweisen, da das Gesamtlimit bei 600 Inflight pro API-Server bleibt.

Wenn Sie Änderungen an den APF-Standardeinstellungen vornehmen, sollten diese Metriken auf einem Cluster außerhalb der Produktion überwacht werden, um sicherzustellen, dass eine Änderung der Einstellungen nicht zu unbeabsichtigten 429s führt:

1. Die Metrik für `apiserver_flowcontrol_rejected_requests_total` sollte für alle überwacht werden, FlowSchemas um sicherzustellen, dass keine Buckets anfangen, Anfragen zu verwerfen.
2. Die Werte für `apiserver_flowcontrol_nominal_limit_seats` und `apiserver_flowcontrol_current_executing_seats` sollten miteinander verglichen werden, um sicherzustellen, dass bei der Verwendung von Parallelität kein Risiko besteht, dass das Limit für diese Prioritätsstufe überschritten wird.

Ein häufiger Anwendungsfall für die Definition eines neuen FlowSchema und PriorityLevelConfiguration ist die Isolierung. Nehmen wir an, wir möchten lang andauernde Listenereignisaufrufe von Pods auf ihren eigenen Anteil an Anfragen isolieren. Dadurch wird verhindert, dass wichtige Anfragen von Pods, die die vorhandenen Dienstkonten verwenden, 429 Anfragen FlowSchema erhalten und ihnen die Anforderungskapazität entzogen wird. Denken Sie daran, dass die Gesamtzahl der Anfragen während der Übertragung begrenzt ist. Dieses Beispiel zeigt jedoch, dass die APF-Einstellungen geändert werden können, um die Anforderungskapazität besser auf die jeweilige Arbeitslast aufzuteilen:

FlowSchema Beispielobjekt zur Isolierung von Listenereignisanfragen:

```
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
```

- Es FlowSchema erfasst alle Listenereignisaufrufen, die von Dienstkonten im Standard-Namespace getätigt wurden.
- Die Vergleichspriorität 8000 ist niedriger als der Wert 9000, der von den vorhandenen Dienstkonten verwendet wird, FlowSchema sodass diese Listenereignisaufrufe eher mit list-events-default-service -accounts als mit service-accounts übereinstimmen.
- Wir verwenden den Catch-All, um diese Anfragen PriorityLevelConfiguration zu isolieren. In diesem Bucket können nur 13 eingehende Anfragen für diese lang andauernden List-Event-Aufrufe verwendet werden. Pods erhalten ab sofort 429 Anfragen, sobald sie versuchen, mehr als 13 dieser Anfragen gleichzeitig zu stellen.

Ressourcen auf dem API-Server werden abgerufen

Das Abrufen von Informationen vom API-Server ist ein erwartetes Verhalten für Cluster jeder Größe. Wenn Sie die Anzahl der Ressourcen im Cluster skalieren, können die Häufigkeit der Anfragen und das Datenvolumen schnell zu einem Engpass für die Kontrollebene werden und zu Latenz und Langsamkeit der API führen. Je nach Schweregrad der Latenz kann es zu unerwarteten Ausfallzeiten kommen, wenn Sie nicht vorsichtig sind.

Zu wissen, was Sie anfordern und wie oft, sind die ersten Schritte, um diese Art von Problemen zu vermeiden. Im Folgenden finden Sie Anleitungen zur Begrenzung des Abfragevolumens auf der Grundlage der bewährten Skalierungsmethoden. In diesem Abschnitt werden die Vorschläge in der Reihenfolge gegeben, mit den Optionen zu beginnen, von denen bekannt ist, dass sie sich am besten skalieren lassen.

Verwenden Sie Shared Informers

Bei der Entwicklung von Controllern und Automatisierungen, die in die Kubernetes-API integriert sind, müssen Sie häufig Informationen aus Kubernetes-Ressourcen abrufen. Wenn Sie regelmäßig nach diesen Ressourcen fragen, kann dies zu einer erheblichen Belastung des API-Servers führen.

Wenn Sie einen Informer aus der Client-Go-Bibliothek verwenden, haben Sie den Vorteil, dass Sie anhand von Ereignissen nach Änderungen an den Ressourcen Ausschau halten können, anstatt nach Änderungen zu suchen. Informer reduzieren die Auslastung weiter, indem sie einen gemeinsamen Cache für die Ereignisse und Änderungen verwenden, sodass mehrere Controller, die dieselben Ressourcen überwachen, keine zusätzliche Last verursachen.

Controller sollten es vermeiden, clusterweite Ressourcen ohne Labels und Feldselektoren abzufragen, insbesondere in großen Clustern. Bei jeder ungefilterten Abfrage müssen viele unnötige Daten von etcd über den API-Server gesendet werden, um vom Client gefiltert zu werden. Durch Filtern auf der Grundlage von Labels und Namespaces können Sie den Arbeitsaufwand reduzieren, den der API-Server ausführen muss, um die Anfrage und die an den Client gesendeten Daten zu erfüllen.

Optimieren Sie die Nutzung der Kubernetes-API

Wenn Sie die Kubernetes-API mit benutzerdefinierten Controllern oder Automatisierung aufrufen, ist es wichtig, dass Sie die Aufrufe nur auf die Ressourcen beschränken, die Sie benötigen. Ohne Einschränkungen können Sie den API-Server usw. unnötig belasten.

Es wird empfohlen, wann immer möglich das Watch-Argument zu verwenden. Ohne Argumente ist das Standardverhalten das Auflisten von Objekten. Um watch statt list zu verwenden, können Sie es `?watch=true` an das Ende Ihrer API-Anfrage anhängen. Um beispielsweise alle Pods im Standard-Namespace mit einer Uhr abzurufen, verwenden Sie:

```
/api/v1/namespaces/default/pods?watch=true
```

Wenn Sie Objekte auflisten, sollten Sie den Umfang dessen, was Sie auflisten, und die Menge der zurückgegebenen Daten einschränken. Sie können die zurückgegebenen Daten einschränken, indem Sie den Anfragen `limit=500` Argumente hinzufügen. Das `fieldSelector` Argument und der `/namespace/` Pfad können nützlich sein, um sicherzustellen, dass Ihre Listen so eng wie nötig abgegrenzt sind. Um beispielsweise nur laufende Pods im Standard-Namespace aufzulisten, verwenden Sie den folgenden API-Pfad und die folgenden Argumente.

```
/api/v1/namespaces/default/pods?fieldSelector=status.phase=Running&limit=500
```

Oder listen Sie alle Pods auf, die ausgeführt werden mit:

```
/api/v1/pods?fieldSelector=status.phase=Running&limit=500
```

Eine weitere Option, um Watch-Aufrufe oder aufgelistete Objekte einzuschränken, ist `resourceVersions`die Verwendung dieser Option. Weitere Informationen finden Sie in der Kubernetes-Dokumentation. Ohne `resourceVersion` Argument erhalten Sie die neueste verfügbare Version, für die ein etcd-Quorum-Read erforderlich ist. Dies ist der teuerste und langsamste Lesevorgang für die Datenbank. Die ResourceVersion hängt davon ab, welche Ressourcen Sie abfragen möchten, und kann im Feld gefunden werden. `metadata.resourseVersion` Dies wird auch empfohlen, wenn Sie Watch-Anrufe und nicht nur Listenanrufe verwenden

Es ist ein spezielles Programm `resourceVersion=0` verfügbar, das Ergebnisse aus dem API-Server-Cache zurückgibt. Dies kann die Belastung von etcd reduzieren, unterstützt aber keine Paginierung.

```
/api/v1/namespaces/default/pods?resourceVersion=0
```

Es wird empfohlen, watch mit einer ResourceVersion zu verwenden, die auf den neuesten bekannten Wert gesetzt ist, der aus der vorherigen Liste oder Überwachung empfangen wurde. Dies wird

automatisch in Client-Go abgewickelt. Es wird jedoch empfohlen, dies noch einmal zu überprüfen, wenn Sie einen k8s-Client in anderen Sprachen verwenden.

```
/api/v1/namespaces/default/pods?watch=true&resourceVersion=362812295
```

Wenn Sie die API ohne Argumente aufrufen, ist sie für den API-Server usw. am ressourcenintensivsten. Dieser Aufruf ruft alle Pods in allen Namespaces ab, ohne Seitennummerierung oder Einschränkung des Gültigkeitsbereichs und erfordert ein Quorum, das aus etcd gelesen wird.

```
/api/v1/pods
```

DaemonSet Beugen Sie donnernden Herden vor

A DaemonSet stellt sicher, dass auf allen (oder einigen) Knoten eine Kopie eines Pods ausgeführt wird. Wenn Knoten dem Cluster beitreten, erstellt der Daemonset-Controller Pods für diese Knoten. Wenn Knoten den Cluster verlassen, werden diese Pods im Müll gesammelt. Durch das Löschen eines DaemonSet werden die erstellten Pods bereinigt.

Einige typische Verwendungen von a DaemonSet sind:

- Auf jedem Knoten wird ein Cluster-Speicher-Daemon ausgeführt
- Auf jedem Knoten wird ein Daemon zur Erfassung von Protokollen ausgeführt
- Auf jedem Knoten wird ein Daemon zur Knotenüberwachung ausgeführt

In Clustern mit Tausenden von Knoten kann das Erstellen eines neuen DaemonSet, das Aktualisieren eines DaemonSet oder das Erhöhen der Anzahl von Knoten zu einer hohen Belastung der Steuerungsebene führen. Wenn DaemonSet Pods beim Start des Pods teure API-Serveranfragen ausgeben, können sie aufgrund einer großen Anzahl gleichzeitiger Anfragen zu einem hohen Ressourcenverbrauch auf der Steuerungsebene führen.

Im Normalbetrieb können Sie a verwenden, `RollingUpdate` um die schrittweise Einführung neuer DaemonSet Pods sicherzustellen. Bei einer `RollingUpdate` Aktualisierungsstrategie beendet der Controller nach dem Aktualisieren einer DaemonSet Vorlage alte DaemonSet Pods und erstellt automatisch und kontrolliert neue DaemonSet Pods. Während des gesamten Aktualisierungsvorgangs DaemonSet wird auf jedem Knoten höchstens ein Pod von ausgeführt. Sie können einen schrittweisen Rollout durchführen, indem Sie `maxUnavailable` die Werte `maxSurge` auf 1, 0 und 60 `minReadySeconds` einstellen. Wenn Sie keine Aktualisierungsstrategie angeben,

erstellt Kubernetes standardmäßig eine `RollingUpdate` mit `maxUnavailable` den Werten 1, `maxSurge` 0 und `minReadySeconds` 0.

```
minReadySeconds: 60
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 0
    maxUnavailable: 1
```

A `RollingUpdate` gewährleistet die schrittweise Einführung neuer DaemonSet Pods, sofern der DaemonSet bereits erstellt wurde und die erwartete Anzahl von `Ready` Pods auf allen Knoten aufweist. Unter bestimmten Bedingungen, die nicht durch Strategien abgedeckt sind, kann es zu Problemen mit donnernden Herden `RollingUpdate` kommen.

Vermeiden Sie donnernde Herden auf der Erde DaemonSet

Standardmäßig, unabhängig von der `RollingUpdate` Konfiguration, erstellt der Daemonset-Controller im Pods für alle passenden Knoten gleichzeitig, wenn kube-controller-manager Sie einen neuen erstellen. DaemonSet Um einen schrittweisen Rollout von Pods zu erzwingen, nachdem Sie einen erstellt haben DaemonSet, können Sie entweder ein oder verwenden. `NodeSelector` `NodeAffinity` Dadurch wird ein Knoten erstellt DaemonSet , der Null entspricht. Anschließend können Sie die Knoten schrittweise aktualisieren, sodass sie für die Ausführung eines Pods mit einer kontrollierten Geschwindigkeit DaemonSet in Frage kommen. Sie können diesem Ansatz folgen:

- Fügen Sie allen Knoten für ein Label hinzu`run-daemonset=false`.

```
kubectl label nodes --all run-daemonset=false
```

- Erstellen Sie Ihre DaemonSet mit einer `NodeAffinity` Einstellung, die zu jedem Knoten ohne `run-daemonset=false` Label passt. Dies führt zunächst dazu, DaemonSet dass Sie keine entsprechenden Pods haben.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
```

```
        - key: run-daemonset
          operator: NotIn
          values:
          - "false"
```

- Entfernen Sie das `run-daemonset=false` Etikett mit einer kontrollierten Geschwindigkeit von Ihren Knoten. Sie können dieses Bash-Skript als Beispiel verwenden:

```
#!/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
```

- Optional können Sie die `NodeAffinity` Einstellung aus Ihrem DaemonSet Objekt entfernen. Beachten Sie, dass dadurch auch ein Pod ausgelöst `RollingUpdate` und nach und nach alle vorhandenen DaemonSet Pods ersetzt werden, da sich die DaemonSet Vorlage geändert hat.

Vermeiden Sie donnernde Herden bei Node-Scale-Outs

Ähnlich wie bei der DaemonSet Erstellung kann die schnelle Erstellung neuer Knoten dazu führen, dass eine große Anzahl von DaemonSet Pods gleichzeitig gestartet wird. Sie sollten neue Knoten mit einer kontrollierten Geschwindigkeit erstellen, damit der Controller DaemonSet Pods mit derselben Geschwindigkeit erstellt. Falls dies nicht möglich ist, können Sie dafür sorgen, dass die neuen Knoten zunächst nicht für die vorhandenen Knoten in Frage kommen, DaemonSet indem Sie `NodeAffinity` Als Nächstes können Sie den neuen Knoten schrittweise ein Label hinzufügen, sodass der Daemonset-Controller Pods mit einer kontrollierten Geschwindigkeit erzeugt. Sie können diesem Ansatz folgen:

- Fügen Sie allen vorhandenen Knoten ein Label hinzu für `run-daemonset=true`

```
kubectl label nodes --all run-daemonset=true
```

- Aktualisieren Sie Ihre DaemonSet mit einer `NodeAffinity` Einstellung, die jedem Knoten mit einer `run-daemonset=true` Bezeichnung entspricht. Beachten Sie, dass dadurch auch ein Pod ausgelöst `RollingUpdate` und nach und nach alle vorhandenen DaemonSet Pods ersetzt werden, da sich die DaemonSet Vorlage geändert hat. Sie sollten warten, `RollingUpdate` bis der Vorgang abgeschlossen ist, bevor Sie mit dem nächsten Schritt fortfahren.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: run-daemonset
          operator: In
          values:
          - "true"
```

- Erstellen Sie neue Knoten in Ihrem Cluster. Beachten Sie, dass diese Knoten nicht mit der `run-daemonset=true` Bezeichnung versehen DaemonSet werden, sodass sie nicht mit diesen Knoten übereinstimmen.
- Fügen Sie das `run-daemonset=true` Label Ihren neuen Knoten (die derzeit nicht über das `run-daemonset` Label verfügen) in kontrollierter Geschwindigkeit hinzu. Sie können dieses Bash-Skript als Beispiel verwenden:

```
#!/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
```

- Entfernen Sie optional die `NodeAffinity` Einstellung von Ihrem DaemonSet Objekt und entfernen Sie die `run-daemonset` Bezeichnung von allen Knoten.

Vermeiden Sie donnernde Herden bei Updates DaemonSet

Eine `RollingUpdate` Richtlinie berücksichtigt nur die `maxUnavailable` Einstellung für DaemonSet Pods, bei denen dies der Fall ist. `Ready` Wenn a nur `NotReady` Pods oder einen großen Prozentsatz von `NotReady` Pods DaemonSet hat und Sie seine Vorlage aktualisieren, erstellt der Daemonset-Controller gleichzeitig neue Pods für alle Pods. `NotReady` Dies kann bei einer großen Anzahl von `NotReady` Pods zu Problemen mit donnernden Herden führen, z. B. wenn die Pods ständig abstürzen oder keine Bilder abrufen können.

Um einen schrittweisen Rollout von Pods zu erzwingen, wenn Sie einen aktualisieren DaemonSet und es `NotReady` Pods gibt, können Sie die Aktualisierungsstrategie auf der DaemonSet Seite Von bis vorübergehend ändern. `RollingUpdate` `OnDelete` Wenn Sie eine DaemonSet Vorlage aktualisiert haben, erstellt der Controller neue Pods, nachdem Sie die alten manuell gelöscht haben, sodass Sie den Rollout neuer Pods steuern können. `OnDelete` Sie können diesem Ansatz folgen:

- Überprüfen Sie, ob Sie irgendwelche `NotReady` Pods in Ihrem haben DaemonSet.
- Falls nein, können Sie die DaemonSet Vorlage problemlos aktualisieren und die `RollingUpdate` Strategie gewährleistet eine schrittweise Einführung.
- Falls ja, sollten Sie zuerst Ihre aktualisieren, DaemonSet um die `OnDelete` Strategie verwenden zu können.

```
updateStrategy:
  type: OnDelete
```

- Aktualisieren Sie als Nächstes Ihre DaemonSet Vorlage mit den erforderlichen Änderungen.
- Nach diesem Update können Sie die alten DaemonSet Pods löschen, indem Sie Anfragen zum Löschen von Pods mit kontrollierter Geschwindigkeit stellen. Sie können dieses Bash-Skript als Beispiel verwenden, bei dem der DaemonSet Name im Kube-System-Namespace fluentd-elasticsearch lautet:

```
#!/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
```

- Schließlich können Sie Ihren Rücken zur früheren Strategie aktualisieren. DaemonSet `RollingUpdate`

Kubernetes-Datenebene

Die Auswahl von EC2-Instance-Typen ist möglicherweise eine der schwierigsten Entscheidungen, mit denen Kunden konfrontiert sind, da es sich um Cluster mit mehreren Workloads handelt. Es gibt keine Komplettlösung. one-size-fits Im Folgenden finden Sie einige Tipps, mit denen Sie häufig auftretende Fallstricke bei der Skalierung von Rechenleistung vermeiden können.

Automatische Knotenskalierung

Wir empfehlen Ihnen, Node Autoscaling zu verwenden, das den Arbeitsaufwand reduziert und sich tief in Kubernetes integriert. Verwaltete Knotengruppen und Karpenter werden für große Cluster empfohlen.

Verwaltete Knotengruppen bieten Ihnen die Flexibilität von Amazon EC2 Auto Scaling Scaling-Gruppen mit zusätzlichen Vorteilen für verwaltete Upgrades und Konfigurationen. Es kann mit dem Kubernetes Cluster Autoscaler skaliert werden und ist eine gängige Option für Cluster mit unterschiedlichen Rechenanforderungen.

Karpenter ist ein von AWS entwickelter Open-Source-Workload-nativer Node-Autoscaler. Er skaliert Knoten in einem Cluster auf der Grundlage der Arbeitslastanforderungen für Ressourcen (z. B. GPU) sowie von Verschmutzungen und Toleranzen (z. B. Zonenverteilung), ohne Knotengruppen zu verwalten. Knoten werden direkt in EC2 erstellt, wodurch Standardquoten für Knotengruppen — 450 Knoten pro Gruppe — vermieden werden. Dies bietet eine größere Flexibilität bei der Instanzauswahl bei geringerem Betriebsaufwand. Wir empfehlen Kunden, wenn möglich, Karpenter zu verwenden.

Verwenden Sie viele verschiedene EC2-Instance-Typen

Jede AWS-Region hat eine begrenzte Anzahl verfügbarer Instances pro Instance-Typ. Wenn Sie einen Cluster erstellen, der nur einen Instance-Typ verwendet, und die Anzahl der Knoten über die Kapazität der Region hinaus skalieren, erhalten Sie die Fehlermeldung, dass keine Instances verfügbar sind. Um dieses Problem zu vermeiden, sollten Sie die Art der Instances, die in Ihrem Cluster verwendet werden können, nicht willkürlich einschränken.

Karpenter verwendet standardmäßig eine breite Palette kompatibler Instance-Typen und wählt zum Zeitpunkt der Bereitstellung eine Instanz auf der Grundlage der ausstehenden Workload-Anforderungen, der Verfügbarkeit und der Kosten aus. Sie können die Liste der im Schlüssel von verwendeten Instanztypen erweitern. `karpenter.k8s.aws/instance-category` NodePools

Für den Kubernetes Cluster Autoscaler müssen Knotengruppen eine ähnliche Größe haben, damit sie konsistent skaliert werden können. Sie sollten mehrere Gruppen basierend auf der CPU- und Speichergröße erstellen und diese unabhängig voneinander skalieren. Verwenden Sie den ec2-instance-selector, um Instances mit ähnlicher Größe für Ihre Knotengruppen zu identifizieren.

```
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
```

Bevorzugen Sie größere Knoten, um die API-Serverlast zu reduzieren

Bei der Entscheidung, welche Instance-Typen verwendet werden sollen, belasten weniger große Knoten die Kubernetes-Steuerebene, da weniger Kubelets laufen werden. DaemonSets Große Knoten können jedoch möglicherweise nicht wie kleinere Knoten vollständig genutzt werden. Die Knotengröße sollte auf der Grundlage Ihrer Workload-Verfügbarkeit und Skalierungsanforderungen bewertet werden.

Ein Cluster mit drei U-24tb1.metal-Instances (24 TB Arbeitsspeicher und 448 Kerne) hat 3 Kubelets und wäre standardmäßig auf 110 Pods pro Knoten beschränkt. Wenn Ihre Pods jeweils 4 Kerne verwenden, ist dies möglicherweise zu erwarten (4 Kerne x 110 = 440 Kerne/Knoten). Bei einem Cluster mit 3 Knoten wäre Ihre Fähigkeit, einen Instance-Vorfall zu bewältigen, gering, da sich der Ausfall einer Instanz auf 1/3 des Clusters auswirken könnte. Sie sollten die Knotenanforderungen und die Verteilung der Pods in Ihren Workloads angeben, damit der Kubernetes-Scheduler die Workloads richtig platzieren kann.

Workloads sollten die benötigten Ressourcen und die erforderliche Verfügbarkeit anhand von Taints, Toleranzen und definieren. PodTopologySpread Sie sollten die größten Knoten bevorzugen, die voll ausgelastet werden können und die Verfügbarkeitsziele erfüllen, um die Auslastung der Steuerungsebene zu reduzieren, den Betrieb zu senken und die Kosten zu senken.

Der Kubernetes Scheduler versucht automatisch, Workloads auf Availability Zones und Hosts zu verteilen, sofern Ressourcen verfügbar sind. Wenn keine Kapazität verfügbar ist, versucht der Kubernetes Cluster Autoscaler, Knoten in jeder Availability Zone gleichmäßig hinzuzufügen. Karpenter wird versuchen, Knoten so schnell und kostengünstig wie möglich hinzuzufügen, sofern die Arbeitslast keine anderen Anforderungen erfordert.

Um zu erzwingen, dass sich Workloads mit dem Scheduler verteilen und neue Knoten auf mehrere Availability Zones erstellt werden, sollten Sie Folgendes verwenden: 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
```

Verwenden Sie ähnliche Knotengrößen für eine konsistente Workload-Leistung

Workloads sollten definieren, auf welcher Knotengröße sie ausgeführt werden müssen, um eine konsistente Leistung und eine vorhersehbare Skalierung zu ermöglichen. Ein Workload, der 500 Millionen CPUs anfordert, wird auf einer Instanz mit 4 Kernen anders abschneiden als auf einer Instanz mit 16 Kernen. Vermeiden Sie Instance-Typen, die Burstable CPUs wie Instances der T-Serie verwenden.

Um sicherzustellen, dass Ihre Workloads eine konsistente Leistung erzielen, kann ein Workload die unterstützten Karpenter-Labels verwenden, um auf bestimmte Instanzgrößen abzuzielen.

```
kind: deployment
...
spec:
  template:
    spec:
    containers:
    nodeSelector:
      karpenter.k8s.aws/instance-size: 8xlarge
```

Workloads, die in einem Cluster mit dem Kubernetes Cluster Autoscaler geplant werden, sollten basierend auf dem Label-Abgleich einen Knotenselektor den Knotengruppen zuordnen.

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/nodegroup
            operator: In
            values:
            - 8-core-node-group    # match your node group name
```

Verwenden Sie Rechenressourcen effizient

Zu den Rechenressourcen gehören EC2-Instances und Availability Zones. Die effektive Nutzung von Rechenressourcen erhöht Ihre Skalierbarkeit, Verfügbarkeit und Leistung und senkt Ihre Gesamtkosten. In einer Autoscaling-Umgebung mit mehreren Anwendungen ist es äußerst schwierig, eine effiziente Ressourcennutzung vorherzusagen. Karpenter wurde für die On-Demand-Bereitstellung von Instanzen auf der Grundlage der Workload-Anforderungen entwickelt, um die Auslastung und Flexibilität zu maximieren.

Karpenter ermöglicht es Workloads, die Art der benötigten Rechenressourcen zu deklarieren, ohne zuerst Knotengruppen zu erstellen oder Label-Taints für bestimmte Knoten zu konfigurieren. Weitere Informationen finden Sie in den Best Practices von Karpenter. Erwägen Sie, die Konsolidierung in Ihrem Karpenter Provisioner zu aktivieren, um unterausgelastete Knoten zu ersetzen.

Automatisieren Sie Amazon Machine Image (AMI) -Updates

Wenn Sie die Worker Node-Komponenten auf dem neuesten Stand halten, stellen Sie sicher, dass Sie über die neuesten Sicherheitspatches und kompatiblen Funktionen mit der Kubernetes-API verfügen. Die Aktualisierung des Kubelets ist die wichtigste Komponente der Kubernetes-Funktionalität. Durch die Automatisierung von Betriebssystem-, Kernel- und lokal installierten Anwendungspatches wird der Wartungsaufwand bei der Skalierung jedoch reduziert.

Es wird empfohlen, das neueste für Amazon EKS optimierte Amazon Linux 2 oder Amazon EKS-optimierte Bottlerocket AMI für Ihr Node-Image zu verwenden. Karpenter verwendet automatisch das neueste verfügbare AMI, um neue Knoten im Cluster bereitzustellen. Verwaltete Knotengruppen aktualisieren das AMI während eines Knotengruppen-Updates, aktualisieren die AMI-ID jedoch nicht zum Zeitpunkt der Knotenbereitstellung.

Für verwaltete Knotengruppen müssen Sie die Startvorlage für Auto Scaling Group (ASG) mit neuen AMIs aktualisieren, IDs sobald diese für Patch-Versionen verfügbar sind. AMI-Nebenversionen (z. B. 1.23.5 bis 1.24.3) werden in der EKS-Konsole und API als Upgrades für die Knotengruppe verfügbar sein. Patch-Release-Versionen (z. B. 1.23.5 bis 1.23.6) werden nicht als Upgrades für die Knotengruppen angeboten. Wenn Sie Ihre Knotengruppe mit AMI-Patch-Releases auf dem neuesten Stand halten möchten, müssen Sie eine neue Version der Startvorlage erstellen und die Knotengruppe Instances durch die neue AMI-Version ersetzen lassen.

Sie können das neueste verfügbare AMI auf dieser Seite finden oder die AWS-CLI verwenden.

```
aws ssm get-parameter \
  --name /aws/service/eks/optimized-ami/1.24/amazon-linux-2/recommended/image_id \
  --query "Parameter.Value" \
  --output text
```

Verwenden Sie mehrere EBS-Volumes für Container

EBS-Volumes verfügen über ein input/output (I/O-) Kontingent, das auf dem Volumetyp (z. B. gp3) und der Größe der Festplatte basiert. Wenn sich Ihre Anwendungen ein einzelnes EBS-Root-Volume mit dem Host teilen, kann dies das Festplattenkontingent für den gesamten Host erschöpfen und andere Anwendungen dazu veranlassen, auf verfügbare Kapazität zu warten. Anwendungen schreiben auf die Festplatte, wenn sie Dateien auf ihre Overlay-Partition schreiben, ein lokales Volume vom Host mounten und je nach verwendetem Logging-Agent auch, wenn sie sich bei der Standardausgabe (STDOUT) anmelden.

Um eine Überlastung der I/O Festplatte zu vermeiden, sollten Sie ein zweites Volume in den Container-State-Ordner einhängen (z. B. /run/containerd), separate EBS-Volumes als Workload-Speicher verwenden und unnötiges lokales Logging deaktivieren.

Um mithilfe von eksctl ein zweites Volume in Ihre EC2-Instances einzubinden, können Sie eine Knotengruppe mit dieser Konfiguration verwenden:

```
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"
```

Wenn Sie Terraform zur Bereitstellung Ihrer Knotengruppen verwenden, sehen Sie sich die Beispiele in EKS Blueprints for Terraform an. Wenn Sie Karpenter zur Bereitstellung von Knoten verwenden, können Sie diese `blockDeviceMappings`zusammen mit Knoten-Benutzerdaten verwenden, um zusätzliche Volumes hinzuzufügen.

Um ein EBS-Volume direkt auf Ihrem Pod zu mounten, sollten Sie den AWS EBS CSI-Treiber verwenden und ein Volume mit einer Speicherklasse verwenden.

```
---
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
```

Vermeiden Sie Instances mit niedrigen EBS-Attach-Limits, wenn Workloads EBS-Volumes verwenden

EBS ist eine der einfachsten Möglichkeiten, Workloads persistenten Speicher zur Verfügung zu stellen, bringt aber auch Einschränkungen bei der Skalierbarkeit mit sich. Jeder Instance-Typ hat eine maximale Anzahl von EBS-Volumes, die angehängt werden können. Workloads müssen angeben, auf welchen Instance-Typen sie ausgeführt werden sollen, und die Anzahl der Replikate auf einer einzelnen Instance mit Kubernetes-Taints begrenzen.

Deaktivieren Sie unnötiges Logging auf der Festplatte

Vermeiden Sie unnötige lokale Protokollierung, indem Sie Ihre Anwendungen nicht mit Debug-Protokollierung in der Produktion ausführen und die Protokollierung deaktivieren, bei der häufig gelesen und auf die Festplatte geschrieben wird. Journald ist der lokale Protokollierungsdienst, der einen Protokollpuffer im Speicher hält und in regelmäßigen Abständen auf die Festplatte leert. Journald wird Syslog vorgezogen, das jede Zeile sofort auf der Festplatte protokolliert. Die

Deaktivierung von Syslog senkt auch den Gesamtspeicherbedarf und vermeidet komplizierte Regeln für die Protokollrotation. Um Syslog zu deaktivieren, können Sie Ihrer Cloud-Init-Konfiguration den folgenden Ausschnitt hinzufügen:

```
runcmd:
  - [ systemctl, disable, --now, syslog.service ]
```

Vorhandene Patch-Instanzen, wenn die Geschwindigkeit des Betriebssystem-Updates erforderlich ist

> **Important**
>
> Vorhandene Instanzen sollten nur bei Bedarf gepatcht werden. Amazon empfiehlt, die Infrastruktur als unveränderlich zu behandeln und Updates, die in niedrigeren Umgebungen verbreitet werden, genauso gründlich zu testen wie Anwendungen. Dieser Abschnitt gilt, wenn dies nicht möglich ist.

Die Installation eines Pakets auf einem vorhandenen Linux-Host dauert Sekunden, ohne containerisierte Workloads zu unterbrechen. Das Paket kann installiert und validiert werden, ohne die Instanz zu sperren, zu entleeren oder zu ersetzen.

Um eine Instanz zu ersetzen, müssen Sie zunächst eine neue Instanz erstellen, validieren und verteilen. AMIs Für die Instanz muss ein Ersatz erstellt werden, und die alte Instanz muss abgesperrt und entleert werden. Anschließend müssen Workloads auf der neuen Instanz erstellt, verifiziert und für alle Instanzen, die gepatcht werden müssen, wiederholt werden. Es dauert Stunden, Tage oder Wochen, um Instanzen sicher zu ersetzen, ohne die Workloads zu unterbrechen.

Amazon empfiehlt, eine unveränderliche Infrastruktur zu verwenden, die auf einem automatisierten, deklarativen System aufgebaut, getestet und beworben wird. Wenn Sie jedoch Systeme schnell patchen müssen, müssen Sie die vorhandenen Systeme patchen und sie ersetzen, sobald neue verfügbar AMIs sind. Aufgrund des großen Zeitunterschieds zwischen dem Patchen und dem Austausch von Systemen empfehlen wir die Verwendung von AWS Systems Manager Patch Manager, um das Patchen von Knoten bei Bedarf zu automatisieren.

Durch das Patchen von Knoten können Sie schnell Sicherheitsupdates bereitstellen und die Instances nach einem regelmäßigen Zeitplan austauschen, nachdem Ihr AMI aktualisiert wurde. Wenn Sie ein Betriebssystem mit einem schreibgeschützten Root-Dateisystem wie Flatcar Container

Linux oder Bottlerocket OS verwenden, empfehlen wir die Verwendung der Update-Operatoren, die mit diesen Betriebssystemen funktionieren. Der Flatcar Linux Update Operator und der Bottlerocket Update Operator starten die Instanzen neu, um die Knoten automatisch auf dem neuesten Stand zu halten.

Cluster-Dienste

Clusterdienste werden innerhalb eines EKS-Clusters ausgeführt, es handelt sich jedoch nicht um Benutzerarbeitslasten. Wenn Sie einen Linux-Server haben, müssen Sie häufig Dienste wie NTP, Syslog und eine Container-Runtime ausführen, um Ihre Workloads zu unterstützen. Clusterdienste sind ähnlich und unterstützen Dienste, die Sie bei der Automatisierung und dem Betrieb Ihres Clusters unterstützen. In Kubernetes werden diese normalerweise im Kube-System-Namespace ausgeführt und einige werden als ausgeführt. DaemonSets

Es wird erwartet, dass Clusterdienste eine hohe Verfügbarkeit aufweisen und sind bei Ausfällen und bei der Fehlerbehebung häufig von entscheidender Bedeutung. Wenn ein zentraler Clusterdienst nicht verfügbar ist, verlieren Sie möglicherweise den Zugriff auf Daten, die bei der Wiederherstellung helfen oder einen Ausfall verhindern können (z. B. hohe Festplattenauslastung). Sie sollten auf dedizierten Recheninstanzen wie einer separaten Knotengruppe oder AWS Fargate ausgeführt werden. Dadurch wird sichergestellt, dass die Cluster-Services auf gemeinsam genutzten Instances nicht durch Workloads beeinträchtigt werden, die möglicherweise skaliert werden oder mehr Ressourcen verbrauchen.

CoreDNS skalieren

Die Skalierung von CoreDNS hat zwei Hauptmechanismen. Reduzierung der Anzahl der Aufrufe des CoreDNS-Dienstes und Erhöhung der Anzahl der Replikate.

Reduzieren Sie externe Abfragen, indem Sie die Anzahl der Punkte verringern

Die Einstellung ndots gibt an, wie viele Punkte (auch bekannt als „Punkte“) in einem Domainnamen als ausreichend angesehen werden, um DNS-Abfragen zu vermeiden. Wenn Ihre Anwendung eine ndots-Einstellung von 5 (Standard) hat und Sie Ressourcen von einer externen Domain wie api.example.com (2 Punkte) anfordern, wird CoreDNS für jede in /etc/resolv.conf definierte Suchdomäne für eine spezifischere Domain abgefragt. Standardmäßig werden die folgenden Domänen durchsucht, bevor eine externe Anfrage gestellt wird.

```
api.example.<namespace>.svc.cluster.local
```

```
api.example.svc.cluster.local
api.example.cluster.local
api.example.<region>.compute.internal
```

Die `region` Werte `namespace` und werden durch Ihren Workloads-Namespace und Ihre Rechenregion ersetzt. Je nach Ihren Cluster-Einstellungen verfügen Sie möglicherweise über zusätzliche Suchdomänen.

Sie können die Anzahl der Anfragen an CoreDNS reduzieren, indem Sie die ndots-Option Ihres Workloads verringern oder Ihre Domain-Anfragen vollständig qualifizieren, indem Sie ein Trailing hinzufügen. (z. B.). `api.example.com.` Wenn Ihr Workload über DNS eine Verbindung zu externen Diensten herstellt, empfehlen wir, ndots auf 2 zu setzen, damit Workloads keine überflüssigen Cluster-DNS-Abfragen innerhalb des Clusters durchführen. Sie können einen anderen DNS-Server und eine andere Suchdomäne einrichten, wenn für den Workload kein Zugriff auf Dienste innerhalb des Clusters erforderlich ist.

```
spec:
  dnsPolicy: "None"
  dnsConfig:
    options:
      - name: ndots
        value: "2"
      - name: edns0
```

Wenn Sie Ndots auf einen zu niedrigen Wert herabsetzen oder die Domänen, zu denen Sie eine Verbindung herstellen, nicht genügend Spezifität aufweisen (einschließlich Trailing.), ist es möglich, dass DNS-Lookups fehlschlagen. Stellen Sie sicher, dass Sie testen, wie sich diese Einstellung auf Ihre Workloads auswirkt.

CoreDNS horizontal skalieren

CoreDNS-Instanzen können skaliert werden, indem der Bereitstellung zusätzliche Replikate hinzugefügt werden. Es wird empfohlen, NodeLocal DNS oder den Cluster Proportional Autoscaler zu verwenden, um CoreDNS zu skalieren.

NodeLocal DNS erfordert die Ausführung einer Instanz pro Knoten — als Ganzes DaemonSet —, was mehr Rechenressourcen im Cluster erfordert, aber dadurch werden fehlgeschlagene DNS-Anfragen vermieden und die Antwortzeit für DNS-Abfragen im Cluster verringert. Der proportionale Cluster-Autoscaler skaliert CoreDNS basierend auf der Anzahl der Knoten oder Kerne im Cluster. Dies ist kein direkter Zusammenhang mit Anforderungsabfragen, kann aber je nach Arbeitslast und

Clustergröße nützlich sein. Die proportionale Standardskalierung besteht darin, für jeweils 256 Kerne oder 16 Knoten im Cluster ein zusätzliches Replikat hinzuzufügen — je nachdem, was zuerst eintritt.

Wenn Sie das CoreDNS EKS-Add-On verwenden, sollten Sie erwägen, die Autoscaling-Option zu aktivieren. Der CoreDNS-Autoscaler passt die Anzahl der CoreDNS-Replikate dynamisch an, indem er die Anzahl der Knoten und die CPU-Kerne überwacht. Dabei wird eine Formel verwendet, die das Maximum von (Knoten÷16) oder (CPU-Kern÷256) verwendet und bei Bedarf sofort nach oben und schrittweise herunterskaliert, um die Stabilität aufrechtzuerhalten.

Skalieren Sie den Kubernetes Metrics Server vertikal

Der Kubernetes Metrics Server unterstützt die horizontale und vertikale Skalierung. Durch die horizontale Skalierung wird der Metrics Server zwar hochverfügbar sein, aber er kann nicht horizontal skaliert werden, um mehr Cluster-Metriken verarbeiten zu können. Sie müssen den Metrics Server auf der Grundlage seiner Empfehlungen vertikal skalieren, wenn Knoten und gesammelte Metriken dem Cluster hinzugefügt werden.

Der Metrics Server speichert die Daten, die er sammelt, aggregiert und bereitstellt, im Arbeitsspeicher. Wenn ein Cluster wächst, nimmt die Datenmenge zu, die der Metrics Server speichert. In großen Clustern benötigt der Metrics Server mehr Rechenressourcen als die in der Standardinstallation angegebene Speicher- und CPU-Reservierung. Sie können den Vertical Pod Autoscaler (VPA) oder Addon Resizer verwenden, um den Metrics Server zu skalieren. Der Addon Resizer skaliert vertikal proportional zu den Worker-Knoten und VPA skaliert basierend auf der CPU- und Speicherauslastung.

Dauer CoreDNS CoreDNS-Lameduck

Pods verwenden den `kube-dns` Dienst für die Namensauflösung. Kubernetes verwendet Destination NAT (DNAT), um den `kube-dns` Verkehr von Knoten zu CoreDNS-Backend-Pods umzuleiten. Während Sie die CoreDNS-Bereitstellung skalieren, werden die iptables-Regeln und -Ketten auf Knoten `kube-proxy` aktualisiert, um den DNS-Verkehr auf CoreDNS-Pods umzuleiten. Das Propagieren neuer Endpunkte beim Hochskalieren und das Löschen von Regeln beim Herunterskalieren von CoreDNS kann je nach Größe des Clusters zwischen 1 und 10 Sekunden dauern.

Diese Übertragungsverzögerung kann zu Fehlern bei der DNS-Suche führen, wenn ein CoreDNS-Pod beendet wird, die iptables-Regeln des Knotens jedoch nicht aktualisiert wurden. In diesem Szenario sendet der Knoten möglicherweise weiterhin DNS-Abfragen an einen beendeten CoreDNS-Pod.

Sie können Fehler bei der DNS-Suche reduzieren, indem Sie in Ihren CoreDNS-Pods eine Lameduck-Dauer festlegen. Im Lameduck-Modus wird CoreDNS weiterhin auf Anfragen während des Fluges antworten. Die Festlegung einer Lameduck-Dauer verzögert den Vorgang zum Herunterfahren von CoreDNS und gibt den Knoten die Zeit, die sie benötigen, um ihre iptables-Regeln und -Ketten zu aktualisieren.

Wir empfehlen, die Dauer CoreDNS CoreDNS-Lameduck auf 30 Sekunden einzustellen.

CoreDNS-Bereitschaftstest

Wir empfehlen die Verwendung `/ready` anstelle von `/health` für die Bereitschaftsprüfung von CoreDNS.

In Übereinstimmung mit der früheren Empfehlung, die Lameduck-Dauer auf 30 Sekunden festzulegen, um ausreichend Zeit für die Aktualisierung der iptables-Regeln des Nodes vor der Pod-Beendigung zu haben, stellt die Verwendung `/ready` statt `/health` für die CoreDNS-Bereitschaftsprobe sicher, dass der CoreDNS-Pod beim Start vollständig darauf vorbereitet ist, umgehend auf DNS-Anfragen zu antworten.

```
readinessProbe:
  httpGet:
    path: /ready
    port: 8181
    scheme: HTTP
```

Weitere Informationen zum CoreDNS Ready-Plugin finden Sie unter https://coredns. io/plugins/ready/

Agenten protokollieren und überwachen

Protokollierungs- und Überwachungsagenten können Ihre Cluster-Steuerebene erheblich belasten, da die Agenten den API-Server abfragen, um Protokolle und Metriken mit Workload-Metadaten anzureichern. Der Agent auf einem Knoten hat nur Zugriff auf die lokalen Knotenressourcen, um Dinge wie Container und Prozessnamen zu sehen. Bei der Abfrage des API-Servers können weitere Details wie der Name und die Labels der Kubernetes-Bereitstellung hinzugefügt werden. Dies kann bei der Fehlerbehebung äußerst hilfreich sein, sich jedoch nachteilig auf die Skalierung auswirken.

Da es so viele verschiedene Optionen für die Protokollierung und Überwachung gibt, können wir nicht für jeden Anbieter Beispiele zeigen. Bei Fluentbit empfehlen wir, Use_Kubelet zum Abrufen von Metadaten aus dem lokalen Kubelet statt vom Kubernetes API-Server zu aktivieren und auf

eine Zahl festzulegen, die wiederholte Aufrufe reduziert, wenn Daten `Kube_Meta_Cache_TTL` zwischengespeichert werden können (z. B. 60).

Die Skalierung von Überwachung und Protokollierung bietet zwei allgemeine Optionen:

- Integrationen deaktivieren
- Probenahme und Filterung

Das Deaktivieren von Integrationen ist oft keine Option, da Sie Protokoll-Metadaten verlieren. Dadurch wird das Problem der API-Skalierung behoben, es werden jedoch weitere Probleme auftreten, da die erforderlichen Metadaten nicht zur Verfügung stehen, wenn sie benötigt werden.

Durch Sampling und Filterung wird die Anzahl der gesammelten Metriken und Protokolle reduziert. Dadurch wird die Anzahl der Anfragen an die Kubernetes-API reduziert und der Speicherbedarf für die gesammelten Metriken und Protokolle reduziert. Durch die Reduzierung der Speicherkosten werden die Kosten für das Gesamtsystem gesenkt.

Die Möglichkeit, die Probennahme zu konfigurieren, hängt von der Agentsoftware ab und kann an verschiedenen Aufnahmepunkten implementiert werden. Es ist wichtig, die Probenahme so nah wie möglich am Agenten vorzunehmen, da dort wahrscheinlich die API-Serveraufrufe stattfinden. Wenden Sie sich an Ihren Anbieter, um mehr über den Sampling-Support zu erfahren.

Wenn Sie CloudWatch und CloudWatch Logs verwenden, können Sie Agentenfilterung mithilfe der in der Dokumentation beschriebenen Muster hinzufügen.

Um den Verlust von Protokollen und Messdaten zu vermeiden, sollten Sie Ihre Daten an ein System senden, das Daten im Falle eines Ausfalls am Empfangsendpunkt zwischenspeichern kann. Mit Fluentbit können Sie Amazon Kinesis Data Firehose verwenden, um Daten vorübergehend aufzubewahren, wodurch das Risiko einer Überlastung Ihres endgültigen Datenspeicherorts verringert werden kann.

Workloads

Workloads wirken sich darauf aus, wie groß Ihr Cluster skalieren kann. Workloads, die Kubernetes APIs stark nutzen, begrenzen die Gesamtmenge der Workloads, die Sie in einem einzelnen Cluster haben können. Es gibt jedoch einige Standardeinstellungen, die Sie ändern können, um die Last zu reduzieren.

Workloads in einem Kubernetes-Cluster haben Zugriff auf Funktionen, die in die Kubernetes-API integriert sind (z. B. Secrets und ServiceAccounts). Diese Funktionen sind jedoch nicht immer erforderlich und sollten deaktiviert werden, wenn sie nicht verwendet werden. Durch die Einschränkung des Workload-Zugriffs und der Abhängigkeit von der Kubernetes-Steuerebene wird die Anzahl der Workloads, die Sie im Cluster ausführen können, erhöht und die Sicherheit Ihrer Cluster verbessert, indem unnötiger Zugriff auf Workloads vermieden und Verfahren mit den geringsten Rechten implementiert werden. Weitere Informationen finden Sie in den bewährten Sicherheitsmethoden.

IPv6 Für Pod-Netzwerke verwenden

Sie können eine VPC nicht von IPv4 zu wechseln, IPv6 daher ist es wichtig, IPv6 sie vor der Bereitstellung eines Clusters zu aktivieren. Wenn Sie die Option IPv6 in einer VPC aktivieren, bedeutet das nicht, dass Sie sie verwenden müssen. Wenn Ihre Pods und Dienste sie verwenden, können IPv6 Sie weiterhin Traffic zu und von IPv4 Adressen weiterleiten. Weitere Informationen finden Sie in den Best Practices für EKS-Netzwerke.

Durch die Verwendung IPv6 in Ihrem Cluster werden einige der häufigsten Einschränkungen bei der Cluster- und Workload-Skalierung vermieden. IPv6 vermeidet die Erschöpfung von IP-Adressen, wenn Pods und Knoten nicht erstellt werden können, weil keine IP-Adresse verfügbar ist. Außerdem wird die Leistung pro Knoten verbessert, da Pods IP-Adressen schneller empfangen, da die Anzahl der ENI-Anhänge pro Knoten reduziert wird. Sie können eine ähnliche Knotenleistung erzielen, indem Sie den IPv4 Präfixmodus in der VPC-CNI verwenden. Sie müssen jedoch sicherstellen, dass in der VPC genügend IP-Adressen verfügbar sind.

Beschränken Sie die Anzahl der Dienste pro Namespace

Die maximale Anzahl von Diensten in einem Namespaces ist 5.000 und die maximale Anzahl von Diensten in einem Cluster ist 10.000. Wir empfehlen, die Anzahl der Dienste pro Namespace auf 500 zu beschränken, um Workloads und Dienste besser organisieren zu können, die Leistung zu steigern und kaskadierende Auswirkungen auf Ressourcen im Namespace-Bereich zu vermeiden.

Die Anzahl der IP-Tabellenregeln, die pro Knoten mit kube-proxy erstellt werden, wächst mit der Gesamtzahl der Dienste im Cluster. Das Generieren von Tausenden von IP-Tabellenregeln und das Routing von Paketen anhand dieser Regeln wirken sich auf die Leistung der Knoten aus und erhöhen die Netzwerklatenz.

Erstellen Sie Kubernetes-Namespaces, die eine einzige Anwendungsumgebung umfassen, sofern die Anzahl der Dienste pro Namespace unter 500 liegt. Dadurch wird die Serviceerkennung klein

genug gehalten, um Einschränkungen bei der Serviceerkennung zu vermeiden, und es kann Ihnen auch helfen, Konflikte bei der Benennung von Diensten zu vermeiden. Anwendungsumgebungen (z. B. dev, test, prod) sollten separate EKS-Cluster anstelle von Namespaces verwenden.

Elastic Load Balancer Balancer-Kontingente verstehen

Denken Sie bei der Erstellung Ihrer Dienste darüber nach, welche Art von Load Balancing Sie verwenden werden (z. B. Network Load Balancer (NLB) oder Application Load Balancer (ALB)). Jeder Load Balancer-Typ bietet unterschiedliche Funktionen und hat unterschiedliche Kontingente. Einige der Standardkontingente können angepasst werden, es gibt jedoch einige Kontingentobergrenzen, die nicht geändert werden können. Um Ihre Kontokontingente und Ihre Nutzung einzusehen, rufen Sie das Service-Kontingents-Dashboard in der AWS-Konsole auf.

Die ALB-Standardziele sind beispielsweise 1000. Wenn Sie einen Dienst mit mehr als 1000 Endpunkten haben, müssen Sie das Kontingent erhöhen oder den Dienst auf mehrere aufteilen oder Kubernetes ALBs Ingress verwenden. Die Standard-NLB-Ziele sind 3000, sind jedoch auf 500 Ziele pro AZ begrenzt. Wenn Ihr Cluster mehr als 500 Pods für einen NLB-Dienst ausführt, müssen Sie mehrere Pods verwenden AZs oder eine Erhöhung des Kontingentlimits beantragen.

Eine Alternative zur Verwendung eines an einen Dienst gekoppelten Load Balancers ist die Verwendung eines Ingress-Controllers. Der AWS Load Balancer-Controller kann ALBs für eingehende Ressourcen erstellen, Sie können jedoch erwägen, einen dedizierten Controller in Ihrem Cluster auszuführen. Ein Incluster-Ingress-Controller ermöglicht es Ihnen, mehrere Kubernetes-Services von einem einzigen Load Balancer aus verfügbar zu machen, indem Sie einen Reverse-Proxy in Ihrem Cluster ausführen. Controller verfügen über unterschiedliche Funktionen, z. B. Unterstützung für die Gateway-API, was je nach Anzahl und Umfang Ihrer Workloads Vorteile haben kann.

Verwenden Sie Route 53, Global Accelerator oder CloudFront

Um einen Service, der mehrere Load Balancer verwendet, als einen einzigen Endpunkt verfügbar zu machen, müssen Sie Amazon CloudFront, AWS Global Accelerator oder Amazon Route 53 verwenden, um alle Load Balancer als einen einzigen, kundenorientierten Endpunkt verfügbar zu machen. Jede Option bietet unterschiedliche Vorteile und kann je nach Bedarf separat oder zusammen verwendet werden.

Route 53 kann mehrere Load Balancer unter einem gemeinsamen Namen verfügbar machen und je nach zugewiesener Gewichtung Traffic an jeden von ihnen weiterleiten. Weitere Informationen zu

DNS-Gewichtungen finden Sie in der Dokumentation. Wie Sie diese mit dem externen Kubernetes-DNS-Controller implementieren, können Sie in der Dokumentation zum AWS Load Balancer Controller nachlesen.

Global Accelerator kann Workloads auf der Grundlage der angeforderten IP-Adresse an die nächstgelegene Region weiterleiten. Dies kann für Workloads nützlich sein, die in mehreren Regionen bereitgestellt werden, verbessert jedoch nicht das Routing zu einem einzelnen Cluster in einer einzelnen Region. Die Verwendung von Route 53 in Kombination mit dem Global Accelerator bietet zusätzliche Vorteile wie Zustandsprüfung und automatisches Failover, falls keine AZ verfügbar ist. Ein Beispiel für die Verwendung von Global Accelerator mit Route 53 finden Sie in diesem Blogbeitrag.

CloudFront kann mit Route 53 und Global Accelerator oder alleine verwendet werden, um den Verkehr an mehrere Ziele weiterzuleiten. CloudFront speichert Ressourcen, die von den ursprünglichen Quellen bereitgestellt werden, wodurch die Bandbreitenanforderungen je nach dem, was Sie bereitstellen, reduziert werden können.

Verwenden Sie EndpointSlices anstelle von Endpunkten

Wenn Sie Pods finden, die einem Service-Label entsprechen, sollten Sie EndpointSlicesanstelle von Endpoints verwenden. Endpunkte waren eine einfache Möglichkeit, Dienste in kleinem Umfang verfügbar zu machen, aber große Dienste, die automatisch skaliert werden oder über Updates verfügen, verursachen eine Menge Traffic auf der Kubernetes-Steuerebene. EndpointSlices verfügen über eine automatische Gruppierung, die beispielsweise Hinweise auf Topologie ermöglicht.

Nicht alle Controller verwenden EndpointSlices standardmäßig. Sie sollten Ihre Controller-Einstellungen überprüfen und sie bei Bedarf aktivieren. Für den AWS Load Balancer Controller sollten Sie das `--enable-endpoint-slices` optionale Flag zur Verwendung EndpointSlices aktivieren.

Verwenden Sie nach Möglichkeit unveränderliche und externe Geheimnisse

Das Kubelet speichert die aktuellen Schlüssel und Werte für die Secrets, die in Volumes für Pods auf diesem Knoten verwendet werden. Das Kubelet überwacht die Secrets, um Änderungen zu erkennen. Wenn der Cluster skaliert wird, kann sich die wachsende Anzahl von Watches negativ auf die Leistung des API-Servers auswirken.

Es gibt zwei Strategien, um die Anzahl der Zugriffe auf Secrets zu reduzieren:

- Für Anwendungen, die keinen Zugriff auf Kubernetes-Ressourcen benötigen, können Sie das automatische Mounten von Dienstkontogeheimnissen deaktivieren, indem Sie Token: false festlegen automountServiceAccount
- Wenn die Geheimnisse Ihrer Anwendung statisch sind und in future nicht geändert werden, markieren Sie das Geheimnis als unveränderlich. Das Kubelet überwacht die API nicht nach unveränderlichen Geheimnissen.

Um das automatische Mounten eines Dienstkontos auf Pods zu deaktivieren, können Sie die folgende Einstellung in Ihrem Workload verwenden. Sie können diese Einstellungen überschreiben, wenn für bestimmte Workloads ein Dienstkonto erforderlich ist.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app
automountServiceAccountToken: true
```

Überwachen Sie die Anzahl der Secrets im Cluster, bevor sie das Limit von 10.000 überschreitet. Mit dem folgenden Befehl können Sie die Gesamtzahl der Geheimnisse in einem Cluster anzeigen. Sie sollten dieses Limit mit Ihren Cluster-Überwachungstools überwachen.

```
kubectl get secrets -A | wc -l
```

Sie sollten die Überwachung so einrichten, dass ein Cluster-Administrator benachrichtigt wird, bevor dieses Limit erreicht wird. Erwägen Sie die Verwendung externer Secrets-Management-Optionen wie AWS Key Management Service (AWS KMS) oder Hashicorp Vault mit dem Secrets Store CSI-Treiber.

Beschränken Sie den Bereitstellungsverlauf

Pods können beim Erstellen, Aktualisieren oder Löschen langsam sein, da alte Objekte immer noch im Cluster nachverfolgt werden. Sie können die Anzahl `revisionHistoryLimit` der Bereitstellungen reduzieren, um ältere Versionen zu bereinigen ReplicaSets , wodurch die Gesamtanzahl der vom Kubernetes Controller Manager verfolgten Objekte reduziert wird. Das Standardlimit für die Historie von Bereitstellungen beträgt 10.

Wenn Ihr Cluster viele Auftragsobjekte durch CronJobs oder andere Mechanismen erstellt, sollten Sie diese `ttlSecondsAfterFinished`Einstellung verwenden, um alte Pods im Cluster automatisch

zu bereinigen. Dadurch werden erfolgreich ausgeführte Jobs nach einer bestimmten Zeit aus dem Jobverlauf entfernt.

enableServiceLinks Standardmäßig deaktivieren

Wenn ein Pod auf einem Knoten ausgeführt wird, fügt das Kubelet für jeden aktiven Dienst eine Reihe von Umgebungsvariablen hinzu. Linux-Prozesse haben eine maximale Größe für ihre Umgebung, die erreicht werden kann, wenn Sie zu viele Dienste in Ihrem Namespace haben. Die Anzahl der Dienste pro Namespace sollte 5.000 nicht überschreiten. Danach überschreitet die Anzahl der Dienstumgebungsvariablen die Shell-Limits, was dazu führt, dass Pods beim Start abstürzen.

Es gibt noch andere Gründe, warum Pods keine Dienstumgebungsvariablen für die Diensterkennung verwenden sollten. Namenskonflikte bei Umgebungsvariablen, undichte Dienstnamen und die Gesamtgröße der Umgebung sind nur einige Beispiele. Sie sollten CoreDNS verwenden, um Service-Endpunkte zu erkennen.

Beschränken Sie Webhooks mit dynamischer Zulassung pro Ressource

Zu den Webhooks mit dynamischer Zulassung gehören Webhooks für die Zulassung und Webhooks mit mutierender Zulassung. Sie sind API-Endpunkte, die nicht Teil der Kubernetes Control Plane sind und nacheinander aufgerufen werden, wenn eine Ressource an die Kubernetes-API gesendet wird. Jeder Webhook hat ein Standard-Timeout von 10 Sekunden und kann die Dauer einer API-Anfrage verlängern, wenn Sie mehrere Webhooks oder einen Timeout für einen dieser Webhooks haben.

Stellen Sie sicher, dass Ihre Webhooks hochverfügbar sind — insbesondere während eines AZ-Vorfalls — und dass die FailurePolicy so eingestellt ist, dass sie die Ressource ablehnt oder den Fehler ignoriert. Rufen Sie Webhooks nicht auf, wenn sie nicht benötigt werden, indem Sie zulassen, dass --dry-run kubectl-Befehle den Webhook umgehen.

```
apiVersion: admission.k8s.io/v1
kind: AdmissionReview
request:
  dryRun: False
```

Durch das Mutieren von Webhooks können Ressourcen häufig hintereinander verändert werden. Wenn Sie 5 mutierende Webhooks haben und 50 Ressourcen bereitstellen, speichert etcd alle Versionen jeder Ressource, bis die Komprimierung ausgeführt wird — alle 5 Minuten —, um alte Versionen der geänderten Ressourcen zu entfernen. In diesem Szenario, wenn etcd veraltete Ressourcen entfernt, werden 200 Ressourcenversionen von etcd entfernt. Je nach Größe der

Ressourcen kann dies beträchtlichen Speicherplatz auf dem etcd-Host beanspruchen, bis die Defragmentierung alle 15 Minuten ausgeführt wird.

Diese Defragmentierung kann zu Pausen in etcd führen, die andere Auswirkungen auf die Kubernetes-API und die Controller haben könnten. Sie sollten häufige Änderungen großer Ressourcen oder die Änderung von Hunderten von Ressourcen in schneller Folge vermeiden.

Vergleichen Sie Workloads in mehreren Clustern

Wenn Sie zwei Cluster haben, die eine ähnliche Leistung haben sollten, dies aber nicht tun, versuchen Sie, die Metriken zu vergleichen, um den Grund zu ermitteln.

Beispielsweise ist der Vergleich der Cluster-Latenz ein häufiges Problem. Dies wird normalerweise durch einen Unterschied im Volumen der API-Anfragen verursacht. Sie können die folgende CloudWatch LogInsight Abfrage ausführen, um den Unterschied zu verstehen.

```
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
```

Sie können zusätzliche Filter hinzufügen, um es einzugrenzen. Sie können sich z. B. auf alle Listenanfragen von konzentrieren`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
```

Theorie der Kubernetes-Skalierung

Nodes im Vergleich zur Abwanderungsrate

Wenn wir über die Skalierbarkeit von Kubernetes sprechen, beziehen wir uns oft darauf, wie viele Knoten es in einem einzelnen Cluster gibt. Interessanterweise ist dies selten die nützlichste Metrik, um die Skalierbarkeit zu verstehen. Beispielsweise würde ein 5.000-Knoten-Cluster mit einer großen,

aber festen Anzahl von Pods die Steuerungsebene nach der ersten Einrichtung nicht stark stress. Wenn wir jedoch einen Cluster mit 1.000 Knoten nehmen und versuchen würden, in weniger als einer Minute 10.000 kurzlebige Arbeitsplätze zu schaffen, würde das die Kontrollebene nachhaltig unter Druck setzen.

Es kann irreführend sein, einfach die Anzahl der Knoten zu verwenden, um die Skalierung zu verstehen. Es ist besser, in Bezug auf die Änderungsrate zu denken, die innerhalb eines bestimmten Zeitraums stattfindet (verwenden wir für diese Diskussion ein 5-Minuten-Intervall, da dies normalerweise standardmäßig in Prometheus-Abfragen verwendet wird). Lassen Sie uns untersuchen, warum wir, wenn wir das Problem anhand der Änderungsrate eingrenzen, eine bessere Vorstellung davon bekommen, was wir anpassen müssen, um den gewünschten Umfang zu erreichen.

Denken Sie in Abfragen pro Sekunde

Kubernetes verfügt über eine Reihe von Schutzmechanismen für jede Komponente — Kubelet, Scheduler, Kube Controller Manager und API-Server —, um zu verhindern, dass das nächste Glied in der Kubernetes-Kette überlastet wird. Kubelet verfügt beispielsweise über ein Flag, mit dem Aufrufe an den API-Server mit einer bestimmten Geschwindigkeit gedrosselt werden. Diese Schutzmechanismen werden im Allgemeinen, aber nicht immer, in Form von Abfragen ausgedrückt, die pro Sekunde erlaubt sind, oder QPS.

Bei der Änderung dieser QPS-Einstellungen ist große Vorsicht geboten. Die Beseitigung eines Engpasses, z. B. der Abfragen pro Sekunde auf einem Kubelet, hat Auswirkungen auf andere nachgelagerte Komponenten. Dies kann und wird das System ab einer bestimmten Geschwindigkeit überfordern. Daher ist es entscheidend, jeden Teil der Servicekette zu verstehen und zu überwachen, um Workloads auf Kubernetes erfolgreich zu skalieren.

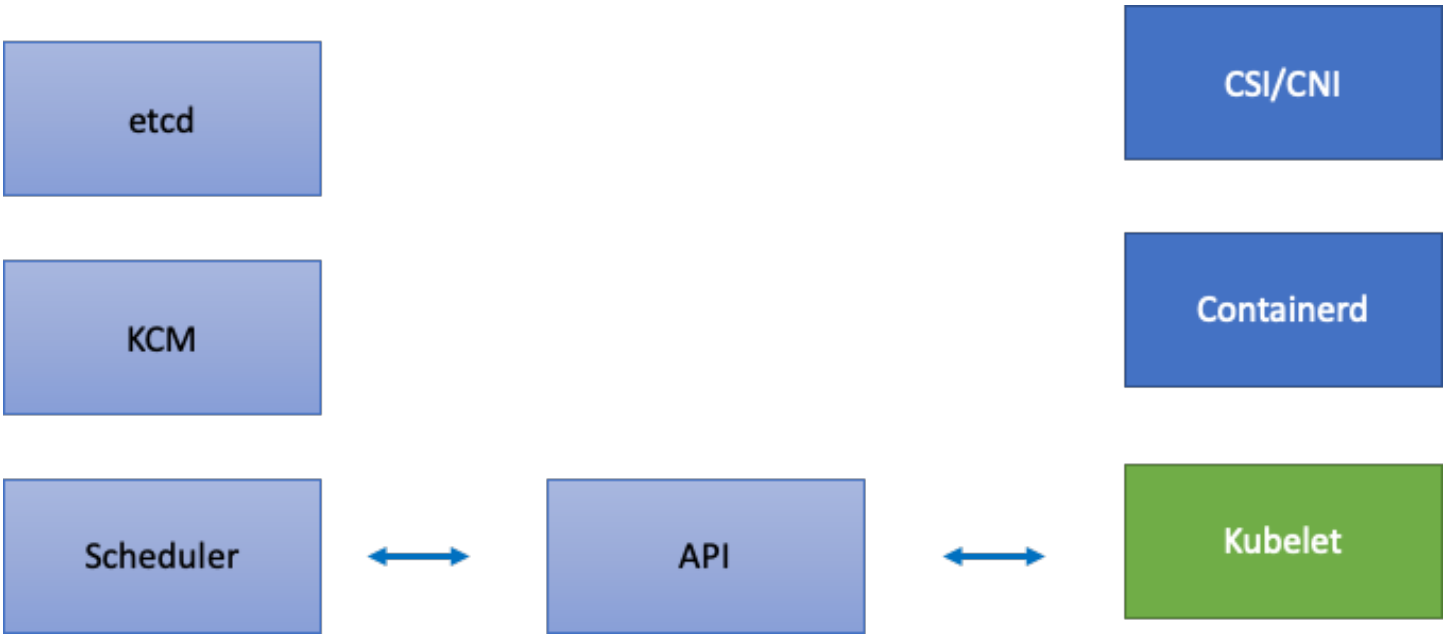
> **Note**
>
> Der API-Server hat ein komplexeres System mit der Einführung von API-Priorität und Fairness, auf die wir separat eingehen werden.

> **Note**
>
> Vorsicht, einige Metriken scheinen die richtige Wahl zu sein, messen aber in Wirklichkeit etwas anderes. `kubelet_http_inflight_requests`Bezieht sich beispielsweise

nur auf den Metrikserver in Kubelet, nicht auf die Anzahl der Anfragen von Kubelet an Apiserver-Anfragen. Dies könnte dazu führen, dass wir das QPS-Flag auf dem Kubelet falsch konfigurieren. Eine Abfrage von Audit-Logs für ein bestimmtes Kubelet wäre eine zuverlässigere Methode, um Metriken zu überprüfen.

Skalierung verteilter Komponenten

Da EKS ein verwalteter Service ist, teilen wir die Kubernetes-Komponenten in zwei Kategorien auf: AWS-verwaltete Komponenten, zu denen etcd, Kube Controller Manager und der Scheduler gehören (im linken Teil des Diagramms), und vom Kunden konfigurierbare Komponenten wie Kubelet, Container Runtime und die verschiedenen Operatoren, die AWS aufrufen, APIs wie die Netzwerk- und Speichertreiber (im rechten Teil des Diagramms). Wir belassen den API-Server in der Mitte, obwohl er von AWS verwaltet wird, da die Einstellungen für API-Priorität und Fairness von Kunden konfiguriert werden können.



Engpässe im Upstream- und Downstream-Bereich

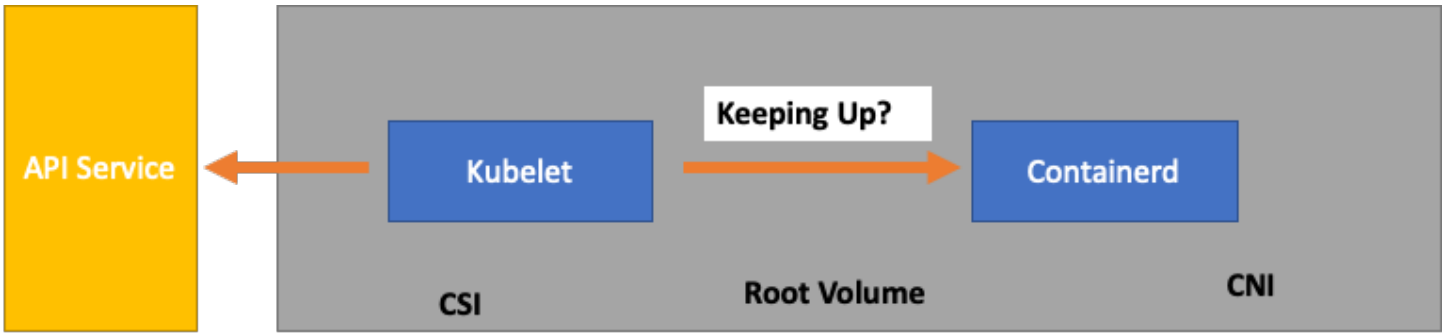
Bei der Überwachung der einzelnen Services ist es wichtig, Metriken in beide Richtungen zu betrachten, um nach Engpässen zu suchen. Lassen Sie uns am Beispiel von Kubelet lernen, wie das geht. Kubelet kommuniziert sowohl mit dem API-Server als auch mit der Container-Laufzeit. Wie und was müssen wir überwachen, um festzustellen, ob bei einer der Komponenten ein Problem auftritt?

Wie viele Pods pro Knoten

Wenn wir uns Skalierungszahlen ansehen, z. B. wie viele Pods auf einem Knoten ausgeführt werden können, könnten wir die 110 Pods pro Knoten, die Upstream unterstützt, für bare Münze nehmen.

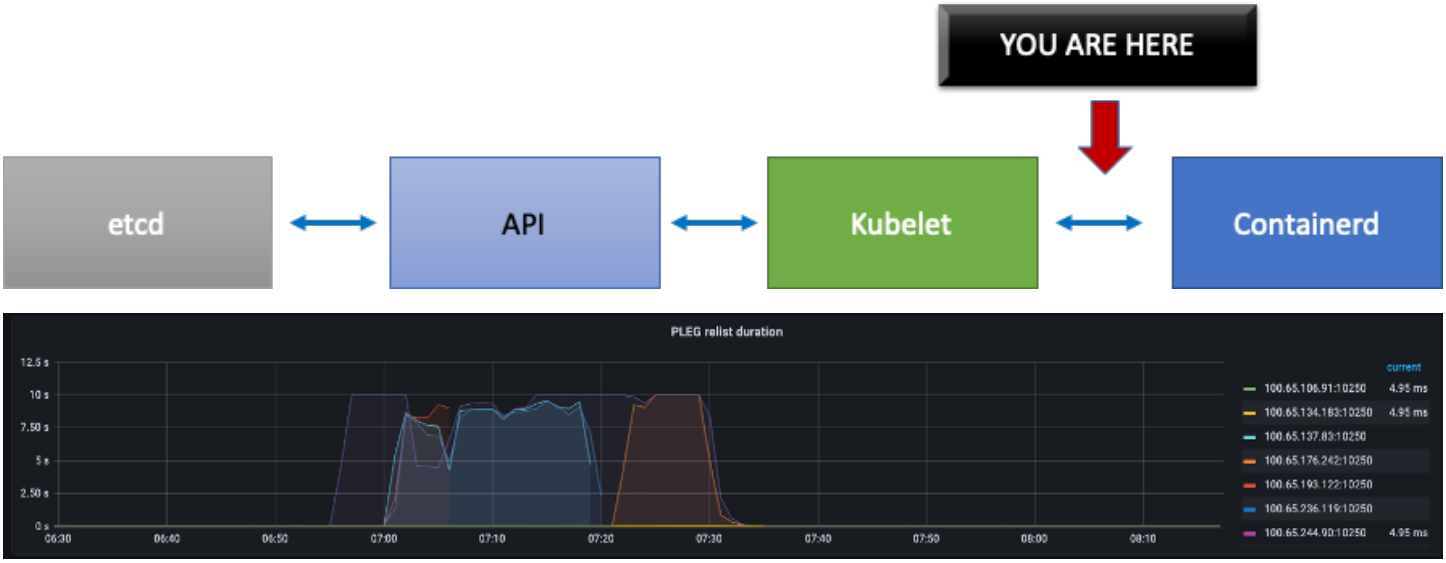
> **Note**
>
> https://kubernetes. io/docs/setup/best-practices/cluster-groß/

Ihr Workload ist jedoch wahrscheinlich komplexer als das, was in einem Skalierbarkeitstest in Upstream getestet wurde. Um sicherzustellen, dass wir die Anzahl der Pods, die wir in der Produktion ausführen möchten, bedienen können, sollten wir sicherstellen, dass das Kubelet mit der Containerd-Laufzeit „Schritt hält“.



Der Einfachheit halber erhält das Kubelet den Status der Pods aus der Container-Laufzeit (in unserem Fall Containerd). Was wäre, wenn wir zu viele Pods hätten, die ihren Status zu schnell ändern würden? Wenn die Änderungsrate zu hoch ist, kann es bei Anfragen [an die Container-Laufzeit] zu einem Timeout kommen.

> **Note**
>
> Kubernetes entwickelt sich ständig weiter, dieses Subsystem wird derzeit geändert. https://github.com/kubernetes/Erweiterungen/Probleme/3386



In der obigen Grafik sehen wir eine flache Linie, die angibt, dass wir gerade den Timeout-Wert für die Metrik für die Dauer der Generierung von Pod-Lebenszyklus-Ereignissen erreicht haben. Wenn Sie dies in Ihrem eigenen Cluster sehen möchten, könnten Sie die folgende PromQL-Syntax verwenden.

```
increase(kubelet_pleg_relist_duration_seconds_bucket{instance="$instance"}
[$__rate_interval])
```

Wenn wir dieses Timeout-Verhalten beobachten, wissen wir, dass wir den Knoten über das Limit gebracht haben, zu dem er fähig war. Wir müssen die Ursache für den Timeout beheben, bevor wir weitermachen können. Dies könnte erreicht werden, indem die Anzahl der Pods pro Knoten reduziert oder nach Fehlern gesucht wird, die möglicherweise zu einer hohen Anzahl von Wiederholungsversuchen führen (was sich auf die Abwanderungsrate auswirkt). Die wichtige Erkenntnis ist, dass Metriken der beste Weg sind, um zu verstehen, ob ein Knoten in der Lage ist, die Abwanderungsrate der zugewiesenen Pods im Vergleich zur Verwendung einer festen Anzahl zu bewältigen.

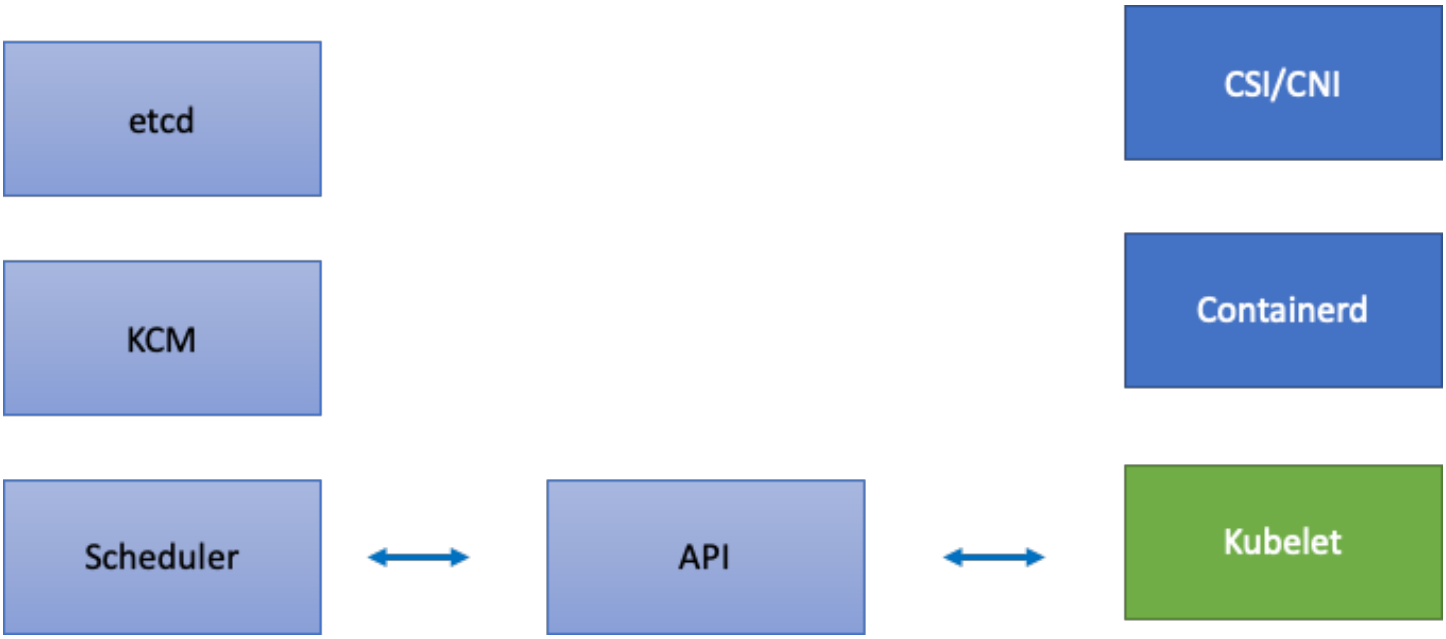
Skalieren nach Metriken

Das Konzept, Metriken zur Systemoptimierung zu verwenden, ist zwar alt, wird aber zu Beginn der Kubernetes-Reise oft übersehen. Anstatt uns auf bestimmte Zahlen zu konzentrieren (d. h. 110 Pods pro Knoten), konzentrieren wir uns darauf, die Metriken zu finden, die uns helfen, Engpässe in unserem System zu finden. Wenn wir die richtigen Schwellenwerte für diese Metriken kennen, können wir uns darauf verlassen, dass unser System optimal konfiguriert ist.

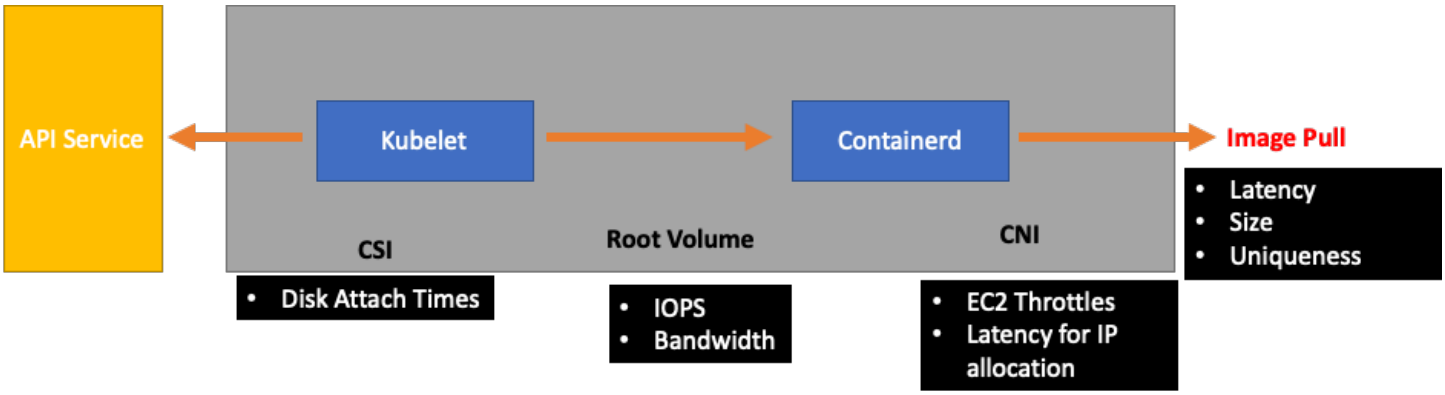
Die Auswirkungen von Änderungen

Ein häufiges Muster, das uns in Schwierigkeiten bringen könnte, besteht darin, uns auf den ersten Metrik- oder Protokollfehler zu konzentrieren, der verdächtig aussieht. Als wir sahen, dass das Kubelet früher eine Zeitüberschreitung hatte, konnten wir zufällige Dinge ausprobieren, wie zum Beispiel die Rate pro Sekunde zu erhöhen, die das Kubelet senden darf, usw. Es ist jedoch ratsam, sich das Gesamtbild von allem anzusehen, was hinter dem Fehler liegt, den wir zuerst finden. Nehmen Sie jede Änderung zielgerichtet und datengestützt vor.

Dem Kubelet nachgeschaltet wäre die Containerd-Runtime (Pod-Fehler), DaemonSets wie der Speichertreiber (CSI) und der Netzwerktreiber (CNI), die mit der EC2-API kommunizieren, usw.



Lassen Sie uns unser früheres Beispiel fortsetzen, bei dem Kubelet nicht mit der Laufzeit Schritt hält. Es gibt eine Reihe von Punkten, an denen wir einen Knoten so dicht zusammenpacken könnten, dass er Fehler auslöst.



Bei der Entwicklung der richtigen Knotengröße für unsere Workloads sind dies easy-to-overlook Signale, die das System möglicherweise unnötig belasten und so sowohl unseren Umfang als auch unsere Leistung einschränken.

Die Kosten unnötiger Fehler

Kubernetes-Controller zeichnen sich durch Wiederholungsversuche aus, wenn Fehler auftreten. Dies ist jedoch mit Kosten verbunden. Diese erneuten Versuche können den Druck auf Komponenten wie den Kube Controller Manager erhöhen. Es ist ein wichtiger Bestandteil von Skalentests, solche Fehler zu erkennen.

Wenn weniger Fehler auftreten, ist es einfacher, Probleme im System zu erkennen. Indem wir regelmäßig sicherstellen, dass unsere Cluster vor größeren Vorgängen (wie Upgrades) fehlerfrei sind, können wir die Fehlerbehebung bei unvorhergesehenen Ereignissen vereinfachen.

Wir erweitern unsere Sichtweise

In großen Clustern mit 1.000 Knoten möchten wir nicht einzeln nach Engpässen suchen. In PromQL können wir mithilfe einer Funktion namens topk die höchsten Werte in einem Datensatz ermitteln. K ist eine Variable und wir platzieren die Anzahl der gewünschten Elemente. Hier verwenden wir drei Knoten, um eine Vorstellung davon zu bekommen, ob alle Kubelets im Cluster gesättigt sind. Wir haben uns bis zu diesem Zeitpunkt mit der Latenz befasst. Lassen Sie uns nun sehen, ob das Kubelet Ereignisse verwirft.

```
topk(3, increase(kubelet_pleg_discard_events{}[$__rate_interval]))
```

Diese Aussage aufschlüsseln.

- Wir verwenden die Grafana-Variable`$__rate_interval`, um sicherzustellen, dass sie die vier benötigten Stichproben erhält. Dadurch wird ein komplexes Thema bei der Überwachung mit einer einfachen Variablen umgangen.
- `topk`geben Sie uns nur die besten Ergebnisse und die Zahl 3 begrenzt diese Ergebnisse auf drei. Dies ist eine nützliche Funktion für clusterweite Metriken.
- `{}`sagen Sie uns, dass es keine Filter gibt. Normalerweise würden Sie den Jobnamen unabhängig von der Scraping-Regel eingeben. Da diese Namen jedoch variieren, lassen wir das Feld leer.

Das Problem in zwei Hälften teilen

Um einen Engpass im System zu beheben, verfolgen wir den Ansatz, eine Metrik zu finden, die uns zeigt, dass vor- oder nachgelagerte Probleme vorliegen, da wir das Problem auf diese Weise in zwei Hälften teilen können. Dies wird auch ein zentraler Grundsatz dafür sein, wie wir unsere Metrikdaten anzeigen.

Ein guter Ausgangspunkt für diesen Prozess ist der API-Server, da wir so feststellen können, ob ein Problem mit einer Client-Anwendung oder der Control Plane vorliegt.

Überwachung der Kontrollebene

API-Server

Wenn Sie sich unseren API-Server ansehen, sollten Sie bedenken, dass eine seiner Funktionen darin besteht, eingehende Anfragen zu drosseln, um eine Überlastung der Kontrollebene zu verhindern. Was auf der Ebene des API-Servers wie ein Engpass erscheinen kann, schützt ihn möglicherweise vor ernsteren Problemen. Wir müssen die Vor- und Nachteile einer Erhöhung des Anforderungsvolumens im System berücksichtigen. Um zu entscheiden, ob die API-Serverwerte erhöht werden sollten, finden Sie hier eine kleine Auswahl der Dinge, auf die wir achten müssen:

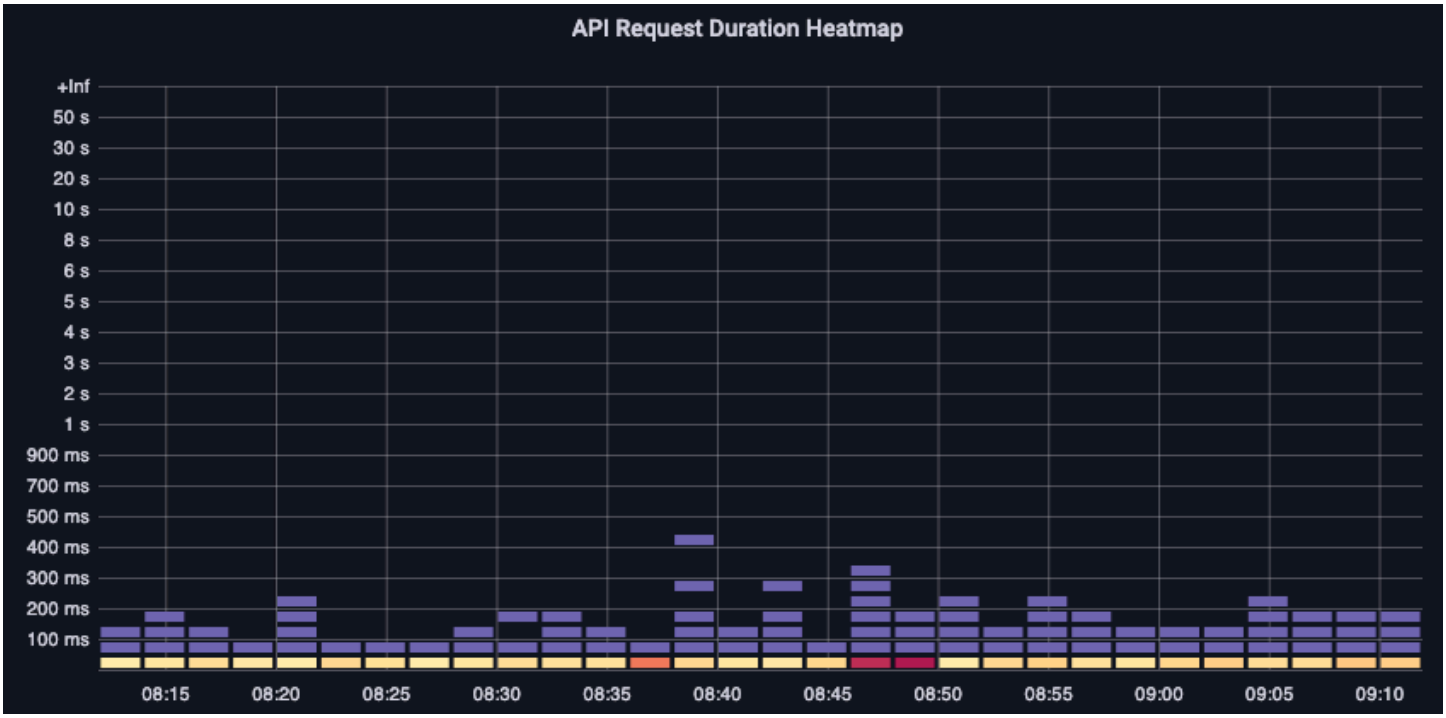
1. Wie lang ist die Latenz der Anfragen, die das System durchlaufen?
2. Ist diese Latenz der API-Server selbst oder etwas „Downstream“ wie etcd?
3. Ist die Warteschlangentiefe des API-Servers ein Faktor für diese Latenz?
4. Sind die APF-Warteschlangen (API Priority and Fairness) korrekt für die von uns gewünschten API-Aufrufmuster eingerichtet?

Wo ist das Problem?

Zunächst können wir anhand der Metrik für die API-Latenz ermitteln, wie lange es dauert, bis der API-Server Anfragen bearbeitet. Verwenden wir die folgende PromQL- und Grafana-Heatmap, um diese Daten anzuzeigen.

```
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**
>
> Eine ausführliche Beschreibung der Überwachung des API-Servers mit dem in diesem Artikel verwendeten API-Dashboard finden Sie im folgenden Blog



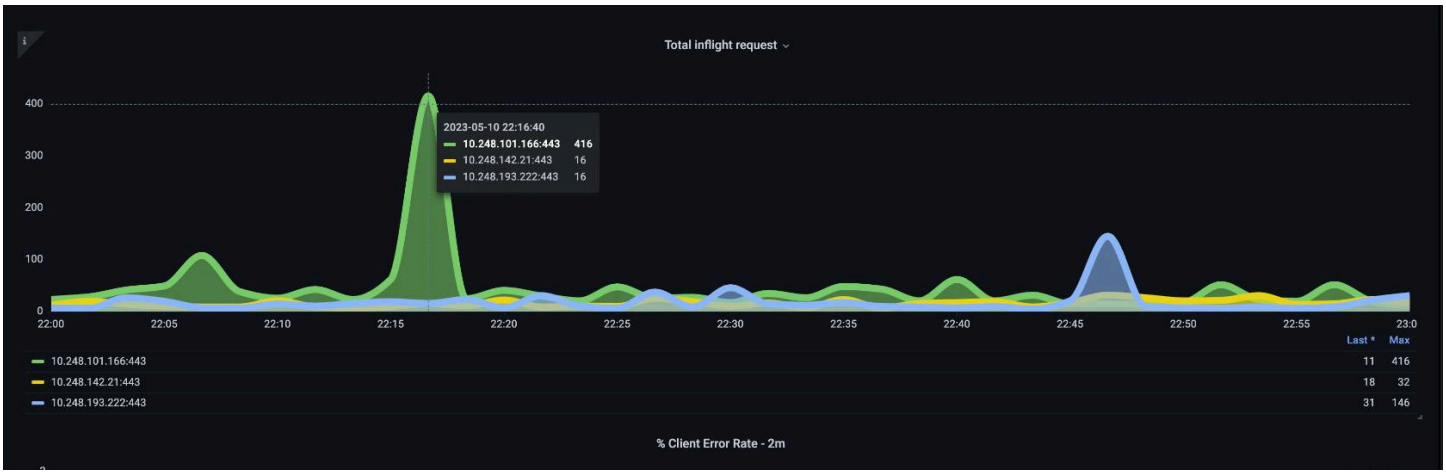
Diese Anfragen liegen alle unter einer Sekunde, was ein gutes Zeichen dafür ist, dass die Kontrollebene Anfragen zeitnah bearbeitet. Was aber, wenn das nicht der Fall wäre?

Das Format, das wir in der obigen API-Anforderungsdauer verwenden, ist eine Heatmap. Das Schöne am Heatmap-Format ist, dass es uns standardmäßig den Timeout-Wert für die API mitteilt (60 Sekunden). Was wir jedoch wirklich wissen müssen, ist, bei welchem Schwellenwert dieser Wert besorgniserregend sein sollte, bevor wir den Timeout-Schwellenwert erreichen. Für eine grobe Richtlinie darüber, was akzeptable Schwellenwerte sind, können wir das Upstream-Kubernetes-SLO verwenden, das Sie hier finden

> **Note**
>
> Haben Sie die Max-Funktion in dieser Anweisung bemerkt? Bei der Verwendung von Metriken, die mehrere Server aggregieren (standardmäßig zwei API-Server auf EKS), ist es wichtig, dass der Durchschnitt dieser Server nicht zusammengenommen wird.

Asymmetrische Verkehrsmuster

Was wäre, wenn ein API-Server [Pod] leicht und der andere stark ausgelastet wäre? Wenn wir den Durchschnitt dieser beiden Zahlen zusammen errechnen würden, könnten wir das Geschehene falsch interpretieren. Hier haben wir zum Beispiel drei API-Server, aber die gesamte Last befindet sich auf einem dieser API-Server. In der Regel sollte alles, was mehrere Server hat, wie Etcd- und API-Server, bei Investitionen, Skalierung und Leistung herausgenommen werden.



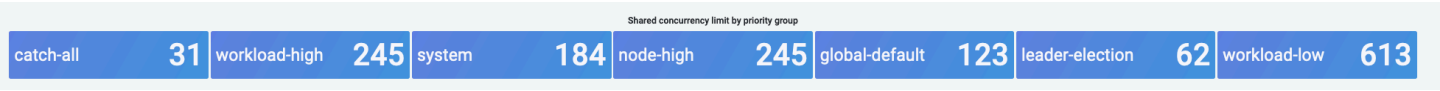
Mit der Umstellung auf API Priority and Fairness ist die Gesamtzahl der Anfragen im System nur ein Faktor, anhand dessen überprüft werden kann, ob der API-Server überbucht ist. Da das System jetzt mit einer Reihe von Warteschlangen arbeitet, müssen wir prüfen, ob eine dieser Warteschlangen voll ist und ob der Verkehr für diese Warteschlange unterbrochen wird.

Schauen wir uns diese Warteschlangen mit der folgenden Abfrage an:

```
max without(instance)(apiserver_flowcontrol_nominal_limit_seats{})
```

> **Note**
>
> Weitere Informationen zur Funktionsweise von API A&F finden Sie im folgenden Best-Practice-Leitfaden

Hier sehen wir die sieben verschiedenen Prioritätsgruppen, die standardmäßig im Cluster enthalten sind



Als Nächstes wollen wir sehen, wie viel Prozent dieser Prioritätsgruppe genutzt werden, damit wir nachvollziehen können, ob eine bestimmte Prioritätsstufe bereits ausgelastet ist. Eine Drosselung von Anfragen auf der Ebene „Niedrige Arbeitslast“ könnte wünschenswert sein, eine Senkung der Anzahl der Anfragen bei der Wahl von Führungskräften wäre jedoch nicht wünschenswert.

Das API-System Priority and Fairness (APF) bietet eine Reihe komplexer Optionen, von denen einige unbeabsichtigte Folgen haben können. Ein häufiges Problem, das wir in der Praxis beobachten, ist die Erhöhung der Warteschlangentiefe bis zu dem Punkt, an dem unnötige Latenz entsteht. Wir können dieses Problem anhand der `apiserver_flowcontrol_current_inqueue_request` Metrik überwachen. Wir können mit dem nach Stürzen suchen`apiserver_flowcontrol_rejected_requests_total`. Diese Metriken weisen einen Wert ungleich Null auf, wenn ein Bucket seine Parallelität überschreitet.



Eine Erhöhung der Warteschlangentiefe kann den API-Server zu einer erheblichen Latenzquelle machen und sollte mit Vorsicht erfolgen. Wir empfehlen, mit der Anzahl der erstellten Warteschlangen vorsichtig umzugehen. Die Anzahl der Shares in einem EKS-System beträgt beispielsweise 600. Wenn wir zu viele Warteschlangen erstellen, kann dies die Anzahl der Anteile in wichtigen Warteschlangen reduzieren, die den Durchsatz benötigen, wie z. B. die Warteschlange für die Wahl des Leiters oder die Systemwarteschlange. Wenn zu viele zusätzliche Warteschlangen erstellt werden, kann es schwieriger sein, die richtige Größe dieser Warteschlangen festzulegen.

Um uns auf eine einfache, wirkungsvolle Änderung zu konzentrieren, die Sie in APF vornehmen können, nehmen wir einfach Anteile aus nicht ausgelasteten Buckets und erhöhen die Größe der Buckets, die maximal ausgelastet sind. Durch eine intelligente Umverteilung der Anteile auf diese Buckets können Sie die Wahrscheinlichkeit von Kursverlusten verringern.

Weitere Informationen finden Sie unter Einstellungen für API-Priorität und Fairness im EKS Best Practices Guide.

API- und etcd-Latenz

Wie können wir anhand metrics/logs des API-Servers feststellen, ob ein Problem mit dem API-Server oder ein Problem mit dem API-Server oder eine Kombination aus beidem vorliegt? upstream/downstream Um dies besser zu verstehen, schauen wir uns an, wie API-Server und etcd zusammenhängen können und wie einfach es sein kann, Fehler am falschen System zu beheben.

In der folgenden Tabelle sehen wir die API-Serverlatenz, aber wir sehen auch, dass ein Großteil dieser Latenz mit dem etcd-Server korreliert, da die Balken in der Grafik den größten Teil der Latenz auf etcd-Ebene zeigen. Wenn es 15 Sekunden etcd-Latenz gibt und gleichzeitig 20 Sekunden API-Serverlatenz, dann liegt der Großteil der Latenz tatsächlich auf der etcd-Ebene.

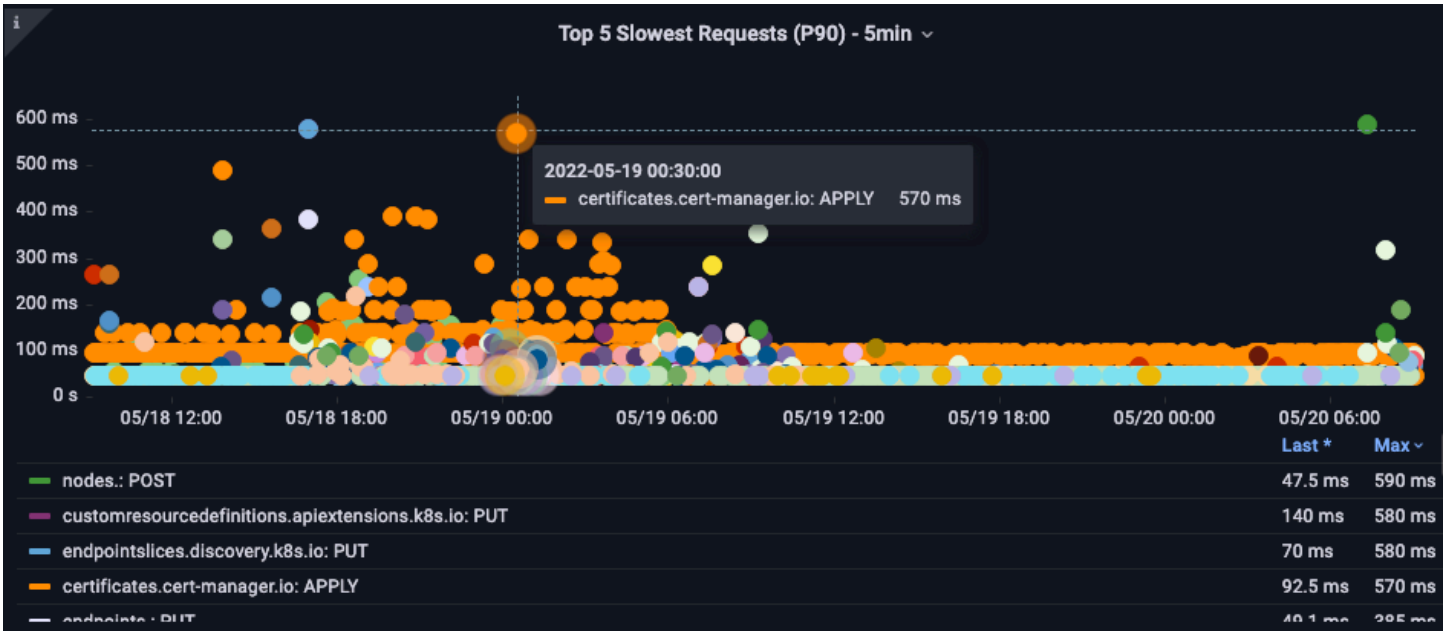
Wenn wir uns den gesamten Ablauf ansehen, erkennen wir, dass es ratsam ist, sich nicht nur auf den API-Server zu konzentrieren, sondern auch nach Signalen zu suchen, die darauf hinweisen, dass etcd unter Druck steht (d. h. die Anzahl der langsamen Anwendungszähler steigt). Die Fähigkeit, mit nur einem Blick schnell zum richtigen Problembereich zu gelangen, macht ein leistungsstarkes Dashboard aus.

> **Note**
>
> Das Dashboard im Abschnitt finden Sie unter https://github.com/RiskyAdventure/Dashboards/blob/main/api Troubleshooter.json



Probleme auf der Kontrollebene im Vergleich zur Kundenseite

In dieser Tabelle suchen wir nach den API-Aufrufen, deren Ausführung in diesem Zeitraum am meisten Zeit in Anspruch genommen hat. In diesem Fall sehen wir, dass eine benutzerdefinierte Ressource (CRD) eine APPLY-Funktion aufruft. Dies ist der latenteste Aufruf im Zeitrahmen von 05:40 Uhr.



Mit diesen Daten können wir eine Ad-Hoc PromQL- oder eine CloudWatch Insights-Abfrage verwenden, um während dieses Zeitraums LIST-Anfragen aus dem Audit-Log abzurufen, um zu sehen, um welche Anwendung es sich handeln könnte.

Die Quelle finden mit CloudWatch

Metriken eignen sich am besten, um den Problembereich zu finden, den wir untersuchen möchten, und um sowohl den Zeitrahmen als auch die Suchparameter des Problems einzugrenzen. Sobald wir diese Daten haben, wollen wir zu den Protokollen übergehen, um detailliertere Zeiten und Fehler zu finden. Zu diesem Zweck werden wir unsere Logs mithilfe CloudWatch von Logs Insights in Metriken umwandeln.

Um beispielsweise das oben genannte Problem zu untersuchen, verwenden wir die folgende CloudWatch Logs Insights-Abfrage, um UserAgent und RequestURI abzurufen, sodass wir herausfinden können, welche Anwendung diese Latenz verursacht.

> **Note**
>
> Es muss ein entsprechender Count verwendet werden, um ein normales List/Resync Verhalten auf einer Uhr zu verhindern.

```
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
```

Mithilfe dieser Abfrage haben wir zwei verschiedene Agenten gefunden, die eine große Anzahl von Listenoperationen mit hoher Latenz ausführen. Splunk und CloudWatch Agent. Mit den Daten können wir entscheiden, diesen Controller zu entfernen, zu aktualisieren oder durch ein anderes Projekt zu ersetzen.

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**
>
> Weitere Informationen zu diesem Thema finden Sie im folgenden Blog

Scheduler

Da die Instanzen der EKS-Kontrollebene in einem separaten AWS-Konto ausgeführt werden, können wir diese Komponenten nicht nach Metriken durchsuchen (der API-Server ist die Ausnahme). Da wir jedoch Zugriff auf die Auditprotokolle für diese Komponenten haben, können wir diese Protokolle in Metriken umwandeln, um festzustellen, ob eines der Subsysteme einen Skalierungsengpass verursacht. Lassen Sie uns mithilfe von CloudWatch Logs Insights sehen, wie viele ungeplante Pods sich in der Scheduler-Warteschlange befinden.

Nicht geplante Pods im Scheduler-Protokoll

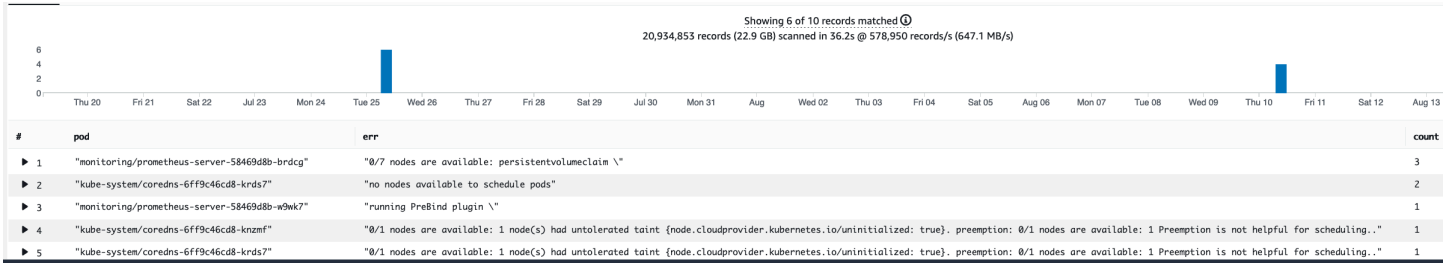
Wenn wir Zugriff auf das Scraping der Scheduler-Metriken direkt auf einem selbstverwalteten Kubernetes (wie Kops) hätten, würden wir das folgende PromQL verwenden, um den Scheduler-Backlog zu verstehen.

```
max without(instance)(scheduler_pending_pods)
```

Da wir in EKS keinen Zugriff auf die oben genannte Metrik haben, verwenden wir die unten stehende CloudWatch Logs Insights-Abfrage, um den Backlog zu überprüfen, indem wir überprüfen, wie viele Pods in einem bestimmten Zeitraum nicht ausgeplant werden konnten. Dann könnten wir uns eingehender mit den Nachrichten zu Spitzenzeiten befassen, um die Art des Engpasses zu verstehen. Zum Beispiel Knoten, die nicht schnell genug hochfahren, oder der Ratenbegrenzer im Scheduler selbst.

```
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
```

Hier sehen wir die Fehler des Schedulers, die besagen, dass der Pod nicht bereitgestellt wurde, weil das Speicher-PVC nicht verfügbar war.



> **Note**
>
> Die Auditprotokollierung muss auf der Steuerungsebene aktiviert sein, um diese Funktion zu aktivieren. Es ist auch eine bewährte Methode, die Aufbewahrung von Protokollen zu begrenzen, um die Kosten im Laufe der Zeit nicht unnötig in die Höhe zu treiben. Im

Folgenden finden Sie ein Beispiel für die Aktivierung aller Protokollierungsfunktionen mit dem EKSCTL-Tool.

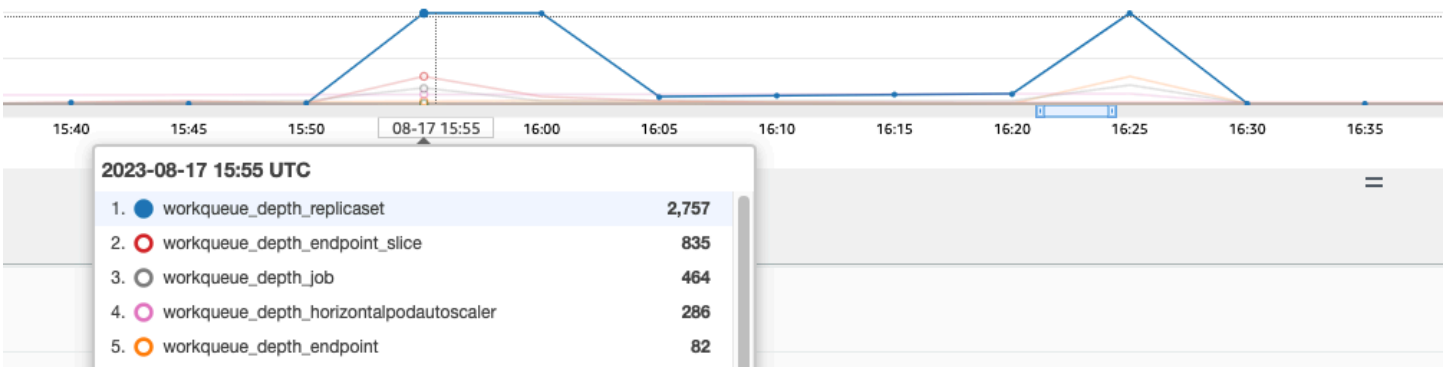
```
cloudWatch:
  clusterLogging:
    enableTypes: ["*"]
    logRetentionInDays: 10
```

Kube Controller-Manager

Kube Controller Manager hat, wie alle anderen Controller, Beschränkungen für die Anzahl der Operationen, die er gleichzeitig ausführen kann. Schauen wir uns an, was einige dieser Flags sind, indem wir uns eine KOPS-Konfiguration ansehen, in der wir diese Parameter einstellen können.

```
  kubeControllerManager:
    concurrentEndpointSyncs: 5
    concurrentReplicasetSyncs: 5
    concurrentNamespaceSyncs: 10
    concurrentServiceaccountTokenSyncs: 5
    concurrentServiceSyncs: 5
    concurrentResourceQuotaSyncs: 5
    concurrentGcSyncs: 20
    kubeAPIBurst: 30
    kubeAPIQPS: "20"
```

Diese Controller haben Warteschlangen, die sich in Zeiten hoher Fluktuation auf einem Cluster füllen. In diesem Fall sehen wir, dass der Replicaset-Set-Controller einen großen Rückstand in seiner Warteschlange hat.



Wir haben zwei verschiedene Möglichkeiten, mit einer solchen Situation umzugehen. Wenn wir selbst verwaltet würden, könnten wir einfach die Anzahl der gleichzeitigen Goroutinen erhöhen, was sich jedoch auf etcd auswirken würde, da mehr Daten im KCM verarbeitet würden. Die andere Option wäre, die Anzahl der Replicaset-Objekte, die bei der Bereitstellung verwendet werden, zu reduzieren, um die Anzahl der Replicaset-Objekte zu reduzieren, die wir zurücksetzen können, und so den Druck `.spec.revisionHistoryLimit` auf diesen Controller zu verringern.

```
spec:
  revisionHistoryLimit: 2
```

Andere Kubernetes-Funktionen können ein- oder ausgeschaltet werden, um den Druck in Systemen mit hoher Abwanderungsrate zu verringern. Wenn die Anwendung in unseren Pods beispielsweise nicht direkt mit der k8s-API kommunizieren muss, würde das Ausschalten des in diese Pods projizierten Secret die Auslastung verringern. ServiceaccountTokenSyncs Dies ist nach Möglichkeit der wünschenswertere Weg, um solche Probleme zu lösen.

```
kind: Pod
spec:
  automountServiceAccountToken: false
```

In Systemen, in denen wir keinen Zugriff auf die Metriken haben, können wir uns erneut die Protokolle ansehen, um Konflikte zu erkennen. Wenn wir die Anzahl der Anfragen sehen möchten, die pro Controller oder auf aggregierter Ebene verarbeitet werden, würden wir die folgende CloudWatch Logs Insights-Abfrage verwenden.

Gesamtvolumen, das vom KCM verarbeitet wurde

```
# 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
```

Die wichtigste Erkenntnis dabei ist, dass Sie bei der Untersuchung von Skalierbarkeitsproblemen jeden Schritt im Pfad (API, Scheduler, KCM usw.) überprüfen sollten, bevor Sie zur detaillierten Problembehebungsphase übergehen. In der Produktion werden Sie häufig feststellen, dass Anpassungen an mehr als einem Teil von Kubernetes erforderlich sind, damit das System optimal arbeiten kann. Es ist leicht, versehentlich Fehler zu beheben, was lediglich ein Symptom (z. B. ein Node-Timeout) eines viel größeren Engpasses ist.

USW

etcd verwendet eine dem Speicher zugeordnete Datei, um Schlüssel-Wert-Paare effizient zu speichern. Es gibt einen Schutzmechanismus, mit dem die Größe dieses verfügbaren Speicherplatzes üblicherweise auf die Grenzwerte von 2, 4 und 8 GB festgelegt wird. Weniger Objekte in der Datenbank bedeuten, dass weniger Bereinigungen von etcd durchgeführt werden müssen, wenn Objekte aktualisiert und ältere Versionen gelöscht werden müssen. Dieser Vorgang, bei dem alte Versionen eines Objekts entfernt werden, wird als Komprimierung bezeichnet. Nach einer Reihe von Verdichtungsvorgängen folgt ein weiterer Prozess, bei dem nutzbarer Speicherplatz wiederhergestellt wird. Dies wird als Defragmentierung bezeichnet. Dieser Vorgang erfolgt über einem bestimmten Schwellenwert oder nach einem festen Zeitplan.

Wir können einige benutzerbezogene Maßnahmen ergreifen, um die Anzahl der Objekte in Kubernetes zu begrenzen und so die Auswirkungen sowohl des Komprimierungs- als auch des Defragmentierungsprozesses zu verringern. Helm hält zum Beispiel einen hohen Wert.

`revisionHistoryLimit` Dadurch bleiben ältere Objekte ReplicaSets auf dem System, sodass Rollbacks durchgeführt werden können. Indem wir die Verlaufsgrenzen auf 2 reduzieren, können wir die Anzahl der Objekte (wie ReplicaSets) von zehn auf zwei reduzieren, was wiederum das System weniger belasten würde.

```
apiVersion: apps/v1
kind: Deployment
spec:
  revisionHistoryLimit: 2
```

Aus Sicht der Überwachung kann es hilfreich sein, zu überprüfen, ob dieser Defragmentierungsprozess die Ursache ist, wenn die Systemlatenz in einem festgelegten Muster, getrennt durch Stunden, auftritt. Wir können dies anhand von Protokollen überprüfen. CloudWatch

Wenn Sie die start/end Zeiten der Defragmentierung sehen möchten, verwenden Sie die folgende Abfrage:

```
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

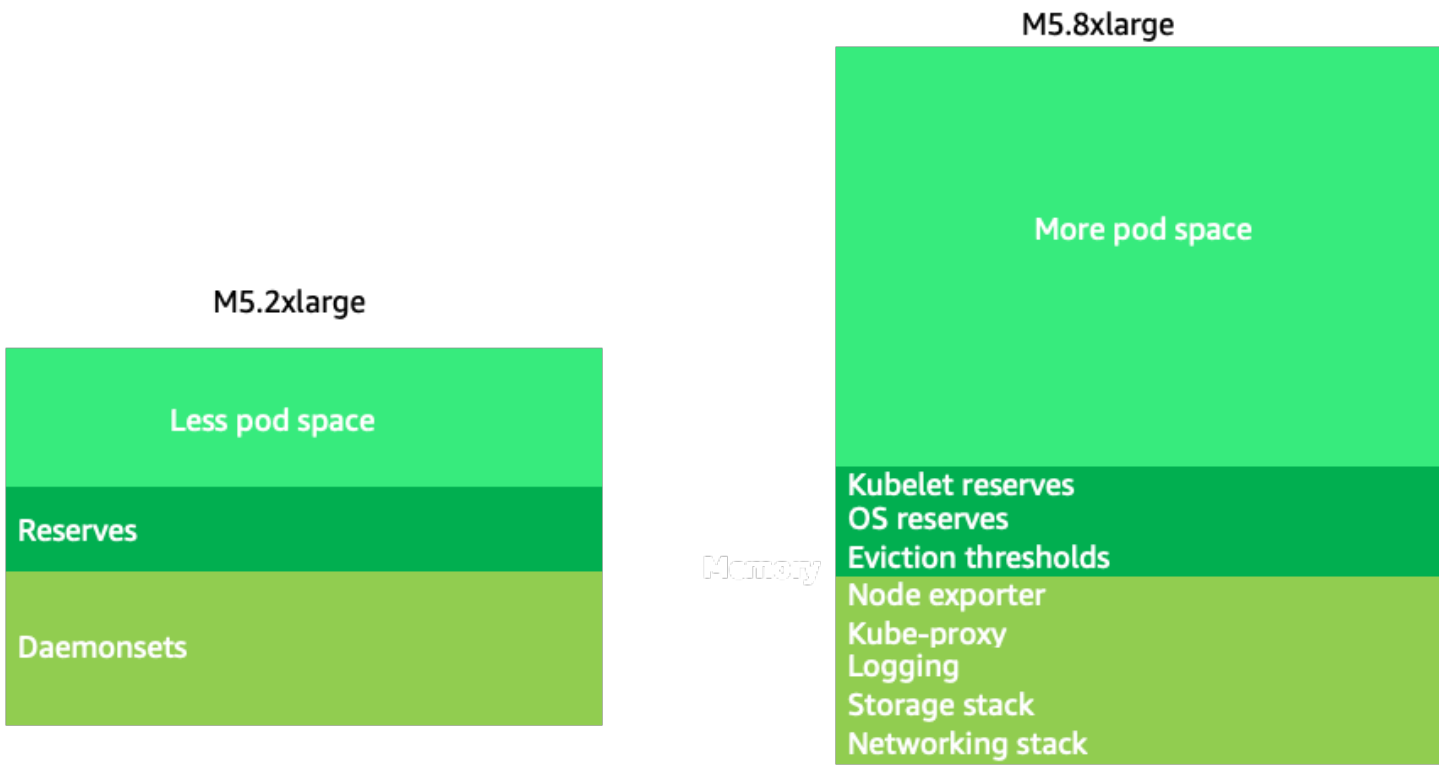
Effizienz von Knoten und Workloads

Wenn wir effizient mit unseren Workloads und Nodes umgehen, reduzieren wir complexity/cost gleichzeitig die Leistung und Skalierbarkeit. Bei der Planung dieser Effizienz sind viele Faktoren zu berücksichtigen, und es ist am einfachsten, Kompromisse im Vergleich zu einer Best-Practice-Einstellung für jede Funktion in Betracht zu ziehen. Lassen Sie uns diese Kompromisse im folgenden Abschnitt eingehend untersuchen.

Auswahl des Knotens

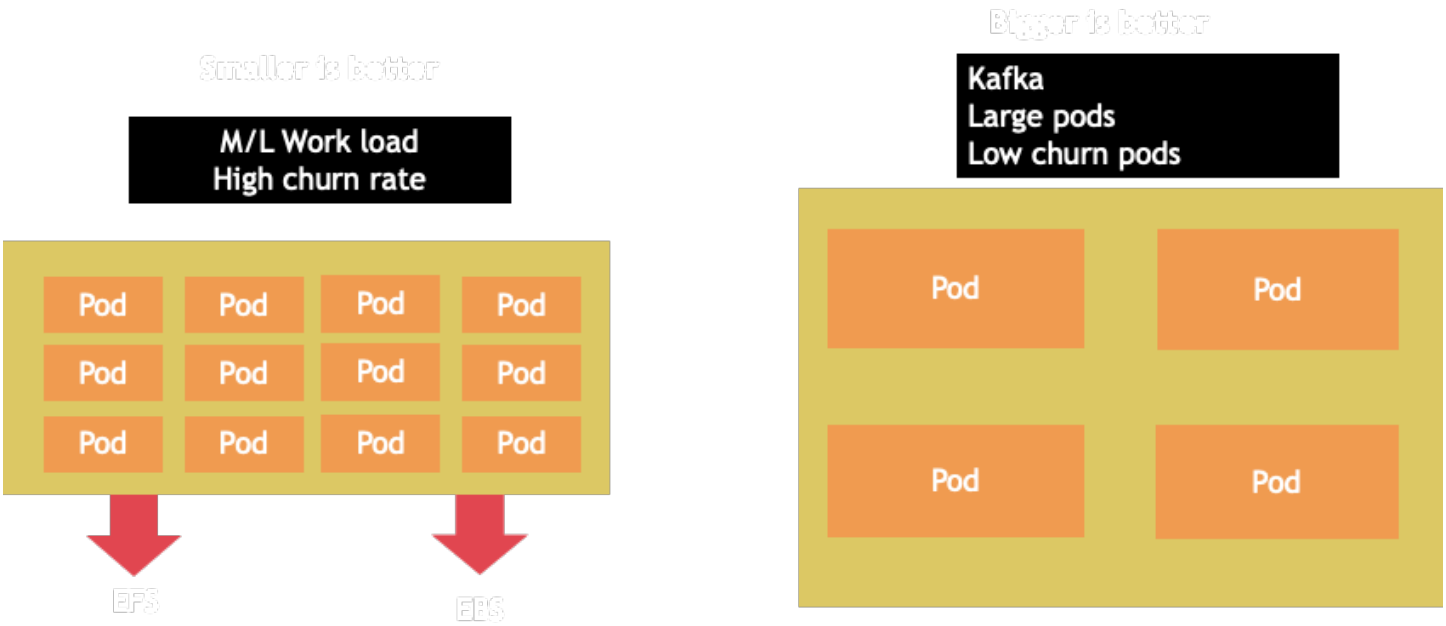
Durch die Verwendung etwas größerer Knoten (4-12x) wird der verfügbare Speicherplatz für die Ausführung von Pods erhöht, da dadurch der Prozentsatz des Knotens reduziert wird, der für „Overhead“ DaemonSetsund Reserven für Systemkomponenten verwendet wird. In der Abbildung unten sehen wir den Unterschied zwischen dem nutzbaren Speicherplatz auf einem 2xlarge-System und einem 8xlarge-System mit nur einer moderaten Anzahl von. DaemonSets

> **Note**
>
> Da k8s in der Regel horizontal skaliert wird, ist es für die meisten Anwendungen nicht sinnvoll, die Auswirkungen auf die Leistung von Knoten der NUMA-Größe zu berücksichtigen, weshalb ein Bereich unterhalb dieser Knotengröße empfohlen wird.



Große Knoten ermöglichen uns einen höheren Prozentsatz an nutzbarem Speicherplatz pro Knoten. Dieses Modell kann jedoch auf die Spitze getrieben werden, indem der Knoten mit so vielen Pods ausgestattet wird, dass dadurch Fehler auftreten oder der Knoten überlastet wird. Die Überwachung der Knotensättigung ist der Schlüssel zur erfolgreichen Verwendung größerer Knotengrößen.

Die Auswahl von Knoten ist selten ein one-size-fits-all Vorschlag. Oft ist es am besten, Workloads mit stark unterschiedlichen Abwanderungsraten auf verschiedene Knotengruppen aufzuteilen. Kleine Batch-Workloads mit einer hohen Abwanderungsrate würden am besten von der 4xlarge-Instance-Familie bedient werden, während eine umfangreiche Anwendung wie Kafka, die 8 vCPUs benötigt und eine niedrige Abwanderungsrate aufweist, besser von der 12xlarge-Familie bedient werden könnte.



> **Note**
>
> Ein weiterer Faktor, der bei sehr großen Knoten berücksichtigt werden muss, ist, dass CGROUPS die Gesamtzahl der vCPUs nicht vor der containerisierten Anwendung verbergen. Dynamische Laufzeiten können oft eine unbeabsichtigte Anzahl von Betriebssystem-Threads erzeugen, was zu einer Latenz führt, die schwer zu beheben ist. Für diese Anwendungen wird CPU-Pinning empfohlen. Für eine genauere Erläuterung des Themas schauen Sie sich bitte das folgende Video an https://www.youtube.com /watch? v= _ NqtfDy KAqg

Bin-Packen für Knoten

Regeln für Kubernetes und Linux

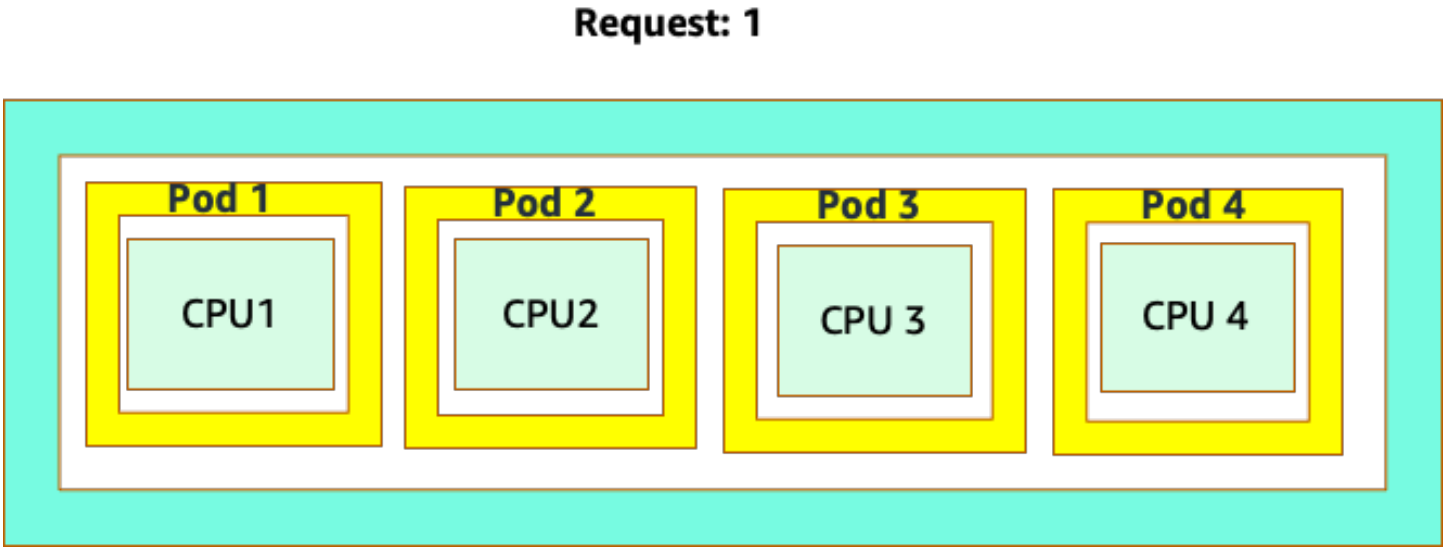
Es gibt zwei Regelwerke, die wir beim Umgang mit Workloads auf Kubernetes beachten müssen. Die Regeln des Kubernetes-Schedulers, der den Anforderungswert verwendet, um Pods auf einem

Knoten zu planen, und dann, was passiert, nachdem der Pod geplant wurde. Das ist der Bereich von Linux, nicht von Kubernetes.

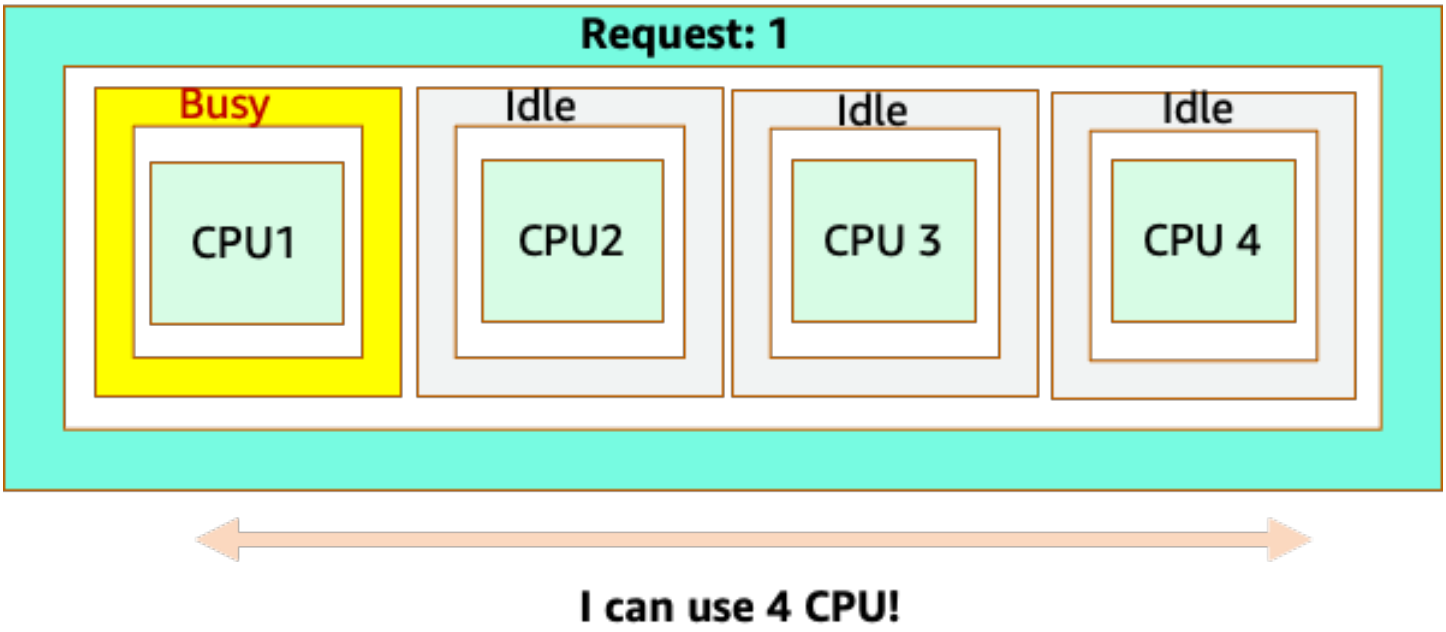
Sobald der Kubernetes-Scheduler fertig ist, übernimmt ein neues Regelwerk, der Linux Completely Fair Scheduler (CFS). Die wichtigste Erkenntnis ist, dass Linux CFS nicht das Konzept eines Kerns hat. Wir werden erörtern, warum das Denken in Kernen zu großen Problemen bei der Skalierung von Workloads führen kann.

In Kernen denken

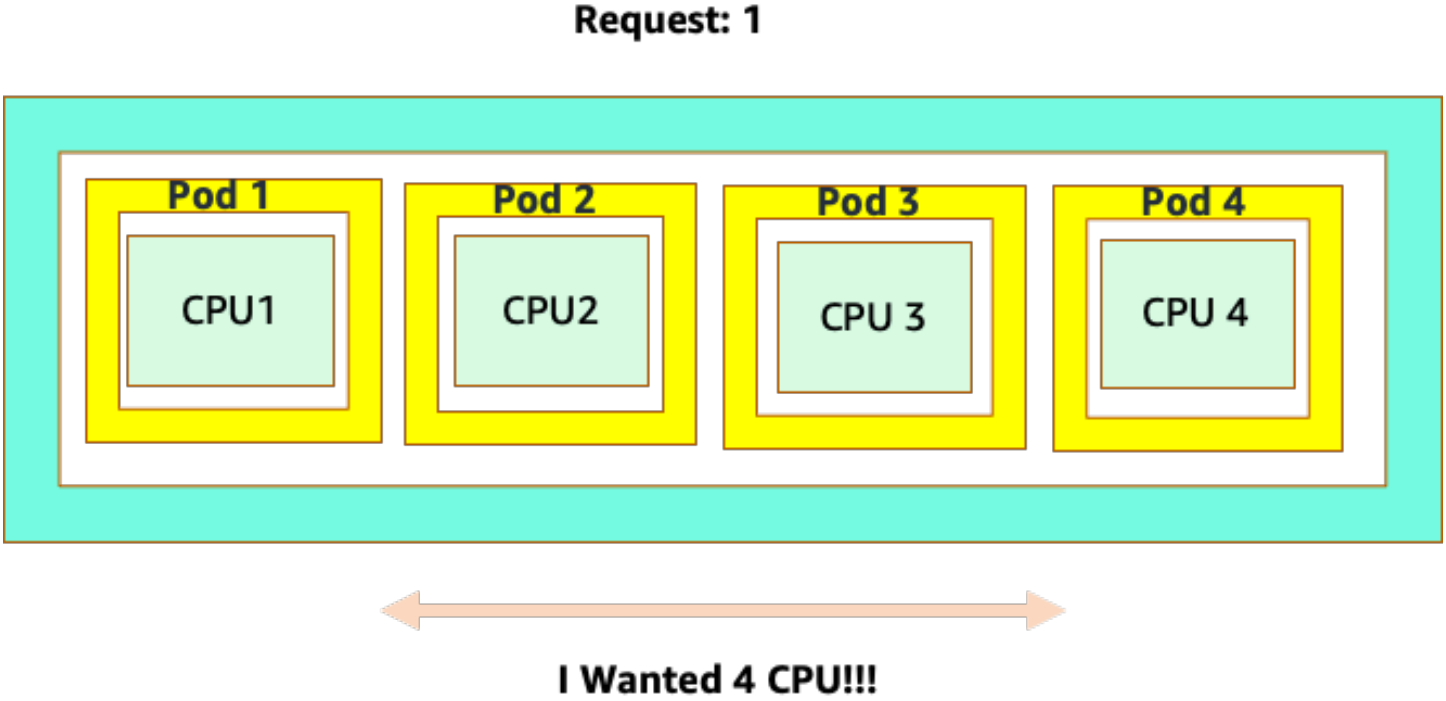
Die Verwirrung beginnt, weil der Kubernetes-Scheduler das Konzept der Kerne hat. Aus Sicht des Kubernetes-Schedulers würde der Knoten so aussehen, wenn wir uns einen Knoten mit 4 NGINX-Pods ansehen würden, von denen jeder die Anforderung eines Core-Sets hat.



Lassen Sie uns jedoch ein Gedankenexperiment durchführen, um zu untersuchen, wie unterschiedlich das aus einer Linux-CFS-Perspektive aussieht. Das Wichtigste, an das Sie denken sollten, wenn Sie das Linux-CFS-System verwenden, ist: Busy-Container (CGROUPS) sind die einzigen Container, die auf das Share-System angerechnet werden. In diesem Fall ist nur der erste Container ausgelastet, sodass er alle 4 Kerne auf dem Knoten verwenden darf.



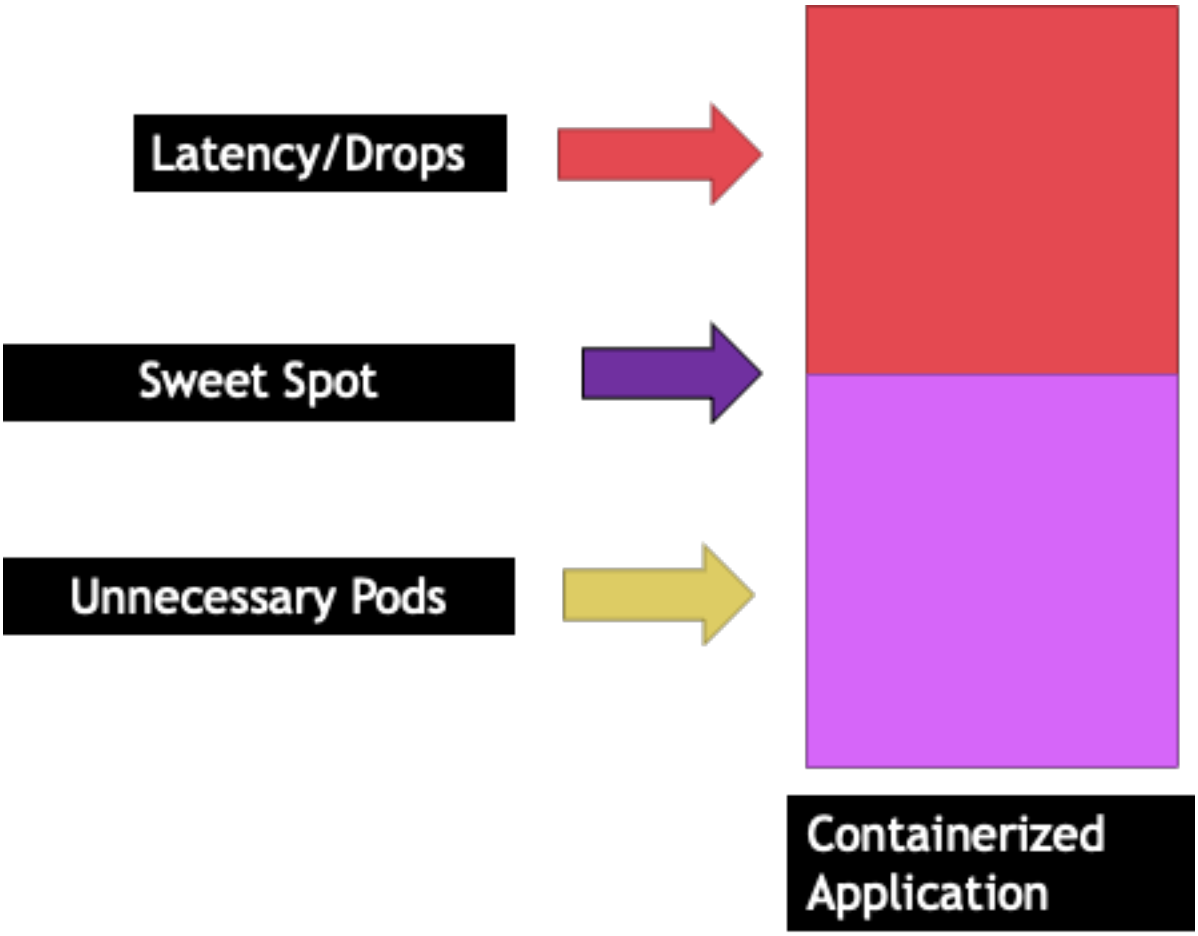
Warum ist das wichtig? Nehmen wir an, wir haben unsere Leistungstests in einem Entwicklungscluster durchgeführt, in dem eine NGINX-Anwendung der einzige ausgelastete Container auf diesem Knoten war. Wenn wir die App in die Produktion überführen, würde Folgendes passieren: Die NGINX-Anwendung benötigt 4 vCPU-Ressourcen, da jedoch alle anderen Pods auf dem Knoten ausgelastet sind, ist die Leistung unserer App eingeschränkt.



Diese Situation würde dazu führen, dass wir unnötig mehr Container hinzufügen müssten, weil wir nicht zulassen würden, dass unsere Anwendungen nach ihrem „`Sweet Spot“ skaliert werden. Lassen Sie uns dieses wichtige Konzept von a etwas `"sweet spot"` genauer untersuchen.

Richtige Dimensionierung der Anwendung

Jede Anwendung hat einen bestimmten Punkt, an dem sie keinen Verkehr mehr aufnehmen kann. Eine Überschreitung dieses Punktes kann die Verarbeitungszeiten verlängern und sogar den Datenverkehr verringern, wenn dieser Punkt deutlich überschritten wird. Dieser Wert wird als Sättigungspunkt der Anwendung bezeichnet. Um Skalierungsprobleme zu vermeiden, sollten wir versuchen, die Anwendung zu skalieren, bevor sie ihren Sättigungspunkt erreicht. Nennen wir diesen Punkt den Sweet Spot.



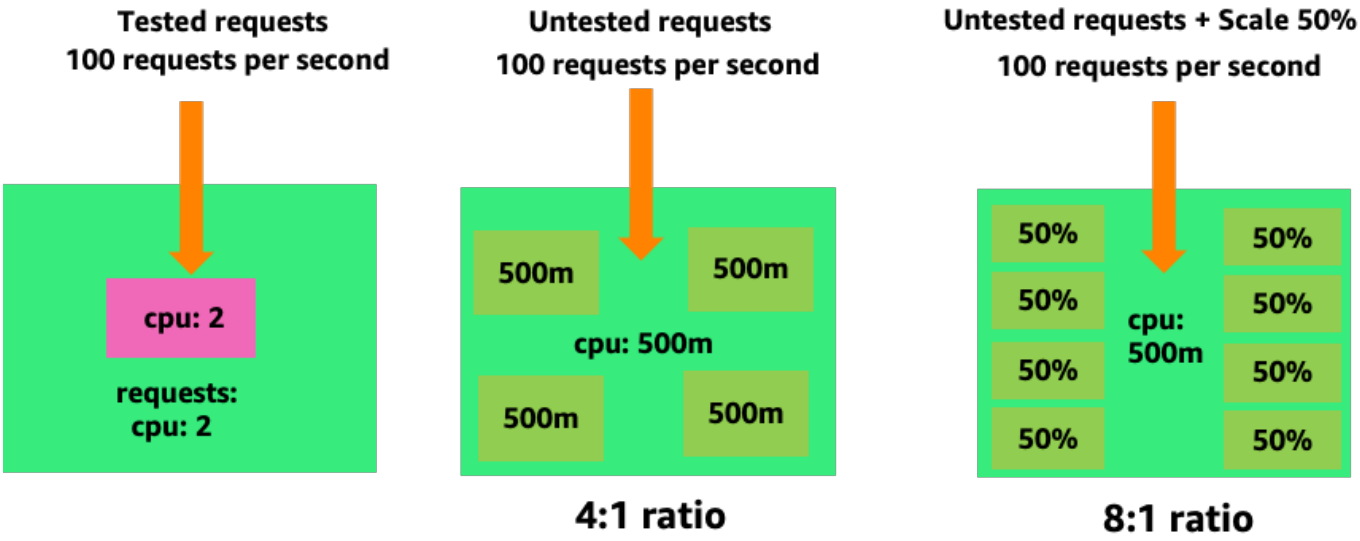
Wir müssen jede unserer Anwendungen testen, um herauszufinden, was für sie am besten geeignet ist. Hier wird es keine allgemeingültige Anleitung geben, da jede Anwendung anders ist. Während dieser Tests versuchen wir, die beste Metrik zu finden, die den Sättigungspunkt unserer Anwendungen angibt. Häufig werden Nutzungskennzahlen verwendet, um anzuzeigen, dass eine Anwendung ausgelastet ist. Dies kann jedoch schnell zu Skalierungsproblemen führen (wir werden

dieses Thema in einem späteren Abschnitt ausführlich behandeln). Sobald wir diesen „Sweet Spot“ haben, können wir ihn nutzen, um unsere Workloads effizient zu skalieren.

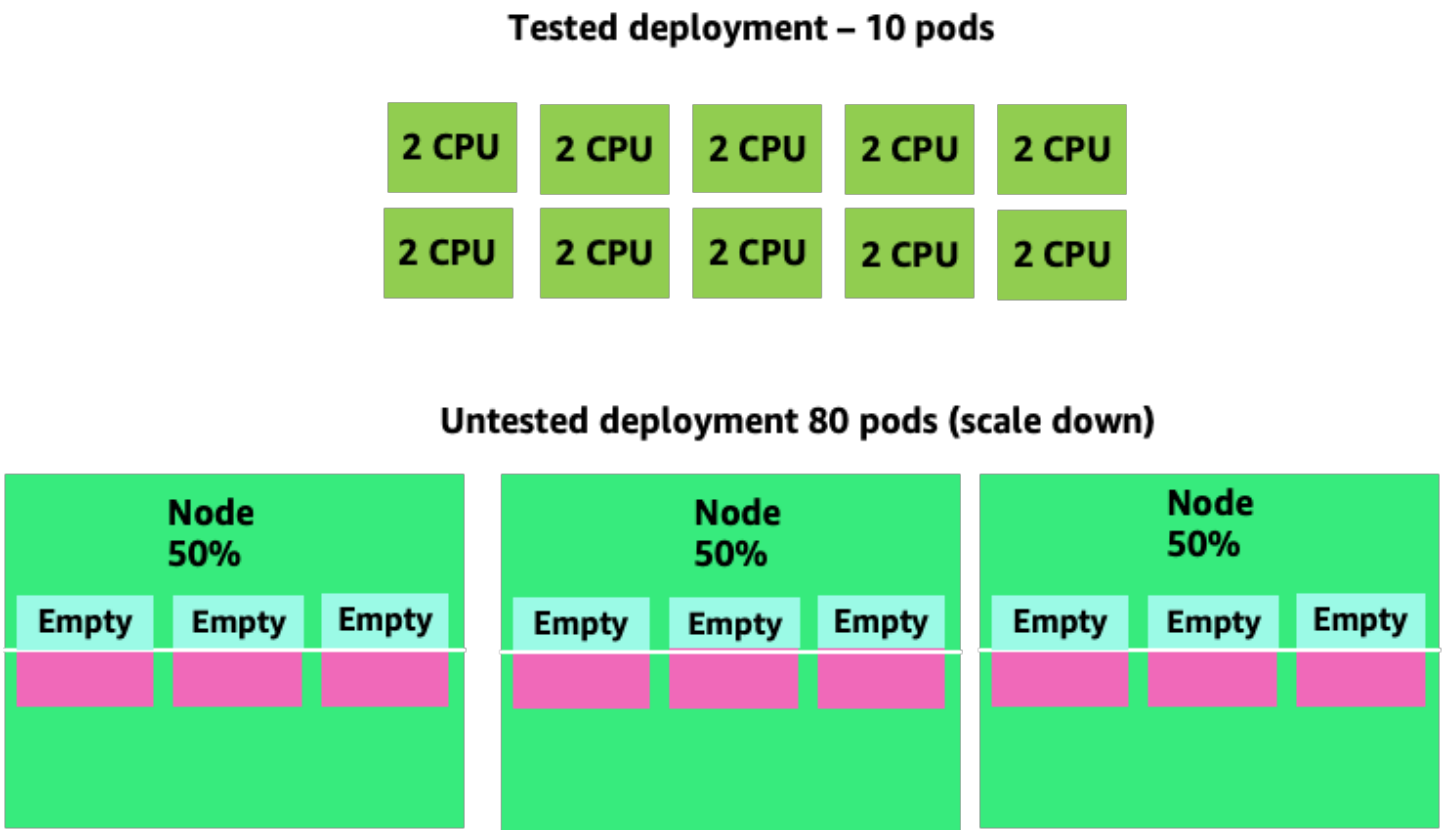
Was würde umgekehrt passieren, wenn wir weit vor dem Sweet Spot hochskalieren und unnötige Pods erstellen würden? Lassen Sie uns das im nächsten Abschnitt untersuchen.

Ausbreitung der Hülsen

Schauen wir uns das erste Beispiel auf der linken Seite an, um zu sehen, wie das Erstellen unnötiger Pods schnell außer Kontrolle geraten kann. Die korrekte vertikale Skalierung dieses Containers beansprucht etwa zwei V CPUs Auslastung, wenn 100 Anfragen pro Sekunde bearbeitet werden. Wenn wir jedoch den Wert der Anfragen zu gering bereitstellen würden, indem wir Anfragen auf einen halben Kern setzen, bräuchten wir jetzt 4 Pods für jeden Pod, den wir tatsächlich benötigen. Verschärft wird dieses Problem noch dadurch, dass, wenn unser HPA auf den Standardwert von 50% CPU eingestellt wäre, diese Pods halb leer skaliert würden, sodass ein Verhältnis von 8:1 entsteht.

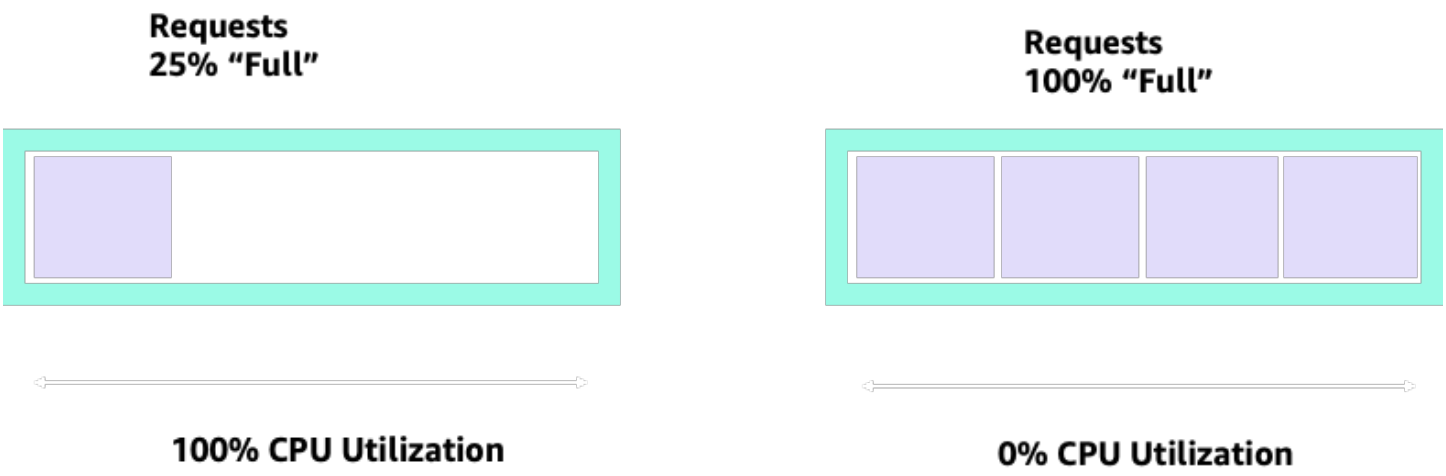


Wenn wir dieses Problem vergrößern, können wir schnell erkennen, wie es außer Kontrolle geraten kann. Eine Bereitstellung von zehn Pods, deren Sweet Spot falsch eingestellt war, könnte schnell zu 80 Pods und der zusätzlichen Infrastruktur führen, die für deren Betrieb erforderlich ist.



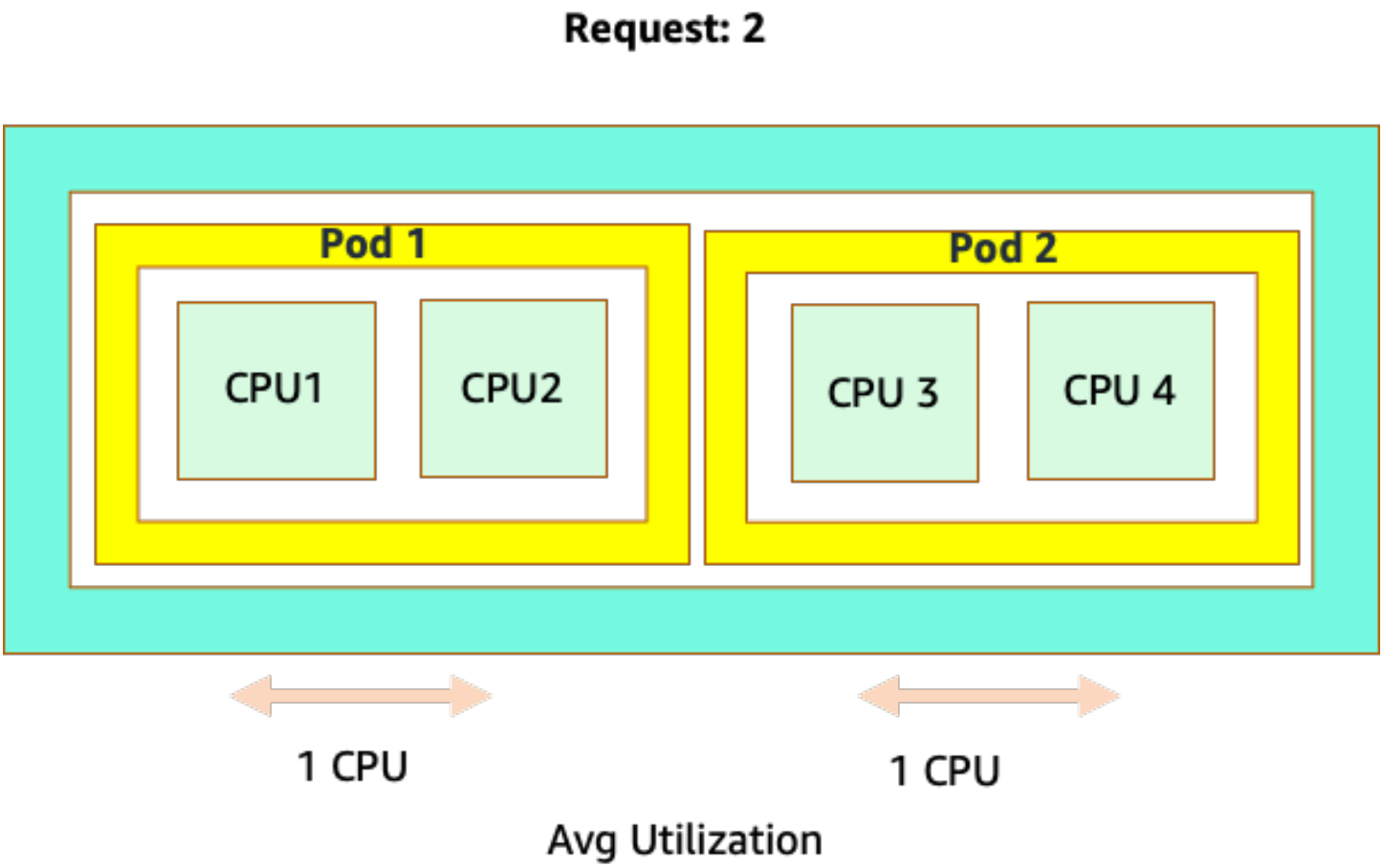
Nachdem wir nun verstanden haben, welche Auswirkungen es hat, wenn Anwendungen nicht in ihrem optimalen Zustand arbeiten können, wollen wir zur Knotenebene zurückkehren und uns fragen, warum dieser Unterschied zwischen dem Kubernetes-Scheduler und Linux CFS so wichtig ist.

Bei der Hoch- und Herunterskalierung mit HPA können wir ein Szenario haben, in dem wir viel Speicherplatz haben, um mehr Pods zuzuweisen. Dies wäre eine schlechte Entscheidung, da der links abgebildete Knoten bereits zu 100% CPU-Auslastung hat. In einem unrealistischen, aber theoretisch möglichen Szenario könnten wir das andere Extrem haben, bei dem unser Knoten vollständig voll ist, unsere CPU-Auslastung jedoch Null ist.

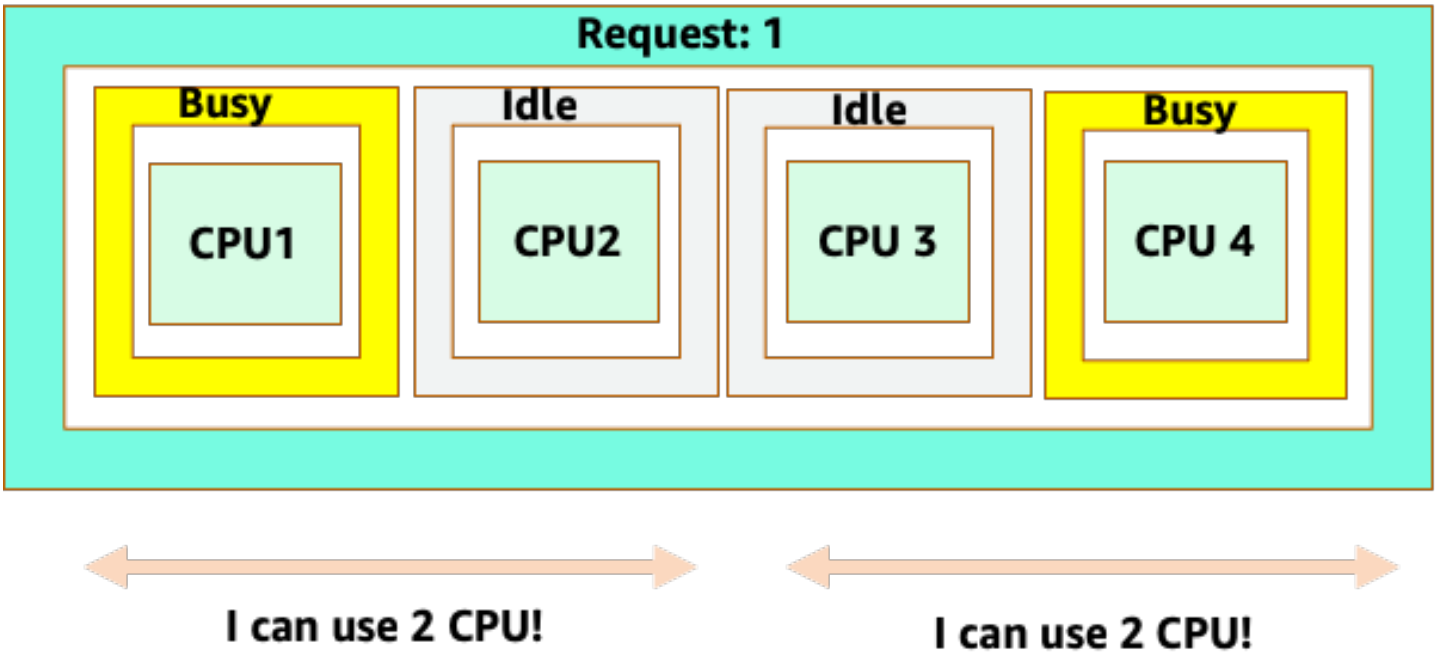


Anfragen einrichten

Es wäre verlockend, die Anfrage auf den für diese Anwendung geltenden „Sweet Spot“ -Wert zu setzen. Dies würde jedoch zu Ineffizienzen führen, wie in der Abbildung unten dargestellt. Hier haben wir den Anforderungswert auf 2 vCPU gesetzt, die durchschnittliche Auslastung dieser Pods beträgt jedoch die meiste Zeit nur 1 CPU. Diese Einstellung würde dazu führen, dass wir 50% unserer CPU-Zyklen verschwenden würden, was inakzeptabel wäre.



Dies bringt uns zur komplexen Antwort auf das Problem. Die Nutzung von Containern kann nicht im luftleeren Raum betrachtet werden. Man muss die anderen Anwendungen berücksichtigen, die auf dem Knoten ausgeführt werden. Im folgenden Beispiel werden Container, bei denen es sich um Burst-Container handelt, mit zwei Containern mit niedriger CPU-Auslastung vermischt, für die möglicherweise nur wenig Speicherplatz zur Verfügung steht. Auf diese Weise ermöglichen wir es den Containern, ihren optimalen Standort zu erreichen, ohne den Knoten zu belasten.



Das wichtige Konzept, das man aus all dem mitnehmen kann, ist, dass die Verwendung des Kubernetes-Scheduler-Konzepts der Kerne, um die Leistung von Linux-Containern zu verstehen, zu schlechten Entscheidungen führen kann, da sie nicht miteinander verwandt sind.

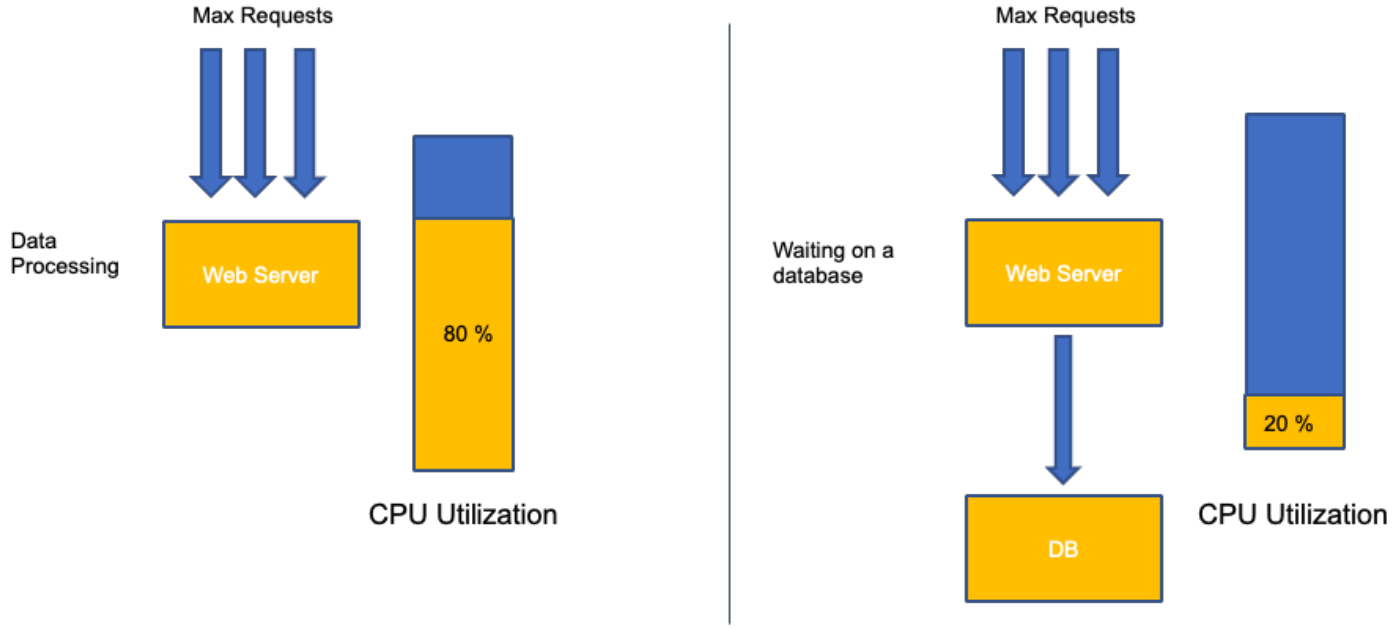
> **Note**
>
> Linux CFS hat seine Stärken. Dies gilt insbesondere für I/O basierte Workloads. Wenn Ihre Anwendung jedoch volle Kerne ohne Nebenkosten verwendet und keine I/O Anforderungen gestellt werden, kann CPU-Pinning diesen Prozess erheblich vereinfachen und ist mit diesen Einschränkungen empfehlenswert.

Auslastung versus Sättigung

Ein häufiger Fehler bei der Anwendungsskalierung besteht darin, nur die CPU-Auslastung für Ihre Skalierungsmetrik zu verwenden. In komplexen Anwendungen ist dies fast immer ein schlechter Indikator dafür, dass eine Anwendung tatsächlich mit Anfragen überlastet ist. Im Beispiel auf der linken Seite sehen wir, dass alle unsere Anfragen tatsächlich den Webserver erreichen, sodass die CPU-Auslastung mit der Sättigung gut abschneidet.

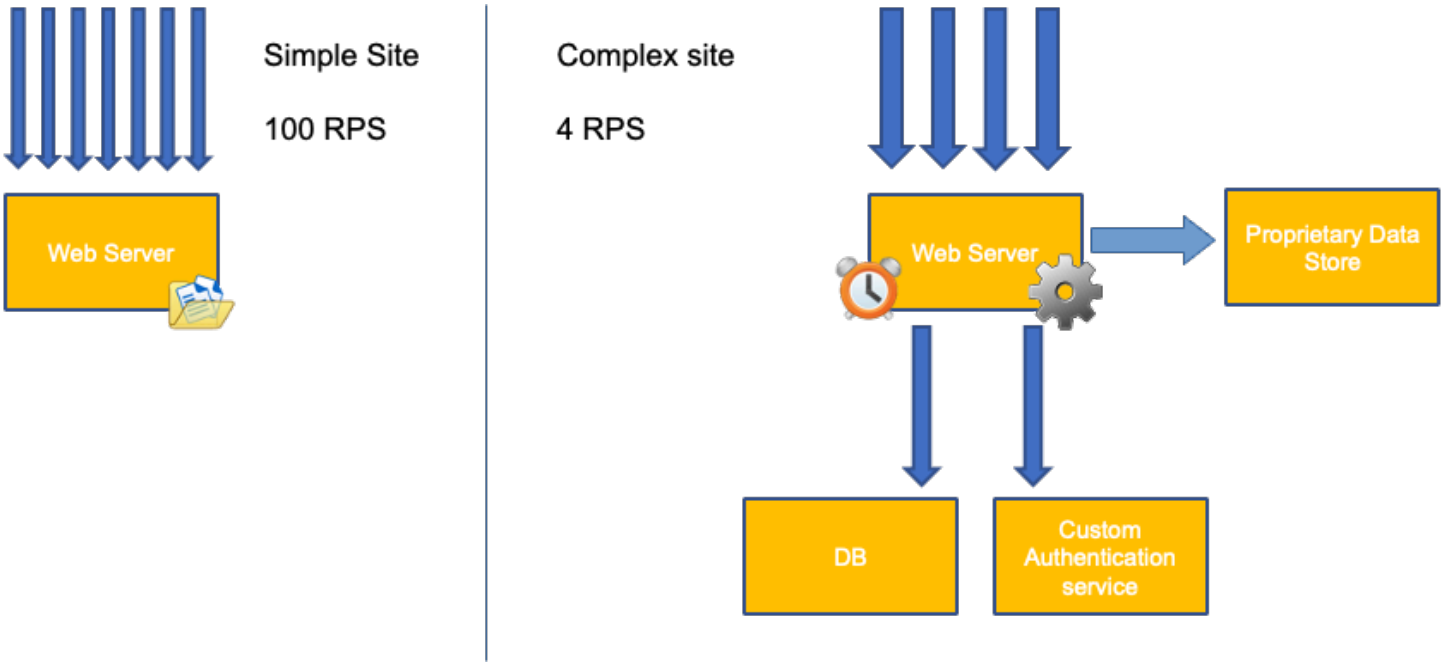
In realen Anwendungen ist es wahrscheinlich, dass einige dieser Anfragen von einer Datenbankschicht oder einer Authentifizierungsebene usw. bearbeitet werden. In diesem häufigeren

Fall stellen Sie fest, dass die CPU die Nachverfolgung nicht voll ausgelastet hat, da die Anfrage von anderen Entitäten bearbeitet wird. In diesem Fall ist die CPU ein sehr schlechter Indikator für die Sättigung.



Die Verwendung der falschen Metrik bei der Anwendungsleistung ist der Hauptgrund für unnötige und unvorhersehbare Skalierung in Kubernetes. Bei der Auswahl der richtigen Sättigungsmetrik für die Art der Anwendung, die Sie verwenden, ist große Vorsicht geboten. Es ist wichtig zu beachten, dass es keine allgemeingültige Empfehlung gibt, die für alle geeignet ist. Abhängig von der verwendeten Sprache und der Art der jeweiligen Anwendung gibt es unterschiedliche Messwerte für die Sättigung.

Wir denken vielleicht, dass dieses Problem nur bei der CPU-Auslastung liegt, aber auch andere gängige Messwerte wie die Anfrage pro Sekunde können auf genau das gleiche Problem zurückzuführen sein, das oben erörtert wurde. Beachten Sie, dass die Anfrage auch auf DB-Ebenen oder Authentifizierungsebenen gehen kann und nicht direkt von unserem Webserver bedient wird. Daher ist sie eine schlechte Metrik für die tatsächliche Auslastung des Webservers selbst.



Leider gibt es keine einfachen Antworten, wenn es darum geht, die richtige Sättigungsmetrik auszuwählen. Hier sind einige Richtlinien, die Sie berücksichtigen sollten:

- Machen Sie sich mit Ihrer Sprache vertraut — Sprachen mit mehreren Betriebssystem-Threads reagieren anders als Single-Thread-Anwendungen und wirken sich daher unterschiedlich auf den Knoten aus.
- Machen Sie sich mit der korrekten vertikalen Skala vertraut — wie viel Puffer benötigen Sie für die vertikale Skalierung Ihrer Anwendungen, bevor Sie einen neuen Pod skalieren?
- Welche Metriken spiegeln wirklich die Sättigung Ihrer Anwendung wider? — Die Sättigungsmetrik für einen Kafka Producer wäre etwas ganz anderes als für eine komplexe Webanwendung.
- Wie wirken sich alle anderen Anwendungen auf dem Knoten gegenseitig aus? Die Anwendungsleistung erfolgt nicht im luftleeren Raum, die anderen Workloads auf dem Knoten haben erhebliche Auswirkungen.

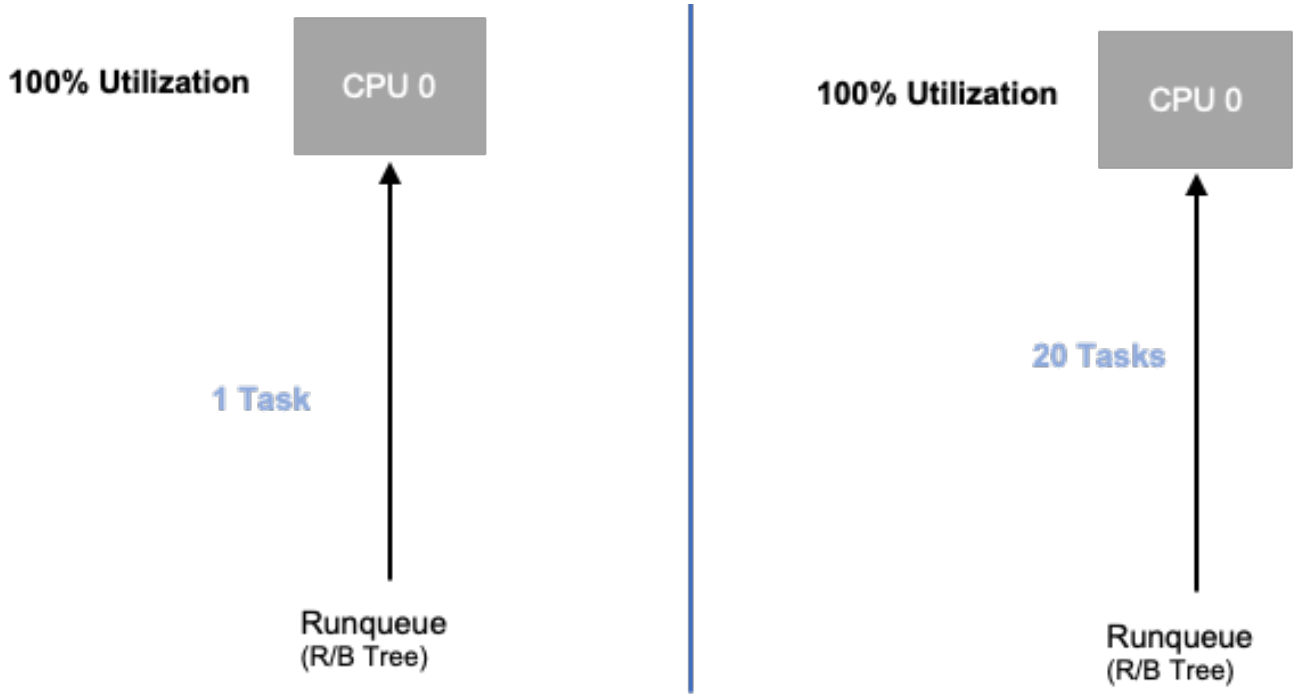
Um diesen Abschnitt abzuschließen, wäre es leicht, die obigen Ausführungen als zu komplex und unnötig abzutun. Es kann oft vorkommen, dass wir ein Problem haben, uns aber der wahren Natur des Problems nicht bewusst sind, weil wir uns die falschen Kennzahlen ansehen. Im nächsten Abschnitt werden wir uns ansehen, wie das passieren könnte.

Knotensättigung

Nachdem wir uns nun mit der Anwendungssättigung befasst haben, wollen wir dasselbe Konzept aus der Sicht der Knoten betrachten. Schauen wir uns zwei an CPUs , die zu 100% ausgelastet sind, um den Unterschied zwischen Auslastung und Sättigung zu erkennen.

Die vCPU auf der linken Seite ist zu 100% ausgelastet, es warten jedoch keine anderen Aufgaben darauf, auf dieser vCPU ausgeführt zu werden. Aus rein theoretischer Sicht ist dies also ziemlich effizient. In der Zwischenzeit warten im zweiten Beispiel 20 Single-Thread-Anwendungen darauf, von einer vCPU verarbeitet zu werden. Bei allen 20 Anwendungen kommt es jetzt zu einer gewissen Latenz, während sie darauf warten, dass sie an der Reihe sind, von der vCPU verarbeitet zu werden. Mit anderen Worten, die vCPU auf der rechten Seite ist gesättigt.

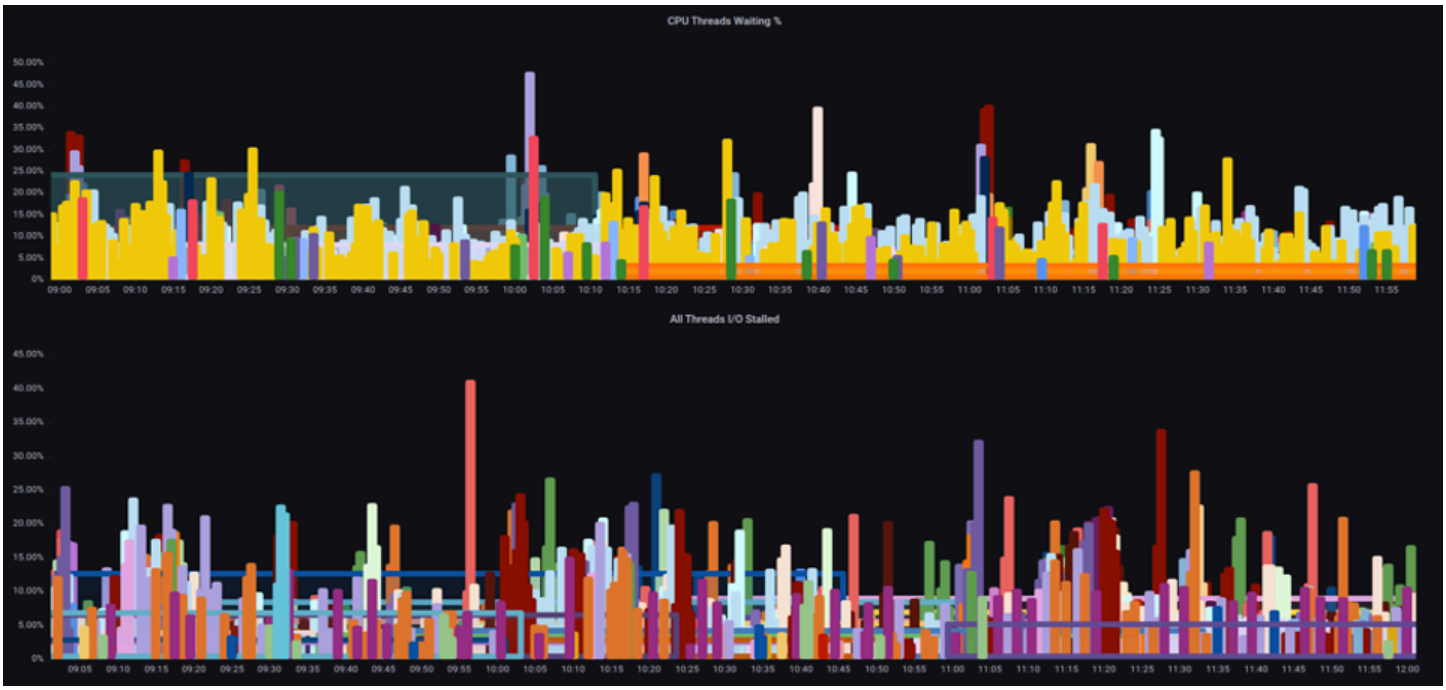
Wir würden dieses Problem nicht nur nicht sehen, wenn wir uns nur die Auslastung ansehen würden, sondern wir könnten diese Latenz auch auf etwas zurückführen, das nichts damit zu tun hat, wie z. B. Netzwerke, was uns auf den falschen Weg führen würde.



Wenn die Gesamtzahl der Pods erhöht wird, die zu einem bestimmten Zeitpunkt auf einem Knoten ausgeführt werden, ist es wichtig, die Sättigungsmetriken und nicht nur die Nutzungsmetriken zu betrachten, da wir leicht übersehen können, dass ein Knoten überlastet ist. Für diese Aufgabe können wir Messwerte zur Information über den Druckausfall verwenden, wie in der folgenden Tabelle dargestellt.

PromQL — I/O blockiert

```
topk(3, ((irate(node_pressure_io_stalled_seconds_total[1m])) * 100))
```



> **Note**
>
> Weitere Informationen zu Messwerten zum Stillstand bei Pressure finden Sie unter https://facebookmicrosites.github. io/psi/docs/overview*

Anhand dieser Metriken können wir feststellen, ob Threads auf die CPU warten oder ob sogar jeder Thread auf der Box beim Warten auf Ressourcen wie Speicher oder I/O. For example, we could see what percentage every thread on the instance was stalled waiting on I/O über einen Zeitraum von 1 Minute blockiert ist.

```
topk(3, ((irate(node_pressure_io_stalled_seconds_total[1m])) * 100))
```

Anhand dieser Metrik können wir in der obigen Tabelle sehen, dass jeder Thread auf der Box in 45% der Fälle blockiert war, während er I/O auf die Höchstgrenze wartete, was bedeutet, dass wir all diese CPU-Zyklen in dieser Minute weggeworfen haben. Wenn wir verstehen, dass dies geschieht, können wir einen erheblichen Teil der vCPU-Zeit zurückgewinnen und so die Skalierung effizienter gestalten.

HPA V2

Es wird empfohlen, die Autoscaling/v2-Version der HPA-API zu verwenden. Bei den älteren Versionen der HPA-API konnte es in bestimmten Randfällen zu Verzögerungen bei der Skalierung kommen. Es war auch auf Pods beschränkt, die sich bei jedem Skalierungsschritt nur verdoppelten, was bei kleinen Bereitstellungen, die schnell skaliert werden mussten, zu Problemen führte.

Autoscaling/v2 bietet uns mehr Flexibilität, da wir mehrere Kriterien für die Skalierung berücksichtigen können, und ermöglicht uns ein hohes Maß an Flexibilität bei der Verwendung benutzerdefinierter und externer Metriken (keine K8s-Metriken).

Als Beispiel können wir auf den höchsten von drei Werten skalieren (siehe unten). Wir skalieren, wenn die durchschnittliche Auslastung aller Pods über 50% liegt, wenn benutzerdefinierte Metriken die Pakete pro Sekunde des Eingangs den Durchschnitt von 1.000 überschreiten oder das Eingangsobjekt 10.000 Anfragen pro Sekunde überschreitet.

> **Note**
>
> Dies dient nur dazu, die Flexibilität der Auto-Scaling-API zu demonstrieren. Wir empfehlen, nicht zu komplexe Regeln zu verwenden, die in der Produktion schwierig zu beheben sein können.

```
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
```

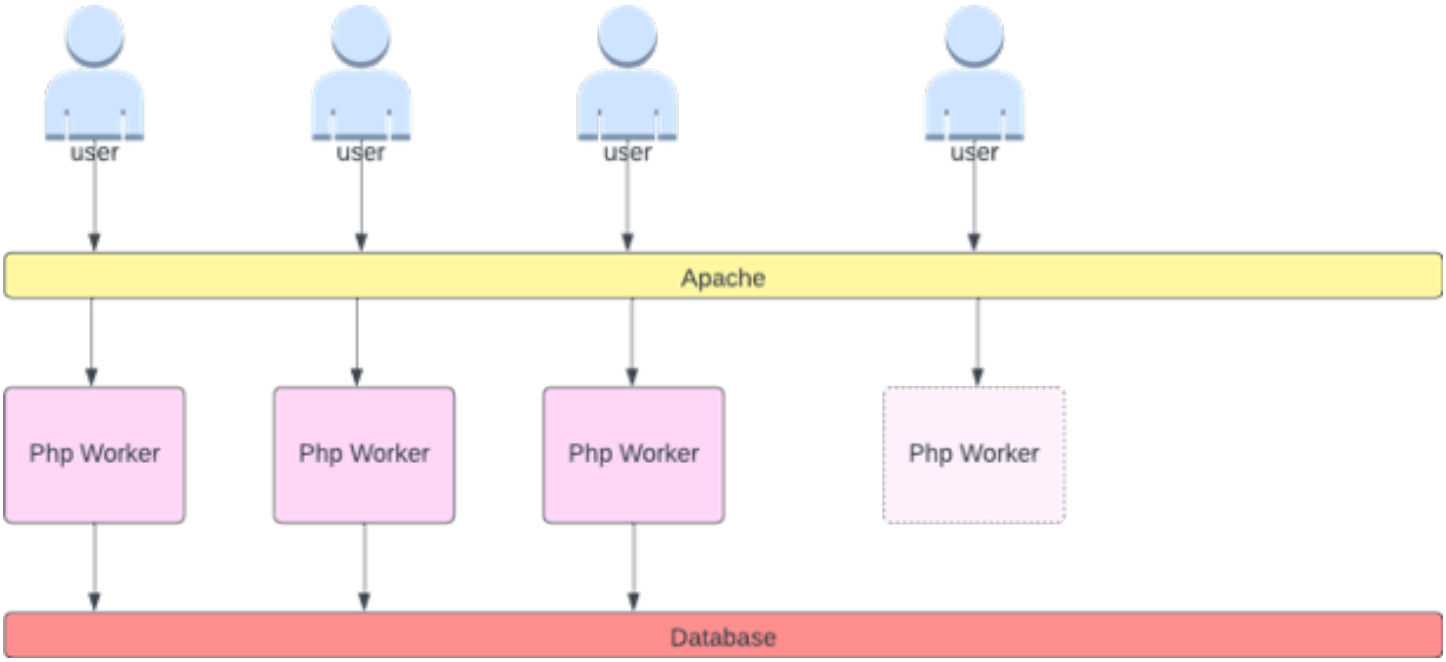
```
      averageUtilization: 50
  - type: Pods
    pods:
      metric:
        name: packets-per-second
      target:
        type: AverageValue
        averageValue: 1k
  - type: Object
    object:
      metric:
        name: requests-per-second
      describedObject:
        apiVersion: networking.k8s.io/v1
        kind: Ingress
        name: main-route
      target:
        type: Value
        value: 10k
```

Wir haben jedoch die Gefahr erkannt, die mit der Verwendung solcher Metriken für komplexe Webanwendungen verbunden ist. In diesem Fall wären wir besser bedient, wenn wir benutzerdefinierte oder externe Metriken verwenden würden, die die Auslastung unserer Anwendung im Vergleich zur Auslastung genau wiedergeben. HPAv2 ermöglicht dies, indem wir nach jeder beliebigen Metrik skalieren können. Wir müssen diese Metrik jedoch immer noch finden und zur Verwendung nach Kubernetes exportieren.

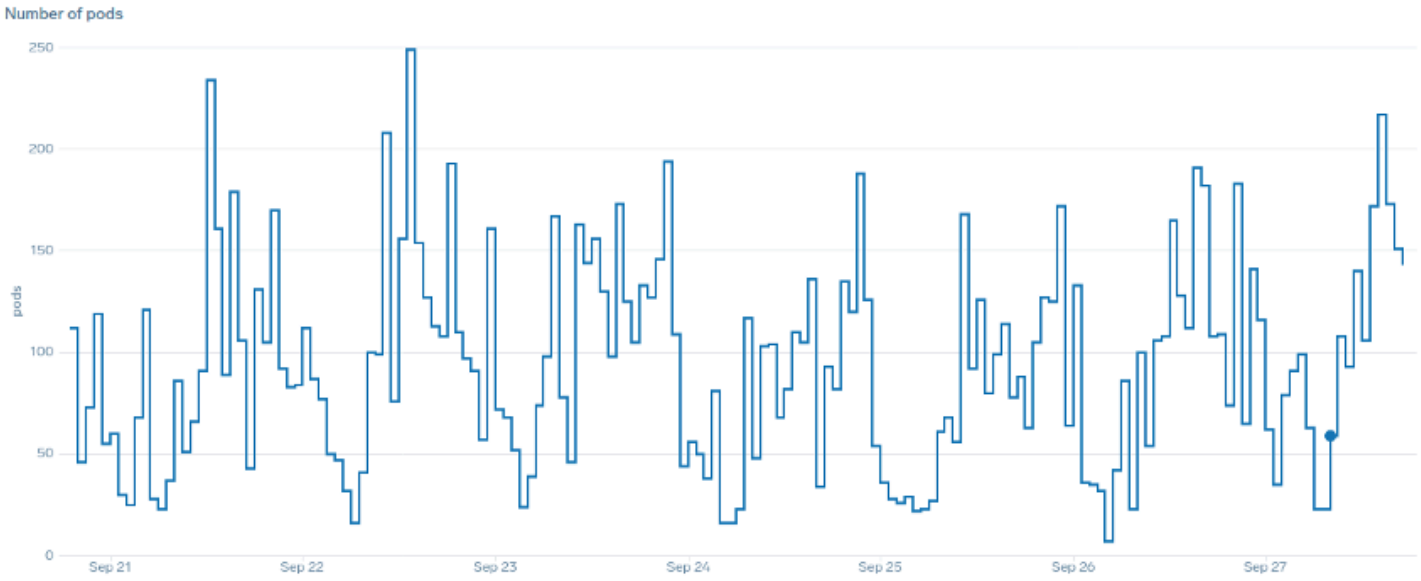
Wir können uns zum Beispiel die Anzahl der aktiven Thread-Warteschlangen in Apache ansehen. Dadurch entsteht oft ein „glatteres“ Skalierungsprofil (mehr zu diesem Begriff in Kürze). Wenn ein Thread aktiv ist, spielt es keine Rolle, ob dieser Thread auf einer Datenbankebene wartet oder eine Anfrage lokal bearbeitet. Wenn alle Anwendungs-Threads verwendet werden, ist das ein gutes Zeichen dafür, dass die Anwendung überlastet ist.

Wir können diese Thread-Erschöpfung als Signal verwenden, um einen neuen Pod mit einem vollständig verfügbaren Thread-Pool zu erstellen. Dies gibt uns auch die Kontrolle darüber, wie groß der Puffer sein soll, den die Anwendung in Zeiten mit hohem Datenverkehr aufnehmen soll. Wenn wir beispielsweise einen Thread-Pool von insgesamt 10 hätten, hätte die Skalierung bei 4 verwendeten Threads gegenüber 8 verwendeten Threads große Auswirkungen auf den Puffer, den wir bei der Skalierung der Anwendung zur Verfügung haben. Eine Einstellung von 4 wäre für eine Anwendung sinnvoll, die unter hoher Last schnell skaliert werden muss, wobei eine Einstellung von 8 unsere

Ressourcen effizienter einsetzen würde, wenn wir genügend Zeit für die Skalierung hätten, da die Anzahl der Anfragen im Laufe der Zeit langsam und nicht stark zunimmt.

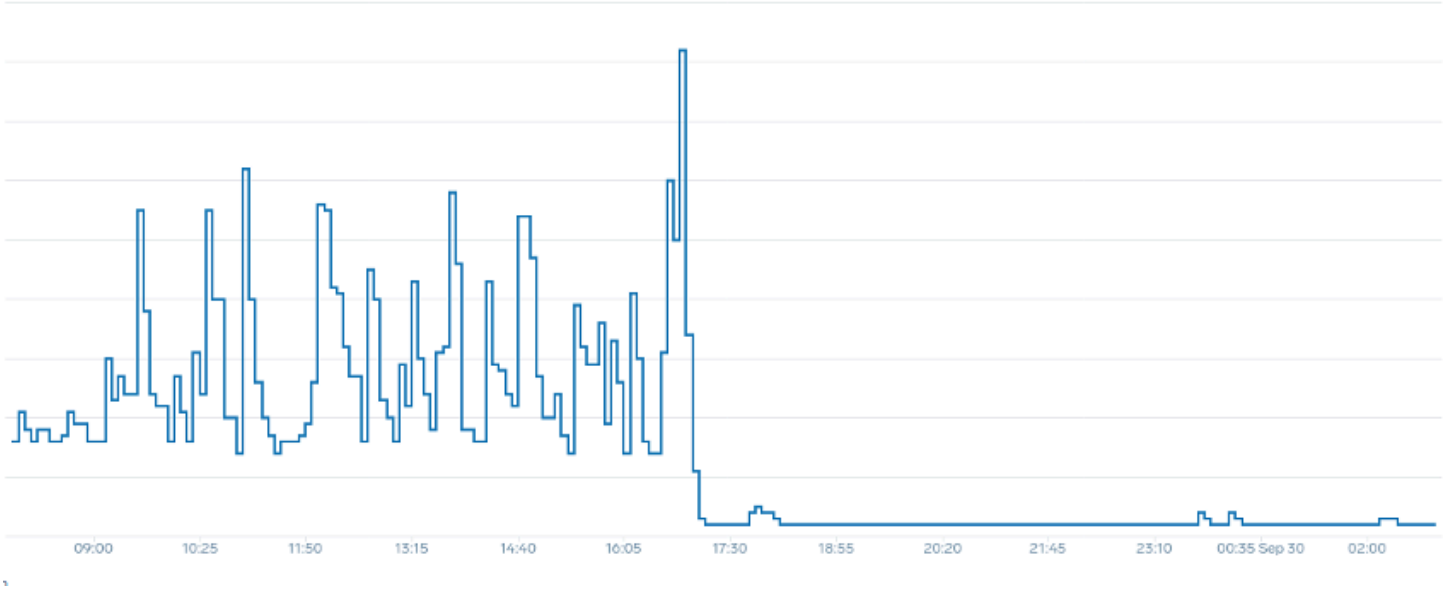


Was meinen wir mit dem Begriff „glatt“, wenn es um Skalierung geht? Beachten Sie die folgende Tabelle, in der wir die CPU als Metrik verwenden. Die Pods in dieser Bereitstellung steigen innerhalb kurzer Zeit sprunghaft an und reichen von 50 Pods bis hin zu 250 Pods, um dann sofort wieder herunterskaliert zu werden. Diese äußerst ineffiziente Skalierung ist die Hauptursache für die Abwanderung von Clustern.



Beachten Sie, dass wir nach der Umstellung auf eine Metrik, die den korrekten Sweetspot unserer Anwendung (mittlerer Teil des Diagramms) widerspiegelt, problemlos skalieren können. Unsere

Skalierung ist jetzt effizient, und unsere Pods können mit dem Spielraum, den wir durch die Anpassung der Anforderungseinstellungen geschaffen haben, vollständig skalieren. Jetzt erledigt eine kleinere Gruppe von Pods die Arbeit, die Hunderte von Pods zuvor erledigt haben. Daten aus der Praxis zeigen, dass dies der wichtigste Faktor für die Skalierbarkeit von Kubernetes-Clustern ist.

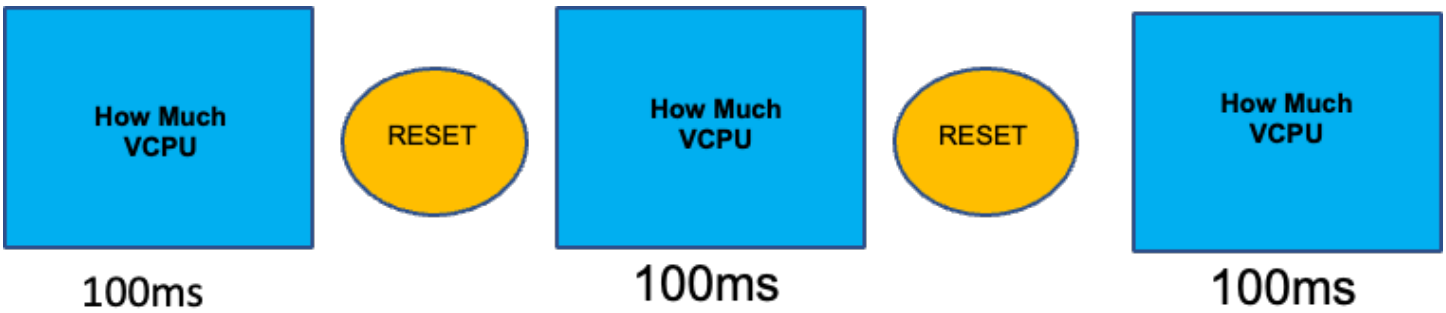


Die wichtigste Erkenntnis ist, dass die CPU-Auslastung nur eine Dimension der Anwendungs- und Knotenleistung ist. Die Verwendung der CPU-Auslastung als alleinigen Gesundheitsindikator für unsere Knoten und Anwendungen führt zu Problemen bei Skalierung, Leistung und Kosten, die allesamt eng miteinander verknüpft sind. Je leistungsfähiger die Anwendung und die Knoten sind, desto weniger müssen Sie skalieren, was wiederum Ihre Kosten senkt.

Wenn Sie die richtigen Sättigungsmetriken für die Skalierung Ihrer speziellen Anwendung finden und verwenden, können Sie auch die tatsächlichen Engpässe für diese Anwendung überwachen und entsprechende Warnmeldungen auslösen. Wenn dieser wichtige Schritt übersprungen wird, sind Berichte über Leistungsprobleme nur schwer, wenn nicht gar unmöglich nachzuvollziehen.

CPU-Grenzwerte festlegen

Um diesen Abschnitt über missverstandene Themen abzurunden, werden wir uns mit CPU-Limits befassen. Kurz gesagt, Grenzwerte sind Metadaten, die mit dem Container verknüpft sind, der über einen Zähler verfügt, der alle 100 ms zurückgesetzt wird. Auf diese Weise kann Linux nachverfolgen, wie viele CPU-Ressourcen knotenweit von einem bestimmten Container in einem Zeitraum von 100 ms genutzt werden.



Ein häufiger Fehler beim Setzen von Grenzwerten ist die Annahme, dass die Anwendung Single-Thread-fähig ist und nur auf der ihr „`zugewiesenen“ vCPU läuft. Im obigen Abschnitt haben wir gelernt, dass CFS keine Kerne zuweist, und dass ein Container, der große Thread-Pools ausführt, in Wirklichkeit alle verfügbaren vCPUs auf der Box plant.

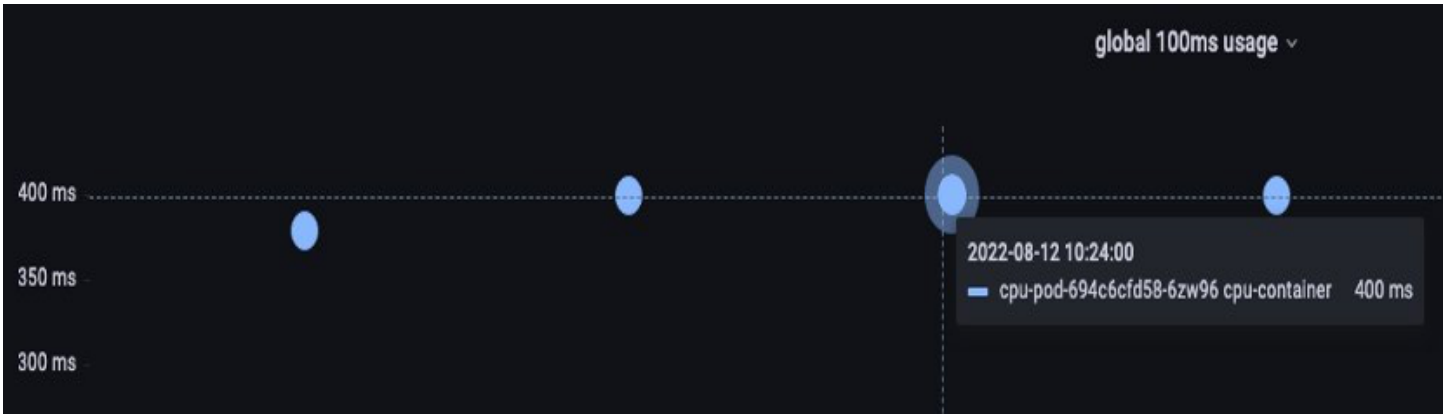
Wenn 64 Betriebssystem-Threads auf 64 verfügbaren Kernen laufen (aus Sicht eines Linux-Knotens), machen wir die Gesamtsumme der verbrauchten CPU-Zeit in einem Zeitraum von 100 ms ziemlich hoch, nachdem die auf all diesen 64 Kernen laufende Zeit zusammengezählt wurde. Da dies möglicherweise nur während eines Garbage-Collection-Prozesses passiert, kann es ziemlich leicht sein, so etwas zu übersehen. Aus diesem Grund ist es notwendig, Metriken zu verwenden, um sicherzustellen, dass wir im Laufe der Zeit die richtige Nutzung haben, bevor wir versuchen, ein Limit festzulegen.

Zum Glück können wir genau sehen, wie viel vCPU von allen Threads in einer Anwendung verwendet wird. Wir werden die Metrik `container_cpu_usage_seconds_total` für diesen Zweck verwenden.

Da die Drosselungslogik alle 100 ms erfolgt und es sich bei dieser Metrik um eine Metrik pro Sekunde handelt, werden wir PromQL entsprechend diesem Zeitraum von 100 ms anpassen. Wenn Sie sich eingehend mit dieser PromQL-Aussage befassen möchten, lesen Sie bitte den folgenden Blog.

PromQL-Abfrage:

```
topk(3, max by (pod, container)(rate(container_cpu_usage_seconds_total{image!="",
 instance="$instance"}[$__rate_interval]))) / 10
```



Sobald wir das Gefühl haben, dass wir den richtigen Wert haben, können wir das Limit für die Produktion festlegen. Dann muss geprüft werden, ob unsere Anwendung aufgrund eines Unerwarteten gedrosselt wird. Wir können das tun, indem wir uns Folgendes ansehen `container_cpu_throttled_seconds_total`

```
topk(3, max by (pod, container)(rate(container_cpu_cfs_throttled_seconds_total{image!
=``""``, instance=``"$instance"``}[$__rate_interval]))) / 10
```



Arbeitsspeicher

Die Speicherzuweisung ist ein weiteres Beispiel, bei dem es leicht ist, das Kubernetes-Scheduling-Verhalten mit dem Linux-Verhalten zu verwechseln. CGroup Dies ist ein differenzierteres Thema, da sich die Art und Weise, wie CGroup v2 unter Linux mit Speicher umgeht, erheblich geändert hat und Kubernetes seine Syntax geändert hat, um dies widerzuspiegeln. Weitere Informationen finden Sie in diesem Blog.

Im Gegensatz zu CPU-Anfragen bleiben Speicheranforderungen nach Abschluss des Planungsprozesses ungenutzt. Dies liegt daran, dass wir den Speicher in CGroup Version 1 nicht auf

die gleiche Weise komprimieren können wie mit der CPU. Damit bleiben uns nur noch Speicherlimits übrig, die als Ausfallschutz gegen Speicherlecks dienen sollen, indem sie den Pod vollständig beenden. Dabei handelt es sich um ein Alles-oder-nichts-Konzept, aber uns wurden jetzt neue Möglichkeiten aufgezeigt, dieses Problem zu lösen.

Zunächst ist es wichtig zu verstehen, dass das Einstellen der richtigen Speichermenge für Container nicht so einfach ist, wie es scheint. Das Dateisystem unter Linux verwendet Speicher als Cache, um die Leistung zu verbessern. Dieser Cache wird mit der Zeit wachsen, und es kann schwierig sein, zu wissen, wie viel Speicher für den Cache einfach nützlich ist, aber ohne nennenswerte Auswirkungen auf die Anwendungsleistung zurückgewonnen werden kann. Dies führt häufig zu einer Fehlinterpretation der Speichernutzung.

Die Fähigkeit, Speicher zu „komprimieren“, war einer der Hauptgründe für CGroup Version 2. Weitere Informationen zur Notwendigkeit von CGroup V2 finden Sie in Chris Downs Präsentation, in der er darlegt, warum die Tatsache, dass er nicht LISA21 in der Lage war, den Mindestspeicher korrekt einzustellen, einer der Gründe war, die ihn dazu bewogen haben, die Messwerte für CGroup Version v2 und Druckabfall zu entwickeln.

Zum Glück hat Kubernetes jetzt das Konzept von `memory.min` und `memory.high` unter. `requests.memory` Dies gibt uns die Möglichkeit, diesen zwischengespeicherten Speicher aggressiv für andere Container freizugeben. Sobald der Container die Speicherobergrenze erreicht hat, kann der Kernel den Speicher dieses Containers bis zu dem auf festgelegten Wert aggressiv zurückfordern. `memory.min` Das gibt uns mehr Flexibilität, wenn ein Knoten unter Speicherdruck gerät.

Die zentrale Frage lautet: Auf welchen Wert `memory.min` soll gesetzt werden? Hier kommen die Kennzahlen zum Stillstand des Speicherdrucks ins Spiel. Wir können diese Metriken verwenden, um zu erkennen, dass Speicher auf Containerebene überlastet wird. Dann können wir Controller wie fbtax verwenden, um die richtigen Werte für zu ermitteln, `memory.min` indem wir nach diesem Speicherüberhang suchen, und den `memory.min` Wert dynamisch auf diese Einstellung setzen.

Zusammenfassung

Um den Abschnitt zusammenzufassen, lassen sich die folgenden Konzepte leicht miteinander verbinden:

- Auslastung und Sättigung
- Linux-Leistungsregeln mit Kubernetes Scheduler-Logik

Es muss sehr sorgfältig darauf geachtet werden, diese Konzepte voneinander zu trennen. Leistung und Skalierbarkeit sind tief miteinander verknüpft. Unnötige Skalierung führt zu Leistungsproblemen, was wiederum zu Skalierungsproblemen führt.

Kubernetes Upstream SLOs

Amazon EKS führt denselben Code aus wie die Upstream-Versionen von Kubernetes und stellt sicher, dass EKS-Cluster innerhalb der von der SLOs Kubernetes-Community definierten Werte betrieben werden. Die Kubernetes Scalability Special Interest Group (SIG) definiert die Skalierbarkeitsziele und untersucht Leistungsengpässe durch und. SLIs SLOs

SLIs sind die Art und Weise, wie wir ein System messen, wie Metriken oder Kennzahlen, anhand derer festgestellt werden kann, wie „gut“ das System läuft, z. B. die Latenz oder die Anzahl der Anfragen. SLOs definieren die Werte, die erwartet werden, wenn das System „gut“ läuft, z. B. wenn die Anforderungslatenz weniger als 3 Sekunden beträgt. Die Kubernetes-Lösungen SLOs SLIs konzentrieren sich auf die Leistung der Kubernetes-Komponenten und sind völlig unabhängig vom Amazon EKS Service SLAs , der sich auf die Verfügbarkeit des EKS-Cluster-Endpunkts konzentriert.

Kubernetes bietet eine Reihe von Funktionen, mit denen Benutzer das System mit benutzerdefinierten Add-Ons oder Treibern wie CSI-Treibern, Admissions-Webhooks und Auto-Scalern erweitern können. Diese Erweiterungen können die Leistung eines Kubernetes-Clusters auf unterschiedliche Weise drastisch beeinflussen, d. h. ein Zugangswebhook `failurePolicy=Ignore` könnte die Latenz bei K8s-API-Anfragen erhöhen, wenn das Webhook-Ziel nicht verfügbar ist. Die Kubernetes Scalability SIG definiert Skalierbarkeit anhand eines „Sie versprechen, wir versprechen es“ -Framework:

> Wenn Sie Folgendes versprechen: - Ihren Cluster korrekt zu konfigurieren - Erweiterungsfunktionen „angemessen“ zu verwenden - die Last im Cluster innerhalb der empfohlenen Grenzwerte zu halten
>
> dann versprechen wir, dass Ihr Cluster skaliert, d. h.: - alle SLOs sind zufrieden.

Kubernetes SLOs

Kubernetes berücksichtigt SLOs nicht alle Plugins und externen Einschränkungen, die sich auf einen Cluster auswirken könnten, wie z. B. die Skalierung von Worker-Nodes oder Admissions-Webhooks. Diese SLOs konzentrieren sich auf Kubernetes-Komponenten und stellen sicher, dass die Aktionen und Ressourcen von Kubernetes den Erwartungen entsprechen. Sie SLOs helfen

Kubernetes-Entwicklern dabei, sicherzustellen, dass Änderungen am Kubernetes-Code die Leistung des gesamten Systems nicht beeinträchtigen.

Das Kubernetes Scalability SIG definiert den folgenden offiziellen SLO/S. SLIs Das Amazon EKS-Team führt regelmäßig Skalierbarkeitstests an EKS-Clustern durch SLOs/SLIs , um zu überwachen, ob Leistungseinbußen auftreten, wenn Änderungen vorgenommen und neue Versionen veröffentlicht werden.

Ziel	Definition	SLO
Latenz bei API-Anfragen (mutierend)	Latenz bei der Verarbeitung mutierender API-Aufrufe für einzelne Objekte für jedes Paar (Ressourcen, Verb), gemessen als 99. Perzentil in den letzten 5 Minuten	In der Kubernetes-Standardinstallation gilt für jedes Paar (Ressourcen, Verb), ausgenommen virtuelle und aggregierte Ressourcen und benutzerdefinierte Ressourcendefinitionen, das 99. Perzentil pro Cluster-Tag <= 1s
Latenz bei API-Anfragen (schreibgeschützt)	Latenz bei der Verarbeitung schreibgeschützter API-Aufrufe ohne Streaming für jedes Paar (Ressourcen, Bereich), gemessen als 99. Perzentil in den letzten 5 Minuten	In der Kubernetes-Standardinstallation gilt für jedes Paar (Ressourcen, Bereich), ausgenommen virtuelle und aggregierte Ressourcen und benutzerdefinierte Ressourcendefinitionen, 99. Perzentil pro Cluster-Tag: (a) <= 1s if (b) <= 30s, andernfalls (wenn oder) `scope=resource scope=namespace scope=cluster`
Latenz beim Starten des Pods	Die Latenz beim Starten planbarer statusloser Pods, ohne die Zeit zum Abrufen von Images und zum Ausführen von Init-Containern, gemessen vom Zeitstempel der Pod-	In der Kubernetes-Standardinstallation ist das 99. Perzentil pro Cluster-Tag <= 5 Sekunden

Ziel	Definition	SLO
	Erstellung bis zu dem Zeitpunkt, zu dem alle zugehörigen Container als gestartet gemeldet und per Watch beobachtet wurden, gemessen als 99. Perzentil in den letzten 5 Minuten	

Latenz bei API-Anfragen

Die `kube-apiserver` wurde als `1m0s` Standard `--request-timeout` definiert, was bedeutet, dass eine Anfrage bis zu einer Minute (60 Sekunden) lang ausgeführt werden kann, bevor ein Timeout eintritt und storniert wird. Die für die Latenz SLOs definierten Werte sind nach der Art der Anfrage aufgeteilt, die mutierend oder schreibgeschützt sein kann:

Mutation

Durch mutierende Anfragen in Kubernetes werden Änderungen an einer Ressource vorgenommen, wie z. B. Erstellungen, Löschungen oder Aktualisierungen. Diese Anfragen sind teuer, da diese Änderungen in das etcd-Backend geschrieben werden müssen, bevor das aktualisierte Objekt zurückgegeben wird. Etcd ist ein verteilter Schlüsselwertspeicher, der für alle Kubernetes-Clusterdaten verwendet wird.

Diese Latenz wird als 99. Perzentil über 5 Minuten für (Ressourcen-, Verb-) Paare von Kubernetes-Ressourcen gemessen. Dies würde beispielsweise die Latenz für Create-Pod-Anfragen und Update-Node-Anfragen messen. Die Latenz der Anfrage muss <= 1 Sekunde betragen, um den SLO zu erfüllen.

Schreibgeschützt

Schreibgeschützte Anfragen rufen eine einzelne Ressource (wie Get Pod X) oder eine Sammlung (wie „Get all Pods from Namespace X“) ab. Der `kube-apiserver` verwaltet einen Cache mit Objekten, sodass die angeforderten Ressourcen möglicherweise aus dem Cache zurückgegeben werden oder sie müssen möglicherweise zuerst von etcd abgerufen werden. Diese Latenzen werden ebenfalls anhand des 99. Perzentils über einen Zeitraum von 5 Minuten gemessen, allerdings können nur lesbare Anfragen unterschiedliche Bereiche haben. Das SLO definiert zwei verschiedene Ziele:

- Bei Anfragen für eine einzelne Ressource (d. h.`kubectl get pod -n mynamespace my-controller-xxx`) sollte die Anforderungslatenz bei <= 1 Sekunde bleiben.
- Bei Anfragen, die für mehrere Ressourcen in einem Namespace oder einem Cluster gestellt werden (z. B.`kubectl get pods -A`), sollte die Latenz bei <= 30 Sekunden bleiben

Das SLO hat unterschiedliche Zielwerte für verschiedene Anforderungsbereiche, da bei Anfragen für eine Liste von Kubernetes-Ressourcen erwartet wird, dass die Details aller Objekte in der Anfrage innerhalb des SLO zurückgegeben werden. Bei großen Clustern oder großen Ressourcensammlungen kann dies zu großen Antwortgrößen führen, deren Rückgabe einige Zeit in Anspruch nehmen kann. In einem Cluster mit Zehntausenden von Pods, wobei jeder Pod bei JSON-Kodierung etwa 1 KiB groß ist, würde die Rückgabe aller Pods im Cluster aus 10 MB oder mehr bestehen. Kubernetes-Clients können helfen, diese Antwortgröße zu reduzieren, indem sie APIListChunking verwenden, um große Ressourcensammlungen abzurufen.

Latenz beim Pod-Start

Bei diesem SLO geht es hauptsächlich um die Zeit, die von der Pod-Erstellung bis zum tatsächlichen Beginn der Ausführung der Container in diesem Pod vergeht. Um dies zu messen, wird der Unterschied zwischen dem auf dem Pod aufgezeichneten Erstellungszeitstempel und dem Zeitpunkt berechnet, zu dem eine WATCH auf diesem Pod meldet, dass die Container gestartet wurden (ohne die Zeit für das Abrufen von Container-Images und die Initialisierung der Containerausführung). Um den SLO zu erfüllen, muss das 99. Perzentil pro Cluster-Tag dieser Pod-Startlatenz bei <=5 Sekunden bleiben.

Beachten Sie, dass bei diesem SLO davon ausgegangen wird, dass die Worker-Knoten in diesem Cluster bereits vorhanden sind und sich in einem Zustand befinden, in dem der Pod geplant werden kann. Dieses SLO berücksichtigt keine Image-Pulls oder Init-Container-Ausführungen und beschränkt den Test auch auf „statusfreie Pods“, die keine persistenten Speicher-Plug-ins nutzen.

Kubernetes SLI-Metriken

Kubernetes verbessert auch die Observability rund um die, SLIs indem es Prometheus-Metriken zu Kubernetes-Komponenten hinzufügt, die diese im Laufe der Zeit verfolgen. SLIs Mit der Prometheus Query Language (PromQL) können wir Abfragen erstellen, die die SLI-Leistung im Laufe der Zeit in Tools wie Prometheus- oder Grafana-Dashboards anzeigen. Im Folgenden finden Sie einige Beispiele dafür. SLOs

Latenz bei API-Serveranfragen

Metrik	Definition
apiserver_request_sli_duration_seconds	Verteilung der Antwortlatenz (ohne Webhook-Dauer und Wartezeiten in der Warteschl ange „Priority & Fairness“) in Sekunden für jedes Verb, jede Gruppe, Version, Ressource , Unterressource, Geltungsbereich und Komponente.
apiserver_request_duration_seconds	Verteilung der Antwortlatenz in Sekunden für jedes Verb, jeden Probelauf, jede Gruppe, Version, Ressource, Unterressource, jeden Bereich und jede Komponente.

> **Note**
>
> Die `apiserver_request_sli_duration_seconds` Metrik ist ab Kubernetes 1.27 verfügbar.

Sie können diese Metriken verwenden, um die Antwortzeiten des API-Servers zu untersuchen und festzustellen, ob es Engpässe bei den Kubernetes-Komponenten oder anderen Plugins/Komponenten gibt. Die folgenden Abfragen basieren auf dem SLO-Dashboard der Community.

API-Anforderungslatenz SLI (mutierend) — dieses Mal sind die Ausführung von Webhooks oder die Wartezeit in der Warteschlange nicht enthalten. `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`

API-Anforderungslatenz insgesamt (mutierend) — Dies ist die Gesamtzeit, die die Anfrage auf dem API-Server benötigt hat. Diese Zeit kann länger sein als die SLI-Zeit, da sie die Webhook-Ausführung sowie die Wartezeiten für API-Priorität und Fairness beinhaltet. `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
```

In diesen Abfragen schließen wir die Streaming-API-Anfragen aus, die nicht sofort zurückgegeben werden, wie z. B. `kubectl port-forward` or `kubectl exec` requests (`subresource!~"proxy|attach|log|exec|portforward"`), und wir filtern nur nach den Kubernetes-Verben, die objects () ändern. `verb=~"CREATE|DELETE|PATCH|POST|PUT"` Anschließend berechnen wir das 99. Perzentil dieser Latenz in den letzten 5 Minuten.

Wir können eine ähnliche Abfrage für die schreibgeschützten API-Anfragen verwenden. Wir ändern einfach die Verben, nach denen wir filtern, sodass sie die schreibgeschützten Aktionen und enthalten. `LIST` `GET` Je nach Umfang der Anfrage, d. h. beim Abrufen einer einzelnen Ressource oder beim Auflisten mehrerer Ressourcen, gibt es auch unterschiedliche SLO-Schwellenwerte.

SLI für API-Anforderungslatenz (schreibgeschützt) — diesmal sind die Ausführung von Webhooks oder die Wartezeit in der Warteschlange nicht enthalten. Für eine einzelne Ressource (scope=resource, threshold=1s) `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"GET", scope=~"resource"}[5m])) by (resource, subresource, verb, scope, le))`

Für eine Sammlung von Ressourcen im selben Namespace (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))`

Für eine Sammlung von Ressourcen im gesamten Cluster (scope=cluster, threshold=30s) `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"LIST", scope=~"cluster"}[5m])) by (resource, subresource, verb, scope, le))`

API-Anforderungslatenz insgesamt (schreibgeschützt) — Dies ist die Gesamtzeit, die die Anfrage auf dem API-Server in Anspruch genommen hat. Diese Zeit kann länger sein als die SLI-Zeit, da sie die Ausführung und Wartezeiten von Webhooks beinhaltet. Für eine einzelne Ressource (scope=resource, threshold=1s) `histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_bucket{verb=~"GET", scope=~"resource"}[5m])) by (resource, subresource, verb, scope, le))`

Für eine Sammlung von Ressourcen im selben Namespace (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))
```

Für eine Sammlung von Ressourcen im gesamten Cluster (scope=cluster, threshold=30s)

```
histogram_quantile(0.99,
sum(rate(apiserver_request_duration_seconds_bucket{verb=~"LIST",
scope=~"cluster"}[5m])) by (resource, subresource, verb, scope, le))
```

Die SLI-Metriken geben Aufschluss über die Leistung von Kubernetes-Komponenten, indem sie die Zeit ausklammern, die Anfragen in API-Prioritäts- und Fairness-Warteschlangen warten, Zulassungs-Webhooks oder andere Kubernetes-Erweiterungen bearbeiten. Die Gesamtmetriken bieten einen ganzheitlicheren Überblick, da sie die Zeit widerspiegeln, die Ihre Anwendungen auf eine Antwort vom API-Server warten würden. Ein Vergleich dieser Kennzahlen kann Aufschluss darüber geben, wo die Verzögerungen bei der Bearbeitung von Anfragen entstehen.

Latenz beim Starten des Pods

Metrik	Definition
kubelet_pod_start_sli_duration_seconds	Dauer in Sekunden bis zum Starten eines Pods, ohne die Zeit zum Abrufen von Bildern und zum Ausführen von Init-Containern, gemessen vom Zeitpunkt der Pod-Erste llung bis zu dem Zeitpunkt, zu dem alle seine Container als gestartet gemeldet und per Watch beobachtet gemeldet wurden
kubelet_pod_start_duration_seconds	Dauer in Sekunden von der ersten Erkennung eines Pods durch Kubelet bis zum Start des Pods. Dies beinhaltet nicht die Zeit für die Planung des Pods oder die Skalierung der Kapazität des Worker-Knotens.

> **Note**
>
> `kubelet_pod_start_sli_duration_seconds`ist ab Kubernetes 1.27 verfügbar.

Ähnlich wie bei den obigen Abfragen können Sie diese Metriken verwenden, um zu erfahren, wie lange Node-Skalierung, Image-Pulls und Init-Container den Pod-Start im Vergleich zu Kubelet-Aktionen verzögern.

Latenz beim Pod-Start (SLI) — das ist die Zeit von der Erstellung des Pods bis zu dem Zeitpunkt, zu dem die Anwendungscontainer als aktiv gemeldet wurden. Dies beinhaltet die Zeit, die benötigt wird, bis die Kapazität des Worker-Nodes verfügbar ist und der Pod geplant ist, aber nicht die Zeit, die zum Abrufen von Images oder zum Ausführen der Init-Container benötigt wird. `histogram_quantile(0.99, sum(rate(kubelet_pod_start_sli_duration_seconds_bucket[5m])) by (le))`

Latenz beim Pod-Start insgesamt — das ist die Zeit, die das Kubelet benötigt, um den Pod zum ersten Mal zu starten. Dies wird ab dem Zeitpunkt gemessen, zu dem das Kubelet den Pod über WATCH empfängt. Die Zeit für die Skalierung oder Planung des Worker-Nodes ist nicht enthalten. Dazu gehört auch die Zeit, die benötigt wird, um Images abzurufen und Container für die Ausführung zu initialisieren. `histogram_quantile(0.99, sum(rate(kubelet_pod_start_duration_seconds_bucket[5m])) by (le))`

SLOs auf Ihrem Cluster

Wenn Sie die Prometheus-Metriken aus den Kubernetes-Ressourcen in Ihrem EKS-Cluster sammeln, können Sie tiefere Einblicke in die Leistung der Komponenten der Kubernetes-Steuerebene gewinnen.

Das Perf-Tests-Repo enthält Grafana-Dashboards, in denen die Latenzen und kritischen Leistungskennzahlen für den Cluster während der Tests angezeigt werden. Die Perf-Tests-Konfiguration nutzt das kube-prometheus-stack, ein Open-Source-Projekt, das für die Erfassung von Kubernetes-Metriken konfiguriert ist. Sie können aber auch Amazon Managed Prometheus und Amazon Managed Grafana verwenden.

Wenn Sie die `kube-prometheus-stack` oder eine ähnliche Prometheus-Lösung verwenden, können Sie dasselbe Dashboard installieren, um die SLOs auf Ihrem Cluster in Echtzeit zu beobachten.

1. Sie müssen zuerst die Prometheus-Regeln installieren, die in den Dashboards mit verwendet werden. `kubectl apply -f prometheus-rules.yaml` Sie können eine Kopie der Regeln hier herunterladen: https://github.com/kubernetes/ perf- -rules.yaml tests/blob/master/ clusterloader2/pkg/prometheus/manifests/prometheus
   a. Stellen Sie sicher, dass der Namespace in der Datei zu Ihrer Umgebung passt

   b. Stellen Sie sicher, dass die Labels mit dem `prometheus.prometheusSpec.ruleSelector` Helm-Wert übereinstimmen, wenn Sie `kube-prometheus-stack`

2. Anschließend können Sie die Dashboards in Grafana installieren. Die JSON-Dashboards und Python-Skripte zu ihrer Generierung sind hier verfügbar: perf- https://github.com/kubernetes/ tests/tree/master/clusterloader2/pkg/prometheus/manifests/dashboards

   a. Das `slo.json` Dashboard zeigt die Leistung des Clusters im Verhältnis zu Kubernetes SLOs

Bedenken Sie, dass SLOs sie sich auf die Leistung der Kubernetes-Komponenten in Ihren Clustern konzentrieren, aber es gibt zusätzliche Metriken, die Sie überprüfen können und die unterschiedliche Perspektiven oder Einblicke in Ihren Cluster bieten. Kubernetes-Community-Projekte wie K ube-state-metrics können Ihnen helfen, Trends in Ihrem Cluster schnell zu analysieren. Die meisten gängigen Plugins und Treiber aus der Kubernetes-Community geben auch Prometheus-Metriken aus, sodass Sie Dinge wie Autoscaler oder benutzerdefinierte Scheduler untersuchen können.

Der Observability Best Practices Guide enthält Beispiele für andere Kubernetes-Metriken, anhand derer Sie weitere Einblicke gewinnen können.

Bekannte Limits und Service Quotas

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Amazon EKS kann für eine Vielzahl von Workloads verwendet werden und kann mit einer Vielzahl von AWS-Services interagieren. Wir haben festgestellt, dass Kunden-Workloads auf eine ähnliche Bandbreite von AWS-Servicekontingenten und andere Probleme stoßen, die die Skalierbarkeit beeinträchtigen.

Ihr AWS-Konto hat Standardkontingente (eine Obergrenze für die Anzahl der einzelnen AWS-Ressourcen, die Ihr Team anfordern kann). Jeder AWS-Service definiert sein eigenes Kontingent, und Kontingente sind im Allgemeinen regionsspezifisch. Für einige Kontingente können Sie Erhöhungen beantragen (weiche Limits), und andere Kontingente können nicht erhöht werden (harte Limits). Sie sollten diese Werte bei der Architektur Ihrer Anwendungen berücksichtigen. Erwägen Sie, diese Service-Limits regelmäßig zu überprüfen und sie bei der Entwicklung Ihrer Anwendung zu berücksichtigen.

Sie können die Nutzung in Ihrem Konto überprüfen und in der AWS-Servicekonsole oder mithilfe der AWS-CLI eine Anfrage zur Erhöhung des Kontingents stellen. Weitere Informationen zu den Service-Kontingenten und zu weiteren Einschränkungen oder Hinweisen zu deren Erhöhung finden Sie in der AWS-Dokumentation des jeweiligen AWS-Service.

> **Note**
>
> Amazon EKS Service Quotas listet die Servicekontingenten auf und enthält Links, über die Sie, sofern verfügbar, Erhöhungen beantragen können.

Andere AWS-Servicekontingente

Wir haben festgestellt, dass EKS-Kunden von den unten aufgeführten Kontingenten für andere AWS-Services betroffen waren. Einige davon gelten möglicherweise nur für bestimmte Anwendungsfälle oder Konfigurationen. Sie können jedoch in Betracht ziehen, ob Ihre Lösung bei der Skalierung auf diese Probleme stoßen wird. Die Kontingente sind nach Service organisiert und jedes Kontingent hat eine ID im Format L-XXXXXXXX, mit der Sie sie in der AWS-Servicekonsole nachschlagen können.

Service	Kontingent (L-xxxxx)	Auswirkung	Kennung (L-xxxxx)	default
IAM	Rollen pro Konto	Kann die Anzahl der Cluster oder IRSA-Rollen in einem Konto einschränken.	L- D64 FE177	1.000
IAM	OpenId Anbieter pro Konto verbinden	Kann die Anzahl der Cluster pro Konto begrenzen, OpenID Connect wird von IRSA verwendet	L-858F3967	100
IAM	Länge der Vertrauen	Kann die Anzahl der Cluster	L-C07B4B0D	2 048

Service	Kontingent (L-xxxxx)	Auswirkung	Kennung (L-xxxxx)	default
	srichtlinie für Rollen	begrenzen, denen eine IAM-Rolle für IRSA zugeordnet ist		
VPC	Sicherheitsgruppen pro Netzwerkschnittstelle	Kann die Steuerung oder Konnektivität des Netzwerks für Ihren Cluster einschränken	L-2 AFB9258	5
VPC	IPv4 CIDR-Blöcke pro VPC	Kann die Anzahl der EKS-Worker-Knoten begrenzen	L-83 A9D CA0	5
VPC	Routen pro Routing-Tabelle	Kann die Steuerung oder Konnektivität des Netzwerks für Ihren Cluster einschränken	L-93826ACB	50
VPC	Aktive VPC-Peering-Verbindungen pro VPC	Kann die Steuerung oder Konnektivität des Netzwerks für Ihren Cluster einschränken	L-7E9ECCDB	50

Service	Kontingent (L-xxxxx)	Auswirkung	Kennung (L-xxxxx)	default
VPC	Regeln für eingehenden oder ausgehenden Datenverkehr pro Sicherheitsgruppe.	Kann die Steuerung oder Konnektivität des Netzwerks für Ihren Cluster einschränken. Einige Controller in EKS erstellen neue Regeln	L-0 F EA8095	50
VPC	VPCs pro Region	Kann die Anzahl der Cluster pro Konto oder die Steuerung oder Konnektivität des Netzwerks für Ihren Cluster einschränken	L-F678F1CE	5
VPC	Internet-Gateways pro Region	Kann die Anzahl der Cluster pro Konto oder die Steuerung oder Konnektivität des Netzwerks für Ihren Cluster einschränken	L-A4707A72	5

Service	Kontingent (L-xxxxx)	Auswirkung	Kennung (L-xxxxx)	default
VPC	Netzwerks chnittstellen pro Region	Kann die Anzahl der EKS-Worke r-Knoten oder die Aktivitäten auf der EKS-Kontrollebene einschränken. scaling/update	L- DF5 E4 CA3	5,000
VPC	Netzwerka dressennutzung	Kann die Anzahl der Cluster pro Konto oder die Steuerung oder Konnektivität des Netzwerks für Ihren Cluster einschränken	L- BB24 F6E5	64.000
VPC	Peering-N etzwerkad ressennutzung	Kann die Anzahl der Cluster pro Konto oder die Steuerung oder Konnektivität des Netzwerks für Ihren Cluster einschränken	L- CD17 FD4 B	128.000
ELB	Listener pro Network Load Balancer	Kann die Kontrolle des in den Cluster eingehenden Datenverkehrs einschränken.	L-57A373D6	50

Service	Kontingent (L-xxxxx)	Auswirkung	Kennung (L-xxxxx)	default
ELB	Zielgruppen pro Region	Kann die Kontrolle des in den Cluster eingehenden Datenverkehrs einschränken.	L-B22855CB	3,000
ELB	Ziele pro Application Load Balancer	Kann die Kontrolle des in den Cluster eingehenden Datenverkehrs einschränken.	L-7E6692B2	1.000
ELB	Ziele pro Network Load Balancer	Kann die Kontrolle des in den Cluster eingehenden Datenverkehrs einschränken.	L- EEF1 AD04	3,000
ELB	Ziele pro Availability Zone und Network Load Balancer	Kann die Kontrolle des in den Cluster eingehenden Datenverkehrs einschränken.	L-B211E961	500
ELB	Ziele pro Zielgruppe und Region	Kann die Kontrolle des in den Cluster eingehenden Datenverkehrs einschränken.	L-A0D0B863	1.000

Service	Kontingent (L-xxxxx)	Auswirkung	Kennung (L-xxxxx)	default
ELB	Application Load Balancer pro Region	Kann die Kontrolle des in den Cluster eingehenden Datenverkehrs einschränken.	L-53 B97 DA6	50
ELB	Classic Load Balancer pro Region	Kann die Kontrolle des in den Cluster eingehenden Datenverkehrs einschränken.	L-E9E9831D	20
ELB	Network Load Balancers pro Region	Kann die Kontrolle des in den Cluster eingehenden Datenverkehrs einschränken.	L-69A177A2	50
EC2	Ausführen von On-Demand -Standard-Instances (A, C, D, H, I, M, R, T, Z) (als maximale vCPU-Anzahl)	Kann die Anzahl der EKS-Worker-Knoten begrenzen	L-1216C47A	5

Service	Kontingent (L-xxxxx)	Auswirkung	Kennung (L-xxxxx)	default
EC2	Alle Standard- (A, C, D, H, I, M, R, T, Z) Spot-Instance-Anfragen (als maximale vCPU-Anzahl)	Kann die Anzahl der EKS-Worker-Knoten begrenzen	L-34B43A08	5
EC2	EC2-VPC Elastisch IPs	Kann die Anzahl der NAT GWs (und damit VPCs) begrenzen, was die Anzahl der Cluster in einer Region einschränken kann	L-0263D0A3	5
EBS	Snapshots pro Region	Kann die Backup-Strategie für statusbehaftete Workloads einschränken	L-309 BACF6	100 000
EBS	Speicher für Universelle-SSD-Volumes (gp3), in TiB	Kann die Anzahl der EKS-Worker-Knoten oder PersistentVolume den Speicher begrenzen	L-7A658B76	50

Service	Kontingent (L-xxxxx)	Auswirkung	Kennung (L-xxxxx)	default
EBS	Speicher für Universelle-SSD-Volumes (gp2), in TiB	Kann die Anzahl der EKS-Worker-Knoten oder den Speicher begrenzen Persistent Volume	L-D18D FCD1	50
ECR	Registered repositories (Registrierte Repositorys)	Kann die Anzahl der Workloads in Ihren Clustern begrenzen	L- E8D CFEB8	100 000
ECR	Image pro Repository	Kann die Anzahl der Workloads in Ihren Clustern begrenzen	L-03A36 CE1	20 000
SecretsManager	Geheimnisse pro Region	Kann die Anzahl der Workloads in Ihren Clustern begrenzen	L-2F66C23C	500 000

Drosselung von AWS-Anfragen

AWS-Services implementieren auch Anforderungsdrosselung, um sicherzustellen, dass sie leistungsfähig und für alle Kunden verfügbar bleiben. Ähnlich wie bei Service Quotas behält jeder AWS-Service seine eigenen Schwellenwerte für die Einschränkung von Anfragen bei. Erwägen Sie, die entsprechende AWS-Servicedokumentation zu lesen, wenn Ihre Workloads schnell eine große Anzahl von API-Aufrufen ausgeben müssen oder wenn Sie in Ihrer Anwendung Fehler bei der Anforderungsdrosselung feststellen.

Bei EC2-API-Anfragen zur Bereitstellung von EC2-Netzwerkschnittstellen oder IP-Adressen kann es in großen Clustern oder bei drastischer Clusterskalierung zu einer Drosselung von Anfragen

kommen. In der folgenden Tabelle sind einige der API-Aktionen aufgeführt, bei denen Kunden unserer Erfahrung nach auf eine Drosselung von Anfragen gestoßen sind. Die Standardeinstellungen für das EC2-Ratenlimit und die Schritte zur Beantragung einer Erhöhung der Ratenbegrenzung finden Sie in der EC2-Dokumentation zur Ratendrosselung.

Mutierende Aktionen	Schreibgeschützte Aktionen
AssignPrivateIpAddresses	DescribeDhcpOptions
AttachNetworkInterface	DescribeInstances
CreateNetworkInterface	DescribeNetworkInterfaces
DeleteNetworkInterface	DescribeSecurityGroups
DeleteTags	DescribeTags
DetachNetworkInterface	DescribeVpcs
ModifyNetworkInterfaceAttribute	DescribeVolumes
UnassignPrivateIpAddresses	

Andere bekannte Grenzwerte

- Route 53 hat auch ein relativ niedriges Ratenlimit von 5 Anfragen pro Sekunde an die Route 53 53-API. Wenn Sie eine große Anzahl von Domains mit einem Projekt wie External DNS aktualisieren müssen, kann es zu Ratendrosselungen und Verzögerungen bei der Aktualisierung von Domains kommen.
  - Einige Nitro-Instance-Typen haben ein Limit für Volumenanhänge von 28, das von Amazon EBS-Volumes, Netzwerkschnittstellen und NVMe Instance-Speicher-Volumes gemeinsam genutzt wird. Wenn Ihre Workloads zahlreiche EBS-Volumes umfassen, kann es sein, dass die Pod-Dichte, die Sie mit diesen Instance-Typen erreichen können, begrenzt ist
  - Es gibt eine maximale Anzahl von Verbindungen, die pro Ec2-Instance verfolgt werden können. Wenn Ihre Workloads eine große Anzahl von Verbindungen verarbeiten, treten möglicherweise Kommunikationsausfälle oder -fehler auf, weil dieses Maximum erreicht wurde. Sie können die Messwerte `conntrack_allowance_available` und die

`conntrack_allowance_exceeded` Netzwerkleistungskennzahlen verwenden, um die Anzahl der verfolgten Verbindungen auf Ihren EKS-Worker-Knoten zu überwachen.

- In einer EKS-Umgebung liegt das etcd-Speicherlimit gemäß den Upstream-Richtlinien bei 8 GiB. Bitte überwachen Sie die Metrika`piserver_storage_size_bytes`, um die Größe der ETCD-Datenbank zu verfolgen. Sie können sich auf die Warnungsregeln `etcdExcessiveDatabaseGrowth` beziehen `etcdBackendQuotaLowSpace` und diese Überwachung einrichten.

Bewährte Methoden für Cluster-Upgrades

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

In diesem Handbuch erfahren Clusteradministratoren, wie sie ihre Amazon EKS-Upgrade-Strategie planen und ausführen können. Außerdem wird beschrieben, wie selbstverwaltete Knoten, verwaltete Knotengruppen, Karpenter-Knoten und Fargate-Knoten aktualisiert werden. Es beinhaltet keine Anleitungen zu EKS Anywhere, selbstverwaltetem Kubernetes, AWS Outposts oder AWS Local Zones.

-Übersicht

Eine Kubernetes-Version umfasst sowohl die Steuerungsebene als auch die Datenebene. Um einen reibungslosen Betrieb zu gewährleisten, sollten sowohl auf der Steuerungsebene als auch auf der Datenebene dieselbe Kubernetes-Nebenversion wie 1.24 ausgeführt werden. Während AWS die Kontrollebene verwaltet und aktualisiert, liegt die Aktualisierung der Worker-Knoten in der Datenebene in Ihrer Verantwortung.

- Kontrollebene — Die Version der Kontrollebene wird vom Kubernetes-API-Server bestimmt. In Amazon EKS-Clustern kümmert sich AWS um die Verwaltung dieser Komponente. Upgrades der Steuerungsebene können über die AWS-API initiiert werden.
- Datenebene — Die Datenebenenversion ist den Kubelet-Versionen zugeordnet, die auf Ihren einzelnen Knoten ausgeführt werden. Es ist möglich, dass Knoten im selben Cluster unterschiedliche Versionen ausführen. Sie können die Versionen aller Knoten überprüfen, indem Sie Folgendes ausführen`kubectl get nodes`.

Vor dem Upgrade

Wenn Sie planen, Ihre Kubernetes-Version in Amazon EKS zu aktualisieren, sollten Sie einige wichtige Richtlinien, Tools und Verfahren einrichten, bevor Sie mit einem Upgrade beginnen.

- Verstehen Sie die Richtlinien für veraltete Versionen — Verschaffen Sie sich ein tiefes Verständnis dafür, wie die Kubernetes-Richtlinie für veraltete Produkte funktioniert. Seien Sie sich aller

bevorstehenden Änderungen bewusst, die sich auf Ihre bestehenden Anwendungen auswirken könnten. In neueren Versionen von Kubernetes werden bestimmte APIs Funktionen häufig schrittweise eingestellt, was möglicherweise zu Problemen bei der Ausführung von Anwendungen führen kann.

- Kubernetes-Änderungsprotokoll überprüfen — Prüfen Sie das Kubernetes-Änderungsprotokoll zusammen mit den Amazon EKS Kubernetes-Versionen gründlich, um mögliche Auswirkungen auf Ihren Cluster zu verstehen, z. B. wichtige Änderungen, die sich auf Ihre Workloads auswirken könnten.
- Beurteilen Sie die Kompatibilität von Cluster-Add-Ons — Amazon EKS aktualisiert ein Add-on nicht automatisch, wenn neue Versionen veröffentlicht werden oder nachdem Sie Ihren Cluster auf eine neue Kubernetes-Nebenversion aktualisiert haben. Weitere Informationen zur Kompatibilität vorhandener Cluster-Add-Ons mit der Cluster-Version, auf die Sie ein Upgrade durchführen möchten, finden Sie unter Aktualisierung eines Add-ons.
- Protokollierung auf Kontrollebene aktivieren — Aktivieren Sie die Protokollierung auf der Kontrollebene, um Protokolle, Fehler oder Probleme aufzuzeichnen, die während des Upgrade-Vorgangs auftreten können. Erwägen Sie, diese Protokolle auf etwaige Anomalien zu überprüfen. Testen Sie Cluster-Upgrades in einer Umgebung außerhalb der Produktionsumgebung oder integrieren Sie automatisierte Tests in Ihren kontinuierlichen Integrationsablauf, um die Versionskompatibilität mit Ihren Anwendungen, Controllern und benutzerdefinierten Integrationen zu bewerten.
- Entdecken Sie eksctl für Clustermanagement — Erwägen Sie, eksctl zur Verwaltung Ihres EKS-Clusters zu verwenden. Es bietet Ihnen die Möglichkeit, die Steuerungsebene zu aktualisieren, Add-Ons zu verwalten und Worker-Node-Updates durchzuführen. out-of-the-box
- Entscheiden Sie sich für den automatischen Modus oder verwaltete Knotengruppen — Optimieren und automatisieren Sie Worker-Node-Upgrades mithilfe des Auto-Modus oder der von EKS verwalteten Knotengruppen. Diese Optionen vereinfachen den Prozess und reduzieren die Anzahl manueller Eingriffe.
- Nutzen Sie das kubectl Convert Plugin — Nutzen Sie das kubectl convert-Plugin, um die Konvertierung von Kubernetes-Manifestdateien zwischen verschiedenen API-Versionen zu erleichtern. Dadurch kann sichergestellt werden, dass Ihre Konfigurationen mit der neuen Kubernetes-Version kompatibel bleiben.

Behalten Sie Ihren Cluster up-to-date

Für eine sichere und effiziente EKS-Umgebung ist es von größter Bedeutung, über Kubernetes-Updates auf dem Laufenden zu bleiben. Dies spiegelt das Modell der gemeinsamen Verantwortung in Amazon EKS wider. Indem Sie diese Strategien in Ihren betrieblichen Arbeitsablauf integrieren, positionieren Sie sich in der Lage up-to-date, sichere Cluster zu verwalten, die alle Vorteile der neuesten Funktionen und Verbesserungen nutzen. Taktiken:

- Richtlinie für unterstützte Versionen — Amazon EKS ist auf die Kubernetes-Community abgestimmt und bietet in der Regel drei aktive Kubernetes-Versionen. Eine Kubernetes-Nebenversion wird in den ersten 14 Monaten nach ihrer Veröffentlichung standardmäßig in Amazon EKS unterstützt. Sobald eine Version das Ende des Standard-Supportdatums überschritten hat, wird für die folgenden 12 Monate ein erweiterter Support angeboten. Benachrichtigungen über veraltete Versionen werden mindestens 60 Tage vor dem Ende des Standard-Supports veröffentlicht. Weitere Informationen finden Sie in den Dokumenten zum Lebenszyklus der EKS-Version.
- Richtlinie für automatische Upgrades — Wir empfehlen dringend, mit den Kubernetes-Updates in Ihrem EKS-Cluster synchron zu bleiben. Cluster, die in einer Kubernetes-Version ausgeführt werden, deren 26-monatiger Lebenszyklus (14 Monate Standard-Support plus 12 Monate erweiterter Support) abgelaufen ist, werden automatisch auf die nächste Version aktualisiert. Beachten Sie, dass Sie den erweiterten Support deaktivieren können. Wenn Sie vor einer Version nicht proaktiv ein Upgrade durchführen, end-of-life wird ein automatisches Upgrade ausgelöst, wodurch Ihre Workloads und Systeme gestört werden könnten. Weitere Informationen finden Sie in der EKS-Version. FAQs
- Upgrade-Runbooks erstellen — Richten Sie einen gut dokumentierten Prozess für die Verwaltung von Upgrades ein. Entwickeln Sie im Rahmen Ihres proaktiven Ansatzes Runbooks und spezielle Tools, die auf Ihren Upgrade-Prozess zugeschnitten sind. Dies verbessert nicht nur Ihre Vorbereitung, sondern vereinfacht auch komplexe Umstellungen. Machen Sie es sich zur Standardpraxis, Ihre Cluster mindestens einmal pro Jahr zu aktualisieren. Diese Vorgehensweise passt Sie an den laufenden technologischen Fortschritt an und erhöht so die Effizienz und Sicherheit Ihrer Umgebung.

Sehen Sie sich den EKS-Veröffentlichungskalender an

Im EKS Kubernetes-Veröffentlichungskalender erfahren Sie, wann neue Versionen verfügbar sind und wann der Support für bestimmte Versionen endet. Im Allgemeinen veröffentlicht EKS jährlich drei Nebenversionen von Kubernetes, und jede Nebenversion wird etwa 14 Monate lang unterstützt.

Lesen Sie außerdem die Upstream-Versionsinformationen von Kubernetes.

Erfahren Sie, wie das Modell der gemeinsamen Verantwortung für Cluster-Upgrades gilt

Sie sind dafür verantwortlich, das Upgrade sowohl für die Clustersteuerungsebene als auch für die Datenebene zu initiieren. Erfahren Sie, wie Sie ein Upgrade initiieren. Wenn Sie ein Cluster-Upgrade initiieren, verwaltet AWS das Upgrade der Cluster-Kontrollebene. Sie sind für die Aktualisierung der Datenebene verantwortlich, einschließlich der Fargate-Pods und -Addons. Sie müssen Upgrades für Workloads, die auf Ihrem Cluster ausgeführt werden, validieren und planen, um sicherzustellen, dass deren Verfügbarkeit und Betrieb nach dem Cluster-Upgrade nicht beeinträchtigt werden

Führen Sie ein direktes Upgrade der Cluster durch

EKS unterstützt eine Strategie für ein direktes Cluster-Upgrade. Dadurch werden die Cluster-Ressourcen verwaltet und die Cluster-Konfiguration konsistent gehalten (z. B. API-Endpunkt, OIDC ENIs, Load Balancer). Dies ist für Cluster-Benutzer weniger störend und nutzt die vorhandenen Workloads und Ressourcen im Cluster, ohne dass Sie Workloads erneut bereitstellen oder externe Ressourcen (z. B. DNS, Speicher) migrieren müssen.

Bei der Durchführung eines direkten Cluster-Upgrades ist zu beachten, dass jeweils nur ein kleineres Versionsupgrade ausgeführt werden kann (z. B. von 1.24 auf 1.25).

Das bedeutet, dass, wenn Sie mehrere Versionen aktualisieren müssen, eine Reihe von aufeinanderfolgenden Upgrades erforderlich sind. Die Planung sequentieller Upgrades ist komplizierter und birgt ein höheres Ausfallrisiko. In dieser Situation finden Sie weitere Informationen unterthe section called “Evaluieren Sie Blue/Green Cluster als Alternative zu direkten Cluster-Upgrades”.

Rüsten Sie Ihre Steuerungsebene und Datenebene nacheinander auf

Um einen Cluster zu aktualisieren, müssen Sie die folgenden Maßnahmen ergreifen:

1. Lesen Sie die Versionshinweise zu Kubernetes und EKS.
2. Erstellen Sie eine Sicherungskopie des Clusters. (fakultativ)
3. Identifizieren und korrigieren Sie die veraltete und entfernte API-Nutzung in Ihren Workloads.
4. Stellen Sie sicher, dass sich Managed Node Groups, sofern sie verwendet werden, auf derselben Kubernetes-Version wie die Steuerungsebene befinden. EKS-verwaltete Knotengruppen und Knoten, die von EKS Fargate Profiles erstellt wurden, unterstützen zwei geringfügige Versionsunterschiede zwischen der Steuerungsebene und der Datenebene für Kubernetes Version 1.27 und niedriger. Ab Version 1.28 unterstützen von EKS verwaltete Knotengruppen und Knoten, die von EKS Fargate Profiles erstellt wurden, drei kleinere Versionsunterschiede zwischen Steuerungsebene und Datenebene. Wenn Ihre Version der EKS-Kontrollebene beispielsweise 1.28 ist, können Sie problemlos Kubelet-Versionen verwenden, die so alt sind wie 1.25. Wenn Ihre EKS-Version 1.27 ist, ist die älteste Kubelet-Version, die Sie verwenden können, 1.25.
5. Führen Sie ein Upgrade der Cluster-Steuerebene mithilfe der AWS-Konsole oder CLI durch.
6. Überprüfen Sie die Kompatibilität der Add-Ons. Aktualisieren Sie Ihre Kubernetes-Add-Ons und benutzerdefinierten Controller nach Bedarf.
7. Aktualisieren Sie kubectl.
8. Aktualisieren Sie die Cluster-Datenebene. Aktualisieren Sie Ihre Knoten auf dieselbe Kubernetes-Nebenversion wie Ihr aktualisiertes Cluster.

> **Tip**
>
> Wenn Ihr Cluster mit dem automatischen Modus von EKS erstellt wurde, müssen Sie Ihre Cluster-Datenebene nicht aktualisieren. Nach dem Upgrade Ihrer Steuerungsebene beginnt der automatische Modus von EKS mit der schrittweisen Aktualisierung der verwalteten Knoten und berücksichtigt dabei alle Budgets für Pod-Unterbrechungen. Achten Sie darauf, diese Updates zu überwachen, um sicherzustellen, dass Ihre betrieblichen Anforderungen eingehalten werden.

Verwenden Sie die EKS-Dokumentation, um eine Upgrade-Checkliste zu erstellen

Die EKS Kubernetes-Versionsdokumentation enthält eine detaillierte Liste der Änderungen für jede Version. Erstellen Sie für jedes Upgrade eine Checkliste.

Spezifische Hinweise zum Upgrade von EKS-Versionen finden Sie in der Dokumentation auf wichtige Änderungen und Überlegungen für jede Version.

- Lesen Sie die Versionshinweise für Kubernetes-Versionen zur Standardunterstützung
- Lesen Sie die Versionshinweise für Kubernetes-Versionen zum erweiterten Support

Aktualisieren Sie Add-Ons und Komponenten mithilfe der Kubernetes-API

Bevor Sie einen Cluster aktualisieren, sollten Sie wissen, welche Versionen der Kubernetes-Komponenten Sie verwenden. Inventarisieren Sie die Cluster-Komponenten und identifizieren Sie Komponenten, die die Kubernetes-API direkt verwenden. Dazu gehören wichtige Cluster-Komponenten wie Überwachungs- und Logging-Agenten, Cluster-Autoscaler, Container-Speichertreiber (z. B. EBS CSI, EFS CSI), Ingress-Controller und alle anderen Workloads oder Add-Ons, die direkt auf der Kubernetes-API basieren.

> **Tip**
>
> Kritische Clusterkomponenten werden häufig in einem Namespace installiert `*-system`

```
kubectl get ns | grep '-system'
```

Sobald Sie Komponenten identifiziert haben, die auf der Kubernetes-API basieren, überprüfen Sie deren Dokumentation auf Versionskompatibilität und Upgrade-Anforderungen. Informationen zur Versionskompatibilität finden Sie beispielsweise in der Dokumentation zum AWS Load Balancer Controller. Einige Komponenten müssen möglicherweise aktualisiert oder die Konfiguration geändert werden, bevor mit einem Cluster-Upgrade fortgefahren werden kann. Zu den wichtigsten Komponenten, die überprüft werden müssen, gehören CoreDNS, Kube-Proxy, VPC CNI und Speichertreiber.

Cluster enthalten oft viele Workloads, die die Kubernetes-API verwenden und für Workload-Funktionen wie Ingress-Controller, Continuous-Delivery-Systeme und Überwachungstools erforderlich sind. Wenn Sie einen EKS-Cluster aktualisieren, müssen Sie auch Ihre Add-Ons und Tools von Drittanbietern aktualisieren, um sicherzustellen, dass sie kompatibel sind.

Sehen Sie sich die folgenden Beispiele für gängige Add-Ons und die entsprechende Upgrade-Dokumentation an:

- Amazon VPC CNI: Die empfohlene Version des Amazon VPC CNI-Add-ons für jede Cluster-Version finden Sie unter Aktualisieren des Amazon VPC CNI-Plug-ins für das selbstverwaltete Kubernetes-Add-on. Wenn es als Amazon EKS Add-on installiert wird, kann es jeweils nur für eine Nebenversion aktualisiert werden.
- kube-proxy: Weitere Informationen finden Sie unter Aktualisierung des selbstverwalteten Kubernetes-Kube-Proxy-Add-ons.
- CoreDNS: Siehe Aktualisierung des selbstverwalteten CoreDNS-Add-ons.
- AWS Load Balancer Controller: Der AWS Load Balancer Controller muss mit der von Ihnen bereitgestellten EKS-Version kompatibel sein. Weitere Informationen finden Sie in der Installationsanleitung.
- Amazon Elastic Block Store (Amazon EBS) Container Storage Interface (CSI) -Treiber: Informationen zur Installation und zum Upgrade finden Sie unter Den Amazon EBS CSI-Treiber als Amazon EKS-Add-on verwalten.
- Amazon Elastic File System (Amazon EFS) Container Storage Interface (CSI) -Treiber: Informationen zur Installation und zum Upgrade finden Sie unter Amazon EFS CSI-Treiber.
- Kubernetes Metrics Server: Weitere Informationen finden Sie unter metrics-server on. GitHub
- Kubernetes Cluster Autoscaler: Um die Version von Kubernetes Cluster Autoscaler zu aktualisieren, ändern Sie die Version des Images in der Bereitstellung. Der Cluster Autoscaler ist eng mit dem Kubernetes-Scheduler verbunden. Sie müssen ihn immer aktualisieren, wenn Sie den Cluster aktualisieren. Überprüfen Sie die GitHub Versionen, um die Adresse der neuesten Version zu finden, die Ihrer Kubernetes-Nebenversion entspricht.
- Karpenter: Informationen zur Installation und zum Upgrade finden Sie in der Karpenter-Dokumentation.

> **Tip**
>
> Sie müssen keine der Funktionen von Amazon EKS Auto Mode manuell aktualisieren, einschließlich der Funktionen für automatische Rechenskalierung, Blockspeicher und Lastenausgleich.

Überprüfen Sie vor dem Upgrade die grundlegenden EKS-Anforderungen

AWS benötigt bestimmte Ressourcen in Ihrem Konto, um den Upgrade-Vorgang abzuschließen. Wenn diese Ressourcen nicht vorhanden sind, kann der Cluster nicht aktualisiert werden. Für ein Upgrade der Steuerungsebene sind die folgenden Ressourcen erforderlich:

1. Verfügbare IP-Adressen: Amazon EKS benötigt bis zu fünf verfügbare IP-Adressen aus den Subnetzen, die Sie bei der Erstellung des Clusters angegeben haben, um den Cluster zu aktualisieren. Falls nicht, aktualisieren Sie Ihre Cluster-Konfiguration, sodass sie neue Cluster-Subnetze enthält, bevor Sie das Versionsupdate durchführen.
2. EKS-IAM-Rolle: Die IAM-Rolle der Steuerungsebene ist weiterhin im Konto vorhanden und verfügt über die erforderlichen Berechtigungen.
3. Wenn in Ihrem Cluster die geheime Verschlüsselung aktiviert ist, stellen Sie sicher, dass die Cluster-IAM-Rolle berechtigt ist, den AWS Key Management Service (AWS KMS) -Schlüssel zu verwenden.

Überprüfen Sie die verfügbaren IP-Adressen

Um den Cluster zu aktualisieren, benötigt Amazon EKS bis zu fünf verfügbare IP-Adressen aus den Subnetzen, die beim Erstellen des Clusters bereitgestellt wurden.

Um zu überprüfen, ob Ihre Subnetze über genügend IP-Adressen für ein Upgrade des Clusters verfügen, können Sie den folgenden Befehl ausführen:

```
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 |
+--------------------------+-------------+-------+
```

Der VPC CNI Metrics Helper kann verwendet werden, um ein CloudWatch Dashboard für VPC-Metriken zu erstellen. Amazon EKS empfiehlt, die Cluster-Subnetze vor Beginn eines Kubernetes-Versionsupgrades mithilfe der UpdateClusterConfiguration "" API zu aktualisieren, wenn Ihnen die IP-Adressen in den Subnetzen ausgehen, die ursprünglich bei der Clustererstellung angegeben wurden. Bitte überprüfen Sie, ob Ihnen die neuen Subnetze zur Verfügung gestellt werden:

- gehören zu derselben Gruppe von AZs denen, die bei der Clustererstellung ausgewählt wurden.
- gehören zu derselben VPC, die bei der Clustererstellung bereitgestellt wurde

Bitte erwägen Sie, zusätzliche CIDR-Blöcke zuzuordnen, falls die IP-Adressen im vorhandenen VPC-CIDR-Block aufgebraucht sind. AWS ermöglicht die Zuordnung zusätzlicher CIDR-Blöcke zu Ihrer vorhandenen Cluster-VPC, wodurch Ihr IP-Adresspool effektiv erweitert wird. Diese Erweiterung kann durch die Einführung zusätzlicher privater IP-Bereiche (RFC 1918) oder, falls erforderlich, öffentlicher IP-Bereiche (nicht RFC 1918) erreicht werden. Sie müssen neue VPC-CIDR-Blöcke hinzufügen und warten, bis die VPC-Aktualisierung abgeschlossen ist, bevor Amazon EKS das neue CIDR verwenden kann. Danach können Sie die Subnetze basierend auf den neu eingerichteten CIDR-Blöcken für die VPC aktualisieren.

Überprüfen Sie die EKS-IAM-Rolle

Um zu überprüfen, ob die IAM-Rolle verfügbar ist und in Ihrem Konto über die richtige Richtlinie „Rolle annehmen“ verfügt, können Sie die folgenden Befehle ausführen:

```
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"
        }
    ]
}
```

Migrieren Sie zu EKS Add-ons

Amazon EKS installiert automatisch Add-Ons wie das Amazon VPC CNI-Plugin für Kubernetes und CoreDNS für `kube-proxy` jeden Cluster. Add-Ons können selbst verwaltet oder als Amazon EKS-Add-Ons installiert werden. Amazon EKS Add-ons ist eine alternative Methode zur Verwaltung von Add-Ons mithilfe der EKS-API.

Sie können Amazon EKS Add-ons verwenden, um Versionen mit einem einzigen Befehl zu aktualisieren. Zum Beispiel:

```
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
```

Prüfen Sie, ob Sie über EKS-Add-ons verfügen mit:

```
aws eks list-addons --cluster-name <cluster name>
```

> **Warning**
>
> EKS-Add-Ons werden während eines Upgrades der Steuerungsebene nicht automatisch aktualisiert. Sie müssen EKS-Add-On-Updates initiieren und die gewünschte Version auswählen.

- Sie sind dafür verantwortlich, aus allen verfügbaren Versionen eine kompatible Version auszuwählen. Lesen Sie die Hinweise zur Kompatibilität von Add-On-Versionen.
- Amazon EKS Add-ons können jeweils nur für eine Nebenversion aktualisiert werden.

Erfahren Sie mehr darüber, welche Komponenten als EKS-Add-ons verfügbar sind und wie Sie damit beginnen können.

Erfahren Sie, wie Sie ein EKS-Add-on mit einer benutzerdefinierten Konfiguration ausstatten.

Identifizieren und korrigieren Sie die fehlende API-Nutzung, bevor Sie die Steuerungsebene aktualisieren

Sie sollten die API-Nutzung von removed identifizieren, APIs bevor Sie Ihre EKS-Kontrollebene aktualisieren. Zu diesem Zweck empfehlen wir die Verwendung von Tools, die einen laufenden Cluster oder statische, gerenderte Kubernetes-Manifestdateien überprüfen können.

Die Prüfung anhand statischer Manifestdateien durchzuführen, ist im Allgemeinen genauer. Wenn diese Tools auf Live-Clustern ausgeführt werden, geben sie möglicherweise falsch positive Ergebnisse zurück.

Eine veraltete Kubernetes-API bedeutet nicht, dass die API entfernt wurde. Sie sollten die Kubernetes-Richtlinie für veraltete Versionen überprüfen, um zu erfahren, wie sich das Entfernen der API auf Ihre Workloads auswirkt.

Einblicke in Cluster

Cluster Insights ist eine Funktion, die Erkenntnisse zu Problemen liefert, die sich auf die Möglichkeit auswirken können, einen EKS-Cluster auf neuere Versionen von Kubernetes zu aktualisieren. Diese Ergebnisse werden von Amazon EKS kuratiert und verwaltet und bieten Empfehlungen zu deren Behebung. Durch die Nutzung von Cluster Insights können Sie den Aufwand für ein Upgrade auf neuere Kubernetes-Versionen minimieren.

Um Einblicke in einen EKS-Cluster zu erhalten, können Sie den folgenden Befehl ausführen:

```
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."
        }
    ]
}
```


Für eine aussagekräftigere Ausgabe der erhaltenen Erkenntnisse können Sie den folgenden Befehl ausführen:

```
aws eks describe-insight --region <region-code> --id <insight-id> --cluster-name <my-
cluster>
```

Sie haben auch die Möglichkeit, Einblicke in der Amazon EKS-Konsole einzusehen. Nachdem Sie Ihren Cluster aus der Cluster-Liste ausgewählt haben, befinden sich die Insight-Ergebnisse unter der `Upgrade Insights` Registerkarte.

Wenn Sie einen Cluster-Einblick bei finden`"status": ERROR`, müssen Sie das Problem beheben, bevor Sie das Cluster-Upgrade durchführen. Führen Sie den `aws eks describe-insight` Befehl aus, der Ihnen die folgenden Hinweise zur Problembehebung enthält:

Betroffene Ressourcen:


```
"resources": [
      {
        "insightStatus": {
          "status": "ERROR"
        },
        "kubernetesResourceUri": "/apis/policy/v1beta1/podsecuritypolicies/null"
      }
]
```

APIs veraltet:

```
"deprecationDetails": [
      {
        "usage": "/apis/flowcontrol.apiserver.k8s.io/v1beta2/flowschemas",
        "replacedWith": "/apis/flowcontrol.apiserver.k8s.io/v1beta3/flowschemas",
        "stopServingVersion": "1.29",
        "clientStats": [],
        "startServingReplacementVersion": "1.26"
      }
]
```

Empfohlene Maßnahme:

```
"recommendation": "Update manifests and API clients to use newer Kubernetes APIs if
 applicable before upgrading to Kubernetes v1.26."
```

Die Nutzung von Cluster-Erkenntnissen über die EKS-Konsole oder CLI trägt dazu bei, den Prozess der erfolgreichen Aktualisierung von EKS-Clusterversionen zu beschleunigen. Erfahren Sie mehr mit den folgenden Ressourcen: * Offizielle EKS-Dokumente * Blog zum Start von Cluster Insights.

K ube-no-trouble

K ube-no-trouble ist ein Open-Source-Befehlszeilenprogramm mit dem Befehl`kubent`. Wenn Sie es `kubent` ohne Argumente ausführen, verwendet es Ihren aktuellen KubeConfig Kontext, scannt den Cluster und druckt einen Bericht mit dem, was veraltet ist und entfernt APIs wird.

```
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)
```

Es kann auch verwendet werden, um statische Manifestdateien und Helm-Pakete zu scannen. Es wird empfohlen, es im `kubent` Rahmen eines Continuous Integration (CI) -Prozesses zu verwenden, um Probleme zu identifizieren, bevor die Manifeste bereitgestellt werden. Das Scannen von Manifesten ist auch genauer als das Scannen von Live-Clustern.

Kube-no-trouble bietet ein Beispiel für ein Dienstkonto und eine Rolle mit den entsprechenden Berechtigungen für das Scannen des Clusters.

Pluto

Eine weitere Option ist Pluto, das dem ähnelt, `kubent` weil es das Scannen eines Live-Clusters, von Manifestdateien und Helmdiagrammen unterstützt und über eine GitHub Aktion verfügt, die Sie in Ihren CI-Prozess aufnehmen können.

```
pluto detect-all-in-cluster

NAME             KIND                VERSION          REPLACEMENT   REMOVED
 DEPRECATED   REPL AVAIL
eks.privileged   PodSecurityPolicy   policy/v1beta1                 false     true
    true
```

Ressourcen

Um APIs vor dem Upgrade sicherzustellen, dass Ihr Cluster nicht veraltete Versionen verwendet, sollten Sie Folgendes überwachen:

- Metrik `apiserver_requested_deprecated_apis` seit 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
```

- Ereignisse in den Audit-Logs mit `k8s.io/deprecated` folgender Einstellung: `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
```

Was gibt Zeilen aus, wenn sie verwendet werden, wenn sie veraltet APIs sind:

```
{
    "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\"}}"
            },
[...]
```

Aktualisieren Sie Kubernetes-Workloads. Verwenden Sie kubectl-convert, um Manifeste zu aktualisieren

Nachdem Sie herausgefunden haben, welche Workloads und Manifeste aktualisiert werden müssen, müssen Sie möglicherweise den Ressourcentyp in Ihren Manifestdateien ändern

(z. B. in). PodSecurityPolicies PodSecurityStandards Dies erfordert eine Aktualisierung der Ressourcenspezifikation und zusätzliche Nachforschungen, je nachdem, welche Ressource ersetzt wird.

Wenn der Ressourcentyp gleich bleibt, die API-Version jedoch aktualisiert werden muss, können Sie den `kubectl-convert` Befehl verwenden, um Ihre Manifestdateien automatisch zu konvertieren. Zum Beispiel, um ein älteres Deployment zu zu konvertieren`apps/v1`. Weitere Informationen finden Sie unter Installieren des kubectl convert-Plugins auf der Kubernetes-Website.

```
kubectl-convert -f <file> --output-version <group>/<version>
```

Konfigurieren Sie Ihre Workloads PodDisruptionBudgets und topologySpreadConstraints stellen Sie deren Verfügbarkeit sicher, während die Datenebene aktualisiert wird

Stellen Sie sicher, dass Ihre Workloads über die richtigen PodDisruptionBudgetsund topologySpreadConstraintsverfügbaren Workloads verfügen, während die Datenebene aktualisiert wird. Nicht jeder Workload erfordert das gleiche Maß an Verfügbarkeit, daher müssen Sie den Umfang und die Anforderungen Ihres Workloads überprüfen.

Wenn Sie sicherstellen, dass Workloads auf mehrere Availability Zones und auf mehrere Hosts mit Topologieverteilungen verteilt sind, können Sie sich darauf verlassen, dass Workloads automatisch und ohne Zwischenfälle auf die neue Datenebene migriert werden.

Hier ist ein Beispiel für einen Workload, bei dem immer 80% der Replikate verfügbar sind und die Replikate auf Zonen und Hosts verteilt sind

```
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 hat Amazon Elastic Kubernetes Service (Amazon EKS) als unterstützte Ressource hinzugefügt. Resilience Hub bietet einen zentralen Ort, an dem Sie die Widerstandsfähigkeit Ihrer Anwendungen definieren, validieren und verfolgen können, sodass Sie unnötige Ausfallzeiten aufgrund von Software-, Infrastruktur- oder Betriebsunterbrechungen vermeiden können.

Verwenden Sie Managed Node Groups oder Karpenter, um Upgrades auf Datenebene zu vereinfachen

Managed Node Groups und Karpenter vereinfachen beide Knoten-Upgrades, verfolgen jedoch unterschiedliche Ansätze.

Verwaltete Knotengruppen automatisieren die Bereitstellung und das Lebenszyklusmanagement von Knoten. Das bedeutet, dass Sie Knoten mit einem einzigen Vorgang erstellen, automatisch aktualisieren oder beenden können.

In der Standardkonfiguration erstellt Karpenter automatisch neue Knoten mit dem neuesten kompatiblen EKS-optimierten AMI. Sobald EKS das aktualisierte EKS Optimized veröffentlicht AMIs oder der Cluster aktualisiert wird, verwendet Karpenter automatisch diese Images. Karpenter implementiert auch Node Expiry, um Knoten zu aktualisieren.

Karpenter kann so konfiguriert werden, dass es benutzerdefiniert verwendet wird. AMIs Wenn Sie Custom AMIs mit Karpenter verwenden, sind Sie für die Version von Kubelet verantwortlich.

Bestätigen Sie die Versionskompatibilität mit vorhandenen Knoten und der Steuerungsebene

Bevor Sie mit einem Kubernetes-Upgrade in Amazon EKS fortfahren, müssen Sie unbedingt die Kompatibilität zwischen Ihren verwalteten Knotengruppen, selbstverwalteten Knoten und der Kontrollebene sicherstellen. Die Kompatibilität hängt von der Kubernetes-Version ab, die Sie verwenden, und sie variiert je nach Szenario. Taktik:

- Kubernetes v1.28+ — * * Ab Kubernetes Version 1.28 und höher gibt es eine mildere Versionsrichtlinie für Kernkomponenten. Insbesondere wurde der unterstützte Unterschied zwischen dem Kubernetes-API-Server und dem Kubelet um eine Nebenversion erweitert, und zwar von n-2 auf n-3. Wenn Ihre Version der EKS-Kontrollebene beispielsweise 1.28 ist, können Sie problemlos Kubelet-Versionen verwenden, die so alt sind wie 1.25. Dieser Versionsversatz wird in AWS Fargate, verwalteten Knotengruppen und selbstverwalteten Knoten unterstützt. Aus Sicherheitsgründen empfehlen wir dringend, Ihre Amazon Machine Image (AMI) up-to-date - Versionen beizubehalten. Ältere Kubelet-Versionen können aufgrund potenzieller allgemeiner Sicherheitslücken und Risiken (CVEs) Sicherheitsrisiken bergen, die die Vorteile der Verwendung älterer Kubelet-Versionen überwiegen könnten.

- Kubernetes < v1.28 — Wenn Sie eine ältere Version als v1.28 verwenden, beträgt der unterstützte Unterschied zwischen dem API-Server und dem Kubelet n-2. Wenn Ihre EKS-Version beispielsweise 1.27 ist, ist die älteste Kubelet-Version, die Sie verwenden können, 1.25. Dieser Versionsunterschied gilt für AWS Fargate, verwaltete Knotengruppen und selbstverwaltete Knoten.

Aktivieren Sie das Ablaufen von Knoten für von Karpenter verwaltete Knoten

Eine Möglichkeit, wie Karpenter Knoten-Upgrades implementiert, ist die Verwendung des Konzepts des Ablaufs von Knoten. Dadurch wird der Planungsaufwand für Knoten-Upgrades reduziert. Wenn Sie in Ihrem Provisioner einen Wert für ttlSecondsUntilAbgelaufen festlegen, wird dadurch das Ablaufen des Knotens aktiviert. Sobald Knoten innerhalb von Sekunden das definierte Alter erreicht haben, werden sie sicher entleert und gelöscht. Dies gilt auch dann, wenn sie verwendet werden, sodass Sie Knoten durch neu bereitgestellte, aktualisierte Instanzen ersetzen können. Wenn ein Knoten ersetzt wird, verwendet Karpenter die neueste EKS-optimierte Version. AMIs Weitere Informationen finden Sie unter Deprovisioning auf der Karpenter-Website.

Karpenter fügt diesem Wert nicht automatisch Jitter hinzu. Um übermäßige Unterbrechungen der Arbeitslast zu vermeiden, definieren Sie ein Budget für Pod-Unterbrechungen, wie in der Kubernetes-Dokumentation beschrieben.

Wenn Sie ttlSecondsUntilExpired auf einem Provisioner konfigurieren, gilt dies für bestehende Knoten, die dem Provisioner zugeordnet sind.

Verwenden Sie die Drift-Funktion für von Karpenter verwaltete Knoten

Die Drift-Funktion von Karpenter kann die von Karpenter bereitgestellten Knoten automatisch aktualisieren, um mit der EKS-Steuerebene synchron zu bleiben. Karpenter Drift muss derzeit mithilfe eines Feature-Gates aktiviert werden. Die Standardkonfiguration von Karpenter verwendet das neueste EKS-optimierte AMI für dieselbe Haupt- und Nebenversion wie die Steuerungsebene des EKS-Clusters.

Nach Abschluss eines EKS-Cluster-Upgrades erkennt die Drift-Funktion von Karpenter, dass die von Karpenter bereitgestellten Knoten EKS-optimiert AMIs für die vorherige Clusterversion verwenden, und sperrt, entleert und ersetzt diese Knoten automatisch. Um die Migration von Pods

auf neue Knoten zu unterstützen, folgen Sie den bewährten Methoden von Kubernetes, indem Sie entsprechende Pod-Ressourcenkontingente festlegen und Budgets für Pod-Unterbrechungen (PDB) verwenden. Bei der Deprovisionierung von Karpenter werden auf der Grundlage der Pod-Ressourcenanforderungen vorab Ersatzknoten eingerichtet und die bei der Deprovisionierung von Knoten festgelegten Regeln berücksichtigt. PDBs

Verwenden Sie eksctl, um Upgrades für selbstverwaltete Knotengruppen zu automatisieren

Selbstverwaltete Knotengruppen sind EC2-Instances, die in Ihrem Konto bereitgestellt und außerhalb des EKS-Dienstes an den Cluster angehängt wurden. Diese werden in der Regel mithilfe von Automatisierungstools bereitgestellt und verwaltet. Informationen zum Upgrade von selbstverwalteten Knotengruppen finden Sie in der Dokumentation zu Ihren Tools.

eksctl unterstützt beispielsweise das Löschen und Entleeren von selbstverwalteten Knoten.

Zu den gängigen Tools gehören:

- eksctl
- OPs
- EKS-Entwürfe

Backup den Cluster vor dem Upgrade

Neue Versionen von Kubernetes führen zu erheblichen Änderungen an Ihrem Amazon EKS-Cluster. Nachdem Sie einen Cluster aktualisiert haben, können Sie ihn nicht mehr herabstufen.

Velero ist ein von der Community unterstütztes Open-Source-Tool, mit dem Sie Backups vorhandener Cluster erstellen und die Backups auf einen neuen Cluster anwenden können.

Beachten Sie, dass Sie nur neue Cluster für Kubernetes-Versionen erstellen können, die derzeit von EKS unterstützt werden. Wenn die Version, die Ihr Cluster derzeit ausführt, weiterhin unterstützt wird und ein Upgrade fehlschlägt, können Sie einen neuen Cluster mit der Originalversion erstellen und die Datenebene wiederherstellen. Beachten Sie, dass AWS-Ressourcen, einschließlich IAM, nicht im Backup von Velero enthalten sind. Diese Ressourcen müssten neu erstellt werden.

Starten Sie die Fargate-Bereitstellungen nach dem Upgrade der Steuerungsebene neu

Um die Fargate-Datenebenenknoten zu aktualisieren, müssen Sie die Workloads erneut bereitstellen. Sie können feststellen, welche Workloads auf Fargate-Knoten ausgeführt werden, indem Sie alle Pods mit der Option auflisten. `-o wide` Jeder Knotenname, der mit `fargate-` beginnt, muss im Cluster erneut bereitgestellt werden.

Evaluieren Sie Blue/Green Cluster als Alternative zu direkten Cluster-Upgrades

Einige Kunden bevorzugen eine blue/green Upgrade-Strategie. Dies kann Vorteile haben, beinhaltet aber auch Nachteile, die berücksichtigt werden sollten.

Zu den Vorteilen gehören:

- Es ist möglich, mehrere EKS-Versionen gleichzeitig zu ändern (z. B. 1.23 auf 1.25)
- Kann zurück zum alten Cluster wechseln
- Erzeugt einen neuen Cluster, der mit neueren Systemen (z. B. Terraform) verwaltet werden kann
- Workloads können einzeln migriert werden

Zu den Nachteilen gehören:

- API-Endpunkt und OIDC-Änderung, die eine Aktualisierung der Verbraucher erfordert (z. B. kubectl und CI/CD)
- Erfordert, dass während der Migration 2 Cluster parallel ausgeführt werden, was teuer sein und die Kapazität der Region einschränken kann
- Wenn Workloads voneinander abhängen und gemeinsam migriert werden müssen, ist mehr Koordination erforderlich
- Load Balancer und externes DNS können sich nicht einfach über mehrere Cluster erstrecken

Diese Strategie ist zwar möglich, aber sie ist teurer als ein Upgrade vor Ort und erfordert mehr Zeit für die Koordination und Workload-Migrationen. Sie kann in einigen Situationen erforderlich sein und sollte sorgfältig geplant werden.

Bei einem hohen Automatisierungsgrad und solchen GitOps deklarativen Systemen ist dies möglicherweise einfacher. Sie müssen zusätzliche Vorsichtsmaßnahmen für statusbehaftete Workloads treffen, sodass Daten gesichert und auf neue Cluster migriert werden.

Weitere Informationen finden Sie in diesen Blogbeiträgen:

- Kubernetes-Cluster-Upgrade: Die blaugrüne Bereitstellungsstrategie
- Migration von blauen/grünen oder kanarischen Amazon EKS-Clustern für statusfreie ArgoCD-Workloads

Verfolgen Sie geplante größere Änderungen im Kubernetes-Projekt — denken Sie voraus

Schauen Sie sich nicht nur die nächste Version an. Überprüfen Sie neue Versionen von Kubernetes, sobald sie veröffentlicht werden, und identifizieren Sie wichtige Änderungen. Beispielsweise verwendeten einige Anwendungen direkt die Docker-API, und die Unterstützung für Container Runtime Interface (CRI) für Docker (auch bekannt als Dockershim) wurde in Kubernetes entfernt. `1.24` Diese Art von Änderung erfordert mehr Zeit, um sich darauf vorzubereiten.

Lesen Sie alle dokumentierten Änderungen für die Version, auf die Sie aktualisieren, und notieren Sie sich alle erforderlichen Upgrade-Schritte. Beachten Sie auch alle Anforderungen oder Verfahren, die für von Amazon EKS verwaltete Cluster spezifisch sind.

- Kubernetes-Changelog

Spezifische Hinweise zum Entfernen von Funktionen

Entfernung von Dockershim in 1.25 — Verwenden Sie Detector for Docker Socket (DDS)

Das EKS-optimierte AMI für 1.25 bietet keine Unterstützung mehr für Dockersheim. Wenn Sie von Dockershim abhängig sind, z. B. wenn Sie den Docker-Socket mounten, müssen Sie diese Abhängigkeiten entfernen, bevor Sie Ihre Worker-Knoten auf 1.25 aktualisieren.

Suchen Sie nach Instanzen, in denen Sie vom Docker-Socket abhängig sind, bevor Sie auf 1.25 aktualisieren. Wir empfehlen die Verwendung von Detector for Docker Socket (DDS), einem Kubectl-Plugin. .

Entfernung von PodSecurityPolicy in 1.25 — Migration zu Pod-Sicherheitsstandards oder einer Lösung policy-as-code

`PodSecurityPolicy`war in Kubernetes 1.21 veraltet und wurde in Kubernetes 1.25 entfernt. Wenn Sie PodSecurityPolicy in Ihrem Cluster verwenden, müssen Sie zu den integrierten Kubernetes Pod Security Standards (PSS) oder zu einer policy-as-code Lösung migrieren, bevor Sie Ihren Cluster auf Version 1.25 aktualisieren, um Unterbrechungen Ihrer Workloads zu vermeiden.

AWS hat eine ausführliche FAQ in der EKS-Dokumentation veröffentlicht.

Lesen Sie die bewährten Methoden für Pod Security Standards (PSS) und Pod Security Admission (PSA).

Lesen Sie den PodSecurityPolicyDeprecation-Blogbeitrag auf der Kubernetes-Website.

Der In-Tree-Storage-Treiber in Version 1.23 ist veraltet — Migration zu Container-Storage-Interface-Treibern (CSI)

Das Container Storage Interface (CSI) wurde entwickelt, um Kubernetes dabei zu unterstützen, seine bestehenden Treibermechanismen für In-Tree-Speicher zu ersetzen. Die Funktion zur Migration des Container Storage Interface (CSI) von Amazon EBS ist in Clustern von Amazon EKS `1.23` und höher standardmäßig aktiviert. Wenn Sie Pods auf einem Cluster der Version `1.22` oder einer früheren Version ausführen, müssen Sie den Amazon EBS CSI-Treiber installieren, bevor Sie Ihren Cluster auf Version aktualisieren, um Dienstunterbrechungen `1.23` zu vermeiden.

Lesen Sie die häufig gestellten Fragen zur Amazon EBS CSI-Migration.

Weitere Ressourcen

ClowdHaus Anleitung zum EKS-Upgrade

ClowdHaus EKS Upgrade Guidance ist eine CLI, die Sie beim Upgrade von Amazon EKS-Clustern unterstützt. Es kann einen Cluster auf mögliche Probleme hin analysieren, die vor dem Upgrade behoben werden müssen.

GoNoGo

GoNoGoist ein Tool in der Alpha-Phase, mit dem Sie die Zuverlässigkeit Ihrer Cluster-Add-Ons beim Upgrade ermitteln können.

Bewährte Methoden zur Kostenoptimierung

Kostenoptimierung bedeutet, Ihre Geschäftsergebnisse zum niedrigsten Preis zu erzielen. Wenn Sie der Dokumentation in diesem Handbuch folgen, optimieren Sie Ihre Amazon EKS-Workloads.

Allgemeine Richtlinien

In der Cloud gibt es eine Reihe allgemeiner Richtlinien, die Ihnen helfen können, die Kosten Ihrer Microservices zu optimieren:

- Stellen Sie sicher, dass Workloads, die auf Amazon EKS ausgeführt werden, unabhängig von bestimmten Infrastrukturtypen für den Betrieb Ihrer Container sind. Dadurch erhalten Sie mehr Flexibilität bei der Ausführung auf den kostengünstigsten Infrastrukturtypen. Bei der Verwendung von Amazon EKS mit EC2 kann es aufgrund der Art der Arbeitslast zu Ausnahmen kommen, wenn wir Workloads haben, die bestimmte Arten von EC2 Instance-Typen erfordern, wie z. B. die Anforderung einer GPU oder andere Instance-Typen.
- Wählen Sie Container-Instances mit optimalem Profil aus — profilieren Sie Ihre Produktions- oder Vorproduktionsumgebungen und überwachen Sie wichtige Kennzahlen wie CPU und Speicher mithilfe von Services wie Amazon CloudWatch Container Insights for Amazon EKS oder Tools von Drittanbietern, die im Kubernetes-Ökosystem verfügbar sind. Dadurch wird sichergestellt, dass wir die richtige Menge an Ressourcen zuweisen und eine Verschwendung von Ressourcen vermeiden können.
- Nutzen Sie die verschiedenen Kaufoptionen, die in AWS für die Ausführung von EKS verfügbar sind EC2, z. B. mit On-Demand-, Spot- und Savings Plan.

Bewährte Methoden zur Kostenoptimierung von EKS

Es gibt drei allgemeine Best-Practice-Bereiche für die Kostenoptimierung in der Cloud:

- Kostengünstige Ressourcen (Auto Scaling, Down Scaling, Richtlinien und Kaufoptionen)
- Sensibilisierung für Ausgaben (mithilfe von Tools von AWS und Drittanbietern)
- Optimierung im Laufe der Zeit (richtige Dimensionierung)

Wie bei jeder Anleitung gibt es Kompromisse. Stellen Sie sicher, dass Sie mit Ihrer Organisation zusammenarbeiten, um zu verstehen, welche Prioritäten für diese Arbeitslast gesetzt werden und welche bewährten Verfahren am wichtigsten sind.

Verwendung dieses Leitfadens

Dieser Leitfaden richtet sich an DevOps-Teams, die für die Implementierung und Verwaltung der EKS-Cluster und der von ihnen unterstützten Workloads verantwortlich sind. Der Leitfaden ist zur leichteren Nutzung in verschiedene Bereiche mit bewährten Verfahren unterteilt. Jedes Thema enthält eine Liste mit Empfehlungen, zu verwendenden Tools und bewährten Methoden zur Kostenoptimierung Ihrer EKS-Cluster. Die Themen müssen nicht in einer bestimmten Reihenfolge gelesen werden.

Die wichtigsten Funktionen von AWS-Services und Kubernetes

Die Kostenoptimierung wird durch die folgenden AWS-Services und -Funktionen unterstützt:

- EC2 Instance-Typen, Savings Plan (und Reserved Instances) und Spot-Instances zu unterschiedlichen Preisen.
- Auto Scaling zusammen mit den nativen Auto Scaling Scaling-Richtlinien von Kubernetes. Ziehen Sie einen Savings Plan (zuvor Reserved Instances) für vorhersehbare Workloads in Betracht. Verwenden Sie verwaltete Datenspeicher wie EBS und EFS, um die Elastizität und Haltbarkeit der Anwendungsdaten zu gewährleisten.
- Das Dashboard der Billing and Cost Management-Konsole bietet zusammen mit dem AWS Cost Explorer einen Überblick über Ihre AWS-Nutzung. Verwenden Sie AWS Organizations für detaillierte Abrechnungsdetails. Details zu mehreren Tools von Drittanbietern wurden ebenfalls weitergegeben.
- Amazon CloudWatch Container Metrics bietet Metriken zur Nutzung von Ressourcen durch den EKS-Cluster. Neben dem Kubernetes-Dashboard gibt es im Kubernetes-Ökosystem mehrere Tools, mit denen Verschwendung reduziert werden kann.

Dieses Handbuch enthält eine Reihe von Empfehlungen, mit denen Sie die Kostenoptimierung Ihres Amazon EKS-Clusters verbessern können.

Feedback

Dieser Leitfaden wird am GitHub veröffentlicht, um direktes Feedback und Vorschläge von der breiteren EKS/Kubernetes-Community zu sammeln. Wenn Sie eine bewährte Methode haben, die wir Ihrer Meinung nach in den Leitfaden aufnehmen sollten, reichen Sie bitte ein Problem ein oder reichen Sie eine PR im Repository ein. GitHub Wir beabsichtigen, den Leitfaden regelmäßig zu aktualisieren, sobald der Service um neue Funktionen erweitert wird oder wenn sich eine neue bewährte Methode herausstellt.

Framework zur Kostenoptimierung

AWS Cloud Economics ist eine Disziplin, die Kunden dabei unterstützt, ihre Effizienz zu steigern und ihre Kosten durch den Einsatz moderner Computertechnologien wie Amazon EKS zu senken. Die Disziplin empfiehlt, einer Methode zu folgen, die als „Cloud Financial Management (CFM) - Framework“ bezeichnet wird und aus 4 Säulen besteht:



Die Säule See: Messung und Rechenschaftspflicht

Bei der Säule See handelt es sich um eine Reihe grundlegender Aktivitäten und Technologien, die festlegen, wie Cloud-Ausgaben gemessen, überwacht und zur Rechenschaft gezogen werden können. Sie wird oft als „Beobachtbarkeit“, „Instrumentierung“ oder „Telemetrie“ bezeichnet. Die Möglichkeiten und Grenzen der „Observability“ -Infrastruktur bestimmen, was optimiert werden kann. Ein klares Bild Ihrer Kosten zu erhalten, ist ein entscheidender erster Schritt zur Kostenoptimierung,

da Sie wissen müssen, wo Sie beginnen. Diese Art von Transparenz dient auch als Leitfaden für die Arten von Aktivitäten, die Sie zur weiteren Optimierung Ihrer Umgebung durchführen müssen.

Im Folgenden finden Sie einen kurzen Überblick über unsere bewährten Verfahren für den Bereich See:

- Definieren und pflegen Sie eine Tagging-Strategie für Ihre Workloads.
  - Verwenden Sie Instance Tagging. Durch das Taggen von EKS-Clustern können Sie die einzelnen Clusterkosten einsehen und sie in Ihren Kosten- und Nutzungsberichten zuordnen.
- Richten Sie mithilfe von Technologien wie Kubecost eine Berichterstattung und Überwachung der EKS-Nutzung ein.
  - Aktivieren Sie Cloud Intelligence-Dashboards. Indem Sie Ressourcen ordnungsgemäß kennzeichnen und Visualisierungen verwenden, können Sie die Kosten messen und abschätzen.
- Ordnen Sie die Cloud-Kosten Anwendungen, Geschäftsbereichen (LoBs) und Einnahmequellen zu.
- Definieren, messen und kommunizieren Sie efficiency/value KPIs mit Stakeholdern im Unternehmen. Erstellen Sie beispielsweise einen KPI für „Maßeinheiten“, mit dem die Kosten pro Transaktion gemessen werden, z. B. könnte ein Mitfahrservice einen KPI für „Kosten pro Fahrt“ haben.

Weitere Informationen zu den empfohlenen Technologien und Aktivitäten im Zusammenhang mit dieser Säule finden Sie im Abschnitt Kostenoptimierung — Beobachtbarkeit dieses Leitfadens.

Die Säule „Sparen“: Kostenoptimierung

Diese Säule basiert auf den Technologien und Fähigkeiten, die im Rahmen der Säule „See“ entwickelt wurden. Diese Säule umfasst in der Regel die folgenden Aktivitäten:

- Identifizieren und eliminieren Sie Abfall in Ihrer Umgebung.
- Architektur und Design im Hinblick auf Kosteneffizienz.
- Wählen Sie die beste Kaufoption, z. B. On-Demand-Instances oder Spot-Instances.
- Passen Sie sich an die Weiterentwicklung der Services an: Mit der Weiterentwicklung der AWS-Services kann sich auch die Art und Weise ändern, wie diese Services effizient genutzt werden können. Seien Sie bereit, sich anzupassen, um diesen Änderungen Rechnung zu tragen.

Da diese Aktivitäten betriebsbereit sind, hängen sie stark von den Eigenschaften Ihrer Umgebung ab. Fragen Sie sich, was die wichtigsten Kostentreiber sind? Welchen geschäftlichen Nutzen bieten

Ihre verschiedenen Umgebungen? Welche Kaufoptionen und Infrastrukturoptionen, z. B. Instance-Familientypen, eignen sich am besten für die jeweilige Umgebung?

Im Folgenden finden Sie eine priorisierte Liste der häufigsten Kostentreiber für EKS-Cluster:

1. Rechenkosten: Die Kombination mehrerer Typen von Instance-Familien, Kaufoptionen und ein ausgewogenes Verhältnis zwischen Skalierbarkeit und Verfügbarkeit erfordern sorgfältige Überlegungen. Weitere Informationen finden Sie in den Empfehlungen im Abschnitt Kostenoptimierung — Datenverarbeitung in diesem Leitfaden.
2. Netzwerkkosten: Die Verwendung von 3 AZs für EKS-Cluster kann die Kosten für den Inter-AZ-Verkehr potenziell erhöhen. Unsere Empfehlungen für ein ausgewogenes Verhältnis zwischen HA-Anforderungen und niedrigen Kosten für den Netzwerkverkehr finden Sie im Abschnitt Kostenoptimierung — Netzwerke in diesem Leitfaden.
3. Speicherkosten: Abhängig von der stateful/stateless Art der Workloads in den EKS-Clustern und davon, wie die verschiedenen Speichertypen verwendet werden, kann Speicher als Teil der Arbeitslast betrachtet werden. Informationen zu den EKS-Speicherkosten finden Sie im Abschnitt Kostenoptimierung — Speicher in diesem Leitfaden.

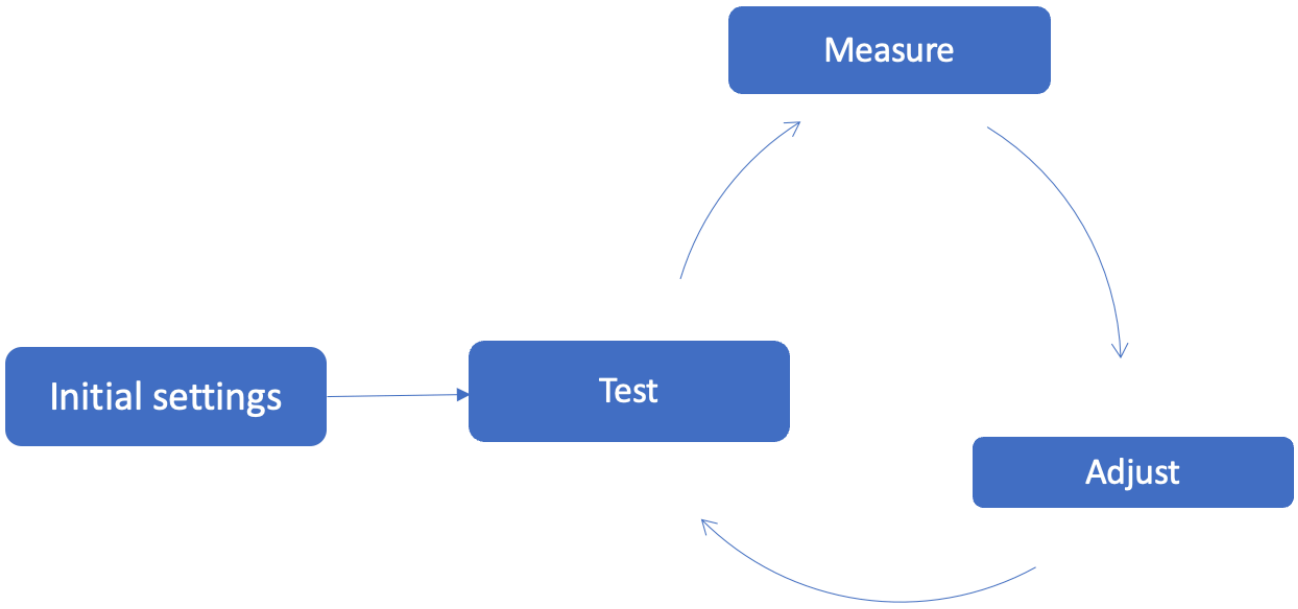
Die Säule des Plans: Planung und Prognose

Sobald die Empfehlungen in der Säule See umgesetzt sind, werden die Cluster fortlaufend optimiert. Da Erfahrungen mit dem effizienten Betrieb von Clustern gesammelt wurden, können sich die Planungs- und Prognoseaktivitäten auf folgende Bereiche konzentrieren:

- Dynamische Budgetierung und Prognose der Cloud-Kosten.
- Quantifizierung des Geschäftswerts, den die EKS-Containerdienste bieten.
- Integration des EKS-Cluster-Kostenmanagements in die Planung des IT-Finanzmanagements.

Die Run-Säule

Die Kostenoptimierung ist ein kontinuierlicher Prozess, der eine Vielzahl inkrementeller Verbesserungen beinhaltet:



Für die Integration der EKS-Cluster-Optimierung in die „FinOps" Bemühungen der Organisation ist es von entscheidender Bedeutung, sich die Unterstützung der Führungskräfte für diese Art von Aktivitäten zu sichern. Sie ermöglicht die Abstimmung der Interessengruppen durch ein gemeinsames Verständnis der EKS-Cluster-Kosten, die Implementierung von EKS-Cluster-Kostenplanken und die Sicherstellung, dass sich die Tools, die Automatisierung und die Aktivitäten an die Bedürfnisse des Unternehmens anpassen.

Referenzen

- AWS-Cloud-Wirtschaft, Cloud-Finanzmanagement

Bewusstsein für Ausgaben

Ausgabenbewusstsein bedeutet, zu verstehen, wer, wo und was die Ausgaben in Ihrem EKS-Cluster verursacht. Wenn Sie sich ein genaues Bild von diesen Daten machen, können Sie das Bewusstsein für Ihre Ausgaben schärfen und Bereiche aufzeigen, in denen Abhilfemaßnahmen erforderlich sind.

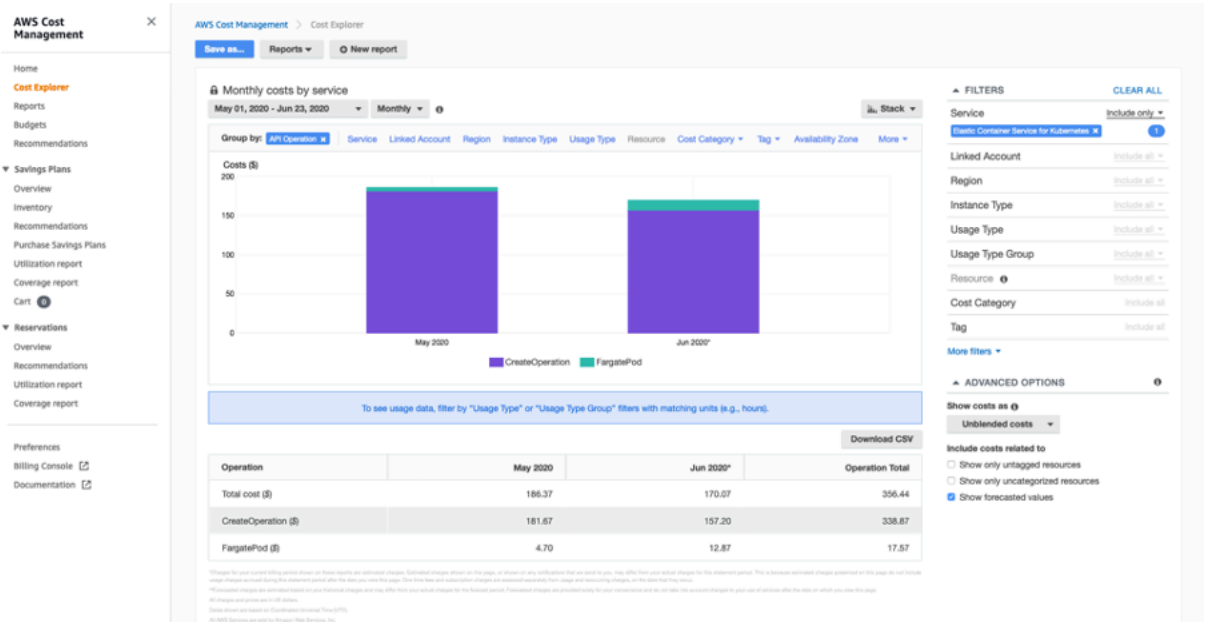
Empfehlungen

Verwenden Sie den Cost Explorer

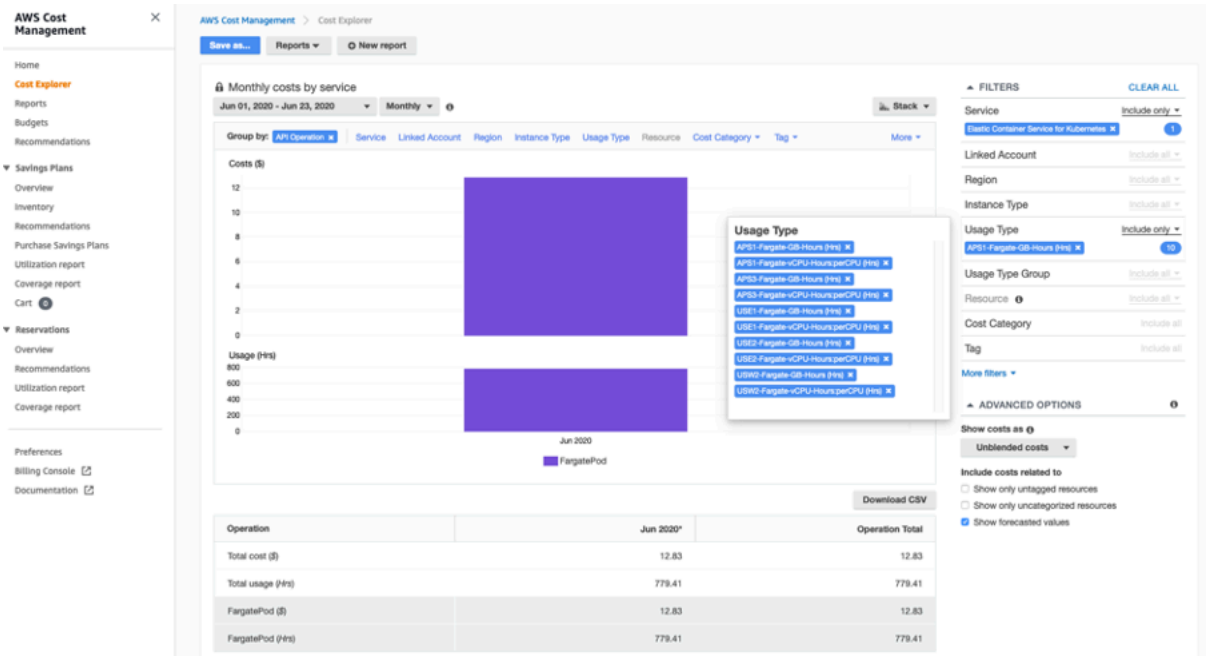
AWS Cost Explorer verfügt über eine easy-to-use Oberfläche, mit der Sie Ihre AWS-Kosten und -Nutzung im Laufe der Zeit visualisieren, verstehen und verwalten können. Mithilfe der im Cost Explorer verfügbaren Filter können Sie Kosten- und Nutzungsdaten auf verschiedenen Ebenen analysieren.

Kosten für EKS Control Plane und EKS Fargate

Mithilfe der Filter können wir die Kosten abfragen, die für die EKS-Kosten auf der Kontrollebene und dem Fargate-Pod entstanden sind, wie in der folgenden Abbildung dargestellt:



Mithilfe der Filter können wir die Gesamtkosten abfragen, die für die Fargate-Pods in den verschiedenen Regionen in EKS anfallen. Dazu gehören vCPU vCPU-Stunden pro CPU als auch GB-Stunden, wie in der folgenden Abbildung dargestellt:



Verschlagwortung von Ressourcen

Amazon EKS unterstützt das Hinzufügen von AWS-Tags zu Ihren Amazon EKS-Clustern. Dies macht es einfach, den Zugriff auf die EKS-API für die Verwaltung Ihrer Cluster zu kontrollieren. Zu einem EKS-Cluster hinzugefügte Tags sind spezifisch für die AWS EKS-Clusterressource. Sie werden nicht auf andere vom Cluster verwendete AWS-Ressourcen wie EC2 Instances oder Load Balancer übertragen. Heute wird Cluster-Tagging für alle neuen und vorhandenen EKS-Cluster über die AWS-API, Konsole und SDKs unterstützt.

AWS Fargate ist eine Technologie, die bedarfsgerecht dimensionierte Rechenkapazität für Container bereitstellt. Bevor Sie Pods auf Fargate in Ihrem Cluster planen können, müssen Sie mindestens ein Fargate-Profil definieren, das festlegt, welche Pods Fargate verwenden sollen, wenn sie gestartet werden.

Hinzufügen und Auflisten von Tags zu einem EKS-Cluster:

```
$ 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"
    }
}
```

Nachdem Sie die Kostenzuordnungs-Tags im AWS Cost Explorer aktiviert haben, verwendet AWS die Kostenzuordnungs-Tags, um Ihre Ressourcenkosten in Ihrem Kostenzuordnungsbericht zu organisieren, sodass Sie Ihre AWS-Kosten einfacher kategorisieren und verfolgen können.

Tags haben keine semantische Bedeutung für Amazon EKS und werden ausschließlich als Zeichenfolgen interpretiert. Sie können beispielsweise eine Reihe von Tags für Ihre Amazon-EKS-Cluster definieren, um Ihnen dabei zu helfen, den Eigentümer und die Stack-Ebene jedes einzelnen Clusters nachzuverfolgen.

Verwenden Sie AWS Trusted Advisor

AWS Trusted Advisor bietet eine Vielzahl von Best-Practice-Prüfungen und Empfehlungen in fünf Kategorien: Kostenoptimierung, Sicherheit, Fehlertoleranz, Leistung und Servicebeschränkungen.

Zur Kostenoptimierung hilft Trusted Advisor dabei, ungenutzte und ungenutzte Ressourcen zu eliminieren, und empfiehlt, Verpflichtungen für reservierte Kapazitäten einzugehen. Die wichtigsten Maßnahmen, die Amazon EKS helfen werden, betreffen unter anderem wenig genutzte EC2 Instances, nicht zugeordnete Elastic IP-Adressen, Idle Load Balancer und nicht ausgelastete EBS-Volumes. Die vollständige Liste der Prüfungen finden Sie unter -practice-checklist/. https:// aws.amazon.com/premiumsupport/ technology/trusted-advisor/best

Der Trusted Advisor bietet auch Empfehlungen zu Savings Plans und Reserved EC2 Instances für Instances und Fargate, sodass Sie sich gegen vergünstigte Tarife auf einen gleichbleibenden Nutzungsbetrag festlegen können.

> **Note**
>
> Die Empfehlungen von Trusted Advisor sind allgemeine Empfehlungen und nicht spezifisch für EKS.

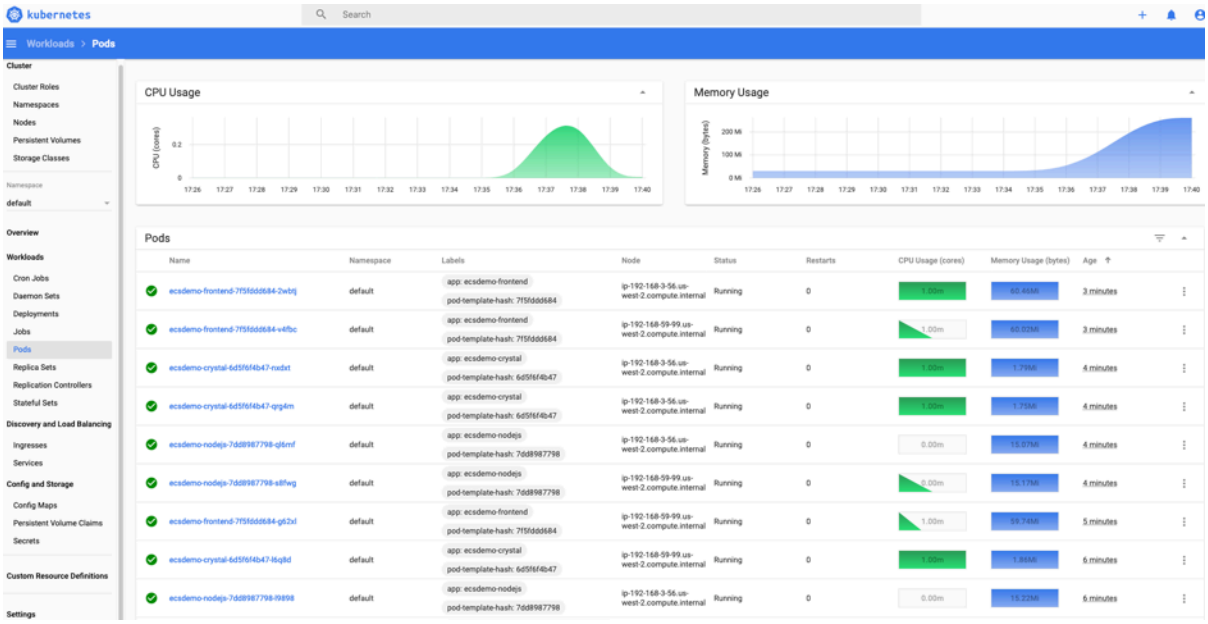
Verwenden Sie das Kubernetes-Dashboard

Kubernetes-Dashboard

Das Kubernetes-Dashboard ist eine allgemeine, webbasierte Benutzeroberfläche für Kubernetes-Cluster, die Informationen über den Kubernetes-Cluster einschließlich der Ressourcennutzung auf

Cluster-, Knoten- und Pod-Ebene bereitstellt. Die Bereitstellung des Kubernetes-Dashboards auf einem Amazon EKS-Cluster wird in der Amazon EKS-Dokumentation beschrieben.

Das Dashboard bietet Aufschlüsselungen zur Ressourcennutzung für jeden Knoten und Pod sowie detaillierte Metadaten zu Pods, Diensten, Bereitstellungen und anderen Kubernetes-Objekten. Diese konsolidierten Informationen bieten Einblick in Ihre Kubernetes-Umgebung.



kubectl Befehle top und describe

Metriken zur Ressourcennutzung mit den Befehlen kubectl top und kubectl describe anzeigen. kubectl top zeigt die aktuelle CPU- und Speicherauslastung für die Pods oder Knoten in Ihrem Cluster oder für einen bestimmten Pod oder Knoten an. Der Befehl kubectl describe liefert detailliertere Informationen zu einem bestimmten Knoten oder Pod.

```
$ kubectl top pods
$ kubectl top nodes
$ kubectl top pod pod-name --namespace mynamespace --containers
```

Mit dem Befehl top zeigt die Ausgabe die Gesamtmenge an CPU (in Kernen) und Speicher (in MiB) an, die der Knoten verwendet, sowie die Prozentsätze der zuweisbaren Kapazität des Knotens, die diese Zahlen darstellen. Sie können dann einen Drilldown zur nächsten Ebene, der Containerebene innerhalb von Pods, durchführen, indem Sie das Flag --containers hinzufügen.

```
$ kubectl describe node <node>
$ kubectl describe pod <pod>
```

kubectl describe gibt den Prozentsatz der verfügbaren Gesamtkapazität zurück, den jede Ressourcenanforderung oder jedes Limit darstellt.

kubectl top and describe verfolgt die Nutzung und Verfügbarkeit kritischer Ressourcen wie CPU, Arbeitsspeicher und Speicher in Kubernetes-Pods, -Knoten und Containern. Dieses Bewusstsein hilft dabei, die Ressourcennutzung zu verstehen und die Kosten zu kontrollieren.

Verwenden Sie CloudWatch Container Insights

Verwenden Sie CloudWatch Container Insights, um Metriken und Logs aus Ihren containerisierten Anwendungen und Microservices zu sammeln, zu aggregieren und zusammenzufassen. Container Insights ist für Amazon Elastic Kubernetes Service auf EC2 und Kubernetes-Plattformen auf Amazon verfügbar. EC2 Die Metriken umfassen die Auslastung für Ressourcen wie z B. CPU, Arbeitsspeicher, Datenträger und Netzwerk.

Die Installation von Insights ist in der Dokumentation beschrieben.

CloudWatch erstellt aggregierte Metriken auf Cluster-, Knoten-, Pod-, Aufgaben- und Serviceebene als CloudWatch Metriken.

Die folgende Abfrage zeigt eine Liste von Knoten, sortiert nach der durchschnittlichen CPU-Auslastung der Knoten

```
STATS avg(node_cpu_utilization) as avg_node_cpu_utilization by NodeName
| SORT avg_node_cpu_utilization DESC
```

CPU-Auslastung nach Container-Namen

```
stats pct(container_cpu_usage_total, 50) as CPUPercMedian by kubernetes.container_name
| filter Type="Container"
```

Festplattennutzung nach Container-Namen

```
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
```

Weitere Beispielabfragen finden Sie in der Dokumentation zu Container Insights

Dieses Bewusstsein wird dazu beitragen, den Ressourcenverbrauch zu verstehen und die Kosten zu kontrollieren.

Nutzung von Kubecost zur Sensibilisierung und Orientierung bei den Ausgaben

Tools von Drittanbietern wie kubecost können auch auf Amazon EKS bereitgestellt werden, um einen Überblick über die Kosten für den Betrieb Ihres Kubernetes-Clusters zu erhalten. Informationen zur Kostenverfolgung mit Kubecost finden Sie in diesem AWS-Blog

Bereitstellung von Kubecost mit 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

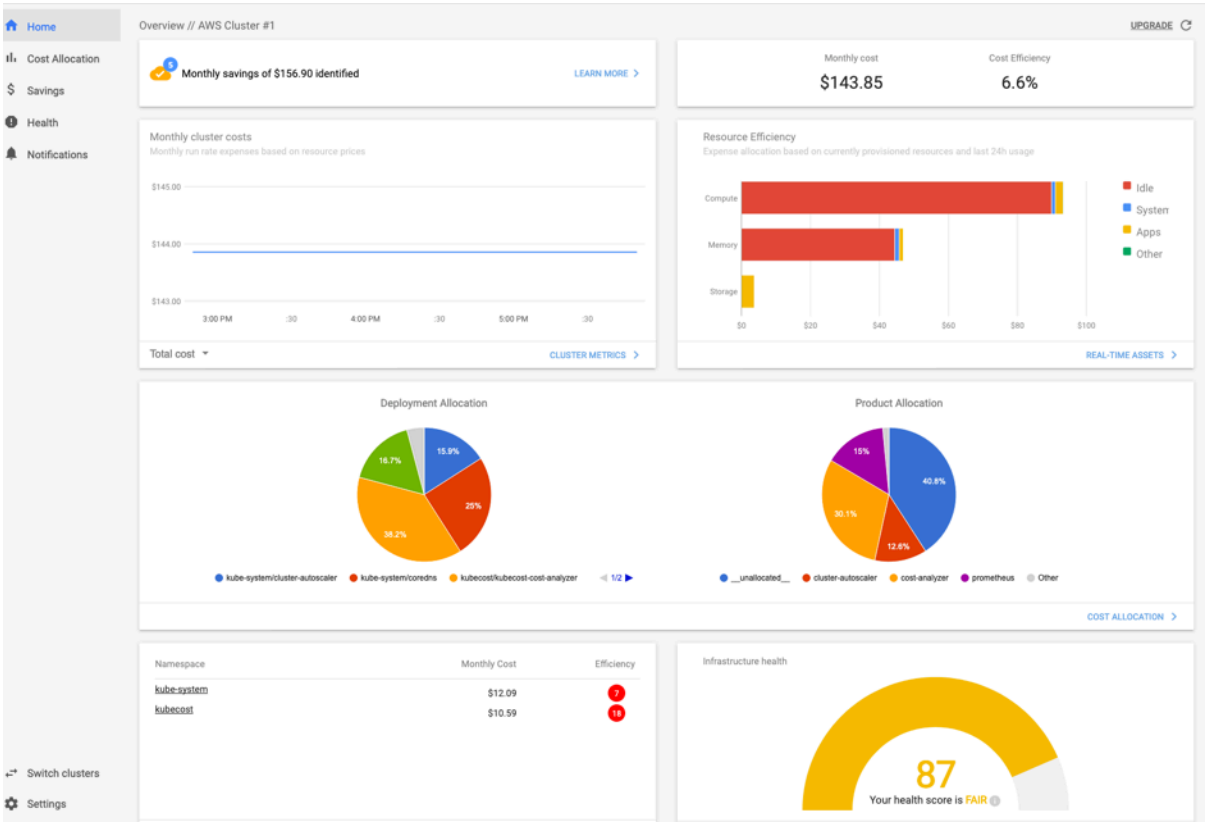
$ helm install kubecost kubecost/cost-analyzer --namespace kubecost --set
 kubecostToken="aGRoZEBqc2pzLmNvbQ==xm343yadf98"
NAME: kubecost
LAST DEPLOYED: Mon May 18 08:49:05 2020
NAMESPACE: kubecost
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
--------------------------------------------------Kubecost 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

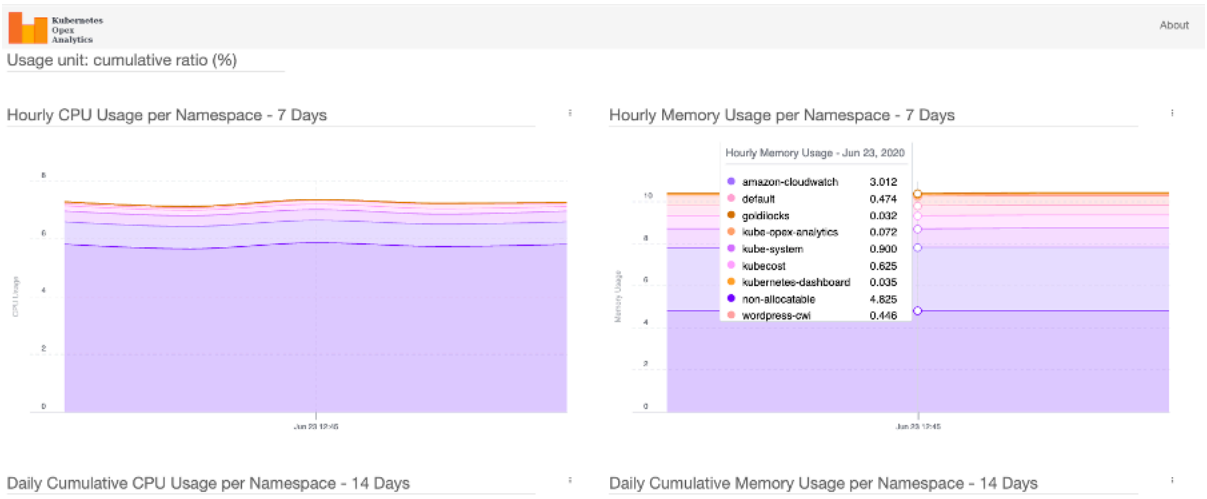
NOTE: If you are using Cloud 9 or have a need to forward it to a different port like
 8080, issue the following command
$ kubectl port-forward --namespace kubecost deployment/kubecost-cost-analyzer 8080:9090
```

Kubecost-Dashboard -



Verwenden Sie das Kubernetes-Analysetool für Kostenzuweisung und Kapazitätsplanung

Kubernetes Opex Analytics ist ein Tool, mit dem Unternehmen nachverfolgen können, welche Ressourcen von ihren Kubernetes-Clustern verbraucht werden, um zu hohe Zahlungen zu vermeiden. Zu diesem Zweck werden kurz- (7 Tage), mittel- (14 Tage) und langfristige (12 Monate) Nutzungsberichte generiert, die relevante Erkenntnisse darüber enthalten, wie viel Ressourcen jedes Projekt im Laufe der Zeit verbraucht.



Yotaskala

Yotascale hilft bei der genauen Zuordnung der Kubernetes-Kosten. Die Funktion Yotascale Kubernetes Cost Allocation verwendet tatsächliche Kostendaten, einschließlich Reserved Instance-Rabatte und Spot-Instance-Preise, anstelle von generischen Marktpreisschätzungen, um den gesamten Kostenaufwand für Kubernetes zu ermitteln

Weitere Informationen finden Sie auf ihrer Website.

Berater von Alcide

Alcide ist ein Advanced Technology Partner des AWS-Partnernetzwerks (APN). Alcide Advisor hilft sicherzustellen, dass Ihre Amazon EKS-Cluster-, Knoten- und Pods-Konfiguration so abgestimmt sind, dass sie den bewährten Sicherheitsmethoden und internen Richtlinien entsprechen. Alcide Advisor ist ein agentenloser Service für die Prüfung und Einhaltung von Vorschriften bei Kubernetes, der darauf ausgelegt ist, einen reibungslosen und sicheren DevSecOps Ablauf zu gewährleisten, indem er die Entwicklungsphase vor der Umstellung auf die Produktion verhärtet.

Weitere Einzelheiten finden Sie in diesem Blogbeitrag.

Andere Tools

Kubernetes-Garbage-Collection

Die Rolle des Kubernetes-Garbage-Collectors besteht darin, bestimmte Objekte zu löschen, die einmal einen Besitzer hatten, aber keinen Besitzer mehr haben.

Grafschaft Fargate

Fargatecount ist ein nützliches Tool, mit dem AWS-Kunden anhand einer benutzerdefinierten CloudWatch Metrik die Gesamtzahl der EKS-Pods verfolgen können, die auf Fargate in einer bestimmten Region eines bestimmten Kontos bereitgestellt wurden. Dies hilft dabei, den Überblick über alle Fargate-Pods zu behalten, die in einem EKS-Cluster ausgeführt werden.

Popeye — Ein Kubernetes-Cluster-Desinfizierer

Popeye — Ein Kubernetes Cluster Sanitizer ist ein Hilfsprogramm, das Live-Kubernetes-Cluster scannt und potenzielle Probleme mit bereitgestellten Ressourcen und Konfigurationen meldet. Es bereinigt Ihren Cluster auf der Grundlage dessen, was bereitgestellt wird und nicht darauf, was sich auf der Festplatte befindet. Durch das Scannen Ihres Clusters werden Fehlkonfigurationen erkannt und Sie können sicherstellen, dass bewährte Verfahren angewendet werden

Ressourcen

In den folgenden Ressourcen erfahren Sie mehr über bewährte Methoden zur Kostenoptimierung.

Dokumentation und Blogs

- Amazon EKS unterstützt Tagging

Tools

- Was ist AWS Billing and Cost Management?
- Einblicke in Amazon CloudWatch Container
- So verfolgen Sie die Kosten in Amazon EKS-Clustern mit mehreren Mandanten mithilfe von Kubecost
- Kubecost
- Kube Opsview
- Kubernetes Opex-Analytik

Berechnung und Autoscaling

Als Entwickler erstellen Sie Schätzungen zu den Ressourcenanforderungen Ihrer Anwendung, z. B. CPU und Arbeitsspeicher. Wenn Sie sie jedoch nicht ständig anpassen, können sie veraltet sein, was Ihre Kosten erhöhen und Leistung und Zuverlässigkeit verschlechtern könnte. Es ist wichtiger, die

Ressourcenanforderungen einer Anwendung kontinuierlich anzupassen, als sie gleich beim ersten Mal richtig zu machen.

Die unten genannten Best Practices helfen Ihnen beim Aufbau und Betrieb kostenbewusster Workloads, mit denen Sie Geschäftsergebnisse erzielen und gleichzeitig die Kosten minimieren und Ihrem Unternehmen eine maximale Investitionsrendite ermöglichen. Für die Optimierung Ihrer Cluster-Rechenkosten sind folgende Punkte von großer Bedeutung:

1. Workloads in der richtigen Größe
2. Reduzieren Sie ungenutzte Kapazitäten
3. Optimieren Sie Rechenkapazitätstypen (z. B. Spot) und Beschleuniger (z. B. GPUs)

Passen Sie die Größe Ihrer Workloads an

In den meisten EKS-Clustern entfällt der Großteil der Kosten auf die EC2 Instances, die zur Ausführung Ihrer containerisierten Workloads verwendet werden. Sie werden nicht in der Lage sein, Ihre Rechenressourcen richtig zu dimensionieren, ohne Ihre Workload-Anforderungen zu verstehen. Aus diesem Grund ist es wichtig, dass Sie die entsprechenden Anfragen und Grenzwerte verwenden und diese Einstellungen bei Bedarf anpassen. Darüber hinaus können Abhängigkeiten wie Instanzgröße und Speicherauswahl die Workload-Leistung beeinträchtigen, was eine Vielzahl unbeabsichtigter Folgen auf Kosten und Zuverlässigkeit haben kann.

Anfragen sollten sich an der tatsächlichen Nutzung orientieren. Wenn die Anfragen eines Containers zu hoch sind, bleibt ungenutzte Kapazität übrig, was einen großen Faktor bei den Gesamtkosten des Clusters ausmacht. Für jeden Container in einem Pod, z. B. für Anwendung und Sidecars, sollten eigene Anfragen und Grenzwerte festgelegt werden, um sicherzustellen, dass die aggregierten Pod-Grenzwerte so genau wie möglich sind.

Verwenden Sie Tools wie Goldilocks, KRR und Kubecost, mit denen Sie die Ressourcenanforderungen und -limits für Ihre Container abschätzen können. Je nach Art der Anwendungen, performance/cost Anforderungen und Komplexität müssen Sie abwägen, auf welche Metriken Sie am besten skalieren können, ab welchem Punkt sich Ihre Anwendungsleistung verschlechtert (Sättigungspunkt) und wie Sie Anforderungen und Limits entsprechend anpassen können. Weitere Hinweise zu diesem Thema finden Sie unter Richtige Dimensionierung von Anwendungen.

Wir empfehlen, den Horizontal Pod Autoscaler (HPA) zu verwenden, um zu steuern, wie viele Replikate Ihrer Anwendung ausgeführt werden sollen, den Vertical Pod Autoscaler (VPA), um

anzupassen, wie viele Anfragen und Beschränkungen Ihre Anwendung pro Replikat benötigt, und einen Node-Autoscaler wie Karpenter oder Cluster Autoscaler, um die Gesamtzahl der Knoten in Ihrem Cluster kontinuierlich anzupassen. Techniken zur Kostenoptimierung mit Karpenter und Cluster Autoscaler werden in einem späteren Abschnitt dieses Dokuments dokumentiert.

Der Vertical Pod Autoscaler kann die Anforderungen und Grenzwerte, die Containern zugewiesen sind, anpassen, sodass Workloads optimal ausgeführt werden. Sie sollten den VPA im Auditing-Modus ausführen, damit er nicht automatisch Änderungen vornimmt, und Ihre Pods neu starten. Es schlägt Änderungen vor, die auf beobachteten Messwerten basieren. Bei allen Änderungen, die sich auf Produktionsworkloads auswirken, sollten Sie diese Änderungen zunächst in einer Umgebung außerhalb der Produktionsumgebung überprüfen und testen, da sie sich auf die Zuverlässigkeit und Leistung Ihrer Anwendung auswirken können.

Reduzieren Sie den Verbrauch

Der beste Weg, Geld zu sparen, besteht darin, weniger Ressourcen bereitzustellen. Eine Möglichkeit, dies zu tun, besteht darin, die Workloads an ihre aktuellen Anforderungen anzupassen. Sie sollten alle Bemühungen zur Kostenoptimierung damit beginnen, sicherzustellen, dass Ihre Workloads ihre Anforderungen definieren und dynamisch skalieren. Dazu müssen Sie Metriken aus Ihren Anwendungen abrufen und Konfigurationen wie `PodDisruptionBudgets`Pod Readiness Gates einrichten, um sicherzustellen, dass Ihre Anwendung sicher dynamisch hoch- und herunterskaliert werden kann. Es ist wichtig zu berücksichtigen, dass eine Einschränkung verhindern PodDisruptionBudgets kann, dass Cluster Autoscaler und Karpenter Knoten herunterskalieren, da dies sowohl Cluster Autoscaler als auch Karpenter respektieren. PodDisruptionBudgets Der Wert 'minAvailable' in der PodDisruptionBudget sollte immer niedriger sein als die Anzahl der Pods in der Bereitstellung und Sie sollten einen guten Puffer zwischen den beiden haben, z. B. bei einer Bereitstellung von 6 Pods, bei der mindestens 4 Pods gleichzeitig laufen sollen, setzen Sie den Wert 'minAvailable' in Ihrem auf 4. PodDisruptionBidget Auf diese Weise können Cluster Autoscaler und Karpenter während eines Node-Scale-Down-Ereignisses Pods sicher aus den nicht ausgelasteten Knoten entleeren und entfernen. Weitere Informationen finden Sie im Dokument mit den häufig gestellten Fragen zu Cluster Autoscaler.

Der Horizontal Pod Autoscaler ist ein flexibler Workload-Autoscaler, mit dem Sie anpassen können, wie viele Replikate benötigt werden, um die Leistungs- und Zuverlässigkeitsanforderungen Ihrer Anwendung zu erfüllen. Er verfügt über ein flexibles Modell, mit dem anhand verschiedener Messwerte wie CPU, Arbeitsspeicher oder benutzerdefinierter Messwerte wie Warteschlangentiefe, Anzahl der Verbindungen zu einem Pod usw. festgelegt werden kann, wann hoch- und herunterskaliert werden soll.

Der Kubernetes Metrics Server ermöglicht die Skalierung als Reaktion auf integrierte Messwerte wie CPU- und Speicherauslastung. Wenn Sie jedoch auf der Grundlage anderer Metriken wie Amazon CloudWatch oder SQS-Warteschlangentiefe skalieren möchten, sollten Sie ereignisgesteuerte Autoscaling-Projekte wie KEDA in Betracht ziehen. In diesem Blogbeitrag erfahren Sie, wie Sie KEDA mit Metriken verwenden können. CloudWatch Wenn Sie sich nicht sicher sind, welche Kennzahlen Sie überwachen und auf deren Grundlage skalieren sollen, schauen Sie sich die Best Practices zur Überwachung wichtiger Kennzahlen an.

Die Reduzierung des Workload-Verbrauchs führt zu überschüssiger Kapazität in einem Cluster. Mit der richtigen Autoscaling-Konfiguration können Sie Knoten automatisch herunterskalieren und so Ihre Gesamtausgaben reduzieren. Wir empfehlen Ihnen, nicht zu versuchen, die Rechenkapazität manuell zu optimieren. Der Kubernetes-Scheduler und die Node-Autoscaler wurden so konzipiert, dass sie diesen Prozess für Sie übernehmen.

Reduzieren Sie ungenutzte Kapazitäten

Nachdem Sie die richtige Größe für Anwendungen ermittelt und so die Anzahl der Anfragen reduziert haben, können Sie damit beginnen, die bereitgestellte Rechenkapazität zu reduzieren. Sie sollten in der Lage sein, dies dynamisch zu tun, wenn Sie sich die Zeit genommen haben, Ihre Workloads anhand der obigen Abschnitte richtig zu dimensionieren. Es gibt zwei primäre Node-Autoscaler, die mit Kubernetes in AWS verwendet werden.

Karpenter und Cluster Autoscaler

Sowohl Karpenter als auch der Kubernetes Cluster Autoscaler skalieren die Anzahl der Knoten in Ihrem Cluster, wenn Pods erstellt oder entfernt werden und sich die Rechenanforderungen ändern. Das Hauptziel ist bei beiden dasselbe, aber Karpenter verfolgt einen anderen Ansatz für die Bereitstellung und Deprovisionierung von Knoten, was zur Kostensenkung und Optimierung der clusterweiten Nutzung beitragen kann.

Je größer die Cluster werden und die Vielfalt der Workloads zunimmt, desto schwieriger wird es, Knotengruppen und Instanzen vorzukonfigurieren. Genau wie bei Workload-Anfragen ist es wichtig, eine anfängliche Ausgangsbasis festzulegen und diese bei Bedarf kontinuierlich anzupassen.

Wenn Sie Cluster Autoscaler verwenden, berücksichtigt es die „Minimal“ - und „Maximum“ -Werte jeder Auto Scaling Scaling-Gruppe (ASG) und passt nur den „gewünschten“ Wert an. Es ist wichtig, dass Sie bei der Festlegung dieser Werte für die zugrunde liegende ASG darauf achten, da Cluster Autoscaler eine ASG nicht über ihre „minimale“ Anzahl hinaus herunterskalieren kann. Stellen Sie die „gewünschte“ Anzahl als die Anzahl der Knoten ein, die Sie während der normalen Geschäftszeiten

benötigen, und „Minimum“ als die Anzahl der Knoten, die Sie außerhalb der Geschäftszeiten benötigen. Weitere Informationen finden Sie im Dokument mit den häufig gestellten Fragen zu Cluster Autoscaler.

Cluster Autoscaler Priority Expander

Der Kubernetes Cluster Autoscaler funktioniert, indem er Gruppen von Knoten — eine so genannte Knotengruppe — nach oben und unten skaliert, während Anwendungen nach oben und unten skaliert werden. Wenn Sie Workloads nicht dynamisch skalieren, hilft Ihnen der Cluster Autoscaler nicht dabei, Geld zu sparen. Für den Cluster-Autoscaler muss ein Clusteradministrator im Voraus Knotengruppen erstellen, damit die Workloads genutzt werden können. Die Knotengruppen müssen so konfiguriert werden, dass sie Instanzen verwenden, die dasselbe „Profil“ haben, d. h. ungefähr die gleiche Menge an CPU und Arbeitsspeicher.

Sie können mehrere Knotengruppen haben und der Cluster Autoscaler kann so konfiguriert werden, dass er Prioritätsskalierungsstufen festlegt, und jede Knotengruppe kann Knoten unterschiedlicher Größe enthalten. Knotengruppen können unterschiedliche Kapazitätstypen haben und der Priority Expander kann verwendet werden, um kostengünstigere Gruppen zuerst zu skalieren.

Im Folgenden finden Sie ein Beispiel für einen Ausschnitt aus der Clusterkonfiguration, der a verwendet, um reservierte Kapazität `ConfigMap` zu priorisieren, bevor On-Demand-Instances verwendet werden. Sie können dieselbe Technik verwenden, um Graviton- oder Spot-Instances gegenüber anderen Typen zu priorisieren.

```
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.*
```

Die Verwendung von Knotengruppen kann dazu beitragen, dass die zugrunde liegenden Rechenressourcen standardmäßig das Erwartete tun, z. B. Knoten über mehrere Knoten verteilen AZs, aber nicht alle Workloads haben dieselben Anforderungen oder Erwartungen, und es ist besser, Anwendungen ihre Anforderungen explizit deklarieren zu lassen. Weitere Informationen zu Cluster Autoscaler finden Sie im Abschnitt mit bewährten Methoden.

Descheduler

Der Cluster Autoscaler kann einem Cluster Knotenkapazität hinzufügen oder daraus entfernen, wenn neue Pods geplant werden müssen oder Knoten nicht ausgelastet sind. Die Pod-Platzierung, nachdem sie für einen Knoten geplant wurde, wird nicht ganzheitlich betrachtet. Wenn Sie den Cluster Autoscaler verwenden, sollten Sie sich auch den Kubernetes-Descheduler ansehen, um Kapazitätsverschwendung in Ihrem Cluster zu vermeiden.

Wenn Sie 10 Knoten in einem Cluster haben und jeder Knoten zu 60% ausgelastet ist, nutzen Sie nicht 40% der bereitgestellten Kapazität im Cluster. Mit dem Cluster Autoscaler können Sie den Auslastungsschwellenwert pro Knoten auf 60% festlegen. Dabei wird jedoch nur versucht, einen einzelnen Knoten herunterzuskalieren, wenn die Auslastung unter 60% gesunken ist.

Mit dem Descheduler kann er sich die Clusterkapazität und -auslastung ansehen, nachdem Pods geplant oder Knoten zum Cluster hinzugefügt wurden. Es versucht, die Gesamtkapazität des Clusters über einem bestimmten Schwellenwert zu halten. Es kann auch Pods entfernen, die auf Knotenfehlern oder neuen Knoten basieren, die dem Cluster beitreten, um sicherzustellen, dass Pods in ihrer optimalen Rechenumgebung ausgeführt werden. Beachten Sie, dass der Descheduler den Austausch entfernter Pods nicht plant, sondern sich dafür auf den Standard-Scheduler stützt.

Konsolidierung von Karpenter

Karpenter verfolgt bei der Knotenverwaltung einen „gruppenlosen“ Ansatz. Dieser Ansatz ist flexibler für verschiedene Workload-Typen und erfordert weniger Vorabkonfiguration für Clusteradministratoren. Anstatt Gruppen vorab zu definieren und jede Gruppe nach den

Anforderungen der Workloads zu skalieren, verwendet Karpenter Provisioner und Knotenvorlagen, um allgemein zu definieren, welche Art von EC2 Instanzen erstellt werden können und welche Einstellungen für die Instanzen bei ihrer Erstellung vorgenommen werden.

Beim Bin-Packing werden mehr Ressourcen der Instanz genutzt, indem mehr Workloads auf weniger Instanzen mit optimaler Größe gepackt werden. Dies trägt zwar dazu bei, Ihre Rechenkosten zu senken, da nur Ressourcen bereitgestellt werden, die Ihre Workloads nutzen, aber es hat auch einen Kompromiss. Das Starten neuer Workloads kann länger dauern, da dem Cluster Kapazität hinzugefügt werden muss, insbesondere bei großen Skalierungsereignissen. Achten Sie bei der Einrichtung von Bin Packing auf das Gleichgewicht zwischen Kostenoptimierung, Leistung und Verfügbarkeit.

Karpenter kann die Instanzressourcen kontinuierlich überwachen und nach Bedarf zusammenführen, um die Auslastung der Instanzressourcen zu verbessern und Ihre Rechenkosten zu senken. Karpenter kann auch einen kostengünstigeren Worker-Node für Ihre Arbeitslast auswählen. Dies kann erreicht werden, indem das Flag „Konsolidierung“ im Provisioner auf true gesetzt wird (Beispielcodeausschnitt unten). Das folgende Beispiel zeigt einen Beispiel-Provisioner, der die Konsolidierung ermöglicht. Zum Zeitpunkt der Erstellung dieses Handbuchs wird Karpenter eine laufende Spot-Instance nicht durch eine günstigere Spot-Instance ersetzen. Weitere Informationen zur Konsolidierung von Karpenter finden Sie in diesem Blog.

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
  name: enable-binpacking
spec:
  consolidation:
    enabled: true
```

Bei Workloads, die möglicherweise nicht unterbrochen werden können, z. B. lange laufende Batch-Jobs ohne Checkpointing, sollten Sie in Erwägung ziehen, Pods mit der Anmerkung zu annotieren. `do-not-evict` Indem Sie Pods von der Räumung ausschließen, teilen Sie Karpenter mit, dass es nicht freiwillig Knoten entfernen soll, die diesen Pod enthalten. Wenn jedoch ein `do-not-evict` Pod zu einem Knoten hinzugefügt wird, während der Knoten leer ist, werden die verbleibenden Pods trotzdem entfernt, aber dieser Pod blockiert die Beendigung, bis er entfernt wird. In beiden Fällen wird der Knoten gesperrt, um zu verhindern, dass zusätzliche Arbeiten am Knoten geplant werden. Im Folgenden finden Sie ein Beispiel, das zeigt, wie die Anmerkung gesetzt wird:

```
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
```

Entfernen Sie nicht ausgelastete Knoten, indem Sie die Cluster Autoscaler-Parameter anpassen

Die Knotenauslastung ist definiert als die Summe der angeforderten Ressourcen geteilt durch die Kapazität. Standardmäßig `scale-down-utilization-threshold` ist der Wert auf 50% festgelegt. Dieser Parameter kann zusammen mit und verwendet werden`scale-down-unneeded-time`, wodurch festgelegt wird, wie lange ein Knoten nicht benötigt werden soll, bevor er herunterskaliert werden kann. Die Standardeinstellung ist 10 Minuten. Pods, die immer noch auf einem Knoten laufen, der herunterskaliert wurde, werden vom kube-scheduler auf anderen Knoten geplant. Das Anpassen dieser Einstellungen kann dabei helfen, nicht ausgelastete Knoten zu entfernen. Es ist jedoch wichtig, dass Sie diese Werte zuerst testen, damit Sie den Cluster nicht zu einer vorzeitigen Herunterskalierung zwingen.

Sie können ein Herunterskalieren verhindern, indem Sie sicherstellen, dass Pods, deren Entfernung teuer ist, durch ein Etikett geschützt sind, das vom Cluster Autoscaler erkannt wird. Stellen Sie zu diesem Zweck sicher, dass Pods, deren Entfernung teuer ist, über die Anmerkung verfügen. `cluster-autoscaler.kubernetes.io/safe-to-evict=false` Im Folgenden finden Sie ein Beispiel für Yaml zum Setzen der Anmerkung:

```
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
```

Markieren Sie Knoten mit Cluster Autoscaler und Karpenter

AWS-Ressourcen-Tags werden verwendet, um Ihre Ressourcen zu organisieren und Ihre AWS-Kosten detailliert zu verfolgen. Sie korrelieren nicht direkt mit Kubernetes-Labels zur Kostenverfolgung. Es wird empfohlen, mit der Kennzeichnung von Kubernetes-Ressourcen zu beginnen und Tools wie Kubecost zu verwenden, um Berichte über die Infrastrukturkosten auf der Grundlage von Kubernetes-Labels auf Pods, Namespaces usw. zu erstellen.

Worker-Knoten müssen über Tags verfügen, um Rechnungsinformationen im AWS Cost Explorer anzuzeigen. Taggen Sie mit Cluster Autoscaler Ihre Worker-Knoten innerhalb einer verwalteten Knotengruppe mithilfe einer Startvorlage. Taggen Sie Ihre Instances für selbstverwaltete Knotengruppen mithilfe der EC2 Auto Scaling-Gruppe. Von Karpenter bereitgestellte Instanzen kennzeichnen Sie sie mit spec.tags in der Node-Vorlage.

Cluster mit mehreren Mandanten

Wenn Sie an Clustern arbeiten, die von verschiedenen Teams gemeinsam genutzt werden, haben Sie möglicherweise keinen Einblick in andere Workloads, die auf demselben Knoten ausgeführt werden. Ressourcenanfragen können zwar dazu beitragen, einige Probleme mit „störenden Nachbarn“ zu isolieren, wie z. B. die gemeinsame Nutzung der CPU, aber sie isolieren möglicherweise nicht alle Ressourcengrenzen, wie z. B. die Erschöpfung der Festplatte I/O . Nicht jede Ressource, die von einem Workload genutzt werden kann, kann isoliert oder begrenzt werden. Workloads, die gemeinsam genutzte Ressourcen häufiger verbrauchen als andere Workloads, sollten durch Knotenfehler und Toleranzen isoliert werden. Eine weitere fortschrittliche Technik für solche Workloads ist das CPU-Pinning, das sicherstellt, dass für den Container ausschließlich CPU statt gemeinsam genutzter CPU verwendet wird.

Das Isolieren von Workloads auf Knotenebene kann teurer sein, aber es ist möglicherweise möglich, BestEffortJobs zu planen oder zusätzliche Einsparungen durch den Einsatz von Reserved Instances, Graviton-Prozessoren oder Spot zu nutzen.

Bei gemeinsam genutzten Clustern kann es auch zu Ressourcenbeschränkungen auf Clusterebene kommen, wie z. B. IP-Erschöpfung, Kubernetes-Servicebeschränkungen oder API-Skalierungsanforderungen. Sie sollten den Leitfaden mit bewährten Methoden zur Skalierbarkeit lesen, um sicherzustellen, dass Ihre Cluster diese Beschränkungen vermeiden.

Sie können Ressourcen auf Namespace- oder Karpenter Provisioner-Ebene isolieren. Ressourcenkontingente bieten eine Möglichkeit, Grenzen dafür festzulegen, wie viele Ressourcen Workloads in einem Namespace verbrauchen können. Dies kann ein guter erster Schutzwall sein, sollte aber kontinuierlich überprüft werden, um sicherzustellen, dass die Skalierung von Workloads nicht künstlich eingeschränkt wird.

Die Provisioner von Karpenter können Beschränkungen für einige der nutzbaren Ressourcen in einem Cluster (z. B. CPU, GPU) festlegen, aber Sie müssen Mandantenanwendungen so konfigurieren, dass sie den entsprechenden Provisioner verwenden. Auf diese Weise kann verhindert werden, dass ein einzelner Provisioner zu viele Knoten in einem Cluster erstellt. Es sollte jedoch kontinuierlich überprüft werden, um sicherzustellen, dass das Limit nicht zu niedrig angesetzt ist, um zu verhindern, dass Workloads skaliert werden.

Geplante automatische Skalierung

Möglicherweise müssen Sie Ihre Cluster an Wochenenden und außerhalb der Geschäftszeiten herunterskalieren. Dies ist besonders relevant für Test- und Nicht-Produktionscluster, bei denen Sie auf Null herunterskalieren möchten, wenn sie nicht verwendet werden. Lösungen wie Cluster-Turndown können die Replikate auf der Grundlage eines Cron-Zeitplans auf Null herunterskalieren. Sie können dies auch mit Karpenter erreichen, das im folgenden AWS-Blog beschrieben wird.

Optimieren Sie die Arten von Rechenkapazitäten

Nachdem Sie die Gesamtkapazität der Rechenleistung in Ihrem Cluster optimiert und Bin-Packing verwendet haben, sollten Sie sich ansehen, welche Art von Rechenleistung Sie in Ihren Clustern bereitgestellt haben und wie Sie für diese Ressourcen bezahlen. AWS bietet Compute Saving-Pläne, mit denen Sie die Kosten für Ihre Rechenleistung senken können. Wir unterteilen diese Pläne in die folgenden Kapazitätstypen:

- Spot-Instances
- Savings Plans
- On-Demand
- Fargate

Jeder Kapazitätstyp hat unterschiedliche Kompromisse in Bezug auf Verwaltungsaufwand, Verfügbarkeit und langfristige Verpflichtungen, und Sie müssen entscheiden, welcher für Ihre Umgebung am besten geeignet ist. Keine Umgebung sollte auf einen einzelnen Kapazitätstyp angewiesen sein. Sie können mehrere Ausführungstypen in einem einzigen Cluster kombinieren, um spezifische Workload-Anforderungen und Kosten zu optimieren.

Spot-Instances

Der Spot-Kapazitätstyp stellt EC2 Instances aus freien Kapazitäten in einer Availability Zone bereit. Spot bietet die größten Rabatte — bis zu 90% —, aber diese Instances können unterbrochen werden, wenn sie an anderer Stelle benötigt werden. Darüber hinaus ist möglicherweise nicht immer Kapazität für die Bereitstellung neuer Spot-Instances vorhanden, und bestehende Spot-Instances können mit einer Unterbrechungsbenachrichtigung von 2 Minuten zurückgeholt werden. Wenn Ihre Anwendung einen langen Start- oder Abschaltvorgang hat, sind Spot-Instances möglicherweise nicht die beste Option.

Spot-Computing sollte eine Vielzahl von Instance-Typen verwenden, um die Wahrscheinlichkeit zu verringern, dass keine Spot-Kapazität verfügbar ist. Instance-Unterbrechungen müssen behandelt werden, um Knoten sicher herunterzufahren. Knoten, die mit Karpenter bereitgestellt wurden oder Teil einer Managed Node Group sind, unterstützen automatisch Benachrichtigungen über Instanzunterbrechungen. Wenn Sie selbstverwaltete Knoten verwenden, müssen Sie den Node Termination Handler separat ausführen, um Spot-Instances ordnungsgemäß herunterzufahren.

Es ist möglich, Spot- und On-Demand-Instances in einem einzigen Cluster auszugleichen. Mit Karpenter können Sie gewichtete Provisioner erstellen, um ein ausgewogenes Verhältnis zwischen verschiedenen Kapazitätstypen zu erreichen. Mit Cluster Autoscaler können Sie gemischte Knotengruppen mit Spot - und On-Demand-Instances oder Reserved Instances erstellen.

Hier ist ein Beispiel für die Verwendung von Karpenter zur Priorisierung von Spot-Instances vor On-Demand-Instances. Wenn Sie einen Provisioner erstellen, können Sie entweder Spot, On-Demand oder beides angeben (wie unten gezeigt). Wenn Sie beide angeben und der Pod nicht explizit angibt, ob er Spot oder On-Demand verwenden muss, priorisiert Karpenter Spot bei der Bereitstellung eines Knotens mit Zuweisungsstrategie. 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, Reserved Instances und AWS EDP

Sie können Ihre Rechenausgaben reduzieren, indem Sie einen Sparplan für Rechenleistung verwenden. Sparpläne bieten reduzierte Preise für eine Laufzeit von 1 oder 3 Jahren für die Computernutzung. Die Nutzung kann für EC2 Instances in einem EKS-Cluster gelten, gilt aber auch für jede Computenutzung wie Lambda und Fargate. Mit Sparplänen können Sie die Kosten senken und während Ihres Abonnementzeitraums trotzdem einen beliebigen EC2 Instance-Typ auswählen.

Mit dem Compute Savings Plan können Sie Ihre EC2 Kosten um bis zu 66% senken, ohne dass Sie sich darauf festlegen müssen, welche Instance-Typen, Familien oder Regionen Sie verwenden möchten. Einsparungen werden automatisch auf Instances angewendet, sobald Sie sie verwenden.

EC2 Instance Savings Plans bieten bis zu 72% Einsparungen bei der Rechenleistung, wenn die Nutzung in einer bestimmten Region und EC2 Familie festgelegt ist, z. B. bei Instances der C-Familie. Sie können die Nutzung auf eine beliebige AZ innerhalb der Region verlagern, eine beliebige Generation der Instance-Familie verwenden, z. B. c5 oder c6, und Instances jeder Größe innerhalb der Familie verwenden. Der discount wird automatisch für jede Instanz auf Ihrem Konto angewendet, die den Sparplankriterien entspricht.

Reserved Instances ähneln dem EC2 Instance-Sparplan, garantieren jedoch auch Kapazität in einer Availability Zone oder Region und reduzieren die Kosten im Vergleich zu On-Demand-Instances um bis zu 72%. Sobald Sie berechnet haben, wie viel reservierte Kapazität Sie benötigen, können Sie auswählen, für wie lange Sie sie reservieren möchten (1 Jahr oder 3 Jahre). Die Rabatte werden automatisch angewendet, wenn Sie diese EC2 Instances in Ihrem Konto ausführen.

Kunden haben auch die Möglichkeit, ein Enterprise Agreement mit AWS abzuschließen. Unternehmensverträge bieten Kunden die Möglichkeit, Verträge so zu gestalten, dass sie ihren Bedürfnissen am besten entsprechen. Kunden können Rabatte auf die Preise erhalten, die auf dem AWS EDP (Enterprise Discount Program) basieren. Für weitere Informationen zu Enterprise Agreements wenden Sie sich bitte an Ihren AWS-Vertriebsmitarbeiter.

On-Demand

EC2 On-Demand-Instances bieten im Vergleich zu Spot-Instances den Vorteil einer unterbrechungsfreien Verfügbarkeit und im Vergleich zu Sparplänen keine langfristigen

Verpflichtungen. Wenn Sie die Kosten in einem Cluster senken möchten, sollten Sie die Nutzung von EC2 On-Demand-Instances reduzieren.

Nach der Optimierung Ihrer Workload-Anforderungen sollten Sie in der Lage sein, eine Mindest- und Höchstkapazität für Ihre Cluster zu berechnen. Diese Zahl kann sich im Laufe der Zeit ändern, sinkt aber selten. Erwägen Sie, einen Savings Plan für alles zu verwenden, was unter dem Minimum liegt, und achten Sie auf Kapazitäten, die die Verfügbarkeit Ihrer Anwendung nicht beeinträchtigen. Alles andere, was möglicherweise nicht kontinuierlich genutzt wird oder für die Verfügbarkeit erforderlich ist, kann bei Bedarf verwendet werden.

Wie in diesem Abschnitt erwähnt, können Sie Ihren Verbrauch am besten reduzieren, indem Sie weniger Ressourcen verbrauchen und die von Ihnen bereitgestellten Ressourcen so weit wie möglich nutzen. Mit dem Cluster Autoscaler können Sie nicht ausgelastete Knoten mit der Einstellung entfernen. `scale-down-utilization-threshold` Bei Karpenter wird empfohlen, die Konsolidierung zu aktivieren.

Um manuell EC2 Instance-Typen zu identifizieren, die mit Ihren Workloads verwendet werden können, sollten Sie den ec2-instance-selector verwenden. Dieser kann sowohl Instances anzeigen, die in jeder Region verfügbar sind, als auch Instances, die mit EKS kompatibel sind. Anwendungsbeispiel für Instances mit x86-Prozessarchitektur, 4 GB Arbeitsspeicher, 2 V CPUs und verfügbar in der Region 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
```

In Umgebungen außerhalb der Produktionsumgebung können Sie Cluster während ungenutzter Stunden, z. B. nachts und am Wochenende, automatisch herunterskalieren lassen. Das Cluster-Turndown des Kubecost-Projekts ist ein Beispiel für einen Controller, der Ihren Cluster nach einem festgelegten Zeitplan automatisch herunterskalieren kann.

Fargate-Computer

Fargate Compute ist eine vollständig verwaltete Rechenoption für EKS-Cluster. Sie ermöglicht die Pod-Isolierung, indem ein Pod pro Knoten in einem Kubernetes-Cluster geplant wird. Es ermöglicht Ihnen, Ihre Rechenknoten an die CPU- und RAM-Anforderungen Ihres Workloads anzupassen, um die Workload-Nutzung in einem Cluster genau zu kontrollieren.

Fargate kann Workloads auf bis zu 2,5 vCPU mit 0,5 GB Arbeitsspeicher und bis zu 16 vCPUs mit 120 GB Arbeitsspeicher skalieren. Die Anzahl der verfügbaren Varianten der Pod-Größe ist begrenzt, und Sie müssen wissen, wie Ihr Workload am besten in eine Fargate-Konfiguration passt. Wenn Ihre Arbeitslast beispielsweise 1 vCPU mit 0,5 GB Arbeitsspeicher erfordert, ist der kleinste Fargate-Pod 1 vCPU mit 2 GB Arbeitsspeicher.

Fargate bietet zwar viele Vorteile, z. B. keine EC2 Instanz- oder Betriebssystemverwaltung, erfordert jedoch möglicherweise mehr Rechenkapazität als herkömmliche EC2 Instances, da jeder bereitgestellte Pod als separater Knoten im Cluster isoliert ist. Dies erfordert mehr Duplizierung für Dinge wie Kubelet, Logging-Agenten und alles, was DaemonSets Sie normalerweise auf einem Knoten bereitstellen würden. DaemonSets werden in Fargate nicht unterstützt und müssen in Pod-„`Sidecars"` umgewandelt und zusammen mit der Anwendung ausgeführt werden.

Fargate kann nicht von Bin-Packing oder CPU-Over-Provisioning profitieren, da die Grenze für die Arbeitslast ein Knoten ist, der nicht burstable ist oder von mehreren Workloads gemeinsam genutzt werden kann. Fargate spart Ihnen Zeit bei der EC2 Instanzverwaltung, was wiederum Kosten verursacht, aber die CPU- und Speicherkosten können teurer sein als bei anderen EC2 Kapazitätstypen. Fargate-Pods können den Rechensparplan nutzen, um die On-Demand-Kosten zu senken.

Optimieren Sie die Computernutzung

Eine weitere Möglichkeit, Geld bei Ihrer Recheninfrastruktur zu sparen, besteht darin, effizientere Rechenleistung für die Arbeitslast zu verwenden. Dies kann durch leistungsfähigere Allzweck-Rechenleistung wie Graviton-Prozessoren erreicht werden, die bis zu 20% günstiger und 60% energieeffizienter als x86 sind, oder durch Workload-spezifische Beschleuniger wie und. GPUs FPGAs Sie müssen Container erstellen, die auf einer ARM-Architektur ausgeführt werden können, und Knoten mit den richtigen Beschleunigern für Ihre Workloads einrichten.

EKS ist in der Lage, Cluster mit gemischter Architektur (z. B. amd64 und arm64) auszuführen. Wenn Ihre Container für mehrere Architekturen kompiliert sind, können Sie die Vorteile von Graviton-

Prozessoren mit Karpenter nutzen, indem Sie beide Architekturen in Ihrem Provisioner zulassen. Um eine gleichbleibende Leistung zu gewährleisten, wird jedoch empfohlen, jeden Workload auf einer einzigen Rechenarchitektur zu belassen und nur dann eine andere Architektur zu verwenden, wenn keine zusätzliche Kapazität verfügbar ist.

Bereitsteller können mit mehreren Architekturen konfiguriert werden, und Workloads können in ihrer Workload-Spezifikation auch bestimmte Architekturen anfordern.

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
  name: default
spec:
  requirements:
  - key: "kubernetes.io/arch"
    operator: In
    values: ["arm64", "amd64"]
```

Mit Cluster Autoscaler müssen Sie eine Knotengruppe für Graviton-Instanzen erstellen und Knotentoleranzen für Ihren Workload festlegen, um die neue Kapazität nutzen zu können.

GPUs und FPGAs kann die Leistung für Ihren Workload erheblich steigern, allerdings muss der Workload für die Nutzung des Accelerators optimiert werden. Viele Workload-Typen, die für maschinelles Lernen und künstliche Intelligenz GPUs zur Datenverarbeitung verwendet werden können, und Instanzen können einem Cluster hinzugefügt und mithilfe von Ressourcenanfragen in einen Workload eingebunden werden.

```
spec:
  template:
    spec:
    - containers:
      ...
      resources:
          limits:
            nvidia.com/gpu: "1"
```

Manche GPU-Hardware kann von mehreren Workloads gemeinsam genutzt werden, sodass eine einzelne GPU bereitgestellt und verwendet werden kann. Weitere Informationen zur Konfiguration der GPU-Sharing für Workloads finden Sie im Plug-in für virtuelle GPU-Geräte. Sie können sich auch auf die folgenden Blogs beziehen:

- GPU-Sharing auf Amazon EKS mit NVIDIA Timeslicing und beschleunigten Instances EC2
- Maximierung der GPU-Auslastung mit NVIDIA Multi-Instance GPU (MIG) in Amazon EKS: Ausführung von mehr Pods pro GPU für verbesserte Leistung

Kostenoptimierung — Netzwerke

Die Architektur von Systemen für hohe Verfügbarkeit (HA) ist eine bewährte Methode, um Ausfallsicherheit und Fehlertoleranz zu erreichen. In der Praxis bedeutet dies, dass Sie Ihre Workloads und die zugrunde liegende Infrastruktur auf mehrere Availability Zones (AZs) in einer bestimmten AWS-Region verteilen. Wenn Sie sicherstellen, dass diese Eigenschaften für Ihre Amazon EKS-Umgebung vorhanden sind, wird die allgemeine Zuverlässigkeit Ihres Systems verbessert. In Verbindung damit werden Ihre EKS-Umgebungen wahrscheinlich auch aus einer Vielzahl von Konstrukten (d. h. VPCs), Komponenten (d. h.) und Integrationen (d. h. ECR und anderen Container-Registern ELBs) bestehen.

Die Kombination aus hochverfügbaren Systemen und anderen anwendungsspezifischen Komponenten kann eine wichtige Rolle bei der Übertragung und Verarbeitung von Daten spielen. Dies wird sich wiederum auf die Kosten auswirken, die durch die Datenübertragung und -verarbeitung entstehen.

Die unten aufgeführten Methoden helfen Ihnen beim Entwerfen und Optimieren Ihrer EKS-Umgebungen, um Kosteneffektivität für verschiedene Bereiche und Anwendungsfälle zu erreichen.

Kommunikation von Pod zu Pod

Abhängig von Ihrer Konfiguration können Netzwerkkommunikation und Datenübertragung zwischen Pods erhebliche Auswirkungen auf die Gesamtkosten der Ausführung von Amazon EKS-Workloads haben. In diesem Abschnitt werden verschiedene Konzepte und Ansätze zur Reduzierung der mit der Kommunikation zwischen Pods verbundenen Kosten behandelt, wobei hochverfügbare Architekturen (HA), Anwendungsleistung und Ausfallsicherheit berücksichtigt werden.

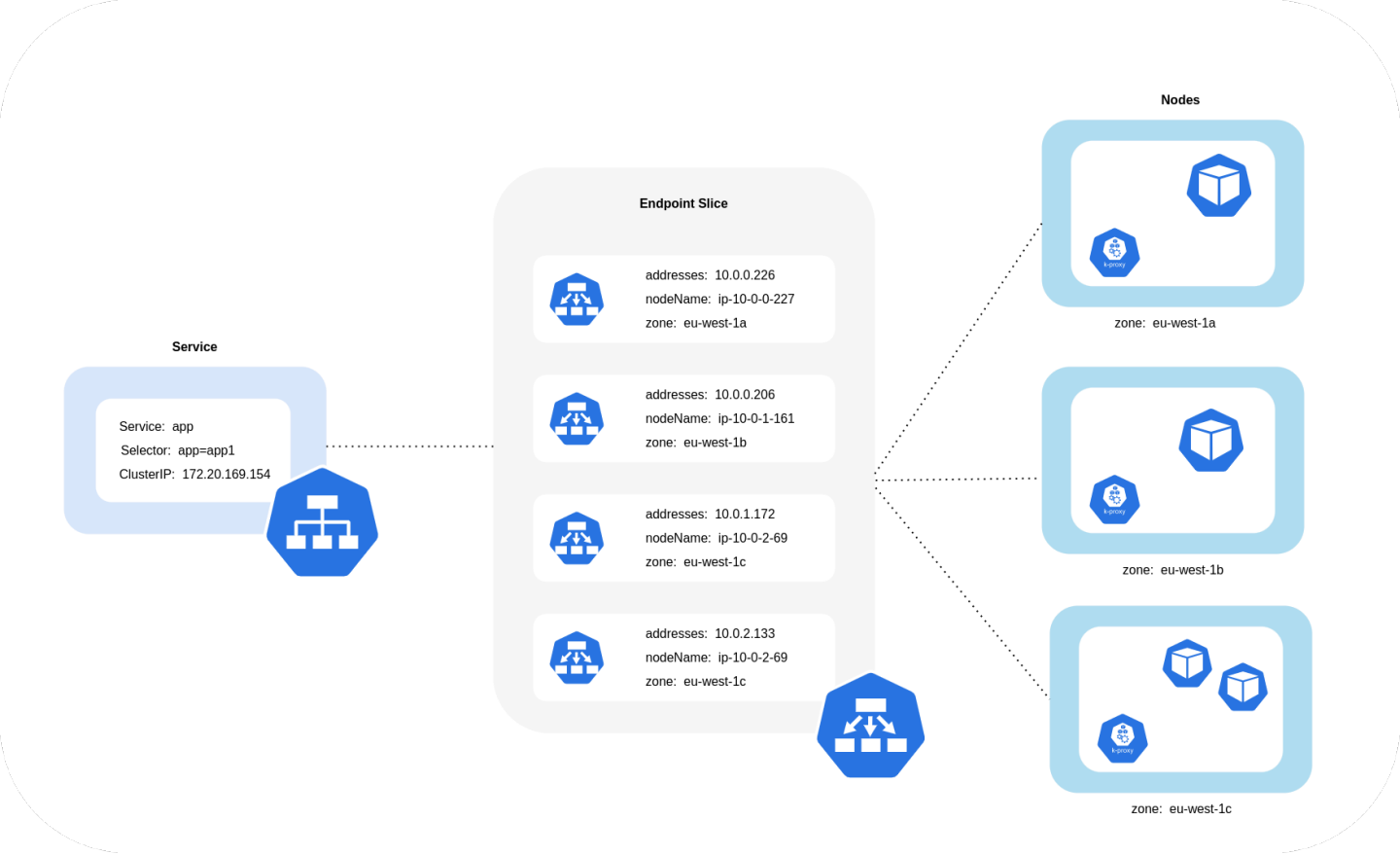
Beschränkung des Datenverkehrs auf eine Availability Zone

Das Kubernetes-Projekt begann schon früh mit der Entwicklung topologieorientierter Konstrukte, einschließlich Bezeichnungen wie Kubernetes. io/hostname, topology.kubernetes.io/region, and topology.kubernetes.io/zoneKnoten zugewiesen, um Funktionen wie die Verteilung der Arbeitslast auf Ausfalldomänen und topologieorientierte Volume Provisioner zu ermöglichen. Nach Abschluss

des Studiums in Kubernetes 1.17 wurden die Labels auch genutzt, um topologieorientierte Routing-Funktionen für die Pod-zu-Pod-Kommunikation zu ermöglichen.

Im Folgenden finden Sie einige Strategien zur Steuerung des AZ-übergreifenden Datenverkehrs zwischen Pods in Ihrem EKS-Cluster, um die Kosten zu senken und die Latenz zu minimieren.

Wenn Sie einen detaillierten Überblick über die Menge des AZ-übergreifenden Datenverkehrs zwischen Pods in Ihrem Cluster wünschen (z. B. die Menge der übertragenen Daten in Byte), lesen Sie diesen Beitrag.



Wie das obige Diagramm zeigt, sind Dienste die stabile Netzwerkabstraktionsschicht, die den für Ihre Pods bestimmten Datenverkehr empfängt. Wenn ein Dienst erstellt wird, werden mehrere EndpointSlices erstellt. Jeder EndpointSlice hat eine Liste von Endpunkten, die eine Teilmenge von Pod-Adressen zusammen mit den Knoten, auf denen sie ausgeführt werden, und allen zusätzlichen Topologieinformationen enthält. Bei Verwendung des Amazon VPC CNI verwaltet kube-proxy, ein Daemonset, das auf jedem Knoten läuft, Netzwerkregeln, um die Pod-Kommunikation und die Serviceerkennung zu ermöglichen (alternative EBPF-basierte Lösungen verwenden CNIs möglicherweise keinen Kube-Proxy, bieten aber ein gleichwertiges Verhalten). Es erfüllt die Rolle

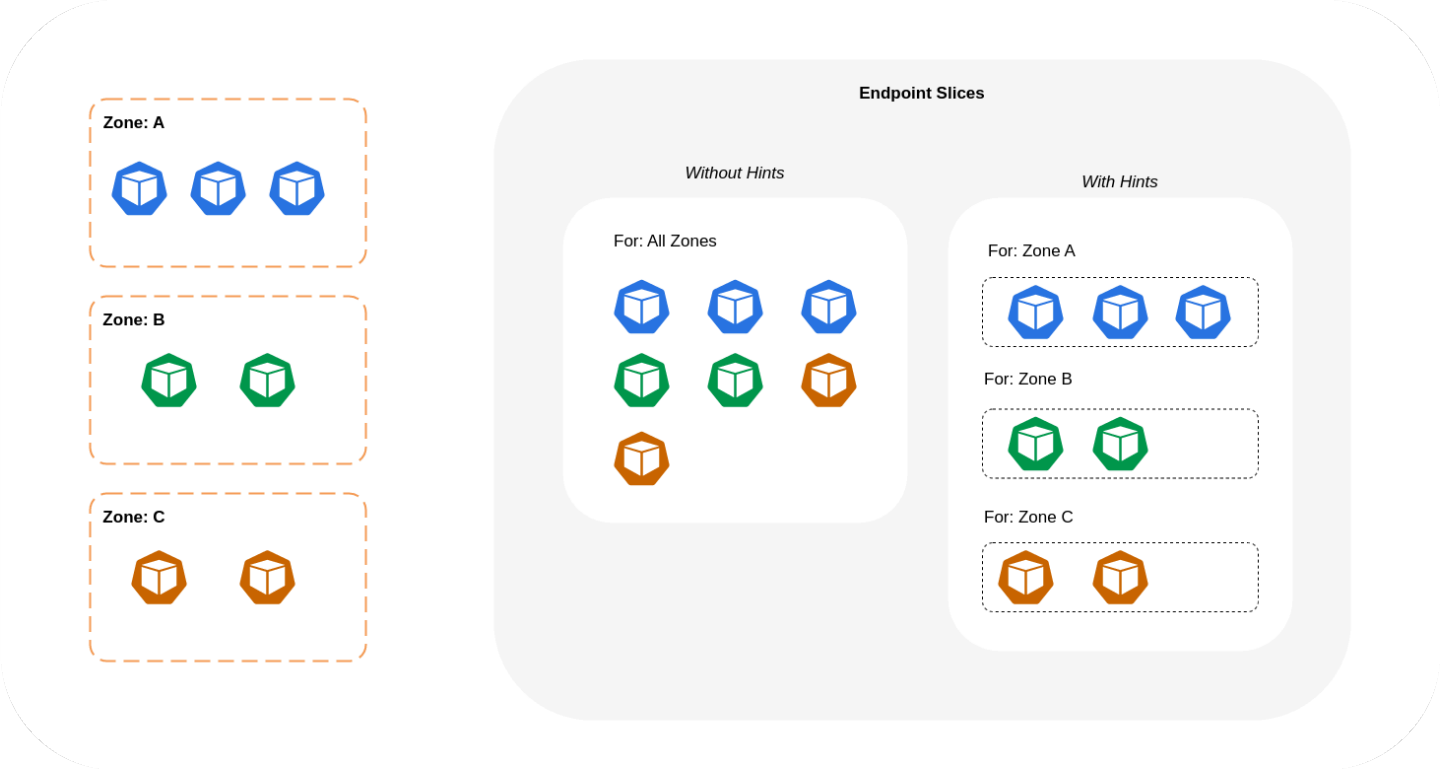
des internen Routings, aber es tut dies auf der Grundlage dessen, was es von den erstellten Daten verbraucht. EndpointSlices

Auf EKS verwendet Kube-Proxy hauptsächlich iptables-NAT-Regeln (oder IPVS NFTablesals Alternative) für die Verteilung des Datenverkehrs auf alle Pods im Cluster, unabhängig von deren Knoten- oder AZ-Platzierung. Diese Standardverteilung kann zu einer AZ-übergreifenden Weiterleitung des Datenverkehrs führen, was möglicherweise zu einer erhöhten Latenz für sensible Anwendungen und zu Gebühren für die Datenübertragung zwischen den einzelnen AZ in großen Bereitstellungen führen kann.

Verwenden von Topology Aware Routing (früher bekannt als Topology Aware Hints)

Wenn topologiebewusstes Routing aktiviert und auf einem Kubernetes-Service implementiert ist, weist der EndpointSlice Controller den verschiedenen Zonen, auf die Ihr Cluster verteilt ist, Endpunkte proportional zu. Für jeden dieser Endpunkte setzt der EndpointSlice Controller auch einen Hinweis für die Zone. Hinweise beschreiben, für welche Zone ein Endpunkt den Verkehr bereitstellen soll. `kube-proxy`leitet dann den Verkehr von einer Zone zu einem Endpunkt weiter, basierend auf den Hinweisen, die angewendet werden.

Das folgende Diagramm zeigt, wie EndpointSlices die With-Hinweise so organisiert sind, dass anhand ihres zonalen Ausgangspunkts sofort erkannt werden `kube-proxy` kann, zu welchem Ziel sie gehen sollen. Ohne Hinweise gibt es keine solche Zuordnung oder Organisation, und der Verkehr wird an verschiedene zonale Ziele weitergeleitet, unabhängig davon, woher er kommt.



In einigen Fällen wendet der EndpointSlice Controller möglicherweise einen Hinweis für eine andere Zone an, was bedeutet, dass der Endpunkt am Ende Datenverkehr aus einer anderen Zone bedienen könnte. Der Grund dafür ist der Versuch, eine gleichmäßige Verteilung des Datenverkehrs zwischen Endpunkten in verschiedenen Zonen aufrechtzuerhalten.

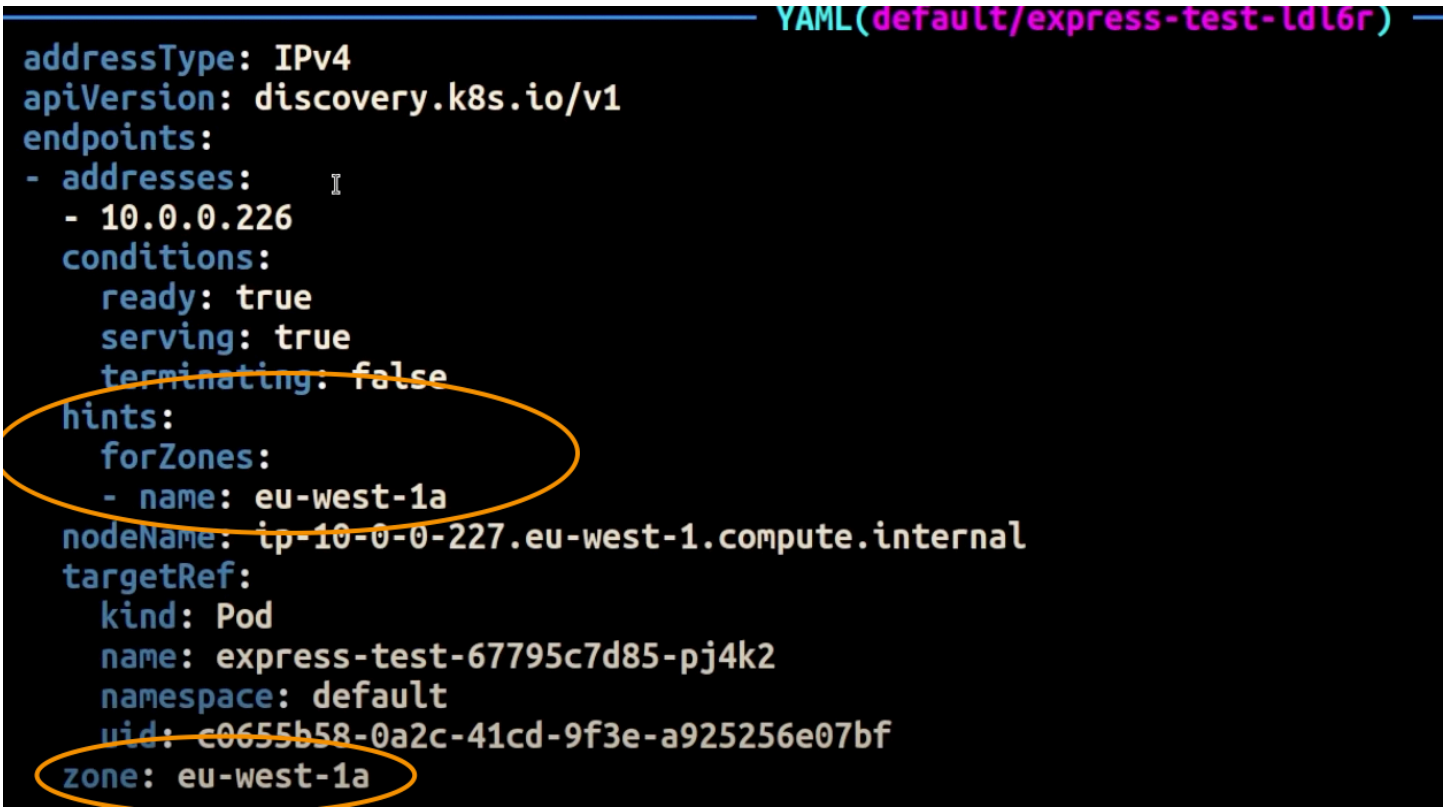
Im Folgenden finden Sie einen Codeausschnitt zur Aktivierung von topologieorientiertem Routing für einen Dienst.

```
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
```

Der folgende Screenshot zeigt das Ergebnis, dass der EndpointSlice Controller erfolgreich einen Hinweis auf einen Endpunkt für ein Pod-Replikat angewendet hat, das in der AZ ausgeführt wird. `eu-west-1a`



> **Note**
>
> Es ist wichtig zu beachten, dass sich topologiebewusstes Routing noch in der Betaphase befindet. Bei gleichmäßig über die Clustertopologie verteilten Arbeitslasten arbeitet diese Funktion besser vorhersehbar, da der Controller die Endpunkte proportional auf die Zonen verteilt, aber die Zuweisungen von Hinweisen überspringen kann, wenn die Knotenressourcen in einer Zone zu unausgewogen sind, um eine übermäßige Überlastung zu vermeiden. Es wird daher dringend empfohlen, sie zusammen mit Einschränkungen bei der Terminplanung zu verwenden, die die Verfügbarkeit einer Anwendung erhöhen, wie z. B. Einschränkungen bei der Verteilung der Pod-Topologie. Beachten Sie, dass Hinweise möglicherweise auch nicht zugewiesen werden, wenn die Kapazität zwischen den Zonen schwankt, z. B. bei der Verwendung von Amazon EC2-Spot-Instances, da Unterbrechungen

oder Ersetzungen bei der Berechnung der proportionalen Verteilung nicht in Echtzeit erkannt werden.

Verwendung der Verkehrsverteilung

Traffic Distribution wurde in Kubernetes 1.30 eingeführt und in Version 1.33 allgemein verfügbar gemacht. Es bietet eine einfachere Alternative zu topologiebewusstem Routing, um Traffic in derselben Zone zu bevorzugen. Topology Aware Routing versucht zwar, einen intelligenten Ansatz für das Routing des Datenverkehrs zu verwenden, um eine Überlastung der Endpunkte zu vermeiden, führte jedoch zu unvorhersehbarem Verhalten. Bei der Verkehrsverteilung wird stattdessen der Vorhersehbarkeit Priorität eingeräumt. Die PreferClose Option weist Kube-Proxy an, Regeln zu erstellen, die den Datenverkehr auf der Grundlage des vom Controller festgelegten zonalen Hinweises zuerst an Endpunkte derselben Zone weiterleiten. EndpointSlice Wenn keine Endpunkte in derselben Zone verfügbar sind, wird der Datenverkehr für den Service auf einen beliebigen Cluster-Endpunkt verteilt. Diese Funktion wurde für Workloads entwickelt, bei denen der Kompromiss zwischen der Optimierung der Nähe und nicht der versuchten gleichmäßigen Lastverteilung, die Topology Aware Routing bietet, in Kauf genommen wird.

Im Folgenden finden Sie einen Codeausschnitt zur Aktivierung der Verkehrsverteilung für einen Dienst.

```
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
```

Bei der Aktivierung der Verkehrsverteilung tritt ein häufiges Problem auf: Endpunkte innerhalb einer einzelnen AZ können überlastet werden, wenn der Großteil des Datenverkehrs aus derselben Zone stammt. Diese Überlastung kann zu erheblichen Problemen führen:

- Ein einziger Horizontal Pod Autoscaler (HPA), der eine Multi-AZ-Bereitstellung verwaltet, kann darauf reagieren, indem er Pods auf verschiedene Pods skaliert. AZs Mit dieser Aktion kann die erhöhte Last in der betroffenen Zone jedoch nicht wirksam behoben werden.
- Diese Situation wiederum kann zu einer ineffizienten Nutzung der Ressourcen führen. Wenn Cluster-Autoscaler wie Karpenter erkennen, dass der Pod über verschiedene Bereiche skaliert wird, können sie zusätzliche Knoten in den nicht betroffenen Gebieten bereitstellen AZs, was zu einer AZs unnötigen Ressourcenzuweisung führt.

Um diese Herausforderung zu bewältigen:

- Erstellen Sie separate Bereitstellungen pro Zone, die über eigene Bereitstellungen verfügen HPAs , sodass sie unabhängig voneinander skaliert werden können.
- Nutzen Sie Topology Spread Constraints, um die Verteilung der Arbeitslast im Cluster sicherzustellen und so eine Überlastung der Endpunkte in stark frequentierten Zonen zu verhindern.

Verwenden von Autoscalern: Stellen Sie Knoten für eine bestimmte AZ bereit

Wir empfehlen dringend, Ihre Workloads in hochverfügbaren Umgebungen in mehreren Umgebungen auszuführen. AZs Dies verbessert die Zuverlässigkeit Ihrer Anwendungen, insbesondere wenn ein Problem mit einer AZ auftritt. Falls Sie bereit sind, die Zuverlässigkeit zu opfern, um die Netzwerkkosten zu senken, können Sie Ihre Knoten auf eine einzige AZ beschränken.

Wenn Sie alle Ihre Pods in derselben AZ ausführen möchten, stellen Sie entweder die Worker-Knoten in derselben AZ bereit oder planen Sie die Pods auf den Worker-Knoten, die auf derselben AZ laufen. Um Knoten innerhalb einer einzigen AZ bereitzustellen, definieren Sie mit Cluster Autoscaler (CA) eine Knotengruppe mit Subnetzen, die zu derselben AZ gehören. Verwenden Sie für Karpenter die AZ, in der Sie die Worker-Knoten erstellen möchten, `topology.kubernetes.io/zone` und geben Sie sie an. Das folgende Karpenter-Provisioner-Snippet stellt beispielsweise die Knoten in der AZ us-west-2a bereit.

Karpenter

```
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 (Kalifornien)

```
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
...
```

Verwendung von Pod-Zuweisung und Knotenaffinität

Wenn Sie Worker-Knoten haben, die in mehreren ausgeführt werden AZs, hätte alternativ jeder Knoten die Bezeichnung topology.kubernetes.io/zone mit dem Wert seiner AZ (z. B. us-west-2a oder us-west-2b). Sie können Pods für die Knoten in einer einzigen AZ verwenden oder so planen. `nodeSelector` `nodeAffinity` Mit der folgenden Manifestdatei wird beispielsweise der Pod innerhalb eines Knotens geplant, der in AZ us-west-2a ausgeführt wird.

```
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
```

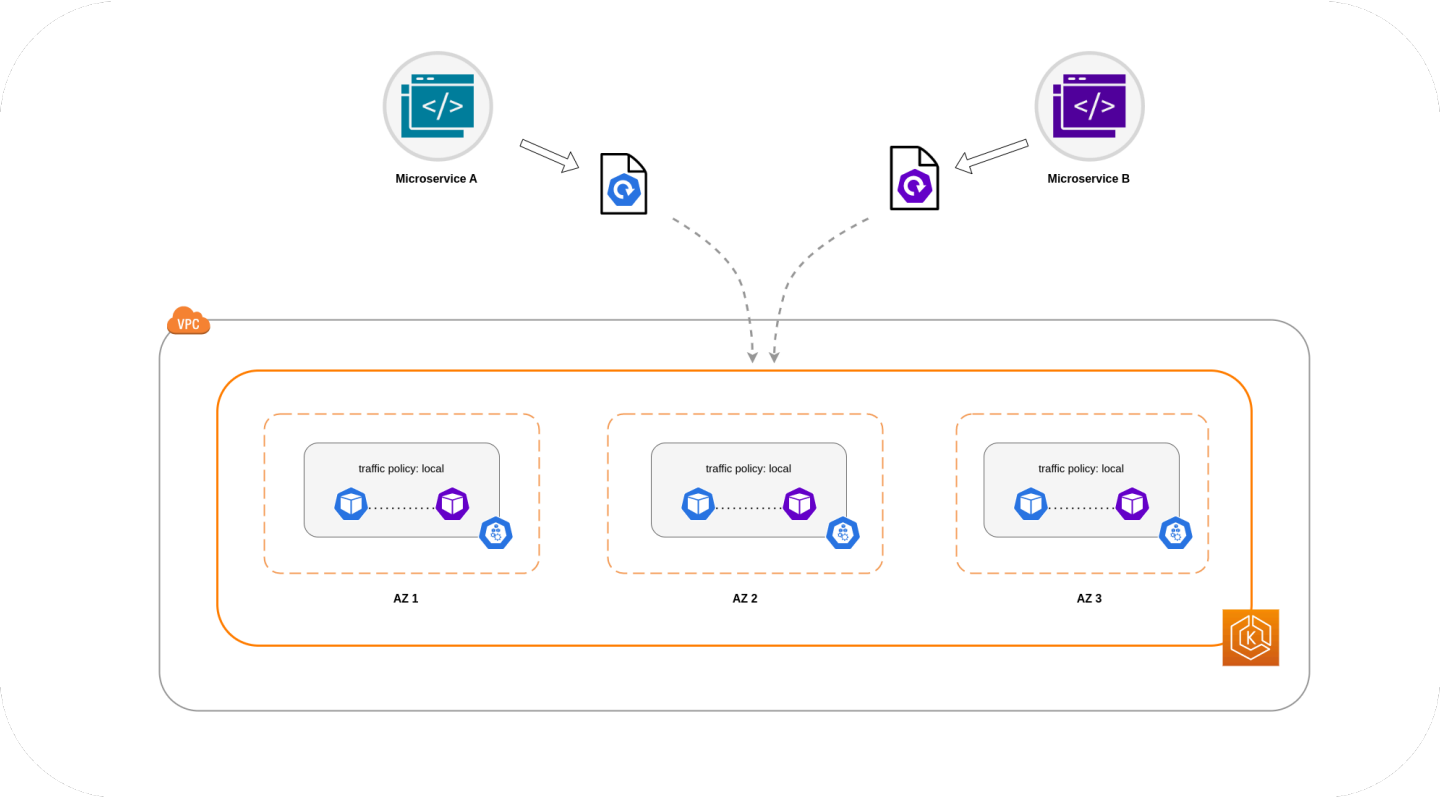
Beschränkung des Datenverkehrs auf einen Knoten

Es gibt Fälle, in denen es nicht ausreicht, den Verkehr auf zonaler Ebene zu beschränken. Neben der Kostensenkung müssen Sie möglicherweise auch die Netzwerklatenz zwischen bestimmten Anwendungen reduzieren, die häufig miteinander kommunizieren. Um eine optimale Netzwerkleistung zu erzielen und die Kosten zu senken, benötigen Sie eine Möglichkeit, den Datenverkehr auf einen bestimmten Knoten zu beschränken. Beispielsweise sollte Microservice A immer mit Microservice B auf Knoten 1 kommunizieren, selbst in Umgebungen mit hoher Verfügbarkeit (HA). Wenn Microservice A auf Knoten 1 mit Microservice B auf Knoten 2 kommuniziert, kann sich dies negativ auf die gewünschte Leistung für Anwendungen dieser Art auswirken, insbesondere wenn sich Knoten 2 in einer separaten AZ befindet.

Verwendung der internen Verkehrsrichtlinie des Dienstes

Um den Pod-Netzwerkverkehr auf einen Knoten zu beschränken, können Sie die interne Verkehrsrichtlinie des Dienstes verwenden. Standardmäßig wird der an den Dienst eines Workloads gesendete Datenverkehr nach dem Zufallsprinzip auf die verschiedenen generierten Endpunkte verteilt. In einer HA-Architektur bedeutet das also, dass der Datenverkehr von Microservice A zu einem beliebigen Replikat von Microservice B auf einem beliebigen Knoten auf den verschiedenen Knoten übertragen werden kann. AZs Wenn die interne Verkehrsrichtlinie des Dienstes jedoch auf eingestellt ist`Local`, wird der Verkehr auf Endpunkte auf dem Knoten beschränkt, von dem der Datenverkehr stammt. Diese Richtlinie schreibt die ausschließliche Verwendung von knotenlokalen Endpunkten vor. Das bedeutet, dass Ihre mit dem Netzwerkverkehr verbundenen Kosten für diesen Workload niedriger sein werden, als wenn die Verteilung clusterweit erfolgen würde. Außerdem wird die Latenz geringer sein, wodurch Ihre Anwendung leistungsfähiger wird.

> **Note**
>
> Es ist wichtig zu beachten, dass diese Funktion nicht mit topologieorientiertem Routing in Kubernetes kombiniert werden kann.



Im Folgenden finden Sie einen Codeausschnitt zum Festlegen der internen Verkehrsrichtlinie für einen Dienst.

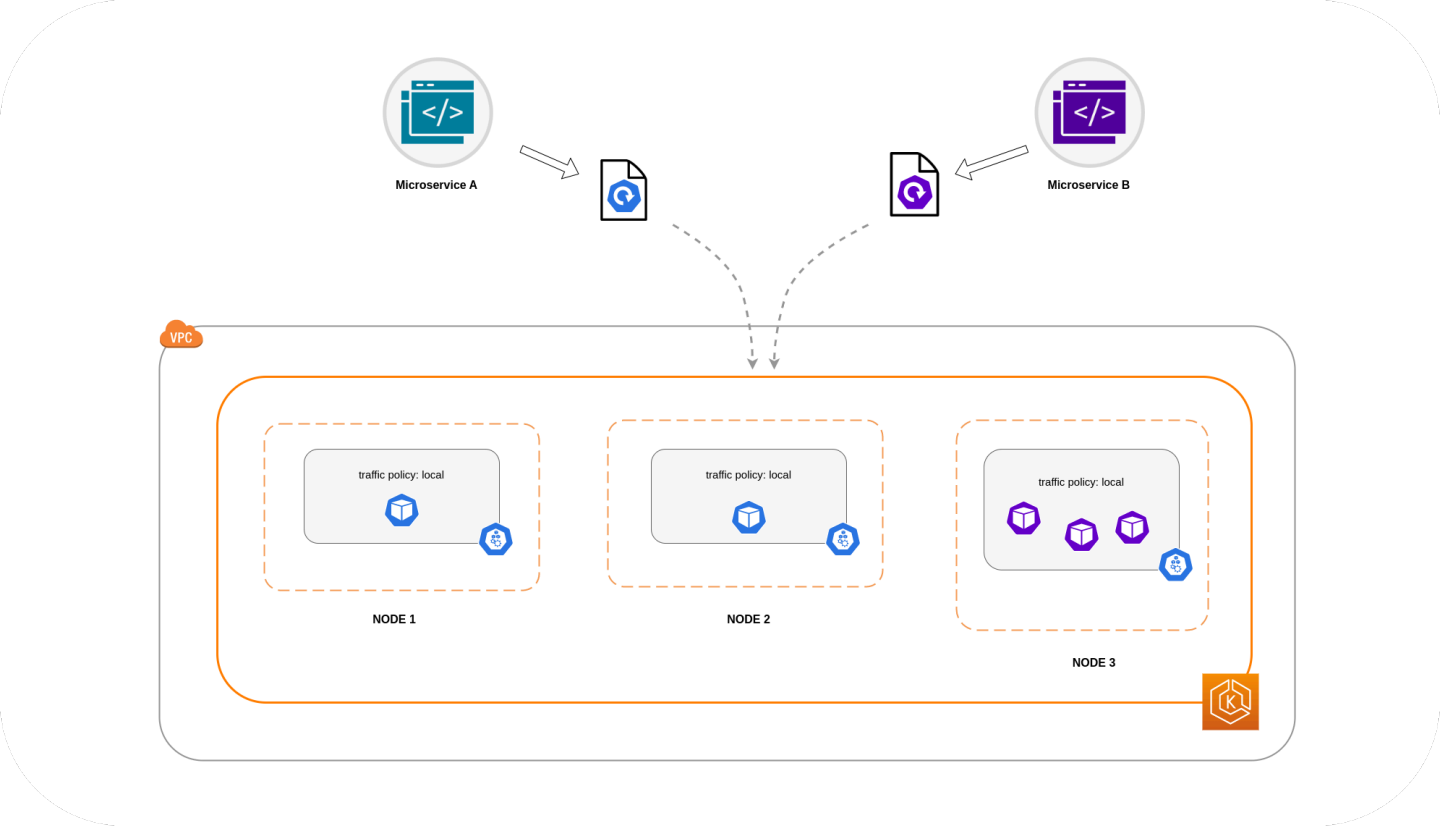
```
apiVersion: v1
kind: Service
metadata:
  name: orders-service
  namespace: ecommerce
spec:
  selector:
    app: orders
  type: ClusterIP
  ports:
```

```
* protocol: TCP
port: 3003
targetPort: 3003
  internalTrafficPolicy: Local
```

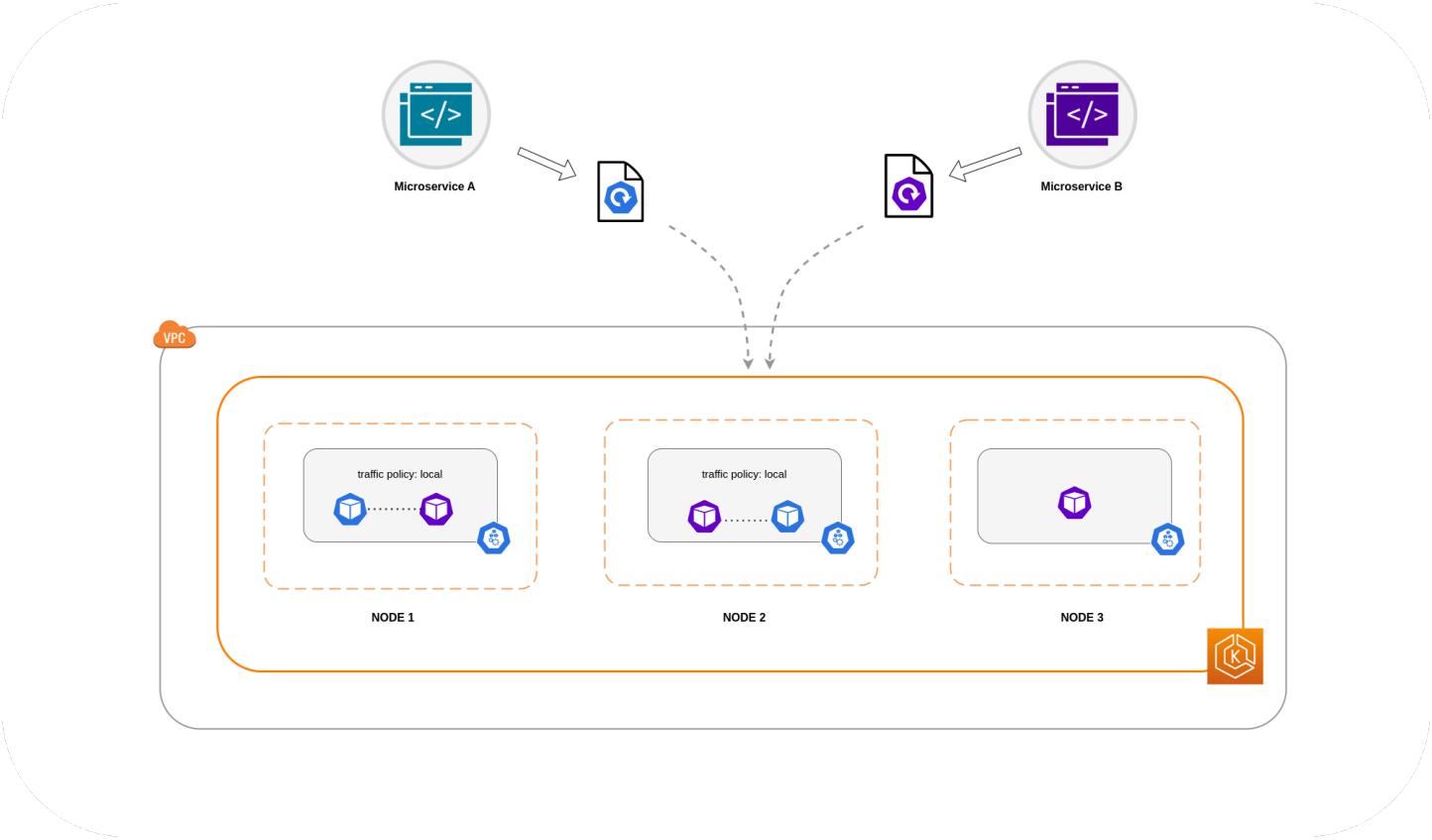
Um ein unerwartetes Verhalten Ihrer Anwendung aufgrund von Datenverkehrsausfällen zu vermeiden, sollten Sie die folgenden Ansätze in Betracht ziehen:

- Führen Sie genügend Replikate für jeden der kommunizierenden Pods aus
- Sorgen Sie mithilfe von Beschränkungen der Topologieverteilung für eine relativ gleichmäßige Verteilung der Pods
- Verwenden Sie Pod-Affinitätsregeln für die Kolokation kommunizierender Pods

In diesem Beispiel haben Sie 2 Replikate von Microservice A und 3 Repliken von Microservice B. Wenn Microservice A seine Replikate auf Knoten 1 und 2 verteilt hat und Microservice B alle 3 Replikate auf Knoten 3 hat, können sie aufgrund der internen Datenverkehrsrichtlinie nicht kommunizieren. `Local` Wenn keine knotenlokalen Endpunkte verfügbar sind, wird der Datenverkehr unterbrochen.



Wenn Microservice B 2 seiner 3 Replikate auf den Knoten 1 und 2 hat, findet eine Kommunikation zwischen den Peer-Anwendungen statt. Aber Sie hätten immer noch ein isoliertes Replikat von Microservice B ohne ein Peer-Replikat, mit dem Sie kommunizieren könnten.



> **Note**
>
> In einigen Szenarien ist ein isoliertes Replikat wie das im obigen Diagramm dargestellte möglicherweise kein Grund zur Sorge, wenn es immer noch einem Zweck dient (z. B. der Bearbeitung von Anfragen aus eingehendem externen Verkehr).

Verwenden der internen Datenverkehrsrichtlinie des Dienstes mit Einschränkungen der Topologieverteilung

Die Verwendung der internen Verkehrsrichtlinie in Verbindung mit Einschränkungen der Topologieverteilung kann nützlich sein, um sicherzustellen, dass Sie über die richtige Anzahl von Replikaten für die Kommunikation von Microservices auf verschiedenen Knoten verfügen.

```
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
```

Verwenden der internen Verkehrsrichtlinie des Dienstes mit Pod-Affinitätsregeln

Ein anderer Ansatz besteht darin, Pod-Affinitätsregeln zu verwenden, wenn die interne Verkehrsrichtlinie des Dienstes verwendet wird. Mit Pod-Affinität können Sie den Terminplaner so beeinflussen, dass er bestimmte Pods aufgrund ihrer häufigen Kommunikation zusammenbringt. Durch die Anwendung strenger Planungsbeschränkungen (`requiredDuringSchedulingIgnoredDuringExecution`) auf bestimmte Pods erzielen Sie auf diese Weise bessere Ergebnisse bei der Pod-Kolokation, wenn der Scheduler Pods auf Knoten platziert.

```
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"
```

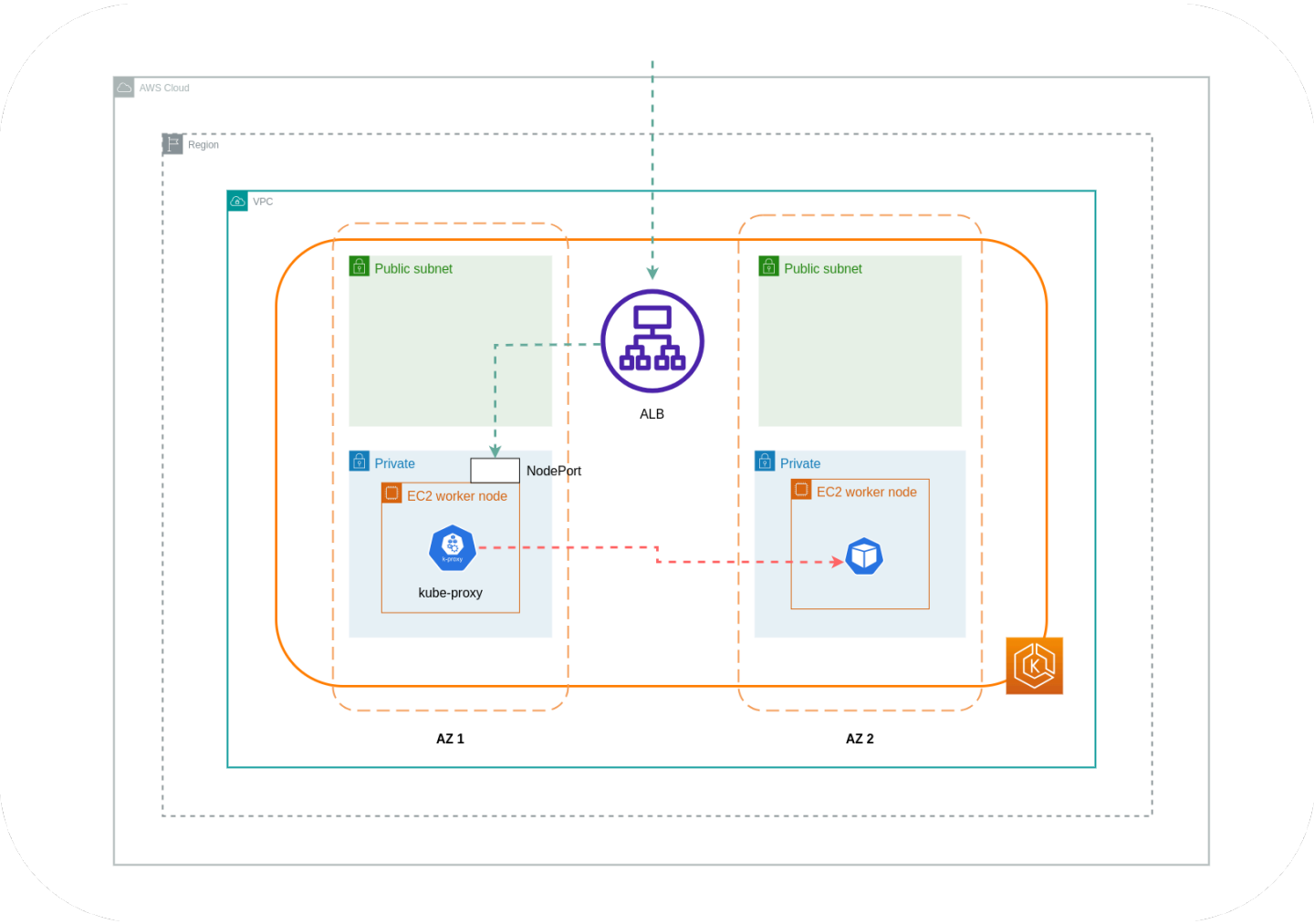
Kommunikation zwischen Load Balancer und Pod

EKS-Workloads werden in der Regel von einem Load Balancer unterstützt, der den Datenverkehr an die entsprechenden Pods in Ihrem EKS-Cluster verteilt. Ihre Architektur kann interne, nach and/or außen gerichtete Load Balancer umfassen. Abhängig von Ihrer Architektur und der Konfiguration des Netzwerkverkehrs kann die Kommunikation zwischen Load Balancern und Pods einen erheblichen Beitrag zu den Datenübertragungsgebühren leisten.

Sie können den AWS Load Balancer Controller verwenden, um die Erstellung von ELB-Ressourcen (ALB und NLB) automatisch zu verwalten. Die Datenübertragungsgebühren, die Ihnen bei solchen Konfigurationen entstehen, hängen vom Pfad ab, den der Netzwerkverkehr zurücklegt. Der AWS Load Balancer Controller unterstützt zwei Netzwerkverkehrsmodi: den Instanzmodus und den IP-Modus.

Wenn Sie den Instance-Modus verwenden, NodePort wird auf jedem Knoten in Ihrem EKS-Cluster eine geöffnet. Der Load Balancer leitet den Datenverkehr dann gleichmäßig über die Knoten weiter. Wenn auf einem Knoten der Ziel-Pod ausgeführt wird, fallen keine Datenübertragungskosten an. Befindet sich der Ziel-Pod jedoch auf einem separaten Knoten und in einer anderen AZ als der, der den Datenverkehr NodePort empfängt, erfolgt ein zusätzlicher Netzwerk-Hop vom Kube-Proxy zum Ziel-Pod. In einem solchen Szenario fallen AZ-übergreifende Datenübertragungsgebühren an. Aufgrund der gleichmäßigen Verteilung des Datenverkehrs auf die Knoten ist es sehr wahrscheinlich, dass im Zusammenhang mit zonenübergreifenden Netzwerk-Traffic-Hops von Kube-Proxys zu den entsprechenden Ziel-Pods zusätzliche Datenübertragungsgebühren anfallen.

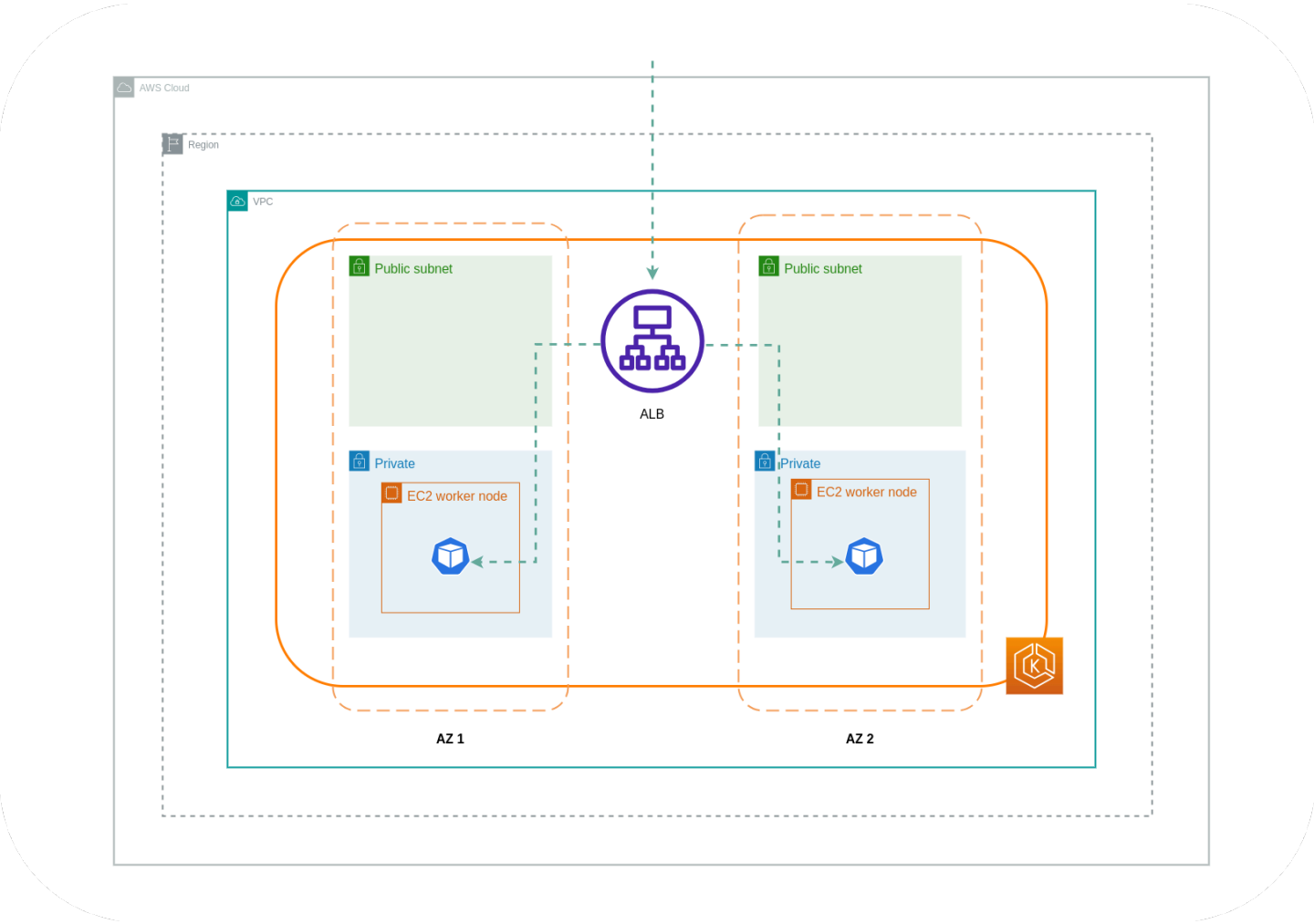
Das folgende Diagramm zeigt einen Netzwerkpfad für den Datenverkehr, der vom Load Balancer zum und anschließend vom Ziel-Pod auf einem separaten `kube-proxy` Knoten in einer anderen AZ fließt. NodePort Dies ist ein Beispiel für die Einstellung des Instanzmodus.



Bei Verwendung des IP-Modus wird der Netzwerkverkehr vom Load Balancer direkt an den Ziel-Pod weitergeleitet. Daher fallen bei diesem Ansatz keine Datenübertragungsgebühren an.

> **Note**
>
> Es wird empfohlen, Ihren Load Balancer auf den IP-Verkehrsmodus einzustellen, um die Datenübertragungsgebühren zu senken. Für dieses Setup ist es auch wichtig, sicherzustellen, dass Ihr Load Balancer in allen Subnetzen in Ihrer VPC eingesetzt wird.

Das folgende Diagramm zeigt Netzwerkpfade für den Datenverkehr, der im Netzwerk-IP-Modus vom Load Balancer zu den Pods fließt.



Datenübertragung aus Container Registry

Amazon ECR

Die Datenübertragung in die private Registrierung von Amazon ECR ist kostenlos. Für die Datenübertragung innerhalb der Region fallen keine Kosten an, aber für Datenübertragungen ins Internet und zwischen Regionen fallen Gebühren für die Internet-Datenübertragung auf beiden Seiten der Übertragung an.

Sie sollten die ECRs integrierte Image-Replikationsfunktion verwenden, um die relevanten Container-Images in dieselbe Region zu replizieren, in der sich Ihre Workloads befinden. Auf diese Weise würde die Replikation einmal in Rechnung gestellt werden, und alle Abrufe von Images aus derselben Region (innerhalb der Region) wären kostenlos.

Sie können die mit dem Abrufen von Bildern aus ECR (ausgehende Datenübertragung) verbundenen Datenübertragungskosten weiter reduzieren, indem Sie Interface VPC Endpoints verwenden, um

eine Verbindung zu den ECR-Repositorys in der Region herzustellen. Der alternative Ansatz, eine Verbindung zum öffentlichen AWS-Endpunkt von ECR herzustellen (über ein NAT-Gateway und ein Internet Gateway), wird höhere Datenverarbeitungs- und Übertragungskosten mit sich bringen. Im nächsten Abschnitt wird die Reduzierung der Datenübertragungskosten zwischen Ihren Workloads und AWS-Services ausführlicher behandelt.

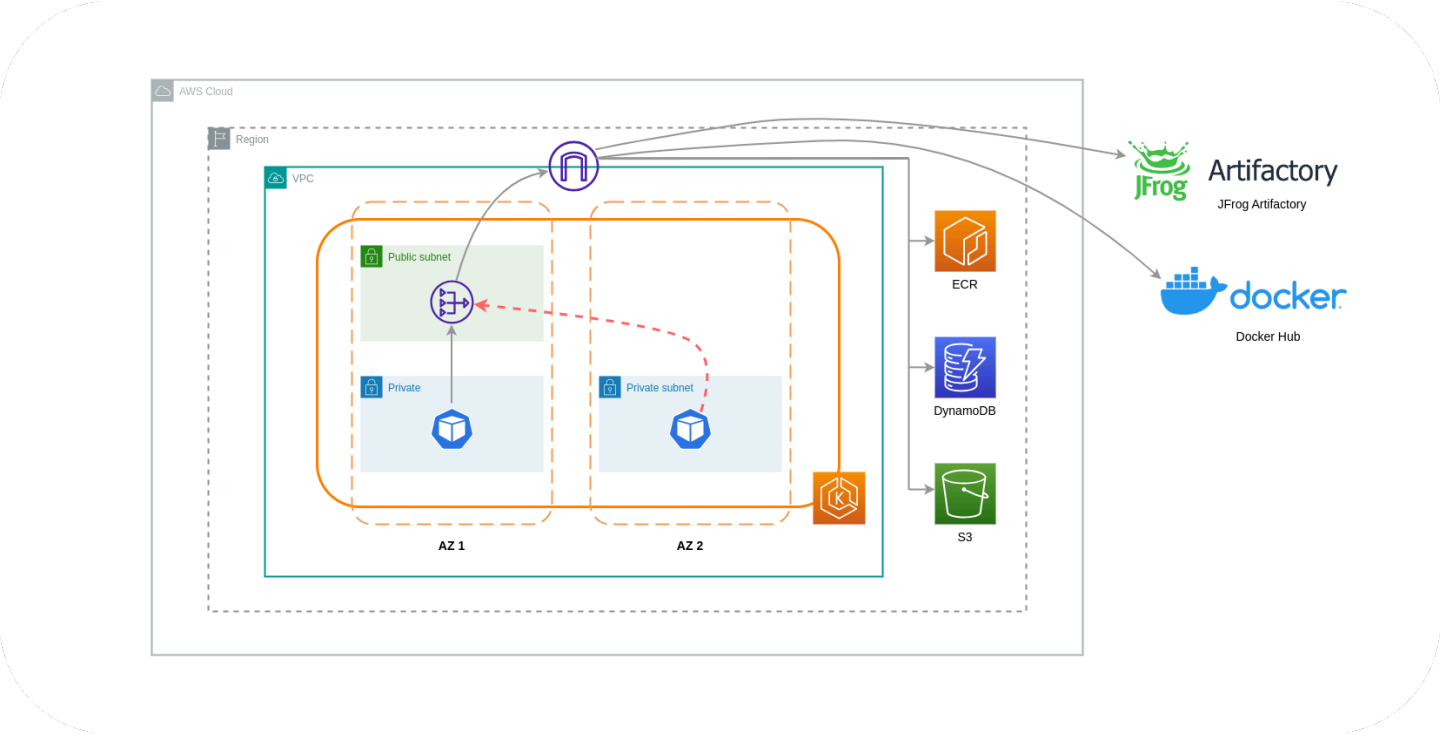
Wenn Sie Workloads mit besonders großen Images ausführen, können Sie Ihre eigenen benutzerdefinierten Amazon Machine Images (AMIs) mit vorab zwischengespeicherten Container-Images erstellen. Dadurch können die Zeit für den ersten Abruf von Images und die potenziellen Kosten für die Datenübertragung von einer Container-Registry zu den EKS-Worker-Knoten reduziert werden.

Datenübertragung ins Internet und zu AWS-Services

Es ist üblich, Kubernetes-Workloads über das Internet in andere AWS-Services oder Tools und Plattformen von Drittanbietern zu integrieren. Die zugrunde liegende Netzwerkinfrastruktur, die für die Weiterleitung des Datenverkehrs zum und vom jeweiligen Ziel verwendet wird, kann sich auf die Kosten auswirken, die beim Datenübertragungsprozess anfallen.

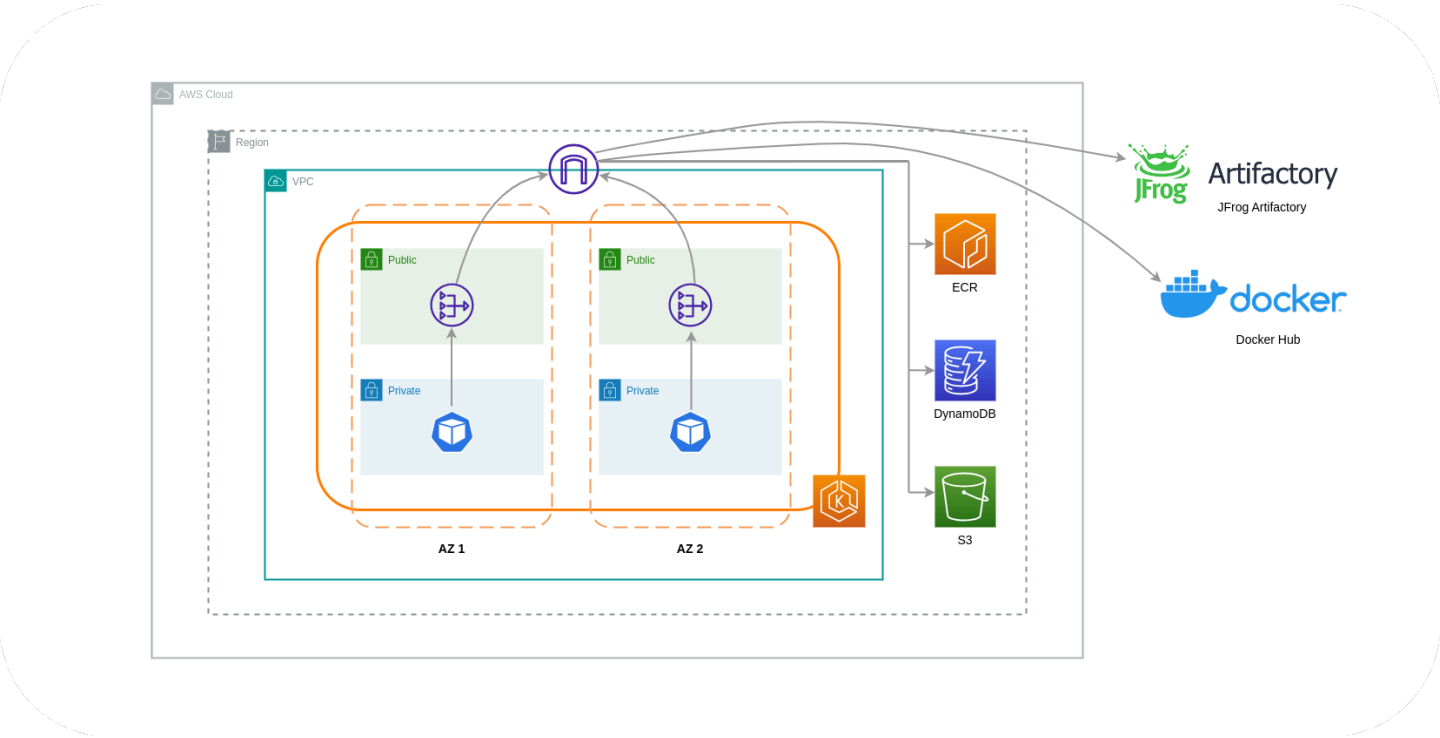
Verwendung von NAT-Gateways

NAT-Gateways sind Netzwerkkomponenten, die die Netzwerkadressübersetzung (NAT) durchführen. Das folgende Diagramm zeigt Pods in einem EKS-Cluster, die mit anderen AWS-Services (Amazon ECR, DynamoDB und S3) und Plattformen von Drittanbietern kommunizieren. In diesem Beispiel werden die Pods in separaten privaten Subnetzen ausgeführt. AZs Zum Senden und Empfangen von Datenverkehr aus dem Internet wird ein NAT-Gateway im öffentlichen Subnetz einer AZ bereitgestellt, sodass alle Ressourcen mit privaten IP-Adressen gemeinsam auf eine einzige öffentliche IP-Adresse zugreifen können, um auf das Internet zuzugreifen. Dieses NAT-Gateway kommuniziert wiederum mit der Internet-Gateway-Komponente, sodass Pakete an ihr endgültiges Ziel gesendet werden können.



Wenn Sie NAT-Gateways für solche Anwendungsfälle verwenden, können Sie die Datenübertragungskosten minimieren, indem Sie in jeder AZ ein NAT-Gateway bereitstellen. Auf diese Weise wird der an das Internet weitergeleitete Datenverkehr über das NAT-Gateway in derselben AZ geleitet, wodurch eine Datenübertragung zwischen den AZ vermieden wird. Sie sparen zwar die Kosten für die Datenübertragung zwischen den AZ-Datenbanken, aber diese Konfiguration hat zur Folge, dass Ihnen die Kosten für ein zusätzliches NAT-Gateway in Ihrer Architektur entstehen.

Dieser empfohlene Ansatz ist in der folgenden Abbildung dargestellt.



Verwenden eines VPC-Endpunkts

Um die Kosten in solchen Architekturen weiter zu senken, sollten Sie VPC-Endpunkte verwenden, um Konnektivität zwischen Ihren Workloads und AWS-Services herzustellen. Mit VPC-Endpunkten können Sie von einer VPC aus auf AWS-Services zugreifen, ohne dass data/network Pakete das Internet durchqueren. Der gesamte Datenverkehr ist intern und verbleibt innerhalb des AWS-Netzwerks. Es gibt zwei Arten von VPC-Endpunkten: Interface-VPC-Endpunkte (von vielen AWS-Services unterstützt) und Gateway-VPC-Endpunkte (nur von S3 und DynamoDB unterstützt).

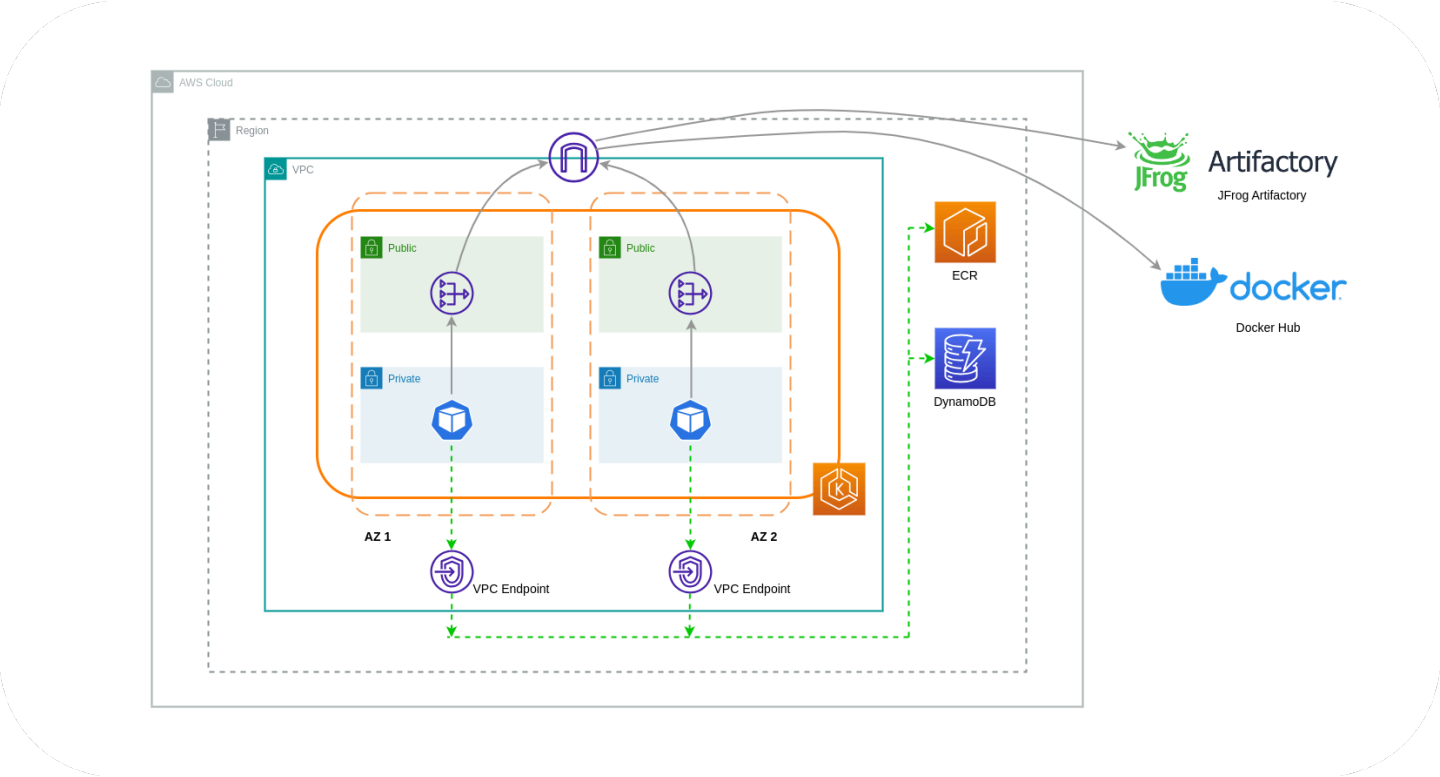
Gateway-VPC-Endpunkte

Im Zusammenhang mit Gateway-VPC-Endpunkten fallen keine Stunden- oder Datenübertragungskosten an. Bei der Verwendung von Gateway-VPC-Endpunkten ist zu beachten, dass sie nicht über VPC-Grenzen hinweg erweiterbar sind. Sie können nicht für VPC-Peering, VPN-Netzwerke oder über Direct Connect verwendet werden.

Schnittstelle VPC-Endpunkte

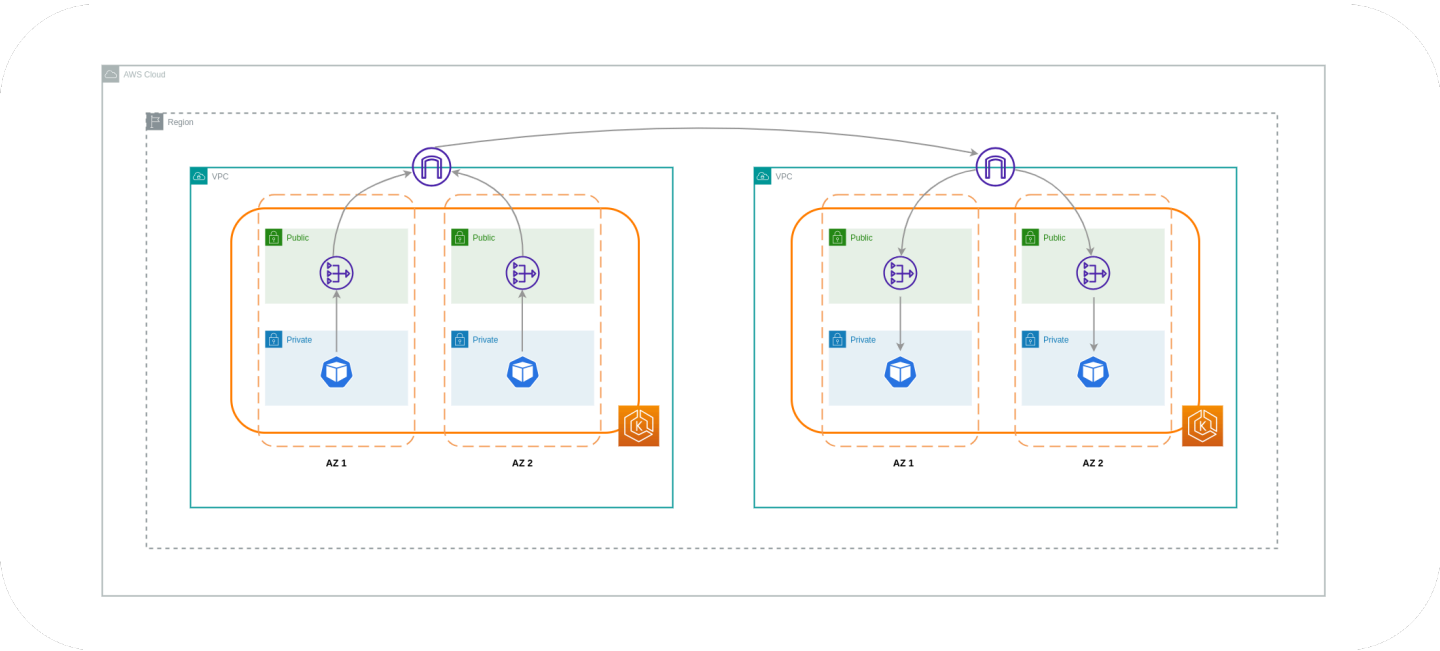
Für VPC-Endpunkte fällt eine stündliche Gebühr an. Für die Datenverarbeitung über das zugrunde liegende ENI fällt eine zusätzliche Gebühr an. Beachten Sie, dass die Datenübertragung zwischen den AZ-Datenbanken nicht in Rechnung gestellt wird.

Das folgende Diagramm zeigt Pods, die über VPC-Endpunkte mit AWS-Services kommunizieren.



Datenübertragung zwischen VPCs

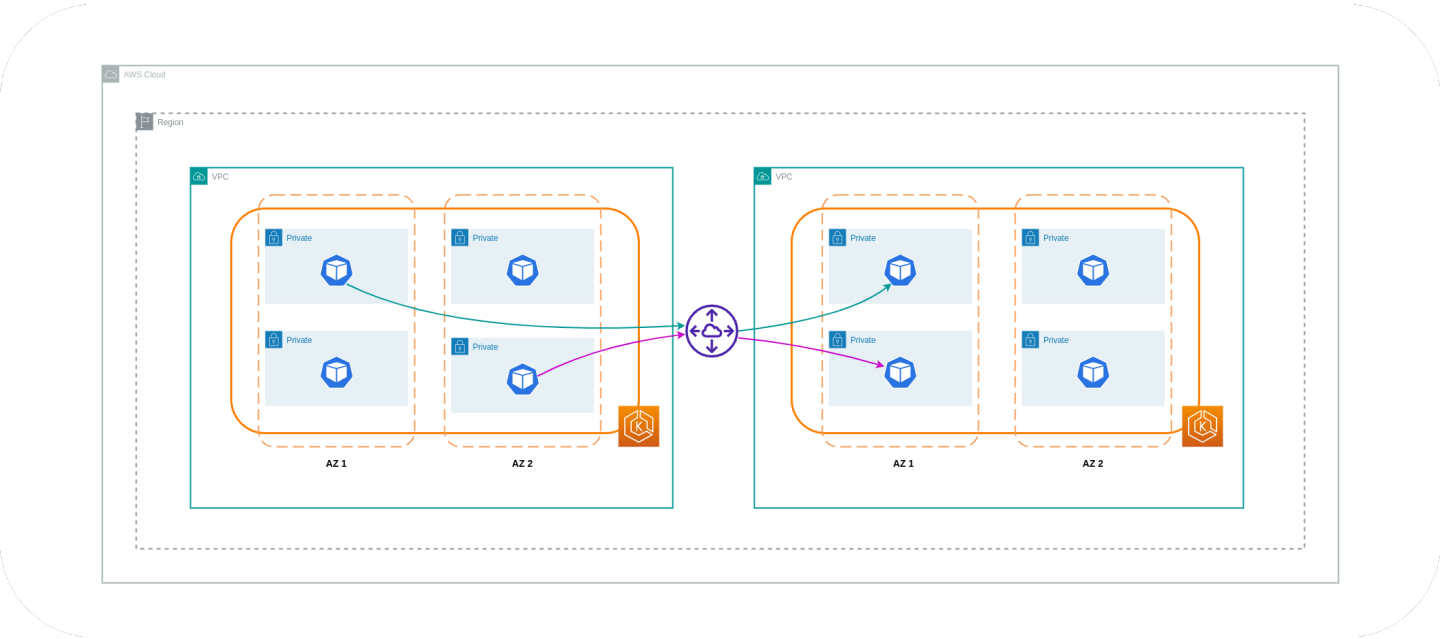
In einigen Fällen haben Sie möglicherweise Workloads in unterschiedlichen VPCs (innerhalb derselben AWS-Region), die miteinander kommunizieren müssen. Dies kann erreicht werden, indem der Datenverkehr über Internet-Gateways, die an die jeweiligen VPCs angeschlossen sind, das öffentliche Internet durchqueren kann. Eine solche Kommunikation kann durch den Einsatz von Infrastrukturkomponenten wie EC2-Instances, NAT-Gateways oder NAT-Instances in öffentlichen Subnetzen ermöglicht werden. Bei einem Setup, das diese Komponenten enthält, fallen jedoch Gebühren für ein- und processing/transferring ausgehende Daten an. VPCs Wenn der Verkehr zu und von der separaten VPCs Anlage übertragen wird AZs, fällt für die Datenübertragung eine zusätzliche Gebühr an. Das folgende Diagramm zeigt ein Setup, das NAT-Gateways und Internet-Gateways verwendet, um die Kommunikation zwischen Workloads in unterschiedlichen Umgebungen herzustellen. VPCs



VPC-Peering-Verbindungen

Um die Kosten für solche Anwendungsfälle zu senken, können Sie VPC Peering verwenden. Bei einer VPC-Peering-Verbindung fallen keine Datenübertragungsgebühren für Netzwerkverkehr an, der innerhalb derselben AZ bleibt. Wenn sich der Verkehr kreuzt AZs, fallen Kosten an. Dennoch wird der VPC-Peering-Ansatz für eine kostengünstige Kommunikation zwischen separaten Workloads VPCs innerhalb derselben AWS-Region empfohlen. Es ist jedoch wichtig zu beachten, dass VPC-Peering in erster Linie für 1:1 VPC-Konnektivität wirksam ist, da es keine transitiven Netzwerke ermöglicht.

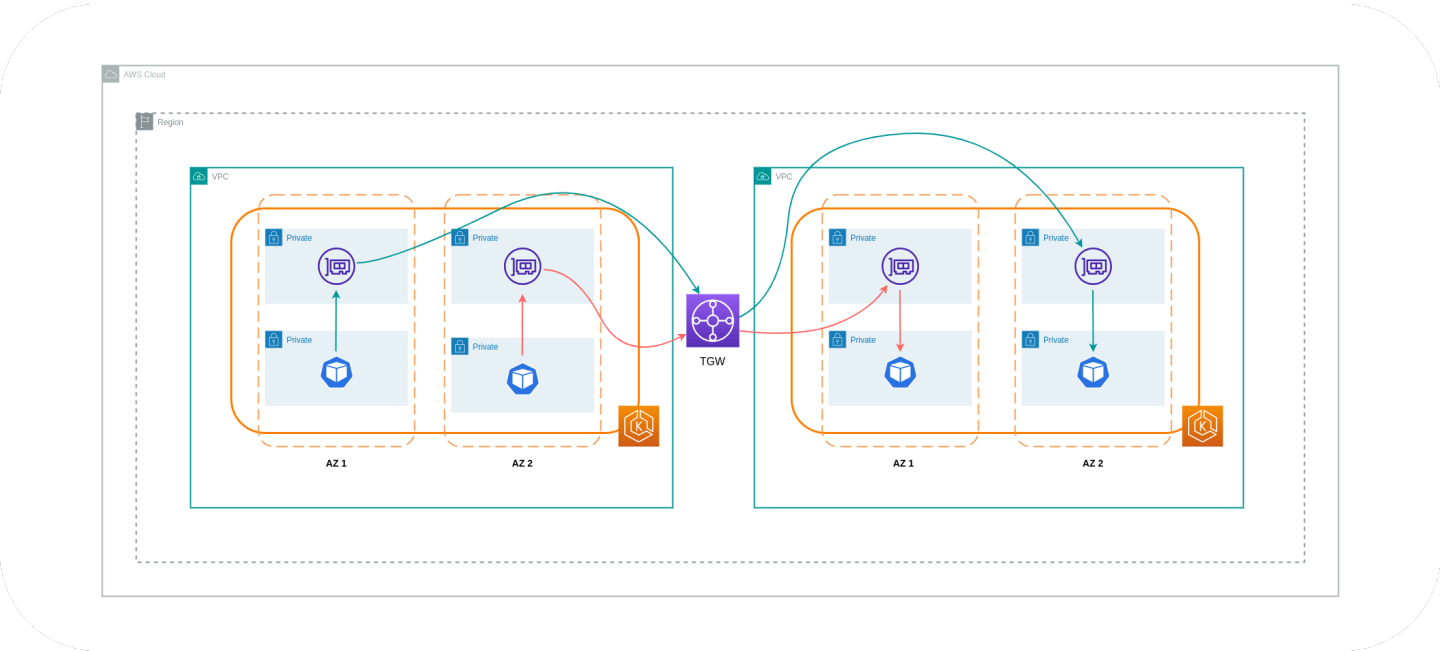
Das folgende Diagramm zeigt eine allgemeine Darstellung der Workload-Kommunikation über eine VPC-Peering-Verbindung.



Transitive Netzwerkverbindungen

Wie im vorherigen Abschnitt erwähnt, ermöglichen VPC-Peering-Verbindungen keine transitive Netzwerkkonnektivität. Wenn Sie 3 oder mehr Verbindungen VPCs mit transitiven Netzwerkanforderungen herstellen möchten, sollten Sie ein Transit Gateway (TGW) verwenden. Auf diese Weise können Sie die Grenzen des VPC-Peerings oder jeglichen betrieblichen Mehraufwand überwinden, der mit mehreren VPC-Peering-Verbindungen zwischen mehreren verbunden ist. VPCs Die an das TGW gesendeten Daten werden Ihnen auf Stundenbasis in Rechnung gestellt. Für den Inter-AZ-Verkehr, der über den TGW fließt, fallen keine Zielkosten an.

Das folgende Diagramm zeigt den Inter-AZ-Verkehr, der über ein TGW zwischen Workloads in verschiedenen, VPCs aber innerhalb derselben AWS-Region fließt.



Verwenden eines Service Mesh

Service Meshes bieten leistungsstarke Netzwerkfunktionen, mit denen Sie die Netzwerkkosten in Ihren EKS-Cluster-Umgebungen senken können. Sie sollten jedoch die betrieblichen Aufgaben und die Komplexität, die ein Service Mesh für Ihre Umgebung mit sich bringen wird, sorgfältig abwägen, wenn Sie eines einführen.

Beschränkung des Datenverkehrs auf Availability Zones

Verwendung der nach Lokalität gewichteten Verteilung von Istio

Mit Istio können Sie Netzwerkrichtlinien auf den Datenverkehr anwenden, nachdem das Routing erfolgt ist. Dies erfolgt mithilfe von Zielregeln wie z. B. der nach Lokalität gewichteten Verteilung. Mit dieser Funktion können Sie das Gewicht (ausgedrückt als Prozentsatz) des Verkehrs, der ein bestimmtes Ziel erreichen kann, anhand seines Ursprungs steuern. Die Quelle dieses Datenverkehrs kann entweder von einem externen (oder öffentlich zugänglichen) Load Balancer oder von einem Pod innerhalb des Clusters selbst stammen. Wenn alle Pod-Endpunkte verfügbar sind, wird der Standort auf der Grundlage eines gewichteten Round-Robin-Load-Balancing-Algorithmus ausgewählt. Falls bestimmte Endpunkte fehlerhaft oder nicht verfügbar sind, wird die Gewichtung der Lokalität automatisch angepasst, um diese Änderung an den verfügbaren Endpunkten widerzuspiegeln.

> **Note**
>
> Bevor Sie die nach Lokalität gewichtete Verteilung implementieren, sollten Sie sich zunächst mit Ihren Netzwerkverkehrsmustern und den Auswirkungen vertraut machen, die die Zielregelrichtlinie auf das Verhalten Ihrer Anwendung haben kann. Daher ist es wichtig, verteilte Verfolgungsmechanismen mit Tools wie AWS X-Ray oder Jaeger einzurichten.

Die oben aufgeführten Istio-Zielregeln können auch zur Verwaltung des Datenverkehrs von einem Load Balancer zu Pods in Ihrem EKS-Cluster angewendet werden. Lokalitätsgewichtete Verteilungsregeln können auf einen Dienst angewendet werden, der Datenverkehr von einem hochverfügbaren Load Balancer (insbesondere dem Ingress Gateway) empfängt. Mit diesen Regeln können Sie anhand seines zonalen Ursprungs — in diesem Fall dem Load Balancer — steuern, wie viel Verkehr wohin fließt. Bei richtiger Konfiguration fällt weniger ausgehender zonenübergreifender Datenverkehr an als bei einem Load Balancer, der den Datenverkehr gleichmäßig oder nach dem Zufallsprinzip auf Pod-Replikate in verschiedenen Pod-Replikaten verteilt. AZs

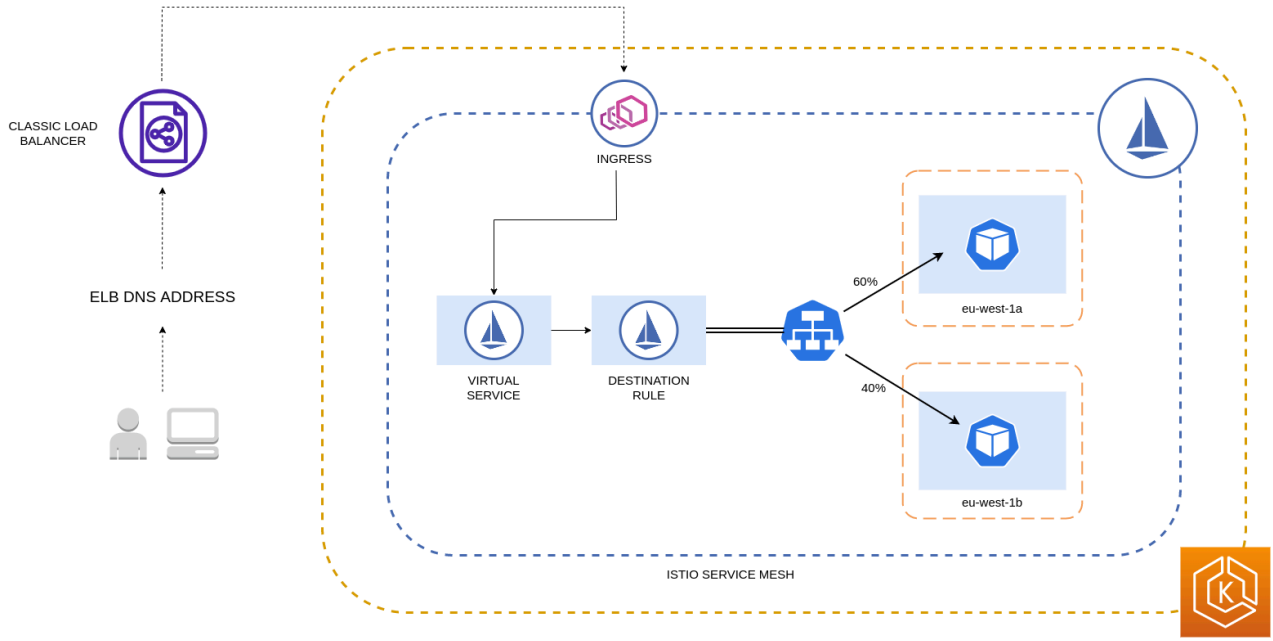
Im Folgenden finden Sie ein Codeblock-Beispiel für eine Zielregelressource in Istio. Wie unten zu sehen ist, spezifiziert diese Ressource gewichtete Konfigurationen für eingehenden Verkehr aus 3 verschiedenen Quellen AZs in der `eu-west-1` Region. Diese Konfigurationen legen fest, dass ein Großteil des eingehenden Datenverkehrs (in diesem Fall 70%) von einer bestimmten AZ an ein Ziel in derselben AZ weitergeleitet werden sollte, von der er stammt.

```
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**
>
> Das Mindestgewicht, das an ein Ziel verteilt werden kann, beträgt 1%. Der Grund dafür ist die Beibehaltung von Failover-Regionen und -Zonen für den Fall, dass die Endpunkte im Hauptziel fehlerhaft oder nicht verfügbar sind.

Das folgende Diagramm zeigt ein Szenario, in dem in der Region EU-West-1 ein hochverfügbarer Load-Balancer vorhanden ist und eine nach der Lokalität gewichtete Verteilung angewendet wird. Die Zielregelrichtlinie für dieses Diagramm ist so konfiguriert, dass 60% des Datenverkehrs von eu-west-1a an Pods in derselben AZ weitergeleitet werden, wohingegen 40% des Datenverkehrs von eu-west-1a an Pods in eu-west-1b weitergeleitet werden sollten.



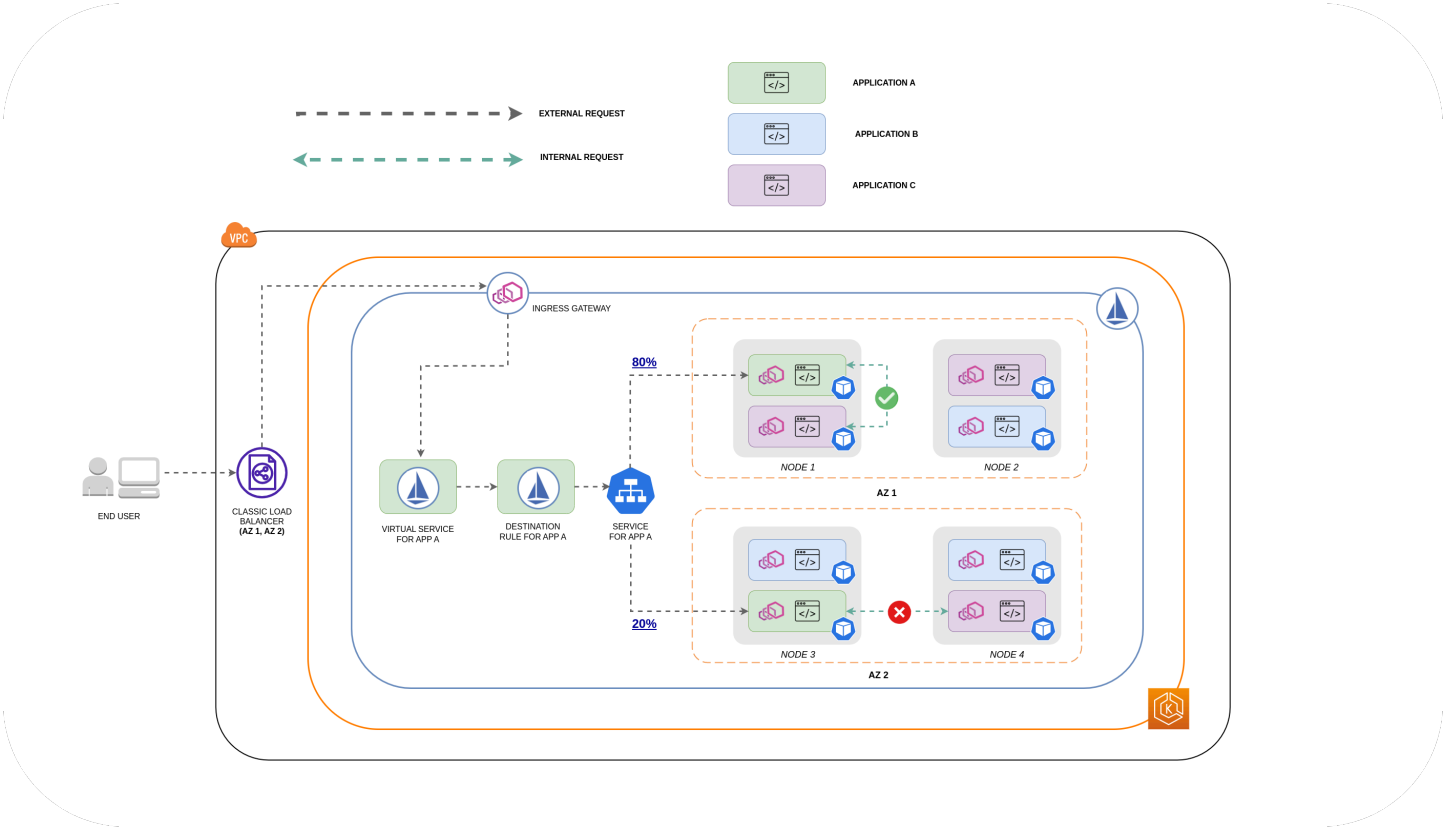
Beschränkung des Datenverkehrs auf Availability Zones und Nodes

Verwenden der internen Verkehrsrichtlinie des Dienstes mit Istio

Um die Netzwerkkosten im Zusammenhang mit eingehendem externem Datenverkehr und internem Verkehr zwischen Pods zu senken, können Sie die Zielregeln von Istio und die interne Verkehrsrichtlinie des Kubernetes-Service kombinieren. Die Art und Weise, wie die Zielregeln von Istio mit der internen Verkehrsrichtlinie des Dienstes kombiniert werden können, hängt weitgehend von drei Faktoren ab:

- Die Rolle der Microservices
- Muster des Netzwerkverkehrs zwischen den Microservices
- Wie sollten die Microservices in der gesamten Kubernetes-Cluster-Topologie bereitgestellt werden

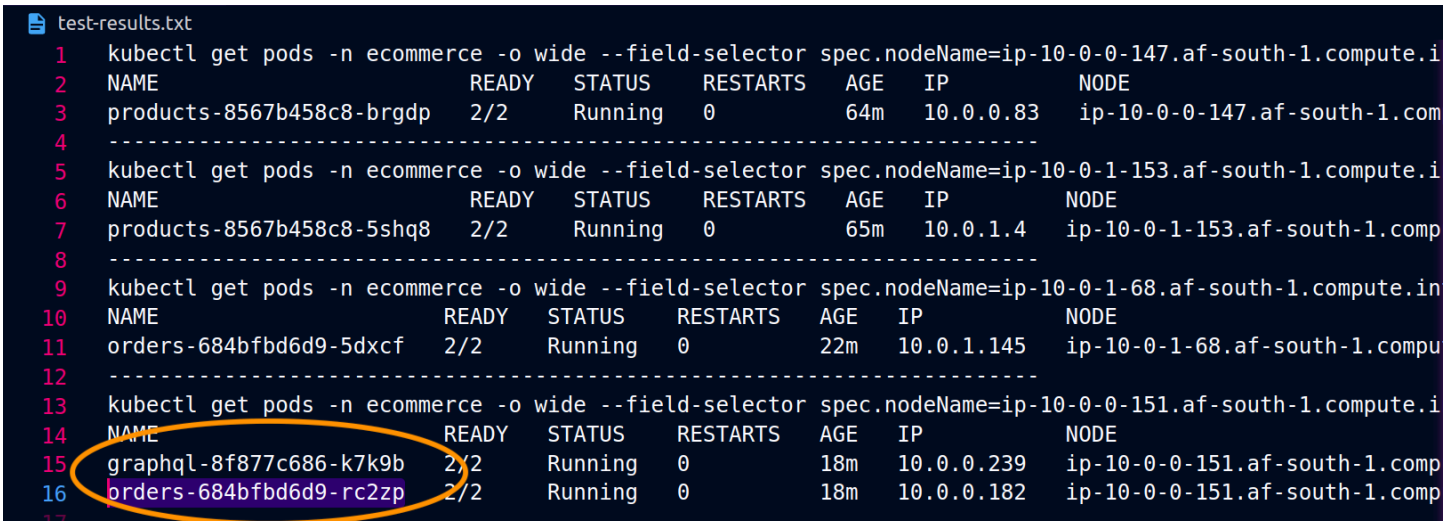
Das folgende Diagramm zeigt, wie der Netzwerkfluss im Fall einer verschachtelten Anfrage aussehen würde und wie die oben genannten Richtlinien den Datenverkehr steuern würden.

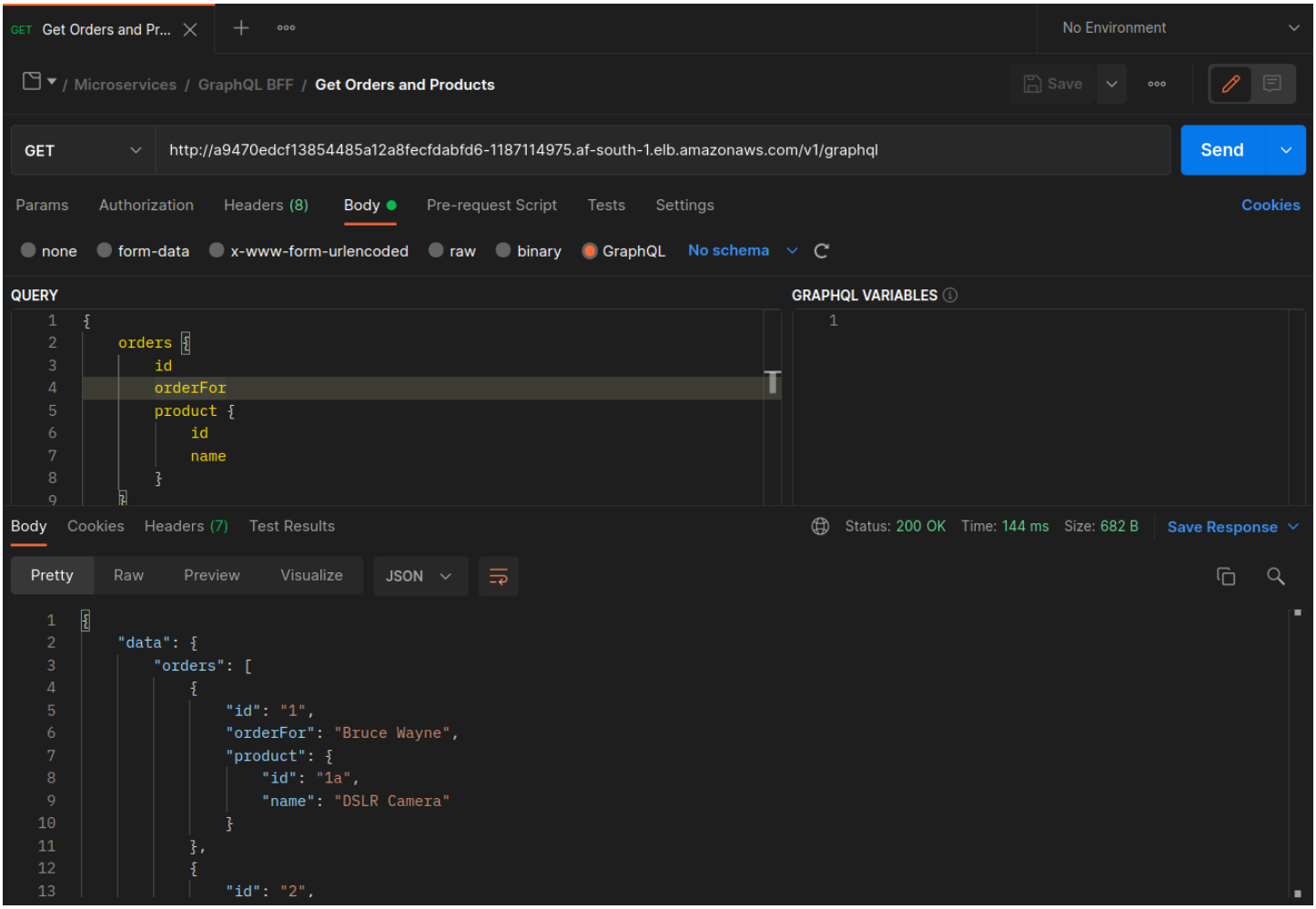


1. Der Endbenutzer stellt eine Anfrage an APP A, die wiederum eine verschachtelte Anfrage an APP C sendet. Diese Anfrage wird zunächst an einen hochverfügbaren Load Balancer gesendet, der über Instanzen in AZ 1 und AZ 2 verfügt, wie das obige Diagramm zeigt.
2. Die externe eingehende Anfrage wird dann vom Istio Virtual Service an das richtige Ziel weitergeleitet.
3. Nachdem die Anfrage weitergeleitet wurde, steuert die Istio-Zielregel, wie viel Verkehr an das jeweilige Objekt weitergeleitet wird, AZs je nachdem, woher er stammt (AZ 1 oder AZ 2).
4. Der Datenverkehr geht dann an den Service für APP A und wird dann per Proxy an die jeweiligen Pod-Endpunkte weitergeleitet. Wie im Diagramm dargestellt, werden 80% des eingehenden Datenverkehrs an Pod-Endpunkte in AZ 1 und 20% des eingehenden Datenverkehrs an AZ 2 gesendet.
5. APP A stellt dann eine interne Anfrage an APP C. Für den Dienst von APP C ist eine interne Verkehrsrichtlinie aktiviert (`internalTrafficPolicy`: Lokal`).
6. Die interne Anfrage von APP A (auf KNOTEN 1) an APP C ist erfolgreich, da der knotenlokale Endpunkt für APP C verfügbar ist.

7. Die interne Anfrage von APP A (auf KNOTEN 3) an APP C schlägt fehl, da keine knotenlokalen Endpunkte für APP C verfügbar sind. Wie das Diagramm zeigt, hat APP C keine Replikate auf NODE 3. *

Die folgenden Screenshots wurden anhand eines Live-Beispiels für diesen Ansatz aufgenommen. Die ersten Screenshots zeigen eine erfolgreiche externe Anfrage an einen `graphql` und eine erfolgreiche verschachtelte Anfrage von der `graphql` an ein `orders` Replikat auf dem Knoten. `ip-10-0-0-151.af-south-1.compute.internal`





Mit Istio können Sie die Statistiken aller Upstream-Cluster und Endpoints überprüfen und exportieren, die Ihren Proxys bekannt sind. Auf diese Weise können Sie sich ein Bild vom Netzwerkfluss und vom Verteilungsanteil der Dienste eines Workloads machen. Um mit demselben Beispiel fortzufahren: Die `orders` Endpunkte, die dem `graphql` Proxy bekannt sind, können mit dem folgenden Befehl abgerufen werden:

```
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**
```

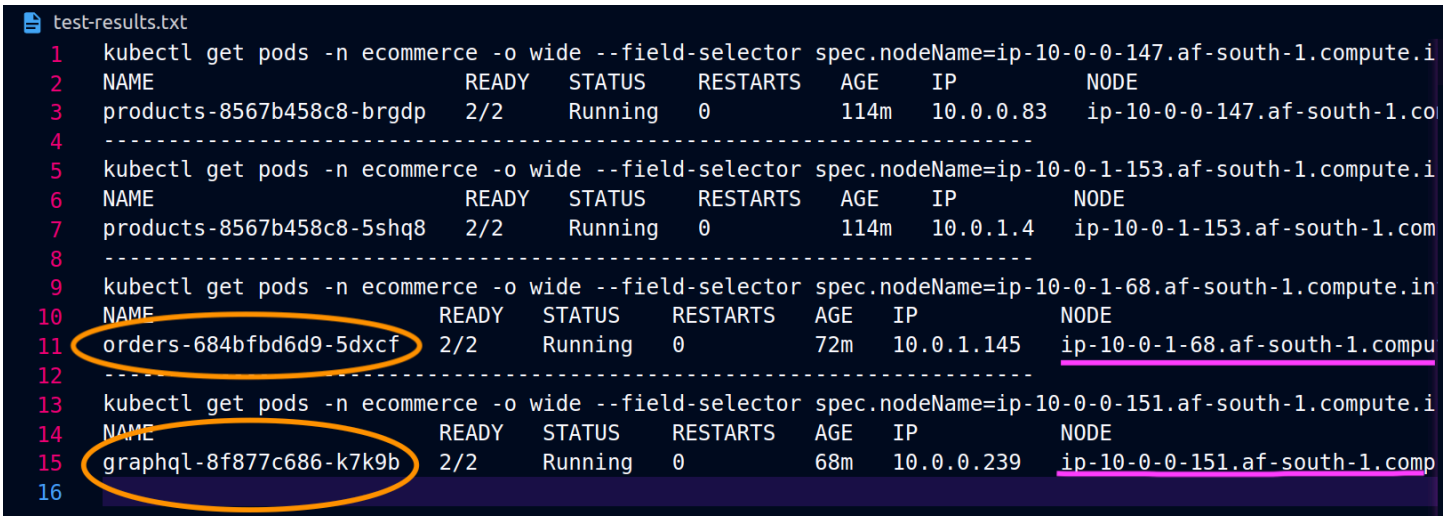
```
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**zone::af-south-1b**
...
```

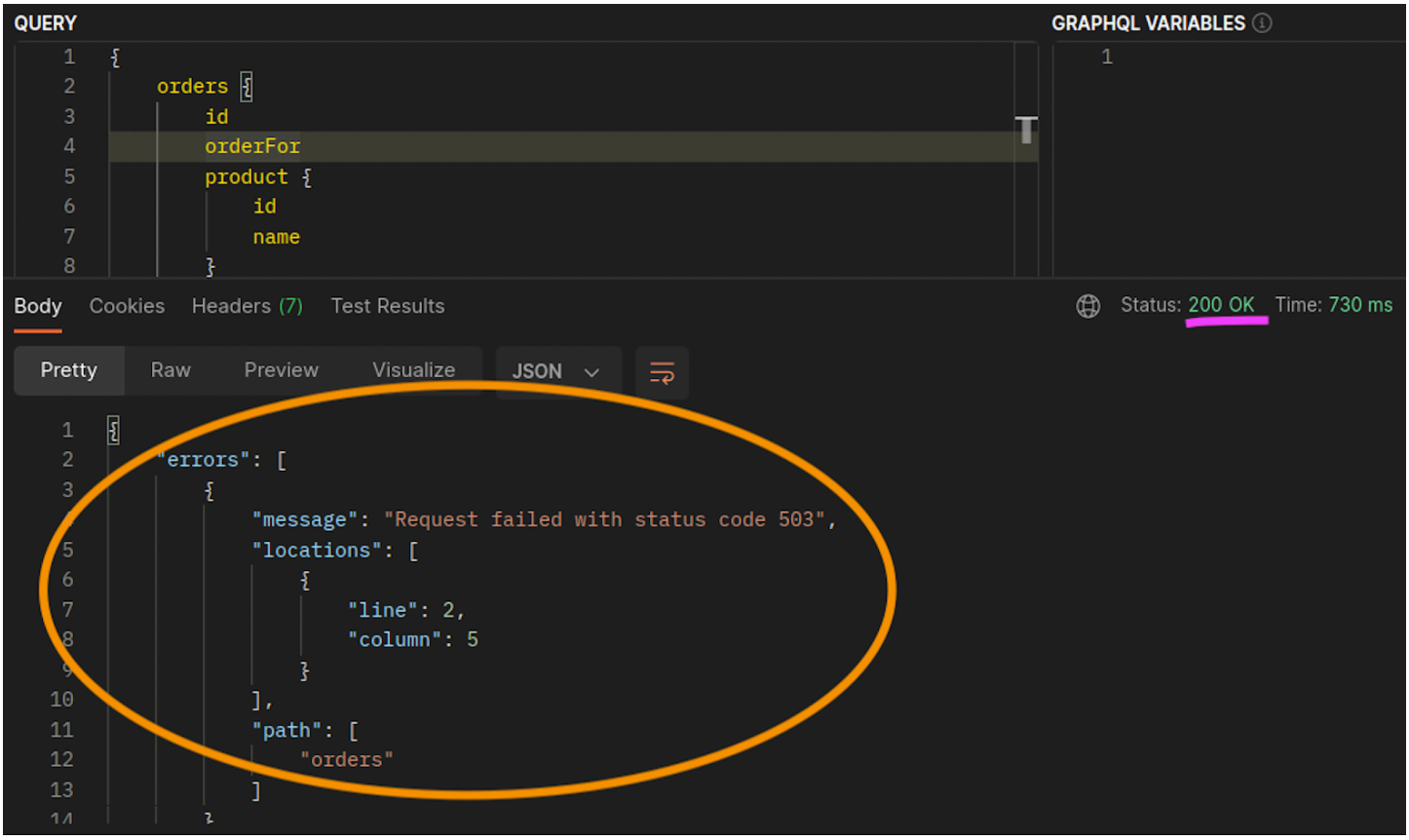
In diesem Fall kennt der `graphql` Proxy nur den `orders` Endpunkt für das Replikat, mit dem er sich einen Knoten teilt. Wenn Sie die `internalTrafficPolicy: Local` Einstellung aus dem Orders Service entfernen und einen Befehl wie den obigen erneut ausführen, geben die Ergebnisse alle Endpunkte der Replikate zurück, die auf die verschiedenen Knoten verteilt sind. Wenn Sie sich die `rq_total` für die jeweiligen Endpunkte ansehen, werden Sie außerdem feststellen, dass die Netzwerkverteilung relativ gleichmäßig verteilt ist. Wenn die Endpunkte also mit Upstream-Diensten verknüpft sind, die in unterschiedlichen Zonen ausgeführt werden AZs, führt diese Verteilung des Netzwerks auf mehrere Zonen zu höheren Kosten.

Wie in einem vorherigen Abschnitt oben erwähnt, können Sie Pods, die häufig kommunizieren, gemeinsam platzieren, indem Sie die Pod-Affinität nutzen.

```
...
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
...
```

Wenn die `orders` Replikate `graphql` und die Replikate nicht gleichzeitig auf demselben Knoten (`ip-10-0-0-151.af-south-1.compute.internal`) existieren, `graphql` ist die erste Anfrage an erfolgreich, wie `200 response code` im Postman-Screenshot unten angegeben, wohingegen die zweite verschachtelte Anfrage von an mit a fehlschlägt. `graphql orders 503 response code`





Weitere Ressourcen

- Behebung von Latenz- und Datenübertragungskosten auf EKS mithilfe von Istio
- Untersuchung der Auswirkungen von Topology Aware Hints auf den Netzwerkverkehr in Amazon Elastic Kubernetes Service
- Erhalten Sie Einblick in Ihre AZ-übergreifenden Pod-zu-Pod-Netzwerk-Bytes in Amazon EKS
- Optimieren Sie den AZ-Verkehr mit Istio
- Optimieren Sie den AZ-Verkehr mit topologiebewusstem Routing
- Optimieren Sie die Kosten und Leistung von Kubernetes mit der internen Traffic-Richtlinie für den Service
- Optimieren Sie die Kosten und Leistung von Kubernetes mit Istio und Service Internal Traffic Policy
- Überblick über die Datenübertragungskosten für gängige Architekturen
- Grundlegendes zu den Datenübertragungskosten für AWS-Container-Services

Speicher

Übersicht

Es gibt Szenarien, in denen Sie möglicherweise Anwendungen ausführen möchten, die Daten kurz- oder langfristig aufbewahren müssen. Für solche Anwendungsfälle können Volumes von Pods definiert und bereitgestellt werden, sodass ihre Container unterschiedliche Speichermechanismen nutzen können. Kubernetes unterstützt verschiedene Arten von Volumes für kurzlebige und persistente Speicherung. Die Wahl des Speichers hängt weitgehend von den Anwendungsanforderungen ab. Jeder Ansatz hat Auswirkungen auf die Kosten, und die unten aufgeführten Methoden helfen Ihnen dabei, die Kosteneffizienz für Workloads zu erreichen, die in Ihren EKS-Umgebungen irgendeine Form von Speicher benötigen.

Kurzlebige Volumen

Ephemere Volumes eignen sich für Anwendungen, die vorübergehende lokale Volumes benötigen, bei denen jedoch keine Daten nach Neustarts dauerhaft gespeichert werden müssen. Beispiele hierfür sind Anforderungen an Scratch-Speicherplatz, Zwischenspeicherung und schreibgeschützte Eingabedaten wie Konfigurationsdaten und geheime Daten. Weitere Informationen zu kurzlebigen Kubernetes-Volumes finden Sie hier. Die meisten kurzlebigen Volumes (z. B. emptyDir, configMap,

downwardAPI, secret, hostpath) werden von lokal angeschlossenen beschreibbaren Geräten (normalerweise der Root-Disk) oder RAM unterstützt. Daher ist es wichtig, das kostengünstigste und leistungsstärkste Host-Volume zu wählen.

EBS-Volumes verwenden

Wir empfehlen, mit gp3 als Host-Root-Volume zu beginnen. Es ist das neueste Allzweck-SSD-Volume, das von Amazon EBS angeboten wird, und bietet im Vergleich zu GP2-Volumes auch einen niedrigeren Preis (bis zu 20%) pro GB.

Amazon EC2 Instance Stores verwenden

Amazon EC2 Instance Stores bieten temporären Speicher auf Blockebene für Ihre EC2 Instances. Auf den von EC2 Instance-Speichern bereitgestellten Speicher kann über Festplatten zugegriffen werden, die physisch an die Hosts angeschlossen sind. Im Gegensatz zu Amazon EBS können Sie Instance-Speicher-Volumes nur anhängen, wenn die Instance gestartet wird, und diese Volumes existieren nur während der Lebensdauer der Instance. Sie können nicht getrennt und wieder an andere Instances angehängt werden. Sie können hier mehr über Amazon EC2 Instance Stores erfahren. Für das Volumen eines Instance-Speichers fallen keine zusätzlichen Gebühren an. Dadurch sind sie (Instance-Speicher-Volumes) kostengünstiger als allgemeine EC2 Instances mit großen EBS-Volumes.

Um lokale Speichervolumes in Kubernetes zu verwenden, sollten Sie die Festplatten anhand der EC2 Amazon-Benutzerdaten partitionieren, konfigurieren und formatieren, sodass Volumes wie HostPathin der Pod-Spezifikation bereitgestellt werden können. Alternativ können Sie den Local Persistent Volume Static Provisioner nutzen, um die lokale Speicherverwaltung zu vereinfachen. Mit dem statischen Local Persistent Volume Provisioner können Sie über die Standardschnittstelle von Kubernetes PersistentVolumeClaim (PVC) auf lokale Instance-Speicher-Volumes zugreifen. Darüber hinaus wird das Paket provision PersistentVolumes (PVs) bereitgestellt, das Informationen zur Knotenaffinität enthält, um Pods den richtigen Knoten zuzuordnen. Obwohl Kubernetes verwendet wird PersistentVolumes, sind EC2 Instance-Speicher-Volumes ihrer Natur nach kurzlebig. Daten, die auf kurzlebige Festplatten geschrieben wurden, sind nur während der Lebensdauer der Instanz verfügbar. Wenn die Instanz beendet wird, werden auch die Daten beendet. Weitere Informationen finden Sie in diesem Blog.

Beachten Sie, dass bei der Verwendung von Amazon EC2 Instance Store-Volumes das gesamte IOPS-Limit mit dem Host geteilt wird und Pods an einen bestimmten Host gebunden werden. Sie sollten Ihre Workload-Anforderungen gründlich prüfen, bevor Sie Amazon EC2 Instance Store-Volumes einführen.

Persistente Volumes

Kubernetes wird in der Regel mit der Ausführung von zustandslosen Anwendungen in Verbindung gebracht. Es gibt jedoch Szenarien, in denen Sie möglicherweise Microservices ausführen möchten, die persistente Daten oder Informationen von einer Anfrage zur nächsten beibehalten müssen. Datenbanken sind ein gängiges Beispiel für solche Anwendungsfälle. Pods und die darin enthaltenen Behälter oder Prozesse sind jedoch kurzlebig. Um Daten über die Lebensdauer eines Pods hinaus beizubehalten, können Sie PVs damit den Zugriff auf den Speicher an einem bestimmten Ort definieren, der unabhängig vom Pod ist. Die damit PVs verbundenen Kosten hängen stark von der Art des verwendeten Speichers und der Art der Nutzung durch Anwendungen ab.

Hier sind verschiedene Arten von Speicheroptionen aufgeführt, die Kubernetes PVs auf Amazon EKS unterstützen. Die unten beschriebenen Speicheroptionen sind Amazon EBS, Amazon EFS, Amazon FSx for Lustre, Amazon FSx for NetApp ONTAP.

Volumen im Amazon Elastic Block Store (EBS)

Amazon EBS-Volumes können als Kubernetes genutzt werden, PVs um Speichervolumes auf Blockebene bereitzustellen. Diese eignen sich gut für Datenbanken, die auf zufälligen Lese- und Schreibvorgängen basieren, sowie für durchsatzintensive Anwendungen, die lange, kontinuierliche Lese- und Schreibvorgänge ausführen. Der Amazon Elastic Block Store Container Storage Interface (CSI) -Treiber ermöglicht es Amazon EKS-Clustern, den Lebenszyklus von Amazon EBS-Volumes für persistente Volumes zu verwalten. Das Container Storage Interface ermöglicht und erleichtert die Interaktion zwischen Kubernetes und einem Speichersystem. Wenn ein CSI-Treiber in Ihrem EKS-Cluster bereitgestellt wird, können Sie über die nativen Kubernetes-Speicherressourcen wie Persistent Volumes (PVs), Persistent Volume Claims () und Storage Classes (PVCs) auf seine Funktionen zugreifen. SCs Dieser Link enthält praktische Beispiele für die Interaktion mit Amazon EBS-Volumes mit dem Amazon EBS CSI-Treiber.

Auswahl des richtigen Volumes

Wir empfehlen die Verwendung der neuesten Generation von Blockspeicher (GP3), da diese das richtige Gleichgewicht zwischen Preis und Leistung bietet. Außerdem können Sie damit Volumen-IOPS und Durchsatz unabhängig von der Volume-Größe skalieren, ohne zusätzliche Blockspeicherkapazität bereitstellen zu müssen. Wenn Sie derzeit GP2-Volumes verwenden, empfehlen wir dringend, auf GP3-Volumes zu migrieren. Der Blogbeitrag Migration von Amazon EKS-Clustern von GP2- zu GP3-EBS-Volumes erklärt, wie Sie mit der Funktion CSI Volume Snapshots von GP2 auf GP3 auf Amazon EKS-Clustern mit Sicherung und Wiederherstellung migrieren, was Anwendungsausfälle erfordert.

Amazon EBS ermöglicht die Online-Änderung von Volumeneigenschaften wie Volumengröße, IOPS und Durchsatz. Mit dieser Funktion kann man ohne Anwendungsausfälle von gp2 auf gp3 migrieren, indem man entweder PVC-Anmerkungen verwendet, wie in diesem Blog beschrieben, was den EBS-CSI-Treiber v1.19.0+ erfordert, oder ab Amazon EKS v1.31 und EBS CSI-Treiber 1.35, indem man die hier beschriebene API verwendet. VolumeAttributesClass

Wenn Sie über Anwendungen verfügen, die eine höhere Leistung erfordern und Volumen benötigen, die größer sind als das, was ein einzelnes GP3-Volume unterstützen kann, sollten Sie die Verwendung von io2 Block Express in Betracht ziehen. Dieser Speichertyp ist ideal für Ihre größte, I/O intensivste und geschäftskritischste Bereitstellung wie SAP HANA oder andere große Datenbanken mit geringen Latenzanforderungen. Beachten Sie, dass die EBS-Leistung einer Instance durch die Leistungsgrenzen der Instance begrenzt ist, sodass nicht alle Instances io2 Block Express-Volumes unterstützen. In diesem Dokument finden Sie Informationen zu den unterstützten Instance-Typen und anderen Überlegungen.

Ein einzelnes GP3-Volume kann bis zu 16.000 maximale IOPS, einen maximalen Durchsatz von 1.000 und MiB/s maximal 16 TiB unterstützen. Die neueste Generation des bereitgestellten IOPS-SSD-Volumes, das bis zu 256.000 IOPS, 4.000 MiB/s Durchsatz und 64 TiB bietet.

Unter diesen Optionen sollten Sie Ihre Speicherleistung und Ihre Kosten am besten an die Anforderungen Ihrer Anwendungen anpassen.

Überwachen und optimieren Sie im Laufe der Zeit

Es ist wichtig, die Ausgangsleistung Ihrer Anwendung zu verstehen und sie für ausgewählte Volumes zu überwachen, um zu überprüfen, ob sie Ihren Anforderungen entspricht requirements/expectations oder ob sie überdimensioniert ist (z. B. ein Szenario, in dem bereitgestellte IOPS nicht vollständig genutzt werden).

Anstatt von Anfang an ein großes Volumen zuzuweisen, können Sie die Größe des Volumes schrittweise erhöhen, während Sie Daten ansammeln. Mithilfe der Funktion zur Volumengrößenänderung im Amazon Elastic Block Store CSI-Treiber (aws-ebs-csi-driver) können Sie die Größe von Volumes dynamisch ändern. Beachten Sie, dass Sie nur die Größe des EBS-Volumes erhöhen können.

Um hängengebliebene EBS-Volumes zu identifizieren und zu entfernen, können Sie die Kostenoptimierungskategorie von AWS Trusted Advisor verwenden. Diese Funktion hilft Ihnen dabei, nicht angehängte Volumes oder Volumes mit sehr geringer Schreibaktivität für einen bestimmten Zeitraum zu identifizieren. Es gibt ein cloudnatives, schreibgeschütztes Open-Source-Tool namens

Popeye, das Kubernetes-Cluster in Echtzeit scannt und potenzielle Probleme mit bereitgestellten Ressourcen und Konfigurationen meldet. Es kann beispielsweise nach ungenutzten Dateien suchen PVs PVCs und prüfen, ob sie gebunden sind oder ob ein Fehler beim Einbinden des Volumes vorliegt.

Ausführliche Informationen zur Überwachung finden Sie im EKS-Leitfaden zur Kostenoptimierung zur Beobachtbarkeit.

Eine weitere Option, die Sie in Betracht ziehen können, sind die Amazon EBS-Volumenempfehlungen von AWS Compute Optimizer. Dieses Tool identifiziert automatisch die optimale Volume-Konfiguration und das richtige erforderliche Leistungsniveau. Es kann beispielsweise für optimale Einstellungen in Bezug auf bereitgestellte IOPS, Volume-Größen und Typen von EBS-Volumes verwendet werden, basierend auf der maximalen Auslastung in den letzten 14 Tagen. Es quantifiziert auch die potenziellen monatlichen Kosteneinsparungen, die sich aus den Empfehlungen ergeben. Weitere Informationen finden Sie in diesem Blog.

Richtlinie zur Aufbewahrung von Backup

Sie können die Daten auf Ihren Amazon EBS-Volumes sichern, indem Sie point-in-time Snapshots erstellen. Der Amazon EBS CSI-Treiber unterstützt Volume-Snapshots. Mithilfe der hier beschriebenen Schritte erfahren Sie, wie Sie einen Snapshot erstellen und einen EBS-PV wiederherstellen.

Nachfolgende Snapshots sind inkrementelle Backups, d. h., dass nur die Blöcke auf dem Gerät gespeichert werden, die sich nach Ihrem letzten Snapshot geändert haben. Hierdurch wird die zum Erstellen des Snapshots erforderliche Zeit verringert und es werden Speicherkosten eingespart, weil keine Datenduplikate angelegt werden. Die zunehmende Anzahl alter EBS-Snapshots ohne angemessene Aufbewahrungsrichtlinien kann jedoch zu unerwarteten Kosten führen, wenn der Betrieb in großem Umfang erfolgt. Wenn Sie Amazon EBS-Volumes direkt über die AWS-API sichern, können Sie Amazon Data Lifecycle Manager (DLM) nutzen, der eine automatisierte, richtlinienbasierte Lifecycle-Management-Lösung für Amazon Elastic Block Store (EBS) -Snapshots und EBS-gestützte Amazon Machine Images () bietet. AMIs Die Konsole erleichtert die Automatisierung der Erstellung, Aufbewahrung und Löschung von EBS-Snapshots und. AMIs

> **Note**
>
> Derzeit gibt es keine Möglichkeit, Amazon DLM über den Amazon EBS CSI-Treiber zu nutzen.

In einer Kubernetes-Umgebung können Sie ein Open-Source-Tool namens Velero nutzen, um Ihre EBS Persistent Volumes zu sichern. Sie können ein TTL-Flag setzen, wenn Sie den Job so planen, dass Backups ablaufen. Hier ist ein Leitfaden von Velero als Beispiel.

Amazon Elastic File System (EFS)

Amazon Elastic File System (EFS) ist ein serverloses, vollständig elastisches Dateisystem, mit dem Sie Dateidaten über eine standardmäßige Dateisystemschnittstelle und Dateisystemsemantik für ein breites Spektrum von Workloads und Anwendungen gemeinsam nutzen können. Beispiele für Workloads und Anwendungen sind Wordpress und Drupal, Entwicklertools wie JIRA und Git und gemeinsam genutzte Notebook-Systeme wie Jupyter sowie Home-Verzeichnisse.

Einer der Hauptvorteile von Amazon EFS besteht darin, dass es von mehreren Containern bereitgestellt werden kann, die auf mehrere Knoten und mehrere Availability Zones verteilt sind. Ein weiterer Vorteil besteht darin, dass Sie nur für den Speicherplatz zahlen, den Sie tatsächlich nutzen. EFS-Dateisysteme werden beim Hinzufügen und Entfernen von Dateien automatisch vergrößert und verkleinert, sodass keine Kapazitätsplanung erforderlich ist.

Um Amazon EFS in Kubernetes zu verwenden, müssen Sie den Amazon Elastic File System Container Storage Interface (CSI) -Treiber verwenden. aws-efs-csi-driver Derzeit kann der Treiber Zugriffspunkte dynamisch erstellen. Das Amazon EFS-Dateisystem muss jedoch zuerst bereitgestellt und als Eingabe für den Kubernetes-Speicherklassenparameter bereitgestellt werden.

Auswahl der richtigen EFS-Speicherklasse

Amazon EFS bietet vier Speicherklassen.

Zwei Standard-Speicherklassen:

- Amazon EFS-Standard
- Amazon EFS Standard — Seltener Zugriff (EFS Standard-IA)

Zwei Speicherklassen für eine Zone:

- Amazon EFS, eine Zone
- Amazon EFS One Zone — Seltener Zugriff (EFS One Zone-IA)

Die Speicherklassen Infrequent Access (IA) sind kostenoptimiert für Dateien, auf die nicht täglich zugegriffen wird. Mit Amazon EFS Lifecycle Management können Sie Dateien, auf die während der

Dauer der Lebenszyklusrichtlinie (7, 14, 30, 60 oder 90 Tage) nicht zugegriffen wurde, in die IA-Speicherklassen verschieben, wodurch die Speicherkosten im Vergleich zu den Speicherklassen EFS Standard und EFS One Zone um bis zu 92 Prozent gesenkt werden können.

Mit EFS Intelligent-Tiering überwacht das Lebenszyklusmanagement die Zugriffsmuster Ihres Dateisystems und verschiebt Dateien automatisch in die optimalste Speicherklasse.

> **Note**
>
> aws-efs-csi-driver hat derzeit keine Kontrolle über die Änderung der Speicherklassen, das Lebenszyklusmanagement oder Intelligent-Tiering. Diese sollten manuell in der AWS-Konsole oder über EFS eingerichtet APIs werden.

> **Note**
>
> aws-efs-csi-driver ist nicht mit Windows-basierten Container-Images kompatibel.

> **Note**
>
> Es liegt ein bekanntes Speicherproblem vor, wenn vol-metrics-opt-in(zur Ausgabe von Volumenmetriken) aktiviert ist, da die DiskUsageFunktion eine Speichermenge verbraucht, die proportional zur Größe Ihres Dateisystems ist. Derzeit empfehlen wir, die Option `--vol-metrics-opt-in `auf großen Dateisystemen zu deaktivieren, um zu vermeiden, dass zu viel Speicher verbraucht wird. Hier ist ein Link zu einem Github-Problem für weitere Informationen.

Amazon FSx für Lustre

Lustre ist ein leistungsstarkes paralleles Dateisystem, das häufig in Workloads verwendet wird, die einen Durchsatz von bis zu Hunderten von GB/s und unter einer Millisekunde pro Vorgang erfordern. Es wird für Szenarien wie maschinelles Lernen, Finanzmodellierung, HPC und Videoverarbeitung verwendet. Amazon FSx for Lustre bietet einen vollständig verwalteten gemeinsamen Speicher mit Skalierbarkeit und Leistung, der nahtlos in Amazon S3 integriert ist.

Sie können persistente Kubernetes-Speichervolumes verwenden, die von for Lustre unterstützt werden, indem Sie den FSx FSx for Lustre CSI-Treiber von Amazon EKS oder Ihren

selbstverwalteten Kubernetes-Cluster auf AWS verwenden. Weitere Informationen und Beispiele finden Sie in der Amazon EKS-Dokumentation.

Link zu Amazon S3

Es wird empfohlen, ein äußerst beständiges Langzeitdaten-Repository auf Amazon S3 mit Ihrem FSx for Lustre-Dateisystem zu verknüpfen. Nach der Verknüpfung werden große Datensätze nach Bedarf verzögert von Amazon S3 in die FSx for Lustre-Dateisysteme geladen. Sie können Ihre Analysen und Ergebnisse auch wieder in S3 ausführen und dann Ihr Lustre-Dateisystem löschen.

Auswahl der richtigen Bereitstellungs- und Speicheroptionen

FSx for Lustre bietet verschiedene Bereitstellungsoptionen. Die erste Option heißt Scratch und repliziert keine Daten, während die zweite Option als persistent bezeichnet wird, was, wie der Name schon sagt, Daten persistiert.

Die erste Option (Scratch) kann verwendet werden, um die Kosten für die temporäre Datenverarbeitung mit kürzerer Laufzeit zu senken. Die persistente Bereitstellungsoption ist für eine längerfristige Speicherung konzipiert, bei der Daten innerhalb einer AWS Availability Zone automatisch repliziert werden. Sie unterstützt auch sowohl SSD- als auch HDD-Speicher.

Sie können den gewünschten Bereitstellungstyp unter Parametern im Kubernetes des FSx For Lustre-Dateisystems konfigurieren. StorageClass Hier ist ein Link, der Beispielvorlagen bereitstellt.

> **Note**
>
> Für latenzempfindliche Workloads oder Workloads, die ein Höchstmaß an IOPS/Durchsatz erfordern, sollten Sie SSD-Speicher wählen. Für durchsatzorientierte Workloads, die nicht latenzempfindlich sind, sollten Sie HDD-Speicher wählen.

Aktivieren Sie die Datenkomprimierung

Sie können die Datenkomprimierung auch in Ihrem Dateisystem aktivieren, indem Sie "LZ4" als Datenkomprimierungstyp angeben. Sobald sie aktiviert ist, werden alle neu geschriebenen Dateien FSx für Lustre automatisch komprimiert, bevor sie auf die Festplatte geschrieben und beim Lesen dekomprimiert werden. LZ4 Der Datenkomprimierungsalgorithmus ist verlustfrei, sodass die Originaldaten aus den komprimierten Daten vollständig rekonstruiert werden können.

Sie können den Datenkomprimierungstyp wie LZ4 unter den Parametern im Kubernetes des FSx For Lustre-Dateisystems konfigurieren. StorageClass Die Komprimierung ist deaktiviert, wenn der Wert auf NONE gesetzt ist, was die Standardeinstellung ist. Dieser Link enthält Beispielvorlagen.

> **Note**
>
> Amazon FSx for Lustre ist nicht mit Windows-basierten Container-Images kompatibel.

Amazon FSx für NetApp ONTAP

Amazon FSx for NetApp ONTAP ist ein vollständig verwalteter gemeinsam genutzter Speicher, der auf dem NetApp ONTAP-Dateisystem basiert. FSx for ONTAP bietet funktionsreichen, schnellen und flexiblen gemeinsamen Dateispeicher, auf den von Linux-, Windows- und macOS-Recheninstanzen aus, die in AWS oder vor Ort ausgeführt werden, umfassend zugegriffen werden kann.

Amazon FSx für NetApp ONTAP unterstützt zwei Speicherstufen: 1 primäre Stufe und 2 Speicherkapazitätspoolstufe.

Die primäre Stufe ist eine bereitgestellte, leistungsstarke SSD-basierte Stufe für aktive, latenzempfindliche Daten. Der vollständig elastische Kapazitätspool-Tier ist kostenoptimiert für Daten, auf die selten zugegriffen wird, wird automatisch skaliert, wenn die Daten darauf verteilt werden, und bietet eine praktisch unbegrenzte Kapazität im Petabyte-Bereich. Sie können Datenkomprimierung und Deduplizierung im Kapazitätspoolspeicher aktivieren und so den Speicherverbrauch Ihrer Daten weiter reduzieren. NetAppDie systemeigene, richtlinienbasierte FabricPool Funktion überwacht kontinuierlich die Datenzugriffsmuster und überträgt Daten automatisch bidirektional zwischen Speicherebenen, um Leistung und Kosten zu optimieren.

NetAppAstra Trident bietet dynamische Speicherorchestrierung mithilfe eines CSI-Treibers, der es Amazon EKS-Clustern ermöglicht, den Lebenszyklus persistenter Volumes zu verwalten, die von Amazon FSx für NetApp ONTAP-Dateisysteme PVs unterstützt werden. Erste Schritte finden Sie unter Verwenden von Astra Trident mit Amazon FSx für NetApp ONTAP in der Astra Trident-Dokumentation.

Weitere Überlegungen

Minimiere die Größe des Container-Images

Sobald Container bereitgestellt sind, werden Container-Images auf dem Host in mehreren Layern zwischengespeichert. Durch die Reduzierung der Größe von Bildern kann der auf dem Host benötigte Speicherplatz reduziert werden.

Indem Sie von Anfang an abgespeckte Basis-Images wie Scratch-Images oder Container-Images ohne Distribution (die nur Ihre Anwendung und ihre Laufzeitabhängigkeiten enthalten) verwenden, können Sie die Speicherkosten senken und zusätzlich weitere Vorteile wie die Verringerung der Angriffsfläche und kürzere Abrufzeiten von Images nutzen.

Sie sollten auch die Verwendung von Open-Source-Tools wie Slim.ai in Betracht ziehen, die eine einfache und sichere Methode zur Erstellung minimaler Images bieten.

Mehrere Ebenen von Paketen, Tools, Anwendungsabhängigkeiten und Bibliotheken können die Größe des Container-Images leicht erhöhen. Durch die Verwendung mehrstufiger Builds können Sie Artefakte selektiv von einer Phase in eine andere kopieren und dabei alles, was nicht benötigt wird, aus dem endgültigen Image ausschließen. Weitere bewährte Methoden zur Image-Erstellung finden Sie hier.

Eine weitere Sache, die Sie berücksichtigen sollten, ist, wie lange zwischengespeicherte Bilder gespeichert werden sollen. Möglicherweise möchten Sie die veralteten Bilder aus dem Image-Cache löschen, wenn eine bestimmte Menge an Speicherplatz belegt ist. Auf diese Weise können Sie sicherstellen, dass genügend Speicherplatz für den Betrieb des Hosts zur Verfügung steht. Standardmäßig führt das Kubelet alle fünf Minuten eine Garbage-Collection für unbenutzte Bilder und jede Minute für unbenutzte Container durch.

Um Optionen für die Garbage-Collection von ungenutzten Containern und Bildern zu konfigurieren, optimieren Sie das Kubelet mithilfe einer Konfigurationsdatei und ändern Sie die Parameter für die Garbage-Collection anhand des Ressourcentyps. *`KubeletConfiguration`*

In der Kubernetes-Dokumentation erfahren Sie mehr darüber.

Beobachtbarkeit

Einführung

Observability-Tools helfen Ihnen dabei, Ihre Workloads effizient zu erkennen, zu beheben und zu untersuchen. Die Kosten für Telemetriedaten steigen natürlich mit zunehmender Nutzung von EKS. Manchmal kann es schwierig sein, Ihre betrieblichen Anforderungen in Einklang zu bringen und zu messen, was für Ihr Unternehmen wichtig ist, und die Kosten für die Beobachtbarkeit unter Kontrolle zu halten. Dieser Leitfaden konzentriert sich auf Strategien zur Kostenoptimierung für die drei Säulen der Beobachtbarkeit: Logs, Metriken und Traces. Jede dieser bewährten Methoden kann unabhängig voneinander angewendet werden, um den Optimierungszielen Ihres Unternehmens gerecht zu werden.

Protokollierung

Die Protokollierung spielt eine wichtige Rolle bei der Überwachung und Fehlerbehebung der Anwendungen in Ihrem Cluster. Es gibt verschiedene Strategien, mit denen die Protokollierungskosten optimiert werden können. Zu den im Folgenden aufgeführten Best-Practice-Strategien gehören die Überprüfung Ihrer Richtlinien zur Aufbewahrung von Protokollen, um detaillierte Kontrollen darüber zu implementieren, wie lange Protokolldaten aufbewahrt werden, das Senden von Protokolldaten je nach Wichtigkeit an verschiedene Speicheroptionen und die Verwendung von Protokollfiltern, um die Arten der gespeicherten Protokollnachrichten einzugrenzen. Eine effiziente Verwaltung der Protokolltelemetrie kann zu Kosteneinsparungen für Ihre Umgebungen führen.

EKS-Steuerebene

Optimieren Sie Ihre Steuerebenenprotokolle

Die Kubernetes-Steuerebene besteht aus einer Reihe von Komponenten, die die Cluster verwalten. Diese Komponenten senden verschiedene Arten von Informationen als Protokollstreams an eine Protokollgruppe in Amazon. CloudWatch Die Aktivierung aller Protokolltypen auf der Kontrollebene bietet zwar Vorteile, Sie sollten sich jedoch der Informationen in den einzelnen Protokollen und der damit verbundenen Kosten für die Speicherung der gesamten Protokolltelemetrie bewusst sein. Ihnen werden die standardmäßigen CloudWatch Logs-Datenaufnahme- und Speicherkosten für Logs in Rechnung gestellt, die von Ihren Clustern an Amazon CloudWatch Logs gesendet werden. Bevor Sie sie aktivieren, sollten Sie prüfen, ob jeder Protokollstream erforderlich ist.

Aktivieren Sie beispielsweise in Clustern, die nicht zur Produktion gehören, bestimmte Protokolltypen, z. B. die API-Serverprotokolle, selektiv nur zur Analyse und deaktivieren Sie sie anschließend. Aber für Produktionscluster, bei denen Sie Ereignisse möglicherweise nicht reproduzieren können und für die Lösung von Problemen mehr Protokollinformationen erforderlich sind, können Sie alle Protokolltypen aktivieren. Weitere Einzelheiten zur Implementierung der Kostenoptimierung auf Kontrollebene finden Sie in diesem Blogbeitrag.

Protokolle auf S3 streamen

Eine weitere bewährte Methode zur Kostenoptimierung ist das Streamen von Protokollen der Kontrollebene über CloudWatch Logs-Abonnements nach S3. Durch die Nutzung von CloudWatch Logs-Abonnements können Sie Protokolle selektiv an S3 weiterleiten, was eine kostengünstigere Langzeitspeicherung ermöglicht als die unbefristete Aufbewahrung von Protokollen. CloudWatch Für Produktionscluster können Sie beispielsweise eine kritische Protokollgruppe erstellen und Abonnements nutzen, um diese Protokolle nach 15 Tagen an S3 zu streamen. Auf diese Weise können Sie schnell auf die zu analysierenden Protokolle zugreifen, sparen aber auch Kosten, indem Sie die Protokolle in einen kostengünstigeren Speicher verschieben.

> **Important**
>
> Seit dem 5.9.2023 werden EKS-Logs in Amazon Logs als verkaufte Logs eingestuft. CloudWatch Vended Logs sind spezifische AWS-Serviceprotokolle, die nativ von AWS-Services im Namen des Kunden veröffentlicht werden und zu Mengenrabatten erhältlich sind. Bitte besuchen Sie die Seite mit den CloudWatch Amazon-Preisen, um mehr über die Preise von Vended Logs zu erfahren.

EKS-Datenebene

Aufbewahrung von Protokollen

CloudWatchDie Standard-Aufbewahrungsrichtlinie von Amazon sieht vor, dass Protokolle auf unbestimmte Zeit aufbewahrt werden und niemals ablaufen, wodurch Speicherkosten anfallen, die für Ihre AWS-Region gelten. Um die Speicherkosten zu reduzieren, können Sie die Aufbewahrungsrichtlinie für jede Protokollgruppe an Ihre Workload-Anforderungen anpassen.

In einer Entwicklungsumgebung ist eine lange Aufbewahrungsfrist möglicherweise nicht erforderlich. In einer Produktionsumgebung können Sie jedoch eine längere Aufbewahrungsrichtlinie festlegen,

um die Anforderungen an Problembehandlung, Compliance und Kapazitätsplanung zu erfüllen. Wenn Sie beispielsweise in der Hauptsaison an Feiertagen eine E-Commerce-Anwendung ausführen, das System stärker ausgelastet ist und Probleme auftreten können, die möglicherweise nicht sofort erkennbar sind, sollten Sie eine längere Protokollspeicherung für eine detaillierte Fehlerbehebung und Analyse nach dem Ereignis einrichten.

Sie können Ihre Aufbewahrungszeiträume in der CloudWatch AWS-Konsole oder AWS-API mit einer Dauer von 1 Tag bis 10 Jahren konfigurieren, basierend auf jeder Protokollgruppe. Durch eine flexible Aufbewahrungsfrist können Sie Kosten für die Speicherung von Protokollen sparen und gleichzeitig wichtige Protokolle verwalten.

Optionen zum Speichern von Protokollen

Die Speicherung ist ein großer Kostenfaktor für die Beobachtbarkeit. Daher ist es wichtig, Ihre Strategie zur Speicherung von Protokollen zu optimieren. Ihre Strategien sollten auf Ihre Workload-Anforderungen abgestimmt sein und gleichzeitig Leistung und Skalierbarkeit beibehalten. Eine Strategie zur Senkung der Kosten für das Speichern von Protokollen besteht darin, die AWS S3-Buckets und die verschiedenen Speicherstufen zu nutzen.

Leiten Sie Protokolle direkt an S3 weiter

Erwägen Sie, weniger kritische Protokolle, z. B. Entwicklungsumgebungen, direkt an S3 statt an Cloudwatch weiterzuleiten. Dies kann sich unmittelbar auf die Kosten für die Protokollspeicherung auswirken. Eine Option besteht darin, die Protokolle mithilfe von Fluentbit direkt an S3 weiterzuleiten. Sie definieren dies in dem `[OUTPUT]` Abschnitt, dem Ziel, FluentBit an das Container-Logs zur Aufbewahrung übertragen werden. Weitere Konfigurationsparameter finden Sie hier.

```
[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
```

Logs CloudWatch nur zur kurzfristigen Analyse weiterleiten

Für kritischere Protokolle, z. B. Produktionsumgebungen, in denen Sie möglicherweise eine sofortige Analyse der Daten durchführen müssen, sollten Sie die Protokolle weiterleiten an CloudWatch. Sie

definieren dies in dem `[OUTPUT]` Abschnitt, dem Ziel, an das FluentBit Container-Protokolle zur Aufbewahrung übertragen werden. Weitere Konfigurationsparameter finden Sie hier.

```
[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
```

Dies wird sich jedoch nicht sofort auf Ihre Kosteneinsparungen auswirken. Um zusätzliche Einsparungen zu erzielen, müssen Sie diese Protokolle nach Amazon S3 exportieren.

Export nach Amazon S3 von CloudWatch

Für die langfristige Speicherung von CloudWatch Amazon-Protokollen empfehlen wir, Ihre Amazon CloudWatch EKS-Protokolle nach Amazon Simple Storage Service (Amazon S3) zu exportieren. Sie können die Protokolle an den Amazon S3 S3-Bucket weiterleiten, indem Sie eine Exportaufgabe über die Konsole oder die API erstellen. Nachdem Sie dies getan haben, bietet Amazon S3 viele Optionen zur weiteren Kostensenkung. Sie können Ihre eigenen Amazon S3 Lifecycle-Regeln definieren, um Ihre Protokolle in eine Speicherklasse zu verschieben, die Ihren Anforderungen entspricht, oder die Amazon S3 S3-Speicherklasse Intelligent-Tiering nutzen, damit AWS Daten basierend auf Ihrem Nutzungsmuster automatisch in den Langzeitspeicher verschiebt. Weitere Informationen finden Sie in diesem Blog. Für Ihre Produktionsumgebung werden die Protokolle CloudWatch beispielsweise länger als 30 Tage gespeichert und dann in den Amazon S3 S3-Bucket exportiert. Sie können dann Amazon Athena verwenden, um die Daten im Amazon S3 S3-Bucket abzufragen, falls Sie zu einem späteren Zeitpunkt auf die Protokolle zurückgreifen müssen.

Reduzieren Sie die Protokollebenen

Üben Sie die selektive Protokollierung für Ihre Anwendung. Sowohl Ihre Anwendungen als auch Ihre Knoten geben standardmäßig Protokolle aus. Passen Sie für Ihre Anwendungsprotokolle die Protokollebenen an die Wichtigkeit der Arbeitslast und der Umgebung an. Die unten stehende Java-Anwendung gibt beispielsweise `INFO` Protokolle aus. Dies ist die typische Standardanwendungskonfiguration, die je nach Code zu einer großen Menge an Protokolldaten führen kann.

```
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!");    } }
```

Ändern Sie in einer Entwicklungsumgebung Ihre Protokollebene auf`DEBUG`, da dies Ihnen helfen kann, Probleme zu debuggen oder catch Probleme zu erkennen, bevor sie produktiv werden.

```
      log.setLevel(Level.DEBUG);
```

In einer Produktionsumgebung sollten Sie erwägen, Ihre Protokollebene auf `ERROR` oder `FATAL` zu ändern. Dadurch wird das Protokoll nur dann ausgegeben, wenn Ihre Anwendung Fehler aufweist. Dadurch wird die Protokollausgabe reduziert und Sie können sich auf wichtige Daten zum Status Ihrer Anwendung konzentrieren.

```
      log.setLevel(Level.ERROR);
```

Sie können die Protokollebenen verschiedener Kubernetes-Komponenten fein abstimmen. Wenn Sie beispielsweise Bottlerocket als Ihr EKS-Node-Betriebssystem verwenden, gibt es Konfigurationseinstellungen, mit denen Sie die Protokollebene des Kubelet-Prozesses anpassen können. Ein Auszug dieser Konfigurationseinstellung finden Sie weiter unten. Beachten Sie die Standardprotokollebene von 2, mit der die Ausführlichkeit der Protokollierung des Prozesses angepasst wird. `kubelet`

```
[settings.kubernetes]
log-level = "2"
image-gc-high-threshold-percent = "85"
image-gc-low-threshold-percent = "80"
```

In einer Entwicklungsumgebung können Sie die Protokollebene auf einen höheren Wert als 2 setzen, um zusätzliche Ereignisse anzuzeigen. Dies ist ideal für das Debuggen. In einer Produktionsumgebung können Sie die Stufe auf 0 setzen, um nur kritische Ereignisse anzuzeigen.

Nutzen Sie Filter

Wenn Sie eine standardmäßige EKS-Fluentbit-Konfiguration verwenden, um Container-Logs an Cloudwatch zu senden, werden ALLE mit Kubernetes-Metadaten angereicherten Anwendungscontainer-Logs FluentBit erfasst und an Cloudwatch gesendet, wie im Konfigurationsblock unten gezeigt. `[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}
```

Im obigen `[INPUT]` Abschnitt werden alle Container-Logs aufgenommen. Dadurch kann eine große Datenmenge generiert werden, die möglicherweise nicht erforderlich ist. Durch das Herausfiltern dieser Daten kann die Menge der gesendeten Protokolldaten reduziert und CloudWatch somit Ihre Kosten gesenkt werden. Sie können einen Filter auf Ihre Protokolle anwenden, bevor sie ausgegeben werden CloudWatch. Fluentbit definiert dies im `[FILTER]` Abschnitt. Wenn Sie beispielsweise die Kubernetes-Metadaten herausfiltern, damit sie nicht an Protokollereignisse angehängt werden, können Sie Ihr Protokollvolumen reduzieren.

```
    [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.
```

Metriken

Metriken liefern wertvolle Informationen über die Leistung Ihres Systems. Durch die Konsolidierung aller systembezogenen oder verfügbaren Ressourcenmetriken an einem zentralen Ort erhalten Sie die Möglichkeit, Leistungsdaten zu vergleichen und zu analysieren. Dieser zentralisierte Ansatz ermöglicht es Ihnen, fundiertere strategische Entscheidungen zu treffen, z. B. die Skalierung von Ressourcen nach oben oder unten. Darüber hinaus spielen Kennzahlen eine entscheidende Rolle bei der Bewertung des Zustands von Ressourcen, sodass Sie bei Bedarf proaktive Maßnahmen ergreifen können. Im Allgemeinen steigen die Kosten für Beobachtbarkeit mit der Erfassung und Aufbewahrung von Telemetriedaten. Im Folgenden finden Sie einige Strategien, mit denen Sie die Kosten der metrischen Telemetrie senken können: Erfassung nur wichtiger Metriken, Reduzierung der Kardinalität Ihrer Telemetriedaten und Feinabstimmung der Granularität Ihrer Telemetriedatenerfassung.

Überwachen Sie, worauf es ankommt, und sammeln Sie nur das, was Sie benötigen

Die erste Strategie zur Kostensenkung besteht darin, die Anzahl der gesammelten Kennzahlen zu reduzieren und damit die Aufbewahrungskosten zu senken.

1. Gehen Sie zunächst von Ihren und/oder den Anforderungen Ihrer Stakeholder aus, um die wichtigsten Kennzahlen zu ermitteln. Erfolgskennzahlen sind für jeden anders! Wissen Sie, wie gut aussieht, und messen Sie es.
2. Erwägen Sie, sich eingehend mit den Workloads, die Sie unterstützen, zu befassen und deren Key Performance Indicators (KPIs), auch bekannt als „Goldene Signale“, zu identifizieren. Diese sollten den Anforderungen der Unternehmen und der Interessengruppen entsprechen. Die Berechnung

SLIs und SLAs Verwendung von Amazon CloudWatch und Metric Math ist entscheidend für die Verwaltung der Servicezuverlässigkeit. SLOs Folgen Sie den in diesem Leitfaden beschriebenen bewährten Methoden, um die Leistung Ihrer EKS-Umgebung effektiv zu überwachen und aufrechtzuerhalten.

3. Fahren Sie dann mit den verschiedenen Ebenen der Infrastruktur fort, um EKS-Cluster, Knoten und zusätzliche Infrastrukturkennzahlen miteinander zu verbinden und mit Ihrer Arbeitslast KPIs zu korrelieren. Speichern Sie Ihre Geschäfts- und Betriebskennzahlen in einem System, in dem Sie sie miteinander korrelieren und Schlussfolgerungen auf der Grundlage der beobachteten Auswirkungen auf beide ziehen können.
4. EKS stellt Metriken aus der Kontrollebene, dem Cluster kube-state-metrics, den Pods und den Knoten zur Verfügung. Die Relevanz all dieser Metriken hängt von Ihren Anforderungen ab. Es ist jedoch wahrscheinlich, dass Sie nicht jede einzelne Metrik auf den verschiedenen Ebenen benötigen. Sie können diesen Leitfaden zu den wichtigsten EKS-Metriken als Grundlage für die Überwachung des allgemeinen Zustands eines EKS-Clusters und Ihrer Workloads verwenden.

Hier ist ein Beispiel für eine Prometheus-Scrape-Konfiguration, in der wir die verwenden, `relabel_config` um nur Kubelet-Metriken beizubehalten und `metric_relabel_config` alle Container-Metriken zu löschen.

```
  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
```

Reduzieren Sie gegebenenfalls die Kardinalität

Kardinalität bezieht sich auf die Einzigartigkeit der Datenwerte in Kombination mit ihren Dimensionen (z. B. Prometheus-Bezeichnungen) für einen bestimmten Metriksatz. Metriken mit hoher Kardinalität haben viele Dimensionen, und jede Kombination von Dimensionsmetriken weist eine höhere Eindeutigkeit auf. Eine höhere Kardinalität führt zu einer größeren Größe und einem größeren Speicherbedarf für metrische Telemetriedaten, was die Kosten erhöht.

Im Beispiel mit hoher Kardinalität unten sehen wir, dass die Metrik Latenz Dimensionen, RequestID, CustomerID und Service hat und dass jede Dimension viele eindeutige Werte hat. Die Kardinalität ist das Maß für die Kombination der Anzahl möglicher Werte pro Dimension. In Prometheus wird jeder Satz einzigartiger Dimensionen/Labels als neue Metrik betrachtet, daher bedeutet eine hohe Kardinalität mehr Metriken.

In EKS-Umgebungen mit vielen Metriken und Dimensionen/Labels pro Metrik (Cluster, Namespace, Service, Pod, Container usw.) nimmt die Kardinalität tendenziell zu. Um die Kosten zu optimieren, sollten Sie die Kardinalität der von Ihnen gesammelten Kennzahlen sorgfältig berücksichtigen. Wenn Sie beispielsweise eine bestimmte Metrik für die Visualisierung auf Clusterebene aggregieren, können Sie zusätzliche Labels löschen, die sich auf einer niedrigeren Ebene befinden, z. B. das Namespace-Label.

Um Metriken mit hoher Kardinalität in Prometheus zu identifizieren, können Sie die folgende PROMQL-Abfrage ausführen, um zu ermitteln, welche Scrape-Ziele die höchste Anzahl von Metriken haben (Kardinalität):

```
topk_max(5, max_over_time(scrape_samples_scraped[1h]))
```

und die folgende PROMQL-Abfrage kann Ihnen dabei helfen, festzustellen, welche Scrape-Ziele die höchsten Abwanderungsraten (wie viele neue Metrikreihen wurden in einem bestimmten Scrape erstellt) aufweisen:

```
topk_max(5, max_over_time(scrape_series_added[1h]))
```

Wenn Sie Grafana verwenden, können Sie das Mimirtool von Grafana Lab verwenden, um Ihre Grafana-Dashboards und Prometheus-Regeln zu analysieren, um ungenutzte Metriken mit hoher Kardinalität zu identifizieren. Folgen Sie dieser Anleitung zur Verwendung der `mimirtool analyze prometheus` Befehle `mimirtool analyze` und, um aktive Metriken zu identifizieren, auf die in Ihren Dashboards nicht verwiesen wird.

Beachten Sie die Granularität der Metriken

Das Sammeln von Metriken mit einer höheren Granularität, z. B. jede Sekunde im Vergleich zu jeder Minute, kann einen großen Einfluss darauf haben, wie viel Telemetrie gesammelt und gespeichert wird, was die Kosten erhöht. Legen Sie sinnvolle Intervalle für die Erfassung oder Erfassung von Metriken fest, die zwischen ausreichender Granularität zur Erkennung vorübergehender Probleme und ausreichend niedrig sind, um kostengünstig zu sein. Verringern Sie die Granularität von Kennzahlen, die für die Kapazitätsplanung und die Analyse größerer Zeitfenster verwendet werden.

Im Folgenden finden Sie einen Auszug aus der Standardkonfiguration von AWS Distro for Opentelemetry (ADOT) EKS Addon Collector.

> **Important**
>
> Das globale Prometheus-Scrape-Intervall ist auf 15 Sekunden festgelegt. Dieses Scrape-Intervall kann verlängert werden, was zu einer Verringerung der Menge der in Prometheus gesammelten metrischen Daten führt.

```
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
```

Nachverfolgung

Die Hauptkosten im Zusammenhang mit der Rückverfolgung sind auf die Generierung von Trace-Speichern zurückzuführen. Bei der Rückverfolgung geht es darum, genügend Daten zu sammeln, um Leistungsaspekte zu diagnostizieren und zu verstehen. Da die Kosten für Röntgen-Traces jedoch auf Daten basieren, die an X-Ray weitergeleitet werden, reduziert das Löschen von Spuren nach der Weiterleitung Ihre Kosten nicht. Lassen Sie uns untersuchen, wie Sie Ihre Kosten für die Rückverfolgung senken und gleichzeitig die Daten verwalten können, damit Sie eine ordnungsgemäße Analyse durchführen können.

Wenden Sie Stichprobenregeln an

Die Röntgenabtastrate ist standardmäßig konservativ. Definieren Sie Stichprobenregeln, mit denen Sie die Menge der gesammelten Daten steuern können. Dies verbessert die Leistungseffizienz und senkt gleichzeitig die Kosten. Indem Sie die Samplingrate verringern, können Sie Traces aus der Anfrage nur für das sammeln, was Ihre Workloads benötigen, und gleichzeitig eine niedrigere Kostenstruktur beibehalten.

Sie haben beispielsweise eine Java-Anwendung, mit der Sie die Traces aller Anfragen für eine problematische Route debuggen möchten.

Konfigurieren Sie über das SDK, um Sampling-Regeln aus einem JSON-Dokument zu laden

```
{
"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
  }
}
```

Über die Konsole

Wenden Sie Tail Sampling mit AWS Distro for OpenTelemetry (ADOT) an

Mit ADOT Tail Sampling können Sie das Volumen der im Service aufgenommenen Spuren kontrollieren. Mit Tail Sampling können Sie die Sampling-Richtlinien jedoch erst definieren, nachdem alle Zeiträume der Anfrage abgeschlossen wurden, und nicht zu Beginn. Dadurch wird die Menge der übertragenen Rohdaten weiter begrenzt CloudWatch, wodurch die Kosten gesenkt werden.

Wenn Sie beispielsweise 1% der Zugriffe auf eine Landingpage und 10% der Anfragen auf eine Zahlungsseite auswerten, kann dies dazu führen, dass Sie über einen Zeitraum von 30 Minuten 300 Traces haben. Mit einer ADOT Tail Sampling-Regel, die bestimmte Fehler filtert, könnten Ihnen 200 Traces übrig bleiben, was die Anzahl der gespeicherten Traces verringert.

```
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]
```

Nutzen Sie die Amazon S3 S3-Speicheroptionen

Sie sollten den AWS S3-Bucket und seine verschiedenen Speicherklassen nutzen, um die Traces zu speichern. Exportieren Sie Traces nach S3, bevor die Aufbewahrungsfrist abläuft. Verwenden Sie die Amazon S3 S3-Lebenszyklusregeln, um die Trace-Daten in die Speicherklasse zu verschieben, die Ihren Anforderungen entspricht.

Wenn Sie beispielsweise Traces haben, die 90 Tage alt sind, kann Amazon S3 Intelligent-Tiering die Daten basierend auf Ihrem Nutzungsmuster automatisch in den Langzeitspeicher verschieben. Sie können Amazon Athena verwenden, um die Daten in Amazon S3 abzufragen, wenn Sie zu einem späteren Zeitpunkt auf die Traces zurückgreifen müssen. Dies kann Ihre Kosten für verteiltes Tracing weiter senken.

Zusätzliche Ressourcen:

- Leitfaden zu bewährten Methoden im Bereich Observability
- Erfassung von Kennzahlen zu bewährten Verfahren
- AWS re:Invent 2022 — Bewährte Methoden für Observability bei Amazon () COP343
- AWS re:Invent 2022 — Observability: Bewährte Methoden für moderne Anwendungen () COP344

Bewährte Methoden für Windows

Dieses Handbuch enthält Hinweise zum Ausführen von Windows-Containern und -Knoten.

Themen

- Amazon EKS hat die Windows AMI-Verwaltung optimiert
- gMSA für Windows-Pods und Container konfigurieren
- Härtung von Windows-Worker-Knoten
- Scannen von Container-Bildern
- Windows Server-Version und -Lizenz
- Protokollierung
- Überwachen
- Windows-Netzwerke
- Vermeidung von OOM-Fehlern
- Patchen von Windows-Servern und Containern
- Ausführung heterogener Workloads
- Sicherheitskontexte für Pods
- Persistente Speicheroptionen
- Härten von Windows-Container-Images

Amazon EKS hat die Windows AMI-Verwaltung optimiert

Windows Amazon EKS Optimized AMIs basiert auf Windows Server 2019 und Windows Server 2022. Sie sind so konfiguriert, dass sie als Basis-Image für Amazon-EKS-Knoten dienen. Standardmäßig AMIs enthalten sie die folgenden Komponenten:

- Kubelet
- kube-proxy
- AWS IAM Authenticator für Kubernetes
- csi-proxy
- containerd

Sie können die Amazon Machine Image (AMI) -ID für Amazon EKS programmgesteuert abrufen, AMIs indem Sie die AWS Systems Manager Parameter Store Store-API abfragen. Mit diesem Parameter müssen Sie nicht manuell nach Amazon EKS-optimiertem AMI suchen IDs. Weitere Informationen zur Systems Manager Parameter Store-API finden Sie unter GetParameter. Ihr Benutzerkonto muss über die ssm: GetParameter IAM-Berechtigung verfügen, um die für Amazon EKS optimierten AMI-Metadaten abzurufen.

Im folgenden Beispiel wird die AMI-ID für das neueste Amazon EKS-optimierte AMI für Windows Server 2019 LTSC Core abgerufen. Die im AMI-Namen aufgeführte Versionsnummer bezieht sich auf den entsprechenden Kubernetes-Build, für den es vorbereitet ist.

```
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
```

Beispielausgabe:

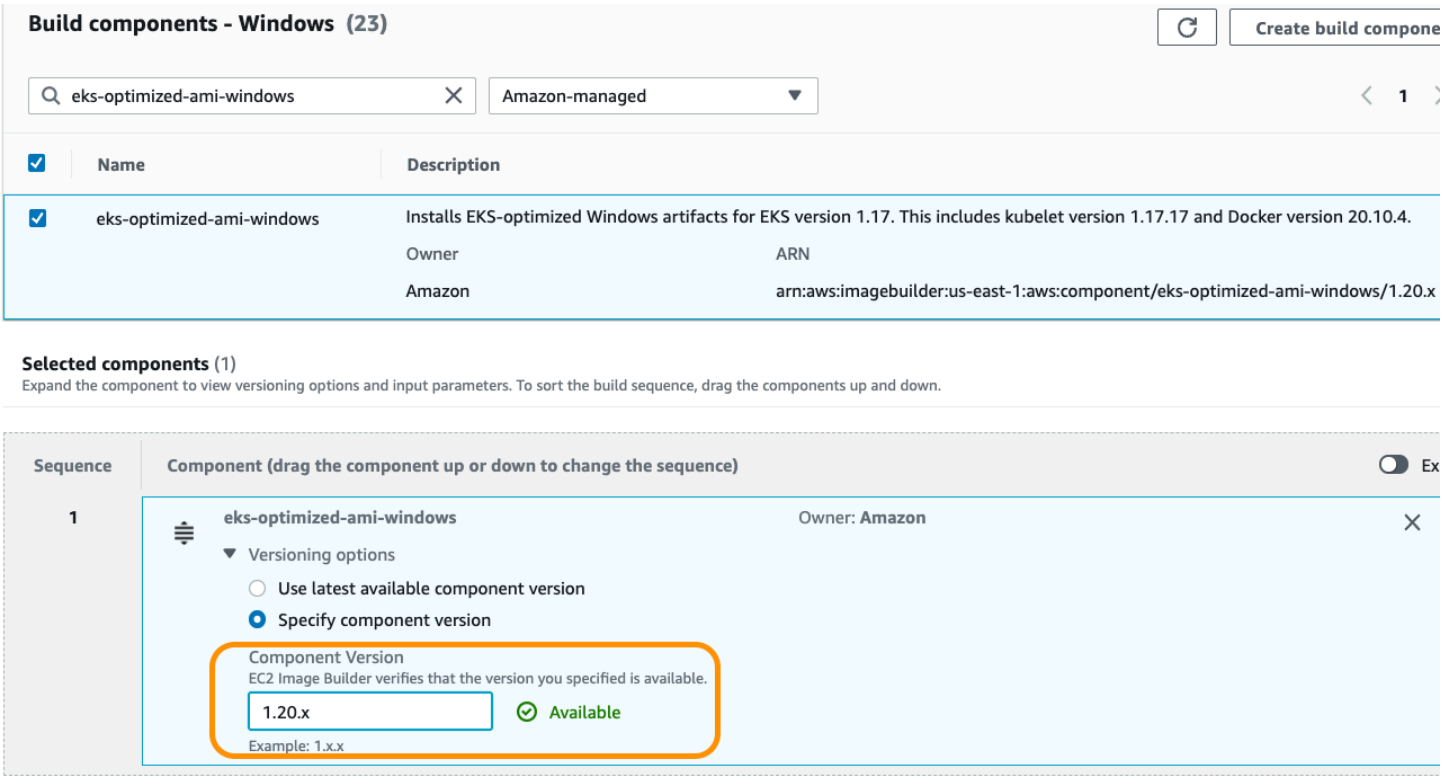
```
ami-09770b3eec4552d4e
```

Verwaltung Ihres eigenen für Amazon EKS optimierten Windows-AMI

Ein wichtiger Schritt in Richtung Produktionsumgebungen ist die Beibehaltung derselben für Amazon EKS optimierten Windows AMI- und Kubelet-Version im gesamten Amazon EKS-Cluster.

Die Verwendung derselben Version im gesamten Amazon EKS-Cluster reduziert die Zeit bei der Fehlerbehebung und erhöht die Cluster-Konsistenz. Amazon EC2 Image Builder hilft bei der Erstellung und Verwaltung von benutzerdefinierten, für Amazon EKS optimierten Windows AMIs , die in einem Amazon EKS-Cluster verwendet werden können.

Verwenden Sie Amazon EC2 Image Builder, um zwischen Windows Server-Versionen, AWS Windows Server AMI-Veröffentlichungsdaten und and/or Betriebssystem-Build-Versionen zu wählen. Im Schritt Komponenten erstellen können Sie zwischen vorhandenen EKS-optimierten Windows-Artefakten und den Kubelet-Versionen wählen. Weitere Informationen finden Sie unter: .html https:// docs.aws.amazon.com/eks/ latest/userguide/eks custom-ami-windows



HINWEIS: Bevor Sie ein Basis-Image auswählen, finden Sie im Abschnitt Windows Server-Version und -Lizenz wichtige Informationen zu Release-Channel-Updates.

Konfiguration eines schnelleren Starts für benutzerdefinierte EKS-Optimierungen AMIs

Bei Verwendung eines benutzerdefinierten, für Windows Amazon EKS optimierten AMI können Windows-Worker-Knoten bis zu 65% schneller gestartet werden, wenn die Schnellstartfunktion aktiviert wird. Diese Funktion verwaltet eine Reihe von vorab bereitgestellten Snapshots, für die die Schritte Sysprep Specialized, Windows Out of Box Experience (OOBE) und erforderliche Neustarts bereits abgeschlossen sind. Diese Snapshots werden dann bei nachfolgenden Starts verwendet, wodurch die Zeit für das Skalieren oder Ersetzen von Knoten reduziert wird. Fast Launch kann nur für AMIs Sie selbst über die EC2-Konsole oder in der AWS-CLI aktiviert werden, und die Anzahl der verwalteten Snapshots ist konfigurierbar.

HINWEIS: Fast Launch ist nicht mit dem von Amazon bereitgestellten EKS-optimierten Standard-AMI kompatibel. Erstellen Sie wie oben beschrieben ein benutzerdefiniertes AMI, bevor Sie versuchen, es zu aktivieren.

Weitere Informationen finden Sie unter AWS Windows AMIs — Konfigurieren Sie Ihr AMI für einen schnelleren Start

Zwischenspeichern von Windows-Basislayern auf benutzerdefinierten AMIs

Windows-Container-Images sind größer als ihre Linux-Gegenstücke. Wenn Sie eine containerisierte .NET Framework-basierte Anwendung ausführen, liegt die durchschnittliche Imagegröße bei etwa 8,24 GB. Während der Pod-Planung muss das Container-Image vollständig abgerufen und auf die Festplatte extrahiert werden, bevor der Pod den Status Running erreicht.

Während dieses Vorgangs ruft die Container-Laufzeit (containerd) das gesamte Container-Image auf der Festplatte ab und extrahiert es. Der Pull-Vorgang ist ein paralleler Prozess, was bedeutet, dass die Container-Laufzeit die Container-Image-Ebenen parallel abruft. Im Gegensatz dazu erfolgt der Extraktionsvorgang in einem sequentiellen Prozess und ist I/O intensiv. Aus diesem Grund kann es mehr als 8 Minuten dauern, bis das Container-Image vollständig extrahiert ist und von der Container-Laufzeit (containerd) verwendet werden kann. Daher kann die Startzeit des Pods mehrere Minuten dauern.

Wie im Thema Patchen von Windows Server und Containern erwähnt, besteht die Möglichkeit, ein benutzerdefiniertes AMI mit EKS zu erstellen. Während der AMI-Vorbereitung können Sie eine zusätzliche EC2 Image Builder-Komponente hinzufügen, um alle erforderlichen Windows-Container-Images lokal abzurufen und dann das AMI zu generieren. Durch diese Strategie wird die Zeit, in der ein Pod den Status Running erreicht, drastisch reduziert.

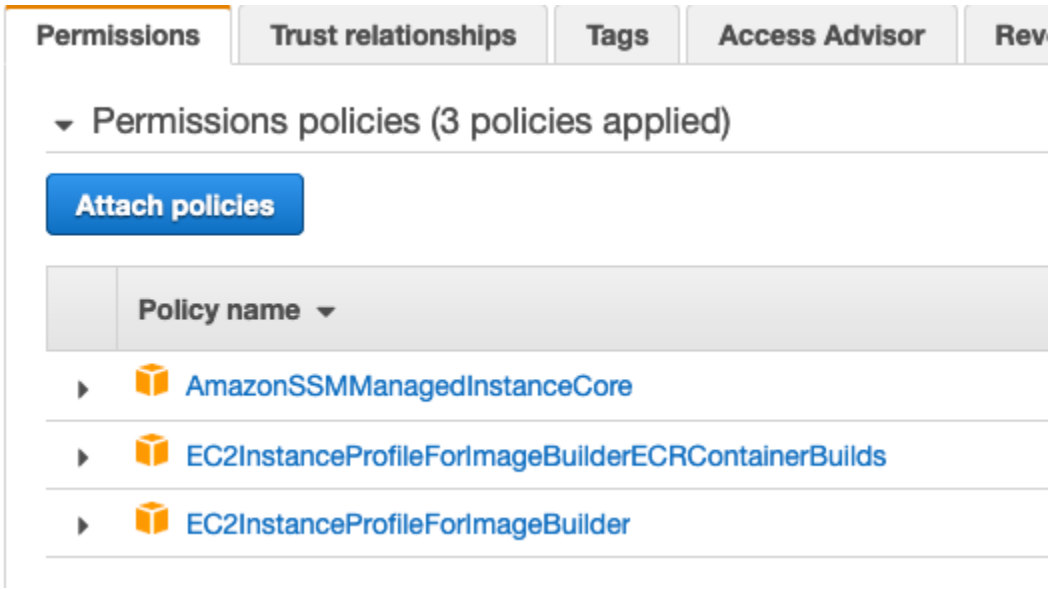
Erstellen Sie in Amazon EC2 Image Builder eine Komponente, um die erforderlichen Bilder herunterzuladen, und hängen Sie sie an das Image-Rezept an. Im folgenden Beispiel wird ein bestimmtes Bild aus einem ECR-Repository abgerufen.

```
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
```

Um sicherzustellen, dass die folgende Komponente wie erwartet funktioniert, überprüfen Sie, ob die von EC2 Image Builder (EC2InstanceProfileForImageBuilder) verwendete IAM-Rolle die angehängten Richtlinien enthält:



Blogbeitrag

Im folgenden Blogbeitrag erfahren Sie Schritt für Schritt, wie Sie die Caching-Strategie für benutzerdefinierte Amazon EKS-Windows AMIs implementieren:

Verkürzung der Startzeiten von Windows-Containern mit EC2 Image Builder und Image-Cache-Strategie

gMSA für Windows-Pods und Container konfigurieren

Was ist ein gMSA-Konto

Windows-basierte Anwendungen wie .NET-Anwendungen verwenden häufig Active Directory als Identitätsanbieter, wobei die Bereitstellung authorization/authentication über das NTLM- oder Kerberos-Protokoll erfolgt.

Ein Anwendungsserver für den Austausch von Kerberos-Tickets mit Active Directory muss in eine Domäne eingebunden sein. Windows-Container unterstützen keine Domänenbeitritte und wären

daher wenig sinnvoll, da es sich bei Containern um kurzlebige Ressourcen handelt, die den Active Directory-RID-Pool belasten.

Administratoren können jedoch gMSA Active Directory-Konten nutzen, um eine Windows-Authentifizierung für Ressourcen wie Windows-Container, NLB und Serverfarmen auszuhandeln.

Windows-Container und GMSA-Anwendungsfall

Anwendungen, die die Windows-Authentifizierung nutzen und als Windows-Container ausgeführt werden, profitieren von gMSA, da der Windows Node verwendet wird, um das Kerberos-Ticket im Namen des Containers auszutauschen. Für die Einrichtung des Windows-Worker-Knotens zur Unterstützung der gMSA-Integration stehen zwei Optionen zur Verfügung:

1 — Windows-Worker-Knoten, die in eine Domäne eingebunden sind

In diesem Setup ist der Windows-Worker-Knoten in der Active Directory-Domäne domänengebunden, und das AD-Computerkonto der Windows-Worker-Knoten wird verwendet, um sich bei Active Directory zu authentifizieren und die gMSA-Identität abzurufen, die mit dem Pod verwendet werden soll.

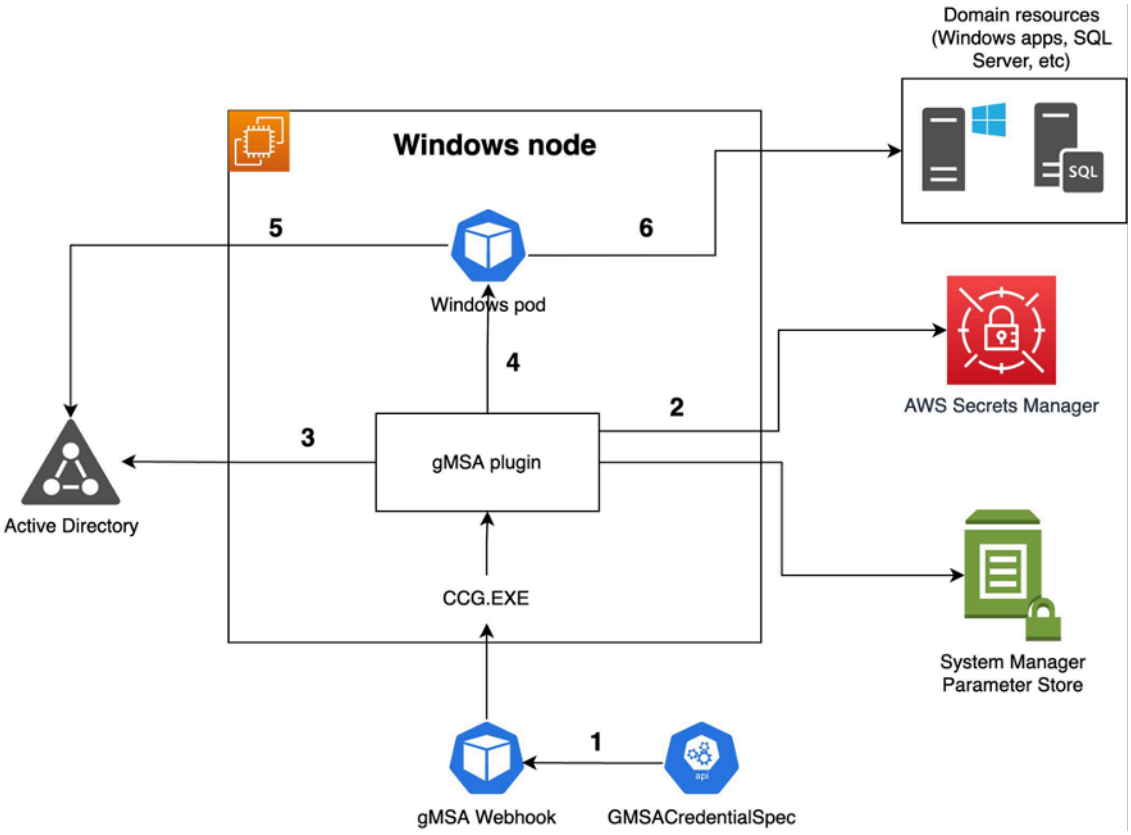
Beim domänengebundenen Ansatz können Sie Ihre Windows-Worker-Knoten mithilfe von vorhandenem Active Directory GPOs problemlos verwalten und absichern. Dies verursacht jedoch zusätzlichen Betriebsaufwand und Verzögerungen beim Beitritt von Windows-Worker-Knoten zum Kubernetes-Cluster, da zusätzliche Neustarts während des Knotenstarts und der Active Directory-Garagenreinigung erforderlich sind, nachdem der Kubernetes-Cluster die Knoten beendet hat.

Im folgenden Blogbeitrag finden Sie ausführliche Informationen zur Implementierung des Ansatzes mit domänengebundenen step-by-step Windows-Worker-Knoten:

Windows-Authentifizierung auf Amazon EKS-Windows-Pods

2 — Domänenlose Windows-Worker-Knoten

In diesem Setup gehört der Windows-Worker-Knoten nicht zur Active Directory-Domäne, und eine „portable“ Identität (Benutzer/Passwort) wird verwendet, um sich bei Active Directory zu authentifizieren und die gMSA-Identität abzurufen, die mit dem Pod verwendet werden soll.



Die portable Identität ist ein Active Directory-Benutzer; die Identität (Benutzer/Passwort) wird in AWS Secrets Manager oder AWS System Manager Parameter Store gespeichert, und ein von AWS entwickeltes Plugin namens ccg_plugin wird verwendet, um diese Identität aus AWS Secrets Manager oder AWS System Manager Parameter Store abzurufen und an containerd weiterzuleiten, um die gMSA-Identität abzurufen und für den Pod verfügbar zu machen.

Bei diesem domänenlosen Ansatz können Sie davon profitieren, dass beim Start des Windows-Worker-Knotens keine Active Directory-Interaktion stattfindet, wenn Sie gMSA verwenden, und der Betriebsaufwand für Active Directory-Administratoren reduziert wird.

Im folgenden Blogbeitrag finden Sie ausführliche Informationen step-by-step zur Implementierung des domänenlosen Windows-Worker-Node-Ansatzes:

Domänenlose Windows-Authentifizierung für Amazon EKS-Windows-Pods

Wichtiger Hinweis

Obwohl der Pod ein gMSA-Konto verwenden kann, ist es notwendig, auch die Anwendung oder den Dienst entsprechend einzurichten, um die Windows-Authentifizierung zu unterstützen. Um

beispielsweise Microsoft IIS für die Unterstützung der Windows-Authentifizierung einzurichten, sollten Sie ihn über Dockerfile vorbereiten:

```
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'
```

Härtung von Windows-Worker-Knoten

OS Hardening ist eine Kombination aus Betriebssystemkonfiguration, Patchen und Entfernen unnötiger Softwarepakete, die darauf abzielen, ein System zu sperren und die Angriffsfläche zu reduzieren. Es hat sich bewährt, Ihr eigenes EKS-optimiertes Windows-AMI mit den von Ihrem Unternehmen geforderten Härtungskonfigurationen vorzubereiten.

AWS stellt jeden Monat ein neues EKS-optimiertes Windows-AMI bereit, das die neuesten Windows Server-Sicherheitspatches enthält. Es liegt jedoch weiterhin in der Verantwortung des Benutzers, sein AMI zu schützen, indem er die erforderlichen Betriebssystemkonfigurationen anwendet, unabhängig davon, ob er selbstverwaltete oder verwaltete Knotengruppen verwendet.

Microsoft bietet eine Reihe von Tools wie das Microsoft Security Compliance Toolkit und Security Baselines, die Ihnen dabei helfen, Schutzmaßnahmen auf der Grundlage Ihrer Sicherheitsrichtlinien zu erreichen. CIS-Benchmarks sind ebenfalls verfügbar und sollten zusätzlich zu einem für Amazon EKS optimierten Windows-AMI für Produktionsumgebungen implementiert werden.

Reduzierung der Angriffsfläche mit Windows Server Core

Windows Server Core ist eine Minimalinstallationsoption, die als Teil des EKS-optimierten Windows-AMI verfügbar ist. Die Bereitstellung von Windows Server Core bietet eine Reihe von Vorteilen. Erstens hat es einen relativ geringen Festplattenbedarf, nämlich 6 GB auf Server Core gegenüber 10 GB auf Windows Server mit Desktop-Erfahrung. Zweitens hat es aufgrund seiner kleineren Codebasis eine kleinere Angriffsfläche und ist verfügbar. APIs

AWS stellt seinen Kunden AMIs jeden Monat ein neues Amazon EKS-optimiertes Windows zur Verfügung, das die neuesten Microsoft-Sicherheitspatches enthält, unabhängig von der von Amazon

EKS unterstützten Version. Als bewährte Methode gilt, dass Windows-Worker-Knoten durch neue ersetzt werden müssen, die auf dem neuesten Amazon EKS-optimierten AMI basieren. Für jeden Knoten, der länger als 45 Tage läuft, ohne dass ein Update installiert oder ein Knoten ausgetauscht wurde, fehlen bewährte Sicherheitsmethoden.

Vermeidung von RDP-Verbindungen

Remote Desktop Protocol (RDP) ist ein Verbindungsprotokoll, das von Microsoft entwickelt wurde, um Benutzern eine grafische Oberfläche zur Verfügung zu stellen, mit der sie über ein Netzwerk eine Verbindung zu einem anderen Windows-Computer herstellen können.

Als bewährte Methode sollten Sie Ihre Windows-Worker-Knoten so behandeln, als wären sie kurzlebige Hosts. Das bedeutet, dass keine Verwaltungsverbindungen, keine Updates und keine Problembehandlung erforderlich sind. Jede Änderung und Aktualisierung sollte als neues benutzerdefiniertes AMI implementiert und durch die Aktualisierung einer Auto Scaling Scaling-Gruppe ersetzt werden. Weitere Informationen finden Sie unter Patchen von Windows-Servern und Containern und optimiertes Windows AMI-Management mit Amazon EKS.

Deaktivieren Sie RDP-Verbindungen auf Windows-Knoten während der Bereitstellung, indem Sie der Eigenschaft ssh den Wert false übergeben, wie im folgenden Beispiel:

```
nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
```

Wenn Zugriff auf den Windows-Knoten erforderlich ist, verwenden Sie AWS System Manager Session Manager, um eine sichere PowerShell Sitzung über die AWS-Konsole und den SSM-Agenten einzurichten. Informationen zur Implementierung der Lösung finden Sie unter Sicherer Zugriff auf Windows-Instances mithilfe von AWS Systems Manager Session Manager

Um System Manager Session Manager verwenden zu können, muss eine zusätzliche IAM-Richtlinie auf die IAM-Rolle angewendet werden, die zum Starten des Windows-Worker-Knotens verwendet wird. Im Folgenden finden Sie ein Beispiel, in dem Amazon im `eksctl` Cluster-Manifest angegeben SSMManaged InstanceCore ist:

```
 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

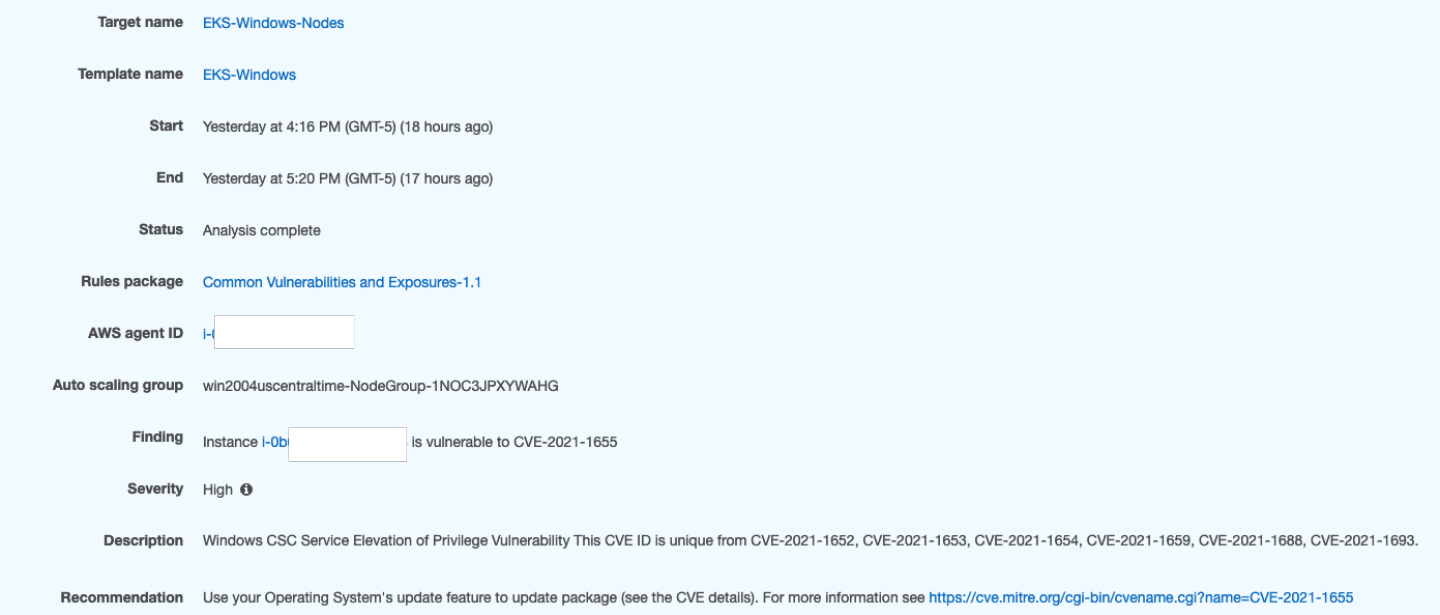
> Amazon Inspector ist ein automatisierter Sicherheitsbewertungsservice, der dazu beiträgt, die Sicherheit und Konformität der auf AWS bereitgestellten Anwendungen zu verbessern. Amazon Inspector bewertet Anwendungen automatisch auf Sicherheitslücken, Sicherheitslücken und Abweichungen von bewährten Methoden. Nach der Durchführung einer Bewertung erstellt Amazon Inspector eine detaillierte Liste der Sicherheitsfeststellungen, die nach Schweregrad priorisiert sind. Diese Ergebnisse können direkt oder als Teil detaillierter Bewertungsberichte überprüft werden, die über die Amazon Inspector Inspector-Konsole oder API verfügbar sind.

Amazon Inspector kann verwendet werden, um die CIS-Benchmark-Bewertung auf dem Windows-Worker-Knoten auszuführen, und es kann auf einem Windows Server Core installiert werden, indem die folgenden Aufgaben ausgeführt werden:

1. Laden Sie die folgende EXE-Datei herunter: https://inspector-agent.amazonaws.com/windows/installer/latest/AWSAgentInstall .exe
2. Übertragen Sie den Agenten auf den Windows-Worker-Knoten.
3. Führen Sie den folgenden Befehl aus PowerShell , um den Amazon Inspector Agent zu installieren: `.\AWSAgentInstall.exe /install`

Im Folgenden finden Sie die Ausgabe nach dem ersten Lauf. Wie Sie sehen können, wurden Ergebnisse auf der Grundlage der CVE-Datenbank generiert. Sie können dies verwenden, um

Ihre Worker-Knoten zu härten oder ein AMI auf der Grundlage der gehärteten Konfigurationen zu erstellen.



Weitere Informationen zu Amazon Inspector, einschließlich der Installation von Amazon Inspector-Agenten, der Einrichtung der CIS-Benchmark-Bewertung und der Erstellung von Berichten, finden Sie im Video Verbesserung der Sicherheit und Konformität von Windows-Workloads mit Amazon Inspector.

Amazon GuardDuty

> Amazon GuardDuty ist ein Service zur Bedrohungserkennung, der kontinuierlich nach böswilligen Aktivitäten und unberechtigtem Verhalten sucht, um Ihre AWS-Konten, Workloads und in Amazon S3 gespeicherten Daten zu schützen. Mit der Cloud wird die Erfassung und Aggregation von Konto- und Netzwerkaktivitäten vereinfacht, aber es kann für Sicherheitsteams zeitaufwändig sein, Ereignisprotokolldaten kontinuierlich auf potenzielle Bedrohungen zu analysieren.

Durch die Nutzung von Amazon haben GuardDuty Sie Einblick in böswillige Aktivitäten gegen Windows-Worker-Knoten, wie RDP-Brute-Force- und Port-Probe-Angriffe.

Sehen Sie sich das GuardDuty Video Bedrohungserkennung für Windows-Workloads mithilfe von Amazon an, um zu erfahren, wie Sie CIS-Benchmarks auf optimiertem EKS-Windows-AMI implementieren und ausführen

Sicherheit in Amazon EC2 für Windows

Informieren Sie sich über die bewährten Sicherheitsmethoden für Amazon EC2 Windows-Instances zur Implementierung von Sicherheitskontrollen auf jeder Ebene.

Scannen von Container-Bildern

Beim Scannen von Bildern handelt es sich um eine automatische Funktion zur Schwachstellenbewertung, die dazu beiträgt, die Sicherheit der Container-Images Ihrer Anwendung zu verbessern, indem sie auf eine Vielzahl von Sicherheitslücken im Betriebssystem gescannt werden.

Derzeit ist Amazon Elastic Container Registry (ECR) nur in der Lage, Linux-Container-Images auf Sicherheitslücken zu scannen. Es gibt jedoch Tools von Drittanbietern, die in eine bestehende CI/CD Pipeline für das Scannen von Windows-Container-Images integriert werden können.

- Anker
- PaloAlto Prisma-Wolke
- Trend Micro — Smart Check für tiefe Sicherheit

Weitere Informationen zur Integration dieser Lösungen in Amazon Elastic Container Repository (ECR) finden Sie unter:

- Anchore, Scannen von Bildern in Amazon Elastic Container Registry (ECR)
- PaloAlto, Scannen von Bildern auf Amazon Elastic Container Registry (ECR)
- TrendMicro, Scannen von Bildern auf Amazon Elastic Container Registry (ECR)

Windows Server-Version und -Lizenz

Windows Server Version

Ein für Amazon EKS optimiertes Windows-AMI basiert auf den Windows Server 2019 und 2022 Datacenter Edition auf dem Long-Term Servicing Channel (LTSC). Die Datacenter-Version hat keine Beschränkung der Anzahl der Container, die auf einem Worker-Knoten ausgeführt werden. Für weitere Informationen: https://docs.microsoft.com/en-us/ virtualization/windowscontainers/about/faq

Langfristiger Servicekanal (LTSC)

Früher als „Long-Term Servicing Branch“ bezeichnet, ist dies das Release-Modell, mit dem Sie bereits vertraut sind. Dabei wird alle 2-3 Jahre eine neue Hauptversion von Windows Server veröffentlicht. Benutzer haben Anspruch auf 5 Jahre Standardsupport und 5 Jahre erweiterten Support.

Lizenzen

Wenn Sie eine Amazon EC2 EC2-Instance mit einem Windows Server-basierten AMI starten, übernimmt Amazon die Lizenzkosten und die Einhaltung der Lizenzbestimmungen für Sie.

Protokollierung

Containerisierte Anwendungen leiten Anwendungsprotokolle in der Regel an STDOUT weiter. Die Container-Runtime fängt diese Protokolle ab und macht etwas mit ihnen — normalerweise schreibt sie in eine Datei. Wo diese Dateien gespeichert werden, hängt von der Laufzeit und Konfiguration des Containers ab.

Ein grundlegender Unterschied zu Windows-Pods besteht darin, dass sie kein STDOUT generieren. Sie können ausführen LogMonitor, um ETW (Event Tracing for Windows), Windows-Ereignisprotokolle und andere anwendungsspezifische Protokolle aus laufenden Windows-Containern abzurufen und formatierte Protokollausgaben an STDOUT weiterzuleiten. Diese Protokolle können dann mit Fluent-Bit oder Fluentd an Ihr gewünschtes Ziel wie Amazon gestreamt werden. CloudWatch

Der Mechanismus zur Protokollerfassung ruft STDOUT/STDERR Protokolle von Kubernetes-Pods ab. A DaemonSetist eine gängige Methode zum Sammeln von Protokollen aus Containern. Es gibt Ihnen die Möglichkeit, Protokolle routing/filtering/enrichment unabhängig von der Anwendung zu verwalten. Ein Fluentd DaemonSet kann verwendet werden, um diese Protokolle und alle anderen von der Anwendung generierten Protokolle an einen gewünschten Protokollaggregator zu streamen.

Ausführlichere Informationen zum Log-Streaming von Windows-Workloads zu finden Sie hier CloudWatch

Empfehlungen zur Protokollierung

Die allgemeinen Best Practices für die Protokollierung unterscheiden sich nicht vom Betrieb von Windows-Workloads in Kubernetes.

- Protokollieren Sie immer strukturierte Protokolleinträge (JSON/SYSLOG), was die Handhabung von Protokolleinträgen erleichtert, da es viele vorgefertigte Parser für solche strukturierten Formate gibt.
- Zentralisierung von Protokollen — spezielle Logging-Container können speziell zum Sammeln und Weiterleiten von Protokollnachrichten aus allen Containern an ein Ziel verwendet werden
- Halten Sie die Log-Ausführlichkeit niedrig, außer beim Debuggen. Ausführlichkeit stellt eine große stress für die Protokollinfrastruktur dar, und wichtige Ereignisse können im Lärm untergehen.
- Protokollieren Sie aus Gründen der Nachvollziehbarkeit immer die Anwendungsinformationen zusammen mit der Transaktions-/Anforderungs-ID. Kubernetes-Objekte tragen nicht den Namen der Anwendung, sodass beispielsweise ein Pod-Name beim Debuggen von Protokollen `windows-twryrqyw` möglicherweise keine Bedeutung hat. Dies hilft bei der Rückverfolgbarkeit und bei der Fehlerbehebung von Anwendungen mit Ihren aggregierten Protokollen.

  Wie Sie diese transaction/correlation IDs generieren, hängt vom Programmierkonstrukt ab. Ein sehr verbreitetes Muster ist jedoch die Verwendung eines Logging-Aspect/Interceptors, der MDC (Mapped Diagnostic Context) verwenden kann, um jeder eingehenden Anfrage eine eindeutige transaction/correlation ID zuzuweisen, etwa so:

```
import org.slf4j.MDC;
import java.util.UUID;
Class LoggingAspect { //interceptor

    @Before(value = "execution(* *.*(..))")
    func before(...) {
        transactionId = generateTransactionId();
        MDC.put(CORRELATION_ID, transactionId);
    }

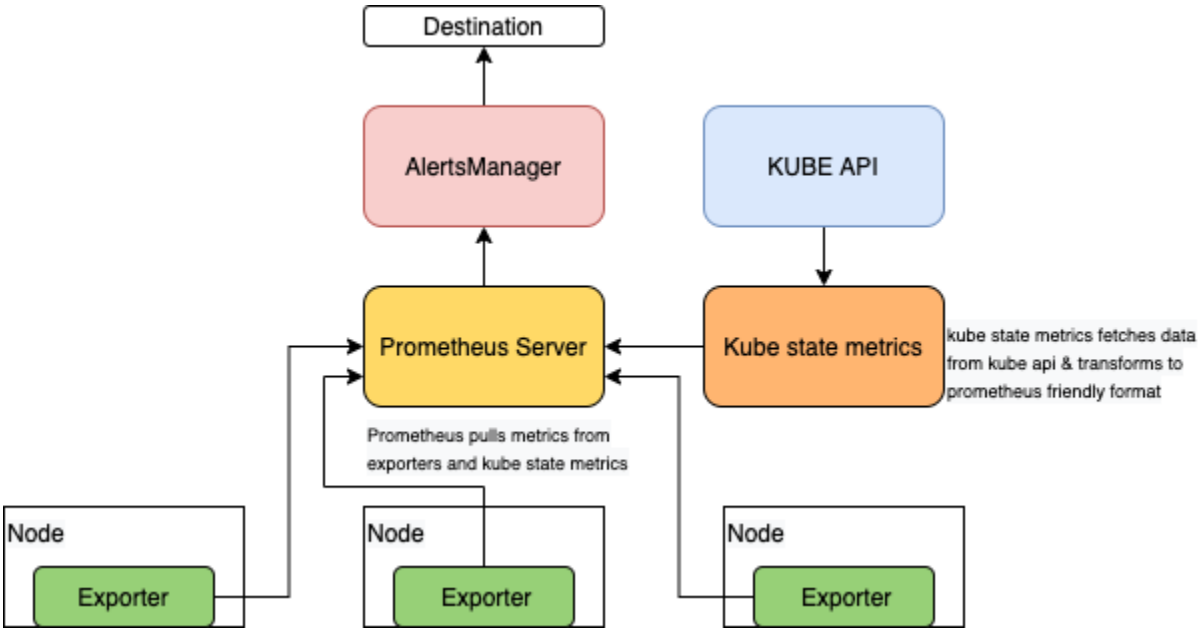
    func generateTransactionId() {
        return UUID.randomUUID().toString();
    }
}
```

Überwachen

Prometheus, ein abgeschlossenes CNCF-Projekt, ist mit Abstand das beliebteste Monitoring-System mit nativer Integration in Kubernetes. Prometheus sammelt Metriken rund um Container, Pods,

Knoten und Cluster. Darüber hinaus können Sie mit Prometheus Warnmeldungen programmieren AlertsManager , die Sie warnen, wenn etwas in Ihrem Cluster schief läuft. Prometheus speichert die Metrikdaten als Zeitreihendaten, die durch Metriknamen und key/value Paare identifiziert werden. Prometheus bietet eine Möglichkeit, Abfragen mit einer Sprache namens PromQL durchzuführen, was die Abkürzung für Prometheus Query Language ist.

Die allgemeine Architektur der Prometheus-Metrikerfassung ist unten dargestellt:



Prometheus verwendet einen Pull-Mechanismus und scannt Metriken von Zielen mithilfe von Exportern und von der Kubernetes-API mithilfe der Kube-State-Metriken. Das bedeutet, dass Anwendungen und Dienste einen HTTP (S) -Endpunkt bereitstellen müssen, der Metriken im Prometheus-Format enthält. Prometheus ruft dann gemäß seiner Konfiguration regelmäßig Metriken von diesen HTTP (S) -Endpunkten ab.

Mit einem Exporter können Sie Metriken von Drittanbietern als Metriken im Prometheus-Format verwenden. Ein Prometheus-Exporter wird normalerweise auf jedem Knoten bereitgestellt. Eine vollständige Liste der Exporteure finden Sie bei den Prometheus-Exporteuren. Der Node Exporter eignet sich zwar für den Export von Host-Hardware und Betriebssystem-Metriken für Linux-Knoten, funktioniert aber nicht für Windows-Knoten.

Wenn Sie in einem EKS-Cluster mit gemischten Knoten und Windows-Knoten das stabile Prometheus-Helmdiagramm verwenden, werden Sie ausgefallene Pods auf den Windows-Knoten sehen, da dieser Exporter nicht für Windows vorgesehen ist. Sie müssen den Windows-Worker-Pool separat behandeln und stattdessen den Windows-Exporter in der Windows-Worker-Knotengruppe installieren.

Um die Prometheus-Überwachung für Windows-Knoten einzurichten, müssen Sie den WMI-Exporter herunterladen und auf dem Windows-Server selbst installieren und dann die Ziele in der Scrape-Konfiguration der Prometheus-Konfigurationsdatei einrichten. Auf der Releases-Seite finden Sie alle verfügbaren .msi-Installationsprogramme mit den entsprechenden Funktionen und Bugfixes. Das Installationsprogramm richtet den windows_exporter als Windows-Dienst ein und erstellt eine Ausnahme in der Windows-Firewall. Wenn das Installationsprogramm ohne Parameter ausgeführt wird, wird der Exporter mit den Standardeinstellungen für aktivierte Collectors, Ports usw. ausgeführt.

Sie können sich den Abschnitt mit bewährten Methoden RuntimeClass zur Terminplanung dieses Handbuchs ansehen, in dem die Verwendung taints/tolerations oder selektive Bereitstellung des Node-Exporters nur für Linux-Knoten empfohlen wird, während der Windows-Exporter auf Windows-Knoten installiert wird, während Sie den Knoten booten oder ein Konfigurationsmanagement-Tool Ihrer Wahl verwenden (z. B. Chef, Ansible, SSM usw.).

Beachten Sie, dass im Gegensatz zu den Linux-Knoten, auf denen der Node Exporter als Daemonset installiert ist, der WMI-Exporter auf Windows-Knoten auf dem Host selbst installiert ist. Der Exporter exportiert Messwerte wie die CPU-Auslastung, den Arbeitsspeicher und die I/O Festplattennutzung und kann auch zur Überwachung von IIS-Websites und -Anwendungen, Netzwerkschnittstellen und -diensten verwendet werden.

Der windows_exporter macht standardmäßig alle Metriken von aktivierten Collectors verfügbar. Dies ist die empfohlene Methode zum Sammeln von Metriken, um Fehler zu vermeiden. Für fortgeschrittene Benutzer kann dem windows_exporter jedoch eine optionale Liste von Kollektoren zum Filtern von Metriken übergeben werden. Mit dem Parameter collect [] in der Prometheus-Konfiguration können Sie das tun.

Zu den Standardinstallationsschritten für Windows gehört das Herunterladen und Starten des Exporters als Dienst während des Bootstrapping-Vorgangs mit Argumenten, z. B. den Collectors, die Sie filtern möchten.

```
> 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"
```

Standardmäßig können die Metriken am Endpunkt /metrics auf Port 9182 gescrapt werden. Zu diesem Zeitpunkt kann Prometheus die Metriken verwenden, indem es der Prometheus-Konfiguration die folgende scrape_config hinzufügt

```
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', ...]
```

Prometheus Prometheus-Konfiguration wird neu geladen mit

```
> ps aux | grep prometheus
> kill HUP <PID>
```

Eine bessere und empfohlene Methode zum Hinzufügen von Zielen besteht darin, eine benutzerdefinierte Ressourcendefinition namens zu verwenden ServiceMonitor, die Teil des Prometheus-Operators ist], der die Definition für ein ServiceMonitor Objekt und einen Controller bereitstellt, der die We-Definition aktiviert und automatisch die ServiceMonitors erforderliche Prometheus-Konfiguration erstellt.

Der ServiceMonitor, der deklarativ festlegt, wie Gruppen von Kubernetes-Diensten überwacht werden sollen, wird verwendet, um eine Anwendung zu definieren, aus der Sie Metriken in Kubernetes auslesen möchten. Darin spezifizieren ServiceMonitor wir die Kubernetes-Labels, anhand derer der Betreiber den Kubernetes-Dienst identifizieren kann, der wiederum die Pods identifiziert, die wir überwachen möchten.

Um das zu nutzen ServiceMonitor, erstellen Sie ein Endpunktobjekt, das auf bestimmte Windows-Ziele verweist, einen Headless-Service und einen ServiceMontor für die Windows-Knoten.

```
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
```

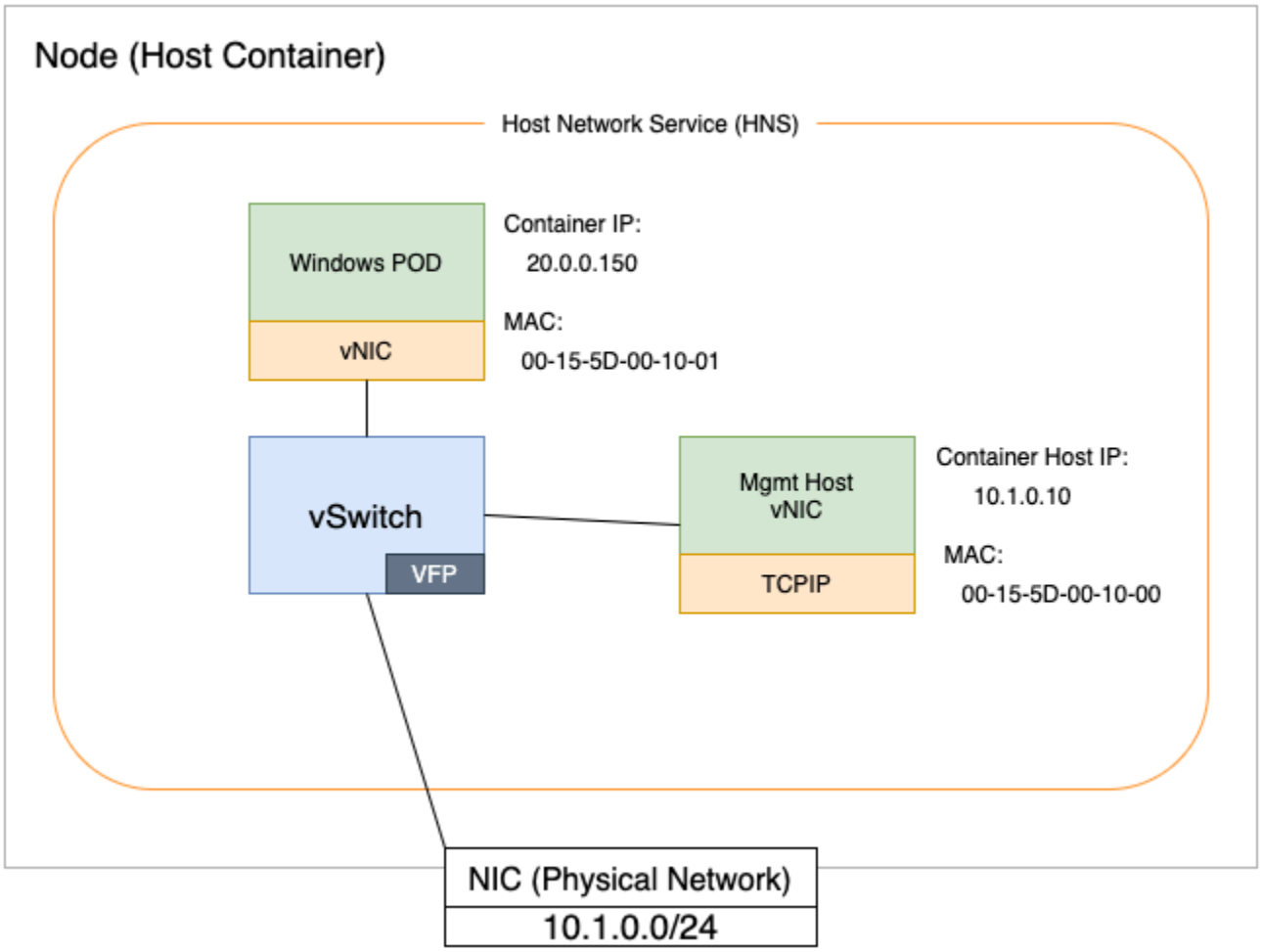
```
  namespace: monitoring
spec:
  endpoints:
  - interval: 30s
    port: http-metrics
  jobLabel: k8s-app
  namespaceSelector:
    matchNames:
    - kube-system
  selector:
    matchLabels:
      k8s-app: wmiexporter
```

Weitere Informationen zum Betreiber und zur Verwendung von finden Sie in der ServiceMonitor offiziellen Betreiberdokumentation. Beachten Sie, dass Prometheus die dynamische Zielerkennung mithilfe vieler Service Discovery-Optionen unterstützt.

Windows-Netzwerke

Überblick über Windows-Containernetzwerke

Windows-Container unterscheiden sich grundlegend von Linux-Containern. Linux-Container verwenden Linux-Konstrukte wie Namespaces, das Union-Dateisystem und Cgroups. Unter Windows werden diese Konstrukte aus dem Container des Host Compute Service (HCS) abstrahiert. HCS fungiert als API-Schicht, die sich über der Container-Implementierung unter Windows befindet. Windows-Container nutzen auch den Host Network Service (HNS), der die Netzwerktopologie auf einem Knoten definiert.



Aus Netzwerksicht sorgen HCS und HNS dafür, dass Windows-Container wie virtuelle Maschinen funktionieren. Beispielsweise hat jeder Container einen virtuellen Netzwerkadapter (vNIC), der mit einem virtuellen Hyper-V-Switch (vSwitch) verbunden ist, wie in der Abbildung oben dargestellt.

IP-Adressverwaltung

Ein Knoten in Amazon EKS verwendet sein Elastic Network Interface (ENI), um eine Verbindung zu einem AWS-VPC-Netzwerk herzustellen. Derzeit wird nur eine einzige ENI pro Windows-Worker-Knoten unterstützt. Die IP-Adressverwaltung für Windows-Knoten wird vom VPC Resource Controller durchgeführt, der auf der Steuerungsebene ausgeführt wird. Weitere Informationen zum Workflow für die IP-Adressverwaltung von Windows-Knoten finden Sie hier.

Die Anzahl der Pods, die ein Windows-Worker-Knoten unterstützen kann, hängt von der Größe des Knotens und der Anzahl der verfügbaren IPv4 Adressen ab. Sie können die auf dem Knoten verfügbare IPv4 Adresse wie folgt berechnen:

- Standardmäßig werden der ENI nur sekundäre IPv4 Adressen zugewiesen. In einem solchen Fall:

```
Total IPv4 addresses available for Pods = Number of supported IPv4 addresses in the
 primary interface - 1
```

  Wir ziehen eine Adresse von der Gesamtzahl ab, da IPv4 eine Adresse als primäre Adresse des ENI verwendet wird und daher den Pods nicht zugewiesen werden kann.

- Wenn der Cluster für eine hohe Pod-Dichte konfiguriert wurde, indem die Funktion zur Präfix-Delegierung aktiviert wurde, dann-

```
Total IPv4 addresses available for Pods = (Number of supported IPv4 addresses in the
 primary interface - 1) * 16
```

  Anstatt sekundäre IPv4 Adressen zuzuweisen, weist VPC Resource Controller hier zu, sodass die Gesamtzahl der verfügbaren IPv4 Adressen um das 16-fache erhöht wird. `/28 prefixes`

Mit der obigen Formel können wir die maximale Anzahl an Pods für einen Windows-Worker berechnen, der auf der Grundlage einer m5.large-Instanz geknotet ist, wie folgt:

- Standardmäßig, wenn es im sekundären IP-Modus ausgeführt wird

```
10 secondary IPv4 addresses per ENI - 1 = 9 available IPv4 addresses
```

- Bei Verwendung von `prefix delegation` -

```
(10 secondary IPv4 addresses per ENI - 1) * 16 = 144 available IPv4 addresses
```

Weitere Informationen darüber, wie viele IP-Adressen ein Instance-Typ unterstützen kann, finden Sie unter IP-Adressen pro Netzwerkschnittstelle pro Instance-Typ.

Ein weiterer wichtiger Aspekt ist der Fluss des Netzwerkverkehrs. Bei Windows besteht das Risiko, dass die Ports auf Knoten mit mehr als 100 Diensten ausgelastet werden. Wenn dieser Zustand eintritt, geben die Knoten Fehler mit der folgenden Meldung aus:

„Fehler bei der Erstellung der Richtlinie: Der hcnCreateLoad Balancer ist in Win32 fehlgeschlagen: Der angegebene Port ist bereits vorhanden.“

Um dieses Problem zu lösen, nutzen wir Direct Server Return (DSR). DSR ist eine Implementierung der asymmetrischen Netzwerklastverteilung. Mit anderen Worten, der Anfrage- und Antwortverkehr verwendet unterschiedliche Netzwerkpfade. Diese Funktion beschleunigt die Kommunikation zwischen Pods und verringert das Risiko einer Überlastung der Ports. Wir empfehlen daher, DSR auf Windows-Knoten zu aktivieren.

DSR ist in Windows Server SAC EKS Optimized standardmäßig aktiviert. AMIs Für Windows Server 2019 LTSC EKS Optimized AMIs müssen Sie es während der Instanzbereitstellung mithilfe des folgenden Skripts und mithilfe von Windows Server 2019 Full oder Core als AMIFamily in der NodeGroup aktivieren. `eksctl` Weitere Informationen finden Sie unter eksctl custom AMI.

```
nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
```

Um DSR in Windows Server 2019 und höher verwenden zu können, müssen Sie beim Start der Instanz die folgenden Kube-Proxy-Flags angeben. Sie können dies tun, indem Sie das Benutzerdatenskript anpassen, das der Startvorlage für selbstverwaltete Knotengruppen zugeordnet ist.

```
<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>
```

Die DSR-Aktivierung kann anhand der Anweisungen im Microsoft Networking-Blog und im Windows Containers on AWS Lab überprüft werden.

```
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}
```

Wenn es für Ihr Subnetz entscheidend ist, Ihre verfügbaren IPv4 Adressen beizubehalten und Verschwendung zu minimieren, wird generell empfohlen, den Präfix-Delegierungsmodus zu vermeiden, wie unter Präfixmodus für Windows — Wann zu vermeiden beschrieben. Wenn die Verwendung der Präfix-Delegierung weiterhin gewünscht wird, können Sie Maßnahmen ergreifen, um die IPv4 Adressnutzung in Ihrem Subnetz zu optimieren. Ausführliche Anweisungen zur Feinabstimmung des IPv4 Adressanforderungs- und Zuweisungsprozesses finden Sie unter Konfiguration der Parameter für die Präfixdelegierung. Die Anpassung dieser Konfigurationen kann Ihnen dabei helfen, ein Gleichgewicht zwischen der Beibehaltung von IPv4 Adressen und den Vorteilen der Präfix-Delegierung in Bezug auf die Pod-Dichte zu finden.

Wenn Sie die Standardeinstellung für die Zuweisung von sekundären IPv4 Adressen verwenden, gibt es derzeit keine unterstützten Konfigurationen, um zu manipulieren, wie der VPC Resource Controller Adressen anfordert und IPv4 zuweist. Genauer gesagt `warm-ip-target` werden sie nur für `minimum-ip-target` den Präfix-Delegierungsmodus unterstützt. Beachten Sie auch, dass der VPC Resource Controller im sekundären IP-Modus, abhängig von den verfügbaren IP-Adressen auf der Schnittstelle, in Ihrem Namen in der Regel 3 ungenutzte IPv4 Adressen auf dem Knoten zuweist, um ihn warm IPs zu halten und so die Startzeiten des Pods zu verkürzen. Wenn Sie die IP-Verschwendung ungenutzter warmer IP-Adressen minimieren möchten, könnten Sie versuchen, mehr Pods auf einem bestimmten Windows-Knoten einzuplanen, sodass Sie so viel IP-Adresskapazität der ENI wie möglich nutzen. Genauer gesagt könnten Sie verhindern, dass Warm

Unused verwendet wird, IPs wenn alle IP-Adressen auf dem ENI bereits von dem Node und den laufenden Pods verwendet werden. Eine weitere Problemumgehung, mit der Sie Einschränkungen bei der Verfügbarkeit von IP-Adressen in Ihren Subnetzen beheben können, könnte darin bestehen, die Größe Ihres Subnetzes zu erhöhen oder Ihre Windows-Knoten in eigene dedizierte Subnetze aufzuteilen.

Darüber hinaus ist es wichtig zu beachten, dass dies IPv6 derzeit auf Windows-Knoten nicht unterstützt wird.

Optionen für Container Network Interface (CNI)

Das AWSVPC CNI ist de facto das CNI-Plugin für Windows- und Linux-Worker-Knoten. Das AWSVPC CNI erfüllt zwar die Bedürfnisse vieler Kunden, dennoch kann es vorkommen, dass Sie Alternativen wie ein Overlay-Netzwerk in Betracht ziehen müssen, um eine IP-Erschöpfung zu vermeiden. In diesen Fällen kann das Calico CNI anstelle des CNI verwendet werden. AWSVPC Project Calico ist Open-Source-Software, die von Tigera entwickelt wurde. Diese Software enthält ein CNI, das mit EKS funktioniert. Anweisungen zur Installation von Calico CNI in EKS finden Sie auf der Installationsseite von Project Calico EKS.

Netzwerkrichtlinien

Es wird als bewährte Methode angesehen, vom Standardmodus der offenen Kommunikation zwischen Pods auf Ihrem Kubernetes-Cluster zur Zugriffsbeschränkung auf der Grundlage von Netzwerkrichtlinien zu wechseln. Das Open-Source-Projekt Calico bietet umfassende Unterstützung für Netzwerkrichtlinien, die sowohl mit Linux- als auch mit Windows-Knoten funktionieren. Diese Funktion ist separat und nicht von der Verwendung des Calico CNI abhängig. Wir empfehlen daher, Calico zu installieren und es für die Verwaltung von Netzwerkrichtlinien zu verwenden.

Anweisungen zur Installation von Calico in EKS finden Sie auf der Seite Calico auf Amazon EKS installieren.

Darüber hinaus gelten die Hinweise im Amazon EKS Best Practices Guide for Security — Network auch für EKS-Cluster mit Windows-Worker-Knoten, allerdings werden einige Funktionen wie „Sicherheitsgruppen für Pods“ derzeit nicht von Windows unterstützt.

Vermeidung von OOM-Fehlern

Windows hat keinen out-of-memory Prozesskiller wie Linux. Windows behandelt alle Speicherzuweisungen im Benutzermodus immer als virtuell, und Auslagerungsdateien sind

obligatorisch. Der Nettoeffekt besteht darin, dass Windows nicht auf die gleiche Weise auf Speicherprobleme zugreift wie Linux. Prozesse werden auf die Festplatte ausgelagert, anstatt dass sie aufgrund unzureichenden Speichers (OOM) beendet werden. Wenn zu viel Arbeitsspeicher zur Verfügung steht und der gesamte physische Speicher erschöpft ist, kann das Auslagern die Leistung beeinträchtigen.

System- und Kubelet-Speicher reservieren

Anders als bei Linux, wo **`--kubelet-reserve`** die Ressourcenreservierung für Kubernetes-System-Daemons wie Kubelet, Container-Runtime usw. **`--system-reserve`** erfasst und die Ressourcenreservierung für Betriebssystem-Daemons wie sshd, udev usw. erfasst wird. Unter Windows erfassen und setzen diese Flags keine Speicherlimits für Kubelet oder Prozesse, die auf dem Knoten ausgeführt werden.

Sie können diese Flags jedoch kombinieren, um die Kapazität auf dem Knoten mit dem Speicherressourcenlimit für das Pod-Manifest zu reduzieren, um die Speicherzuweisung pro Pod zu steuern. NodeAllocatable Mit dieser Strategie haben Sie eine bessere Kontrolle über die Speicherzuweisung sowie einen Mechanismus zur Minimierung out-of-memory (OOM) auf Windows-Knoten.

Auf Windows-Knoten empfiehlt es sich, mindestens 2 GB Arbeitsspeicher für das Betriebssystem und den Prozess zu reservieren. Verwenden Sie `--kubelet-reserve` and/or `--system-reserve`, um zu reduzieren NodeAllocatable.

Folgen Sie der Dokumentation zu selbstverwalteten Windows-Knoten von Amazon EKS und verwenden Sie die CloudFormation Vorlage, um eine neue Windows-Knotengruppe mit Anpassungen an die Kubelet-Konfiguration zu starten. Das CloudFormation hat ein Element namens`BootstrapArguments`, das dasselbe ist wie. `KubeletExtraArgs` Verwenden Sie es mit den folgenden Flags und Werten:

```
--kube-reserved memory=0.5Gi,ephemeral-storage=1Gi --system-
reserved memory=1.5Gi,ephemeral-storage=1Gi --eviction-hard
 memory.available<200Mi,nodefs.available<10%"
```

Wenn eksctl das Bereitstellungstool ist, lesen Sie in der folgenden Dokumentation nach, wie Sie die Kubelet-Konfiguration anpassen können: https://eksctl. io/usage/customizing-das-Kubelet/

Speicheranforderungen für Windows-Container

Gemäß der Microsoft-Dokumentation benötigt ein Windows Server-Basisimage für NANO mindestens 30 MB, wohingegen Server Core 45 MB benötigt. Diese Zahlen steigen, wenn Sie Windows-Komponenten wie das.NET Framework, Webdienste wie IIS und Anwendungen hinzufügen.

Es ist wichtig, dass Sie wissen, wie viel Speicherplatz Ihr Windows-Container-Image, d. h. das Basis-Image und seine Anwendungsebenen, mindestens benötigt, und es resources/requests in der Pod-Spezifikation als Container festlegen. Sie sollten auch ein Limit festlegen, um zu verhindern, dass Pods im Falle eines Anwendungsproblems den gesamten verfügbaren Knotenspeicher verbrauchen.

Wenn im folgenden Beispiel der Kubernetes-Scheduler versucht, einen Pod auf einem Knoten zu platzieren, werden die Anfragen des Pods verwendet, um zu ermitteln, welcher Knoten über ausreichend Ressourcen für die Planung verfügt.

```
spec:
 - name: iis
   image: mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019
   resources:
     limits:
       cpu: 1
       memory: 800Mi
     requests:
       cpu: .1
       memory: 128Mi
```

Schlussfolgerung

Die Verwendung dieses Ansatzes minimiert das Risiko einer Speichererschöpfung, verhindert sie jedoch nicht. Mithilfe von Amazon CloudWatch Metrics können Sie Warnmeldungen und Abhilfemaßnahmen für den Fall einrichten, dass der Speicherplatz knapp wird.

Patchen von Windows-Servern und Containern

Das Patchen von Windows Server ist eine Standardverwaltungsaufgabe für Windows-Administratoren. Dies kann mit verschiedenen Tools wie Amazon System Manager - Patch Manager, WSUS, System Center Configuration Manager und vielen anderen erreicht werden. Windows-Knoten in einem Amazon EKS-Cluster sollten jedoch nicht wie normale Windows-Server behandelt werden.

Sie sollten als unveränderliche Server behandelt werden. Einfach ausgedrückt, vermeiden Sie die Aktualisierung eines vorhandenen Knotens, sondern starten Sie einfach einen neuen, der auf einem neuen aktualisierten AMI basiert.

Mit EC2 Image Builder können Sie den AMIs Build automatisieren, indem Sie Rezepte erstellen und Komponenten hinzufügen.

Das folgende Beispiel zeigt Komponenten, bei denen es sich um bereits vorhandene Komponenten handeln kann, die von AWS (von Amazon verwaltet) erstellt wurden, sowie um die Komponenten, die Sie erstellen (Owned by me). Achten Sie besonders auf die von Amazon verwaltete Komponente namens update-windows, mit der Windows Server aktualisiert wird, bevor das AMI über die EC2 Image Builder Builder-Pipeline generiert wird.

Associated components

Search components

1

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

Mit EC2 Image Builder können Sie AMIs auf Basis von Amazon Managed Public erstellen AMIs und sie an Ihre Geschäftsanforderungen anpassen. Sie können diese dann AMIs mit Launch Templates verknüpfen, sodass Sie ein neues AMI mit der von der EKS Nodegroup erstellten Auto Scaling Group verknüpfen können. Sobald dieser Vorgang abgeschlossen ist, können Sie damit beginnen, die vorhandenen Windows-Knoten zu beenden, und neue Knoten werden auf der Grundlage des neuen aktualisierten AMI gestartet.

Windows-Images übertragen und ziehen

Amazon veröffentlicht EKS-Optimierungen AMIs , die zwei zwischengespeicherte Windows-Container-Images enthalten.

```
mcr.microsoft.com/windows/servercore
mcr.microsoft.com/windows/nanoserver
```

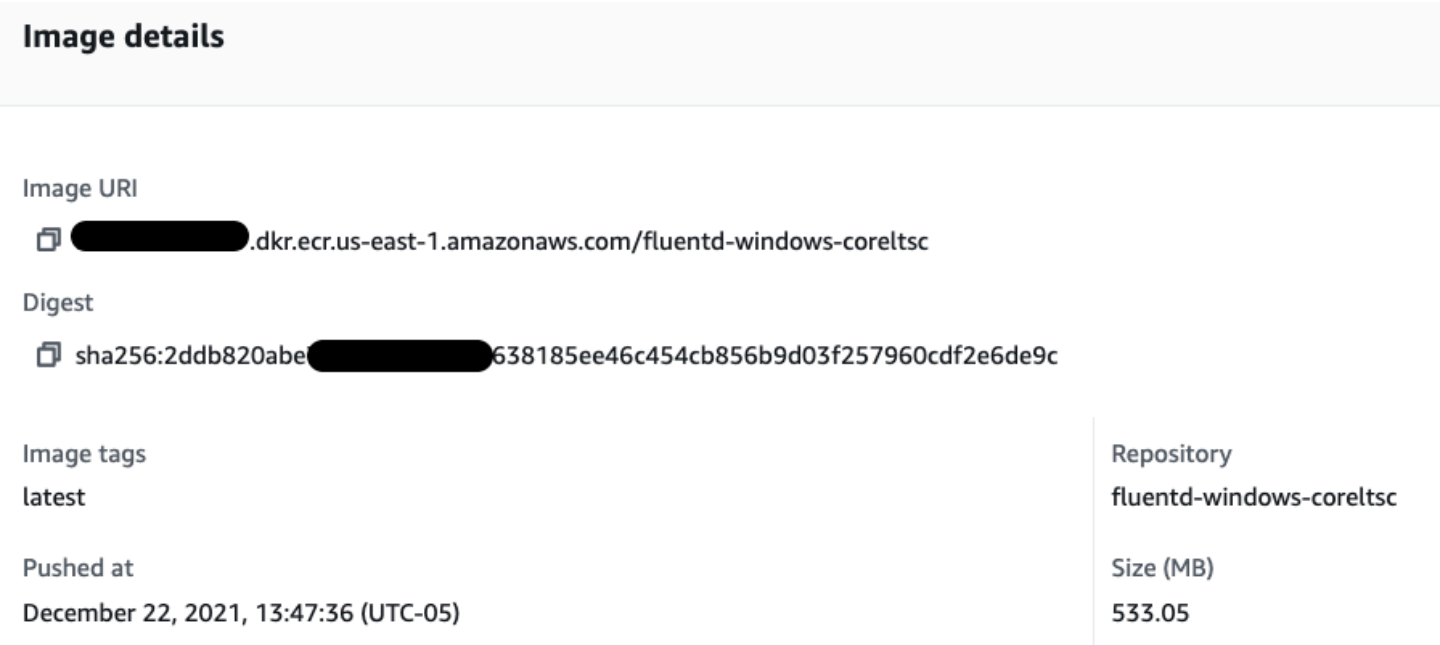
```
REPOSITORY                                                      TAG                       IMAGE ID       CREATED         SIZE
mcr.microsoft.com/windows/servercore                            ltsc2019                  617dd5ead5b8   6 weeks ago     5.74GB
mcr.microsoft.com/windows/nanoserver                            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
```

Zwischengespeicherte Bilder werden nach den Updates auf dem Hauptbetriebssystem aktualisiert. Wenn Microsoft ein neues Windows-Update veröffentlicht, das sich direkt auf das Windows-Container-Basisimage auswirkt, wird das Update als normales Windows Update auf dem Hauptbetriebssystem gestartet. Die Beibehaltung der Umgebung up-to-date bietet eine sicherere Umgebung auf Knoten- und Container-Ebene.

Die Größe eines Windows-Container-Images beeinflusst push/pull Operationen, was zu langsamen Startzeiten von Containern führen kann. Durch das Zwischenspeichern von Windows-Container-Images können die teuren I/O Operationen (Dateiextraktion) bei der AMI-Build-Erstellung statt beim Start des Containers ausgeführt werden. Dadurch werden alle erforderlichen Bildebenen auf dem AMI extrahiert und sind sofort einsatzbereit. Dadurch wird die Zeit, in der ein Windows-Container gestartet wird und Traffic annehmen kann, beschleunigt. Während eines Push-Vorgangs werden nur die Ebenen, aus denen Ihr Bild besteht, in das Repository hochgeladen.

Das folgende Beispiel zeigt, dass die Bilder von fluentd-windows-sac2004 auf Amazon ECR nur 390,18 MB groß sind. Dies ist die Menge an Uploads, die während des Push-Vorgangs durchgeführt wurden.

Das folgende Beispiel zeigt ein flüssiges Windows-LTSC-Image, das in ein Amazon ECR-Repository übertragen wurde. Die Größe der in ECR gespeicherten Ebene beträgt 533,05 MB.



Die folgende Ausgabe von`docker image ls`, die Größe des Fluentd v1.14-windows-ltsc2019-1, beträgt 6,96 GB auf der Festplatte, aber das bedeutet nicht, dass diese Datenmenge heruntergeladen und extrahiert wurde.

In der Praxis werden während des Pull-Vorgangs nur die komprimierten 533,05 MB heruntergeladen und extrahiert.

```
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
```

Die Größenspalte zeigt die Gesamtgröße des Bilds, 6,96 GB. Aufschlüsselung:

- Windows Server Core 2019 LTSC-Basisimage = 5,74 GB
- Fluentd Unkomprimiertes Basisimage = 6,96 GB
- Unterschied auf der Festplatte = 1,2 GB
- Fluentd komprimiertes Endbild ECR = 533,05 MB

Das Basis-Image ist bereits auf der lokalen Festplatte vorhanden, sodass sich die Gesamtmenge auf der Festplatte auf 1,2 GB erhöht. Machen Sie sich keine Sorgen, wenn Sie das nächste Mal die Menge von GBs in der Größenspalte sehen. Wahrscheinlich sind mehr als 70% bereits als zwischengespeichertes Container-Image auf der Festplatte gespeichert.

Referenz

Verkürzung der Startzeiten von Windows-Containern mit EC2 Image Builder und Image-Cache-Strategie

Ausführung heterogener Workloads

Kubernetes unterstützt heterogene Cluster, bei denen Sie eine Mischung aus Linux- und Windows-Knoten im selben Cluster haben können. Innerhalb dieses Clusters können Sie eine Mischung aus Pods haben, die unter Linux ausgeführt werden, und Pods, die unter Windows ausgeführt werden. Sie können sogar mehrere Versionen von Windows im selben Cluster ausführen. Es gibt jedoch mehrere Faktoren (wie unten erwähnt), die bei dieser Entscheidung berücksichtigt werden müssen.

Bewährte Methoden PODs für die Zuweisung zu Knoten

Um Linux- und Windows-Workloads auf ihren jeweiligen betriebssystemspezifischen Knoten zu belassen, müssen Sie eine Kombination aus Knotenselektoren und Fehler/Toleranzen verwenden. Das Hauptziel der Planung von Workloads in einer heterogenen Umgebung besteht darin, zu verhindern, dass die Kompatibilität vorhandener Linux-Workloads beeinträchtigt wird.

Sicherstellen, dass betriebssystemspezifische Workloads auf dem entsprechenden Container-Host landen

Benutzer können mithilfe von NodeSelectors sicherstellen, dass Windows-Container auf dem entsprechenden Host geplant werden können. Alle Kubernetes-Knoten haben heute die folgenden Standardbezeichnungen:

```
kubernetes.io/os = [windows|linux]
kubernetes.io/arch = [amd64|arm64|...]
```

Wenn eine Pod-Spezifikation keinen NodeSelector-Like enthält, kann der Pod auf einem beliebigen Host`"kubernetes.io/os": windows`, Windows oder Linux, geplant werden. Dies kann problematisch sein, da ein Windows-Container nur unter Windows und ein Linux-Container nur unter Linux ausgeführt werden kann.

In Unternehmensumgebungen ist es nicht ungewöhnlich, eine große Anzahl bereits vorhandener Implementierungen für Linux-Container sowie ein Ökosystem von off-the-shelf Konfigurationen wie Helm-Charts zu haben. In diesen Situationen zögern Sie möglicherweise, Änderungen an den NodeSelectors einer Bereitstellung vorzunehmen. Die Alternative besteht darin, Taints zu verwenden.

Beispiel: `--register-with-taints='os=windows:NoSchedule'`

Wenn Sie EKS verwenden, bietet eksctl Möglichkeiten, Taints über ClusterConfig anzuwenden:

```
NodeGroups:
  - name: windows-ng
    amiFamily: WindowsServer2022FullContainer
    ...
    labels:
      nodeclass: windows2022
    taints:
      os: "windows:NoSchedule"
```

Wenn allen Windows-Knoten ein Taint hinzugefügt wird, plant der Scheduler keine Pods auf diesen Knoten, es sei denn, sie tolerieren den Taint. Beispiel für ein Pod-Manifest:

```
nodeSelector:
    kubernetes.io/os: windows
tolerations:
    - key: "os"
      operator: "Equal"
      value: "windows"
      effect: "NoSchedule"
```

Umgang mit mehreren Windows-Builds im selben Cluster

Das von jedem Pod verwendete Windows-Container-Basis-Image muss derselben Kernel-Build-Version wie der Knoten entsprechen. Wenn Sie mehrere Windows Server-Builds im selben Cluster verwenden möchten, sollten Sie zusätzliche Knotenbezeichnungen oder NodeSelectors einrichten oder eine Bezeichnung namens Windows-Build nutzen.

Kubernetes 1.17 fügt automatisch ein neues Label node.kubernetes.io/windows-build hinzu, um die Verwaltung mehrerer Windows-Builds im selben Cluster zu vereinfachen. Wenn Sie eine ältere Version ausführen, wird empfohlen, dieses Label manuell zu Windows-Knoten hinzuzufügen.

Diese Bezeichnung gibt die Haupt-, Minor- und Buildnummer von Windows wieder, die aus Kompatibilitätsgründen übereinstimmen müssen. Im Folgenden sind die Werte aufgeführt, die heute für jede Windows Server-Version verwendet werden.

Es ist wichtig zu beachten, dass Windows Server auf den Long-Term Servicing Channel (LTSC) als primären Release-Channel umsteigt. Der Windows Server Semi-Annual Channel (SAC) wurde am 9. August 2022 eingestellt. Es wird keine future SAC-Versionen von Windows Server geben.

Product Name (Produktname)	Buildnummer (n)
Voller Server 2022 LTSC	10.0.20348
Serverkern 2019 LTSC	10.0.17763

Es ist möglich, die Build-Version des Betriebssystems mit dem folgenden Befehl zu überprüfen:

```
kubectl get nodes -o wide
```

Die Ausgabe von KERNEL-VERSION entspricht der Build-Version des Windows-Betriebssystems.

```
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
```

Das folgende Beispiel wendet einen zusätzlichen NodeSelector auf das Pod-Manifest an, um der richtigen Windows-Build-Version zu entsprechen, wenn verschiedene Betriebssystemversionen für Windows-Knotengruppen ausgeführt werden.

```
nodeSelector:
    kubernetes.io/os: windows
    node.kubernetes.io/windows-build: '10.0.20348'
tolerations:
    - key: "os"
    operator: "Equal"
    value: "windows"
    effect: "NoSchedule"
```

Vereinfachung und Tolerierung NodeSelector in Pod-Manifesten mit RuntimeClass

Sie können Taints and Tolerations auch verwenden RuntimeClass , um den Prozess der Verwendung von Taints and Tolerations zu vereinfachen. Dies kann erreicht werden, indem ein RuntimeClass Objekt erstellt wird, das zur Verkapselung dieser Fehler und Toleranzen verwendet wird.

Erstellen Sie ein, RuntimeClass indem Sie das folgende Manifest ausführen:

```
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"
```

Sobald die Runtime-Klasse erstellt ist, weisen Sie sie als Spezifikation im Pod-Manifest zu:

```
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 für verwaltete Knotengruppen

Um Kunden dabei zu helfen, ihre Windows-Anwendungen effizienter auszuführen, hat AWS am 15. Dezember 2022 den Support für Amazon EKS Managed Node Group (MNG) für Windows-Container eingeführt. Um die Abstimmung der Betriebsteams zu erleichtern, MNGs werden Windows mit denselben Workflows und Tools wie Linux aktiviert. MNGs Die Voll- und Kernversionen der AMI-Familie (Amazon Machine Image) von Windows Server 2019 und 2022 werden unterstützt.

Die folgenden AMI-Familien werden für Managed Node Groups (MNG) unterstützt.

AMI-Familie
Windows_Core_2019_x86_64
Windows_Full_2019_x86_64
Windows_Core_2022_x86_64
Windows_FULL_2022_x86_64

Zusätzliche Dokumentationen

Offizielle AWS-Dokumentation: https://docs.aws.amazon.com/eks/ latest/userguide/windows - support.html

Um besser zu verstehen, wie Pod Networking (CNI) funktioniert, schauen Sie sich den folgenden Link an: -networking.html https://docs.aws.amazon.com/eks/ latest/userguide/pod

AWS-Blog zur Bereitstellung von Managed Node Group für Windows auf EKS: https:// aws.amazon.com/blogs/ containers/ -/deploying-amazon-eks-windowsmanaged-node-groups

Sicherheitskontexte für Pods

Pod-Sicherheitsrichtlinien (PSP) und Pod-Sicherheitsstandards (PSS) sind zwei Hauptmethoden zur Durchsetzung der Sicherheit in Kubernetes. Beachten Sie, dass diese Version ab Kubernetes v1.21 veraltet PodSecurityPolicy ist und in Version 1.25 entfernt wird. Pod Security Standard (PSS) ist der von Kubernetes empfohlene Ansatz zur zukünftigen Durchsetzung der Sicherheit.

Eine Pod Security Policy (PSP) ist eine native Lösung in Kubernetes zur Implementierung von Sicherheitsrichtlinien. PSP ist eine Ressource auf Clusterebene, die sicherheitsrelevante Aspekte der Pod-Spezifikation steuert. Mithilfe der Pod-Sicherheitsrichtlinie können Sie eine Reihe von Bedingungen definieren, die Pods erfüllen müssen, um vom Cluster akzeptiert zu werden. Die PSP-Funktion ist seit den Anfängen von Kubernetes verfügbar und soll verhindern, dass falsch konfigurierte Pods auf einem bestimmten Cluster erstellt werden.

Weitere Informationen zu den Sicherheitsrichtlinien für Pods finden Sie in der Kubernetes-Dokumentation. Gemäß der Richtlinie zur Einstellung von Kubernetes werden ältere Versionen neun Monate nach dem Auslaufen der Funktion nicht mehr unterstützt.

Andererseits werden die Pod Security Standards (PSS), der empfohlene Sicherheitsansatz, der in der Regel mithilfe von Sicherheitskontexten implementiert wird, als Teil der Pod- und Container-Spezifikationen im Pod-Manifest definiert. PSS ist der offizielle Standard, den das Kubernetes-Projektteam definiert hat, um die sicherheitsrelevanten Best Practices für Pods zu berücksichtigen. Er definiert Richtlinien wie Basisrichtlinien (minimal restriktiv, Standard), privilegiert (nicht restriktiv) und eingeschränkt (am restriktivsten).

Wir empfehlen, mit dem Basisprofil zu beginnen. Das PSS-Basisprofil bietet ein solides Gleichgewicht zwischen Sicherheit und potenziellen Problemen, erfordert eine minimale Liste von Ausnahmen und ist ein guter Ausgangspunkt für die Workload-Sicherheit. Wenn Sie derzeit PSS verwenden, empfehlen PSPs wir Ihnen, zu PSS zu wechseln. Weitere Informationen zu den PSS-Richtlinien finden Sie in der Kubernetes-Dokumentation. Diese Richtlinien können mit verschiedenen Tools durchgesetzt werden, darunter denen von OPA und Kyverno. Zum Beispiel bietet Kyverno hier die vollständige Sammlung der PSS-Richtlinien.

Die Einstellungen für den Sicherheitskontext ermöglichen es unter anderem, Berechtigungen für ausgewählte Prozesse zu vergeben, Programmprofile zu verwenden, um Funktionen auf einzelne Programme zu beschränken, die Eskalation von Rechten zu ermöglichen, Systemaufrufe zu filtern.

Windows-Pods in Kubernetes haben einige Einschränkungen und Unterscheidungsmerkmale von standardmäßigen Linux-basierten Workloads, wenn es um Sicherheitskontexte geht.

Windows verwendet ein Job-Objekt pro Container mit einem System-Namespace-Filter, um alle Prozesse in einem Container zu enthalten und eine logische Isolierung vom Host zu gewährleisten. Ohne die Namespace-Filterung ist es nicht möglich, einen Windows-Container auszuführen. Dies bedeutet, dass Systemberechtigungen nicht im Kontext des Hosts geltend gemacht werden können und daher privilegierte Container unter Windows nicht verfügbar sind.

Im Folgenden `windowsOptions` sind die einzigen dokumentierten Windows-Sicherheitskontext-Optionen aufgeführt, während es sich bei den übrigen um allgemeine Sicherheitskontext-Optionen handelt

Eine Liste der Sicherheitskontext-Attribute, die in Windows und Linux unterstützt werden, finden Sie in der offiziellen Dokumentation hier.

Die Pod-spezifischen Einstellungen werden auf alle Container angewendet. Falls nicht angegeben, PodSecurityContext werden die Optionen von verwendet. Wenn SecurityContext sowohl in als auch angegeben PodSecurityContext, hat der in angegebene Wert SecurityContext Vorrang.

Beispielsweise entspricht die Einstellung runAsUser Name für Pods und Container, bei der es sich um eine Windows-Option handelt, grob der Linux-spezifischen runAsUser Einstellung, und im folgenden Manifest wird der Pod-spezifische Sicherheitskontext auf alle Container angewendet

```
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
```

Im Folgenden hat der Sicherheitskontext auf Container-Ebene Vorrang vor dem Sicherheitskontext auf Pod-Ebene.

```
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
```

Beispiele für zulässige Werte für das Feld runAsUser Name: ContainerAdministrator, ContainerUser, NT AUTHORITY\ NETWORK SERVICE, NT AUTHORITY\ LOCAL SERVICE

Im Allgemeinen ist es eine gute Idee, Ihre Container ContainerUser für Windows-Pods auszuführen. Die Benutzer werden nicht vom Container und vom Host gemeinsam genutzt, sie haben ContainerAdministrator jedoch zusätzliche Rechte im Container. Beachten Sie, dass es Einschränkungen bei Benutzernamen gibt, die Sie beachten sollten.

Ein gutes Beispiel für die Verwendung ContainerAdministrator ist die Einstellung von PATH. Sie können dazu die USER-Direktive wie folgt verwenden:

```
USER ContainerAdministrator
RUN setx /M PATH "%PATH%;C:/your/path"
USER ContainerUser
```

Beachten Sie auch, dass Geheimnisse im Klartext auf das Volume des Nodes geschrieben werden (im Vergleich zu tmpfs/in-memory unter Linux). Das bedeutet, dass Sie zwei Dinge tun müssen

- Verwenden Sie die Datei ACLs , um den Speicherort der geheimen Datei zu sichern
- Verwenden Sie Verschlüsselung auf Volume-Ebene mit BitLocker

Persistente Speicheroptionen

Was ist ein In-Tree- oder ein out-of-tree Volume-Plugin?

Vor der Einführung des Container Storage Interface (CSI) befanden sich alle Volume-Plugins in der Baumstruktur, d. h. sie wurden mit den Kern-Binärdateien von Kubernetes erstellt, verknüpft, kompiliert und ausgeliefert und erweiterten die Kern-Kubernetes-API. Das bedeutete, dass für das Hinzufügen eines neuen Speichersystems zu Kubernetes (ein Volume-Plugin) Code in das Code-Repository von Kubernetes eingecheckt werden musste.

Out-of-tree Volume-Plugins werden unabhängig von der Kubernetes-Codebasis entwickelt und als Erweiterungen auf Kubernetes-Clustern bereitgestellt (installiert). Dies gibt Anbietern die Möglichkeit, Treiber zu aktualisieren out-of-band, d. h. unabhängig vom Kubernetes-Release-Zyklus. Dies ist weitgehend möglich, da Kubernetes eine Speicherschnittstelle oder CSI entwickelt hat, die Anbietern eine Standardschnittstelle für K8s bietet.

Weitere Informationen zu den Speicherklassen und CSI-Treibern von Amazon Elastic Kubernetes Services (EKS) finden Sie unter .html https://docs.aws.amazon.com/eks/ latest/userguide/storage

In-Tree-Volume-Plugin für Windows

Kubernetes-Volumes ermöglichen die Bereitstellung von Anwendungen mit Anforderungen an die Datenpersistenz auf Kubernetes. Die Verwaltung persistenter Volumes besteht aus provisioning/de-provisioning/resizing of volumes, attaching/detaching a volume to/from a Kubernetes node, and mounting/dismounting a volume to/from einzelnen Containern in einem Pod. Der Code für die Implementierung dieser Volume-Management-Aktionen für ein bestimmtes Speicher-Back-End oder ein bestimmtes Speicherprotokoll wird in Form eines Kubernetes-Volume-Plug-ins (In-Tree Volume Plugins) ausgeliefert. Auf Amazon Elastic Kubernetes Services (EKS) werden die folgenden Klassen von Kubernetes-Volume-Plug-ins unter Windows unterstützt:

Volume-Plugin im Baum: Store awsElasticBlock

Um das In-Tree Volume Plugin auf Windows-Knoten verwenden zu können, ist es notwendig, ein zusätzliches Plugin zu erstellen, um NTFS als FSType StorageClass zu verwenden. Auf EKS StorageClass verwendet die Standardeinstellung ext4 als Standard-FSType.

A StorageClass bietet Administratoren die Möglichkeit, die von ihnen angebotenen „Speicherklassen“ zu beschreiben. Verschiedene Klassen können quality-of-service Ebenen, Backup-Richtlinien oder beliebigen Richtlinien zugeordnet werden, die von den Cluster-Administratoren festgelegt werden. Kubernetes hat keine Meinung darüber, wofür Klassen stehen. Dieses Konzept wird in anderen Speichersystemen manchmal als „Profile“ bezeichnet.

Sie können es überprüfen, indem Sie den folgenden Befehl ausführen:

```
kubectl describe storageclass gp2
```

Ausgabe:

```
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>
```

Verwenden Sie das folgende Manifest StorageClass , um das neue zu erstellen, das NTFS unterstützt:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gp2-windows
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  fsType: ntfs
volumeBindingMode: WaitForFirstConsumer
```

Erstellen Sie das, StorageClass indem Sie den folgenden Befehl ausführen:

```
kubectl apply -f NTFSStorageClass.yaml
```

Der nächste Schritt besteht darin, einen Persistent Volume Claim (PVC) zu erstellen.

Ein PersistentVolume (PV) ist ein Speicherelement im Cluster, das von einem Administrator oder dynamisch mithilfe von PVC bereitgestellt wurde. Es ist eine Ressource im Cluster, genauso wie ein Knoten eine Clusterressource ist. Dieses API-Objekt erfasst die Details der Implementierung des Speichers, sei es NFS, iSCSI oder ein cloud-provider-specific Speichersystem.

A PersistentVolumeClaim (PVC) ist eine Speicheranforderung durch einen Benutzer. Für Ansprüche können bestimmte Größen und Zugriffsmodi angefordert werden (z. B. können sie montiert ReadWriteOnce werden ReadOnlyMany oder ReadWriteMany).

Benutzer benötigen PersistentVolumes unterschiedliche Eigenschaften, wie z. B. Leistung, für unterschiedliche Anwendungsfälle. Clusteradministratoren müssen in der Lage sein, eine Vielzahl von Optionen anzubieten PersistentVolumes , die sich nicht nur in Bezug auf Größe und Zugriffsmodi unterscheiden, ohne die Benutzer mit den Einzelheiten der Implementierung dieser Volumes vertraut zu machen. Für diese Bedürfnisse gibt es die entsprechende StorageClass Ressource.

Im folgenden Beispiel wurde das PVC innerhalb der Namespace-Fenster erstellt.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ebs-windows-pv-claim
  namespace: windows
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gp2-windows
  resources:
    requests:
      storage: 1Gi
```

Erstellen Sie das PVC, indem Sie den folgenden Befehl ausführen:

```
kubectl apply -f persistent-volume-claim.yaml
```

Das folgende Manifest erstellt einen Windows-Pod, richtet das VolumeMount AS ein `C:\Data` und verwendet das PVC als angeschlossenen Speicher`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'
```

Testen Sie die Ergebnisse, indem Sie wie folgt auf den Windows-Pod zugreifen PowerShell:

```
kubectl exec -it podname powershell -n windows
```

Führen Sie im Windows-Pod Folgendes aus: `ls`

Ausgabe:

```
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
```

Das Datenverzeichnis wird vom EBS-Volume bereitgestellt.

Out-of-tree für Windows

Code, der mit CSI-Plug-ins verknüpft ist, wird als out-of-tree Skripte und Binärdateien ausgeliefert, die normalerweise als Container-Images verteilt und mithilfe von Standard-Kubernetes-Konstrukten wie und bereitgestellt werden. DaemonSets StatefulSets CSI-Plugins verarbeiten eine Vielzahl von Volume-Management-Aktionen in Kubernetes. CSI-Plugins bestehen in der Regel aus Node-Plugins (die auf jedem Knoten als solches ausgeführt werden DaemonSet) und Controller-Plugins.

CSI-Knoten-Plugins (insbesondere solche, die sich auf persistente Volumes beziehen, die entweder als Blockgeräte oder über ein gemeinsam genutztes Dateisystem verfügbar sind) müssen verschiedene privilegierte Operationen wie das Scannen von Festplattengeräten, das Mounten von Dateisystemen usw. ausführen. Diese Operationen unterscheiden sich je nach Host-Betriebssystem. Für Linux-Worker-Knoten werden containerisierte CSI-Knoten-Plugins in der Regel als privilegierte Container bereitgestellt. Für Windows-Worker-Knoten werden privilegierte Operationen für containerisierte CSI-Knoten-Plugins mithilfe von csi-proxy unterstützt, einer von der Community verwalteten, eigenständigen Binärdatei, die auf jedem Windows-Knoten vorinstalliert werden muss.

Das Amazon EKS-optimierte Windows-AMI umfasst ab April 2022 CSI-Proxy. Kunden können den SMB CSI-Treiber auf Windows-Knoten verwenden, um auf Amazon FSx for Windows File Server, Amazon FSx for NetApp ONTAP SMB Shares und and/or AWS Storage Gateway — File Gateway zuzugreifen.

Der folgende Blog enthält Implementierungsdetails zur Einrichtung des SMB CSI-Treibers für die Verwendung von Amazon FSx for Windows File Server als persistenten Speicher für Windows Pods.

Amazon FSx für Windows-Dateiserver

Eine Option besteht darin, Amazon FSx for Windows File Server über eine SMB-Funktion namens SMB Global Mapping zu verwenden, die es ermöglicht, eine SMB-Freigabe auf dem Host zu mounten und dann Verzeichnisse auf dieser Freigabe an einen Container zu übergeben. Der Container muss nicht mit einem bestimmten Server, einer bestimmten Freigabe, einem Benutzernamen oder einem Passwort konfiguriert werden — das alles wird stattdessen auf dem Host abgewickelt. Der Container funktioniert genauso, als ob er lokalen Speicher hätte.

> Das SMB Global Mapping ist für den Orchestrator transparent und wird über diesen bereitgestellt, HostPath was Sicherheitsbedenken nach sich ziehen kann.

Im folgenden Beispiel handelt es sich bei dem Pfad um `G:\Directory\app-state` eine SMB-Freigabe auf dem Windows-Knoten.

```
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
```

Der folgende Blog enthält Implementierungsdetails zur Einrichtung von Amazon FSx for Windows File Server als persistenten Speicher für Windows Pods.

Härten von Windows-Container-Images

Härten Sie Ihre Windows-Container-Images? Im Laufe der Jahre habe ich mit Kunden auf der ganzen Welt zusammengearbeitet, um ihnen bei der Migration älterer Workloads auf Container zu helfen, insbesondere Windows-Workloads. Mit mehr als 20 Jahren Erfahrung habe ich gesehen, wie Unternehmen erhebliche Anstrengungen und Ressourcen darauf verwendet haben, ihre Windows-Server zu sichern und alles zu implementieren, von CIS-Benchmarks bis hin zu Runtime-Virenschutz zum Schutz sensibler Daten.

Es hat sich jedoch ein besorgniserregender Trend herausgebildet. Da diese hochsicheren virtuellen Maschinen zu Containern modernisiert werden, werden viele wichtige Sicherheitsverfahren übersehen. Bewährte Methoden zur Windows-Sicherheit, vom Basisimage (OS) bis hin zu Webdiensten wie IIS, werden häufig vernachlässigt, wobei der Schwerpunkt hauptsächlich auf der Sicherung des Container-Hosts liegt. Es ist wichtig zu wissen, dass Container zwar in isolierten Namespaces operieren, aber dennoch Kernel-Primitive mit dem Host gemeinsam haben. Angreifer sind in der Regel eher an lateralen Bewegungen interessiert, als den Container-Host direkt ins Visier zu nehmen, wodurch sie schwache Sicherheitseinstellungen des Containers ausnutzen und auf sensible Daten zugreifen können.

Ziel der Dokumentation ist es, einige wichtige Sicherheitseinstellungen hervorzuheben, die Sie speziell für Windows-Container implementieren sollten, die `ASP.NET` Websites auf IIS hosten. Wir werden uns auf vier Schlüsselbereiche konzentrieren:

- Richtlinien zur Kontosicherheit
- Audit-Richtlinien
- Bewährte Methoden zur IIS-Sicherheit
- Prinzip der geringsten Rechte

Wir werden zunächst untersuchen, warum jede dieser Sicherheitskonfigurationen für den Schutz Ihrer Windows-Container von entscheidender Bedeutung ist, und untersuchen die spezifischen Risiken, die sie mindern, und die Sicherheitsvorteile, die sie bieten. Als Nächstes gehen wir einen Codeausschnitt durch, der zeigt, wie Sie diese Konfigurationen korrekt in Ihrem Dockerfile implementieren und so sicherstellen, dass Ihr Container vor potenziellen Bedrohungen geschützt ist. Abschließend werden wir jede Einstellung detailliert aufschlüsseln und eine umfassende Erläuterung ihrer Funktion, ihrer Auswirkungen auf die Containersicherheit und ihres Beitrags zum Schutz Ihrer Anwendungen geben. Dieser Ansatz zeigt Ihnen nicht nur, wie Sie diese Best Practices anwenden können, sondern gibt Ihnen auch Einblicke, warum sie für die Aufrechterhaltung eines robusten Sicherheitsstatus in containerisierten Umgebungen unerlässlich sind.

1. Konfigurieren Sie Kontorichtlinien (Passwort oder Sperrung) mithilfe der lokalen Sicherheitsrichtlinien und der Registrierung

Windows Server Core ist eine Minimalinstallationsoption, die als Teil des [EKS Optimized Windows AMI] (https://docs.aws.amazon.com/eks/latest/userguide/eks- optimized-windows-ami .html) verfügbar ist. Die Konfiguration von Kontorichtlinien (Passwort oder Sperrung) mithilfe von lokalen

Sicherheitsrichtlinien und der Registrierung erhöht die Systemsicherheit, indem robuste Passwort- und Sperrregeln durchgesetzt werden. Diese Richtlinien verlangen von Benutzern, sichere Passwörter mit einer definierten Mindestlänge und Komplexität zu erstellen, um sie vor häufigen passwortbezogenen Angriffen zu schützen.

Durch die Festlegung eines maximalen Kennwortalters werden die Benutzer aufgefordert, ihre Passwörter regelmäßig zu aktualisieren, wodurch die Wahrscheinlichkeit, dass Anmeldeinformationen kompromittiert werden, verringert wird. Sperrrichtlinien bieten eine zusätzliche Schutzebene, indem sie Konten nach einer bestimmten Anzahl fehlgeschlagener Anmeldeversuche vorübergehend sperren und so Brute-Force-Angriffe verhindern. Durch die Konfiguration dieser Einstellungen über die Windows-Registrierung können Administratoren diese Sicherheitsmaßnahmen auf Systemebene durchsetzen und so für Einheitlichkeit und Compliance im gesamten Unternehmen sorgen. Die Anwendung dieser Kontorichtlinien in einem Windows-Container ist für die Aufrechterhaltung der Sicherheitskonsistenz unerlässlich, auch wenn Container oft kurzlebig sind und für isolierte Workloads vorgesehen sind:

Konsistenz der Sicherheit

- Konformität: Die Durchsetzung einheitlicher Kennwortrichtlinien und Sperrregeln in Containern trägt zur Einhaltung der Sicherheitsbestimmungen bei, insbesondere in Umgebungen, in denen strenge Zugriffskontrollen erforderlich sind (z. B. Einhaltung gesetzlicher Vorschriften wie HIPAA, PCI-DSS).
- Gehärtete Container: Durch die Anwendung dieser Einstellungen wird sichergestellt, dass Ihr Windows-Container vor unbefugtem Zugriff oder passwortbasierten Angriffen geschützt ist, wodurch die Sicherheitslage Ihres Containers an die umfassenderen Systemsicherheitsrichtlinien angepasst wird.

Schutz vor Brute-Force-Angriffen

- Kontosperrung: Diese Einstellungen schützen vor Brute-Force-Anmeldeversuchen, indem Konten nach einer bestimmten Anzahl fehlgeschlagener Anmeldeversuche gesperrt werden. Dadurch wird verhindert, dass Angreifer eine unbegrenzte Anzahl von Passwörtern ausprobieren.
- Komplexität von Passwörtern: Die Anforderung komplexer Passwörter mit ausreichender Länge verringert die Wahrscheinlichkeit, dass schwache Passwörter ausgenutzt werden, selbst in isolierten containerisierten Umgebungen.

Szenarien mit mehreren Benutzern

- Wenn Ihre containerisierte Anwendung für die Verarbeitung mehrerer Benutzer konzipiert ist oder eine Benutzerauthentifizierung erfordert, stellt die Durchsetzung von Kennwortrichtlinien sicher, dass die Benutzerkonten innerhalb des Containers strenge Sicherheitsregeln einhalten und den Zugriff auf autorisierte Benutzer beschränken.

Persistente Windows-Container

- Container gelten zwar im Allgemeinen als kurzlebig, aber bestimmte Windows-Container können langfristige Dienste ausführen oder die Benutzerverwaltung übernehmen. Daher ist es wichtig, angemessene Sicherheitsrichtlinien durchzusetzen, die denen eines normalen Windows-Servers ähneln.

Konsistenz in Hybridumgebungen

- Wenn Sie in Ihrer Infrastruktur sowohl virtuelle Maschinen als auch Container ausführen, gewährleistet die Anwendung derselben Sicherheitsrichtlinien (z. B. password/lockout Richtlinien) in allen Umgebungen einheitliche Sicherheitsstandards und vereinfacht so die Steuerung und Verwaltung.

Zusammenfassend lässt sich sagen, dass die Anwendung dieser Kontorichtlinien innerhalb von Windows-Containern sicherstellt, dass Ihre Container keine Schwachstelle in Ihrer Sicherheitsstrategie darstellen. Sie schützen vor Passwortangriffen und sorgen für Konsistenz in Ihrer gesamten Umgebung.

Dockerfile:

```
# 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
```

Erklärung:

In diesem Abschnitt werden Kontorichtlinien für Passwort- und Sperreinstellungen über die Windows-Registrierung konfiguriert. Diese Richtlinien tragen zur Durchsetzung der Sicherheit bei, indem sie die Kennwortanforderungen und die Schwellenwerte für die Kontosperrung kontrollieren.

1. MinimumPasswordLength (MINPWLEN) = 14 Diese Einstellung definiert die Mindestanzahl an Zeichen für ein Passwort. Der Bereich umfasst 0-14 Zeichen; die Standardeinstellung sind sechs Zeichen.
2. MaximumPasswordAge (MAXPWAGE) = 60 Diese Einstellung definiert die maximale Anzahl von Tagen, an denen ein Passwort gültig ist. Bei Verwendung von UNLIMITED wird kein Limit angegeben. /MAXPWAGE kann nicht kleiner als /MINPWAGE sein. Der Bereich liegt zwischen 1 und 999. Die Standardeinstellung ist 90 Tage
3. Lockout Threshold (LOCKOUTTHRESHOLD) = 5 Diese Einstellung definiert den Schwellenwert für fehlgeschlagene Anmeldeversuche. Nach 5 falschen Versuchen wird das Konto gesperrt.

Diese Einstellungen tragen dazu bei, die Passwortsicherheit zu verbessern und Brute-Force-Angriffe zu verhindern, indem strenge Passwortrichtlinien durchgesetzt und Konten nach einer bestimmten Anzahl fehlgeschlagener Anmeldeversuche gesperrt werden.

2. Audit-Richtlinien

Überwachungsrichtlinien sind für Windows-Container wichtig, da sie wichtige Einblicke in Sicherheitsereignisse wie Anmeldeversuche und die Nutzung von Rechten bieten und dazu beitragen, unbefugten Zugriff zu erkennen, Benutzeraktivitäten zu überwachen und die Einhaltung gesetzlicher Standards sicherzustellen. Selbst in der kurzlebigen Natur von Containern sind Auditprotokolle für die Untersuchung von Vorfällen, die proaktive Erkennung von Bedrohungen und die Aufrechterhaltung eines konsistenten Sicherheitsstatus in containerisierten Umgebungen unerlässlich.

Sicherheitsüberwachung und Einhaltung von Vorschriften:

- Benutzeraktivitäten nachverfolgen: Mithilfe von Überwachungsrichtlinien können Administratoren Benutzeraktivitäten wie Anmeldeversuche und die Nutzung von Rechten innerhalb des Containers überwachen. Dies ist wichtig, um unbefugten Zugriff oder verdächtiges Verhalten zu erkennen.
- Einhaltung gesetzlicher Vorschriften: Viele Unternehmen müssen Sicherheitsereignisse protokollieren, um Vorschriften wie HIPAA, PCI-DSS und GDPR einzuhalten. Durch die Aktivierung von Überwachungsrichtlinien in Containern wird sichergestellt, dass Sie diese Anforderungen auch in containerisierten Umgebungen erfüllen.

Untersuchung von Vorfällen:

- Forensik und Analyse: Wenn eine containerisierte Anwendung oder ein Dienst gefährdet ist, können Auditprotokolle wertvolle Erkenntnisse für die Analyse nach dem Vorfall liefern. Sie helfen Sicherheitsteams dabei, die von Angreifern ergriffenen Maßnahmen nachzuverfolgen oder festzustellen, wie es zu einer Sicherheitsverletzung kam.
- Erkennung in Echtzeit: Mithilfe von Auditprotokollen können Administratoren Echtzeitwarnungen für kritische Ereignisse einrichten (z. B. fehlgeschlagene Anmeldeversuche, Rechteerweiterungen). Diese proaktive Überwachung hilft, Angriffe frühzeitig zu erkennen und ermöglicht schnellere Reaktionszeiten.

Konsistenz in allen Umgebungen:

- Einheitlicher Sicherheitsstatus: Durch die Anwendung von Überwachungsrichtlinien in Containern über die Registrierung stellen Sie konsistente Sicherheitspraktiken sowohl in containerisierten als auch in nicht containerisierten Umgebungen sicher. Dadurch wird verhindert, dass Container zu einem blinden Fleck für die Sicherheitsüberwachung werden.
- Transparenz in Hybridumgebungen: Für Unternehmen, die sowohl herkömmliche Windows-Server als auch Container betreiben, bieten Prüfungsrichtlinien eine ähnliche Transparenz und Kontrolle auf allen Plattformen, wodurch die Verwaltung einfacher und effektiver wird.

Verfolgung privilegierter Operationen:

- Überwachung der Nutzung von Rechten: In Containerumgebungen, in denen Anwendungen mit erhöhten Rechten ausgeführt werden oder in denen administrative Aufgaben ausgeführt werden, gewährleistet die Prüfung privilegierter Operationen die Rechenschaftspflicht. Sie können protokollieren, wer auf vertrauliche Ressourcen zugegriffen oder kritische Aufgaben innerhalb des Containers ausgeführt hat.
- Missbrauch von Rechten verhindern: Durch die Überwachung der Nutzung von Rechten können Sie erkennen, wenn nicht autorisierte Benutzer versuchen, ihre Rechte zu erweitern oder auf eingeschränkte Bereiche innerhalb des Containers zuzugreifen, wodurch interne oder externe Angriffe verhindert werden.

Erkennung unberechtigter Zugriffsversuche:

- Fehlgeschlagene Anmeldeversuche: Durch die Aktivierung von Überwachungsrichtlinien für fehlgeschlagene Anmeldeversuche können Brute-Force-Angriffe oder unbefugte Versuche, auf

containerisierte Anwendungen zuzugreifen, identifiziert werden. Auf diese Weise erhalten Sie einen Überblick darüber, wer versucht, wie oft Zugriff auf das System zu erhalten.

- Überwachung von Kontosperrungen: Durch die Überwachung von Kontosperrungsereignissen können Administratoren potenzielle Aussperrungen erkennen und untersuchen, die durch verdächtige oder böswillige Aktivitäten verursacht werden.

Dauerhafte Sicherheit auch in kurzlebigen Umgebungen:

- Flüchtig und doch sicher: Container sind zwar kurzlebig, was bedeutet, dass sie häufig gelöscht und neu erstellt werden können, aber Auditing spielt immer noch eine wichtige Rolle, wenn es darum geht, sicherzustellen, dass Sicherheitsereignisse erfasst werden, während der Container läuft. Dadurch wird sichergestellt, dass kritische Sicherheitsereignisse für die Dauer des Container-Lebenszyklus protokolliert werden.

Zentralisierte Protokollierung:

- Weiterleitung von Protokollen an zentrale Systeme: Container können in zentrale Protokollierungssysteme (z. B. ELK-Stack, AWS CloudWatch) integriert werden, um Audit-Logs von mehreren Container-Instances zu erfassen. Dies ermöglicht eine bessere Analyse und Korrelation von Sicherheitsereignissen in Ihrer gesamten Infrastruktur.

Docker-Datei:

```
# 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
```

Erklärung:

In diesem Abschnitt werden Überwachungsrichtlinien mithilfe von Registrierungsänderungen konfiguriert. Mit Überwachungsrichtlinien wird gesteuert, welche Sicherheitsereignisse von Windows protokolliert werden. Dies hilft bei der Überwachung und Erkennung unberechtigter Zugriffsversuche.

1. SCENoApplyLegacyAuditPolicy = 0 Dadurch wird das alte Überwachungsrichtlinienformat deaktiviert, sodass detailliertere Überwachungsrichtlinien möglich sind, die in späteren Versionen von Windows eingeführt wurden. Dies ist wichtig für moderne Überwachungskonfigurationen.
2. Auditpol-Unterkategorie: „Anmeldung“ Mit dieser Einstellung können sowohl erfolgreiche als auch fehlgeschlagene Anmeldeereignisse geprüft werden. Der Wert 3 bedeutet, dass Windows sowohl erfolgreiche als auch fehlgeschlagene Anmeldeversuche protokolliert. Dies hilft bei der Überwachung, wer auf das System zugreift, und bei der Erfassung fehlgeschlagener Anmeldeversuche.

Diese Überwachungsrichtlinien sind für die Sicherheitsüberwachung und die Einhaltung der Vorschriften von entscheidender Bedeutung, da sie detaillierte Protokolle wichtiger Sicherheitsereignisse wie Anmeldeversuche und die Verwendung privilegierter Operationen enthalten.

3. Bewährte Methoden zur IIS-Sicherheit für Windows-Container

Die Implementierung von bewährten IIS-Methoden in Windows-Containern ist aus mehreren Gründen wichtig, um sicherzustellen, dass Ihre Anwendungen sicher, leistungsstark und skalierbar sind. Container bieten zwar eine isolierte und schlanke Umgebung, erfordern aber dennoch eine korrekte Konfiguration, um Sicherheitslücken und Betriebsprobleme zu vermeiden. Aus diesem Grund ist es wichtig, die bewährten Methoden für IIS in Windows-Containern zu befolgen:

Sicherheit

- Vermeidung häufiger Sicherheitslücken: IIS ist häufig ein Ziel für Angriffe wie Cross-Site Scripting (XSS), Clickjacking und Offenlegung von Informationen. Die Implementierung von Sicherheitsheadern (z. B.,, und Strict-Transport-Security) trägt dazu bei X-Content-Type-Options X-Frame-Options, Ihre Anwendung vor diesen Bedrohungen zu schützen.
- Isolation allein reicht nicht aus: Container sind isoliert, aber eine falsch konfigurierte IIS-Instanz kann vertrauliche Informationen wie Serverversionsdetails, Verzeichnislisten oder unverschlüsselte Kommunikation preisgeben. Indem Sie Funktionen wie das Durchsuchen von Verzeichnissen und das Entfernen des IIS-Versionsheaders deaktivieren, minimieren Sie die Angriffsfläche.

- Verschlüsselung und HTTPS: Bewährte Methoden, wie z. B. die Durchsetzung von reinen HTTPS-Verbindungen, stellen sicher, dass Daten während der Übertragung verschlüsselt werden, und schützen so vertrauliche Informationen davor, abgefangen zu werden.

Leistung

- Effiziente Ressourcennutzung: Bewährte IIS-Methoden wie die Aktivierung dynamischer und statischer Komprimierung reduzieren die Bandbreitennutzung und verbessern die Ladezeiten. Diese Optimierungen sind besonders wichtig in containerisierten Umgebungen, in denen Ressourcen von Containern und dem Hostsystem gemeinsam genutzt werden.
- Optimierte Protokollierung: Die korrekte Konfiguration der Protokollierung (z. B. einschließlich des X-Forwarded-For Headers) stellt sicher, dass Sie die Client-Aktivitäten verfolgen und gleichzeitig unnötigen Logging-Overhead minimieren können. Auf diese Weise können Sie relevante Daten für die Fehlerbehebung sammeln, ohne die Leistung zu beeinträchtigen.

Skalierbarkeit und Wartbarkeit

- Konsistenz in allen Umgebungen: Indem Sie bewährte Methoden befolgen, stellen Sie sicher, dass Ihre IIS-Konfiguration über mehrere Container-Instances hinweg konsistent ist. Dies vereinfacht die Skalierung und stellt sicher, dass bei der Bereitstellung neuer Container dieselben Sicherheits- und Leistungsrichtlinien eingehalten werden.
- Automatisierte Konfigurationen: Bewährte Methoden in Dockerfiles, wie das Festlegen von Ordnerberechtigungen und das Deaktivieren unnötiger Funktionen, stellen sicher, dass jeder neue Container automatisch korrekt konfiguriert wird. Dies reduziert manuelle Eingriffe und senkt das Risiko menschlicher Fehler.

Compliance

- Erfüllung regulatorischer Anforderungen: In vielen Branchen gelten strenge regulatorische Anforderungen (z. B. PCI-DSS, HIPAA), die spezifische Sicherheitsmaßnahmen wie verschlüsselte Kommunikation (HTTPS) und Protokollierung von Kundenanfragen vorschreiben. Die Einhaltung der bewährten IIS-Methoden in Containern trägt dazu bei, die Einhaltung dieser Standards sicherzustellen.
- Überprüfbarkeit: Die Implementierung von Prüfungsrichtlinien und die sichere Protokollierung ermöglichen die Rückverfolgbarkeit von Ereignissen, was bei Audits von entscheidender Bedeutung ist. Durch die Protokollierung des X-Forwarded-For Headers wird beispielsweise

sichergestellt, dass Client-IP-Adressen in proxybasierten Architekturen korrekt aufgezeichnet werden.

Minimierung des Risikos in gemeinsam genutzten Umgebungen

- Vermeidung von Fehlkonfigurationen: Container nutzen den Kernel des Hosts gemeinsam, und obwohl sie voneinander isoliert sind, kann eine schlecht konfigurierte IIS-Instanz Sicherheitslücken aufdecken oder Leistungsengpässe verursachen. Bewährte Methoden stellen sicher, dass jede IIS-Instanz optimal läuft, wodurch das Risiko containerübergreifender Probleme reduziert wird.
- Zugriff mit geringsten Rechten: Durch die Festlegung geeigneter Berechtigungen für Ordner und Dateien innerhalb des Containers (z. B. mithilfe von Set-Acl in PowerShell) wird sichergestellt, dass Benutzer und Prozesse innerhalb des Containers nur über den erforderlichen Zugriff verfügen, wodurch das Risiko einer Rechteerweiterung oder Datenmanipulation verringert wird.

Resilienz in kurzlebigen Umgebungen

- Vergänglichkeit von Containern: Container sind oft kurzlebig und werden häufig umgebaut. Durch die Anwendung bewährter IIS-Methoden wird sichergestellt, dass jeder Container sicher und konsistent konfiguriert ist, unabhängig davon, wie oft er erneut bereitgestellt wird. Dadurch wird verhindert, dass im Laufe der Zeit Fehlkonfigurationen eingeführt werden.
- Minimierung potenzieller Fehlkonfigurationen: Durch die automatische Durchsetzung von Best Practices (z. B. Deaktivierung schwacher Protokolle oder Header) wird das Risiko einer Fehlkonfiguration bei Container-Neustarts oder -Updates minimiert.

Docker-Datei:

```
# 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"
```

Erklärung:

Mit diesem Befehl wird IIS so konfiguriert, dass der X-Forwarded-For Header protokolliert wird. Dieser wird üblicherweise verwendet, um die ursprüngliche Client-IP-Adresse zu erfassen, wenn eine Anfrage einen Proxy oder einen Load Balancer durchläuft. Standardmäßig protokolliert IIS nur die IP-Adresse des Load Balancers oder Reverse-Proxys. Wenn Sie also dieses benutzerdefinierte Protokollfeld hinzufügen, können Sie die tatsächliche Client-IP für Sicherheitsüberprüfungen, Analysen und Problembehebung ermitteln.

1. X-Forwarded-For Header, der üblicherweise verwendet wird, um die ursprüngliche Client-IP-Adresse zu erfassen, wenn eine Anfrage einen Proxy oder Load Balancer durchläuft. Standardmäßig protokolliert IIS nur die IP-Adresse des Load Balancers oder Reverse-Proxys. Wenn Sie also dieses benutzerdefinierte Protokollfeld hinzufügen, können Sie die tatsächliche Client-IP für Sicherheitsüberprüfungen, Analysen und Problembehebung ermitteln.
2. Strict-Transport-Security (HSTS) Stellt sicher, dass Browser nur über HTTPS kommunizieren. Der Wert max-age=31536000 gibt an, dass diese Richtlinie für ein Jahr gilt, und wendet die Richtlinie auf alle Subdomains an. includeSubDomains
3. X-Content-Type-Options verhindert, dass Browser eine Antwort außerhalb des deklarierten Inhaltstyps „MIME-Sniffing“ durchführen. Dies hilft, einige Arten von Angriffen zu verhindern.
4. X-XSS-Protection Aktiviert den Cross-Site Scripting (XSS) -Schutz in Browsern.
5. X-Frame-Options verhindert das Einbetten der Seite in Iframes und schützt so vor Clickjacking-Angriffen.

6. Offenlegung der IIS-Version deaktivieren Mit diesem Befehl wird der Server-Header in HTTP-Antworten deaktiviert, wodurch standardmäßig die verwendete IIS-Version angezeigt wird. Das Ausblenden dieser Informationen trägt dazu bei, das Risiko zu verringern, dass Angreifer spezifische Sicherheitslücken in der IIS-Version identifizieren und gezielt einsetzen.
7. Reine HTTPS-Verbindungen aktivieren Dieser (auskommentierte) Abschnitt erzwingt HTTPS-Verbindungen und deaktiviert HTTP. Wenn es nicht kommentiert ist, konfiguriert das Dockerfile IIS so, dass es nur auf Port 443 (HTTPS) lauscht und die standardmäßige HTTP-Bindung auf Port 80 entfernt. Dies ist nützlich, wenn SSL innerhalb des Containers beendet wird, und stellt sicher, dass der gesamte Datenverkehr verschlüsselt wird.
8. Verzeichnissuche deaktivieren Verhindert, dass IIS eine Verzeichnisliste anzeigt, wenn kein Standarddokument vorhanden ist. Dadurch wird vermieden, dass die interne Dateistruktur Benutzern zugänglich gemacht wird.
9. Durch benutzerdefinierte Fehlerseiten weiterleiten Stellt sicher, dass IIS die Fehlerseiten der Anwendung durchlässt, wenn die Anwendung über eine eigene Fehlerbehandlung verfügt, anstatt die IIS-Standardfehlerseiten anzuzeigen.
10. Der detaillierte Fehlermodus konfiguriert IIS so, dass detaillierte Fehlermeldungen nur für lokale Anfragen angezeigt werden, sodass Entwickler Probleme diagnostizieren können, ohne vertrauliche Informationen externen Benutzern zugänglich zu machen.
11. Stellen Sie die richtigen Ordnerberechtigungen sicher Dieser Block konfiguriert die Ordnerberechtigungen für das IIS-Webstammverzeichnis (C:\inetpub\wwwroot). Er legt Lese- und Ausführungsberechtigungen für die Gruppen IIS_IUSRS und Users fest und stellt so sicher, dass diese Benutzer auf den Ordner zugreifen, aber keine Dateien ändern können. Durch das Einstellen der richtigen Berechtigungen wird das Risiko eines unbefugten Zugriffs oder einer Manipulation der auf dem Webserver gehosteten Dateien minimiert.

Die Einhaltung der bewährten IIS-Methoden in Windows-Containern stellt sicher, dass Ihre containerisierten Anwendungen sicher, leistungsstark und skalierbar sind. Diese Methoden tragen dazu bei, Sicherheitslücken zu vermeiden, die Ressourcennutzung zu optimieren, die Einhaltung von Vorschriften sicherzustellen und die Konsistenz zwischen Container-Instances aufrechtzuerhalten. Obwohl Container für den isolierten Einsatz konzipiert sind, ist eine korrekte Konfiguration erforderlich, um Risiken zu minimieren und die Zuverlässigkeit Ihrer Anwendung in Produktionsumgebungen zu gewährleisten.

4. Prinzip der geringsten Rechte

Das Prinzip der geringsten Rechte (Principle of Least Privilege, PolP) ist für Windows-Container aus mehreren wichtigen Gründen von entscheidender Bedeutung, insbesondere zur Erhöhung der Sicherheit und Minimierung von Risiken in containerisierten Umgebungen. Dieses Prinzip schreibt vor, dass ein System oder eine Anwendung mit den Mindestberechtigungen arbeiten sollte, die für ein ordnungsgemäßes Funktionieren erforderlich sind. Hier erfahren Sie, warum es in Windows-Containern wichtig ist:

Minimierung der Angriffsfläche

- In Containern werden häufig Anwendungen ausgeführt, die mit verschiedenen Systemkomponenten interagieren. Je mehr Rechte eine Anwendung hat, desto umfassender ist ihr Zugriff auf diese Komponenten. Durch die Beschränkung der Berechtigungen des Containers auf das Notwendige reduziert PolP die Angriffsfläche erheblich, sodass es für einen Angreifer schwieriger wird, den Container auszunutzen, wenn er kompromittiert wird.

Begrenzung der Auswirkungen kompromittierter Container

- Wenn ein Windows-Container kompromittiert ist, kann ein Angreifer durch die Ausführung von Anwendungen mit übermäßigen Rechten (z. B. Administrator- oder Root-Zugriff) die Kontrolle über wichtige Systemdateien erlangen oder Rechte auf dem gesamten Container-Host erweitern. Durch die Durchsetzung von PolP ist der Angreifer selbst bei einem Angriff auf einen Container in seinen Möglichkeiten eingeschränkt und verhindert so eine weitere Eskalation und den Zugriff auf vertrauliche Ressourcen oder andere Container.

Schutz in Umgebungen mit mehreren Mandanten

- In Cloud- oder Unternehmensumgebungen können mehrere Container auf derselben physischen oder virtuellen Infrastruktur ausgeführt werden. PolP stellt sicher, dass ein kompromittierter Container nicht auf Ressourcen oder Daten anderer Mandanten zugreifen kann. Diese Isolierung ist entscheidend für die Aufrechterhaltung der Sicherheit in gemeinsam genutzten Umgebungen mit mehreren Mandanten und schützt vor seitlicher Bewegung zwischen Containern.

Eindämmung der Eskalation von Rechten

- Container, die mit hohen Rechten ausgeführt werden, können von Angreifern verwendet werden, um Rechte innerhalb des Systems zu erweitern. PolP mindert dieses Risiko, indem es den Zugriff des Containers auf Systemressourcen einschränkt und so unbefugte Aktionen oder Rechteerweiterungen außerhalb der Container-Umgebung verhindert.

Einhaltung von Vorschriften und Prüfung

- Viele regulatorische Standards und Sicherheitsrahmen (z. B. PCI DSS, HIPAA, GDPR) verlangen von Systemen, dass sie sich an PolP halten, um den Zugriff auf sensible Daten zu beschränken. Der Betrieb von Windows-Containern mit eingeschränkten Rechten unterstützt Unternehmen bei der Einhaltung dieser Vorschriften und stellt sicher, dass Anwendungen nur Zugriff auf die Ressourcen erhalten, die sie speziell benötigen.

Verringerung des Risikos von Fehlkonfigurationen

- Wenn Container mit unnötigen Rechten ausgeführt werden, kann selbst eine geringfügige Fehlkonfiguration zu schwerwiegenden Sicherheitslücken führen. Wenn beispielsweise ein Container, der als Administrator ausgeführt wird, versehentlich mit dem Internet in Kontakt kommt, könnte ein Angreifer die Kontrolle über das System erlangen. PolP trägt dazu bei, solche Risiken zu vermeiden, indem es standardmäßig eingeschränkte Rechte verwendet, wodurch Fehlkonfigurationen weniger gefährlich werden.

Verbesserter Sicherheitsstatus bei Containern

- Durch die Einhaltung von PolP sind Container besser vom zugrundeliegenden Hostsystem und voneinander isoliert. Dadurch wird sichergestellt, dass die containerisierte Anwendung weniger wahrscheinlich auf Systemdateien oder Prozesse außerhalb des definierten Bereichs zugreift oder diese verändert, wodurch die Integrität des Host-Betriebssystems und anderer Workloads gewahrt bleibt.

Docker-Datei:

```
# Strongly recommended that when deploying a Windows server container to any multi-
tenant environment that your application runs via the ContainerUser account
USER ContainerUser
```

Erklärung:

In diesem Abschnitt gibt der ContainerUser Befehl USER an, dass die Anwendung im Windows-Container unter dem ContainerUser Konto statt unter dem Standard-Administratorkonto ausgeführt werden soll.

Aus folgenden Gründen ist dies wichtig, insbesondere in einer Umgebung mit mehreren Mandanten:

1. Prinzip der geringsten Rechte: Bei dem ContainerUser Konto handelt es sich um einen Benutzer ohne Administratorrechte mit eingeschränkten Rechten. Die Ausführung der Anwendung unter diesem Konto erfolgt nach dem Prinzip der geringsten Rechte, wodurch das Risiko einer Ausnutzung minimiert wird. Würde ein Angreifer die Anwendung kompromittieren, hätte er eingeschränkten Zugriff auf das System, wodurch der potenzielle Schaden reduziert würde.
2. Verbesserte Sicherheit: In Umgebungen mit mehreren Mandanten können Container dieselbe zugrunde liegende Infrastruktur gemeinsam nutzen. Running as ContainerUser stellt sicher, dass selbst wenn ein Container kompromittiert wird, dieser keine Administratorrechte hat, um auf kritische Systemdateien oder andere Container zuzugreifen oder diese zu ändern. Dadurch wird die Angriffsfläche erheblich reduziert.
3. Vermeidung des Root-Zugriffs: Standardmäßig können Container mit erhöhten Rechten ausgeführt werden (ähnlich dem Root-Zugriff in Linux-Containern), was gefährlich sein kann, wenn sie ausgenutzt werden. Durch die Verwendung ContainerUser wird sichergestellt, dass die Anwendung nicht mit unnötigen Administratorrechten ausgeführt wird, was es für Angreifer schwieriger macht, Rechte zu eskalieren.
4. Bewährte Methode für Umgebungen mit mehreren Mandanten: In Umgebungen, in denen sich mehrere Benutzer oder Organisationen dieselbe Infrastruktur teilen (z. B. in der Cloud), ist Sicherheit von entscheidender Bedeutung. Durch das Ausführen von Anwendungen mit eingeschränkten Berechtigungen wird verhindert, dass sich die Anwendung eines Mandanten auf andere auswirkt, wodurch sensible Daten und Ressourcen auf der gesamten Plattform geschützt werden.

Der ContainerUser Befehl USER stellt sicher, dass die Anwendung mit minimalen Rechten ausgeführt wird. Dadurch wird die Sicherheit in Umgebungen mit mehreren Mandanten erhöht, indem der Schaden begrenzt wird, der entstehen könnte, wenn der Container kompromittiert wird. Dies ist eine bewährte Methode, um unbefugten Zugriff oder die Eskalation von Rechten in einer containerisierten Umgebung zu verhindern.

Das Prinzip der geringsten Rechte ist für Windows-Container unverzichtbar, da es die potenziellen Auswirkungen von Sicherheitsverletzungen begrenzt, die Angriffsfläche reduziert und unbefugten Zugriff auf kritische Systemkomponenten verhindert. Indem Unternehmen containerisierte

Anwendungen nur mit den erforderlichen Berechtigungen ausführen, können sie die Sicherheit und Stabilität ihrer Container-Umgebungen erheblich verbessern, insbesondere in Mehrmandanten- und gemeinsam genutzten Infrastrukturen.

Letzte Gedanken: Warum der Schutz Ihrer Windows-Container in der heutigen Bedrohungslandschaft ein Muss ist

In der heutigen, sich schnell entwickelnden digitalen Welt, in der Bedrohungen immer ausgefeilter und umfangreicher werden, ist der Schutz Ihrer Windows-Container nicht nur eine Empfehlung, sondern eine absolute Notwendigkeit. Container bieten eine einfache und flexible Möglichkeit, Anwendungen zu verpacken und bereitzustellen, sind jedoch nicht vor Sicherheitslücken gefeit. Da immer mehr Unternehmen Container einsetzen, um ihre Infrastruktur zu optimieren, werden sie auch zu einem potenziellen Ziel für Cyberangriffe, wenn sie nicht ausreichend geschützt sind.

Das Internet ist mit verschiedenen Bedrohungen überflutet — von böswilligen Akteuren, die ungepatchte Sicherheitslücken ins Visier nehmen, bis hin zu automatisierten Bots, die nach Fehlkonfigurationen suchen. Ohne die richtigen Sicherheitsmaßnahmen können Container ausgenutzt werden, um sensible Daten preiszugeben, Rechte zu erweitern oder als Einstiegspunkte für Angriffe zu dienen, die Ihre gesamte Infrastruktur gefährden können. Aus diesem Grund ist die Sicherheit von Containern genauso wichtig wie die Sicherung aller anderen Teile Ihrer Umgebung.

Bei der Verwendung von Windows-Containern gelten weiterhin viele traditionelle bewährte Sicherheitsmethoden. Die Implementierung robuster Kontorichtlinien, die Sicherung von IIS-Konfigurationen, die Durchsetzung von HTTPS, die Verwendung strenger Firewallregeln und die Anwendung des Zugriffs mit den geringsten Rechten auf kritische Dateien sind alles wichtige Maßnahmen, die sicherstellen, dass der Container vor Angriffen geschützt bleibt. Darüber hinaus bieten regelmäßige Prüfungen und Protokollierung einen Überblick darüber, was im Container passiert, sodass Sie verdächtige Aktivitäten erkennen können, bevor sie zu einem echten Vorfall werden.

Die Sicherung von Windows-Containern entspricht auch den gesetzlichen Anforderungen, die den Schutz sensibler Daten und die Gewährleistung der Anwendungsintegrität vorschreiben. Da Cloud-native und containerisierte Architekturen immer mehr Verbreitung finden, trägt die Gewährleistung der Sicherheit auf jeder Ebene, vom Basis-Image bis zum laufenden Container, dazu bei, Ihren Betrieb zu schützen und das Vertrauen der Kunden zu wahren.

Zusammenfassend lässt sich sagen, dass die Zunahme containerisierter Anwendungen in Verbindung mit der wachsenden Zahl von Cyberbedrohungen die Containersicherheit zu einem

unverzichtbaren Aspekt des modernen Infrastrukturmanagements macht. Durch die Einhaltung von Best Practices und die kontinuierliche Überwachung auf Sicherheitslücken können Unternehmen die Agilität und Effizienz von Windows-Containern nutzen, ohne Kompromisse bei der Sicherheit einzugehen. In dieser bedrohlichen Umgebung ist der Schutz Ihrer Windows-Container nicht nur eine Option, sondern ein Muss.

Bewährte Methoden für Hybridbereitstellungen

Dieses Handbuch enthält Anleitungen zur Ausführung von Bereitstellungen in lokalen oder Edge-Umgebungen mit EKS Hybrid Nodes oder EKS Anywhere.

Wir haben derzeit Leitfäden zu den folgenden Themen veröffentlicht:

- Bewährte Methoden für EKS-Hybridknoten und Netzwerkunterbrechungen

EKS-Hybridknoten und Netzwerkunterbrechungen

Die EKS-Hybridknoten-Architektur kann für Kunden, die es gewohnt sind, lokale Kubernetes-Cluster vollständig in ihren eigenen Rechenzentren oder Edge-Standorten auszuführen, neu sein. Bei EKS-Hybridknoten läuft die Kubernetes-Steuerebene in einer AWS-Region und nur die Knoten werden lokal ausgeführt, was zu einer „erweiterten“ oder „erweiterten“ Kubernetes-Clusterarchitektur führt.

Dies führt zu einer häufig gestellten Frage: „Was passiert, wenn meine Knoten von der Kubernetes-Steuerebene getrennt werden?“

In diesem Leitfaden beantworten wir diese Frage anhand eines Überblicks über die folgenden Themen. Es wird empfohlen, die Stabilität und Zuverlässigkeit Ihrer Anwendungen durch Netzwerkunterbrechungen zu überprüfen, da sich jede Anwendung je nach ihren Abhängigkeiten, Konfiguration und Umgebung unterschiedlich verhalten kann. Im aws-samples/ eks-hybrid-examples GitHub repo finden Sie Test-Setup, Verfahren und Ergebnisse, auf die Sie zurückgreifen können, um Netzwerkunterbrechungen mit EKS-Hybridknoten und Ihren eigenen Anwendungen zu testen. Das GitHub Repo enthält auch zusätzliche Informationen zu den Tests, die zur Validierung des in diesem Handbuch erläuterten Verhaltens verwendet wurden.

- Bewährte Methoden für Stabilität bei Netzwerkunterbrechungen
- Failover-Verhalten von Kubernetes-Pods durch Netzwerkunterbrechungen
- Netzwerkverkehr von Anwendungen durch Netzwerkunterbrechungen
- Host-Anmeldeinformationen bei Netzwerkunterbrechungen

Bewährte Methoden für Stabilität bei Netzwerkunterbrechungen

Hochverfügbares Netzwerk

Der beste Ansatz, um Netzwerkunterbrechungen zwischen Hybridknoten und der Kubernetes-Steuerebene zu vermeiden, besteht darin, redundante, belastbare Verbindungen von Ihrer lokalen Umgebung zu und von AWS zu verwenden. Weitere Informationen zur Architektur hochverfügbarer Hybridnetzwerke mit diesen Lösungen finden Sie im AWS Direct Connect Resiliency Toolkit und in der AWS Site-to-Site VPN-Dokumentation.

Hochverfügbare Anwendungen

Berücksichtigen Sie bei der Anwendungsarchitektur Ihre Fehlerdomänen und die Auswirkungen verschiedener Arten von Ausfällen. Kubernetes bietet integrierte Mechanismen zur Bereitstellung und Wartung von Anwendungsreplikaten über Knoten-, Zonen- und regionale Domänen hinweg. Die Verwendung dieser Mechanismen hängt von Ihrer Anwendungsarchitektur, Ihren Umgebungen und Ihren Verfügbarkeitsanforderungen ab. Beispielsweise können statuslose Anwendungen häufig mit mehreren Replikaten bereitgestellt werden und sich über beliebige Hosts und Infrastrukturkapazitäten bewegen. Außerdem können Sie mithilfe von Knotenselektoren und Beschränkungen der Topologieverteilung Instanzen der Anwendung in verschiedenen Domänen ausführen. Einzelheiten zu Techniken auf Anwendungsebene zur Erstellung robuster Anwendungen auf Kubernetes finden Sie im EKS Best Practices Guide.

Kubernetes wertet bei der Entscheidung, ob Pods auf andere Knoten verschoben werden sollen, zonale Informationen für Knoten aus, die von der Kubernetes-Steuerebene getrennt sind. Wenn alle Knoten in einer Zone nicht erreichbar sind, storniert Kubernetes die Pod-Räumung für die Knoten in dieser Zone. Als bewährte Methode gilt: Wenn Sie eine Bereitstellung mit Knoten haben, die in mehreren Rechenzentren oder physischen Standorten ausgeführt werden, weisen Sie jedem Knoten eine Zone auf der Grundlage seines Rechenzentrums oder physischen Standorts zu. Wenn Sie EKS mit Knoten in der Cloud ausführen, wird dieses Zonenlabel automatisch von AWS angewendet cloud-controller-manager. A cloud-controller-manager wird jedoch nicht für Hybridknoten verwendet, sodass Sie diese Informationen über Ihre Kubelet-Konfiguration weitergeben können. Ein Beispiel für die Konfiguration einer Zone in Ihrer Knotenkonfiguration für Hybridknoten finden Sie unten. Die Konfiguration wird übergeben, wenn Sie Ihre Hybridknoten mit der Hybridknoten-CLI (`nodeadm`) mit Ihrem Cluster verbinden. Weitere Informationen zu dem `topology.kubernetes.io/zone` Label finden Sie in der Kubernetes-Dokumentation. Weitere Informationen zur CLI für Hybridknoten finden Sie in der Nodeadm-Referenz zu Hybrid Nodes.

```
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:
    ...
```

Netzwerk-Überwachung

Wenn Sie AWS Direct Connect oder AWS Site-to-Site VPN für Ihre Hybrid-Konnektivität verwenden, können Sie CloudWatch Alarme, Protokolle und Metriken nutzen, um den Status Ihrer Hybridverbindung zu beobachten und Probleme zu diagnostizieren. Weitere Informationen finden Sie unter Überwachen von AWS Direct Connect Connect-Ressourcen und Überwachen einer Site-to-Site AWS-VPN-Verbindung.

Es wird empfohlen, Alarme für `NodeNotReady` Ereignisse zu erstellen, die von der auf der EKS-Ebene node-lifecycle-controller ausgeführten Steuerungsebene gemeldet werden. Dies signalisiert, dass bei einem Hybridknoten möglicherweise eine Netzwerkunterbrechung auftritt. Sie können diesen Alarm erstellen, indem Sie die Protokollierung der EKS-Kontrollebene für den Controller Manager aktivieren und einen Metrikfilter CloudWatch für die Meldung „Statusänderungsereignis für Knoten aufzeichnen“ mit dem Status= „“ erstellen. NodeNotReady Nachdem Sie einen Metrikfilter erstellt haben, können Sie einen Alarm für diesen Filter erstellen, der auf Ihren gewünschten Schwellenwerten basiert. Weitere Informationen finden Sie in der Dokumentation unter Alarming for Logs. CloudWatch

Sie können die integrierten Metriken Transit Gateway (TGW) und Virtual Private Gateway (VGW) verwenden, um den Netzwerkverkehr zu und von Ihrem TGW oder VGW zu beobachten. Sie können Alarme für diese Messwerte erstellen, um Szenarien zu erkennen, in denen der Netzwerkverkehr unter das normale Niveau fällt, was auf ein potenzielles Netzwerkproblem zwischen Hybridknoten und der EKS-Steuerebene hindeutet. Die TGW- und VGW-Metriken werden in der folgenden Tabelle beschrieben.

Gateway	Metrik	Description
Transit-Gateway	BytesIn	Die von TGW vom Anhang empfangenen Bytes (EKS-Steu erebene für Hybridknoten)
Transit-Gateway	BytesOut	Die von TGW an den Anhang gesendeten Bytes (Hybridkn oten an die EKS-Steuerebene)
Virtual Private Gateway	TunnelDat aIn	Die von der AWS-Seite der Verbindung durch den VPN-Tunnel zum Kunden-Gateway gesendeten Bytes (EKS-Kont rollebene zu Hybridknoten)
Virtual Private Gateway	TunnelDat aOut	Die auf der AWS-Seite der Verbindung durch den VPN-Tunnel vom Kunden-Gateway empfangenen Bytes (Hybridknoten zur EKS-Steuerebene)

Sie können CloudWatch Network Monitor auch verwenden, um tiefere Einblicke in Ihre Hybridverbindungen zu erhalten, um die durchschnittliche Zeit bis zur Wiederherstellung zu reduzieren und festzustellen, ob Netzwerkprobleme auf AWS oder Ihre Umgebung zurückzuführen sind. CloudWatch Network Monitor kann verwendet werden, um Paketverlust und Latenz in Ihren hybriden Netzwerkverbindungen zu visualisieren, Warnmeldungen und Schwellenwerte festzulegen und dann Maßnahmen zur Verbesserung Ihrer Netzwerkleistung zu ergreifen. Weitere Informationen finden Sie unter Amazon CloudWatch Network Monitor verwenden.

EKS bietet mehrere Optionen zur Überwachung des Zustands Ihrer Cluster und Anwendungen. Für den Clusterstatus können Sie das Observability-Dashboard in der EKS-Konsole verwenden, um Probleme schnell zu erkennen, zu beheben und zu beheben. Sie können auch Amazon Managed Service für Prometheus, AWS Distro for Open Telemetry (ADOT) und für die Cluster-, Anwendungs- und CloudWatch Infrastrukturüberwachung verwenden. Weitere Informationen zu den EKS-Observability-Optionen finden Sie unter Überwachen der Clusterleistung und Anzeigen von Protokollen.

Lokale Problembehandlung

Um Netzwerkunterbrechungen zwischen Hybridknoten und der EKS-Steuerebene vorzubereiten, können Sie sekundäre Überwachungs- und Protokollierungs-Backends einrichten, um die Beobachtbarkeit für Anwendungen aufrechtzuerhalten, wenn regionale AWS-Services nicht erreichbar sind. Sie können beispielsweise den AWS Distro for Open Telemetry (ADOT) Collector so

konfigurieren, dass er Metriken und Protokolle an mehrere Backends sendet. Sie können auch lokale Tools wie die `crictl` CLI verwenden, um lokal mit Pods und Containern zu interagieren, als Ersatz für `kubectl` oder andere Kubernetes-API-kompatible Clients, die normalerweise den Kubernetes-API-Serverendpunkt abfragen. Weitere Informationen dazu finden Sie in der Dokumentation in den `crictl` CRI-Tools. `crictl` GitHub Im Folgenden sind einige nützliche `crictl` Befehle aufgeführt.

Listet die Pods auf, die auf dem Host ausgeführt werden:

```
crictl pods
```

Listet die Container auf, die auf dem Host laufen:

```
crictl ps
```

Listet die auf dem Host laufenden Bilder auf:

```
crictl images
```

Ruft die Protokolle eines Containers ab, der auf dem Host läuft:

```
crictl logs CONTAINER_NAME
```

Holen Sie sich Statistiken über Pods, die auf dem Host ausgeführt werden:

```
crictl statsp
```

Netzwerkverkehr der Anwendung

Bei der Verwendung von Hybridknoten ist es wichtig, die Netzwerkflüsse Ihres Anwendungsverkehrs und die Technologien, mit denen Sie Ihre Anwendungen extern in Ihrem Cluster bereitstellen, zu berücksichtigen und zu verstehen. Verschiedene Technologien für den Lastenausgleich und den eingehenden Datenverkehr von Anwendungen verhalten sich bei Netzwerkunterbrechungen unterschiedlich. Wenn Sie beispielsweise die BGP Control Plane-Funktion von Cilium für den Lastenausgleich von Anwendungen verwenden, kann es sein, dass die BGP-Sitzung für Ihre Pods und Dienste während Netzwerkunterbrechungen unterbrochen wird. Dies liegt daran, dass die BGP-Lautsprecherfunktion in den Cilium-Agent integriert ist und der Cilium-Agent kontinuierlich neu gestartet wird, wenn er von der Kubernetes-Steuerebene getrennt wird. Der Grund für den Neustart liegt darin, dass die Integritätsprüfung von Cilium fehlschlägt, weil sein Zustand mit dem Zugriff auf die Kubernetes-Steuerebene verknüpft ist (siehe CFP: #31702 mit einer Opt-in-Verbesserung in

Cilium v1.17). Wenn Sie Application Load Balancers (ALB) oder Network Load Balancers (NLB) für Anwendungsdatenverkehr aus der AWS-Region verwenden, kann dieser Datenverkehr ebenfalls vorübergehend ausfallen, wenn Ihre lokale Umgebung die Konnektivität zur AWS-Region verliert. Es wird empfohlen, vor der Bereitstellung in der Produktion zu überprüfen, ob die Technologien, die Sie für Load Balancing und Ingress verwenden, auch bei Netzwerkunterbrechungen stabil bleiben. Das Beispiel im AWS-Samples/ eks-hybrid-examples GitHub Repo verwendet MetalB für den Lastenausgleich im L2-Modus, der auch bei Netzwerkunterbrechungen zwischen Hybridknoten und der EKS-Steuerebene stabil bleibt.

Überprüfen Sie die Abhängigkeiten von AWS-Remote-Services

Beachten Sie bei der Verwendung von Hybridknoten die Abhängigkeiten, die Sie von regionalen AWS-Services eingehen, die sich außerhalb Ihrer lokalen oder Edge-Umgebung befinden. Beispiele hierfür sind der Zugriff auf Amazon S3 oder Amazon RDS für Anwendungsdaten, die Verwendung von Amazon Managed Service für Prometheus oder CloudWatch für Metriken und Protokolle, die Verwendung von Application- und Network Load Balancern für regionalen Datenverkehr und das Abrufen von Containern aus Amazon Elastic Container Registry. Auf diese Services kann bei Netzwerkunterbrechungen zwischen Ihrer lokalen Umgebung und AWS nicht zugegriffen werden. Wenn Ihre lokale Umgebung anfällig für Netzwerkunterbrechungen mit AWS ist, überprüfen Sie Ihre Nutzung der AWS-Services und stellen Sie sicher, dass der Verlust der Verbindung zu diesen Services die statische Stabilität Ihrer Anwendungen nicht beeinträchtigt.

Optimieren Sie das Failover-Verhalten von Kubernetes-Pods

Es gibt Optionen zum Optimieren des Pod-Failover-Verhaltens bei Netzwerkunterbrechungen für Anwendungen, die nicht zwischen Hosts portierbar sind, oder für Umgebungen mit eingeschränkten Ressourcen, die keine freien Kapazitäten für Pod-Failover haben. Im Allgemeinen ist es wichtig, die Ressourcenanforderungen Ihrer Anwendungen zu berücksichtigen und über genügend Kapazität zu verfügen, damit eine oder mehrere Instanzen der Anwendung bei einem Ausfall eines Nodes ein Failover auf einen anderen Host durchführen können.

- Option 1 — Verwendung DaemonSets: Diese Option gilt für Anwendungen, die auf allen Knoten im Cluster ausgeführt werden können und sollten. DaemonSets werden automatisch so konfiguriert, dass sie den unerreichbaren Makel tolerieren, der dafür sorgt, dass DaemonSet Pods auch bei Netzwerkunterbrechungen an ihre Knoten gebunden bleiben.
- Option 2 — Auf unerreichbaren Makel einstellen `tolerationSeconds`: Sie können einstellen, wie lange Ihre Pods bei Netzwerkunterbrechungen an Knoten gebunden bleiben. Konfigurieren Sie dazu die Anwendungs-Pods so, dass sie den unerreichbaren Makel mit der `NoExecute` Wirkung

für eine von Ihnen (`tolerationSeconds`in der Anwendungsspezifikation) festgelegte Dauer tolerieren. Mit dieser Option bleiben Ihre Anwendungs-Pods bei Netzwerkunterbrechungen an Knoten gebunden, bis sie ablaufen. `tolerationSeconds` Denken Sie sorgfältig darüber nach, denn wenn Sie den `tolerationSeconds` Fehler „Unerreichbar“ mit erhöhen, kann es länger `NoExecute` dauern, bis Pods, die auf nicht erreichbaren Hosts laufen, auf andere erreichbare, fehlerfreie Hosts verschoben werden.

- Option 3: Benutzerdefinierter Controller: Sie können einen benutzerdefinierten Controller (oder eine andere Software) erstellen und ausführen, der Kubernetes auf den unerreichbaren Makel mit dem Effekt überwacht. `NoExecute` Wenn dieser Fehler erkannt wird, kann der benutzerdefinierte Controller anwendungsspezifische Metriken überprüfen, um den Zustand der Anwendung zu beurteilen. Wenn die Anwendung fehlerfrei ist, kann der benutzerdefinierte Controller den unerreichbaren Makel entfernen und so verhindern, dass bei Netzwerkunterbrechungen Pods von den Knoten entfernt werden.

Im Folgenden finden Sie ein Beispiel dafür, wie Sie ein Deployment `tolerationSeconds` für den unerreichbaren Makel konfigurieren können. Im Beispiel `tolerationSeconds` ist auf `1800` (30 Minuten) gesetzt, was bedeutet, dass Pods, die auf nicht erreichbaren Knoten laufen, nur entfernt werden, wenn die Netzwerkunterbrechung länger als 30 Minuten dauert.

```
apiVersion: apps/v1
kind: Deployment
metadata:
...
spec:
...
      tolerations:
      - key: "node.kubernetes.io/unreachable"
        operator: "Exists"
        effect: "NoExecute"
        tolerationSeconds: 1800
```

Kubernetes-Pod-Failover durch Netzwerkunterbrechungen

Wir beginnen mit einem Überblick über die wichtigsten Konzepte, Komponenten und Einstellungen, die das Verhalten von Kubernetes bei Netzwerkunterbrechungen zwischen Knoten und der Kubernetes-Steuerebene beeinflussen. EKS ist Upstream-Kubernetes-konform, sodass alle hier beschriebenen Kubernetes-Konzepte, -Komponenten und -Einstellungen für EKS- und EKS-Hybridknoten-Bereitstellungen gelten.

Es gibt Verbesserungen, die speziell an EKS vorgenommen wurden, um das Pod-Failover-Verhalten bei Netzwerkunterbrechungen zu verbessern. Weitere Informationen finden Sie in den GitHub Ausgaben #131294 und #131481 im Upstream-Kubernetes-Repository.

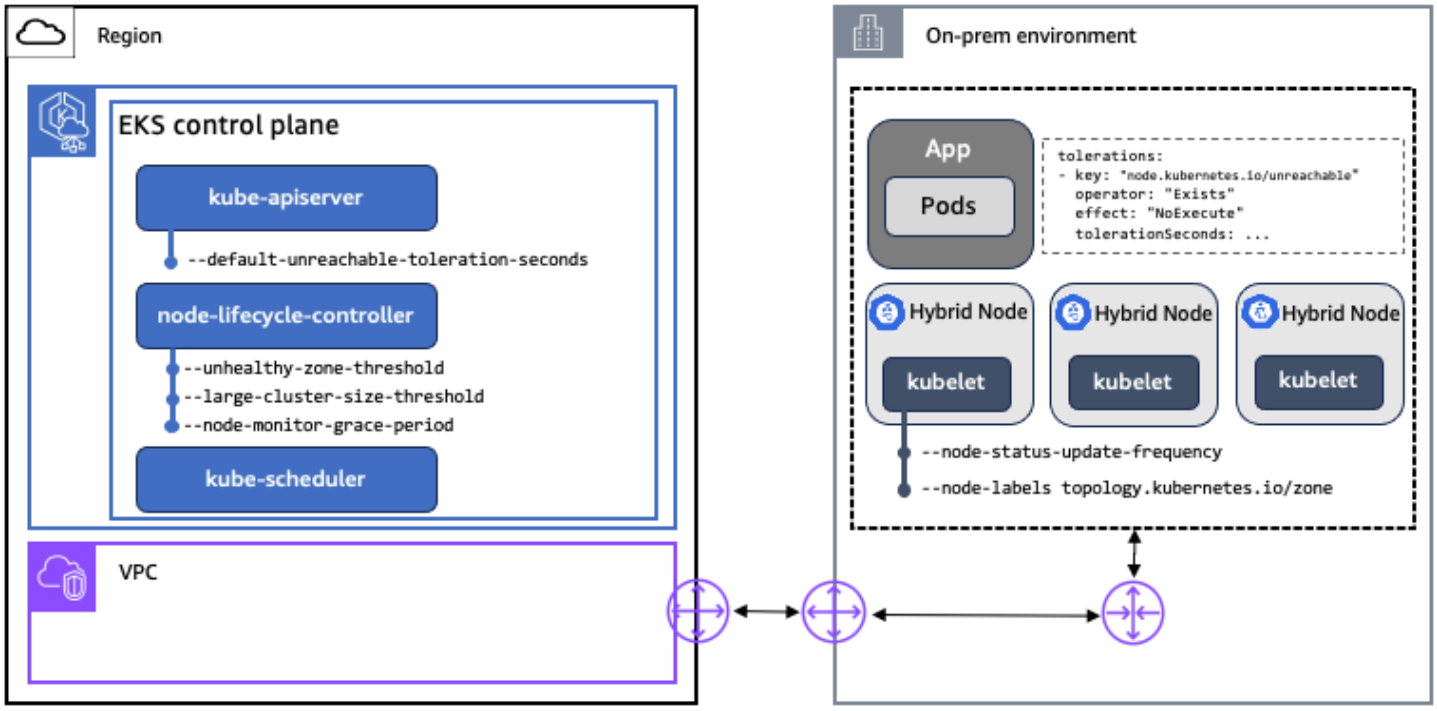
Konzepte

Taints and Tolerations: Taints and Tolerations werden in Kubernetes verwendet, um die Planung von Pods auf Knoten zu kontrollieren. Taints werden von der gesetzt, node-lifecycle-controller um darauf hinzuweisen, dass Knoten nicht für die Planung in Frage kommen oder dass Pods auf diesen Knoten entfernt werden sollten. Wenn Knoten aufgrund einer Netzwerkunterbrechung nicht erreichbar sind, gilt das für node.kubernetes. node-lifecycle-controller io/unreachable taint with a NoSchedule effect, and with a NoExecute effect if certain conditions are met. The node.kubernetes.io/unreachableTaint bedeutet, dass Ready Unknown ist. NodeCondition Benutzer können Toleranzen für Taints auf Anwendungsebene in angeben. PodSpec

- NoSchedule: Auf dem fehlerhaften Knoten sind keine neuen Pods geplant, es sei denn, sie haben eine entsprechende Toleranz. Pods, die bereits auf dem Knoten laufen, werden nicht entfernt.
- NoExecute: Pods, die den Makel nicht vertragen, werden sofort entfernt. Pods, die den Taint tolerieren (ohne TolerationSeconds anzugeben), bleiben für immer gebunden. Pods, die den Taint mit einem bestimmten TolerationSeconds-Wert tolerieren, bleiben für die angegebene Zeit gebunden. Nach Ablauf dieser Zeit entfernt der Node Lifecycle Controller die Pods aus dem Knoten.

Node-Leases: Kubernetes verwendet die Lease-API, um die Heartbeats des Kubelet-Knotens an den Kubernetes-API-Server zu übermitteln. Für jeden Knoten gibt es ein Lease-Objekt mit einem passenden Namen. Intern aktualisiert jeder Kubelet-Heartbeat das Feld spec.RenewTime des Lease-Objekts. Die Kubernetes-Steuerebene verwendet den Zeitstempel dieses Felds, um die Verfügbarkeit des Knotens zu bestimmen. Wenn Knoten von der Kubernetes-Steuerebene getrennt werden, können sie Spec.RenewTime nicht für ihren Lease aktualisieren, und die Kontrollebene interpretiert dies als „Ready being Unknown“. NodeCondition

Komponenten



Komponente	Unterkomponente	Description
Kubernetes-Steuerebene	kube-api-server	Der API-Server ist eine Kernkomponente der Kubernetes-Steuerebene, die die Kubernetes-API verfügbar macht.
Kubernetes-Steuerebene	node-lifecycle-controller	Einer der Controller, die der ausführt. kube-controller-manager Er ist dafür verantwortlich, Knotenprobleme zu erkennen und darauf zu reagieren.
Kubernetes-Steuerebene	Kube-Scheduler	Eine Komponente der Steuerungsebene, die nach neu erstellten Pods ohne zugewiesenen Knoten sucht und einen Knoten auswählt, auf dem sie ausgeführt werden sollen.
Kubernetes-Knoten	kubelet	Ein Agent, der auf jedem Knoten im Cluster ausgeführt wird. Das Kubelet überwacht PodSpecs und stellt sicher, dass die darin beschriebenen Container laufen und PodSpecs fehlerfrei sind.

Konfigurationseinstellungen

Komponente	Einstellung	Description	Die Standardeinstellung von K8	EKS-Standard	In EKS konfigurierbar
kube-api-server	default-unreachable-toleration-seconds	Gibt an, wie hoch die Toleranz `unreachable:NoExecute` dafür ist, die standardmäßig jedem Pod hinzugefügt wird, der noch nicht über eine solche Toleranz verfügt. `tolerationSeconds`	300	300	Nein
node-lifecycle-controller	node-monitor-grace-period	Gibt an, wie lange ein Knoten nicht reagieren kann, bevor er als fehlerhaft markiert wird. Muss N-mal höher sein als der Wert von Kubelet`nodeStatusUpdateFrequency`, wobei N die Anzahl der Wiederholungen ist, die das Kubelet zum Post-Node-Status benötigt.	40	40	Nein
node-lifecycle-controller	large-cluster-size-threshold	Die Anzahl der Knoten, bei denen der den Cluster aus Gründen der node-lifecycle-controller Räumungslogik als groß behandelt. `--secondary-node-eviction-rate` wird für Cluster dieser Größe oder kleiner auf 0 überschrieben.	50	100 000	Nein
node-lifecycle-controller	unhealthy-zone-threshold	Der Prozentsatz der Knoten in einer Zone, die nicht bereit sein müssen, damit diese Zone als fehlerhaft behandelt wird.	55%	55%	Nein
kubelet	node-status-update-frequency	Wie oft sendet das Kubelet den Knotenstatus an die Kontrollebene. Muss mit in	10	10	Ja

Kompor e	Einstellung	Description	Die Standar instellur g von K8	EKS-Stand ard	In EKS konfiguri erbar
		kompatibel `nodeMonitorGracePe riod` sein. node-lifecycle-controller			
kubelet	Knotenbes chriftungen	Labels, die bei der Registrierung des Knotens im Cluster hinzugefügt werden sollen. Das Label `topology. kubernetes.io/zone` kann mit Hybridknoten angegeben werden, um Knoten in Zonen zu gruppieren.	Keine	Keine	Ja

Kubernetes-Pod-Failover durch Netzwerkunterbrechungen

Das hier beschriebene Verhalten setzt voraus, dass Pods als Kubernetes-Bereitstellungen mit Standardeinstellungen ausgeführt werden und dass EKS als Kubernetes-Anbieter verwendet wird. Das tatsächliche Verhalten kann je nach Umgebung, Art der Netzwerktrennung, Anwendungen, Abhängigkeiten und Clusterkonfiguration unterschiedlich sein. Der Inhalt dieses Handbuchs wurde anhand einer bestimmten Anwendung, einer Clusterkonfiguration und einer Teilmenge von Plugins validiert. Es wird dringend empfohlen, das Verhalten in Ihrer eigenen Umgebung und mit Ihren eigenen Anwendungen zu testen, bevor Sie zur Produktion übergehen.

Wenn Netzwerkverbindungen zwischen Knoten und der Kubernetes-Steuerebene unterbrochen werden, kann das Kubelet auf jedem getrennten Knoten nicht mit der Kubernetes-Steuerebene kommunizieren. Folglich kann das Kubelet keine Pods auf diesen Knoten entfernen, bis die Verbindung wiederhergestellt ist. Das bedeutet, dass Pods, die vor der Netzwerkunterbrechung auf diesen Knoten ausgeführt wurden, auch während der Verbindungsunterbrechung weiterlaufen, vorausgesetzt, sie werden nicht durch andere Fehler heruntergefahren. Zusammenfassend lässt sich sagen, dass Sie bei Netzwerkunterbrechungen zwischen Knoten und der Kubernetes-Steuerebene statische Stabilität erreichen können, aber Sie können keine Mutationsoperationen an Ihren Knoten oder Workloads durchführen, bis die Verbindung wiederhergestellt ist.

Es gibt fünf Hauptszenarien, die je nach Art der Netzwerkunterbrechung zu unterschiedlichen Pod-Failover-Verhaltensweisen führen. In allen Szenarien wird der Cluster ohne Eingreifen des

Bedieners wieder funktionsfähig, sobald die Knoten wieder mit der Kubernetes-Steuerebene verbunden sind. In den folgenden Szenarien werden die erwarteten Ergebnisse auf der Grundlage unserer Beobachtungen beschrieben. Diese Ergebnisse gelten jedoch möglicherweise nicht für alle möglichen Anwendungs- und Clusterkonfigurationen.

Szenario 1: Vollständige Unterbrechung des Clusters

Erwartetes Ergebnis: Pods auf nicht erreichbaren Knoten werden nicht entfernt und laufen auf diesen Knoten weiter.

Eine vollständige Clusterunterbrechung bedeutet, dass alle Knoten im Cluster von der Kubernetes-Steuerebene getrennt sind. In diesem Szenario erkennt das System node-lifecycle-controller auf der Steuerungsebene, dass alle Knoten im Cluster nicht erreichbar sind, und bricht alle Pod-Räumungen ab.

Clusteradministratoren sehen während der Verbindungsunterbrechung alle Knoten mit Status`Not Ready`. Der Pod-Status ändert sich nicht, und während der Verbindungsunterbrechung und der anschließenden Wiederverbindung sind auf keinem der Knoten neue Pods geplant.

Szenario 2: Vollständige Zonenunterbrechung

Erwartetes Ergebnis: Pods auf nicht erreichbaren Knoten werden nicht entfernt und laufen auf diesen Knoten weiter.

Eine vollständige Zonenunterbrechung bedeutet, dass alle Knoten in der Zone von der Kubernetes-Steuerebene getrennt sind. In diesem Szenario erkennt die node-lifecycle-controller Kontrollebene, dass alle Knoten in der Zone nicht erreichbar sind, und storniert alle Pod-Räumungen.

Clusteradministratoren sehen während der Verbindungsunterbrechung alle Knoten mit Status`Not Ready`. Der Pod-Status ändert sich nicht, und während der Verbindungsunterbrechung und der anschließenden Wiederverbindung sind auf keinem der Knoten neue Pods geplant.

Szenario 3: Störung der meisten Zonen

Erwartetes Ergebnis: Pods auf nicht erreichbaren Knoten werden nicht entfernt und laufen auf diesen Knoten weiter.

Eine mehrheitliche Zonenunterbrechung bedeutet, dass die meisten Knoten in einer bestimmten Zone von der Kubernetes-Steuerebene getrennt sind. Zonen in Kubernetes werden durch Knoten mit derselben Bezeichnung definiert. `topology.kubernetes.io/zone` Wenn im Cluster keine Zonen

definiert sind, bedeutet eine mehrheitliche Unterbrechung, dass die meisten Knoten im gesamten Cluster getrennt sind. Standardmäßig wird eine Mehrheit durch den node-lifecycle-controller s definiertunhealthy-zone-threshold, der sowohl in Kubernetes als auch in EKS auf 55% festgelegt ist. Da in EKS auf 100.000 festgelegt large-cluster-size-threshold ist, werden Pod-Räumungen abgebrochen, wenn 55% oder mehr der Knoten in einer Zone nicht erreichbar sind (da die meisten Cluster weitaus kleiner als 100.000 Knoten sind).

Clusteradministratoren werden sehen, dass die meisten Knoten in der Zone Not Ready während der Verbindungsunterbrechung einen Status haben, aber der Status der Pods wird sich nicht ändern, und sie werden nicht auf anderen Knoten neu geplant.

Beachten Sie, dass das obige Verhalten nur für Cluster gilt, die größer als drei Knoten sind. In Clustern mit drei oder weniger Knoten wird die Räumung von Pods auf nicht erreichbaren Knoten geplant, und neue Pods werden auf intakten Knoten geplant.

Während der Tests haben wir gelegentlich beobachtet, dass Pods bei Netzwerkunterbrechungen von genau einem nicht erreichbaren Knoten entfernt wurden, selbst wenn ein Großteil der Knoten der Zone nicht erreichbar war. Wir untersuchen immer noch eine mögliche Rennbedingung in Kubernetes node-lifecycle-controller als Ursache für dieses Verhalten.

Szenario 4: Störung der Minderheitenzonen

Erwartetes Ergebnis: Pods werden von Knoten entfernt, die nicht erreichbar sind, und neue Pods werden auf verfügbaren, geeigneten Knoten geplant.

Eine geringfügige Störung bedeutet, dass ein kleinerer Prozentsatz der Knoten in einer Zone von der Kubernetes-Steuerebene getrennt wird. Wenn im Cluster keine Zonen definiert sind, bedeutet eine Störung einer Minderheit, dass die Minderheit der Knoten im gesamten Cluster getrennt ist. Wie bereits erwähnt, wird die Minderheit durch die unhealthy-zone-threshold Einstellung von definiert node-lifecycle-controller, die standardmäßig 55% beträgt. Wenn in diesem Szenario die Netzwerkunterbrechung länger als default-unreachable-toleration-seconds (5 Minuten) und node-monitor-grace-period (40 Sekunden) dauert und weniger als 55% der Knoten in einer Zone nicht erreichbar sind, werden neue Pods auf fehlerfreien Knoten geplant, während Pods auf nicht erreichbaren Knoten zur Räumung markiert werden.

Clusteradministratoren werden sehen, dass neue Pods auf fehlerfreien Knoten erstellt wurden, und die Pods auf getrennten Knoten werden als angezeigt. Terminating Denken Sie daran, dass Pods auf getrennten Knoten zwar einen Terminating Status haben, aber erst dann vollständig gelöscht werden, wenn der Knoten wieder eine Verbindung zur Kubernetes-Steuerebene herstellt.

Szenario 5: Neustart des Knotens während einer Netzwerkunterbrechung

Erwartetes Ergebnis: Pods auf nicht erreichbaren Knoten werden erst gestartet, wenn sich die Knoten wieder mit der Kubernetes-Steuerebene verbinden. Der Pod-Failover folgt der in den Szenarien 1—3 beschriebenen Logik, abhängig von der Anzahl der nicht erreichbaren Knoten.

Ein Knotenneustart während einer Netzwerkunterbrechung bedeutet, dass gleichzeitig mit einer Netzwerkunterbrechung ein weiterer Fehler (z. B. ein Stromausfall, ein out-of-memory Ereignis oder ein anderes Problem) auf einem Knoten aufgetreten ist. Die Pods, die zu Beginn der Netzwerkunterbrechung auf diesem Knoten liefen, werden während der Verbindungsunterbrechung nicht automatisch neu gestartet, wenn das Kubelet ebenfalls neu gestartet wurde. Das Kubelet fragt beim Start den Kubernetes-API-Server ab, um zu erfahren, welche Pods es ausführen soll. Wenn das Kubelet den API-Server aufgrund einer Netzwerkunterbrechung nicht erreichen kann, kann es die zum Starten der Pods erforderlichen Informationen nicht abrufen.

In diesem Szenario können lokale Tools zur Fehlerbehebung wie die `crictl` CLI nicht verwendet werden, um Pods manuell als „Glasbruch“ zu starten. Kubernetes entfernt normalerweise fehlgeschlagene Pods und erstellt neue, anstatt bestehende Pods neu zu starten (weitere Informationen finden Sie unter #10213 containerd GitHub Containerd-Repo). Statische Pods sind das einzige Kubernetes-Workload-Objekt, das vom Kubelet gesteuert wird und in diesen Szenarien neu gestartet werden kann. Es wird jedoch generell nicht empfohlen, statische Pods für Anwendungsbereitstellungen zu verwenden. Stellen Sie stattdessen mehrere Replikate auf verschiedenen Hosts bereit, um die Anwendungsverfügbarkeit bei mehreren gleichzeitigen Ausfällen sicherzustellen, z. B. bei einem Knotenausfall und einer Netzwerkunterbrechung zwischen Ihren Knoten und der Kubernetes-Steuerebene.

Netzwerkverkehr von Anwendungen durch Netzwerkunterbrechungen

Die Themen auf dieser Seite beziehen sich auf Kubernetes-Clusternetzwerke und den Anwendungsdatenverkehr bei Netzwerkunterbrechungen zwischen Knoten und der Kubernetes-Steuerebene.

Cilium

Cilium bietet mehrere Modi für IP-Adressverwaltung (IPAM), Kapselung, Lastenausgleich und Cluster-Routing. Die in diesem Handbuch validierten Modi verwendeten Cluster Scope IPAM, VXLAN-Overlay, BGP-Loadbalancing und Kube-Proxy. Cilium wurde auch ohne BGP-Loadbalancing verwendet und durch MetalLB L2-Load Balancing ersetzt.

Die Basis der Cilium-Installation besteht aus dem Cilium-Operator und den Cilium-Agenten. Der Cilium-Operator wird als Deployment ausgeführt und registriert die Cilium Custom Resource Definitions (CRDs), verwaltet IPAM und synchronisiert Cluster-Objekte unter anderem mit dem Kubernetes-API-Server. Die Cilium-Agenten laufen auf jedem Knoten als DaemonSet und verwalten die eBPF-Programme, um die Netzwerkregeln für Workloads zu kontrollieren, die auf dem Cluster ausgeführt werden.

Im Allgemeinen bleibt das von Cilium konfigurierte In-Cluster-Routing auch bei Netzwerkunterbrechungen verfügbar und an Ort und Stelle. Dies kann anhand der Cluster-Datenflüsse und der IP-Tabellenregeln (iptables) für das Pod-Netzwerk bestätigt werden.

```
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
...
```

Bei Netzwerkunterbrechungen werden der Cilium-Operator und die Cilium-Agenten jedoch neu gestartet, da ihre Integritätsprüfungen mit dem Zustand der Verbindung mit dem Kubernetes-API-Server verknüpft sind. Bei Netzwerkunterbrechungen wird in den Protokollen des Cilium-Operators und der Cilium-Agenten voraussichtlich Folgendes angezeigt: Während der Netzwerkunterbrechungen können Sie Tools wie die `crictl` CLI verwenden, um die Neustarts dieser Komponenten einschließlich ihrer Protokolle zu beobachten.

```
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
```

Wenn Sie die BGP Control Plane-Funktion von Cilium für den Lastenausgleich von Anwendungen verwenden, kann es sein, dass die BGP-Sitzung für Ihre Pods und Dienste während Netzwerkunterbrechungen unterbrochen wird, da die BGP-Lautsprecherfunktion in den Cilium-Agent integriert ist und der Cilium-Agent kontinuierlich neu gestartet wird, wenn die Verbindung zur Kubernetes-Steuerebene getrennt wird. Weitere Informationen finden Sie im Cilium BGP Control Plane Operation Guide in der Cilium-Dokumentation. Außerdem werden die Cilium-Routen bei diesen Aktionen nicht beibehalten, wenn während einer Netzwerkunterbrechung, z. B. bei einem Netzwechsel oder einem Maschinenneustart, gleichzeitig ein Fehler auftritt. Die Routen werden jedoch neu erstellt, wenn der Knoten wieder eine Verbindung zur Kubernetes-Steuerebene herstellt und Cilium erneut gestartet wird.

Calico

Kommt bald

Metall B

MetalLB hat zwei Modi für den Lastenausgleich: den L2-Modus und den BGP-Modus. Einzelheiten zur Funktionsweise dieser Lastenausgleichsmodi und zu ihren Einschränkungen finden Sie in der MetalLB-Dokumentation. Bei der Validierung dieses Handbuchs wurde MetalLB im L2-Modus verwendet, wobei ein Computer im Cluster den Besitz des Kubernetes-Dienstes übernimmt und ARP verwendet, um die IP-Adressen des Load Balancers im lokalen Netzwerk erreichbar IPv4 zu machen. Bei der Ausführung von MetalB gibt es einen Controller, der für die IP-Zuweisung verantwortlich ist, und Lautsprecher, die auf jedem Knoten laufen und für die Werbung von Diensten mit zugewiesenen IP-Adressen verantwortlich sind. Der MetalB-Controller läuft als Deployment und die MetalB-Lautsprecher laufen als. DaemonSet Bei Netzwerkunterbrechungen können der MetalB-Controller und die Lautsprecher den Kubernetes-API-Server nicht nach Cluster-Ressourcen durchsuchen, sondern laufen weiter. Am wichtigsten ist, dass die Dienste, die MetalB für externe Konnektivität verwenden, auch bei Netzwerkunterbrechungen verfügbar und zugänglich bleiben.

kube-proxy

In EKS-Clustern wird Kube-Proxy DaemonSet auf jedem Knoten als a ausgeführt und ist für die Verwaltung von Netzwerkregeln verantwortlich, um die Kommunikation zwischen Diensten und Pods zu ermöglichen, indem Dienst-IP-Adressen in die IP-Adressen der zugrunde liegenden Pods übersetzt werden. Die von kube-proxy konfigurierten Regeln für IP-Tabellen (iptables) werden bei Netzwerkunterbrechungen beibehalten, und das Routing innerhalb des Clusters funktioniert weiterhin und die Kube-Proxy-Pods werden weiterhin ausgeführt.

Sie können die Kube-Proxy-Regeln mit den folgenden iptables-Befehlen einhalten. Der erste Befehl zeigt, dass Pakete, die die Kette durchlaufen, an die `PREROUTING` Kette weitergeleitet werden. `KUBE-SERVICES`

```
iptables -t nat -L PREROUTING
```

```
Chain PREROUTING (policy ACCEPT)
target         prot opt source      destination
KUBE-SERVICES  all  --  anywhere    anywhere      /* kubernetes service portals */
```

Wenn wir uns die `KUBE-SERVICES` Kette ansehen, können wir die Regeln für die verschiedenen Clusterdienste sehen.

```
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 */
```

Wenn wir die Kette des Frontend-Dienstes für die Anwendung untersuchen, können wir die Pod-IP-Adressen sehen, die den Dienst unterstützen.

```
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 */
```

Die folgenden Kube-Proxy-Protokollmeldungen werden bei Netzwerkunterbrechungen erwartet, wenn versucht wird, den Kubernetes-API-Server auf Aktualisierungen der Knoten- und Endpunktressourcen zu überprüfen.

```
"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

Standardmäßig verwenden Pods in EKS-Clustern die CoreDNS-Cluster-IP-Adresse als Nameserver für DNS-Abfragen im Cluster. In EKS-Clustern wird CoreDNS als Deployment auf Knoten ausgeführt.

Mit Hybridknoten können Pods bei Netzwerkunterbrechungen weiterhin mit dem CoreDNS kommunizieren, wenn CoreDNS-Replikate lokal auf Hybridknoten ausgeführt werden. Wenn Sie einen EKS-Cluster mit Knoten in der Cloud und Hybridknoten in Ihrer lokalen Umgebung haben, wird empfohlen, in jeder Umgebung mindestens ein CoreDNS-Replikat zu haben. CoreDNS bedient weiterhin DNS-Abfragen für Datensätze, die vor der Netzwerkunterbrechung erstellt wurden, und führt aus Gründen der statischen Stabilität weiterhin die Netzwerkwiederverbindung durch.

Die folgenden CoreDNS-Protokollmeldungen werden bei Netzwerkunterbrechungen erwartet, wenn versucht wird, Objekte vom Kubernetes-API-Server aufzulisten.

```
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
```

Host-Anmeldeinformationen bei Netzwerkunterbrechungen

EKS Hybrid Nodes ist in Hybridaktivierungen von AWS Systems Manager (SSM) und AWS IAM Roles Anywhere für temporäre IAM-Anmeldeinformationen integriert, die zur Authentifizierung des Knotens mit der EKS-Steuerebene verwendet werden. Sowohl SSM als auch IAM Roles Anywhere aktualisieren automatisch die temporären Anmeldeinformationen, die sie auf lokalen Hosts verwalten. Es wird empfohlen, für alle Hybridknoten in Ihrem Cluster einen einzigen Anmeldeinformationsanbieter zu verwenden — entweder SSM-Hybrid-Aktivierungen oder IAM Roles Anywhere, aber nicht beides.

SSM-Hybrid-Aktivierungen

Die von SSM bereitgestellten temporären Anmeldeinformationen sind eine Stunde lang gültig. Sie können die Gültigkeitsdauer der Anmeldeinformationen nicht ändern, wenn Sie SSM als Ihren Anmeldeinformationsanbieter verwenden. Die temporären Anmeldeinformationen werden von SSM automatisch rotiert, bevor sie ablaufen, und die Rotation hat keinen Einfluss auf den Status Ihrer Knoten oder Anwendungen. Wenn es jedoch zu Netzwerkunterbrechungen zwischen dem SSM-Agent und dem regionalen SSM-Endpunkt kommt, kann SSM die Anmeldeinformationen nicht aktualisieren, und die Anmeldeinformationen laufen möglicherweise ab.

SSM verwendet exponentielles Backoff für Wiederholungsversuche bei der Aktualisierung der Anmeldeinformationen, wenn keine Verbindung zu den regionalen SSM-Endpunkten hergestellt werden kann. In der SSM-Agent-Version `3.3.808.0` und höher (veröffentlicht im August 2024) ist der exponentielle Backoff auf 30 Minuten begrenzt. Abhängig von der Dauer Ihrer Netzwerkunterbrechung kann es bis zu 30 Minuten dauern, bis SSM die Anmeldeinformationen aktualisiert hat, und Hybridknoten stellen erst wieder eine Verbindung zur EKS-Steuerebene her, wenn die Anmeldeinformationen aktualisiert wurden. In diesem Szenario können Sie den SSM-Agent neu starten, um eine Aktualisierung der Anmeldeinformationen zu erzwingen. Als Nebeneffekt des aktuellen Aktualisierungsverhaltens der SSM-Anmeldeinformationen können sich Knoten zu unterschiedlichen Zeiten erneut verbinden, je nachdem, wann der SSM-Agent auf jedem Knoten seine Anmeldeinformationen aktualisieren kann. Aus diesem Grund kann es zu einem Pod-Failover von Knoten, die noch nicht wieder verbunden wurden, zu Knoten kommen, die bereits wieder verbunden sind.

Holen Sie sich die SSM-Agent-Version. Sie können auch den Fleet Manager-Bereich der SSM-Konsole überprüfen:

```
# AL2023, RHEL
yum info amazon-ssm-agent
# Ubuntu
snap list amazon-ssm-agent
```

Starten Sie den SSM-Agenten neu:

```
# AL2023, RHEL
systemctl restart amazon-ssm-agent
# Ubuntu
systemctl restart snap.amazon-ssm-agent.amazon-ssm-agent
```

SSM-Agent-Protokolle anzeigen:

```
tail -f /var/log/amazon/ssm/amazon-ssm-agent.log
```

Erwartete Protokollmeldungen bei Netzwerkunterbrechungen:

```
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
```

IAM Roles Anywhere

Die von IAM Roles Anywhere bereitgestellten temporären Anmeldeinformationen sind standardmäßig für eine Stunde gültig. Sie können die Gültigkeitsdauer der Anmeldeinformationen mit IAM Roles Anywhere über das `durationSeconds`Feld in Ihrem IAM Roles Anywhere-Profil konfigurieren. Die maximale Gültigkeitsdauer der Anmeldeinformationen beträgt 12 Stunden. Die `MaxSessionDuration`Einstellung in Ihrer IAM-Rolle Hybrid Nodes muss größer sein als die `durationSeconds` Einstellung in Ihrem IAM Roles Anywhere-Profil.

Wenn Sie IAM Roles Anywhere als Anmeldeinformationsanbieter für Ihre Hybridknoten verwenden, erfolgt die Wiederherstellung der Verbindung zur EKS-Steuerebene nach Netzwerkunterbrechungen in der Regel innerhalb von Sekunden nach der Netzwerkwiederherstellung, da das Kubelet anruft, um Anmeldeinformationen bei Bedarf abzurufen. `aws_signing_helper credential-process` Wenn Sie IAM Roles Anywhere verwenden, können Sie Benachrichtigungen und Warnmeldungen für den Ablauf von Zertifikaten konfigurieren, obwohl dies nicht direkt mit Hybridknoten oder Netzwerkunterbrechungen zusammenhängt. Weitere Informationen finden Sie unter Anpassen der Benachrichtigungseinstellungen in IAM Roles Anywhere.

Bewährte Methoden für die Ausführung von AI/ML Workloads

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Durch die Implementierung von Best Practices bei der Ausführung von AI/ML Workloads auf EKS kann sichergestellt werden, dass diese Workloads leistungsfähig, kostengünstig, belastbar und mit angemessenen Ressourcen ausgestattet sind. Die bewährten Methoden sind in die folgenden allgemeinen Abschnitte unterteilt: Datenverarbeitung, Netzwerk, Speicher, Beobachtbarkeit und Leistung.

Feedback

Dieser Leitfaden wird am GitHub veröffentlicht, um direktes Feedback und Vorschläge von der breiteren EKS/Kubernetes Community zu sammeln. Wenn Sie eine bewährte Methode haben, die wir Ihrer Meinung nach in den Leitfaden aufnehmen sollten, reichen Sie bitte ein Problem ein oder reichen Sie eine PR im GitHub Repository ein. Wir beabsichtigen, den Leitfaden regelmäßig zu aktualisieren, sobald der Service um neue Funktionen erweitert wird oder wenn sich eine neue bewährte Methode herausstellt.

Berechnung und Autoscaling

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

GPU-Ressourcenoptimierung und Kostenmanagement

Planen Sie Workloads mit GPU-Anforderungen mithilfe von bekannten Labels

Für AI/ML Workloads, die empfindlich auf unterschiedliche GPU-Merkmale reagieren (z. B. GPU, GPU-Speicher), empfehlen wir, die GPU-Anforderungen anhand bekannter Scheduling-Labels

anzugeben, die von den mit Karpenter verwendeten Knotentypen und verwalteten Knotengruppen unterstützt werden. Wenn diese nicht definiert werden, kann dies dazu führen, dass Pods auf Instanzen mit unzureichenden GPU-Ressourcen geplant werden, was zu Ausfällen oder Leistungseinbußen führen kann. Wir empfehlen, NodeSelector oder Node Affinity zu verwenden, um anzugeben, auf welchem Knoten ein Pod ausgeführt werden soll, und Rechenressourcen (CPU, Speicher GPUs usw.) im Ressourcenbereich des Pods festzulegen.

Beispiel

Verwenden Sie beispielsweise den GPU-Namen-Node-Selektor, wenn Sie Karpenter verwenden:

```
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
```

Verwenden Sie das Kubernetes Device Plugin für die Bereitstellung GPUs

Für die Bereitstellung GPUs auf Knoten muss der NVIDIA-GPU-Treiber auf dem Betriebssystem des Knotens installiert und die Container-Laufzeit so konfiguriert sein, dass der Kubernetes-Scheduler Pods Knoten zuweisen kann, auf denen sie verfügbar sind. GPUs Der Einrichtungsprozess für das NVIDIA Kubernetes Device Plugin hängt vom verwendeten EKS Accelerated AMI ab:

- Bottlerocket Accelerated AMI: Dieses AMI beinhaltet den NVIDIA-GPU-Treiber und das NVIDIA Kubernetes Device Plugin ist vorinstalliert und sofort einsatzbereit, sodass GPU-Unterstützung sofort verfügbar ist. Für die Bereitstellung im Kubernetes-Scheduler ist keine zusätzliche Konfiguration erforderlich. GPUs
- AL2023 Beschleunigtes AMI: Dieses AMI enthält den NVIDIA-GPU-Treiber, aber das NVIDIA Kubernetes Device Plugin ist nicht vorinstalliert. Sie müssen das Geräte-Plug-In separat installieren und konfigurieren, in der Regel über ein. DaemonSet Beachten Sie, dass, wenn Sie eksctl verwenden, um Ihren Cluster zu erstellen und einen GPU-Instanztyp (z. B. `g5.xlarge`) in Ihrem

angeben ClusterConfig, `eksctl` automatisch das entsprechende AMI ausgewählt und das NVIDIA Kubernetes Device Plugin installiert wird. Weitere Informationen finden Sie unter GPU-Unterstützung in der eksctl-Dokumentation.

Wenn Sie stattdessen den Operator EKS Accelerated AMIs und NVIDIA GPU verwenden möchten, um Komponenten wie das NVIDIA Kubernetes-Geräte-Plugin zu verwalten, beachten Sie, dass Sie die Verwaltung des NVIDIA-GPU-Treibers und des NVIDIA-Container-Toolkits gemäß der NVIDIA-Dokumentation zu den vorinstallierten NVIDIA-GPU-Treibern und dem NVIDIA-Container-Toolkit deaktivieren müssen.

Um zu überprüfen, ob das NVIDIA-Geräte-Plug-in aktiv ist und GPUs korrekt angezeigt wird, führen Sie folgenden Befehl aus:

```
kubectl describe node | grep nvidia.com/gpu
```

Dieser Befehl überprüft, ob die `nvidia.com/gpu` Ressource die Kapazität des Knotens und die zuweisbaren Ressourcen erfüllt. Zum Beispiel sollte ein Knoten mit einer GPU angezeigt werden. `nvidia.com/gpu: 1` Weitere Informationen finden Sie im Kubernetes GPU Scheduling Guide.

Verwenden Sie viele verschiedene EC2-Instance-Typen

Die Verwendung so vieler verschiedener EC2-Instance-Typen wie möglich ist eine wichtige bewährte Methode für die Skalierbarkeit auf Amazon EKS, wie im the section called “Datenebene” Abschnitt beschrieben. Diese Empfehlung gilt auch für Instances mit beschleunigter Hardware (z. B. GPUs). Wenn Sie einen Cluster erstellen, der nur einen Instance-Typ verwendet, und versuchen, die Anzahl der Knoten über die Kapazität der Region hinaus zu skalieren, wird möglicherweise ein Fehler mit unzureichender Kapazität (ICE) angezeigt, der darauf hinweist, dass keine Instances verfügbar sind. Es ist wichtig, die einzigartigen Merkmale Ihrer AI/ML Workloads zu verstehen, bevor Sie sie beliebig diversifizieren. Überprüfen Sie die verfügbaren Instance-Typen mit dem EC2 Instance Type Explorer-Tool, um eine Liste von Instance-Typen zu erstellen, die Ihren spezifischen Rechenanforderungen entsprechen, und vermeiden Sie, die Art der Instances, die in Ihrem Cluster verwendet werden können, willkürlich einzuschränken.

Accelerated Compute Instances werden in verschiedenen Kaufmodellen angeboten, um kurzfristigen, mittelfristigen und stationären Workloads gerecht zu werden. Für kurzfristige, flexible und fehlertolerante Workloads, bei denen Sie eine Reservierung vermeiden möchten, sollten Sie sich Spot-Instances ansehen. Mit Kapazitätsblöcken, On-Demand-Instances und Speicherplänen können Sie beschleunigte Recheninstanzen für mittel- und langfristige Workloads bereitstellen. Um

die Chancen zu erhöhen, erfolgreich auf die benötigte Kapazität in Ihrer bevorzugten Kaufoption zuzugreifen, wird empfohlen, eine vielfältige Liste von Instance-Typen und Availability Zones zu verwenden. Wenn Sie auf ein bestimmtes Kaufmodell stoßen ICEs , versuchen Sie es alternativ erneut mit einem anderen Modell.

Beispiel Das folgende Beispiel zeigt, wie ein Karpenter in die Lage versetzt werden kann NodePool , G- und P-Instanzen bereitzustellen, die größer sind als die Generationen 3 (z. B. p3). Weitere Informationen finden Sie im Skalierbarkeit Abschnitt.

```
- 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
```

Einzelheiten zur Verwendung von Spot-Instances für GPUs finden Sie weiter unten unter „Erwägen Sie die Verwendung von Amazon EC2-Spot-Instances für GPUs mit Karpenter“.

Erwägen Sie die Verwendung von Amazon EC2-Spot-Instances für GPUs mit Karpenter

Mit Amazon EC2 Spot-Instances können Sie ungenutzte EC2-Kapazität in der AWS-Cloud nutzen und sind mit einem discount von bis zu 90% im Vergleich zu On-Demand-Preisen erhältlich. Amazon EC2 Spot-Instances können mit einer Frist von zwei Minuten unterbrochen werden, wenn EC2 die Kapazität wieder benötigt. Weitere Informationen finden Sie unter Spot-Instances im Amazon EC2 EC2-Benutzerhandbuch. Amazon EC2 Spot kann eine hervorragende Wahl für fehlertolerante, zustandslose und flexible Workloads (Zeit und Instanztyp) sein. Weitere Informationen darüber, wann Sie Spot-Instances verwenden sollten, finden Sie unter Bewährte Methoden für EC2-Spot-Instances. Sie können Spot-Instances auch für AI/ML Workloads verwenden, wenn sie Spot-freundlich sind.

Anwendungsfälle

Spot-freundliche Workloads können Big Data, containerisierte Workloads, CI/CD, statuslose Webserver, High Performance Computing (HPC) und Rendering-Workloads sein. Spot-Instances eignen sich nicht für Workloads, die unflexibel, zustandsbehaftet, fehlerintolerant oder eng zwischen Instanzknoten gekoppelt sind (z. B. Workloads mit parallel Prozessen, die bei der Berechnung stark voneinander abhängen und eine ständige Kommunikation zwischen den Knoten erfordern, wie MPI-basierte Hochleistungsrechneranwendungen wie Computational Fluid Dynamics oder verteilte

Datenbanken mit komplexen Interdependenzen). Hier sind die spezifischen Anwendungsfälle, die wir empfehlen (in keiner bestimmten Reihenfolge):

- Online-Inferenz in Echtzeit: Verwenden Sie Spot-Instances für eine kostenoptimierte Skalierung Ihrer Echtzeit-Inferenz-Workloads, sofern Ihre Workloads Spot-freundlich sind. Mit anderen Worten, die Inferenzzeit beträgt entweder weniger als zwei Minuten, die Anwendung ist fehlertolerant gegenüber Unterbrechungen und kann auf verschiedenen Instance-Typen ausgeführt werden. Stellen Sie eine hohe Verfügbarkeit durch Instanzvielfalt (z. B. über mehrere Instance-Typen und Availability Zones hinweg) oder Reservierungen sicher und implementieren Sie gleichzeitig Fehlertoleranz auf Anwendungsebene, um potenzielle Spot-Unterbrechungen zu bewältigen.
- Hyperparameter-Tuning: Verwenden Sie Spot-Instances, um explorative Optimierungsaufträge opportunistisch auszuführen, da Unterbrechungen ohne nennenswerte Verluste toleriert werden können, insbesondere bei Experimenten mit kurzer Dauer.
- Datenerweiterung: Verwenden Sie Spot-Instances, um Aufgaben zur Datenvorverarbeitung und -erweiterung durchzuführen, die bei einer Unterbrechung von Checkpoints aus neu gestartet werden können, was sie ideal für die variable Verfügbarkeit von Spot macht.
- Modelle zur Feinabstimmung: Verwenden Sie Spot-Instances für die Feinabstimmung mit robusten Checkpoint-Mechanismen, um den Vorgang mit dem zuletzt gespeicherten Status fortzusetzen und so die Auswirkungen von Instance-Unterbrechungen zu minimieren.
- Batch-Inferenz: Verwenden Sie Spot-Instances, um große Batch von Offline-Inferenzanfragen non-real-time so zu verarbeiten, dass Jobs angehalten und wieder aufgenommen werden können. Dies bietet die beste Abstimmung mit den Kosteneinsparungen von Spot und die Bewältigung potenzieller Unterbrechungen durch Wiederholungen oder Diversifizierung.
- Opportunistische Trainingsuntergruppen: Verwenden Sie Spot-Instances für marginale oder experimentelle Trainingsworkloads (z. B. kleinere Modelle mit 10 Millionen Parametern), bei denen Unterbrechungen akzeptabel sind und Effizienzoptimierungen wie Diversifizierung zwischen Instance-Typen oder Regionen angewendet werden können. Für Schulungen im Produktionsmaßstab ist dies jedoch aufgrund potenzieller Unterbrechungen nicht zu empfehlen.

Überlegungen

Um Spot-Instances für beschleunigte Workloads auf Amazon EKS zu verwenden, gibt es eine Reihe wichtiger Überlegungen (in keiner bestimmten Reihenfolge):

- Verwenden Sie Karpenter, um Spot-Instances mit aktivierter erweiterter Konsolidierung zu verwalten. Wenn Sie karpenter.sh/capacity-type in Ihrem Karpenter als „spot“ angeben, stellt

Karpenter NodePool Spot-Instances standardmäßig ohne zusätzliche Konfiguration bereit. Um jedoch eine erweiterte Spot-to-Spot Konsolidierung zu ermöglichen, bei der nicht ausgelastete Spot-Knoten durch kostengünstigere Spot-Alternativen ersetzt werden, müssen Sie das Feature-Gate aktivieren, indem Sie --feature-gates =true in den Controller-Argumenten von Karpenter oder SpotToSpotConsolidation über die Umgebungsvariable FEATURE_GATES setzen. SpotToSpotConsolidation Karpenter verwendet die Zuweisungsstrategie zur Bereitstellung von EC2-Instances. price-capacity-optimized Basierend auf den NodePool Anforderungen und Pod-Einschränkungen packt Karpenter unplanbare Pods in Bin-Packs und sendet eine Vielzahl von Instance-Typen an die Amazon EC2 Fleet API. Sie können das EC2 Instance Type Explorer-Tool verwenden, um eine Liste von Instance-Typen zu generieren, die Ihren spezifischen Rechenanforderungen entsprechen.

- Stellen Sie sicher, dass Workloads zustandslos, fehlertolerant und flexibel sind. Workloads müssen zustandslos, fehlertolerant und in Bezug auf ihre Größe flexibel sein. instance/GPU Dies ermöglicht eine reibungslose Wiederaufnahme nach Spot-Unterbrechungen, und die Flexibilität der Instances ermöglicht es Ihnen, möglicherweise länger vor Ort zu bleiben. Aktivieren Sie die Behandlung von Spot-Unterbrechungen in Karpenter, indem Sie den Helm-Wert settings.interruptionQueue mit dem Namen der AWS SQS-Warteschlange konfigurieren, um Spot-Unterbrechungsereignisse abzufangen. Verwenden Sie beispielsweise bei der Installation über Helm --set „settings.interruptionQueue=$ {CLUSTER_NAME}“. Ein Beispiel finden Sie im Handbuch Erste Schritte mit Karpenter. Wenn Karpenter ein Spot-Unterbrechungsereignis bemerkt, sperrt, verunreinigt, entleert und beendet es automatisch die Knoten vor dem Unterbrechungsereignis, um die Kündigungsfrist der Pods zu maximieren. Gleichzeitig startet Karpenter sofort einen neuen Knoten, sodass dieser so schnell wie möglich einsatzbereit ist.
- Vermeiden Sie es, die Auswahl des Instanztyps zu stark einzuschränken. Sie sollten es so weit wie möglich vermeiden, die Instanztypen einzuschränken. Wenn die Instance-Typen nicht eingeschränkt werden, ist die Wahrscheinlichkeit höher, Spot-Kapazität in großem Umfang mit einer geringeren Häufigkeit von Spot-Instance-Unterbrechungen zu geringeren Kosten zu erwerben. Vermeiden Sie es beispielsweise, sich auf bestimmte Typen zu beschränken (z. B. g5.xlarge). Erwägen Sie die Angabe verschiedener Instanzkategorien und -generationen mithilfe von Schlüsseln wie 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/MLDer Workload erfordert eine bestimmte oder begrenzte Anzahl von Acceleratoren, ist jedoch zwischen den Regionen flexibel. Sie können Spot Placement Score verwenden, um vor dem Start dynamisch die optimale Region für die Bereitstellung Ihres Workloads zu ermitteln.

- Erweitern Sie die NodePool Anforderungen auf eine größere Anzahl ähnlicher EC2-Instance-Familien. Jeder Spot-Instance-Pool besteht aus einer ungenutzten EC2-Instance-Kapazität für einen bestimmten Instance-Typ in einer bestimmten Availability Zone (AZ). Wenn Karpenter versucht, einen neuen Knoten bereitzustellen, wählt es einen Instance-Typ aus, der den NodePool Anforderungen entspricht. Wenn in keiner AZ ein kompatibler Instance-Typ über Spot-Kapazität verfügt, schlägt die Bereitstellung fehl. Um dieses Problem zu vermeiden, sollten Sie umfassendere Instanzen der G-Serie (Generation 4 oder höher) von NVIDIA für alle Größen und Availability Zones (AZs) zulassen und dabei Hardwareanforderungen wie GPU-Speicher oder Raytracing berücksichtigen. Da es sich bei Instances um unterschiedliche Typen handeln kann, müssen Sie sicherstellen, dass Ihr Workload auf jedem Typ ausgeführt werden kann und dass die Leistung, die Sie erhalten, Ihren Anforderungen entspricht.
- Nutzen Sie alle Verfügbarkeitszonen in einer Region. Die verfügbare Kapazität variiert je nach Availability Zone (AZ). Ein bestimmter Instance-Typ ist in einer AZ möglicherweise nicht verfügbar, in einer anderen jedoch reichlich vorhanden. Jede einzigartige Kombination aus einem Instance-Typ und einer Availability Zone bildet einen separaten Spot-Kapazitätspool. Wenn Sie innerhalb Ihrer NodePool Karpenter-Anforderungen für alle Kapazitäten AZs in einer Region anfragen, suchen Sie effektiv nach mehr Pools gleichzeitig. Dadurch wird die Anzahl der Spot-Kapazitätspools maximiert und somit die Wahrscheinlichkeit erhöht, Spot-Kapazität zu erwerben. Um dies zu erreichen, lassen Sie in Ihrer NodePool Konfiguration entweder den Schlüssel topology.kubernetes.io/zone komplett weg, damit Karpenter aus allen in der Region verfügbaren auswählen kann, oder führen Sie die Liste explizit AZs mit dem Operator: AZs In auf und geben Sie die Werte an (z. B. us-west-2a).
- Erwägen Sie, den Spot Placement Score (SPS) zu verwenden, um sich einen Überblick über die Wahrscheinlichkeit zu verschaffen, mit Spot-Instances erfolgreich auf die erforderliche Kapazität zuzugreifen. Der Spot Placement Score (SPS) ist ein Tool, das eine Bewertung liefert, anhand derer Sie beurteilen können, wie wahrscheinlich es ist, dass eine Spot-Anfrage erfolgreich ist. Wenn Sie SPS verwenden, geben Sie zunächst Ihre Rechenanforderungen für Ihre Spot-Instances an, und dann gibt Amazon EC2 die zehn wichtigsten Regionen oder Availability Zones (AZs) zurück, in denen Ihre Spot-Anfrage wahrscheinlich erfolgreich sein wird. Regionen und Availability Zones werden auf einer Skala von 1 bis 10 bewertet. Eine Punktzahl von 10 gibt an, dass Ihre Spot-Anfrage mit hoher Wahrscheinlichkeit erfolgreich sein wird, aber nicht garantiert werden kann. Ein Ergebnis von 1 zeigt an, dass Ihre Spot-Anforderung sehr wahrscheinlich nicht erfolgreich sein wird. Dasselbe Ergebnis kann für verschiedene Regionen oder Availability Zones zurückgegeben werden. Weitere Informationen finden Sie unter Anleitung zur Erstellung eines Spot Placement Score Tracker-Dashboards auf AWS. Da die Spot-Kapazität ständig schwankt, hilft Ihnen SPS dabei, herauszufinden, AZs welche Kombination von Instance-Typen und Regionen für Ihre

Workload-Einschränkungen (d. h. Flexibilität, Leistung, Größe usw.) am besten geeignet ist. Wenn Ihr AI/ML Workload eine bestimmte oder eine begrenzte Anzahl von Accelerators erfordert, aber zwischen den Regionen flexibel ist, können Sie den Spot-Platzierungswert verwenden, um vor dem Start dynamisch die optimale Region für die Bereitstellung Ihres Workloads zu ermitteln. Damit Sie automatisch ermitteln können, wie wahrscheinlich es ist, Spot-Kapazität zu erwerben, stellen wir Ihnen eine Anleitung zur Erstellung eines SPS-Tracker-Dashboards zur Verfügung. Diese Lösung überwacht die SPS-Scores im Laufe der Zeit mithilfe einer YAML-Konfiguration für unterschiedliche Setups (z. B. einschließlich Instanzanforderungen GPUs), speichert Metriken und stellt Dashboards zum Vergleich CloudWatch von Konfigurationen bereit. Definieren Sie Dashboards pro Workload, um die vCPU-, Speicher- und GPU-Anforderungen zu bewerten und so optimale Setups für EKS-Cluster zu gewährleisten, einschließlich der Berücksichtigung der Verwendung anderer AWS-Regionen. Weitere Informationen finden Sie unter So funktioniert der Spot Placement Score.

- Gehen Sie sorgsam mit Spot-Unterbrechungen um und testen Sie. Bei einem Pod mit einer Kündigungsfrist von mehr als zwei Minuten wird der alte Knoten unterbrochen, bevor diese Pods neu geplant werden, was sich auf die Verfügbarkeit der Workloads auswirken kann. Berücksichtigen Sie bei der Entwicklung Ihrer Anwendungen die zweiminütige Spot-Unterbrechungsbenachrichtigung, implementieren Sie Checkpointing in lang laufenden Anwendungen (z. B. Speichern des Fortschritts in persistentem Speicher wie Amazon S3), um nach Unterbrechungen wieder aufgenommen zu werden, verlängern Sie die terminationGracePeriod Sekunden (Standard ist 30 Sekunden) in den Pod-Spezifikationen, um mehr Zeit für ein ordnungsgemäßes Herunterfahren zu haben, und behandeln Sie Unterbrechungen mithilfe von and/or PreStop-Lifecycle-Hook-SIGTERM-Signalen innerhalb Ihrer Anwendung, um Aktivitäten wie Säuberung, Statusspeicherung und Verbindungsabbruch ordnungsgemäß herunterzufahren. Bei Echtzeit-Workloads, bei denen die Skalierungszeit wichtig ist und die Workloads länger als zwei Minuten dauern, bis die Anwendung für den Datenverkehr bereit ist, sollten Sie erwägen, die Startzeiten von Containern und das Laden von ML-Modellen durch Überprüfung the section called “Speicher” und the section called “Leistung” Best Practices zu optimieren. Verwenden Sie zum Testen eines Ersatzknotens den AWS Fault Injection Service (FIS), um Spot-Unterbrechungen zu simulieren.

Zusätzlich zu diesen zentralen Best Practices für Spot sollten Sie diese Faktoren bei der Verwaltung von GPU-Workloads auf Amazon EKS berücksichtigen. Im Gegensatz zu CPU-basierten Workloads reagieren GPU-Workloads besonders empfindlich auf Hardwaredetails wie GPU-Fähigkeiten und verfügbaren GPU-Speicher. GPU-Workloads können durch die Instance-Typen, die sie verwenden können, eingeschränkt werden, wobei im Vergleich zu weniger Optionen verfügbar sind. CPUs Prüfen Sie zunächst, ob Ihr Workload instanzflexibel ist. Wenn Sie nicht wissen, wie

viele Instance-Typen Ihr Workload verwenden kann, testen Sie sie einzeln, um Kompatibilität und Funktionalität sicherzustellen. Identifizieren Sie, wie flexibel Sie sein können, um so viel wie möglich zu diversifizieren, und stellen Sie gleichzeitig sicher, dass die Diversifizierung den Workload am Laufen hält, und verstehen Sie alle Auswirkungen auf die Leistung (z. B. auf den Durchsatz oder die Fertigstellungszeit). Beachten Sie bei der Diversifizierung Ihrer Workloads Folgendes:

- Überprüfen Sie die Kompatibilität von CUDA und Framework. Ihre GPU-Workloads sind möglicherweise für bestimmte Hardware und GPU-Typen (z. B. V100 in p3 gegenüber A100 in p4) optimiert oder für bestimmte CUDA-Versionen für Bibliotheken wie geschrieben. Überprüfen Sie daher unbedingt die TensorFlow Kompatibilität Ihrer Workloads. Diese Kompatibilität ist entscheidend, um Laufzeitfehler, Abstürze, Ausfälle bei der GPU-Beschleunigung (z. B. nicht übereinstimmende CUDA-Versionen mit Frameworks wie PyTorch oder TensorFlow können deren Ausführung verhindern) oder Hardwarefunktionen wie/precision nutzen zu können, zu verhindern. FP16 INT8
- GPU-Speicher. Stellen Sie sicher, dass Sie die Speicheranforderungen Ihrer Modelle evaluieren und die Speichernutzung Ihres Modells während der Laufzeit mithilfe von Tools wie dem DCGM-Exporter profilieren. Geben Sie in bekannten Labels wie karpenter.k8s.aws/ den für den Instance-Typ mindestens erforderlichen GPU-Speicher an. instance-gpu-memory GPU-VRAM variiert je nach Instance-Typ (z. B. hat NVIDIA T4 16 GB, A10G hat 24 GB, V100 hat 16-32 GB), und ML-Modelle (z. B. große Sprachmodelle) können den verfügbaren Speicher überschreiten, was zu out-of-memory (OOM-) Fehlern oder Abstürzen führt. Bei Spot-Instances in EKS kann dies die Diversifizierung einschränken. Sie können beispielsweise keine Typen mit niedrigerem VRAM verwenden, wenn Ihr Modell nicht passt. Dies kann den Zugriff auf Kapazitätspools einschränken und das Risiko von Unterbrechungen erhöhen. Beachten Sie, dass dies bei Inferenz mit einem einzelnen GPU und einem einzelnen Knoten (z. B. mehrere Pods, die auf demselben Knoten geplant sind, um dessen GPU-Ressourcen zu nutzen) die Diversifizierung einschränken kann, da Sie nur Instance-Typen mit ausreichend VRAM in Ihre Spot-Konfiguration aufnehmen können.
- Fließkomma-Präzision und Leistung. Nicht alle Nvidia-GPU-Architekturen haben dieselbe Gleitkomma-Präzision (z. B./). FP16 INT8 Bewerten Sie die Leistung und die Gleitkomma-Präzision der Kerntypen (CUDA/Tensor/RT), die für Ihre Workloads erforderlich sind. Der Betrieb auf einer kostengünstigeren, weniger leistungsfähigen GPU bedeutet nicht, dass sie besser ist. Daher sollten Sie die Leistung anhand der innerhalb eines bestimmten Zeitrahmens abgeschlossenen Arbeit bewerten, um die Auswirkungen der Diversifizierung zu verstehen.

Szenario: Diversifizierung für Inferenz-Workloads in Echtzeit

Für einen Online-Inferenz-Workload in Echtzeit auf Spot-Instances können Sie einen Karpenter so konfigurieren, dass er zwischen kompatiblen NodePool GPU-Instance-Familien und Generationen diversifiziert ist. Dieser Ansatz gewährleistet eine hohe Verfügbarkeit, indem er auf mehrere Spot-Pools zurückgreift und gleichzeitig die Leistung trotz Einschränkungen der GPU-Kapazitäten, des Speichers und der Architektur aufrecht erhält. Er unterstützt die Verwendung von Alternativen, wenn die Instance-Kapazität begrenzt ist, wodurch Unterbrechungen minimiert und die Inferenzlatenz optimiert wird. Dieses Beispiel NodePool besagt, dass Instances der Serien G und P mit mehr als 3 verwendet werden sollten, die über mehr als 20 GB GPU-Speicher verfügen.

Beispiel

```
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
```

Implementieren Sie Checkpointing für lang andauernde Trainingsjobs

Checkpointing ist eine Technik zur Fehlertoleranz, bei der der Status eines Prozesses in regelmäßigen Abständen gespeichert wird, sodass er bei Unterbrechungen vom zuletzt gespeicherten Punkt aus wieder aufgenommen werden kann. Beim maschinellen Lernen wird es häufig mit Training in Verbindung gebracht, bei dem lang andauernde Jobs Modellgewichte und Optimizerstatus speichern können, um das Training nach Ausfällen wie Hardwareproblemen oder Spot-Instance-Unterbrechungen wieder aufzunehmen.

Sie verwenden Checkpoints, um den Status von Modellen für maschinelles Lernen (ML) während des Trainings zu speichern. Checkpoints sind Schnappschüsse des Modells und können mit den Callback-Funktionen von ML-Frameworks konfiguriert werden. Sie können die gespeicherten Checkpoints verwenden, um einen Trainingsjob vom zuletzt gespeicherten Checkpoint aus neu zu starten. Mithilfe von Checkpoints speichern Sie Ihre Modellschnappschüsse während des Trainings aufgrund einer unerwarteten Unterbrechung des Trainingsjobs oder der Trainingsinstanz. Auf diese Weise können Sie das Training des Modells in future von einem Checkpoint aus fortsetzen. Zusätzlich zur Implementierung eines Node-Resiliency-Systems empfehlen wir die Implementierung von Checkpointing, um die Auswirkungen von Unterbrechungen zu minimieren, einschließlich solcher, die durch Hardwareausfälle oder Amazon EC2-Spot-Instance-Unterbrechungen verursacht werden.

Ohne Checkpoints können Unterbrechungen zu verschwendeter Rechenzeit und verlorenen Fortschritten führen, was bei lang andauernden Schulungsaufgaben kostspielig ist. Checkpointing ermöglicht es Jobs, ihren Status regelmäßig zu speichern (z. B. Modellgewichte und Status des Optimierers) und nach einer Unterbrechung ab dem letzten Checkpoint (zuletzt verarbeiteter Stapel) fortzufahren. Um Checkpointing zu implementieren, entwerfen Sie Ihre Anwendung so, dass sie Daten in großen Batches verarbeitet und Zwischenergebnisse in einem persistenten Speicher speichert, z. B. in einem Amazon S3-Bucket über den Mountpoint for Amazon S3 CSI-Treiber, während die Schulung voranschreitet.

Anwendungsfälle

Checkpointing ist in bestimmten Szenarien besonders nützlich, um ein Gleichgewicht zwischen Fehlertoleranz und Leistungsaufwand zu finden. Erwägen Sie die Verwendung von Checkpointing in den folgenden Fällen:

- Die Auftragsdauer überschreitet einige Stunden: Für Trainingsjobs mit langer Laufzeit (z. B. > 1-2 Stunden für kleine Modelle oder days/weeks für große Basismodelle mit Milliarden von Parametern), bei denen der Verlust von Fortschritten aufgrund von Unterbrechungen kostspielig ist. Kürzere Aufträge rechtfertigen möglicherweise nicht den Mehraufwand. I/O
- Für Spot-Instances oder Hardwarefehler: In Umgebungen, die anfällig für Unterbrechungen sind, wie EC2 Spot (2 Minuten im Voraus) oder Hardwarefehler (z. B. GPU-Speicherfehler), ermöglicht Checkpointing eine schnelle Wiederaufnahme, sodass Spot für Kosteneinsparungen bei fehlertoleranten Workloads rentabel ist.
- Verteiltes Training im großen Maßstab: Für Setups mit Beschleunigern (z. B. >100 GPUs), bei denen die mittlere Zeit zwischen Ausfällen linear mit hundreds/thousands der Skalierung abnimmt. Wird für model/data Parallelität verwendet, um den gleichzeitigen Zugriff auf Checkpoints zu handhaben und vollständige Neustarts zu vermeiden.
- Umfangreiche Modelle mit hohem Ressourcenbedarf: Bei LLM-Schulungen im Petabyte-Bereich, bei denen Ausfälle aufgrund der Clustergröße unvermeidlich sind, optimieren mehrstufige Ansätze (schnell lokal alle 5—30 Minuten für Transienten, dauerhaft stündlich für größere Ausfälle) die Wiederherstellungszeit im Verhältnis zur Effizienz.

Verwenden Sie ML-Kapazitätsblöcke für die Kapazitätssicherung von P- und Trainium-Instances

Mit Kapazitätsblöcken für ML können Sie sehr gefragte GPU-Instances, insbesondere P-Instances (z. B. p6-b200, p5, p5e, p5en, p4d, p4de) und Trainium-Instances (z. B. trn1, trn2), reservieren, um entweder fast sofort oder zu einem future Zeitpunkt zu starten, um Ihre kurzfristigen Machine-Learning-Workloads (ML) zu unterstützen. Diese Reservierungen sind ideal, um die Kapazität für rechenintensive Aufgaben wie Modelltraining und Feinabstimmung sicherzustellen. Die Preise für EC2 Capacity Blocks setzen sich aus einer Reservierungsgebühr und einer Betriebssystemgebühr zusammen. Weitere Informationen zur Preisgestaltung finden Sie unter Preise für EC2 Capacity Blocks for ML.

Um AI/ML Workloads auf Amazon EKS GPUs für eine vorhersehbare Kapazitätssicherung zu reservieren, empfehlen wir die Nutzung von ML-Kapazitätsblöcken für kurzfristige oder On-Demand-Kapazitätsreservierungen (ODCRs) zur allgemeinen Kapazitätssicherung.

- Mit ODCRs können Sie EC2-Instance-Kapazität (z. B. GPU-Instances wie g5 oder p5) in einer bestimmten Availability Zone für einen bestimmten Zeitraum reservieren und so die Verfügbarkeit auch bei hoher Nachfrage sicherstellen. ODCRs Sie haben keine langfristige Bindung, aber Sie zahlen den On-Demand-Tarif für die reservierte Kapazität, unabhängig davon, ob sie genutzt oder ungenutzt ist. In EKS ODCRs werden Knotentypen wie Karpenter und verwaltete Knotengruppen unterstützt. Um ODCRs in Karpenter Prioritäten zu setzen, konfigurieren Sie das Feld so, dass es verwendet wird NodeClass . `capacityReservationSelectorTerms` Weitere Informationen finden Sie in der Karpenter Dokumentation. NodePools
- Kapazitätsblöcke sind ein spezieller Reservierungsmechanismus für GPU- (z. B. p5, p4d) oder Trainium-Instances (trn1, trn2), der für kurzfristige ML-Workloads wie Modelltraining, Feinabstimmung oder Experimente konzipiert wurde. Sie reservieren Kapazität für einen bestimmten Zeitraum (in der Regel 24 Stunden bis 182 Tage), der an einem future Datum beginnt, und zahlen nur für die reservierte Zeit. Sie sind im Voraus bezahlt, erfordern eine vorausschauende Planung des Kapazitätsbedarfs und unterstützen keine automatische Skalierung. Sie befinden sich jedoch in EC2 für Netzwerke mit niedriger Latenz. UltraClusters Sie berechnen nur für den reservierten Zeitraum. Weitere Informationen finden Sie unter Suchen und Kaufen von Kapazitätsblöcken oder richten Sie zunächst verwaltete Knotengruppen mit Kapazitätsblöcken ein. Folgen Sie dabei den Anweisungen unter Erstellen einer verwalteten Knotengruppe mit Kapazitätsblöcken für ML.

Reservieren Sie Kapazität über die AWS-Managementkonsole und konfigurieren Sie Ihre Knoten für die Verwendung von ML-Kapazitätsblöcken. Planen Sie Reservierungen auf der Grundlage von Workload-Zeitplänen und testen Sie sie in einem Staging-Cluster. Weitere Informationen finden Sie in der Dokumentation zu Kapazitätsblöcken.

Erwägen Sie On-Demand-, Amazon EC2 Spot- oder On-Demand-Kapazitätsreservierungen (ODCRs) für G Amazon EC2 EC2-Instances

Für G Amazon EC2-Instances sollten Sie die verschiedenen Kaufoptionen wie On-Demand, Amazon EC2 Spot-Instances und On-Demand-Kapazitätsreservierungen in Betracht ziehen. ODCRsermöglichen es Ihnen, EC2-Instance-Kapazität in einer bestimmten Availability Zone für einen bestimmten Zeitraum zu reservieren und so die Verfügbarkeit auch bei hoher Nachfrage sicherzustellen. Im Gegensatz zu ML Capacity Blocks, die nur für P- und Trainium-Instances

verfügbar sind, können ODCRs für ein breiteres Spektrum von Instance-Typen, einschließlich G-Instances, verwendet werden. Dadurch eignen sie sich für Workloads, die unterschiedliche GPU-Fähigkeiten erfordern, wie Inferenz oder Grafik. Bei der Verwendung von Amazon EC2 Spot-Instances ist die Möglichkeit, zwischen verschiedenen Instance-Typen, Größen und Verfügbarkeitszonen zu variieren, entscheidend, um länger vor Ort bleiben zu können.

ODCRs Sie haben kein langfristiges Abonnement, aber Sie zahlen den On-Demand-Tarif für die reservierte Kapazität, unabhängig davon, ob sie genutzt oder ungenutzt ist. ODCRs kann für die sofortige Nutzung erstellt oder für einen future Zeitpunkt geplant werden, was Flexibilität bei der Kapazitätsplanung bietet. In Amazon EKS ODCRs werden sie von Knotentypen wie Karpenter und verwalteten Knotengruppen unterstützt. Um ODCRs in Karpenter Prioritäten zu setzen, konfigurieren Sie das Feld so, dass es verwendet wird NodeClass . `capacityReservationSelectorTerms` Weitere Informationen finden Sie in der Karpenter Dokumentation. NodePools Weitere Informationen zum Erstellen ODCRs, einschließlich CLI-Befehlen, finden Sie unter On-Demand-Kapazitätsreservierung — Erste Schritte.

Ziehen Sie andere beschleunigte Instance-Typen und -Größen in Betracht

Die Auswahl der geeigneten beschleunigten Instance und Größe ist entscheidend für die Optimierung von Leistung und Kosten Ihrer ML-Workloads auf Amazon EKS. Beispielsweise verfügen verschiedene GPU-Instance-Familien über unterschiedliche Leistungen und Funktionen wie GPU-Speicher. Um Ihnen bei der Auswahl der Option mit dem besten Preis-Leistungs-Verhältnis zu helfen, sehen Sie sich die verfügbaren GPU-Instances auf der Seite EC2-Instance-Typen unter Accelerated Computing an. Bewerten Sie mehrere Instance-Typen und -Größen, um die beste Lösung für Ihre spezifischen Workload-Anforderungen zu finden. Berücksichtigen Sie Faktoren wie Anzahl GPUs, Arbeitsspeicher und Netzwerkleistung. Durch die sorgfältige Auswahl des richtigen GPU-Instance-Typs und der richtigen Größe können Sie eine bessere Ressourcennutzung und Kosteneffizienz in Ihren EKS-Clustern erreichen.

Wenn Sie eine GPU-Instanz in einem EKS-Knoten verwenden, befindet sich der `nvidia-device-plugin-daemonset` Pod standardmäßig im `kube-system` Namespace. Um einen schnellen Eindruck davon zu bekommen, ob Sie die GPU (s) in Ihrer Instanz vollständig nutzen, können Sie nvidia-smi wie hier gezeigt verwenden:

```
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
```

- Wenn der `utilization.memory` Wert nahe bei 100% liegt, sind Ihr (e) Code (s) wahrscheinlich speichergebunden. Dies bedeutet, dass die GPU (Speicher) voll ausgelastet ist, könnte aber darauf hindeuten, dass weitere Leistungsoptimierungen untersucht werden sollten.
- Wenn der nahe bei 100% `utilization.gpu` liegt, bedeutet dies nicht unbedingt, dass die GPU voll ausgelastet ist. Eine bessere Metrik, die man sich ansehen sollte, ist das Verhältnis von `power.draw` zu`power.limit`. Wenn dieses Verhältnis 100% oder mehr beträgt, nutzen Ihre Codes die Rechenkapazität der GPU vollständig aus.
- Die `-l 5` Flagge besagt, dass die Metriken alle 5 Sekunden ausgegeben werden sollen. Im Fall eines einzelnen GPU-Instanztyps wird das Indexabfrageflag nicht benötigt.

Weitere Informationen finden Sie unter GPU-Instances in der AWS-Dokumentation.

Optimieren Sie die GPU-Ressourcenzuweisung mit Time-Slicing, MIG und fraktionierter GPU-Zuweisung

Statische Ressourcenbeschränkungen in Kubernetes (z. B. CPU-, Arbeitsspeicher-, GPU-Anzahl) können zu einer Über- oder Unterauslastung führen, insbesondere bei dynamischen Workloads wie Inferenz. AI/ML Die Auswahl der richtigen GPU ist wichtig. Bei Workloads mit geringem Volumen oder hoher Auslastung ermöglicht Time-Slicing, dass sich mehrere Workloads eine einzelne GPU teilen, indem sie ihre Rechenressourcen gemeinsam nutzen, was potenziell die Effizienz verbessert und Verschwendung reduziert. Die gemeinsame Nutzung der GPU kann über verschiedene Optionen erreicht werden:

- Nutzen Sie die Knotenauswahl und die Knotenaffinität, um die Planung zu beeinflussen: Stellen Sie sicher, dass die bereitgestellten Knoten und Pods auf den GPUs für die Arbeitslast geeigneten Knoten und Pods eingeplant werden (z. B. `karpenter.k8s.aws/instance-gpu-name: "a100"`
- Time-Slicing: Plant Workloads so, dass sie die Rechenressourcen einer GPU im Laufe der Zeit gemeinsam nutzen, was eine gleichzeitige Ausführung ohne physische Partitionierung ermöglicht. Dies ist ideal für Workloads mit variablem Rechenbedarf, aber möglicherweise fehlt es an Speicherisolierung.
- Multi-Instance-GPU (MIG): MIG ermöglicht die Partitionierung einer einzelnen NVIDIA-GPU in mehrere isolierte Instanzen und wird von NVIDIA Ampere (z. B. A100-GPU), NVIDIA Hopper (z. B. H100-GPU) und NVIDIA Blackwell (z. B. Blackwell) unterstützt. GPUs GPUs Jede MIG-

Instanz erhält dedizierte Rechen- und Speicherressourcen und ermöglicht so die gemeinsame Nutzung von Ressourcen in Umgebungen mit mehreren Mandanten oder Workloads, für die Ressourcengarantien erforderlich sind. Dadurch können Sie die GPU-Ressourcennutzung optimieren, einschließlich Szenarien wie der Bereitstellung mehrerer Modelle mit unterschiedlichen Batchgrößen durch Timeslicing.

- Teilweise GPU-Zuweisung: Nutzt softwarebasiertes Scheduling, um Teile der Rechenleistung oder des Speichers einer GPU Workloads zuzuweisen, was Flexibilität für dynamische Workloads bietet. Der NVIDIA KAI Scheduler, Teil der Run:AI-Plattform, ermöglicht dies, indem er es Pods ermöglicht, fraktionierte GPU-Ressourcen anzufordern.

Um diese Funktionen in EKS zu aktivieren, können Sie das NVIDIA Device Plugin einsetzen, das GPUs als planbare Ressourcen verfügbar ist und Time-Slicing und MIG unterstützt. Weitere Informationen finden Sie unter Time-Slicing GPUs in Kubernetes und GPU-Sharing auf Amazon EKS mit NVIDIA-Time-Slicing und beschleunigten EC2-Instances.

Beispiel

Um beispielsweise Time-Slicing mit dem NVIDIA Device Plugin zu aktivieren:

```
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
```

Beispiel

Um beispielsweise den KAI Scheduler für die fraktionierte GPU-Zuweisung zu verwenden, stellen Sie ihn zusammen mit dem NVIDIA-GPU-Operator bereit und geben Sie fraktionierte GPU-Ressourcen in der Pod-Spezifikation an:

```
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
```

Knotenausfallsicherheit und Verwaltung von Schulungsaufträgen

Implementieren Sie Node Health Checks mit automatisierter Wiederherstellung

Bei verteilten Trainingsaufgaben auf Amazon EKS, die eine häufige Kommunikation zwischen den Knoten erfordern, wie z. B. das Training mehrerer GPU-Modelle über mehrere Knoten hinweg, können Hardwareprobleme wie GPU- oder EFA-Ausfälle zu Unterbrechungen der Trainingsaufgaben führen. Diese Störungen können zum Verlust von Trainingsfortschritten und zu höheren Kosten führen, insbesondere bei lang andauernden AI/ML Workloads, die auf stabiler Hardware basieren.

Um die Widerstandsfähigkeit gegen Hardwarefehler zu erhöhen, wie z. B. GPU-Ausfälle in EKS-Clustern, auf denen GPU-Workloads ausgeführt werden, empfehlen wir, entweder den EKS Node Monitoring Agent mit Auto Repair oder Amazon zu nutzen. SageMaker HyperPod Der EKS Node Monitoring Agent with Auto Repair bietet Funktionen wie die Überwachung des Knotenzustands und die automatische Reparatur mithilfe von Kubernetes-Standardmechanismen, SageMaker HyperPod bietet gezielte Resilienz und zusätzliche Funktionen, die speziell für groß angelegte ML-Trainings entwickelt wurden, wie z. B. eingehende Gesundheitschecks und automatische Wiederaufnahme von Jobs.

- Der EKS Node Monitoring Agent mit Node Auto Repair überwacht kontinuierlich den Zustand der Knoten NodeConditions, liest Protokolle und wendet Standardbedingungen wie `Ready` und spezifische Bedingungen für beschleunigte Hardware an, um Probleme wie GPU- oder

Netzwerkausfälle zu identifizieren. Wenn ein Knoten als fehlerhaft eingestuft wird, sperrt Node Auto Repair ihn ab und ersetzt ihn durch einen neuen Knoten. Die Neuplanung von Pods und der Neustart von Jobs basieren auf den Standardmechanismen von Kubernetes und der Neustartrichtlinie des Jobs.

- Der Agent für SageMaker HyperPodeingehende Integritätsprüfungen und die Zustandsüberwachung überwacht kontinuierlich den Integritätsstatus von GPU- und Trainium-basierten Instances. Er ist auf AI/ML Workloads zugeschnitten und verwendet Labels (z. B.) zur Verwaltung des Knotenzustands. node-health-status Wenn ein Knoten als fehlerhaft eingestuft wird, wird der automatische Austausch der fehlerhaften Hardware HyperPod ausgelöst, z. B. GPUs Es erkennt standardmäßig netzwerkbedingte Ausfälle für EFA durch seine grundlegenden Integritätsprüfungen und unterstützt die automatische Wiederaufnahme unterbrochener Trainingsjobs, sodass Jobs ab dem letzten Checkpoint fortgesetzt werden können, wodurch Unterbrechungen bei umfangreichen ML-Aufgaben minimiert werden.

Stellen Sie sowohl für EKS Node Monitoring Agent mit Auto Repair als auch für SageMaker HyperPod Cluster, die EFA verwenden, zur Überwachung EFA-spezifischer Metriken wie Remote Direct Memory Access (RDMA) -Fehler und Paketabbrüche sicher, dass der AWS EFA-Treiber installiert ist. Darüber hinaus empfehlen wir, das CloudWatch Observability Add-on einzusetzen oder Tools wie DCGM Exporter mit Prometheus und Grafana zu verwenden, um EFA, GPU und damit spezifische Metriken im Zusammenhang mit seinen Funktionen zu überwachen. SageMaker HyperPod

Deaktivieren Sie Karpenter Consolidation für unterbrechungssensitive Workloads

Für Workloads, die empfindlich auf Unterbrechungen reagieren, wie etwa bei der Verarbeitung, umfangreichen AI/ML Prognoseaufgaben oder Schulungen, empfehlen wir, die Konsolidierungsrichtlinien von Karpenter zu optimieren, um Unterbrechungen bei der Auftragsausführung zu vermeiden. Die Konsolidierungsfunktion von Karpenter optimiert automatisch die Cluster-Kosten, indem sie nicht ausgelastete Knoten beendet oder sie durch kostengünstigere Alternativen ersetzt. Selbst wenn ein Workload eine GPU vollständig ausnutzt, kann Karpenter Knoten konsolidieren, wenn es einen kostengünstigeren Instance-Typ mit passender Größe identifiziert, der die Anforderungen des Pods erfüllt, was zu Jobunterbrechungen führt.

Die `WhenEmptyOrUnderutilized` Konsolidierungsrichtlinie kann Knoten vorzeitig beenden, was zu längeren Ausführungszeiten führt. Beispielsweise können Unterbrechungen die Wiederaufnahme von Jobs verzögern, da Pods neu geplant und Daten neu geladen werden, was bei Batch-Inferenzjobs mit langer Laufzeit kostspielig sein kann. Um dies zu vermeiden, können Sie den Wert

`consolidationPolicy` auf festlegen `WhenEmpty` und eine `consolidateAfter` Dauer, z. B. 1 Stunde, konfigurieren, damit Knoten auch bei Arbeitslastspitzen erhalten bleiben. Beispiel:

```
disruption:
  consolidationPolicy: WhenEmpty
  consolidateAfter: 60m
```

Dieser Ansatz verbessert die Latenz beim Pod-Start bei hohen Batch-Inferenz-Workloads und anderen unterbrechungssensitiven Aufgaben, wie z. B. Online-Inferenzdatenverarbeitung in Echtzeit oder Modelltraining, bei denen die Kosten einer Unterbrechung die Einsparungen bei den Rechenkosten überwiegen. Karpenter Disruption Budgets ist eine weitere Funktion zur Verwaltung von Störungen bei KarpenterNodePool . Mit Budgets können Sie sicherstellen, dass zu einem bestimmten Zeitpunkt nicht mehr als eine bestimmte Anzahl von Knotenknoten in den ausgewählten NodePool Knoten unterbrochen wird. Sie können auch Unterbrechungsbudgets verwenden, um zu verhindern, dass alle Knoten zu einem bestimmten Zeitpunkt (z. B. zu Spitzenzeiten) unterbrochen werden. Weitere Informationen finden Sie in der Dokumentation zu Karpenter Consolidation.

Verwenden Sie ttlSecondsAfter Finished, um Kubernetes-Jobs automatisch zu bereinigen

Wir empfehlen, `ttlSecondsAfterFinished` für Kubernetes-Jobs in Amazon EKS so einzustellen, dass abgeschlossene Auftragsobjekte automatisch gelöscht werden. Verbleibende Auftragsobjekte verbrauchen Clusterressourcen wie API-Serverspeicher und erschweren die Überwachung durch überladene Dashboards (z. B. Grafana, Amazon). CloudWatch Wenn Sie beispielsweise eine TTL von 1 Stunde festlegen, wird sichergestellt, dass Jobs kurz nach Abschluss entfernt werden, sodass Ihr Cluster tidy bleibt. Weitere Informationen finden Sie unter Automatische Bereinigung abgeschlossener Jobs.

Job-Präemption mit niedriger Priorität für Aufträge/Workloads mit höherer Priorität konfigurieren

Für AI/ML Workloads mit gemischter Priorität auf Amazon EKS können Sie das Preemption von Jobs mit niedriger Priorität konfigurieren, um sicherzustellen, dass Aufgaben mit höherer Priorität (z. B. Echtzeit-Inferenz) umgehend Ressourcen erhalten. Ohne Präemption können Workloads mit niedriger Priorität wie Batch-Prozesse (z. B. Batch-Inferenz, Datenverarbeitung), Nicht-Batch-Dienste (z. B. Hintergrundaufgaben, Cron-Jobs) oder CPU-/speicherintensive Jobs (z. B. Webdienste) kritische Pods verzögern, indem sie Knoten belegen. Mit Preemption kann Kubernetes Pods mit niedriger Priorität entfernen, wenn Pods mit hoher Priorität Ressourcen benötigen, wodurch eine

effiziente Ressourcenzuweisung auf Knoten mit, oder Speicher gewährleistet wird. GPUs CPUs Wir empfehlen die Verwendung von `PriorityClass` Kubernetes, um Prioritäten zuzuweisen und das Räumungsverhalten zu kontrollieren. `PodDisruptionBudget`

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: low-priority
value: 100
---
spec:
  priorityClassName: low-priority
```

Weitere Informationen finden Sie in der Dokumentation zu Kubernetes Priority and Preemption.

Skalierung und Leistung von Anwendungen

Passen Sie die Rechenkapazität für ML-Workloads mit Karpenter oder Static Nodes an

Um kosteneffiziente und reaktionsschnelle Rechenkapazitäten für maschinelles Lernen (ML) - Workflows auf Amazon EKS sicherzustellen, empfehlen wir, Ihre Node-Bereitstellungsstrategie an die Merkmale und Kostenverpflichtungen Ihres Workloads anzupassen. Im Folgenden sind zwei Ansätze aufgeführt, die Sie in Betracht ziehen sollten: just-in-time Skalierung mit Karpenter und statische Knotengruppen für reservierte Kapazität.

- Just-in-time Datenebenen-Skalierer wie Karpenter: Für dynamische ML-Workflows mit variablen Rechenanforderungen (z. B. GPU-basierte Inferenz gefolgt von CPU-basierter Darstellung) empfehlen wir die Verwendung von Datenebenen-Skalierern wie Karpenter. just-in-time
- Verwenden Sie statische Knotengruppen für vorhersehbare Workloads: Bei vorhersagbaren, stationären ML-Workloads oder bei der Verwendung von Reserved Instances können von EKS verwaltete Knotengruppen dazu beitragen, dass die reservierte Kapazität vollständig bereitgestellt und genutzt wird, wodurch die Einsparungen maximiert werden. Dieser Ansatz ist ideal für bestimmte Instance-Typen, die über oder festgeschrieben werden. RIs ODCRs

Beispiel

Dies ist ein Beispiel für einen vielseitigen Karpenter NodePool, der das Starten von g Amazon EC2 EC2-Instances ermöglicht, bei denen die Instance-Generierung mehr als drei beträgt.

```
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
```

Beispiel

Beispiel für die Verwendung statischer Knotengruppen für einen Trainings-Workload:

```
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
```

Verwenden Sie Taints und Toleranzen, um zu verhindern, dass nicht beschleunigte Workloads auf beschleunigten Instances geplant werden

Die Planung nicht beschleunigter Workloads auf GPU-Ressourcen ist nicht recheneffizient. Wir empfehlen, Taints und Toleration zu verwenden, um sicherzustellen, dass Pods für nicht beschleunigte Workloads nicht auf ungeeigneten Knoten geplant werden. Weitere Informationen finden Sie in der Kubernetes-Dokumentation.

Skalieren Sie basierend auf der Modellleistung

Für Inferenz-Workloads empfehlen wir die Verwendung von Kubernetes Event-Driven Autoscaling (KEDA) zur Skalierung auf der Grundlage von Modellleistungskennzahlen wie Inferenzanfragen oder Token-Durchsatz mit entsprechenden Abklingzeiten. Statische Skalierungsrichtlinien können zu viel oder zu wenig Ressourcen bereitstellen, was sich auf Kosten und Latenz auswirkt. Weitere Informationen finden Sie in der KEDA-Dokumentation.

Dynamische Ressourcenzuweisung für erweitertes GPU-Management

Die dynamische Ressourcenzuweisung (DRA) stellt einen grundlegenden Fortschritt im Kubernetes-GPU-Ressourcenmanagement dar. DRA geht über die Beschränkungen herkömmlicher Geräte-Plug-ins hinaus und ermöglicht ausgeklügelte GPU-Sharing, Topologieerkennung und knotenübergreifende Ressourcenkoordination. DRA ist in Amazon EKS Version 1.33 verfügbar und adressiert kritische Herausforderungen bei AI/ML Workloads, indem es Folgendes bietet:

- Feinkörnige GPU-Zuweisung
- Fortschrittliche Sharing-Mechanismen wie Multi-Process Service (MPS) und Multi-Instance-GPU (MIG)

- Support für Hardwarearchitekturen der nächsten Generation, einschließlich NVIDIA GB200 UltraServers

Herkömmliche GPU-Zuweisung behandelt Ressourcen GPUs als undurchsichtige Integer-Ressourcen, was zu einer erheblichen Unterauslastung führt (häufig 30-40% in Produktionsclustern). Dies liegt daran, dass Workloads exklusiven Zugriff auf Gesamtressourcen erhalten, GPUs auch wenn sie nur Bruchteile der Ressourcen benötigen. DRA transformiert dieses Modell, indem es eine strukturierte, deklarative Zuweisung einführt, die dem Kubernetes-Scheduler einen vollständigen Überblick über Hardwaremerkmale und Workload-Anforderungen bietet. Dies ermöglicht intelligente Platzierungsentscheidungen und eine effiziente gemeinsame Nutzung von Ressourcen.

Vorteile der Verwendung von DRA anstelle des NVIDIA-Geräte-Plugins

Das NVIDIA-Geräte-Plugin (ab Version`0.12.0`) unterstützt GPU-Sharing-Mechanismen wie Time-Slicing, MPS und MIG. Es gibt jedoch architektonische Einschränkungen, die DRA behebt.

Einschränkungen des NVIDIA-Geräte-Plugins

- Statische Konfiguration: Konfigurationen zur gemeinsamen Nutzung von GPUs (Time-Slicing-Replikate und MPS-Einstellungen) erfordern eine clusterweite Vorkonfiguration. `ConfigMaps` Dies macht es schwierig, unterschiedliche Sharing-Strategien für unterschiedliche Workloads bereitzustellen.
- Eingeschränkte detaillierte Auswahl: Das Geräte-Plug-In macht zwar GPU-Merkmale anhand von Knotenbezeichnungen verfügbar, Workloads können jedoch nicht dynamisch bestimmte GPU-Konfigurationen (Speichergröße und Rechenkapazität) als Teil der Planungsentscheidung anfordern.
- Keine knotenübergreifende Ressourcenkoordination: Es ist nicht möglich, verteilte GPU-Ressourcen über mehrere Knoten zu verwalten oder komplexe Topologieanforderungen wie NVLink Domänen für Systeme wie NVIDIA zu formulieren. GB200
- Scheduler-Einschränkungen: Der Kubernetes-Scheduler behandelt GPU-Ressourcen als undurchsichtige Ganzzahlen, was seine Fähigkeit einschränkt, topologiebewusste Entscheidungen zu treffen oder komplexe Ressourcenabhängigkeiten zu handhaben.
- Komplexität der Konfiguration: Die Einrichtung verschiedener Strategien zur gemeinsamen Nutzung erfordert mehrere `ConfigMaps` und sorgfältige Knotenkennzeichnungen, was zu betrieblicher Komplexität führt.

Lösungen mit DRA

- Dynamische Ressourcenauswahl: Mit DRA können Workloads detaillierte Anforderungen (GPU-Speicher, Treiberversionen und spezifische Attribute) zum Zeitpunkt der Anfrage angeben. `resourceclaims` Dies ermöglicht einen flexibleren Ressourcenabgleich.
- Topologiebewusstsein: Durch strukturierte Parameter und Geräteauswahlen bewältigt DRA komplexe Anforderungen wie knotenübergreifende GPU-Kommunikation und speicherkohärente Verbindungen.
- Knotenübergreifendes Ressourcenmanagement: `computeDomains` ermöglicht die Koordination verteilter GPU-Ressourcen auf mehrere Knoten, was für Systeme wie IMEX-Kanäle von entscheidender Bedeutung ist. GB200
- Workload-spezifische Konfiguration: Jede Konfiguration `ResourceClaim` spezifiziert unterschiedliche Strategien und Konfigurationen für die gemeinsame Nutzung und ermöglicht so eine feinkörnige Steuerung pro Workload anstelle von clusterweiten Einstellungen.
- Verbesserte Scheduler-Integration: DRA stellt dem Scheduler detaillierte Geräteinformationen zur Verfügung und ermöglicht intelligentere Platzierungsentscheidungen auf der Grundlage der Hardwaretopologie und der Ressourceneigenschaften.

Wichtig: DRA ersetzt das NVIDIA-Geräte-Plugin nicht vollständig. Der NVIDIA DRA-Treiber arbeitet zusammen mit dem Geräte-Plug-In, um erweiterte Funktionen bereitzustellen. Das Geräte-Plugin übernimmt weiterhin die grundlegende Erkennung und Verwaltung von GPUs, während DRA erweiterte Zuweisungs- und Planungsfunktionen hinzufügt.

Von DRA unterstützte Instanzen und ihre Funktionen

Die DRA-Unterstützung variiert je nach Amazon EC2 EC2-Instance-Familie und GPU-Architektur, wie in der folgenden Tabelle dargestellt.

Instance-Familie	GPU-Typ	Zeiteinteilung	MIG-Unterstützung	MPS-Unterstützung	IMEX-Unterstützung	Anwendungsfälle
G5	NVIDIA A10G	Ja	Nein	Ja	Nein	Inferenz- und Grafik-Workloads
G6	NVIDIA L4	Ja	Nein	Ja	Nein	KI-Inferenz und Videoverarbeitung

Instance-Familie	GPU-Typ	Zeiteinteilung	MIG-Unterstützung	MPS-Unterstützung	IMEX-Unterstützung	Anwendungsfälle
G6e	NVIDIA L40S	Ja	Nein	Ja	Nein	Training, Inferenz und Grafik
P4d/P4de	NVIDIA A100	Ja	Ja	Ja	Nein	Umfangreiches Training und HPC
P5	NVIDIA H100	Ja	Ja	Ja	Nein	Modelltraining für die Stiftung
P6	NVIDIA B200	Ja	Ja	Ja	Nein	Modelle mit Milliarden oder Billionen Parametern, verteiltes Training und Inferenz
P6e	NVIDIA GB200	Ja	Ja	Ja	Ja	Modelle mit Milliarden oder Billionen Parametern, verteiltes Training und Inferenz

Im Folgenden werden die einzelnen Funktionen in der Tabelle beschrieben:

- Time-Slicing: Ermöglicht es mehreren Workloads, GPU-Rechenressourcen im Laufe der Zeit gemeinsam zu nutzen.
- Multi-Instance-GPU (MIG): Partitionierung auf Hardwareebene, die isolierte GPU-Instanzen erstellt.
- Multi-Process-Service (MPS): Ermöglicht die gleichzeitige Ausführung mehrerer CUDA-Prozesse auf einer einzigen GPU.
- Internode Memory Exchange (IMEX): Speicherkohärente Kommunikation zwischen Knoten für. GB200 UltraServers

Weitere Ressourcen

Weitere Informationen zu Kubernetes DRA- und NVIDIA DRA-Treibern finden Sie in den folgenden Ressourcen unter: GitHub

- Kubernetes dynamic-resource-allocation
- Vorschlag zur Erweiterung von Kubernetes für DRA
- NVIDIA DRA-Treiber für GPUs
- NVIDIA DRA-Beispiele und Schnellstart

Richten Sie eine dynamische Ressourcenzuweisung für erweitertes GPU-Management ein

Das folgende Thema zeigt Ihnen, wie Sie die dynamische Ressourcenzuweisung (DRA) für erweitertes GPU-Management einrichten.

Voraussetzungen

Stellen Sie vor der Implementierung von DRA auf Amazon EKS sicher, dass Ihre Umgebung die folgenden Anforderungen erfüllt.

Cluster-Konfiguration

- Amazon EKS-Cluster mit laufender Version `1.33` oder höher
- Von Amazon EKS verwaltete Knotengruppen (DRA wird derzeit nur von verwalteten Knotengruppen mit AL2023 und für Bottlerocket NVIDIA optimiert AMIs, nicht von Karpenter unterstützt)
- GPU-fähige NVIDIA-Worker-Knoten mit entsprechenden Instance-Typen

Erforderliche Komponenten

- NVIDIA-Geräte-Plug-in-Version `0.17.1` oder höher
- NVIDIA DRA-Treiberversion `25.3.0` oder höher

Schritt 1: Erstellen Sie mithilfe von eksctl einen Cluster mit einer DRA-fähigen Knotengruppe

1. Erstellen Sie eine Cluster-Konfigurationsdatei mit dem Namen: `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. Erstellen Sie den Cluster:

```
eksctl create cluster -f dra-eks-cluster.yaml
```

Schritt 2: Stellen Sie das NVIDIA-Geräte-Plugin bereit

Stellen Sie das NVIDIA-Geräte-Plugin bereit, um die grundlegende GPU-Erkennung zu ermöglichen:

1. Fügen Sie das Helm-Repository für das NVIDIA-Geräte-Plugin hinzu:

```
helm repo add nvidia https://nvidia.github.io/k8s-device-plugin
helm repo update
```

2. Erstellen Sie benutzerdefinierte Werte für das Geräte-Plugin:

```
cat <<EOF > nvidia-device-plugin-values.yaml
gfd:
  enabled: true
nfd:
  enabled: true
tolerations:
```

```
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
EOF
```

3. Installieren Sie das NVIDIA-Geräte-Plug-In:

```
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
```

Schritt 3: Stellen Sie das Helm-Diagramm für den NVIDIA-DRA-Treiber bereit

1. Erstellen Sie eine `dra-driver-values.yaml` Wertedatei für den DRA-Treiber:

```
---
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. Fügen Sie das NVIDIA NGC Helm-Repository hinzu:

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
```

3. Installieren Sie den NVIDIA DRA-Treiber:

```
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
```

Schritt 4: Überprüfen Sie die DRA-Installation

1. Stellen Sie sicher, dass die DRA API-Ressourcen verfügbar sind:

```
kubectl api-resources | grep resource.k8s.io/v1beta1
```

   Das Folgende ist die erwartete Ausgabe:

```
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. Überprüfen Sie die verfügbaren Geräteklassen:

```
kubectl get deviceclasses
```

   Im Folgenden finden Sie ein Beispiel für die erwartete Ausgabe:

```
NAME                                        AGE
compute-domain-daemon.nvidia.com            4h39m
```

```
compute-domain-default-channel.nvidia.com   4h39m
gpu.nvidia.com                              4h39m
mig.nvidia.com                              4h39m
```

Wenn eine neu erstellte G6-GPU-Instance Ihrem Amazon EKS-Cluster mit aktiviertem DRA beitritt, werden die folgenden Aktionen ausgeführt:

- Der NVIDIA DRA-Treiber erkennt automatisch die A10G-GPU und erstellt zwei `resourceslices` auf diesem Knoten.
- Der `gpu.nvidia.com` Slice registriert das physische A10G-GPU-Gerät mit seinen Spezifikationen (Speicher, Rechenkapazität und mehr).
- Da A10G die MIG-Partitionierung nicht unterstützt, erstellt der `compute-domain.nvidia.com` Slice eine einzelne Rechendomäne, die den gesamten Rechenkontext der GPU darstellt.
- Diese `resourceslices` werden dann auf dem Kubernetes-API-Server veröffentlicht, sodass die GPU-Ressourcen für die Planung zur Verfügung stehen. `resourceclaims`

  Der DRA-Scheduler kann diese GPU nun intelligent Pods zuweisen, über die GPU-Ressourcen angefordert werden, was im Vergleich zu herkömmlichen `resourceclaimtemplates` Geräte-Plug-in-Ansätzen eine flexiblere Ressourcenverwaltung ermöglicht. Dies geschieht automatisch ohne manuelles Eingreifen. Der Knoten wird einfach für GPU-Workloads verfügbar, sobald der DRA-Treiber die Ressourcenerkennung und Registrierung abgeschlossen hat.

  Wenn Sie den folgenden Befehl ausführen:

  ```
  kubectl get resourceslices
  ```

  Im Folgenden finden Sie ein Beispiel für die erwartete Ausgabe:

  ```
  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
  ```

Fahren Sie fort mit the section called “Planen Sie einen einfachen GPU-Workload mithilfe der dynamischen Ressourcenzuweisung”.

Planen Sie einen einfachen GPU-Workload mithilfe der dynamischen Ressourcenzuweisung

Gehen Sie wie folgt vor, um einen einfachen GPU-Workload mithilfe der dynamischen Ressourcenzuweisung (DRA) zu planen. Bevor Sie fortfahren, stellen Sie sicher, dass Sie die folgenden Anweisungen befolgt habenthe section called “Richten Sie eine dynamische Ressourcenzuweisung für erweitertes GPU-Management ein”.

1. Erstellen Sie eine Grundlage `ResourceClaimTemplate` für die GPU-Zuweisung mit einer Datei mit dem Namen`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. Wenden Sie die Vorlage an:

```
kubectl apply -f basic-gpu-claim-template.yaml
```

3. Überprüfen Sie den Status:

```
kubectl get resourceclaimtemplates -n gpu-test1
```

Das Folgende ist eine Beispielausgabe:

```
NAME         AGE
single-gpu   9m16s
```

4. Erstellen Sie einen Pod, der den verwendet, `ResourceClaimTemplate` mit einer Datei namens`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. Wenden Sie den Pod an und überwachen Sie ihn:

```
kubectl apply -f basic-gpu-pod.yaml
```

6. Überprüfen Sie den Pod-Status:

```
kubectl get pod -n gpu-test1
```

Das Folgende ist ein Beispiel für eine erwartete Ausgabe:

```
NAME      READY   STATUS    RESTARTS   AGE
gpu-pod   1/1     Running   0          13m
```

7. Überprüfen Sie den `ResourceClaim` Status:

```
kubectl get resourceclaims -n gpu-test1
```

Das Folgende ist ein Beispiel für eine erwartete Ausgabe:

```
NAME                 STATE                AGE
gpu-pod-gpu0-l76cg   allocated,reserved   9m6s
```

8. Sehen Sie sich die Pod-Logs an, um GPU-Informationen zu sehen:

```
kubectl logs gpu-pod -n gpu-test1
```

Im Folgenden finden Sie ein Beispiel für eine erwartete Ausgabe:

```
GPU 0: NVIDIA L4 (UUID: GPU-da7c24d7-c7e3-ed3b-418c-bcecc32af7c5)
```

Weitere fortgeschrittene Techniken the section called “Techniken zur GPU-Optimierung mit dynamischer Ressourcenzuweisung” zur GPU-Optimierung mithilfe von DRA finden Sie weiter.

Techniken zur GPU-Optimierung mit dynamischer Ressourcenzuweisung

Moderne GPU-Workloads erfordern ein ausgeklügeltes Ressourcenmanagement, um eine optimale Nutzung und Kosteneffizienz zu erreichen. DRA ermöglicht mehrere fortschrittliche Optimierungstechniken, die auf unterschiedliche Anwendungsfälle und Hardwarefunktionen zugeschnitten sind:

- Time-Slicing ermöglicht es mehreren Workloads, GPU-Rechenressourcen im Laufe der Zeit gemeinsam zu nutzen, was es ideal für Inferenz-Workloads mit sporadischer GPU-Nutzung macht. Ein Beispiel finden Sie unter the section called “Optimieren Sie GPU-Workloads mit Time-Slicing”.

- Der Multi-Process Service (MPS) ermöglicht die gleichzeitige Ausführung mehrerer CUDA-Prozesse auf einer einzigen GPU und bietet dabei eine bessere Isolierung als Time-Slicing. Ein Beispiel finden Sie unter the section called “Optimieren Sie GPU-Workloads mit MPS”.
- Die Multi-Instance-GPU (MIG) ermöglicht Partitionierung auf Hardwareebene und erstellt isolierte GPU-Instanzen mit dedizierten Rechen- und Speicherressourcen. Ein Beispiel finden Sie unter the section called “Optimieren Sie GPU-Workloads mit Multi-Instance-GPU”.
- Internode Memory Exchange (IMEX) ermöglicht eine speicherkohärente Kommunikation zwischen Knoten für verteilte Schulungen auf NVIDIA-Systemen. GB200 Ein Beispiel finden Sie unter the section called “Optimieren Sie GPU-Workloads mit IMEX mithilfe von GB200 P6e-Instanzen”.

Diese Techniken können die Ressourcennutzung erheblich verbessern. Organizations berichten, dass die GPU-Auslastung von 30 bis 40% bei herkömmlicher Zuweisung auf 80 bis 90% bei optimierten Sharing-Strategien steigt. Die Wahl der Technik hängt von den Eigenschaften der Arbeitslast, den Isolationsanforderungen und den Hardwarefunktionen ab.

Optimieren Sie GPU-Workloads mit Time-Slicing

Time-Slicing ermöglicht es mehreren Workloads, GPU-Rechenressourcen gemeinsam zu nutzen, indem sie so geplant werden, dass sie sequentiell auf derselben physischen GPU ausgeführt werden. Es ist ideal für Inferenz-Workloads mit sporadischer GPU-Nutzung.

Führen Sie die folgenden Schritte aus.

1. Definieren Sie a `ResourceClaimTemplate` für Timeslicing mit einer Datei mit dem Namen: `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. Definieren Sie einen Pod mithilfe von Timeslicing mit einer Datei namens: `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. Wenden Sie die Vorlage und den Pod an:

```
kubectl apply -f timeslicing-claim-template.yaml
kubectl apply -f timeslicing-pod.yaml
```

4. Ressourcenansprüche überwachen:

```
kubectl get resourceclaims -n timeslicing-gpu -w
```

Das Folgende ist eine Beispielausgabe:

```
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
```

Erster Pod (`inference-pod-1`)

- Bundesland: `allocated,reserved`
- Das heißt: DRA hat eine verfügbare GPU gefunden und sie für diesen Pod reserviert
- Pod-Status: Startet sofort

Zweiter Pod (`training-pod-2`)

- Bundesland: `pending`
- Das heißt: Ich warte darauf, dass DRA Time-Slicing auf derselben GPU konfiguriert
- Pod-Status: Wartet darauf, geplant zu werden
- Der Status wird von bis `pending` `allocated,reserved` zu wechseln `running`

Optimieren Sie GPU-Workloads mit MPS

Multi-Process Service (MPS) ermöglicht die gleichzeitige Ausführung mehrerer CUDA-Kontexte auf einer einzigen GPU mit besserer Isolierung als Time-Slicing.

Führen Sie die folgenden Schritte aus.

1. Definieren Sie a `ResourceClaimTemplate` für MPS mit einer Datei namens`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. Definieren Sie einen Pod mithilfe von MPS mit einer Datei namens`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. Wenden Sie die Vorlage an und erstellen Sie mehrere MPS-Pods:

```
kubectl apply -f mps-claim-template.yaml
kubectl apply -f mps-pod.yaml
```

4. Überwachen Sie die Ressourcenansprüche:

```
kubectl get resourceclaims -n mps-gpu -w
```

Das Folgende ist eine Beispielausgabe:

```
NAME                                              STATE                AGE
mps-multi-container-pod-shared-gpu-claim-2p9kx   allocated,reserved   86s
```

Diese Konfiguration demonstriert echte GPU-Sharing mithilfe von NVIDIA Multi-Process Service (MPS) durch dynamische Ressourcenzuweisung (DRA). Im Gegensatz zum Time-Slicing, bei dem Workloads die GPU nacheinander verwenden, ermöglicht MPS, dass beide Container gleichzeitig auf derselben physischen GPU ausgeführt werden. Die wichtigste Erkenntnis ist, dass die gemeinsame Nutzung von DRA MPS mehrere Container innerhalb eines einzigen Pods erfordert, nicht mehrere separate Pods. Bei der Bereitstellung weist der DRA-Treiber dem Pod einen `ResourceClaim` zu und konfiguriert MPS automatisch so, dass sowohl der Inferenz- als auch der Trainingscontainer gleichzeitig ausgeführt werden können.

Jeder Container erhält seinen eigenen isolierten GPU-Speicherplatz und eigene Rechenressourcen, wobei der MPS-Daemon den Zugriff auf die zugrunde liegende Hardware koordiniert. Sie können überprüfen, ob dies funktioniert, indem Sie wie folgt vorgehen:

- Überprüfung`nvidia-smi`, wodurch beide Container als M+C (`MPS + Compute`) -Prozesse angezeigt werden, die sich dasselbe GPU-Gerät teilen.
- Überwachung der Logs aus beiden Containern, wobei verschachtelte Zeitstempel angezeigt werden, die die gleichzeitige Ausführung belegen.

Dieser Ansatz maximiert die GPU-Auslastung, indem ergänzende Workloads die teure GPU-Hardware effizient gemeinsam nutzen können, anstatt sie durch einen einzelnen Prozess ungenutzt zu lassen.

Behälter 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 |
+-----------------------------------------------------------------------------------------
+
```

Behälter 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 |
+-----------------------------------------------------------------------------------------
+
```

Optimieren Sie GPU-Workloads mit Multi-Instance-GPU

Die Multi-Instance-GPU (MIG) ermöglicht Partitionierung auf Hardwareebene und erstellt isolierte GPU-Instanzen mit dedizierten Rechen- und Speicherressourcen.

Für die Verwendung der dynamischen MIG-Partitionierung mit verschiedenen Profilen ist der NVIDIA-GPU-Operator erforderlich. Der NVIDIA GPU-Operator verwendet MIG Manager, um MIG-Profile zu erstellen, und startet die GPU-Instanzen wie P4D, P4De, P5, P6 und mehr neu, um die Konfigurationsänderungen zu übernehmen. Der GPU-Operator verfügt über die MIG Manager-Komponente, die auf Änderungen der Knotenbezeichnung achtet und automatisch die entsprechende MIG-Konfiguration anwendet, umfassende MIG-Verwaltungsfunktionen. Wenn eine Änderung des MIG-Profils angefordert wird, fährt der Operator alle GPU-Clients ordnungsgemäß herunter, wendet die neue Partitionsgeometrie an und startet die betroffenen Dienste neu. Dieser Prozess erfordert einen Neustart des Knotens für GPU-Instanzen, um saubere GPU-Statusübergänge zu gewährleisten. Aus diesem Grund ist die Aktivierung `WITH0REBOOT=true` in der MIG Manager-Konfiguration für erfolgreiche MIG-Bereitstellungen unerlässlich.

Sie benötigen sowohl den NVIDIA DRA-Treiber als auch den NVIDIA GPU Operator, um mit MIG in Amazon EKS zu arbeiten. Darüber hinaus benötigen Sie weder das NVIDIA Device Plugin noch den DCGM Exporter, da diese Teil des NVIDIA GPU Operators sind. Da auf den EKS NVIDIA die NVIDIA-Treiber vorinstalliert AMIs sind, haben wir die Bereitstellung von Treibern durch den GPU-Operator deaktiviert, um Konflikte zu vermeiden und die optimierten Treiber zu nutzen, die bereits auf den Instanzen vorhanden sind. Der NVIDIA DRA-Treiber kümmert sich um die dynamische Ressourcenzuweisung für MIG-Instanzen, während der GPU-Operator den gesamten GPU-Lebenszyklus verwaltet. Dazu gehören die MIG-Konfiguration, die Funktionalität von Geräte-Plug-ins, die Überwachung durch DCGM und die Erkennung von Knotenfunktionen. Dieser integrierte Ansatz bietet eine Komplettlösung für das GPU-Management in Unternehmen mit Isolierung auf Hardwareebene und Funktionen zur dynamischen Ressourcenzuweisung.

Schritt 1: Stellen Sie NVIDIA GPU Operator bereit

1. Fügen Sie das NVIDIA GPU Operator-Repository hinzu:

```
helm repo add nvidia https://nvidia.github.io/gpu-operator
helm repo update
```

2. Erstellen Sie eine `gpu-operator-values.yaml` Datei:

```
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. Installieren Sie GPU Operator mithilfe der `gpu-operator-values.yaml` folgenden Datei:

```
helm install gpu-operator nvidia/gpu-operator \
  --namespace gpu-operator \
  --create-namespace \
  --version v25.3.1 \
  --values gpu-operator-values.yaml
```

Dieses Helm-Diagramm stellt die folgenden Komponenten und mehrere MIG-Profile bereit:

- Geräte-Plugin (GPU-Ressourcenplanung)
- DCGM Exporter (GPU-Metriken und Überwachung)
- Erkennung von Knotenmerkmalen (NFD — Hardwarekennzeichnung)

- GPU Feature Discovery (GFD — GPU-spezifische Kennzeichnung)
- MIG Manager (GPU-Partitionierung mit mehreren Instanzen)
- Container-Toolkit (GPU-Container-Laufzeit)
- Operator Controller (Lebenszyklusmanagement)

4. Überprüfen Sie die Bereitstellungs-Pods:

```
kubectl get pods -n gpu-operator
```

Das Folgende ist eine Beispielausgabe:

```
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. Erstellen Sie einen Amazon EKS-Cluster mit einer von P4de verwalteten Knotengruppe, um die MIG-Beispiele zu testen:

```
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 verwendet das Label, das den Knoten hinzugefügt wurde, `nvidia.com/mig.config: "p4de-half-balanced"` und partitioniert die GPU mit dem angegebenen Profil.

6. Melden Sie sich bei der p4de Instanz an.
7. Führen Sie den folgenden Befehl aus:

```
nvidia-smi -L
```

Sie sollten die folgende Beispielausgabe sehen:

```
[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 hat das `p4de-half-balanced` MIG-Profil erfolgreich auf Ihre P4DE-Instanz angewendet und dabei wie konfiguriert GPU-Partitionen auf Hardwareebene erstellt. So funktioniert die Partitionierung:

Der GPU-Operator hat diese Konfiguration aus Ihrem eingebetteten MIG-Profil übernommen:

```
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
```

Aus Ihrer `nvidia-smi -L` Ausgabe hat der GPU-Operator Folgendes erstellt:

- MIG-fähig GPUs (0-3): Hardware partitioniert
  - GPU 0: NVIDIA A100- -80 GB SXM4
    - MIG 3G.40 GB Gerät 0 — Große Arbeitslasten (40 GB Speicher, 42) SMs
    - MIG 2G.20 GB Gerät 1 — Mittlere Arbeitslasten (20 GB Speicher, 28) SMs
    - MIG 1g.10gb Gerät 2 — Kleine Workloads (10 GB Speicher, 14) SMs
    - MIG 1G.10GB Gerät 3 — Kleine Workloads (10 GB Speicher, 14) SMs
  - GPU 1: NVIDIA A100-80 GB SXM4

    - MIG 3G.40gb Gerät 0 — Identisches Partitionslayout
    - MIG 2g.20gb Gerät 1
    - MIG 1g.10gb Gerät 2
    - MIG 1g.10gb Gerät 3
  - GPU 2 und GPU 3 — Gleiches Muster wie GPU 0 und GPU 1
- Vollständig GPUs (4-7): Keine MIG-Partitionierung
  - GPU 4: NVIDIA A100-80 GB — Vollständige SXM4 80-GB-GPU
  - GPU 5: NVIDIA A100- SXM4 -80 GB — Vollständige 80-GB-GPU
  - GPU 6: NVIDIA A100- SXM4 -80 GB — Vollständige 80-GB-GPU
  - GPU 7: NVIDIA A100- SXM4 -80 GB — Vollständige 80-GB-GPU

Sobald der NVIDIA-GPU-Operator die MIG-Partitionen erstellt hat, erkennt der NVIDIA DRA-Treiber diese hardwareisolierten Instanzen automatisch und stellt sie für die dynamische Ressourcenzuweisung in Kubernetes zur Verfügung. Der DRA-Treiber erkennt jede MIG-Instanz mit ihrem spezifischen Profil (1G.10 GB, 2G.20 GB, 3G.40 GB) und stellt sie über die Geräteklasse als planbare Ressourcen zur Verfügung. `mig.nvidia.com`

Der DRA-Treiber überwacht kontinuierlich die MIG-Topologie und führt eine Bestandsaufnahme aller verfügbaren Instanzen. GPUs Wenn ein Pod über ein ein bestimmtes MIG-Profil anfordert`ResourceClaimTemplate`, wählt der DRA-Treiber auf intelligente Weise eine geeignete MIG-Instanz aus jeder verfügbaren GPU aus und ermöglicht so eine echte Mehrmandantenfähigkeit auf Hardwareebene. Durch diese dynamische Zuweisung können mehrere isolierte Workloads gleichzeitig auf derselben physischen GPU ausgeführt werden, wobei strenge Ressourcengrenzen und Leistungsgarantien eingehalten werden.

Schritt 2: Testen Sie die MIG-Ressourcenzuweisung

Lassen Sie uns nun anhand einiger Beispiele demonstrieren, wie DRA MIG-Instanzen dynamisch verschiedenen Workloads zuweist. Stellen Sie die Pods bereit `resourceclaimtemplates` und testen Sie sie, um zu sehen, wie der DRA-Treiber Workloads auf die verfügbaren MIG-Partitionen verteilt, sodass mehrere Container GPU-Ressourcen gemeinsam nutzen können, wobei die Isolierung auf Hardwareebene erfolgt.

1. Erstellen, um `mig-claim-template.yaml` die MIG einzuschließen: `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. Wenden Sie die drei Vorlagen an:

```
kubectl apply -f mig-claim-template.yaml
```

3. Führen Sie den folgenden Befehl aus:

```
kubectl get resourceclaimtemplates -n mig-gpu
```

   Das Folgende ist eine Beispielausgabe:

```
NAME                  AGE
mig-large-template    71m
mig-medium-template   71m
mig-small-template    71m
```

4. Erstellen Sie`mig-pod.yaml`, um mehrere Jobs zu planen und diese Vorteile zu nutzen`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. Wenden Sie diese Spezifikation an, die drei Pods bereitstellen sollte:

```
kubectl apply -f mig-pod.yaml
```

Diese Pods sollten vom DRA-Treiber geplant werden.

6. Überprüfen Sie die Pod-Logs des DRA-Treibers und Sie erhalten eine Ausgabe, die der folgenden ähnelt:

```
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. Überprüfen Sie`resourceclaims`, um den Pod-Status zu sehen:

```
kubectl get resourceclaims -n mig-gpu -w
```

Das Folgende ist eine Beispielausgabe:

```
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
```

Wie Sie sehen, `allocated,reserved` wurden alle Pods vom DRA-Treiber von „Ausstehend“ zu „Ausstehend“ verschoben.

8. Führen Sie `nvidia-smi` vom Knoten aus aus. Sie werden feststellen, dass drei Python-Prozessoren laufen:

```
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 |**
+-----------------------------------------------------------------------------------------
+
```

Optimieren Sie GPU-Workloads mit IMEX mithilfe von GB200 P6e-Instanzen

IMEX (Internode Memory Exchange) ermöglicht eine speicherkohärente Kommunikation zwischen Knoten für verteiltes Training auf NVIDIA. GB200 UltraServers

Führen Sie die folgenden Schritte aus.

1. Definieren Sie ein `ComputeDomain` für ein Training mit mehreren Knoten mit einer Datei namens`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. Definieren Sie einen Pod mithilfe von IMEX-Kanälen mit einer Datei namens: `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**
>
> Dies erfordert P6e-Instanzen GB200 .

3. Stellen Sie IMEX bereit, indem Sie die `ComputeDomain` Vorlagen und anwenden:

```
kubectl apply -f imex-claim-template.yaml
kubectl apply -f imex-compute-domain.yaml
kubectl apply -f imex-pod.yaml
```

4. Überprüfen Sie den `ComputeDomain` Status.

```
kubectl get computedomain distributed-training-domain
```

5. Überwachen Sie die Bereitstellung des IMEX-Daemons.

```
kubectl get pods -n nvidia-dra-driver -l resource.nvidia.com/computeDomain
```

6. Überprüfen Sie die IMEX-Kanäle im Pod:

```
kubectl exec imex-distributed-training -- ls -la /dev/nvidia-caps-imex-channels/
```

7. Die Pod-Logs anzeigen:

```
kubectl logs imex-distributed-training
```

   Im Folgenden finden Sie ein Beispiel für die erwartete Ausgabe:

```
=== 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
```

Weitere Informationen finden Sie im NVIDIA-Beispiel unter GitHub.

Netzwerk

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Ziehen Sie für Anwendungen mit hoher Kommunikation zwischen Knoten eine höhere Netzwerkbandbreite oder einen Elastic Fabric-Adapter in Betracht

Für verteilte Trainingsworkloads auf Amazon EKS mit hohen Kommunikationsanforderungen zwischen den Knoten sollten Sie Instances mit höherer Netzwerkbandbreite oder Elastic Fabric Adapter (EFA) auswählen. Eine unzureichende Netzwerkleistung kann zu Engpässen bei der Datenübertragung führen und Aufgaben des maschinellen Lernens wie verteiltes Training mit

mehreren GPUs verlangsamen. Beachten Sie, dass bei Inferenz-Workloads in der Regel kein hohes Maß an Kommunikation zwischen den Knoten stattfindet.

Beispiel

Verwenden Sie zum Beispiel 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
```

Stellen Sie sicher, dass Tools wie MPI und NCCL in Ihrem Container-Image installiert sind, um EFA für Schulungsaufgaben nutzen zu können.

Erhöhen Sie die Anzahl der verfügbaren IP-Adressen, um schnellere Pod-Startzeiten zu ermöglichen

In EKS benötigt jeder Pod eine IP-Adresse aus dem VPC-CIDR-Block. Wenn Ihr Cluster mit mehr Knoten und Pods skaliert wird, riskieren Sie eine Erschöpfung der IP-Adressen oder eine langsamere Leistung. Durch die Aktivierung der Präfix-Delegierung können diese Probleme jedoch behoben werden, indem IP-Bereiche vorab zugewiesen und EC2-API-Aufrufe reduziert werden, was zu schnelleren Pod-Startzeiten und verbesserter Skalierbarkeit führt.

Wenn Sie die Präfix-Delegierung nach der Erstellung Ihres Clusters aktivieren, kann das VPC Container Network Interface (CNI) Netzwerkschnittstellen auf EC2-Instances IP-Präfixe (/28, jeweils 16 IP-Adressen) zuweisen. Das bedeutet, dass jeder Knoten mehr Pods unterstützen kann, wodurch das Risiko von IP-Knappheit reduziert wird. Auf einer `c5.4xlarge` Instance können Sie beispielsweise bis zu 110 Pods mit Präfix-Delegierung unterstützen.

Während die Präfix-Delegierung für die Optimierung der IP-Nutzung in Umgebungen mit vielen kleinen Pods entscheidend ist, verwenden AI/ML Workloads oft weniger, größere Pods (z. B. ein Pod pro GPU). Durch die Aktivierung der Präfix-Delegierung kann die VPC-CNI vorab ein Präfix für einen schnelleren Pod-Start zuweisen, indem ein warmer Pool aufrechterhalten wird. Das bedeutet, dass IP-Adressen jederzeit verfügbar sind, was den Zeitaufwand für die Pod-Initialisierung im Vergleich zur On-Demand-Zuweisung im Modus ohne Präfix reduziert. In solchen Fällen bieten die IP-Einsparungen durch die Aktivierung der Präfix-Delegierung Leistungsvorteile für AI/ML Workloads. Durch die Reduzierung der Anzahl der EC2-API-Aufrufe, die für die IP-Adresskonfiguration und die Vorabzuweisung von IP-Bereichen erforderlich sind, ermöglicht die Verwendung der Präfix-Delegierung schnellere Pod-Startzeiten, was besonders für die schnelle Skalierung von Workloads von Vorteil ist. AI/ML

So aktivieren Sie die Präfix-Delegierung:

```
kubectl set env daemonset/aws-node -n kube-system ENABLE_PREFIX_DELEGATION=true
```

Sorgen Sie für eine angemessene Planung der VPC-Subnetze, um eine Überlastung der IP-Adressen zu vermeiden, insbesondere bei großen Bereitstellungen, und verwalten Sie CIDR-Blöcke, um Überschneidungen zwischen den einzelnen Bereichen zu vermeiden. VPCs Weitere Informationen finden Sie unter Optimieren der IP-Adressnutzung und Zuweisen von mehr IP-Adressen zu Amazon EKS-Knoten mit Präfixen.

Sicherheit

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Sicherheit und Compliance

Ziehen Sie S3 mit KMS für verschlüsselungskonformen Speicher in Betracht

Sofern Sie nichts anderes angeben, verwenden alle S3-Buckets standardmäßig SSE-S3, um Objekte im Ruhezustand zu verschlüsseln. Sie können Buckets jedoch so konfigurieren, dass sie stattdessen serverseitige Verschlüsselung mit AWS Key Management Service (AWS KMS) -Schlüsseln (SSE-KMS) verwenden. Die Sicherheitskontrollen in AWS KMS tragen dazu bei, dass Sie Ihre Compliance-

Anforderungen bezüglich der Verschlüsselung erfüllen. Sie können mit diesen KMS-Schlüsseln Ihre Daten in Amazon-S3-Buckets schützen. Wenn Sie die SSE-KMS-Verschlüsselung mit einem S3-Bucket verwenden, müssen sich die AWS-KMS-Schlüssel in derselben Region wie der Bucket befinden.

Konfigurieren Sie Ihre Allzweck-Buckets für die Verwendung von S3 Bucket Keys für SSE-KMS, um Ihre Kosten für AWS KMS-Anfragen um bis zu 99 Prozent zu senken, indem Sie den Anforderungsverkehr von Amazon S3 zu AWS KMS reduzieren. S3-Bucket-Keys sind immer für GET und für PUT Operationen in einem Verzeichnis-Bucket aktiviert und können nicht deaktiviert werden.

Beachten Sie, dass Amazon S3 Express One Zone einen bestimmten Bucket-Typ verwendet, der als S3-Verzeichnis-Bucket bezeichnet wird. Directory-Buckets sind ausschließlich für die Speicherklasse S3 Express One Zone vorgesehen und ermöglichen einen leistungsstarken Zugriff mit niedriger Latenz. Um die Standard-Bucket-Verschlüsselung in einem S3-Verzeichnis-Bucket zu konfigurieren, verwenden Sie die AWS-CLI und geben Sie die KMS-Schlüssel-ID oder den ARN an, nicht den Alias, wie im folgenden Beispiel:

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"}}]}'
```

Stellen Sie sicher, dass die IAM-Rolle Ihres EKS-Pods über KMS-Berechtigungen (z. B.`kms:Decrypt`) für den Zugriff auf verschlüsselte Objekte verfügt. Testen Sie dies in einer Staging-Umgebung, indem Sie ein Beispielmodell in den Bucket hochladen, es in einen Pod einbauen (z. B. über den Mountpoint S3 CSI-Treiber) und überprüfen, ob der Pod die verschlüsselten Daten fehlerfrei lesen kann. Prüfen Sie Protokolle über AWS CloudTrail , um die Einhaltung der Verschlüsselungsanforderungen zu bestätigen. Einzelheiten zur Einrichtung und zur Schlüsselverwaltung finden Sie in der KMS-Dokumentation.

Speicher

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Datenverwaltung und Speicherung

Stellen Sie mithilfe eines CSI-Treibers KI-Modelle auf Pods bereit

KI/ML-Workloads erfordern häufig Zugriff auf große Modellartefakte (z. B. trainierte Gewichte, Konfigurationen), und Pods benötigen eine zuverlässige, skalierbare Methode, um auf diese zuzugreifen, ohne sie in Container-Images einzubetten, was die Bildgröße und die Abrufzeiten der Container-Registry erhöhen kann. Um den betrieblichen Aufwand bei der Verwaltung von Volume-Mounts zu reduzieren, empfehlen wir die Bereitstellung von KI-Modellen auf Pods, indem Amazon-Speicherdienste (z. B. S3, FSx für Lustre, FSx für OpenZFS, EFS) als persistente Volumes (PVs) mithilfe ihrer jeweiligen CSI-Treiber bereitgestellt werden. Einzelheiten zur Implementierung finden Sie in den nachfolgenden Themen in diesem Abschnitt.

Optimieren Sie den Speicher für ML-Modell-Caches auf EKS

Die Nutzung einer optimalen Speicherlösung ist entscheidend, um die Latenz beim Starten von Pods und Anwendungen zu minimieren, den Speicherverbrauch zu reduzieren, das gewünschte Leistungsniveau zur Beschleunigung von Workloads zu erreichen und die Skalierbarkeit von ML-Workloads sicherzustellen. ML-Workloads basieren häufig auf Modelldateien (Gewichtungen), die groß sein können und gemeinsamen Zugriff auf Daten über Pods oder Knoten hinweg erfordern. Die Auswahl der optimalen Speicherlösung hängt von den Merkmalen Ihres Workloads ab, wie z. B. der Effizienz eines einzelnen Knotens, dem Zugriff auf mehrere Knoten, den Latenzanforderungen, den Kostenbeschränkungen und auch den Anforderungen an die Datenintegration (z. B. mit einem Amazon S3 S3-Datenrepository). Wir empfehlen, verschiedene Speicherlösungen mit Ihren Workloads zu vergleichen, um herauszufinden, welche Ihren Anforderungen entspricht. Wir haben die folgenden Optionen bereitgestellt, um Ihnen bei der Bewertung anhand Ihrer Workload-Anforderungen zu helfen.

Der EKS-CSI-Treiber unterstützt die folgenden AWS-Speicherservices, von denen jeder seinen eigenen CSI-Treiber hat und seine eigenen Stärken für KI- und ML-Workflows hat:

- Mountpoint für Amazon S3
- Amazon FSx für Lustre
- Amazon FSx für OpenZFS
- Amazon EFS
- Amazon EBS

Die Wahl des AWS-Speicherservices hängt von Ihrer Bereitstellungsarchitektur, Ihrem Umfang, Ihren Leistungsanforderungen und Ihrer Kostenstrategie ab. Speicher-CSI-Treiber müssen auf Ihrem EKS-Cluster installiert werden, sodass der CSI-Treiber persistente Volumes (PV) außerhalb des Lebenszyklus eines Pods erstellen und verwalten kann. Mithilfe des CSI-Treibers können Sie PV-Definitionen unterstützter AWS-Speicherservices als EKS-Clusterressourcen erstellen. Pods können dann auf diese Speichervolumes für ihre Datenvolumen zugreifen, indem sie einen Persistent Volume Claim (PVC) für den PV erstellen. Abhängig vom AWS-Speicherservice und Ihrem Bereitstellungsszenario kann ein einzelnes PVC (und das zugehörige PV) für einen Workload an mehrere Pods angehängt werden. Bei ML-Trainings werden gemeinsame Trainingsdaten beispielsweise auf einem PV gespeichert und von mehreren Pods abgerufen. Für Online-Inferenzen in Echtzeit werden LLM-Modelle auf einem PV zwischengespeichert und von mehreren Pods abgerufen. Nachfolgend finden Sie Beispiele für PV- und PVC-YAML-Dateien für AWS-Speicherservices, die Ihnen den Einstieg erleichtern.

Leistungsüberwachung Eine schlechte Festplattenleistung kann das Lesen von Container-Images verzögern, die Latenz beim Pod-Start erhöhen und den Inferenz- oder Trainingsdurchsatz verringern. Verwenden Sie Amazon CloudWatch, um die Leistungskennzahlen für Ihre AWS-Speicherservices zu überwachen. Wenn Sie Leistungsengpässe feststellen, ändern Sie Ihre Speicherkonfigurationsparameter, um die Leistung zu optimieren.

Szenario: Arbeitslast mehrerer GPU-Instanzen

Amazon FSx for Lustre: In Szenarien, in denen Sie mehrere EC2-GPU-Computing-Instance-Umgebungen mit latenzsensitiven dynamischen Workloads und hohem Bandbreitendurchsatz haben, wie z. B. verteiltes Training und Model Serving, und Sie eine native Amazon S3 S3-Datenrepository-Integration benötigen, empfehlen wir Amazon for Lustre. FSx FSx for Lustre bietet ein vollständig verwaltetes paralleles Hochleistungsdateisystem, das für rechenintensive Workloads wie Hochleistungsrechnen (HPC) und Machine Learning konzipiert ist.

Sie können den FSx for Lustre CSI-Treiber installieren, um FSx Dateisysteme auf EKS als persistentes Volume (PV) zu mounten, und dann das Lustre-Dateisystem als eigenständigen Hochleistungs-Cache oder als S3-verknüpftes Dateisystem bereitstellen FSx , das als Hochleistungs-Cache für S3-Daten fungiert und einen schnellen I/O und hohen Durchsatz für den Datenzugriff über Ihre GPU-Recheninstanzen hinweg bietet. FSx for Lustre kann entweder mit Scratch-SSD- oder Persistent-SSD-Speicheroptionen bereitgestellt werden:

- Scratch-SSD-Speicher: Empfohlen für kurzlebige oder kurzlebige Workloads (Stunden), bei denen eine feste Durchsatzkapazität pro TIB bereitgestellt wird.

- Persistenter SSD-Speicher: Empfohlen für unternehmenskritische, lang andauernde Workloads, die ein Höchstmaß an Verfügbarkeit erfordern, z. B. HPC-Simulationen, Big-Data-Analysen oder Machine-Learning-Schulungen. Mit persistentem SSD-Speicher können Sie sowohl die erforderliche Speicherkapazität als auch die Durchsatzkapazität (pro TIB) konfigurieren.

Überlegungen zur Leistung:

- Administrativer Pod FSx zur Verwaltung des Lustre-Dateisystems: Konfigurieren Sie einen „administrativen“ Pod, auf dem der Lustre-Client installiert und das FSx-Dateisystem gemountet ist. Dadurch wird ein Access Point zur Feinabstimmung des FSx-Dateisystems und auch in Situationen ermöglicht, in denen Sie das FSx-Dateisystem mit Ihren ML-Trainingsdaten oder LLM-Modellen vorwärmen müssen, bevor Sie Ihre GPU-Recheninstanzen starten. Dies ist besonders wichtig, wenn Ihre Architektur Spot-basierte Amazon EC2 GPU/compute EC2-Instances verwendet, bei denen Sie den administrativen Pod verwenden können, um die gewünschten Daten in das FSx-Dateisystem zu „erwärmen“ oder „vorab zu laden“, sodass die Daten für die Verarbeitung bereit sind, wenn Sie Ihre Spot-basierten Amazon EC2 EC2-Instances ausführen.
- Elastic Fabric Adapter (EFA): Persistent-SSD-Speicherbereitstellungstypen unterstützen den Elastic Fabric Adapter (EFA), wobei die Verwendung von EFA ideal für GPU-basierte Workloads mit hoher Leistung und Durchsatz ist. Beachten Sie, dass FSx For Lustre NVIDIA GPUDirect Storage (GDS) unterstützt. Dabei handelt es sich bei GDS um eine Technologie, die einen direkten Datenpfad zwischen lokalem oder Remote-Speicher und GPU-Speicher herstellt, um einen schnelleren Datenzugriff zu ermöglichen.
- Komprimierung: Aktivieren Sie die Datenkomprimierung im Dateisystem, wenn Sie über Dateitypen verfügen, die komprimiert werden können. Dies kann zur Steigerung der Leistung beitragen, da die Datenkomprimierung die Datenmenge reduziert, die zwischen FSx For Lustre-Dateiservern und Speichern übertragen wird.
- Striping-Konfiguration des Lustre-Dateisystems:
  - Data Striping: Ermöglicht es FSx Luster, die Daten einer Datei auf mehrere Object Storage Targets (OSTs) innerhalb eines Lustre-Dateisystems zu verteilen, wodurch der parallel Zugriff und der Durchsatz maximiert werden, insbesondere für umfangreiche ML-Trainingsaufgaben.
  - Eigenständiges Dateisystem-Striping: Standardmäßig wird mithilfe der PFL-Funktion (Progressive File Layouts) von for Lustre eine 4-Komponenten-Lustre-Striping-Konfiguration für Sie erstellt. FSx In den meisten Szenarien müssen Sie die standardmäßige Anzahl und Größe der PFL-Lustre-Stripes nicht aktualisieren. Wenn Sie das Lustre-Daten-Striping anpassen

müssen, können Sie das Lustre-Striping manuell anpassen, indem Sie sich auf die Striping-Parameter eines for Lustre-Dateisystems beziehen. FSx

- S3-verknüpftes Dateisystem: Dateien, die mit der nativen Amazon S3 S3-Integration (Data Repository Association oder DRA) in das FSx Dateisystem importiert wurden, verwenden nicht das standardmäßige PFL-Layout, sondern das Layout im Dateisystemparameter. `ImportedFileChunkSize` S3-importierte Dateien, die größer als die sind, `ImportedFileChunkSize` werden auf mehreren gespeichert, OSTs wobei die Anzahl der Stripes auf dem `ImportedFileChunkSize` definierten Wert basiert (Standard 1 GiB). Wenn Sie große Dateien haben, empfehlen wir, diesen Parameter auf einen höheren Wert einzustellen.
- Platzierung: Stellen Sie ein FSx for Lustre-Dateisystem in derselben Availability Zone wie Ihre Rechen- oder GPU-Knoten bereit, um den Zugriff auf die Daten mit der geringsten Latenz zu ermöglichen und Zugriffsmuster zwischen Availability Zones zu vermeiden. Wenn Sie mehrere GPU-Knoten in verschiedenen Availability Zones haben, empfehlen wir die Bereitstellung eines FSx Dateisystems in jeder Availability Zone für den Datenzugriff mit niedriger Latenz.

Beispiel

Definition eines persistenten Volumes (PV) FSx für ein For Lustre-Dateisystem unter Verwendung von Static Provisioning (wobei die FSx Instanz bereits bereitgestellt wurde).

```
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]
```

Beispiel

Die Definition von Persistent Volume Claim für PV lautet: `fsx-pv`

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: fsx-claim
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 1200Gi
  volumeName: fsx-pv
```

Beispiel

Konfigurieren Sie einen Pod für die Verwendung eines persistenten Volumenanspruchs von`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
```

Vollständige Beispiele finden Sie unter Beispiele FSx für Lustre-Treiber unter GitHub. Überwachen Sie die Leistungskennzahlen von Amazon FSx for Lustre mithilfe von Amazon CloudWatch. Wenn Leistungsengpässe festgestellt werden, passen Sie Ihre Konfigurationsparameter nach Bedarf an.

Szenario: Arbeitslast einer einzelnen GPU-Instanz

Mountpoint für Amazon S3 mit CSI-Treiber: Mit dem Mountpoint for Amazon S3 CSI-Treiber können Sie einen S3-Bucket als Volume in Ihren Pods mounten. Diese Methode ermöglicht eine detaillierte Zugriffskontrolle darüber, welche Pods auf bestimmte S3-Buckets zugreifen können. Jeder Pod verfügt über eine eigene Mountpoint-Instanz und einen eigenen lokalen Cache (5-10 GB), wodurch die Lade- und Leseleistung des Modells zwischen den Pods isoliert wird. Dieses Setup unterstützt die Authentifizierung auf Pod-Ebene mit IAM Roles for Service Accounts (IRSA) und die unabhängige Modellversionierung für verschiedene Modelle oder Kunden. Der Kompromiss besteht in einer erhöhten Speichernutzung und einem erhöhten API-Verkehr, da jeder Pod S3-API-Aufrufe ausgibt und seinen eigenen Cache verwaltet.

Beispiel für ein teilweises Beispiel für eine Pod-Implementierung mit YAML und CSI-Treiber:

```
# 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
```

Überlegungen zur Leistung:

- Daten-Caching: Mountpoint for S3 kann Inhalte zwischenspeichern, um Kosten zu senken und die Leistung bei wiederholten Lesevorgängen derselben Datei zu verbessern. Informationen zu den Caching-Optionen und -Parametern finden Sie unter Caching-Konfiguration.
- Objektgröße: Wenn Sie Dateien mit einer Größe von mehr als 72 GB speichern und darauf zugreifen, finden Sie unter Konfiguration der Mountpoint-Leistung Informationen darüber, wie Sie die Parameter `--read-part-size` und die `--write-part-size` Befehlszeilenparameter so konfigurieren, dass sie Ihr Datenprofil und Ihre Workload-Anforderungen erfüllen.
- Shared-Cache ist für Objekte mit einer Größe von bis zu 1 MB konzipiert. Es unterstützt keine großen Objekte. Verwenden Sie die Option Lokaler Cache, um Objekte in NVMe oder EBS-Volumes auf dem EKS-Knoten zwischenzuspeichern.
- Gebühren für API-Anfragen: Wenn Sie mit dem Mountpoint für S3 eine große Anzahl von Dateivorgängen ausführen, können die Gebühren für API-Anfragen zu einem Teil der Speicherkosten werden. Um dem entgegenzuwirken, sollten Sie, wenn keine starke Konsistenz erforderlich ist, immer das Zwischenspeichern von Metadaten aktivieren und den `metadata-ttl` Zeitraum festlegen, in dem die Anzahl der API-Operationen auf S3 reduziert werden soll.

Weitere Informationen finden Sie im Mountpoint for Amazon S3 CSI-Treiber in der offiziellen Amazon EKS-Dokumentation. Wir empfehlen, die Leistungskennzahlen von Amazon S3 mit CloudWatch Amazon-Metriken zu überwachen, falls Engpässe auftreten, und Ihre Konfiguration gegebenenfalls anzupassen.

Amazon FSx für OpenZFS — persistenter gemeinsam genutzter Speicher

Für Szenarien mit mehreren EC2-GPU-Recheninstanzen mit latenzempfindlichen Workloads, die hohe Verfügbarkeit, hohe Leistung, Kostensensitivität und mehrere Pod-Bereitstellungen für verschiedene Anwendungen erfordern, empfehlen wir Amazon for OpenZFS. FSx Einige Beispiele für Workloads umfassen Inferenz in Echtzeit, Reinforcement-Learning und Training generativer gegnerischer Netzwerke. FSx für OpenZFS ist besonders vorteilhaft für Workloads, die einen leistungsstarken Zugriff auf eine fokussierte Verzeichnisstruktur mit kleinen Dateien und kleinen I/O-Datenzugriffsmustern benötigen. Außerdem bietet OpenZFS die Flexibilität, FSx die Leistung unabhängig von der Speicherkapazität zu skalieren, sodass Sie eine optimale Kosteneffizienz erzielen können, indem Sie die Speichergröße an die tatsächlichen Bedürfnisse anpassen und gleichzeitig das erforderliche Leistungsniveau beibehalten

Der native CSI-Treiber FSx für OpenZFS ermöglicht die Erstellung mehrerer Dateisysteme in einem einzigen PVCs , indem mehrere Volumes erstellt werden. Dies reduziert den Verwaltungsaufwand und maximiert die Nutzung des Durchsatzes und der IOPS des Dateisystems durch konsolidierte

Anwendungs-Pod-Bereitstellungen auf einem einzigen Dateisystem. Darüber hinaus umfasst es Unternehmensfunktionen wie Zero-Copy-Snapshots, Zero-Copy-Clones sowie Benutzer- und Gruppenkontingente, die dynamisch über den CSI-Treiber bereitgestellt werden können.

FSx for OpenZFS unterstützt bei der Erstellung drei verschiedene Bereitstellungstypen:

- Single-AZ: Die kostengünstigste Option mit Latenzen unter einer Millisekunde, bietet jedoch keine Hochverfügbarkeit auf Dateisystem- oder Availability Zone-Ebene. Empfohlen für Entwicklungs- und Test-Workloads oder solche mit hoher Verfügbarkeit auf Anwendungsebene.
- Single-AZ (HA): Bietet Hochverfügbarkeit auf Dateisystemebene mit Latenzen unter einer Millisekunde. Empfohlen für Workloads mit der höchsten Leistung, die eine hohe Verfügbarkeit erfordern.
- Multi-AZ: Bietet Hochverfügbarkeit auf Dateisystemebene sowie in allen Availability Zones. Empfohlen für Hochleistungs-Workloads, die zusätzliche Verfügbarkeit in allen Availability Zones erfordern.

Überlegungen zur Leistung:

- Bereitstellungstyp: Wenn die zusätzliche Verfügbarkeit in allen Availability Zones nicht erforderlich ist, sollten Sie den Bereitstellungstyp Single-AZ (HA) in Betracht ziehen. Dieser Bereitstellungstyp bietet bis zu 100% des Durchsatzes für Schreibvorgänge, hält Latenzen von unter einer Millisekunde aufrecht, und die Gen2-Dateisysteme verfügen über einen zusätzlichen NVMe Cache zum Speichern von bis zu Terabyte an häufig aufgerufenen Daten. Die Multi-AZ-Dateisysteme bieten bis zu 75% des Durchsatzes für Schreibvorgänge bei einer erhöhten Latenz, um den AZ-übergreifenden Datenverkehr zu bewältigen.
- Durchsatz und IOPS: Sowohl der Durchsatz als auch die für das Dateisystem konfigurierten IOPS können nach der Bereitstellung nach oben oder unten skaliert werden. Sie können bis zu 10% GB/s of disk throughput providing up to 21GB/s des zwischengespeicherten Datenzugriffs bereitstellen. Die IOPS können von der Festplatte auf bis zu 400.000 skaliert werden, und der Cache kann über 1 Million IOPS bereitstellen. Beachten Sie, dass die Durchsatzskalierung eines Single-AZ-Dateisystems zu einem kurzen Ausfall des Dateisystems führt, da keine Hochverfügbarkeit besteht. Die Durchsatzskalierung eines Single-AZ- (HA) - oder Multi-AZ-Dateisystems kann unterbrechungsfrei erfolgen. Die SSD-IOPS können einmal alle sechs Stunden skaliert werden.
- Speicherklasse: FSx Für OpenZFS werden sowohl die SSD-Speicherklasse als auch die Intelligent-Tiering-Speicherklasse unterstützt. Für AI/ML Workloads wird empfohlen, die

SSD-Speicherklasse zu verwenden, um eine gleichbleibende Leistung für die Arbeitslast zu gewährleisten und die CPUs/GPUs so ausgelastet wie möglich zu halten.

- Komprimierung: Aktivieren Sie den LZ4 Komprimierungsalgorithmus, wenn Sie einen Workload haben, der komprimiert werden kann. Dadurch wird die Datenmenge reduziert, die jede Datei im Cache verbraucht, sodass mehr Daten direkt aus dem Cache bereitgestellt werden können, da Netzwerkdurchsatz und IOPS die Last auf der SSD-Festplatte reduzieren.
- Datensatzgröße: Die meisten AI/ML Workloads profitieren davon, die standardmäßige Datensatzgröße von 128 KB beizubehalten. Dieser Wert sollte nur reduziert werden, wenn der Datensatz aus großen Dateien (über 10 GiB) mit konsistenten kleinen Blockzugriffen von der Anwendung aus unter 128 KiB besteht.

Sobald das Dateisystem erstellt ist, wird vom Dienst automatisch ein zugehöriges Root-Volume erstellt. Es hat sich bewährt, Daten in untergeordneten Volumes des Root-Volumes im Dateisystem zu speichern. Mithilfe des CSI-Treibers FSx für OpenZFS erstellen Sie einen zugehörigen Persistent Volume Claim, um das untergeordnete Volume dynamisch zu erstellen.

Beispiele:

Eine Speicherklassendefinition (SC) für ein Volume FSx für OpenZFS, mit der ein untergeordnetes Volume des Root-Volumes ($ROOT_VOL_ID) in einem vorhandenen Dateisystem erstellt und das Volume mithilfe des NFS v4.2-Protokolls in das VPC CIDR ($VPC_CIDR) exportiert wird.

```
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
```

Ein dynamisch erstellter Persistent Volume Claim (PVC) für das oben erstellte Objekt. fsxz-vol-sc Beachten Sie, dass die zugewiesene Speicherkapazität 1 Gi beträgt. Dies ist FSx für OpenZFS-Volumes erforderlich, wie in den häufig gestellten Fragen zu CSI-Treibern angegeben. Dem Volume wird die volle Kapazität zur Verfügung gestellt, die mit dieser Konfiguration für das Dateisystem bereitgestellt wird. Wenn die Volumenkapazität eingeschränkt werden muss, können Sie dies mithilfe von Benutzer- oder Gruppenkontingenten tun.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: dynamic-vol-pvc
  namespace: example
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: fsxz-vol-sc
  resources:
    requests:
      storage: 1Gi
```

Konfigurieren Sie einen Pod so, dass er ein Volume mithilfe des Persistent Volume Claim (PVC) bereitstellt 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 für gemeinsam genutzte Modell-Caches

In Szenarien, in denen Sie über eine Umgebung mit mehreren EC2-GPU-Recheninstanzen verfügen und dynamische Workloads haben, die gemeinsamen Modellzugriff über mehrere Knoten und Availability Zones (z. B. Online-Inferenz in Echtzeit mit Karpenter) mit moderaten Leistungs- und Skalierbarkeitsanforderungen erfordern, empfehlen wir die Verwendung eines Amazon Elastic File System (EFS) -Dateisystems als persistentes Volume über den EFS CSI-Treiber. Amazon EFS ist ein vollständig verwaltetes, hochverfügbares und skalierbares Cloud-basiertes NFS-Dateisystem, das EC2-Instances und Container mit gemeinsam genutztem Dateispeicher mit konsistenter Leistung ermöglicht, bei der keine vorherige Bereitstellung von Speicher erforderlich ist. Verwenden Sie EFS als Modellvolume und mounten Sie das Volume als gemeinsam genutztes Dateisystem, indem Sie ein persistentes Volume auf dem EKS-Cluster definieren. Jeder Persistent Volume Claim (PVC), der von einem EFS-Dateisystem unterstützt wird, wird als EFS-Zugriffspunkt zum EFS-Dateisystem erstellt. EFS ermöglicht mehreren Knoten und Pods den Zugriff auf dieselben Modelldateien, sodass keine Daten mit dem Dateisystem jedes Knotens synchronisiert werden müssen. Installieren Sie den EFS-CSI-Treiber, um EFS mit EKS zu integrieren.

Sie können ein Amazon EFS-Dateisystem mit den folgenden Durchsatzmodi bereitstellen:

- Bursting-Durchsatz: Skaliert den Durchsatz mit der Größe des Dateisystems und eignet sich für unterschiedliche Workloads mit gelegentlichen Bursts.
- Bereitgestellter Durchsatz: Dedizierter Durchsatz, ideal für konsistente ML-Trainingsaufgaben mit vorhersehbaren Leistungsanforderungen innerhalb bestimmter Grenzen.
- Elastischer Durchsatz (empfohlen für ML): Automatische Skalierung je nach Arbeitslast, Kosteneffektivität für unterschiedliche ML-Workloads.

Leistungsspezifikationen finden Sie unter Amazon EFS-Leistungsspezifikationen.

Überlegungen zur Leistung:

- Verwenden Sie Elastic Throughput für unterschiedliche Workloads.

- Verwenden Sie die Standard-Speicherklasse für aktive ML-Workloads.

Vollständige Beispiele für die Verwendung des Amazon EFS-Dateisystems als persistentes Volume innerhalb Ihres EKS-Clusters und Ihrer Pods finden Sie in den EFS-CSI-Treiberbeispielen unter GitHub. Überwachen Sie Amazon EFS-Leistungskennzahlen mithilfe von Amazon CloudWatch. Wenn Leistungsengpässe festgestellt werden, passen Sie Ihre Konfigurationsparameter nach Bedarf an.

Verwenden Sie S3 Express One Zone für latenzempfindliche, objektorientierte Workflows

Für latenzempfindliche AI/ML Workloads auf Amazon EKS, wie z. B. umfangreiches Modelltraining, Inferenz oder Hochleistungsanalysen, empfehlen wir die Verwendung von S3 Express One Zone für das Speichern und Abrufen von Modellen mit hoher Leistung. S3 Express One Zone bietet einen hierarchischen Namespace, ähnlich einem Dateisystem, in den Sie einfach in einen Verzeichnis-Bucket hochladen, der sich dafür eignet, „alles hineinzuwerfen“ und gleichzeitig die hohe Geschwindigkeit aufrechtzuerhalten. Dies ist besonders nützlich, wenn Sie an objektorientierte Workflows gewöhnt sind. Wenn Sie eher an Dateisysteme gewöhnt sind (z. B. POSIX-konform), bevorzugen Sie möglicherweise Amazon FSx for Lustre oder OpenZFS. Amazon S3 Express One Zone speichert Daten in einer einzigen Availability Zone (AZ) mithilfe von Verzeichnis-Buckets und bietet eine geringere Latenz als Standard-S3-Buckets, die Daten auf mehrere verteilen. AZs Um optimale Ergebnisse zu erzielen, stellen Sie sicher, dass sich Ihre EKS-Rechenleistung in derselben AZ wie Ihr Express One Zone-Bucket befindet. Weitere Informationen zu den Unterschieden von S3 Express One Zone finden Sie unter Unterschiede bei Directory-Buckets.

Für den Zugriff auf S3 Express One Zone mit Dateisystemsemantik empfehlen wir die Verwendung des Mountpoint S3 CSI-Treibers, der S3-Buckets (einschließlich Express One Zone) als lokales Dateisystem mountet. Auf diese Weise werden Dateioperationen (z. B. Öffnen, Lesen, Schreiben) in S3-API-Aufrufe umgewandelt. Dadurch wird ein Zugriff mit hohem Durchsatz ermöglicht, der für leseintensive Workloads von mehreren Clients und sequentielle Schreibvorgänge auf neue Objekte optimiert ist. Einzelheiten zu den unterstützten Vorgängen und Einschränkungen (z. B. keine vollständige POSIX-Konformität, aber Anhängen und Umbenennen werden in Express One Zone unterstützt) finden Sie in der Dokumentation zur Mountpoint-Semantik.

Vorteile bei der Leistung

- Bietet bis zu zehnmal schnelleren Datenzugriff als S3 Standard, mit konsistenter Latenz im einstelligen Millisekundenbereich und bis zu 80% niedrigeren Anforderungskosten.

- Skaliert für die Verarbeitung von Hunderttausenden bis Millionen von Anfragen pro Sekunde pro Verzeichnis-Bucket und vermeidet so Drosselungen oder Stromausfälle, wie sie bei Standard-S3 bei extremer Auslastung auftreten (z. B. von Clustern mit Zehntausenden überlasteten Netzwerken). GPUs/CPUs
- Verwendet einen sitzungsbasierten Authentifizierungsmechanismus. Authentifizieren Sie sich einmal, um ein Sitzungstoken zu erhalten, und führen Sie dann wiederholte Operationen mit hoher Geschwindigkeit aus, ohne dass pro Anfrage ein Authentifizierungsaufwand entsteht. Dies ist für Workloads wie häufiges Checkpointing oder das Laden von Daten optimiert.

Empfohlene Anwendungsfälle

- Caching: Einer der häufigsten Anwendungsfälle bei der Verwendung des Mountpoint S3 CSI-Treibers mit S3 Express One Zone ist das Caching. Die erste Instanz liest Daten aus S3 Standard (für allgemeine Zwecke) und speichert sie in der Express One Zone mit niedrigerer Latenz im Cache. Nachfolgende Lesevorgänge durch andere Clients greifen schneller auf die zwischengespeicherten Daten zu. Dies ist ideal für Szenarien mit mehreren Knoten, in denen mehrere EKS-Knoten dieselben Daten lesen (z. B. gemeinsam genutzte Trainingsdatensätze). Dadurch kann die Leistung bei wiederholten Zugriffen um das bis zu 7-fache verbessert und die Rechenkosten gesenkt werden. Für Workloads, die eine vollständige POSIX-Konformität erfordern (z. B. Dateisperren und direkte Änderungen), sollten Sie Amazon FSx for Lustre oder OpenZFS als Alternativen in Betracht ziehen.
- AI/ML Umfangreiches Training und Inferenz: Ideal für Workloads mit Hunderten oder Tausenden von Rechenknoten (z. B. GPUs in EKS-Clustern), bei denen eine allgemeine Drosselung von S3 zu Verzögerungen führen und teure Rechenressourcen verschwenden könnte. Zum Beispiel tests/checkpoints profitieren LLM-Forscher oder Organisationen, die ein Tagesmodell verwenden, von schnellem und zuverlässigem Zugriff, ohne dass die regionale S3 beeinträchtigt wird. Für kleinere Workloads (z. B. 10 Knoten) können S3 Standard oder andere Speicherklassen ausreichend sein.
- Daten-Pipelines: Load/prepare Modelle, archivierte Trainingsdaten oder Stream-Checkpoints. Wenn Ihr Team Objektspeicher herkömmlichen Dateisystemen vorzieht (z. B. aufgrund der Vertrautheit mit S3), verwenden Sie dies, anstatt technische Änderungen für POSIX-konforme Optionen wie für Lustre vorzunehmen. FSx

Überlegungen

- Belastbarkeit: Das Single-AZ-Design bietet eine Beständigkeit von 99,999999999% (wie Standard S3, durch Redundanz innerhalb der AZ), aber eine geringere Verfügbarkeit (99,95%

konzipiert, 99,9% SLA) im Vergleich zu Multi-AZ-Klassen (99,99% Verfügbarkeit). Es ist weniger widerstandsfähig gegenüber AZ-Ausfällen. Wird für wiederherstellbare oder zwischengespeicherte Daten verwendet. Ziehen Sie Multi-AZ-Replikation oder Backups für kritische Workloads in Betracht.

- API- und Funktionsunterstützung: Unterstützt eine Teilmenge von S3 APIs (z. B. keine Lebenszyklusrichtlinien oder Replikation); möglicherweise sind geringfügige Änderungen an der App für die Sitzungsauthentifizierung oder die Objektverwaltung erforderlich.
- EKS-Integration: Platzieren Sie Ihre EKS pods/nodes in derselben AZ wie der Verzeichnis-Bucket, um die Netzwerklatenz zu minimieren. Verwenden Sie Mountpoint for Amazon S3- oder CSI-Treiber für den nativen Kubernetes-Zugriff.
- Testen: Testen Sie die Latenz beim Abruf in einer Umgebung außerhalb der Produktionsumgebung, um Leistungssteigerungen zu überprüfen. Überwachen Sie die Drosselung in S3-Standardszenarien (z. B. hohe GPU-Sättigung) und vergleichen Sie sie.

Die Speicherklasse S3 Express One Zone ist in mehreren Regionen verfügbar und lässt sich in EKS für Workloads integrieren, die Objektzugriff benötigen, ohne auf Speicherplatz warten zu müssen. Weitere Informationen finden Sie unter Erste Schritte mit S3 Express One Zone.

Beobachtbarkeit

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Überwachung und Beobachtbarkeit

GPU-Metriken erklärt

Die Metrik zur GPU-Auslastung zeigt, ob die GPU während des Beispielfensters irgendwelche Arbeiten ausgeführt hat. Diese Metrik erfasst den Prozentsatz der Zeit, in der die GPU mindestens eine Anweisung ausgeführt hat. Sie gibt jedoch nicht Aufschluss darüber, wie effizient die GPU ihre Hardware genutzt hat. Eine GPU enthält mehrere Streaming-Multiprozessoren (SMs), d. h. parallel Verarbeitungseinheiten, die Befehle ausführen. Eine 100-prozentige Auslastung kann bedeuten, dass die GPU über all ihre Funktionen hinweg umfangreiche parallel Workloads ausgeführt hat SMs, oder

es kann bedeuten, dass ein einziger kleiner Befehl die GPU während des Testzeitraums aktiviert hat. Um die tatsächliche Auslastung zu verstehen, müssen Sie die GPU-Metriken auf mehreren Ebenen der Hardwarearchitektur untersuchen. Jeder Streaming-Multiprozessor besteht aus unterschiedlichen Kerntypen, und jede Schicht weist unterschiedliche Leistungsmerkmale auf. Metriken der obersten Ebene (GPU-Auslastung, Speicherauslastung, GPU-Leistung und GPU-Temperatur, sichtbar über nvidia-smi) zeigen, ob das Gerät aktiv ist. Tiefere Messwerte (SM-Auslastung, SM-Aktivität und Tensorkernnutzung) zeigen, wie effizient die GPU ihre Ressourcen nutzt.

Zielen Sie auf einen hohen GPU-Stromverbrauch ab

Zu wenig genutzte Rechenkapazität wird GPUs verschwendet und die Kosten steigen, da Workloads nicht alle GPU-Komponenten gleichzeitig beanspruchen. Verfolgen Sie für AI/ML Workloads auf Amazon EKS den GPU-Stromverbrauch als Proxy, um die tatsächliche GPU-Aktivität zu ermitteln. Die GPU-Auslastung gibt an, wie viel Zeit die GPU einen beliebigen Kernel ausführt. Es wird jedoch nicht angegeben, ob die Streaming-Multiprozessoren, Speichercontroller und Tensorkerne alle gleichzeitig aktiv sind. Durch den Stromverbrauch wird diese Lücke aufgedeckt, da voll ausgelastete Hardware deutlich mehr Energie verbraucht als Hardware, die schlanke Kernel ausführt oder zwischen Aufgaben ungenutzt bleibt. Vergleichen Sie den Stromverbrauch mit der Thermal Design Power (TDP) der GPU, um eine Unterauslastung zu erkennen, und untersuchen Sie dann, ob Ihre Arbeitslast durch CPU-Vorverarbeitung, Netzwerk-I/O oder ineffiziente Batchgrößen behindert wird.

Richten Sie CloudWatch Container Insights auf Amazon EKS ein, um Pods, Knoten oder Workloads mit geringem GPU-Stromverbrauch zu identifizieren. Dieses Tool lässt sich direkt in Amazon EKS integrieren und ermöglicht es Ihnen, den GPU-Stromverbrauch zu überwachen und Pod-Scheduling oder Instance-Typen anzupassen, wenn der Stromverbrauch unter Ihre Zielwerte fällt. Wenn Sie erweiterte Visualisierungen oder benutzerdefinierte Dashboards benötigen, verwenden Sie den DCGM-Exporter von NVIDIA mit Prometheus und Grafana für die Kubernetes-native Überwachung. Beide Ansätze liefern wichtige NVIDIA-Metriken wie (GPU-Stromverbrauch) und (GPU-Temperatur). `nvidia_smi_power_draw` `nvidia_smi_temperature_gpu` Eine Liste der Metriken finden Sie unter https://docs.aws.amazon.com/AmazonCloudWatch/ latest/monitoring/CloudWatch -Agent-NVIDIA-GPU.htm. Achten Sie auf Muster wie z. B. einen konstant niedrigen Stromverbrauch zu bestimmten Zeiten oder für bestimmte Aufgaben. Anhand dieser Trends können Sie ermitteln, wo Workloads konsolidiert oder die Ressourcenzuweisung angepasst werden müssen.

Statische Ressourcenbeschränkungen in Kubernetes (wie CPU-, Arbeitsspeicher- und GPU-Anzahl) führen häufig zu einer Über- oder Unterauslastung, insbesondere bei dynamischen AI/ML Workloads wie Inferenz, bei denen der Bedarf schwankt. Analysieren Sie Ihre Nutzungstrends und konsolidieren Sie Workloads auf weniger. GPUs Stellen Sie sicher, dass jede GPU voll ausgelastet ist, bevor Sie

weitere GPUs zuweisen. Dieser Ansatz reduziert Verschwendung und senkt die Kosten. Detaillierte Anleitungen zur Optimierung von Planungs- und Sharing-Strategien finden Sie in den Best Practices von EKS Compute and Autoscaling

Beobachtbarkeit und Metriken

Verwenden Sie Überwachungs- und Beobachtungstools für Ihre Workloads AI/ML

Moderne AI/ML Dienste erfordern eine Koordination zwischen Infrastruktur, Modellierung und Anwendungslogik. Plattformingenieure verwalten die Infrastruktur und den Observability-Stack. Sie sammeln, speichern und visualisieren Metriken. AI/ML Ingenieure definieren modellspezifische Metriken und überwachen die Leistung bei variierender Last und Datenverteilung. Anwendungsentwickler nutzen APIs, leiten Anfragen weiter und verfolgen Service-Level-Metriken und Benutzerinteraktionen. Ohne einheitliche Beobachtungspraktiken arbeiten diese Teams isoliert und übersehen wichtige Signale über den Zustand und die Leistung des Systems. Die Schaffung einer gemeinsamen Transparenz in allen Umgebungen stellt sicher, dass alle Beteiligten Probleme frühzeitig erkennen und einen zuverlässigen Service aufrechterhalten können.

Die Optimierung von Amazon EKS-Clustern für AI/ML Workloads stellt die Überwachung vor besondere Herausforderungen, insbesondere im Zusammenhang mit der GPU-Speicherverwaltung. Ohne angemessene Überwachung sehen sich Unternehmen mit out-of-memory (OOM-) Fehlern, Ressourcenineffizienzen und unnötigen Kosten konfrontiert. Eine effektive Überwachung sorgt für eine bessere Leistung, Ausfallsicherheit und geringere Kosten für EKS-Kunden. Verwenden Sie einen ganzheitlichen Ansatz, der drei Überwachungsebenen kombiniert. Überwachen Sie zunächst detaillierte GPU-Metriken mit dem NVIDIA DCGM Exporter, um den GPU-Stromverbrauch, die GPU-Temperatur, die SM-Aktivität, die SM-Belegung und XID-Fehler zu verfolgen. Zweitens sollten Sie Frameworks für Inferenzdienste wie Ray und vLLM überwachen, um anhand ihrer nativen Metriken Einblicke in verteilte Workloads zu gewinnen. Drittens: Sammeln Sie Erkenntnisse auf Anwendungsebene, um benutzerdefinierte Metriken zu verfolgen, die für Ihren Workload spezifisch sind. Dieser mehrschichtige Ansatz bietet Ihnen einen Überblick über die Hardwarenutzung bis hin zur Anwendungsleistung.

Tools und Frameworks

Verschiedene Tools und Frameworks bieten native out-of-the-box Metriken für die Überwachung von AI/ML Workloads. Diese integrierten Metriken machen eine benutzerdefinierte Instrumentierung überflüssig und reduzieren die Einrichtungszeit. Die Metriken konzentrieren sich auf Leistungsaspekte wie Latenz, Durchsatz und Token-Generierung, die für die Bereitstellung von Inferenzen und das

Benchmarking von entscheidender Bedeutung sind. Durch die Verwendung nativer Metriken können Sie sofort mit der Überwachung beginnen, ohne benutzerdefinierte Erfassungspipelines erstellen zu müssen.

- vLLM: Eine Serving-Engine mit hohem Durchsatz für umfangreiche Sprachmodelle (LLMs), die native Metriken wie Anforderungslatenz und Speichernutzung bereitstellt.
- Ray: Ein Framework für verteilte Datenverarbeitung, das Metriken für skalierbare KI-Workloads ausgibt, einschließlich der Ausführungszeiten von Aufgaben und der Ressourcennutzung.
- Hugging Face Text Generation Inference (TGI): Ein Toolkit für die Bereitstellung und Bereitstellung LLMs mit integrierten Metriken für die Inferenzleistung.
- NVIDIA genai-perf: Ein Befehlszeilentool zum Benchmarking generativer KI-Modelle, zur Messung von Durchsatz, Latenz und LLM-spezifischen Metriken, wie z. B. Anfragen, die in bestimmten Zeitintervallen abgeschlossen wurden.

Methoden zur Beobachtbarkeit

Wir empfehlen, alle zusätzlichen Beobachtbarkeitsmechanismen auf eine der folgenden Arten zu implementieren.

CloudWatch Container Insights Wenn Ihr Unternehmen AWS-native Tools mit minimalem Einrichtungsaufwand bevorzugt, empfehlen wir CloudWatch Container Insights. Es ist in den NVIDIA DCGM Exporter integriert, um GPU-Metriken zu sammeln, und bietet vorgefertigte Dashboards für schnelle Einblicke. Container Insights wird durch die Installation des CloudWatch Observability-Add-ons auf Ihrem Cluster aktiviert und implementiert und verwaltet den Lebenszyklus des NVIDIA DCGM Exporter, der GPU-Metriken von den Treibern von Nvidia sammelt und sie für sie bereitstellt. CloudWatch

Nach der Installation von Container Insights erkennt Container Insights CloudWatch automatisch NVIDIA GPUs in Ihrer Umgebung und sammelt wichtige Integritäts- und Leistungskennzahlen. Diese Metriken werden auf kuratierten out-of-the-box Dashboards angezeigt. Sie können Ray und vLLM auch integrieren, indem Sie den Unified CloudWatch Agent CloudWatch verwenden, um ihre systemeigenen Metriken zu senden. Dieser einheitliche Ansatz vereinfacht die Beobachtbarkeit in EKS-Umgebungen und ermöglicht es Teams, sich auf die Leistungsoptimierung und Kostenoptimierung zu konzentrieren, anstatt eine Überwachungsinfrastruktur aufzubauen.

Eine vollständige Liste der verfügbaren Metriken finden Sie unter Amazon EKS- und Kubernetes Container Insights-Metriken. step-by-stepAnleitungen zur Implementierung der GPU-Überwachung

finden Sie unter Gain operational insights for NVIDIA GPU-Workloads using Amazon CloudWatch Container Insights. Praktische Beispiele für die Optimierung der Inferenzlatenz finden Sie unter Optimizing AI Responsiveness: A practical guide to Amazon Bedrock latency-optimized inference.

Managed Prometheus und Grafana Wenn Ihr Unternehmen maßgeschneiderte Dashboards und erweiterte Visualisierungsfunktionen benötigt, setzen Sie Prometheus mit dem NVIDIA DCGM-Exporter und Grafana für Kubernetes-natives Monitoring ein. Prometheus scannt und speichert GPU-Metriken aus dem DCGM-Exporter, während Grafana flexible Visualisierungs- und Warnfunktionen bietet. Dieser Ansatz bietet Ihnen im Vergleich zu Container Insights mehr Kontrolle über das Dashboard-Design und die Aufbewahrung von Kennzahlen. CloudWatch

Sie können diesen Monitoring-Stack erweitern, indem Sie Open-Source-Frameworks wie Ray und vLLM Ray und vLLM integrieren, um ihre nativen Metriken nach Prometheus zu exportieren. Sie können Grafana auch mit einer AWS X-Ray X-Ray-Datenquelle verbinden, um verteilte Traces zu visualisieren und Leistungsengpässe in Ihrer Inferenzpipeline zu identifizieren. Diese Kombination bietet end-to-end Transparenz von Metriken auf GPU-Ebene bis hin zu Anforderungsabläufen auf Anwendungsebene.

step-by-stepAnleitungen zur Bereitstellung dieses Monitoring-Stacks finden Sie unter Überwachen von GPU-Workloads auf Amazon EKS mithilfe von AWS-verwalteten Open-Source-Services.

Erwägen Sie die Überwachung von Kernschulungen und die Feinabstimmung von Kennzahlen

Überwachen Sie die wichtigsten Trainingsmetriken, um den Zustand und die Leistung Ihres Amazon EKS-Clusters und der darauf ausgeführten Machine-Learning-Workloads zu verfolgen. Trainingsworkloads haben andere Überwachungsanforderungen als Inferenz-Workloads, da sie über längere Zeiträume laufen, Ressourcen unterschiedlich beanspruchen und Einblicke in die Modellkonvergenz und die Effizienz der Datenpipeline erfordern. Die folgenden Kennzahlen helfen Ihnen dabei, Engpässe zu identifizieren, die Ressourcenzuweisung zu optimieren und sicherzustellen, dass Schulungsaufgaben erfolgreich abgeschlossen werden. step-by-stepAnleitungen zur Implementierung dieses Überwachungsansatzes finden Sie unter Einführung in die Beobachtung von Workloads für maschinelles Lernen auf Amazon EKS.

Metriken zur Ressourcennutzung

Überwachen Sie die Kennzahlen zur Ressourcennutzung, um zu überprüfen, ob Ihre Ressourcen ordnungsgemäß genutzt werden. Diese Kennzahlen helfen Ihnen dabei, Engpässe und die Hauptursachen für Leistungsprobleme zu identifizieren.

- CPU, Arbeitsspeicher, Netzwerk, GPU-Leistung und GPU-Temperatur — Überwachen Sie diese Messwerte, um sicherzustellen, dass die zugewiesenen Ressourcen den Workload-Anforderungen entsprechen, und um Optimierungsmöglichkeiten zu identifizieren. Verfolgen Sie Metriken wie gpu_memory_usage_bytes, um Speicherverbrauchsmuster zu identifizieren und Spitzenauslastungen zu erkennen. Berechne Perzentile wie das 95. Perzentil (P95), um die höchsten Speicheranforderungen während des Trainings zu ermitteln. Diese Analyse hilft Ihnen, Modelle und Infrastruktur zu optimieren, um OOM-Fehler zu vermeiden und Kosten zu senken.
- SM-Belegung, SM-Aktivität, FPxx Aktivität — Überwachen Sie diese Kennzahlen, um zu verstehen, wie die zugrunde liegende Ressource auf der GPU genutzt wird. Als Faustregel gilt das Ziel 0,8 für SM-Aktivität.
- Nutzung von Knoten- und Pod-Ressourcen — Verfolgen Sie die Ressourcennutzung auf Knoten- und Pod-Ebene, um Ressourcenkonflikte und potenzielle Engpässe zu identifizieren. Überwachen Sie, ob Knoten an Kapazitätsgrenzen stoßen, was die Pod-Planung verzögern und Trainingsaufgaben verlangsamen kann.
- Ressourcennutzung im Vergleich zu Anfragen und Grenzwerten — Vergleichen Sie die tatsächliche Ressourcennutzung mit konfigurierten Anfragen und Grenzwerten, um festzustellen, ob Ihr Cluster aktuelle und future Workloads bewältigen kann. Dieser Vergleich zeigt, ob Sie die Ressourcenzuweisungen anpassen müssen, um OOM-Fehler oder Ressourcenverschwendung zu vermeiden.
- Interne Kennzahlen aus ML-Frameworks — Erfassen Sie interne Schulungs- und Konvergenzkennzahlen aus ML-Frameworks wie TensorFlow und PyTorch. Zu diesen Kennzahlen gehören Verlustkurven, Lernrate, Batch-Verarbeitungszeit und Dauer der Trainingsschritte. Visualisieren Sie diese Metriken mit TensorBoard oder ähnlichen Tools, um die Modellkonvergenz zu verfolgen und Ineffizienzen im Training zu identifizieren.

Modellieren Sie Leistungskennzahlen

Überwachen Sie die Modellleistungskennzahlen, um zu überprüfen, ob Ihr Trainingsprozess Modelle hervorbringt, die den Genauigkeits- und Geschäftsanforderungen entsprechen. Anhand dieser Metriken können Sie feststellen, wann das Training beendet werden muss, Modellversionen vergleichen und Leistungseinbußen feststellen.

- Genauigkeit, Präzision, Rückruf und F1-Score — Verfolgen Sie diese Metriken, um zu verstehen, wie gut Ihr Modell bei Validierungsdaten abschneidet. Berechnen Sie den F1-Score eines Validierungssatzes nach jeder Trainingsepoche, um zu beurteilen, ob sich das Modell verbessert und wann es akzeptable Leistungsniveaus erreicht.

- Geschäftsspezifische Kennzahlen und KPIs — Definieren und verfolgen Sie Kennzahlen, mit denen der Geschäftswert Ihrer Initiativen direkt gemessen wird. AI/ML Für ein Empfehlungssystem sollten Sie Kennzahlen wie die Klickrate oder die Konversionsrate verfolgen, um sicherzustellen, dass das Modell die angestrebten Geschäftsergebnisse fördert.
- Leistung im Zeitverlauf — Vergleichen Sie Leistungskennzahlen verschiedener Modellversionen und Trainingsläufe, um Trends zu erkennen und Leistungseinbußen zu erkennen. Verfolgen Sie, ob neuere Modellversionen die Leistung im Vergleich zu Basismodellen beibehalten oder verbessern. Dieser historische Vergleich hilft Ihnen bei der Entscheidung, ob Sie neue Modelle einsetzen oder Schulungsprobleme untersuchen möchten.

Datenqualität und Drift-Metriken

Überwachen Sie die Datenqualität und die Drift-Metriken, um sicherzustellen, dass Ihre Trainingsdaten konsistent und repräsentativ bleiben. Datendrift kann dazu führen, dass sich die Modellleistung im Laufe der Zeit verschlechtert, während Datenqualitätsprobleme verhindern können, dass Modelle konvergieren oder zu unzuverlässigen Ergebnissen führen.

- Statistische Eigenschaften von Eingabedaten — Verfolgen Sie statistische Eigenschaften wie Mittelwert, Standardabweichung und Verteilung der Eingabemerkmale im Laufe der Zeit, um Datenabweichungen oder Anomalien zu erkennen. Überwachen Sie, ob sich die Merkmalsverteilungen signifikant von Ihren Basis-Trainingsdaten unterscheiden. Wenn sich beispielsweise der Mittelwert eines kritischen Merkmals um mehr als zwei Standardabweichungen ändert, untersuchen Sie, ob sich Ihre Datenpipeline geändert hat oder ob sich die zugrunde liegende Datenquelle verschoben hat.
- Erkennung von Datenabweichungen und Warnmeldungen — Implementieren Sie automatisierte Mechanismen, um Datenqualitätsprobleme zu erkennen und darauf hinzuweisen, bevor sie sich auf das Training auswirken. Verwende statistische Tests wie den Kolmogorov-Smirnov-Test oder den Chi-Quadrat-Test, um aktuelle Datenverteilungen mit deinen ursprünglichen Trainingsdaten zu vergleichen. Richten Sie Warnmeldungen ein, wenn Tests signifikante Abweichungen feststellen, sodass Sie Modelle mit aktualisierten Daten erneut trainieren oder Probleme mit der Datenpipeline untersuchen können.

Metriken für Latenz und Durchsatz

Überwachen Sie die Latenz- und Durchsatzkennzahlen, um Engpässe in Ihrer Trainingspipeline zu identifizieren und die Ressourcennutzung zu optimieren. Diese Kennzahlen helfen Ihnen zu

verstehen, wo während der Schulung Zeit aufgewendet wird und worauf Sie sich bei der Optimierung konzentrieren sollten.

- End-to-End Latenz von ML-Trainingspipelines — Messen Sie die Gesamtzeit, in der Daten Ihre gesamte Trainingspipeline durchlaufen, von der Datenaufnahme bis zur Modellaktualisierung. Verfolgen Sie diese Metrik über alle Trainingsläufe hinweg, um festzustellen, ob Änderungen an der Pipeline die Leistung verbessern oder verschlechtern. Eine hohe Latenz weist häufig auf Engpässe beim Laden von Daten, bei der Vorverarbeitung oder bei der Netzwerkkommunikation zwischen Knoten hin.
- Trainingsdurchsatz und Verarbeitungsrate — Verfolgen Sie die Datenmenge, die Ihre Trainingspipeline pro Zeiteinheit verarbeitet, um eine effiziente Nutzung der Ressourcen sicherzustellen. Überwachen Sie Messwerte wie z. B. pro Sekunde verarbeitete Proben oder abgeschlossene Chargen pro Minute. Ein niedriger Durchsatz im Verhältnis zu Ihrer Hardwarekapazität deutet auf Ineffizienzen beim Laden, bei der Vorverarbeitung oder bei der Modellberechnung hin, wodurch GPU-Zyklen verschwendet werden.
- Latenz beim Speichern und Wiederherstellen von Checkpoints — Überwachen Sie die Zeit, die erforderlich ist, um Modell-Checkpoints im Speicher (S3, EFS FSx) zu speichern und sie im GPU- oder CPU-Speicher wiederherzustellen, wenn Jobs wieder aufgenommen oder nach Ausfällen wiederhergestellt werden. Langsame Checkpoint-Operationen verlängern die Wiederherstellungszeit von Jobs und erhöhen die Kosten. Verfolgen Sie die Größe der Checkpoints, die Speicherdauer, die Wiederherstellungsdauer und die Anzahl der Fehler, um Optimierungsmöglichkeiten wie Komprimierung oder schnellere Speicherstufen zu identifizieren.
- Zeit zum Laden und Vorverarbeiten von Daten — Messen Sie die Zeit, die für das Laden von Daten aus dem Speicher und das Anwenden von Transformationen zur Vorverarbeitung aufgewendet wurde. Vergleichen Sie diese Zeit mit der Berechnungszeit des Modells, um festzustellen, ob Ihr Training daten- oder rechengebunden ist. Wenn das Laden von Daten mehr als 20% der gesamten Trainingszeit in Anspruch nimmt, sollten Sie erwägen, Ihre Datenpipeline durch Zwischenspeichern, Prefetching oder schnelleren Speicher zu optimieren.

Fehlerraten und Ausfälle

Überwachen Sie die Fehlerraten und Ausfälle in Ihrer gesamten Schulungspipeline, um die Zuverlässigkeit aufrechtzuerhalten und eine Verschwendung von Rechenressourcen zu verhindern. Unentdeckte Fehler können dazu führen, dass Trainingsaufgaben unbemerkt fehlschlagen, ungültige Modelle produzieren oder Stunden an GPU-Zeit verschwenden, bevor Sie Probleme bemerken.

- Überwachung von Pipeline-Fehlern — Verfolgen Sie Fehler in allen Phasen Ihrer ML-Pipeline, einschließlich Datenvorverarbeitung, Modelltraining und Checkpoint-Operationen. Protokollieren Sie Fehlertypen, -häufigkeiten und betroffene Komponenten, um Probleme schnell zu identifizieren. Zu den häufigsten Fehlern gehören nicht übereinstimmende Datenformate, out-of-memory Fehler bei der Vorverarbeitung und Fehler beim Speichern von Checkpoints aufgrund von Speicherbeschränkungen. Richten Sie Warnmeldungen ein, wenn die Fehlerraten die Ausgangsschwellenwerte überschreiten, damit Sie diese untersuchen können, bevor sich Fehler häufen.
- Analyse wiederkehrender Fehler — Identifizieren und untersuchen Sie Muster bei wiederkehrenden Fehlern, um future Ausfälle zu verhindern und die Zuverlässigkeit der Pipeline zu verbessern. Analysieren Sie Protokolle, um herauszufinden, ob bestimmte Datenproben, Chargengrößen oder Trainingskonfigurationen durchweg zu Ausfällen führen. Wenn beispielsweise bestimmte Eingabedatentypen Fehler bei der Vorverarbeitung auslösen, fügen Sie früher in der Pipeline Validierungsprüfungen hinzu oder aktualisieren Sie Ihre Datenbereinigungslogik. Verfolgen Sie die mittlere Betriebsdauer zwischen Ausfällen (MTBF), um zu messen, ob sich die Zuverlässigkeit Ihrer Pipeline im Laufe der Zeit verbessert. “

Spezifische Kennzahlen für Kubernetes und EKS

Überwachen Sie Kubernetes- und EKS-Metriken, um sicherzustellen, dass Ihre Cluster-Infrastruktur funktionsfähig bleibt und Ihre Trainingsworkloads unterstützen kann. Diese Metriken helfen Ihnen dabei, Infrastrukturprobleme zu erkennen, bevor sie zu Fehlschlägen oder Leistungseinbußen bei der Schulung führen.

- Kubernetes Cluster State Metrics — Überwachen Sie den Zustand und den Status von Kubernetes-Objekten, einschließlich Pods, Knoten, Bereitstellungen und Diensten. Verfolgen Sie den Pod-Status, um Pods zu identifizieren, die im Status „Ausstehend“, „Fehlgeschlagen“ oder „Absturzschleife“ hängen Überwachen Sie den Zustand der Knoten, um Probleme wie Festplattenüberlastung, Speicherauslastung oder Netzwerkausfälle zu erkennen. Verwenden Sie kubectl oder Überwachungstools, um diese Messwerte kontinuierlich zu überprüfen und Warnmeldungen einzurichten, wenn Pods nicht gestartet werden können oder Knoten nicht mehr planbar sind.
- Kennzahlen zur Ausführung von Trainings-Pipelines — Verfolgen Sie erfolgreiche und fehlgeschlagene Pipeline-Läufe, Job-Dauern, Bearbeitungszeiten von Schritten und Orchestrierungsfehler. Überwachen Sie, ob Trainingsaufgaben innerhalb der erwarteten Zeitfenster abgeschlossen werden und ob die Ausfallraten im Laufe der Zeit zunehmen. Verfolgen Sie Kennzahlen wie die Erfolgsquote von Aufträgen, die durchschnittliche Jobdauer und die Zeit bis

zum Scheitern. Anhand dieser Kennzahlen können Sie feststellen, ob Infrastrukturprobleme, Konfigurationsprobleme oder Datenqualitätsprobleme zu Trainingsausfällen führen.

- AWS-Servicemetriken — Verfolgen Sie Metriken für AWS-Services, die Ihre EKS-Infrastruktur und Ihre Schulungs-Workloads unterstützen. Überwachen Sie S3-Metriken wie Anforderungslatenz, Fehlerraten und Durchsatz, um sicherzustellen, dass die Leistung beim Laden von Daten konsistent bleibt. Verfolgen Sie EBS-Volumenmetriken wie IOPS, Durchsatz und Warteschlangenlänge, um Speicherengpässe zu erkennen. Überwachen Sie VPC-Flussprotokolle und Netzwerkmetriken, um Verbindungsprobleme zwischen Knoten oder externen Diensten zu identifizieren.
- Kubernetes Control Plane Metrics — Überwachen Sie den API-Server, den Scheduler, den Controller-Manager und die etcd-Datenbank, um Leistungsprobleme oder Ausfälle zu erkennen, die sich auf den Clusterbetrieb auswirken. Verfolgen Sie die Latenz, die Anforderungsrate und die Fehlerrate von API-Serveranfragen, um sicherzustellen, dass die Kontrollebene schnell auf Planungsanfragen reagiert. Überwachen Sie die Größe der etcd-Datenbank, die Commit-Dauer und die Änderungen am Leader, um Stabilitätsprobleme zu erkennen. Eine hohe API-Serverlatenz oder häufige Änderungen des ETCD-Leiters können die Pod-Planung verzögern und die Startzeiten von Trainingsaufträgen verlängern.

Anwendungs- und Instanzprotokolle

Sammeln und analysieren Sie Anwendungs- und Instanzprotokolle, um Probleme zu diagnostizieren, die mit Metriken allein nicht erklärt werden können. Protokolle bieten detaillierten Kontext zu Fehlern, Statusänderungen und Systemereignissen, sodass Sie nachvollziehen können, warum Schulungsaufgaben fehlschlagen oder schlecht abschneiden. Durch die Korrelation von Protokollen mit Metriken können Sie die Hauptursachen schneller ermitteln.

- Anwendungsprotokolle — Sammeln Sie Anwendungsprotokolle von Ihren Trainingsaufträgen, Daten-Pipelines und ML-Frameworks, um Engpässe zu identifizieren und Fehler zu diagnostizieren. In diesen Protokollen werden detaillierte Informationen zur Auftragsausführung erfasst, darunter Fehler beim Laden von Daten, Fehler bei der Modellinitialisierung, Fehler beim Speichern von Checkpoints und Framework-spezifische Warnungen. Korrelieren Sie Protokollzeitstempel mit Messspitzen, um zu verstehen, was zu Leistungseinbußen oder Ausfällen geführt hat. Wenn beispielsweise die GPU-Auslastung plötzlich abnimmt, überprüfen Sie die Anwendungsprotokolle auf Fehler, die auf Daten-Pipeline-Stopps oder Ausfälle bei der Vorverarbeitung hinweisen. Verwenden Sie zentralisierte Protokollierungstools wie CloudWatch Logs oder Fluent Bit, um Protokolle aus allen Pods zu aggregieren und sie durchsuchbar zu machen.

- Instanzprotokolle — Sammeln Sie Protokolle auf Instanzebene wie Systemjournalprotokolle und dmesg-Ausgaben, um Hardwareprobleme und Probleme auf Kernelebene zu erkennen. Diese Protokolle decken Probleme wie GPU-Treiberfehler, Fehler bei der Speicherzuweisung, I/O Festplattenfehler und Probleme mit der Netzwerkschnittstelle auf, die möglicherweise nicht in Anwendungsprotokollen auftauchen. Korrelieren Sie Instanzprotokolle mit Anwendungsprotokollen und Messdaten, um festzustellen, ob Trainingsfehler auf Hardware- oder Anwendungsprobleme zurückzuführen sind. Wenn beispielsweise ein Trainingsjob mit einem out-of-memory Fehler fehlschlägt, suchen Sie in den dmesg-Protokollen nach Kernel-OOM-Killermeldungen, die angeben, ob dem System der Arbeitsspeicher ausgegangen ist oder ob die Anwendung ihre Container-Limits überschritten hat. Richten Sie Warnmeldungen für kritische Hardwarefehler wie GPU-XID-Fehler oder Festplattenausfälle ein, sodass Sie ausfallende Instanzen ersetzen können, bevor sie zu weitreichenden Trainingsunterbrechungen führen.

In den folgenden Abschnitten wird gezeigt, wie die oben beschriebenen Metriken mithilfe von zwei von AWS empfohlenen Ansätzen erfasst werden: CloudWatch Container Insights und Amazon Managed Prometheus Amazon Managed Prometheus with Amazon Managed Grafana. Wählen Sie CloudWatch Container Insights, wenn Sie AWS-native Tools mit minimalem Einrichtungsaufwand und vorgefertigten Dashboards bevorzugen. Entscheiden Sie sich für Amazon Managed Prometheus mit Amazon Managed Grafana, wenn Sie maßgeschneiderte Dashboards und erweiterte Visualisierungsfunktionen benötigen oder eine Integration in eine bestehende Prometheus basierte Überwachungsinfrastruktur wünschen. Eine vollständige Liste der verfügbaren Container Insights-Metriken finden Sie unter Amazon EKS- und Kubernetes Container Insights-Metriken.

Erwägen Sie die Überwachung von Online-Inferenzmetriken in Echtzeit

In Echtzeitsystemen ist eine geringe Latenz entscheidend, um Benutzern oder anderen abhängigen Systemen zeitnahe Antworten zu geben. Eine hohe Latenz kann die Benutzererfahrung beeinträchtigen oder Leistungsanforderungen verletzen. Zu den Komponenten, die die Inferenzlatenz beeinflussen, gehören die Ladezeit des Modells, die Vorverarbeitungszeit, die tatsächliche Prognosezeit, die Nachverarbeitungszeit und die Netzwerkübertragungszeit. Wir empfehlen, die Inferenzlatenz zu überwachen, um sicherzustellen, dass Antworten mit niedriger Latenz und Service Level Agreements (SLAs) eingehalten werden, und benutzerdefinierte Metriken für Folgendes zu entwickeln. Testen Sie unter erwarteter Last, berücksichtigen Sie die Netzwerklatenz, berücksichtigen Sie gleichzeitige Anfragen und testen Sie mit unterschiedlichen Batchgrößen.

- Time to First Token (TTFT) — Zeitspanne zwischen dem Senden einer Anfrage durch einen Benutzer und dem Erhalt der ersten Antwort (erstes Wort, Token oder Abschnitt). In Chatbots

würden Sie beispielsweise überprüfen, wie lange es dauert, bis die erste Ausgabe (Token) generiert ist, nachdem der Benutzer eine Frage gestellt hat.

- End-to-End Latenz — Dies ist die Gesamtzeit vom Eingang einer Anfrage bis zum Zurücksenden der Antwort. Messen Sie beispielsweise die Zeit von der Anfrage bis zur Antwort.
- Output Tokens Per Second (TPS) — Gibt an, wie schnell Ihr Modell neue Token generiert, nachdem es reagiert hat. In Chatbots würden Sie beispielsweise die Generierungsgeschwindigkeit für Sprachmodelle für einen Basistext verfolgen.
- Fehlerrate — Verfolgt fehlgeschlagene Anfragen, die auf Leistungsprobleme hinweisen können. Überwachen Sie beispielsweise fehlgeschlagene Anfragen für große Dokumente oder bestimmte Zeichen.
- Durchsatz — Messen Sie die Anzahl der Anfragen oder Operationen, die das System pro Zeiteinheit verarbeiten kann. Verfolgen Sie beispielsweise Anfragen pro Sekunde, um Spitzenlasten zu bewältigen.

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/Voder eine Sitzung (Cache verwenden):

- hit/miss Cache-Verhältnis — Messen Sie bei Inferenz-Setups, die Caching nutzen (K/V oder eingebettete Caches), wie oft der Cache hilfreich ist. Niedrige Trefferquoten können auf suboptimale Änderungen der Cache-Konfiguration oder der Arbeitslast hindeuten, die beide die Latenz erhöhen können.

In den nachfolgenden Themen wird das Sammeln von Daten für einige der oben genannten Metriken demonstriert. Wir werden Beispiele mit den beiden von AWS empfohlenen Ansätzen bereitstellen: AWS-native CloudWatch Container Insights und Amazon Managed Prometheus als Open-Source-Lösung mit Amazon Managed Grafana. Sie würden sich je nach Ihren allgemeinen Anforderungen an die Beobachtbarkeit für eine dieser Lösungen entscheiden. Eine vollständige Liste der Container Insights-Metriken finden Sie unter Amazon EKS- und Kubernetes Container Insights-Metriken.

Nachverfolgung der GPU-Speichernutzung

Wie im the section called “Erwägen Sie die Überwachung von Kernschulungen und die Feinabstimmung von Kennzahlen” Thema erörtert, ist die GPU-Speichernutzung unerlässlich, um

out-of-memory (OOM-) Fehler zu vermeiden und eine effiziente Ressourcennutzung sicherzustellen. Die folgenden Beispiele zeigen, wie Sie Ihre Trainingsanwendung so instrumentieren, dass eine benutzerdefinierte Histogrammmetrik angezeigt wird`gpu_memory_usage_bytes`, und wie die P95-Speicherauslastung berechnet wird, um den Spitzenverbrauch zu ermitteln. Stellen Sie sicher, dass Sie den Test anhand eines Beispiel-Trainingsjobs (z. B. der Feinabstimmung eines Transformatormodells) in einer Staging-Umgebung durchführen.

Beispiel für AWS-native CloudWatch Container Insights

Dieses Beispiel zeigt, wie Sie Ihre Trainingsanwendung mithilfe des AWS-nativen Ansatzes `gpu_memory_usage_bytes` so instrumentieren, dass sie als Histogramm angezeigt wird. Beachten Sie, dass Ihr AI/ML Container so konfiguriert sein muss, dass er strukturierte Logs im EMF-Format (CloudWatch Embedded Metrics Format) ausgibt. CloudWatch logs analysiert EMF und veröffentlicht die Metriken. Verwenden Sie aws_embedded_metrics in Ihrer Trainingsanwendung, um strukturierte Logs im EMF-Format an Logs zu senden, das GPU-Metriken extrahiert. 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()
```

Beispiel Prometheus und Grafana

Dieses Beispiel zeigt, wie Sie Ihre Trainingsanwendung mithilfe der Prometheus-Clientbibliothek in Python `gpu_memory_usage_bytes` so instrumentieren, dass sie als Histogramm angezeigt wird.

```
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()
```

Verfolgen Sie die Dauer von Inferenzanfragen für Online-Inferenzen in Echtzeit

Wie bereits im the section called “Erwägen Sie die Überwachung von Kernschulungen und die Feinabstimmung von Kennzahlen” Thema erörtert, ist eine geringe Latenz entscheidend,

um Benutzern oder anderen abhängigen Systemen zeitnahe Antworten zu bieten. Die folgenden Beispiele zeigen, wie Sie eine benutzerdefinierte Histogramm-Metrik verfolgen können`inference_request_duration_seconds`, die von Ihrer Inferenzanwendung bereitgestellt wird. Berechnen Sie die Latenz im 95. Perzentil (P95), um sich auf Worst-Case-Szenarien zu konzentrieren, testen Sie mit synthetischen Inferenzanforderungen (z. B. über Locust) in einer Staging-Umgebung und legen Sie Warnschwellenwerte (z. B. > 500 ms) fest, um SLA-Verstöße zu erkennen.

Beispiel für AWS-native CloudWatch Container Insights

Dieses Beispiel zeigt, wie Sie eine benutzerdefinierte Histogramm-Metrik in Ihrer Inferenzanwendung für inference_request_duration_seconds mithilfe des AWS Embedded Metric Format erstellen. 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()
```

Beispiel Prometheus und Grafana

Dieses Beispiel zeigt, wie Sie eine benutzerdefinierte Histogrammmetrik in Ihrer Inferenzanwendung für inference_request_duration_seconds mithilfe der Prometheus-Clientbibliothek in Python erstellen:

```
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)
```

Verwenden Sie in Grafana die Abfrage, um P95-Latenztrends `histogram_quantile(0.95, sum(rate(inference_request_duration_seconds_bucket[5m])) by (le, pod))` zu visualisieren. Weitere Informationen finden Sie unter Prometheus-Histogramm-Dokumentation und Prometheus-Client-Dokumentation.

Verfolgen Sie den Token-Durchsatz für Online-Inferenz in Echtzeit

Wie im the section called “Erwägen Sie die Überwachung von Kernschulungen und die Feinabstimmung von Kennzahlen” Thema besprochen, empfehlen wir, die Token-Verarbeitungszeit zu überwachen, um die Modellleistung zu messen und Skalierungsentscheidungen zu optimieren. Die folgenden Beispiele zeigen, wie Sie eine benutzerdefinierte Histogramm-Metrik verfolgen können`token_processing_duration_seconds`, die von Ihrer Inferenzanwendung bereitgestellt wird. Berechnen Sie die Dauer des 95. Perzentils (P95), um die Verarbeitungseffizienz zu analysieren, Tests mit simulierten Anforderungslasten (z. B. 100 bis 1000 Anforderungen/Sekunde) in einem Cluster außerhalb der Produktion durchzuführen und KEDA-Trigger anzupassen, um die Skalierung zu optimieren.

Beispiel für AWS-native CloudWatch Container Insights

Dieses Beispiel zeigt, wie Sie eine benutzerdefinierte Histogramm-Metrik in Ihrer Inferenzanwendung für token_processing_duration_seconds mithilfe des AWS Embedded Metric Format erstellen.

CloudWatch Es verwendet die) with a custom `get_duration_bucket Funktion dimensions (`set_dimension), um die Dauer in Bereiche einzuteilen (z. B. „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
```

Beispiel Prometheus und Grafana

Dieses Beispiel zeigt, wie Sie eine benutzerdefinierte Histogrammmetrik in Ihrer Inferenzanwendung für token_processing_duration_seconds mithilfe der Prometheus-Clientbibliothek in Python erstellen.

```
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)
```

Verwenden Sie in Grafana die Abfrage, `histogram_quantile(0.95, sum(rate(token_processing_duration_seconds_bucket[5m])) by (le, pod))` um Trends der P95-Verarbeitungszeit zu visualisieren. Weitere Informationen finden Sie unter Prometheus-Histogramm-Dokumentation und Prometheus-Client-Dokumentation.

Messen Sie die Latenz bei der Checkpoint-Wiederherstellung

Wie im the section called “Erwägen Sie die Überwachung von Kernschulungen und die Feinabstimmung von Kennzahlen” Thema erörtert, ist die Checkpoint-Latenz eine wichtige Kennzahl in mehreren Phasen des Modelllebenszyklus. Die folgenden Beispiele zeigen, wie Sie eine benutzerdefinierte Histogramm-Metrik verfolgen können`checkpoint_restore_duration_seconds`, die von Ihrer Anwendung bereitgestellt wird. Berechnen Sie die Dauer des 95. Perzentils (P95), um die Wiederherstellungsleistung zu überwachen, Tests mit Spot-Unterbrechungen in einem Cluster außerhalb der Produktion durchzuführen und Schwellenwerte für Warnmeldungen festzulegen (z. B. <30 Sekunden), um Verzögerungen zu erkennen.

Beispiel für AWS-native CloudWatch Container Insights

Dieses Beispiel zeigt, wie Sie Ihre Batch-Anwendung instrumentieren, um checkpoint_restore_duration_seconds mithilfe von Insights als Histogramm verfügbar zu machen: 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")
```

Beispiel Prometheus und Grafana

Dieses Beispiel zeigt, wie Sie Ihre Batch-Anwendung mithilfe der Prometheus-Clientbibliothek in Python `checkpoint_restore_duration_seconds` so instrumentieren, dass sie als Histogramm angezeigt wird:

```
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"))
```

Verwenden Sie in Grafana die Abfrage, `histogram_quantile(0.95, sum(rate(checkpoint_restore_duration_seconds_bucket[5m]) by (le))` um

Latenztrends bei der P95-Wiederherstellung zu visualisieren. Weitere Informationen finden Sie unter Prometheus-Histogramm-Dokumentation und Prometheus-Client-Dokumentation.

Skalierung und Leistung von Anwendungen

> **Tip**
>
> Informieren Sie sich in Amazon EKS-Workshops über bewährte Verfahren.

Verwaltung von ML-Artefakten, Bereitstellung von Frameworks und Startup-Optimierung

Die Bereitstellung von Modellen für maschinelles Lernen (ML) auf Amazon EKS erfordert eine sorgfältige Überlegung, wie Modelle in Container-Images und Laufzeitumgebungen integriert werden. Dies gewährleistet Skalierbarkeit, Reproduzierbarkeit und effiziente Ressourcennutzung. In diesem Thema werden die verschiedenen Ansätze zum Umgang mit ML-Modellartefakten, zur Auswahl von Serving-Frameworks und zur Optimierung der Startzeiten von Containern mithilfe von Techniken wie Pre-Caching beschrieben, die allesamt darauf zugeschnitten sind, die Startzeiten von Containern zu verkürzen.

Umgang mit ML-Modellartefakten in Bereitstellungen

Eine wichtige Entscheidung ist, wie mit den ML-Modellartefakten (wie Gewichten und Konfigurationen) selbst umgegangen werden soll. Die Wahl wirkt sich auf die Image-Größe, die Bereitstellungsgeschwindigkeit, die Häufigkeit der Modellaktualisierungen und den betrieblichen Aufwand aus. Beachten Sie, dass wir uns beim Speichern des „Modells“ auf die Modellartefakte (wie trainierte Parameter und Modellgewichte) beziehen. Es gibt verschiedene Ansätze für den Umgang mit ML-Modellartefakten in Amazon EKS. Jeder hat seine Kompromisse, und der beste hängt von der Größe Ihres Modells, der Aktualisierungsfrequenz und den Infrastrukturanforderungen ab. Ziehen Sie die folgenden Ansätze in Betracht, von den am wenigsten empfohlenen bis hin zu den am meisten empfohlenen:

- Integrieren des Modells in das Container-Image: Kopieren Sie die Modelldateien (z. B. .safetensors, .pth, .h5) während der Image-Erstellung in das Container-Image (z. B. Dockerfile). Das Modell ist Teil des unveränderlichen Images. Wir empfehlen, diesen Ansatz für kleinere Modelle mit seltenen Updates zu verwenden. Dies gewährleistet Konsistenz und Reproduzierbarkeit, sorgt für keine Ladeverzögerung und vereinfacht das

Abhängigkeitsmanagement, führt jedoch zu größeren Bildgrößen, verlangsamten Builds und Pushs, erfordert Neuaufbau und erneutes Deployment für Modellaktualisierungen und ist aufgrund des Registry-Pull-Durchsatzes nicht ideal für große Modelle.

- Herunterladen des Modells zur Laufzeit: Beim Start des Containers lädt die Anwendung das Modell über Skripts in einem Init-Container oder Einstiegspunkt von einem externen Speicher herunter (z. B. Amazon S3, unterstützt von S3 CRT für optimierte Übertragungen mit hohem Durchsatz mithilfe von Methoden wie Mountpoint for S3 CSI Driver, AWS S3 CLI oder s5cmd OSS CLI). Für große Modelle mit häufigen Updates empfehlen wir, mit diesem Ansatz zu beginnen. Dadurch bleibt der Fokus der Container-Images auf Code/Laufzeit gerichtet, es werden einfache Modellaktualisierungen ohne Neuerstellungen ermöglicht, die Versionierung über Speichermetadaten unterstützt, aber es führt zu potenziellen Netzwerkausfällen (erfordert Wiederholungslogik) und erfordert Authentifizierung und Caching.

Weitere Informationen finden Sie unter Beschleunigter Pull-Prozess im Workshop KI on EKS.

Bereitstellung von ML-Modellen

Die Bereitstellung und Bereitstellung von Modellen für maschinelles Lernen (ML) auf Amazon EKS erfordert die Auswahl eines geeigneten Model-Serving-Ansatzes, um Latenz, Durchsatz, Skalierbarkeit und einfache Bedienung zu optimieren. Die Wahl hängt von Ihrem Modelltyp (z. B. Sprache, Visionsmodell), den Arbeitslastanforderungen (z. B. Echtzeit-Inferenz) und der Expertise Ihres Teams ab. Zu den gängigen Ansätzen gehören Python-basierte Setups für das Prototyping, dedizierte Modellserver für Funktionen in Produktionsqualität und spezialisierte Inferenz-Engines für hohe Leistung und Effizienz. Jede Methode beinhaltet Kompromisse bei der Komplexität der Einrichtung, der Leistung und der Ressourcennutzung. Beachten Sie, dass die Bereitstellung von Frameworks die Größe von Container-Images aufgrund von Abhängigkeiten (um ein Vielfaches GBs) erhöhen kann, was sich möglicherweise auf die Startzeiten auswirkt. Ziehen Sie eine Entkopplung mithilfe von Techniken zur Behandlung von Artefakten in Betracht, um dieses Problem zu minimieren. Die Optionen sind von den am wenigsten empfohlenen bis hin zu den am meisten empfohlenen Optionen aufgeführt:

Verwenden von Python-Frameworks (z. B. FastAPI, HuggingFace Transformers with PyTorch) Entwickeln Sie eine benutzerdefinierte Anwendung mithilfe von Python-Frameworks und betten Sie Modelldateien (Gewichte, Konfiguration, Tokenizer) in ein containerisiertes Knoten-Setup ein.

- Vorteile: Einfaches Prototyping, nur Python ohne zusätzliche Infrastruktur, kompatibel mit allen HuggingFace Modellen, einfache Kubernetes-Bereitstellung.

- Nachteile: Beschränkt auf request/simple Einzelstapelverarbeitung, langsame Token-Generierung (keine optimierten Kernel), ineffizienter Speicher, mangelnde Skalierung/Überwachung und lange Startzeiten.
- Empfehlung: Wird für die erste Prototypenentwicklung oder für Einzelknotenaufgaben verwendet, die eine benutzerdefinierte Logikintegration erfordern.

Verwenden Sie spezielle Model-Serving-Frameworks (z. B. TensorRT-LLM, TGI) Verwenden Sie spezialisierte Server wie TensorRT-LLM oder TGI für ML-Inferenz und verwalten Sie das Laden, Routing und Optimierung von Modellen. Diese unterstützen Formate wie Safetensors mit optionaler Kompilierung oder Plugins.

- Vorteile: Bietet Batching (Parallelität). static/in-flight or continuous), quantization (INT8, FP8, GPTQ), hardware optimizations (NVIDIA, AMD, Intel, Inferentia), and multi-GPU support (Tensor/ Pipeline TensorRT-LLM unterstützt verschiedene Modelle (LLMs, Encoder-Decoder), während TGI die Integration nutzt. HuggingFace
- Nachteile: TensorRT-LLM muss kompiliert werden und ist nur für NVIDIA verfügbar; TGI ist möglicherweise weniger effizient beim Batching; beide erhöhen den Konfigurationsaufwand und sind möglicherweise nicht für alle Modelltypen geeignet (z. B. Nicht-Transformatoren).
- Empfehlung: Geeignet für PyTorch/TensorFlow Modelle, die Produktionskapazitäten wie Tests oder hohen Durchsatz mit kompatibler Hardware benötigen. A/B

Verwendung spezialisierter Inferenz-Engines mit hohem Durchsatz (z. B. vLLM) Nutzen Sie fortschrittliche Inferenz-Engines wie vLLM und optimieren Sie die LLM-Bereitstellung mit, die Batching während des Fluges und die Quantisierung (PagedAttention, -KV, AWQ), die in die automatische Skalierung von EKS integriert werden können. INT8 FP8

- Vorteile: Hoher Durchsatz und Speichereffizienz (40-60% VRAM-Einsparungen), dynamische Bearbeitung von Anfragen, Token-Streaming, Tensor Parallel Multi-GPU-Unterstützung mit einem Knoten und umfassende Hardwarekompatibilität.
- Nachteile: Optimiert für reine Decoder-Transformatoren (z. B. LLa MA), weniger effektiv für Modelle ohne Transformator, erfordert kompatible Hardware (z. B. NVIDIA) und Einrichtungsaufwand. GPUs
- Empfehlung: Erste Wahl für LLM-Inferenz mit hohem Volumen und geringer Latenz auf EKS, wodurch Skalierbarkeit und Leistung maximiert werden.

Optimierung der Abrufzeiten für Container-Images

Große Container-Images können zu Verzögerungen beim Kaltstart führen, die sich auf die Latenz beim Starten des Pods auswirken. Für latenzempfindliche Workloads, wie z. B. horizontal skalierte Echtzeit-Inferenz-Workloads, ist ein schneller Pod-Start von entscheidender Bedeutung. Ziehen Sie die folgenden Ansätze in Betracht, um die Abrufzeiten von Container-Images zu optimieren:

Reduzierung der Größe von Container-Images

Das Reduzieren der Größe von Container-Images beim Start ist eine weitere Möglichkeit, Bilder zu verkleinern. Sie können bei jedem Schritt des Container-Image-Erstellungsprozesses Reduktionen vornehmen. Wählen Sie zunächst Basis-Images aus, die die geringste Anzahl an erforderlichen Abhängigkeiten enthalten. Schließen Sie bei der Image-Erstellung nur die wesentlichen Bibliotheken und Artefakte ein, die erforderlich sind. Versuchen Sie beim Erstellen von Bildern, mehrere `RUN` `COPY` OD-Befehle zu kombinieren, um eine geringere Anzahl größerer Ebenen zu erstellen. Verwenden Sie für AI/ML Frameworks mehrstufige Builds, um Build und Runtime zu trennen, indem Sie nur benötigte Artefakte kopieren (z. B. via `COPY –from=` für Registries oder lokale Kontexte) und wählen Sie Varianten wie reine Runtime-Images aus (z. B. mit 3,03 GB gegenüber Devel mit 6,66 GB). `pytorch/pytorch:2.7.1-cuda11.8-cudnn9-runtime` Weitere Informationen finden Sie unter Reduzieren der Container-Image-Größe im AI on EKS Workshop.

Verwenden von SOCI-Snapshotter zum Pre-Pull von Bildern

Für sehr große Bilder, die Sie nicht einfach minimieren können, können Sie den Open-Source-Snapshotter Seekable OCI (SOCI) verwenden, der im parallelen Pull & Unpack-Modus konfiguriert ist. Mit dieser Lösung können Sie vorhandene Images verwenden, ohne Ihre Build-Pipelines neu erstellen oder ändern zu müssen. Diese Option ist besonders effektiv, wenn Workloads mit sehr großen Images auf leistungsstarken EC2-Recheninstanzen bereitgestellt werden. Sie eignet sich gut für Netzwerke mit hohem Durchsatz und Hochleistungsspeicherkonfigurationen, wie es bei skalierten Workloads üblich ist. AI/ML

Der pull/unpack SOCI-Parallelmodus verbessert die Leistung end-to-end beim Abrufen von Bildern durch konfigurierbare Parallelisierungsstrategien. Schnelleres Abrufen und Vorbereiten von Images wirken sich direkt darauf aus, wie schnell Sie neue Workloads bereitstellen und Ihren Cluster effizient skalieren können. Das Abrufen von Images erfolgt in zwei Hauptphasen:

1. Ebenen werden aus der Registrierung auf den Knoten abgerufen

   Für die Optimierung des Layer-Abrufs erstellt SOCI mehrere gleichzeitige HTTP-Verbindungen pro Ebene, wodurch der Download-Durchsatz über die Beschränkung für eine einzelne Verbindung hinaus vervielfacht wird. Es teilt große Ebenen in Blöcke auf und lädt sie gleichzeitig über mehrere Verbindungen herunter. Dieser Ansatz trägt dazu bei, Ihre verfügbare Netzwerkbandbreite zu sättigen und die Download-Zeiten erheblich zu reduzieren. Dies ist besonders nützlich für AI/ML Workloads, bei denen eine einzelne Ebene mehrere Gigabyte umfassen kann.

2. Diese Schichten auspacken und vorbereiten, um Container zu erstellen

   Zur Optimierung des Layer-Entpackens verarbeitet SOCI mehrere Ebenen gleichzeitig. Anstatt zu warten, bis jede Ebene vollständig entpackt ist, bevor mit der nächsten begonnen wird, werden Ihre verfügbaren CPU-Kerne verwendet, um mehrere Schichten gleichzeitig zu dekomprimieren und zu extrahieren. Diese Parallelverarbeitung verwandelt die herkömmliche I/O-gebundene Auspackphase in einen CPU-optimierten Vorgang, der mit Ihren verfügbaren Kernen skaliert. Das System orchestriert diese Parallelisierung sorgfältig, um die Konsistenz des Dateisystems aufrechtzuerhalten und gleichzeitig den Durchsatz zu maximieren.

Der parallel SOCI-Pull-Modus verwendet ein Steuerungssystem mit zwei Schwellenwerten und konfigurierbaren Parametern sowohl für die Parallelität beim Herunterladen als auch für die Entpackungsparallelität. Mit dieser detaillierten Steuerung können Sie das Verhalten von SOCI an Ihre spezifischen Leistungsanforderungen und Umgebungsbedingungen anpassen. Wenn Sie diese Parameter verstehen, können Sie Ihre Laufzeit optimieren, um die beste Pull-Leistung zu erzielen.

Referenzen

- Weitere Informationen zur Lösung und zu Kompromissen bei der Optimierung finden Sie in der Feature-Dokumentation im SOCI-Projekt-Repository unter. GitHub
- Ein praktisches Beispiel mit Karpenter auf Amazon EKS finden Sie im Karpenter Blueprint mit SOCI-Snapshotter-Parallelmodus. pull/unpack
- Informationen zur Konfiguration von Bottlerocket für parallelen Pull finden Sie in der Bottlerocket-Dokumentation unter soci-snapshotter Parallel Pull Unpack Mode.

Verwenden von EBS-Snapshots zum Pre-Pull von Bildern

Sie können einen Amazon Elastic Block Store (EBS) -Snapshot von zwischengespeicherten Container-Images erstellen und diesen Snapshot für EKS-Worker-Knoten wiederverwenden. Dadurch wird sichergestellt, dass die Bilder beim Start des Knotens lokal vorab abgerufen werden, wodurch die Pod-Initialisierungszeit reduziert wird. Weitere Informationen zur Verwendung von Karpenter und EKS Terraform Blueprints für verwaltete Knotengruppen finden Sie unter Verkürzen der Container-Startzeit auf Amazon EKS mit Bottlerocket-Datenvolumen.

Weitere Informationen finden Sie unter Verwenden von containerd snapshotter und Preload von Container-Images in Bottlerocket-Datenvolumes mit EBS-Snapshots im AI on EKS Workshop.

Verwenden des Container-Runtime-Cache zum Pre-Pull von Images

Sie können Container-Images mithilfe von Kubernetes-Ressourcen (z. B. DaemonSet oder Deployment) vorab auf Knoten abrufen, um den Container-Runtime-Cache des Knotens zu füllen. Der Container-Runtime-Cache ist der lokale Speicher, der von der Container-Laufzeit verwaltet wird (z. B. containerd), in dem Bilder gespeichert werden, nachdem sie aus einer Registrierung abgerufen wurden. Durch das Pre-Pull wird sichergestellt, dass Bilder lokal verfügbar sind, wodurch Verzögerungen beim Herunterladen beim Pod-Start vermieden werden. Dieser Ansatz ist besonders nützlich, wenn sich Images häufig ändern (z. B. bei häufigen Updates), wenn EBS-Snapshots nicht vorkonfiguriert sind, wenn das Erstellen eines EBS-Volumes zeitaufwändiger wäre als das direkte Abrufen aus einer Container-Registry, oder wenn sich Knoten bereits im Cluster befinden und Pods bei Bedarf mit einem von mehreren möglichen Images hochgefahren werden müssen.

Das Vorabziehen aller Varianten gewährleistet eine schnelle Startzeit, unabhängig davon, welches Image benötigt wird. Bei einem massiv parallel ML-Workload, der 100.000 kleine Modelle erfordert, die mit 10 verschiedenen Techniken erstellt wurden, minimiert das Vorabziehen von 10 Bildern DaemonSet über einen großen Cluster (z. B. Tausende von Knoten) die Pod-Startzeit und ermöglicht die Fertigstellung in weniger als 10 Sekunden, da On-Demand-Pulls vermieden werden. Die Verwendung des Container-Runtime-Cache-Ansatzes macht die Verwaltung von EBS-Snapshots überflüssig und stellt sicher, dass Sie immer die neueste Container-Image-Version erhalten. Bei DaemonSets Inferenz-Workloads in Echtzeit, bei denen Knoten 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 Muster skalieren, kann dies jedoch

dazu führen, dass Bilder auf inaktiven Knoten entfernt werden, was bei nachfolgenden Scale-ups Re-Pulls erforderlich macht und die Zuverlässigkeit des Caches für Burst-Workloads verringert.

Beispiele für das Pre-Pullen von Images in den Container-Runtime-Cache finden Sie im GitHub AWS-Repository.

Ziehen Sie NVMe Kubelet- und Containerspeicher in Betracht

Erwägen Sie `containerd` die Konfiguration `kubelet` und Verwendung von temporären NVMe Instance-Speicherlaufwerken für eine höhere Festplattenleistung. Der Container-Pull-Prozess beinhaltet das Herunterladen eines Container-Images aus einer Registrierung und das Dekomprimieren seiner Schichten in ein verwendbares Format. Um die I/O Abläufe während der Dekomprimierung zu optimieren, sollten Sie abwägen, was für den Instance-Typ Ihres Container-Hosts ein höheres Maß an I/O Leistung und Durchsatz bietet: NVMe Backed-Instances mit lokalem Speicher im Vergleich zu EBS-Volume-IOPS/Durchsatz. Für EC2-Instances mit NVMe lokalem Speicher sollten Sie erwägen, das dem Knoten zugrunde liegende Dateisystem für kubelet (`/var/lib/kubelet`), containerd () und Pod-Logs (`/var/lib/containerd/var/log/pods`) zu konfigurieren, um kurzlebige NVMe Instance-Speicherplatten für ein höheres Maß an Leistung und Durchsatz zu verwenden. I/O

Der kurzlebige Speicher des Knotens kann von Pods gemeinsam genutzt werden, die kurzlebigen Speicher anfordern, und von Container-Images, die auf den Knoten heruntergeladen werden. Wenn Sie Karpenter mit Bottlerocket oder AL2023 EKS Optimized verwenden, kann AMIs dies in der Einstellung auf RAID0oder, bei Verwendung von Managed Node Groups, EC2NodeClassdurch die Einstellung instanceStorePolicy in als Teil der Benutzerdaten konfiguriert werden. localStoragePolicy NodeConfig

Tragen Sie zu diesem Leitfaden bei

Jeder kann zu dem Best-Practice-Leitfaden beitragen. Der EKS Best Practices Guide ist in dem AsciiDoc Format auf verfasst GitHub.

Zusammenfassung für bestehende Mitwirkende

- Öffnen Sie das `bpg-docs.code-workspace`mit VS Code, um die AsciiDoc Erweiterung automatisch zu installieren.
  - Erfahren Sie mehr über die AsciiDoc Erweiterung auf dem Visual Studio Marketplace.
- Die Quelldateien für die AWS Docs-Website werden gespeichert in `latest/bpg`
- Die Syntax ist Markdown sehr ähnlich.
  - Lesen Sie die Syntaxreferenz in den AsciiDoctor Dokumenten.
- Die Docs-Plattform wird nur bereitgestellt. `latest/bpg/images` Jeder der Leitfadenabschnitte enthält einen symbolischen Link zurück zu diesem Verzeichnis. Zeigt zum Beispiel `latest/bpg/networking/images` auf`latest/bpg/images`.

Richten Sie eine lokale Bearbeitungsumgebung ein

Wenn Sie das Handbuch häufig bearbeiten möchten, richten Sie eine lokale Bearbeitungsumgebung ein.

Forke und klone das Repo

Sie müssen mit`git`, `github` und Texteditoren vertraut sein. Informationen zu den ersten Schritten mit `git` und `github` finden Sie in den GitHub Dokumenten unter Erste Schritte mit Ihrem GitHub Konto.

1. Den EKS Best Practices Guide finden Sie unter GitHub.
2. Erstellen Sie einen Fork des Projekt-Repos. In den GitHub Dokumenten erfahren Sie, wie Sie ein Repository forken.
3. Klonen Sie Ihren Fork des Projekt-Repositorys. Erfahre, wie du dein geforktes Repository klonen kannst.

Öffnen Sie den VS Code Workspace

AWS empfiehlt, Visual Studio Code von Microsoft zu verwenden, um den Leitfaden zu bearbeiten. Weitere Informationen zu VS Code finden Sie unter Visual Studio Code herunterladen und Erste Schritte mit Visual Studio Code in der Visual Studio Code-Dokumentation.

1. Öffnen Sie VS Code.
2. Öffnen Sie die `bpg-docs.code-workspace` Datei aus dem geklonten Repo.
3. Wenn Sie diesen Arbeitsbereich zum ersten Mal öffnen, akzeptieren Sie die Aufforderung zur Installation der AsciiDoc Erweiterung. Diese Erweiterung überprüft die Syntax von AsciiDoc Dateien und generiert eine Live-Vorschau.
4. Navigieren Sie zum `latest/bpg` Verzeichnis. Dieses Verzeichnis enthält die Quelldateien, die auf der AWS-Dokumentationsseite bereitgestellt werden. Die Quelldateien sind nach Abschnitten mit Anleitungen wie „Sicherheit“ oder „Netzwerk“ gegliedert.

Bearbeiten einer Datei

1. Öffnen Sie eine Datei im Editor.
   - In der AsciiDoc Syntax erfahren Sie, wie Sie Überschriften, Links und Listen erstellen.
   - Sie können die Markdown-Syntax verwenden, um Text zu formatieren, Listen und Überschriften zu erstellen. Sie können die Markdown-Syntax nicht verwenden, um Links zu erstellen.
2. Öffnen Sie eine Live-Vorschau der Seite.
   - Drücken Sie zunächst `ctrl-k` oder `cmd-k` (je nach Tastatur). Zweitens drücken Siev. Dadurch wird eine Vorschau in geteilter Ansicht geöffnet.

AWS schlägt vor, Feature-Branches zu verwenden, um Ihre Änderungen zu organisieren. Erfahren Sie, wie Sie mit Git Branches erstellen.

Reichen Sie eine Pull-Anfrage ein

Sie können eine Pull-Anfrage von der GitHub Website oder der GitHub CLI aus erstellen.

Erfahre, wie du mithilfe der GitHub Website eine Pull-Anfrage von einem Fork aus erstellst.

Erfahre, wie du mithilfe der GitHub CLI eine Pull-Anfrage erstellst.

Verwenden Sie den webbasierten Editor github.dev

Der `github.dev` webbasierte Editor basiert auf VS Code. Dies ist eine hervorragende Möglichkeit, mehrere Dateien zu bearbeiten und eine Vorschau von Inhalten ohne Einrichtung anzuzeigen.

Es unterstützt die AsciiDoc Erweiterung. Sie können Git-Operationen mit der GUI ausführen. Der webbasierte Editor verfügt weder über eine Shell noch über ein Terminal zum Ausführen von Befehlen.

Sie müssen ein GitHub Konto haben. Bei Bedarf werden Sie aufgefordert, sich anzumelden.

Starten Sie den GitHub webbasierten Editor.

Bearbeiten Sie eine einzelne Seite

Sie können einzelne Seiten schnell aktualisieren, indem Sie GitHub Jede Seite enthält unten den Link GitHub "# Diese Seite bearbeiten auf".

1. Navigieren Sie zu der Seite in diesem Handbuch, die Sie bearbeiten möchten
2. Klicken Sie unten auf den Link GitHub „Diese Seite bearbeiten auf“
3. Klicken Sie oben rechts im GitHub Dateiviewer auf das Bleistiftsymbol „Bearbeiten“ oder drücken Sie e
4. Bearbeiten Sie die Datei
5. Reichen Sie Ihre Änderungen mit der Schaltfläche „Änderungen übernehmen...“ ein. Diese Schaltfläche erstellt einen GitHub Pull-Request. Die Betreuer des Leitfadens werden diesen Pull-Request überprüfen. Ein Prüfer wird den Pull-Request genehmigen oder Änderungen beantragen.

Die ID für eine Seite anzeigen und festlegen

Auf dieser Seite wird erklärt, wie die Seiten-ID angezeigt und festgelegt wird.

Die Seiten-ID ist eine eindeutige Zeichenfolge, die jede Seite auf der Dokumentationsseite identifiziert. Sie können die Seiten-ID in der Adressleiste Ihres Browsers sehen, wenn Sie sich auf einer bestimmten Seite befinden. Die Seiten-ID wird für die URL, den Dateinamen und zum Erstellen von Querverweislinks verwendet.

Wenn Sie sich diese Seite ansehen, sieht die URL in der Adressleiste Ihres Browsers beispielsweise wie folgt aus:

```
https://docs.aws.amazon.com/view-set-page-id.html
```

Der letzte Teil der URL (`view-set-page-id`) ist die Seiten-ID.

Stellen Sie die Seiten-ID ein

Wenn Sie eine neue Seite erstellen, müssen Sie die Seiten-ID in der Quelldatei festlegen. Die Seiten-ID sollte eine kurze Zeichenfolge mit Bindestrich sein, die den Seiteninhalt beschreibt.

1. Öffnen Sie die Quelldatei für Ihre neue Seite in einem Texteditor.
2. Fügen Sie am Anfang der Datei die folgende Zeile hinzu. Sie sollte über der ersten Überschrift stehen.

   ```
   [#my-new-page]
   ```

   `my-new-page`Ersetzen Sie es durch die Seiten-ID für Ihre neue Seite.
3. Speichern Sie die Datei.

> **Note**
>
> Die Seite IDs muss auf der gesamten Dokumentationsseite eindeutig sein. Wenn Sie versuchen, eine vorhandene Seiten-ID zu verwenden, erhalten Sie einen Erstellungsfehler.

Erstellen einer neuen Seite

Erfahren Sie, wie Sie eine neue Seite erstellen und das Inhaltsverzeichnis der Anleitung aktualisieren.

Seitenmetadaten erstellen

1. Ermitteln Sie den Seitentitel und den Kurztitel der Seite. Der Kurztitel der Seite ist optional, wird jedoch empfohlen, wenn der Seitentitel mehr als ein paar Wörter umfasst.
2. Ermitteln Sie die ID der Seite. Dies muss innerhalb des EKS Best Practices Guide eindeutig sein. Die Konvention besteht darin, ausschließlich Kleinbuchstaben und separate Wörter mit `-` zu verwenden.
3. Erstellen Sie eine neue ASCIIDOC-Datei, falls erforderlich, in einem Ordner, und fügen Sie der Datei den folgenden Text hinzu:

Example

[.“ Thema "] [#<page-id>] = <page-title>:info_titleabbrev: < > page-short-title

Zum Beispiel

Example

[.“ topic "] [#scalability] = Bewährte Methoden zur Skalierbarkeit von EKS: info_titleabbrev: Skalierbarkeit

Zum Inhaltsverzeichnis hinzufügen

1. Öffnen Sie die Datei für die übergeordnete Seite im Inhaltsverzeichnis. Für neue Abschnitte des Leitfadens auf oberster Ebene lautet die übergeordnete Dateibook.adoc.
2. Aktualisieren Sie am Ende der übergeordneten Datei die folgende Anweisung und fügen Sie sie ein:

   Example

   beinhalten: <new-filename>[leveloffset=+1]

   Zum Beispiel

   Example

   beinhalten: :dataplane.adoc [leveloffset=+1]

Fügen Sie ein Bild ein

1. Suchen Sie das Bildpräfix für die Seite, die Sie bearbeiten. Überprüfen Sie die :imagesdir: Eigenschaft in der Überschrift der Datei. Zum Beispiel `:imagesdir: images/reliability/
2. Platzieren Sie Ihr Bild in diesem Pfad, z. B. latest/bpg/images/reliability
3. Ermitteln Sie den passenden Alt-Text für Ihr Bild. Schreiben Sie eine kurze allgemeine Beschreibung des Bildes. Beispielsweise ist „Diagramm einer VPC mit drei Verfügbarkeitszonen“ ein geeigneter Alt-Text.

4. Aktualisieren Sie das folgende Beispiel mit dem Alt-Text und dem Bilddateinamen. Fügen Sie es an der gewünschten Stelle ein.

   Example

   Bild:: <image-filename>[< image-alt-text >]

   Zum Beispiel

   Example

   image:: eks-data-plane-connectivity .jpeg [Netzwerkdiagramm]

Überprüfen Sie den Stil mit Vale

1. Installieren Sie die Vale-CLI.
2. Führen Sie Folgendes aus:`vale sync`
3. Installieren Sie die Vale-Erweiterung vom Visual Studio Marketplace.
4. Starten Sie VS Code neu und öffnen Sie eine AsciiDoc Datei
5. VS Code unterstreicht problematischen Text. In den VS Code-Dokumenten erfahren Sie, wie Sie mit Fehlern und Warnungen arbeiten.

Erstellen Sie eine lokale Vorschau

1. Installieren Sie das `asciidoctor` Tool `brew` unter Linux oder macOS
   - In den Dokumenten erfahren Sie, wie Sie asciidoctor cli installieren. AsciiDoctor
   - Erfahren Sie, wie Sie den Brew-Paketmanager installieren.
2. Öffnen Sie ein Terminal und navigieren Sie zu `latest/bpg/`
3. Führen Sie Folgendes aus:`asciidoctor book.adoc`
   - Überprüfen Sie alle Syntaxwarnungen und Fehler
4. Öffnen Sie die `book.html` Ausgabedatei.
   - Unter macOS können Sie ausführen, `open book.html` um die Vorschau in Ihrem Standardbrowser zu öffnen.

AsciiDoc Spickzettel

Grundlegende Formatierung

```
*bold text*
_italic text_
`monospace text`
```

Kopfzeilen

```
= Document Title (Header 1)
== Header 2
=== Header 3
==== Header 4
===== Header 5
====== Header 6
```

Listen

Ungeordnete Listen:

```
- Item 1
- Item 2
-- Subitem 2.1
-- Subitem 2.2
- Item 3
```

Geordnete Listen:

```
. First item
. Second item
.. Subitem 2.1
.. Subitem 2.2
. Third item
```

Links

```
External link:  https://example.com[Link text]
Internal link: <<page-id>>
```

```
Internal link: xref:page-id[Link text]
```

Bilder

```
image::image-file.jpg[Alt text]
```

Code-Blöcke

```
 [source,python]
 ----
 def hello_world():
     print("Hello, World!")
 ----
```

Tabellen

Erfahren Sie, wie Sie eine einfache Tabelle erstellen.

```
[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
|===
```

Ermahnungen

```
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.
```

Vorversion:

> **Note**
>
> Dies ist eine kurze Ermahnung.

Umfasst

```
include::filename.adoc[]
```

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.