Guía de mejores prácticas

Amazon EKS



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

Amazon EKS: Guía de mejores prácticas

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

Las marcas comerciales y la imagen comercial de Amazon no se pueden utilizar en relación con ningún producto o servicio que no sea de Amazon, de ninguna manera que pueda causar confusión entre los clientes y que menosprecie o desacredite a Amazon. Todas las demás marcas comerciales que no son propiedad de Amazon son propiedad de sus respectivos dueños, quienes pueden o no estar afiliados, conectados o patrocinados por Amazon

Table of Contents

Introducción 1
Guías relacionadas 1
Contribución 2
Seguridad 3
Cómo usar esta guía 3
Entendiendo el modelo de responsabilidad compartida 3
Introducción 1
Comentarios 6
Documentación adicional 6
Herramientas y recursos 6
Modo automático EKS: seguridad 6
Arquitectura de seguridad 7
Preguntas frecuentes (FAQ) 17
Gestión de identidad y acceso 20
Control del acceso a los clústeres de EKS 20
Recomendaciones de acceso a clústeres 28
Identidades y credenciales para los pods de EKS 34
Identidades y credenciales para los pods de EKS: recomendaciones 43
Herramientas y recursos 52
Seguridad de Pod 53
Capacidades de Linux 53
Soluciones de seguridad Pod 54
Recomendaciones 64
Herramientas y recursos 52
Tenencia múltiple 71
Multitenencia suave 72
Construcciones de Kubernetes 74
Controles atenuantes 77
Tenencia múltiple dura 85
Orientaciones futuras 85
Herramientas y recursos de administración de varios clústeres 86
Controles de Detective 86
Recomendaciones 64
Herramientas y recursos 52

Seguridad de la red 94
Control de tráfico 94
Network encryption 94
Política de red 95
Recomendaciones 64
Grupos de seguridad 100
¿Cuándo usar la política de red o el grupo de seguridad para los pods? 102
Aplicación de políticas de Service Mesh o política de red de Kubernetes 104
ThirdParty Motores de políticas de red 105
Cifrado en tránsito 107
Herramientas y recursos 52
Cifrado de datos y gestión de secretos 117
Cifrado en reposo 117
Administración de secretos 119
Herramientas y recursos 52
Seguridad en tiempo de ejecución 122
Contextos de seguridad y controles de Kubernetes integrados 123
Recomendaciones 64
Herramientas y recursos 127
Seguridad de infraestructuras 127
Recomendaciones 64
Alternativas 132
Herramientas y recursos 52
Cumplimiento normativo 135
Girando a la izquierda 137
Herramientas y recursos 52
Respuesta a incidentes y análisis forense 139
Ejemplo de plan de respuesta a incidentes 139
Recomendaciones 64
Herramientas y recursos 52
Seguridad de imagen 144
Recomendaciones 64
Herramientas y recursos 52
Estrategia de cuentas múltiples 155
Planeación de una estrategia de cuentas con múltiples cargas de trabajo para clústeres con varios inquilinos 156

Clúster EKS centralizado 157
Clústeres EKS descentralizados 163
Clústeres de EKS centralizados o descentralizados 165
Administración del acceso a los clústeres 168
Opciones de administración de acceso de EKS 168
Opción 1: AWS IAM Identity Center con API de administración de acceso a clústeres (CAM) 169
Opción 2: AWS IAM Users/Roles mapeado a grupos de Kubernetes 171
Opción 3: proveedores de OIDC 172
AWS EKS Pod Identity frente a IRSA para cargas de trabajo 172
Recomendación 173
Escalado automático de clústeres 174
Modo automático de EKS 174
Razones para usar el modo automático 175
Preguntas frecuentes 175
Karpenter 181
Recomendaciones 64
Mejores prácticas de Karpenter 182
Creando NodePools 186
¿Programar pods? 191
Recomendaciones de CoredNS 194
Planos de Karpenter 194
Recursos adicionales 93
Escalador automático de clústeres 195
Descripción general de 195
Optimización del rendimiento y la escalabilidad 200
Optimización del costo y la disponibilidad 203
Casos de uso avanzados 206
Parámetros adicionales 208
Recursos adicionales 93
Referencias 84
Fiabilidad 213
Cómo usar esta guía 3
Introducción 1
Comentarios 6
Aplicaciones 216

Recomendaciones 64
Escalador automático de pod horizontal (HPA) 221
Escalador automático de cápsulas verticales (VPA) 222
Actualización de aplicaciones 223
Health chequeos y autocuración 225
Recomendaciones 225
Hacer frente a las interrupciones 227
Recomendaciones 229
Observabilidad 231
Recomendaciones 231
Plano de control 234
Arquitectura EKS 234
Recomendaciones 235
Supervise las métricas del plano de control 235
Autenticación de 238
Webhooks de admisión 240
Gestión de las actualizaciones de clústeres 241
Conectividad de terminales de clúster 242
Ejecutar clústeres de gran tamaño 243
Recursos adicionales: 243
Plano de datos 244
Recomendaciones 64
Red 251
Modelo de red de Kubernetes 251
Interfaz de red de contenedores (CNI) 252
Amazon Virtual Private Cloud (VPC) CNI 253
Calculadora de subredes 254
Consideraciones sobre la VPC y las subredes 255
Descripción general de 195
Recomendaciones 64
CNI de Amazon VPC 266
Descripción general de 195
Recomendaciones 64
Optimización de la utilización de direcciones IP 279
Optimice el consumo de IP a nivel de nodo 280
Mitigación del riesgo de agotamiento de la IP 280

Ejecutando IPv6 clústeres 287
Descripción general de 195
Recomendaciones 64
Redes personalizadas 297
Configuración de ejemplo 298
Recomendaciones 64
Modo de prefijo para Linux 305
Recomendaciones 64
Modo de prefijo para Windows 310
Recomendaciones 64
Grupos de seguridad por pod 316
Recomendaciones 64
Equilibrio de carga 326
Elegir el tipo de Load Balancer 326
Aprovisionamiento de balanceadores de carga 327
Elegir el tipo de destino del Load Balancer 328
Configuración de las comprobaciones de estado de Load Balancer 330
Disponibilidad y ciclo de vida del pod 331
Referencias 84
Apéndice 336
Supervisión de problemas de rendimiento de la red 338
Supervisión del tráfico de CoredNS para detectar problemas de limitación de DNS 338
Supervisión de los retrasos en las consultas de DNS mediante las métricas de Conntrack .. 339
Otras métricas importantes de rendimiento de la red 339
Capturar las métricas para monitorear las cargas de trabajo en busca de problemas de rendimiento de la red 340
Ejecutando kube-proxy en modo IPVS 348
Descripción general de 195
Escalabilidad 353
Cómo usar esta guía 353
Comprender las dimensiones de escalado 353
Escalado extra grande 354
Plano de control 354
Limite la carga de trabajo y la sobrecarga de nodos 355
Amplíe los nodos y los módulos de forma segura 355
Usa la caché del lado del cliente cuando ejecutes Kubectl 356

Inhabilita la compresión kubectl 357
Escalador automático de clústeres compartidos 357
Prioridad e imparcialidad de la API 359
Recuperando recursos en el servidor de API 366
Plano de datos 373
Escalado automático de nodos 374
Usa muchos tipos de instancias diferentes EC2 374
Prefiera nodos más grandes para reducir la carga del servidor de API 375
Utilice nodos de tamaños similares para lograr un rendimiento uniforme de las cargas de trabajo 376
Utilice los recursos informáticos de forma eficiente 377
Automatice las actualizaciones de Amazon Machine Image (AMI) 377
Utilice varios volúmenes de EBS como contenedores 378
Evite las instancias con límites de conexión de EBS bajos si las cargas de trabajo utilizan volúmenes de EBS 380
Deshabilita los registros innecesarios en el disco 380
Las instancias de parcheo se implementan cuando la velocidad de actualización del sistema operativo es necesaria 380
Servicios de clúster 381
CoredNS de escala 382
Escale el servidor Kubernetes Metrics de forma vertical 383
Duración lamentable de CoredNS 384
Sonda de preparación de CoredNS 384
Agentes de registro y monitoreo 385
Cargas de trabajo 386
Úselo IPv6 para redes de pods 386
Limite la cantidad de servicios por espacio de nombres 387
Comprenda las cuotas de Elastic Load Balancer 387
Utilice Route 53, Global Accelerator o CloudFront 388
Úselo EndpointSlices en lugar de puntos finales 388
Si es posible, utilice secretos externos e inmutables 389
Limite el historial de despliegues 390
Desactivar enableServiceLinks de forma predeterminada 390
Limite los webhooks de admisión dinámica por recurso 390
Compare las cargas de trabajo en varios clústeres 391
La teoría detrás del escalado 392

Los nodos frente a la tasa de abandono ... 392
Pensando en consultas por segundo ... 392
Escalar los componentes distribuidos ... 393
Cuellos de botella ascendentes y descendentes ... 394
Escale por métricas ... 396
Monitorización del plano de control ... 399
Servidor de API ... 399
¿Dónde está el problema? ... 399
Programador ... 405
Kube Controller Manager ... 407
ETC.D. ... 409
Eficiencia y escalado de los nodos ... 410
Selección de nodos ... 410
Empaquetado en contenedores de nodos ... 412
Utilización frente a saturación ... 419
Establecer límites de CPU ... 427
Kubernetes SLOs ... 431
Kubernetes SLOs ... 431
Métricas SLI de Kubernetes ... 434
SLOs en tu clúster ... 438
Known Limits and Service Quotas ... 439
Otras cuotas de AWS Service ... 440
Limitación de solicitudes de AWS ... 447
Otros límites conocidos ... 448
Actualizaciones de clústeres ... 450
Descripción general de ... 450
Antes de actualizar ... 450
Conserve su clúster up-to-date ... 451
Consulte el calendario de lanzamientos de EKS ... 452
Comprenda cómo se aplica el modelo de responsabilidad compartida a las actualizaciones de clústeres ... 452
Actualice los clústeres in situ ... 453
Actualice el plano de control y el plano de datos en secuencia ... 453
Utilice la documentación de EKS para crear una lista de verificación de actualizaciones ... 454
Actualice los complementos y componentes mediante la API de Kubernetes ... 454
Compruebe los requisitos básicos de EKS antes de realizar la actualización ... 456

Compruebe las direcciones IP disponibles 457
Compruebe el rol de IAM de EKS 458
Migre a los complementos de EKS 458
Identifique y corrija el uso de API eliminado antes de actualizar el plano de control 459
Información sobre clústeres 460
K. ube-no-trouble 461
Plutón 462
Recursos 463
Actualice las cargas de trabajo de Kubernetes. Usa kubectl-convert para actualizar los manifiestos 464
Configure PodDisruptionBudgets y garantice topologySpreadConstraints la disponibilidad de sus cargas de trabajo mientras se actualiza el plano de datos 464
Utilice Managed Node Groups o Karpenter para simplificar las actualizaciones de los planos de datos 466
Confirma la compatibilidad de la versión con los nodos existentes y el plano de control 466
Habilite la caducidad de los nodos administrados por Karpenter 467
Utilice la función Drift para los nodos gestionados por Karpenter 467
Usa eksctl para automatizar las actualizaciones de los grupos de nodos autogestionados 468
Backup del clúster antes de la actualización 468
Reinicie los despliegues de Fargate después de actualizar el plano de control 469
Evalúe Blue/Green los clústeres como una alternativa a las actualizaciones de clústeres in situ 469
Realice un seguimiento de los principales cambios planificados en el proyecto Kubernetes: piense en el futuro 470
Guía específica sobre la eliminación de funciones 470
Eliminación de Dockershim en la versión 1.25: utilice un detector para Docker Socket (DDS) 470
Eliminación de la PodSecurityPolicy versión 1.25: migración a los estándares de seguridad de Pod o a una policy-as-code solución 471
El controlador de almacenamiento integrado en árbol quedó obsoleto en la versión 1.23: migración a los controladores de la interfaz de almacenamiento en contenedores (CSI) 471
Recursos adicionales 471
ClowdHaus Guía de actualización de EKS 471
GoNoGo 472
Optimización de costes 473
Directrices generales 473

Mejores prácticas de optimización de costos de EKS 473
Cómo usar esta guía 3
Características clave de los servicios de AWS y Kubernetes 474
Comentarios 6
Marcos 475
El pilar See: medición y rendición de cuentas 475
El pilar del ahorro: optimización de costes 476
El pilar del plan: planificación y previsión 477
El pilar de la ejecución 477
Referencias 84
Conciencia 478
Recomendaciones 64
Otras herramientas 486
Computación 487
Dimensione sus cargas de trabajo de forma adecuada 488
Reduzca el consumo 489
Reduzca la capacidad no utilizada 490
Optimice los tipos de capacidad informática 496
Optimice el uso de la computación 500
Network 502
Comunicación de pod a pod 502
Comunicación entre Load Balancer y Pod 515
Transferencia de datos desde el registro de contenedores 517
Transferencia de datos a Internet y a los servicios de AWS 518
Transferencia de datos entre VPCs 521
Uso de una malla de servicios 524
Recursos adicionales 93
Almacenamiento 533
Descripción general 195
Volúmenes efímeros 533
Volúmenes persistentes 534
Otras consideraciones 542
Observabilidad 543
Introducción 543
Registro 543
Plano de control EKS 543

Plano de datos EKS 544
Métricas 549
Rastreo 552
Recursos adicionales: 93
Windows 556
Administración de AMI 556
Administrar su propia AMI de Windows optimizada para Amazon EKS 557
Configuración de un inicio más rápido para un EKS personalizado y optimizado AMIs 558
Almacenamiento en caché de las capas base de Windows de forma personalizada AMIs ... 559
Publicación de blog. 560
Contenedores gMSA para Windows 560
Qué es una cuenta gMSA 560
Caso de uso de contenedor de Windows y gMSA 561
Fortalecimiento de Windows Server 563
Reducir la superficie de ataque con Windows Server Core 563
Evitar las conexiones RDP 564
Amazon Inspector 565
Amazon GuardDuty 566
Seguridad en Amazon EC2 para Windows 566
Escaneo de imágenes de Windows 567
Versiones y licencias de Windows 567
Versión de Windows Server 567
Licencias 568
Registro 568
Recomendaciones de registro 568
Supervisión de contenedores de Windows 569
Redes de Windows 574
Descripción general de las redes de contenedores de Windows 574
Administración de direcciones IP 575
Opciones de interfaz de red de contenedores (CNI) 579
Políticas de red 579
Administración de memoria y sistemas 579
Reservar la memoria del sistema y del kubelet 580
Requisitos de memoria para contenedores de Windows 580
Conclusión 581
Administración de la infraestructura 581

Empujar y extraer imágenes de Windows ... 582
Referencia ... 584
Programación ... 584
Asignación PODs a nodos: mejores prácticas ... 584
Garantizar que las cargas de trabajo específicas del sistema operativo lleguen al host contenedor adecuado ... 584
Manejo de varias compilaciones de Windows en el mismo clúster ... 586
La simplificación y la tolerancia NodeSelector en el pod se manifiestan mediante RuntimeClass ... 587
Soporte para grupos de nodos gestionados ... 588
Documentación adicional ... 589
Pod Security para contenedores de Windows ... 589
Opciones de almacenamiento ... 592
¿Qué es un plugin integrado en el árbol o un plugin out-of-tree de volumen? ... 592
Complemento In-Tree Volume para Windows ... 592
Out-of-tree para Windows ... 596
Servidor FSx de archivos Amazon para Windows ... 597
Fortalecimiento de las imágenes de los contenedores de Windows ... 598
1. Configure las políticas de cuentas (contraseña o bloqueo) mediante las políticas de seguridad y el registro locales ... 599
2. Políticas de auditoría ... 602
3. Prácticas recomendadas de seguridad de IIS para contenedores de Windows ... 605
4. Principio de privilegios mínimos ... 610
Reflexiones finales: Por qué proteger sus contenedores de Windows es imprescindible en el panorama actual de amenazas ... 613
Híbrido ... 615
Desconexión de red ... 615
Prácticas recomendadas ... 616
Conmutación por error con el pod de Kubernetes ... 621
Tráfico de red de aplicaciones ... 628
Credenciales de host ... 633
AI/ML ... 636
Comentarios ... 636
Computación ... 636
Optimización de recursos de GPU y gestión de costes ... 636
Resiliencia de nodos y gestión de trabajos de formación ... 652

Escalado y rendimiento de las aplicaciones 655
Asignación dinámica de recursos para una gestión avanzada de la GPU 657
Red 704
Considere un mayor ancho de banda de red o un adaptador de estructura elástica para aplicaciones con alta comunicación entre nodos 704
Aumente la cantidad de direcciones IP disponibles para acelerar los tiempos de lanzamiento de los pods 705
Seguridad 706
Seguridad y conformidad 706
Almacenamiento 707
Administración y almacenamiento de datos 707
Observabilidad 721
Monitoreo y observabilidad 721
Observabilidad y métricas 723
Desempeño 740
Administración de artefactos de aprendizaje automático, marcos de servicio y optimización de empresas emergentes 740
Optimización de los tiempos de extracción de imágenes de contenedores 743
NVMe Considérelo para kubelet y almacenamiento en contenedores 746
Contribuciones 747
Resumen para los colaboradores actuales 747
Configure un entorno de edición local 747
Bifurca y clona el repositorio 747
Abra el espacio de trabajo de VS Code 748
Editar un archivo 748
Envía una solicitud de extracción 748
Usa el editor web github.dev 749
Edita una sola página 749
Ver y configurar el ID de una página 749
Establece el ID de la página 750
Cómo crear una nueva página 750
Crea metadatos de página 750
Añadir a la tabla de contenido 751
Inserta una imagen 751
Compruebe el estilo con Vale 752
Crea una vista previa local 752

AsciiDoc Hoja de trucos 753
Formato básico 753
Encabezados 753
Listas 753
Links 753
Imágenes 754
Bloques de código 754
Tablas 754
Advertencias 754
Incluye 755
..... dcclvi

Guía de prácticas recomendadas de Amazon EKS

Bienvenido a las guías de mejores prácticas de EKS. El objetivo principal de este proyecto es ofrecer un conjunto de prácticas recomendadas para las operaciones del segundo día de Amazon EKS. Decidimos publicar esta guía para GitHub poder modificarla rápidamente, ofrecer recomendaciones oportunas y efectivas para una variedad de inquietudes e incorporar fácilmente las sugerencias de la comunidad en general.

Actualmente hemos publicado guías sobre los siguientes temas:

- Mejores prácticas de seguridad
- Mejores prácticas de confiabilidad
- Mejores prácticas para el escalado automático de clústeres: Karpenter
- Mejores prácticas para el escalado automático de clústeres: escalador automático de clústeres
- Mejores prácticas para el escalado automático de clústeres: modo automático EKS
- Mejores prácticas para la creación de redes
- Mejores prácticas de escalabilidad
- Mejores prácticas para las actualizaciones de clústeres
- Mejores prácticas para la optimización de costos
- Prácticas recomendadas para ejecutar contenedores de Windows
- Prácticas recomendadas para implementaciones híbridas
- Prácticas recomendadas para ejecutar cargas de trabajo AI/ML

También hemos creado en código abierto una CLI (interfaz de línea de comandos) basada en Python llamada hardeneks para comprobar algunas de las recomendaciones de esta guía.

En el futuro publicaremos una guía de mejores prácticas para el rendimiento, la optimización de costos y la excelencia operativa.

Guías relacionadas

Además de la Guía del usuario de EKS, AWS ha publicado varias otras guías que pueden ayudarle con la implementación de EKS.

- Guías de prácticas recomendadas para contenedores EMR

- Datos sobre EKS
- Prácticas recomendadas de observabilidad de AWS
- Amazon EKS Blueprints for Terraform
- Inicio rápido de Amazon EKS Blueprints

Contribución

Le animamos a que contribuya a estas guías. Si has implementado una práctica que ha demostrado ser eficaz, compártela con nosotros abriendo una edición o una solicitud de información. Del mismo modo, si descubres un error o una falla en la guía que ya hemos publicado, envía un PR para corregirlo. Las pautas para la presentación PRs se encuentran en nuestras Pautas de contribución.

Mejores prácticas de seguridad

Esta guía proporciona consejos sobre cómo proteger la información, los sistemas y los activos que dependen de EKS y, al mismo tiempo, generar valor empresarial mediante evaluaciones de riesgos y estrategias de mitigación. Esta guía forma parte de una serie de guías de mejores prácticas que AWS publica para ayudar a los clientes a implementar EKS de acuerdo con las mejores prácticas. Las guías sobre el rendimiento, la excelencia operativa, la optimización de costes y la fiabilidad estarán disponibles en los próximos meses.

Cómo usar esta guía

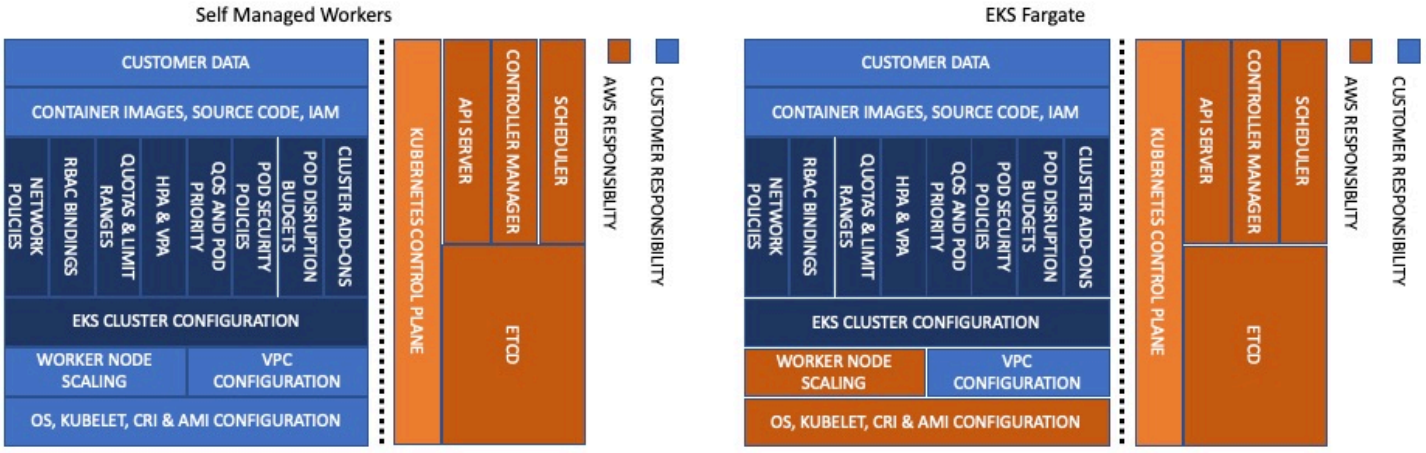
Esta guía está destinada a los profesionales de la seguridad responsables de implementar y supervisar la eficacia de los controles de seguridad para los clústeres de EKS y las cargas de trabajo que soportan. La guía está organizada en diferentes áreas temáticas para facilitar su lectura. Cada tema comienza con una breve descripción general, seguida de una lista de recomendaciones y mejores prácticas para proteger sus clústeres de EKS. No es necesario leer los temas en un orden determinado.

Entendiendo el modelo de responsabilidad compartida

La seguridad y el cumplimiento se consideran responsabilidades compartidas cuando se utiliza un servicio gestionado como EKS. En términos generales, AWS es responsable de la seguridad «de» la nube, mientras que usted, el cliente, es responsable de la seguridad «en» la nube. Con EKS, AWS es responsable de administrar el plano de control de Kubernetes gestionado por EKS. Esto incluye los nodos del plano de control de Kubernetes, la base de datos ETCD y demás infraestructuras necesarias para que AWS ofrezca un servicio seguro y fiable. Como consumidor de EKS, usted es el principal responsable de los temas de esta guía, como la IAM, la seguridad de los módulos, la seguridad del tiempo de ejecución, la seguridad de la red, etc.

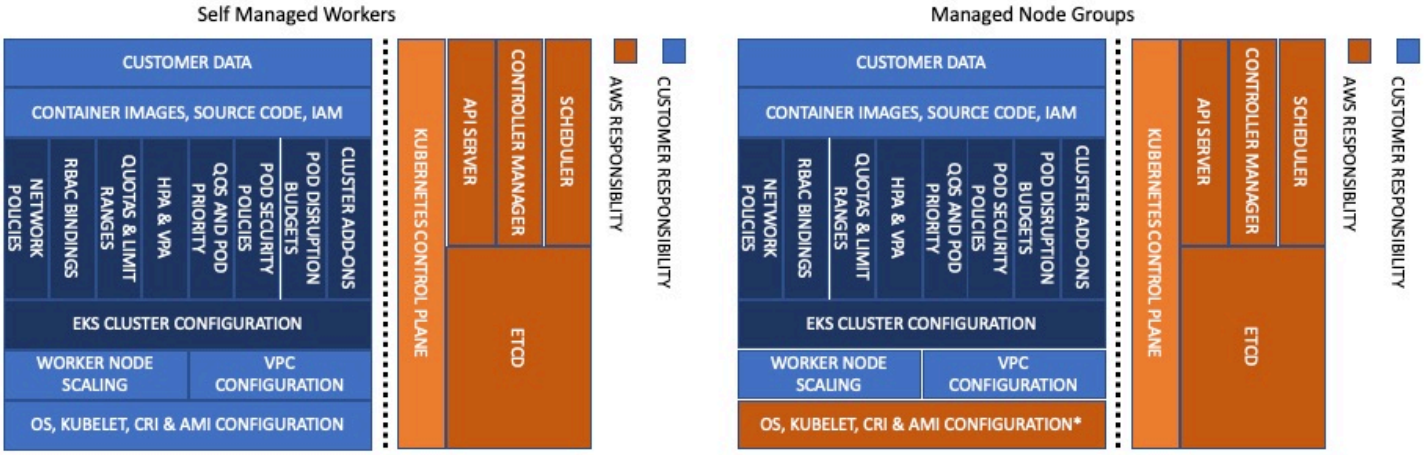
En lo que respecta a la seguridad de la infraestructura, AWS asumirá responsabilidades adicionales a medida que pase de ser trabajadores autogestionados a grupos de nodos gestionados y a Fargate. Por ejemplo, con Fargate, AWS pasa a ser responsable de proteger el elemento subyacente que instance/runtime se utiliza para ejecutar los pods.

Modelo de responsabilidad compartida - Fargate



AWS también asumirá la responsabilidad de mantener actualizada la AMI optimizada para EKS con las versiones de parches y los parches de seguridad de Kubernetes. Los clientes que utilizan grupos de nodos gestionados (MNG) son responsables de actualizar sus grupos de nodos a la AMI más reciente mediante la API de EKS, la CLI, Cloudformation o la consola de AWS. Además, a diferencia de Fargate, no MNGs escalará automáticamente su infraestructura/clúster. Esto se puede gestionar con el escalador automático de clústeres u otras tecnologías, como Karpenter, el escaladoautomático nativo de AWS, Ocean o Atlassian's Escalator. SpotInst

Modelo de responsabilidad compartida: MNG



Antes de diseñar el sistema, es importante saber dónde está la línea de demarcación entre sus responsabilidades y el proveedor del servicio (AWS).

Para obtener información adicional sobre el modelo de responsabilidad compartida, consulte/https://aws.amazon.com/compliance/shared-responsibility-model

Introducción

Hay varias áreas de mejores prácticas de seguridad que son pertinentes cuando se utiliza un servicio de Kubernetes gestionado como EKS:

- Gestión de identidad y acceso
- Pod Security
- Seguridad en tiempo de ejecución
- Seguridad de la red
- Tenencia múltiple
- Cuenta múltiple para tenencia múltiple
- Controles de Detective
- Seguridad de infraestructuras
- Cifrado de datos y gestión de secretos
- Cumplimiento normativo
- Respuesta a incidentes y análisis forense
- Seguridad de imagen

Como parte del diseño de cualquier sistema, debe tener en cuenta sus implicaciones de seguridad y las prácticas que pueden afectar a su postura de seguridad. Por ejemplo, debe controlar quién puede realizar acciones contra un conjunto de recursos. También necesita la capacidad de identificar rápidamente los incidentes de seguridad, proteger sus sistemas y servicios del acceso no autorizado y mantener la confidencialidad e integridad de los datos mediante la protección de los datos. Contar con un conjunto de procesos bien definidos y ensayados para responder a los incidentes de seguridad también mejorará su postura en materia de seguridad. Estas herramientas y técnicas son importantes porque respaldan objetivos como la prevención de pérdidas económicas o el cumplimiento de las obligaciones reglamentarias.

AWS ayuda a las organizaciones a alcanzar sus objetivos de seguridad y conformidad al ofrecer un amplio conjunto de servicios de seguridad que han evolucionado en función de los comentarios de un amplio grupo de clientes preocupados por la seguridad. Al ofrecer una base altamente segura, los clientes pueden dedicar menos tiempo a «tareas pesadas e indiferenciadas» y más tiempo a lograr sus objetivos empresariales.

Comentarios

Esta guía se publica con el GitHub fin de recopilar comentarios y sugerencias directos de la EKS/Kubernetes comunidad en general. Si tienes alguna buena práctica que consideres que deberíamos incluir en la guía, presenta un problema o envía un PR al GitHub repositorio. Nuestra intención es actualizar la guía periódicamente a medida que se agreguen nuevas funciones al servicio o cuando surja una nueva mejor práctica.

Documentación adicional

Documento técnico sobre seguridad de Kubernetes, patrocinado por el Grupo de trabajo de auditoría de seguridad, en el que se describen los aspectos clave de la superficie de ataque y la arquitectura de seguridad de Kubernetes con el objetivo de ayudar a los profesionales de la seguridad a tomar decisiones acertadas de diseño e implementación.

La CNCF publicó también un white paper sobre la seguridad nativa de la nube. El paper examina cómo ha evolucionado el panorama tecnológico y aboga por la adopción de prácticas de seguridad que se alineen con DevOps los procesos y las metodologías ágiles.

Herramientas y recursos

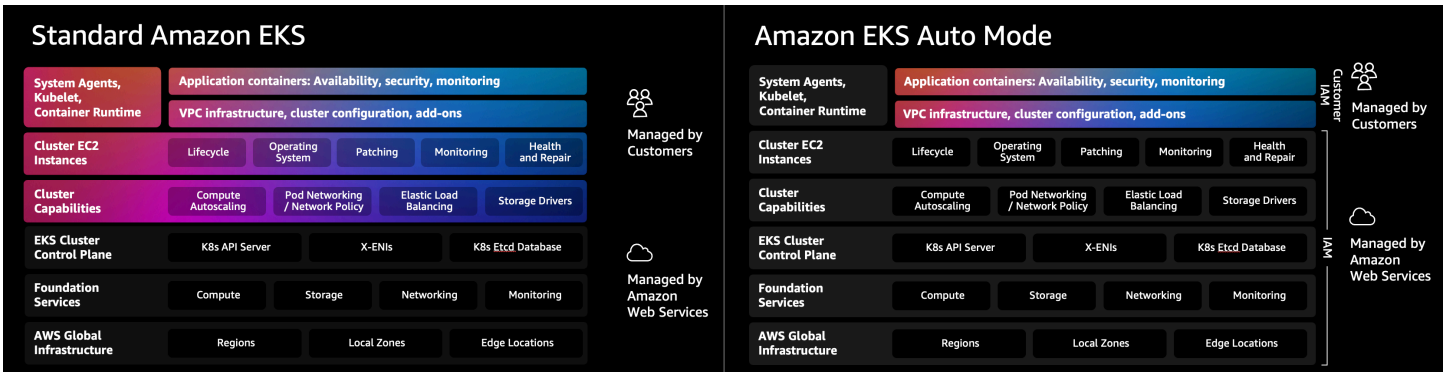
Taller de inmersión en seguridad de Amazon EKS

Modo automático EKS: seguridad

> **Tip**
>
> ¿Le interesa adquirir experiencia práctica con el modo automático Amazon EKS? Inscríbase en AWS Connected Community para participar en un próximo taller virtual de Kubernetes dirigido por expertos de AWS.

Amazon EKS Auto Mode presenta capacidades de seguridad mejoradas al extender la administración de seguridad de AWS más allá del plano de control para incluir nodos de trabajo y componentes principales del clúster. Este modelo de seguridad integral ayuda a las organizaciones a mantener una postura de seguridad sólida y, al mismo tiempo, a reducir la sobrecarga operativa.

Modelo de responsabilidad compartida: modo automático EKS



Las principales mejoras de seguridad del modo automático EKS incluyen:

- Sistema operativo mínimo optimizado para contenedores con una superficie de ataque reducida
- Mejores prácticas de seguridad aplicadas mediante instancias administradas EC2
- Administración automatizada de parches de seguridad con rotación obligatoria de nodos
- Límites de seguridad y aislamiento de nodos integrados
- Integración de IAM simplificada con un mínimo de permisos necesarios

El modo automático de EKS aplica estos controles de seguridad de forma predeterminada, lo que ayuda a las organizaciones a cumplir sus requisitos de seguridad y conformidad, a la vez que simplifica las operaciones de los clústeres. Este enfoque se ajusta a defense-in-depth los principios y proporciona varios niveles de controles de seguridad a nivel de infraestructura, nodo y carga de trabajo.

> **Important**
>
> Si bien el modo automático de EKS ofrece capacidades de seguridad mejoradas, las organizaciones deben seguir implementando los controles de seguridad adecuados en el nivel de aplicación y seguir las mejores prácticas de seguridad para las cargas de trabajo que se ejecutan en el clúster.

Arquitectura de seguridad

El modo automático de EKS implementa controles de seguridad en varios niveles de la infraestructura de EKS, desde el plano de control hasta los nodos individuales. Comprender esta arquitectura es crucial para gestionar y proteger de forma eficaz los clústeres de EKS.

Seguridad del plano de control

El plano de control EKS en el modo automático EKS mantiene los mismos estándares de alta seguridad que los clústeres EKS tradicionales, al tiempo que añade nuevas capacidades de seguridad:

- Cifrado de sobres: todos los datos de la API de Kubernetes se cifran automáticamente mediante cifrado de sobres.
- Integración de KMS: usa AWS KMS con el proveedor de KMS de Kubernetes v2, con opciones para claves propiedad de AWS o claves administradas por el cliente (CMK).
- Administración de componentes mejorada: los componentes críticos, como el autoscalado, la administración de ENI y los controladores de EBS, se trasladan fuera del clúster y son gestionados por AWS.
- Controles de seguridad y capacidades de auditoría mejorados: los permisos necesarios en el modo automático de EKS, más allá de los clústeres de EKS estándar, se gestionan en su totalidad mediante la función de IAM del clúster y no mediante funciones de nodos individuales.

Gestión de acceso e integración de IAM

El modo automático de EKS proporciona una integración mejorada con AWS Identity and Access Management (IAM) a través de EKS Access Entries y EKS Pod Identity.

Administración de acceso a clústeres

El modo automático de EKS introduce mejoras en la gestión del acceso a los clústeres a través de la API de gestión de acceso a los clústeres (CAM):

- Modos de autenticación estandarizados mediante `EKS_API`
- Seguridad mejorada mediante la gestión de acceso basada en API
- Control de acceso simplificado mediante entradas y políticas de acceso

Se pueden crear entradas de acceso para gestionar el acceso a los clústeres:

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

> **Important**
>
> Si bien todavía es posible crear un clúster de modo automático de EKS con el modo de `CONFIG_MAP_AND_API` autenticación, este no es el enfoque estándar y se recomienda encarecidamente utilizar el modo de `API` autenticación predeterminado para los nuevos clústeres. `API`la autenticación basada proporciona una seguridad mejorada y una administración de acceso simplificada ConfigMap en comparación con el enfoque tradicional.

Pod Identity de EKS

El modo automático de EKS viene con el Pod Identity Agent ya implementado, lo que permite conceder permisos de AWS IAM a los pods de forma simplificada:

- Administración simplificada de los permisos de IAM sin la configuración del proveedor de OIDC
- Reducción de la sobrecarga operativa en comparación con el IRSA
- Seguridad mejorada mediante el etiquetado de sesiones y la compatibilidad con ABAC

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

> **Important**
>
> Pod Identity es el enfoque recomendado para los permisos de IAM en el modo automático de EKS, ya que proporciona funciones de seguridad mejoradas y una administración simplificada en comparación con el IRSA.

Rol de IAM del nodo

El modo automático EKS utiliza un nuevo modo `AmazonEKSWorkerNodeMinimalPolicy` que solo proporciona los permisos necesarios para que funcionen los nodos del modo automático EKS. Estos permisos:

- Proporcionan un conjunto de permisos reducido en comparación con las políticas de nodos tradicionales
- Cumpla con el principio de privilegio mínimo
- AWS los administra y actualiza automáticamente

Este enfoque político mínimo ayuda a mejorar la postura de seguridad al limitar los permisos disponibles para el nodo y sus cargas de trabajo.

Seguridad de nodos

El modo automático EKS introduce varias mejoras de seguridad importantes a nivel de nodo:

EC2 Seguridad de instancias gestionadas

Los nodos de EKS Auto Mode utilizan Amazon EC2 Managed Instances con propiedades de seguridad mejoradas:

- Restricciones impuestas por IAM que impiden operaciones que podrían comprometer la capacidad de AWS para operar los nodos
- Patrones de infraestructura inmutables en los que los cambios de configuración requieren la sustitución de nodos
- Sustitución obligatoria de los nodos en un plazo de 21 días para garantizar las actualizaciones de seguridad periódicas
- Acceso restringido a los metadatos de la instancia IMDSv2 con límites de salto controlados

Seguridad del sistema operativo

El sistema operativo es una variante personalizada de Bottlerocket, optimizada para el modo automático EKS, que incluye:

- Sistema de archivos raíz de solo lectura
- SELinux activado de forma predeterminada con controles de acceso obligatorios
- Aislamiento automático de cápsulas mediante etiquetas SELinux MCS únicas
- Se ha desactivado el acceso a SSH y se han eliminado los servicios innecesarios
- Parches de seguridad automatizados mediante la rotación de nodos

Seguridad de los componentes de los nodos

Los componentes del nodo se configuran con las mejores prácticas de seguridad:

- Kubelet está configurado con valores predeterminados seguros
- Configuración reforzada en tiempo de ejecución de contenedores
- Gestión y rotación automatizadas de certificados
- node-to-control-planeComunicación restringida

Seguridad de la red

El modo automático de EKS implementa varias funciones de seguridad de red para garantizar una comunicación segura dentro del clúster y con recursos externos:

Política de red CNI de VPC

El modo automático de EKS aprovecha el soporte nativo de políticas de red de Kubernetes del complemento CNI de Amazon VPC:

- Se integra con la API original de políticas de red de Kubernetes
- Permite un control detallado de la comunicación pod-to-pod
- Soporta reglas de entrada y salida

Para habilitar el soporte de políticas de red en el modo automático de EKS, debe configurar el complemento CNI de VPC con un manifiesto. `configMap` A continuación se muestra un ejemplo:

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

También es necesario definir si el soporte de la política de red está configurado en la clase de nodos, como se muestra a continuación:

```
apiVersion: eks.amazonaws.com/v1
```

```
kind: NodeClass
metadata:
  name: example-node-class
spec:
  networkPolicy: DefaultAllow
  networkPolicyEventLogs: Enabled
```

Una vez activado, puede crear políticas de red para controlar el tráfico:

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

Gestión de ENI mejorada

El modo automático de EKS proporciona una seguridad mejorada para la administración de la interfaz de red elástica (ENI):

- Instalación y configuración de ENI gestionados por AWS
- Separación del tráfico de control del tráfico de datos
- Administración automatizada de direcciones IP con menos privilegios necesarios en los nodos

Seguridad del almacenamiento

El modo automático EKS proporciona funciones de seguridad mejoradas para el almacenamiento efímero y persistente:

Almacenamiento efímero

- Todos los datos escritos en volúmenes efímeros se cifran automáticamente
- Utiliza el algoritmo criptográfico AES-256 estándar de la industria
- El servicio gestiona sin problemas el cifrado y el descifrado

Volúmenes de EBS

- Los volúmenes de EBS raíz y de datos siempre están cifrados
- Los volúmenes están configurados para que se eliminen al finalizar la instancia
- Existe la opción de especificar claves KMS personalizadas para el cifrado

Integración de EFS

- Support para el cifrado en tránsito con EFS
- Cifrado automático en reposo para sistemas de archivos EFS
- Integración con puntos de acceso EFS para un mejor control de acceso

> **Important**
>
> Cuando utilice EFS con el modo automático de EKS, asegúrese de que los ajustes de cifrado adecuados estén configurados en el nivel del sistema de archivos de EFS, ya que el modo automático de EKS no administra el cifrado de EFS directamente.

Monitoreo y registro

El modo automático de EKS ofrece funciones mejoradas de supervisión y registro para ayudarle a mantener la visibilidad del estado operativo y del estado de seguridad del clúster.

Registro de plano de control

El modo automático EKS mantiene las mismas capacidades de registro en el plano de control que el EKS estándar, pero habilita todos los registros de forma predeterminada para mejorar la supervisión.

- Los registros se envían a Amazon CloudWatch Logs
- De forma predeterminada, el modo automático de EKS habilita todos los registros del plano de control: servidor API, auditoría, autenticador, administrador de controladores y programador
- El modo automático EKS permite una visibilidad detallada de las operaciones del clúster y los eventos de seguridad

> **Important**
>
> El registro en el plano de control conlleva costes adicionales por el almacenamiento de registros en CloudWatch él. Considere detenidamente su estrategia de registro para equilibrar las necesidades de seguridad con la administración de costos.

Registro a nivel de nodo

El modo automático EKS mejora el registro a nivel de nodo:

- Los registros del sistema se recopilan automáticamente y se puede acceder a ellos a través de los registros CloudWatch
- Los registros de los nodos se conservan incluso después de la finalización del nodo, lo que facilita el análisis posterior al incidente
- Visibilidad mejorada de los eventos de seguridad y los problemas operativos a nivel de nodo

GuardDuty Integración con Amazon

Los clústeres de modo automático de EKS se integran perfectamente con Amazon GuardDuty para mejorar la detección de amenazas. Entre sus características se incluyen:

- Escaneo automatizado de los registros de auditoría del plano de control
- Supervisión del tiempo de ejecución que se puede habilitar para la supervisión de las cargas de trabajo
- Integración con los GuardDuty hallazgos y los mecanismos de alerta existentes

Para habilitar la protección del modo automático de EKS en Amazon GuardDuty for Kubernetes Audit Logs, puede ejecutar el siguiente comando:

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

GuardDuty Integración de Amazon para seguridad en tiempo de ejecución

Amazon GuardDuty proporciona una supervisión de seguridad esencial en tiempo de ejecución para los clústeres de modo automático de EKS, lo que ofrece capacidades integrales de detección de

amenazas y supervisión de la seguridad. Esta integración es particularmente importante, ya que ayuda a identificar posibles amenazas de seguridad y actividades maliciosas en tiempo real.

GuardDuty Características principales del modo automático EKS

- Supervisión del tiempo de ejecución:
  - Supervisión continua del comportamiento en tiempo de ejecución
  - Detección de actividades maliciosas o sospechosas
  - Identificación de posibles intentos de fuga de contenedores
  - Supervisión de la ejecución de procesos o de las conexiones de red inusuales
- Detección de amenazas específicas de Kubernetes:
  - Identificación de intentos sospechosos de despliegue de cápsulas
  - Detección de contenedores comprometidos
  - Supervisión de los lanzamientos de contenedores privilegiados
  - Identificación del uso sospechoso de cuentas de servicio
- Tipos de hallazgos exhaustivos:
  - Política: Kubernetes/*: detecta las violaciones de las mejores prácticas de seguridad
  - Impacto: Kubernetes/*: identifica los recursos potencialmente afectados
  - Discovery: Kubernetes/*: detecta las actividades de reconocimiento
  - Ejecución:Kubernetes/*: identifica patrones de ejecución sospechosos
  - Persistencia: Kubernetes/*: detecta posibles amenazas persistentes

Para habilitar la protección del modo automático de EKS en Amazon GuardDuty for Kubernetes Audit Logs and Runtime Monitoring, puede ejecutar el siguiente comando:

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

> **Important**
>
> GuardDuty La monitorización del tiempo de ejecución se admite automáticamente en los clústeres de EKS Auto Mode, lo que proporciona una mayor visibilidad de la seguridad sin necesidad de realizar configuraciones adicionales a nivel de nodo.

GuardDuty Integración de los resultados

GuardDuty los hallazgos se pueden integrar con otros servicios de AWS para una respuesta automática:

- EventBridge Reglas:

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

- Integración de Security Hub:

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

Mejores prácticas para GuardDuty el modo automático EKS

1. Habilite todos los tipos de búsqueda:
   - Habilite tanto la supervisión del registro de auditoría de Kubernetes como la supervisión del tiempo de ejecución
   - Configure los resultados para todos los niveles de gravedad
2. Implemente una respuesta automatizada:

- Cree EventBridge reglas para los hallazgos de alta gravedad
- Intégrelo con AWS Security Hub para una administración de seguridad centralizada
- Configure acciones de remediación automatizadas cuando proceda

3. Revisión y ajuste periódicos:
   - Revise periódicamente GuardDuty los hallazgos
   - Ajuste los umbrales de detección en función de su entorno
   - Actualice los procedimientos de respuesta en función de los nuevos tipos de hallazgos
4. Gestión de cuentas cruzadas:
   - Considere la posibilidad de utilizar una cuenta de GuardDuty administrador para la administración centralizada
   - Habilite la agregación de los hallazgos en varias cuentas

> **Warning**
>
> Si bien GuardDuty proporciona una supervisión de seguridad integral, debe formar parte de una defense-in-depth estrategia que incluya otros controles de seguridad, como las políticas de red, los estándares de seguridad de los módulos y la configuración adecuada del RBAC.

Preguntas frecuentes (FAQ)

P: ¿En qué se diferencia el modo automático EKS del EKS estándar en términos de seguridad? R: El modo automático EKS proporciona una seguridad mejorada mediante instancias EC2 gestionadas, parches automatizados, rotación obligatoria de nodos y controles de seguridad integrados. Reduce la sobrecarga operativa y, al mismo tiempo, mantiene una sólida postura de seguridad al permitir que AWS administre más aspectos de seguridad.

P: ¿Puedo seguir utilizando las herramientas y políticas de seguridad existentes con el modo automático de EKS? R: Sí, el modo automático de EKS es compatible con la mayoría de las herramientas y políticas de seguridad existentes. Sin embargo, es posible que algunas herramientas de seguridad a nivel de nodo deban adaptarse debido a la naturaleza gestionada de los nodos del modo automático de EKS.

P: ¿Cómo puedo implementar los agentes de seguridad y las herramientas de monitoreo en el modo automático de EKS? R: En el modo automático de EKS, los agentes de seguridad y las herramientas

de supervisión se deben implementar como cargas de trabajo de Kubernetes (normalmente DaemonSets, se despliega una instancia del pod en cada nodo de forma predeterminada) en lugar de instalarse directamente en el sistema operativo del nodo. Este enfoque se alinea con el modelo de infraestructura inmutable del modo automático de EKS. Ejemplo:

```
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"]
```

P: ¿Las soluciones de seguridad de terceros son compatibles con el modo automático EKS?
R: Muchas soluciones de seguridad populares de terceros se han actualizado para que sean compatibles con el modo automático de EKS, pero siempre se recomienda comprobar con el proveedor de seguridad la versión específica y los requisitos de despliegue, ya que la compatibilidad con el modo automático de EKS puede requerir versiones actualizadas o configuraciones de despliegue específicas.

P: ¿Cuáles son las limitaciones de los agentes de seguridad en el modo automático de EKS? R: Las principales limitaciones incluyen:

- No hay acceso directo para modificar el sistema operativo del nodo
- No hay persistencia en las rotaciones de los nodos
- Debe ser compatible con la implementación basada en contenedores

- Debe respetarse la inmutabilidad de los nodos
- Puede requerir diferentes configuraciones de privilegios
- Cualquier cambio persistente en los nodos debe realizarse a través de `NodePools` los `NodeClasses` recursos.

> **Note**
>
> Si bien el modo automático de EKS puede requerir ajustes en la estrategia de despliegue de las herramientas de seguridad, estos cambios suelen dar como resultado configuraciones más fáciles de mantener y seguras, alineadas con las mejores prácticas nativas de la nube. Se espera que EKS Auto Mode absorba por completo la mayoría de las funciones que gestiona. Por lo tanto, cualquier cambio manual que realices en esas funciones, si es que puedes acceder a ellas, podría ser sobrescrito o descartado por el Modo Automático EKS.

P: ¿Puedo usar el modo personalizado AMIs con el modo automático EKS? R: En este momento, el modo automático EKS no admite el modo personalizado AMIs. Esto se debe a su diseño, ya que AWS administra la seguridad, los parches y el mantenimiento de los nodos como parte del modelo de responsabilidad compartida. Los nodos del modo automático de EKS utilizan una variante especializada de Bottlerocket optimizada y mantenida por AWS.

P: ¿Con qué frecuencia se giran automáticamente los nodos en el modo automático EKS? R: Los nodos en el modo automático EKS tienen una vida útil máxima de 21 días. Se reemplazarán automáticamente antes de este límite, lo que garantizará la aplicación de parches y las actualizaciones de seguridad periódicas.

P: ¿Puedo utilizar SSH en los nodos del modo automático de EKS para solucionar problemas? R: No, el acceso SSH directo no está disponible en el modo automático EKS. En su lugar, puede utilizar la definición de recursos NodeDiagnostic personalizada (CRD) para recopilar los registros del sistema y la información de depuración.

P: ¿La compatibilidad con las políticas de red está habilitada de forma predeterminada en el modo automático de EKS? R: Por ahora, el soporte de políticas de red debe habilitarse de forma explícita a través de la configuración del complemento CNI de la VPC. Una vez activado, puedes usar las políticas de red estándar de Kubernetes.

P: ¿Debo usar IRSA o Pod Identity con el modo automático EKS? R: Si bien ambos son compatibles, Pod Identity es el enfoque recomendado en el modo automático de EKS, ya que ya incluye el

complemento de agente Pod Identity Security y ofrece funciones de seguridad mejoradas y una administración simplificada.

P: ¿Puedo seguir utilizando el aws-auth ConfigMap en el modo EKS Auto? R: Se `aws-auth` ConfigMap trata de una función obsoleta. Se recomienda utilizar el enfoque predeterminado de autenticación basada en API para mejorar la seguridad y simplificar la administración del acceso.

P: ¿Cómo puedo monitorear los eventos de seguridad en el modo automático de EKS? R: El modo automático EKS se integra con múltiples soluciones de monitoreo GuardDuty, que incluyen CloudWatch, y CloudTrail. GuardDuty proporciona una supervisión mejorada de la seguridad en tiempo de ejecución específica para las cargas de trabajo de EKS.

P: ¿Cómo recopilo los registros de los nodos del modo automático de EKS? R: Utilice el NodeDiagnostic CRD, que carga automáticamente los registros a un bucket de S3. También puede utilizar CloudWatch Container Insights y AWS Distro para OpenTelemetry.

> **Note**
>
> Esta sección de preguntas frecuentes se actualiza periódicamente a medida que se añaden nuevas funciones al modo automático de EKS y recibimos preguntas frecuentes de los clientes.

Gestión de identidad y acceso

Identity and Access Management (IAM) es un servicio de AWS que realiza dos funciones esenciales: autenticación y autorización. La autenticación implica la verificación de una identidad, mientras que la autorización regula las acciones que pueden realizar los recursos de AWS. Dentro de AWS, un recurso puede ser otro servicio de AWS, por ejemplo EC2, o un director de AWS, como un usuario o rol de IAM. Las normas que rigen las acciones que un recurso puede realizar se expresan como políticas de IAM.

Control del acceso a los clústeres de EKS

El proyecto Kubernetes admite una variedad de estrategias diferentes para autenticar las solicitudes del servicio kube-apiserver, por ejemplo, los tokens Bearer, los certificados X.509, el OIDC, etc. Actualmente, EKS cuenta con soporte nativo para la autenticación mediante token webhook, para los tokens de cuentas de servicio y, a partir del 21 de febrero de 2021, para la autenticación OIDC.

La estrategia de autenticación de webhook denomina webhook a un webhook que verifica los tokens portadores. En EKS, estos tokens portadores los genera la AWS CLI o el aws-iam-authenticatorcliente cuando se ejecutan `kubectl` comandos. A medida que ejecuta los comandos, el token se pasa al kube-apiserver, que lo reenvía al webhook de autenticación. Si la solicitud está bien formada, el webhook llama a una URL prefirmada incrustada en el cuerpo del token. Esta URL valida la firma de la solicitud y devuelve información sobre el usuario, por ejemplo, la cuenta del usuario, Arn, y al kube-apiserver. UserId

Para generar manualmente un token de autenticación, escribe el siguiente comando en una ventana de terminal:

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

El resultado debe ser similar al siguiente:

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

También puede obtener un token mediante programación. A continuación se muestra un ejemplo escrito en Go:

```
package main

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

func main()  {
  g, _ := token.NewGenerator(false, false)
  tk, err := g.Get("<cluster_name>")
```

```
  if err != nil {
    log.Fatal(err)
  }
  fmt.Println(tk)
}
```

El resultado debería ser similar al siguiente:

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

Cada token comienza `k8s-aws-v1.` seguido de una cadena codificada en base64. La cadena, una vez decodificada, debería parecerse a algo parecido a esto:

```
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
```

El token consiste en una URL prefirmada que incluye una credencial y una firma de Amazon. Para obtener más información, consulta https://docs.aws.amazon.com/STS/ latest/APIReference/API _ GetCallerIdentity .html.

El token tiene un tiempo de vida (TTL) de 15 minutos, tras lo cual será necesario generar un nuevo token. Esto se gestiona automáticamente cuando utilizas un cliente, por ejemplo`kubectl`, si utilizas el panel de control de Kubernetes, tendrás que generar un nuevo token y volver a autenticarte cada vez que el token caduque.

Una vez que el servicio IAM de AWS haya autenticado la identidad del usuario, el kube-apiserver la lee `aws-auth` ConfigMap en el espacio de `kube-system` nombres para determinar el grupo RBAC que se va a asociar al usuario. `aws-auth` ConfigMap Se utiliza para crear un mapeo estático entre

los principales de IAM, es decir, los usuarios y roles de IAM, y los grupos RBAC de Kubernetes. Se puede hacer referencia a los grupos RBAC en Kubernetes o. RoleBindings ClusterRoleBindings Son similares a las funciones de IAM en el sentido de que definen un conjunto de acciones (verbos) que se pueden realizar en un conjunto de recursos (objetos) de Kubernetes.

CloudWatch consulta para ayudar a los usuarios a identificar a los clientes que envían solicitudes al punto final global de STS

Ejecute la siguiente CloudWatch consulta para obtener el punto final de STS. Si stsendpoint es igual a «sts.amazonaws.com», se trata de un punto final STS global. Si stendpoint es igual a «sts». <region>.amazonaws.com», entonces es un punto final STS regional.

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

Administrador de acceso a clústeres

Cluster Access Manager, que ahora es la forma preferida de administrar el acceso de los principales de AWS IAM a los clústeres de Amazon EKS, es una funcionalidad de la API de AWS y es una función opcional para los clústeres de EKS v1.23 y versiones posteriores (nuevos o existentes). Simplifica el mapeo de identidades entre AWS IAM y KubernetesRBACs, lo que elimina la necesidad de cambiar entre AWS y Kubernetes APIs o de editarlo `aws-auth` ConfigMap para la administración del acceso, reduce la sobrecarga operativa y ayuda a corregir los errores de configuración. La herramienta también permite a los administradores de clústeres revocar o restringir `cluster-admin` los permisos concedidos automáticamente al director de AWS IAM utilizado para crear el clúster.

Esta API se basa en dos conceptos:

- Entradas de acceso: una identidad de clúster vinculada directamente a un principal de AWS IAM (usuario o rol) que permite autenticarse en un clúster de Amazon EKS.
- Políticas de acceso: son políticas específicas de Amazon EKS que proporcionan la autorización para que una entrada de acceso realice acciones en el clúster de Amazon EKS.

En el momento del lanzamiento, Amazon EKS solo admite políticas predefinidas y administradas por AWS. Las políticas de acceso no son entidades de IAM y las define y administra Amazon EKS.

Cluster Access Manager permite combinar el RBAC previo con políticas de acceso que permiten permitir y transferir (pero no denegar) las decisiones de AuthZ de Kubernetes en relación con las solicitudes de servidores de API. Se tomará una decisión de denegación cuando los autorizadores anteriores del RBAC y Amazon EKS no puedan determinar el resultado de la evaluación de una solicitud.

Con esta función, Amazon EKS admite tres modos de autenticación:

1. `CONFIG_MAP`para seguir usando `aws-auth` ConfigMap exclusivamente.
2. `API_AND_CONFIG_MAP`para obtener los principios de IAM autenticados tanto de la entrada de acceso de EKS como de ConfigMap`aws-auth`, priorizando APIs las entradas de acceso. Ideal para migrar los permisos existentes a las entradas de acceso. `aws-auth`
3. `API`para confiar exclusivamente en EKS Access Entry APIs. Este es el nuevo enfoque recomendado.

Para empezar, los administradores de clústeres pueden crear o actualizar los clústeres de Amazon EKS, configurar el `API` método de autenticación preferido y definir entradas de acceso para conceder el acceso a los principales de AWS IAM deseados. `API_AND_CONFIG_MAP`

```
$ 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
```

El comando anterior es un ejemplo de cómo crear un clúster de Amazon EKS que ya no tiene los permisos de administrador del creador del clúster.

Es posible actualizar la configuración de los clústeres de Amazon EKS para habilitar `API` AuthenticationMode mediante el `update-cluster-config` comando. Para hacerlo en los clústeres existentes, primero `CONFIG_MAP` tendrá que actualizar a `API_AND_CONFIG_MAP` y luego a. `API` Estas operaciones no se pueden revertir, lo que significa que no es posible cambiar de `API` a `API_AND_CONFIG_MAP` o ni `CONFIG_MAP` tampoco de a. `API_AND_CONFIG_MAP` `CONFIG_MAP`

```
$ aws eks update-cluster-config \
```

```
    --name <CLUSTER_NAME> \
    --access-config authenticationMode=API
```

La API admite comandos para añadir y revocar el acceso al clúster, así como para validar las políticas de acceso y las entradas de acceso existentes para el clúster especificado. Las políticas predeterminadas se crean para que coincidan con RBACs Kubernetes de la siguiente manera.

Política de acceso a EKS	RBAC de Kubernetes
Amazon EKSCluster AdminPolicy	administrador de clústeres
EKSAdminPolítica de Amazon	administrador
EKSEditPolítica de Amazon	editar
EKSViewPolítica de Amazon	ver

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

No hay entradas de acceso disponibles cuando el clúster se crea sin el permiso de administrador del creador del clúster, que es la única entrada que se crea de forma predeterminada.

El **aws-auth** ConfigMap (obsoleto)

Una forma de realizar la integración de Kubernetes con la autenticación de AWS es a través del `aws-auth` ConfigMap, que reside en el espacio de nombres. `kube-system` Es responsable de asignar la autenticación de las identidades de IAM (usuarios, grupos y roles) de AWS a la autorización del control de acceso basado en roles (RBAC) de Kubernetes. `aws-auth` ConfigMap Se crea automáticamente en su clúster de Amazon EKS durante la fase de aprovisionamiento. Se creó inicialmente para permitir que los nodos se unieran a su clúster, pero como se ha mencionado, también puede usarlo ConfigMap para añadir RBACs acceso a los principios de IAM.

Para comprobar el clúster `aws-auth` ConfigMap, puede utilizar el siguiente comando.

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

Este es un ejemplo de la configuración predeterminada del `aws-auth`ConfigMap.

```
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      - system:node-proxier
      rolearn: arn:aws:iam::<AWS_ACCOUNT_ID>:role/kube-system-<SELF_GENERATED_UUID>
      username: system:node:{{SessionName}}
kind: ConfigMap
metadata:
  creationTimestamp: "2023-10-22T18:19:30Z"
  name: aws-auth
  namespace: kube-system
```

La sesión principal de esta ConfigMap, se encuentra debajo `data` del `mapRoles` bloque, que está compuesto básicamente por 3 parámetros.

- grupos: los Kubernetes group/groups a los que asignar el rol de IAM. Puede ser un grupo predeterminado o un grupo personalizado especificado en una o. `clusterrolebinding` `rolebinding` En el ejemplo anterior, solo hemos declarado los grupos del sistema.
- rolearn: El ARN del rol de IAM de AWS debe asignarse al grupo o grupos agregados de Kubernetes con el siguiente formato. `arn:<PARTITION>:iam::<AWS_ACCOUNT_ID>:role/role-name`
- username: el nombre de usuario de Kubernetes que se asigna a la función de IAM de AWS. Puede ser cualquier nombre personalizado.

También es posible mapear los permisos para los usuarios de AWS IAM, definiendo un nuevo bloque de configuración para`mapUsers`, `data` en el `aws-auth`ConfigMap, reemplazar el parámetro rolearn por userarn. Sin embargo, como práctica recomendada, siempre se recomienda al usuario en su lugar. `mapRoles`

Para administrar los permisos, puede editar el acceso para `aws-auth` ConfigMap añadir o eliminar el acceso a su clúster de Amazon EKS. Si bien es posible editarlo `aws-auth` ConfigMap manualmente, se recomienda utilizar herramientas como estas`eksctl`, ya que se trata de una configuración muy delicada y una configuración imprecisa puede bloquearlo fuera de su Amazon EKS Cluster. Consulte la subsección Uso de herramientas para realizar cambios en el ConfigMap aws-auth que aparece a continuación para obtener más información.

Ventajas de la gestión de acceso basada en bases ConfigMap

1. Menor riesgo de errores de configuración: la administración directa basada en la API elimina los errores comunes asociados a la edición manual ConfigMap . Esto ayuda a evitar eliminaciones accidentales o errores de sintaxis que podrían impedir que los usuarios accedan al clúster.
2. Principio de privilegios mínimos mejorado: elimina la necesidad del permiso de administrador del clúster de la identidad del creador del clúster y permite una asignación de permisos más detallada y adecuada. Puede optar por añadir este permiso para los casos de uso de breakglass.
3. Modelo de seguridad mejorado: proporciona una validación integrada de las entradas de acceso antes de que se apliquen. Además, ofrece una integración más estrecha con AWS IAM para la autenticación.
4. Operaciones simplificadas: ofrece una forma más intuitiva de administrar los permisos mediante herramientas nativas de AWS.

Recomendaciones de acceso a clústeres

Combine el centro de identidad de IAM con la API de CAM

- Administración simplificada: al usar la API de administración de acceso a clústeres junto con IAM Identity Center, los administradores pueden administrar el acceso a los clústeres de EKS junto con otros servicios de AWS, lo que reduce la necesidad de cambiar entre diferentes interfaces o editar ConfigMaps manualmente.
- Utilice las entradas de acceso para administrar los permisos de Kubernetes de las entidades principales de IAM ajenos al clúster. Puede añadir y administrar el acceso al clúster mediante la API de EKS, la interfaz de línea de comandos de AWS, AWS SDKs CloudFormation, AWS y la consola de administración de AWS. Esto significa que puede administrar los usuarios con las mismas herramientas con las que creó el clúster.
- Aproveche la automatización, como se muestra en este ejemplo, para implementar clústeres con AWS IAM Identity Center como IdP y con la API CAM como punto de entrada.
- Los permisos detallados de Kubernetes se pueden aplicar mediante la asignación de usuarios o grupos de Kubernetes con los principios de IAM asociados a las identidades de SSO mediante entradas de acceso y políticas de acceso.
- Para empezar, siga Cambiar el modo de autenticación para usar entradas de acceso y, a continuación, Migrar las entradas de aws-auth existentes a entradas de acceso. ConfigMap

Haga que el punto final del clúster EKS sea privado

De forma predeterminada, al aprovisionar un clúster de EKS, el punto final del clúster de API está configurado como público, es decir, se puede acceder a él desde Internet. A pesar de ser accesible desde Internet, el punto final sigue considerándose seguro porque requiere que todas las solicitudes de API sean autenticadas por IAM y luego autorizadas por el RBAC de Kubernetes. Dicho esto, si su política de seguridad corporativa exige que restrinja el acceso a la API desde Internet o le impide enrutar el tráfico fuera de la VPC del clúster, puede:

- Configure el punto final del clúster EKS para que sea privado. Consulte Modificación del acceso al punto final del clúster para obtener más información sobre este tema.
- Deje el punto final del clúster público y especifique qué bloques CIDR pueden comunicarse con el punto final del clúster. En efecto, los bloques son un conjunto de direcciones IP públicas incluidas en la lista blanca que pueden acceder al punto final del clúster.

- Configure el acceso público con un conjunto de bloques de CIDR incluidos en la lista blanca y active el acceso a los terminales privados. Esto permitirá el acceso público desde un rango específico de públicos y, al IPs mismo tiempo, forzará todo el tráfico de red entre los kubelets (trabajadores) y la API de Kubernetes a través de las cuentas cruzadas ENIs que se aprovisionan en la VPC del clúster cuando se aprovisiona el plano de control.

No utilices un token de cuenta de servicio para la autenticación

Un token de cuenta de servicio es una credencial estática de larga duración. Si se ve comprometida, se pierde o es robada, es posible que un atacante pueda realizar todas las acciones asociadas a ese token hasta que se elimine la cuenta de servicio. En ocasiones, es posible que tengas que conceder una excepción a las aplicaciones que tienen que consumir la API de Kubernetes desde fuera del clúster, por ejemplo, una aplicación de canalización de CI/CD. Si dichas aplicaciones se ejecutan en la infraestructura de AWS, como EC2 instancias, considere la posibilidad de usar un perfil de instancia y asignarlo a una función de RBAC de Kubernetes.

Emplee el acceso menos privilegiado a los recursos de AWS

No es necesario asignar privilegios a los recursos de AWS a un usuario de IAM para acceder a la API de Kubernetes. Si necesita conceder a un usuario de IAM acceso a un clúster de EKS, cree una entrada `aws-auth` ConfigMap para ese usuario que se asigne a un grupo RBAC de Kubernetes específico.

Elimine los permisos de administrador del clúster del principal creador del clúster

De forma predeterminada, los clústeres de Amazon EKS se crean con un `cluster-admin` permiso permanente vinculado al creador principal del clúster. Con la API Cluster Access Manager, es posible crear clústeres sin este permiso, configurando el `--access-config bootstrapClusterCreatorAdminPermissions` modo de `API` autenticación`false`, cuándo se utiliza `API_AND_CONFIG_MAP` o si se utiliza. Revocar este acceso se considera una práctica recomendada para evitar cambios no deseados en la configuración del clúster. El proceso para revocar este acceso sigue el mismo proceso para revocar cualquier otro acceso al clúster.

La API le brinda la flexibilidad de disociar únicamente un director de IAM de una política de acceso, en este caso, la. `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
```

O eliminar por completo la entrada de acceso asociada al `cluster-admin` permiso.

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

{
    "accessEntries": []
}

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

Este acceso se puede volver a conceder si es necesario en caso de incidente, emergencia o rotura de un cristal en el que el clúster quede inaccesible de otro modo.

Si el clúster sigue configurado con el método de `CONFIG_MAP` autenticación, se debe conceder acceso al clúster a todos los usuarios adicionales a través de la `aws-auth` ConfigMap función asignada a la entidad que creó el clúster y, una vez `aws-auth` ConfigMap configurada, se puede eliminar y solo volver a crear en caso de incidente, emergencia o rotura de cristales, o cuando `aws-auth` ConfigMap esté dañado y el clúster sea inaccesible de otro modo. Esto puede resultar especialmente útil en los clústeres de producción.

Utilice las funciones de IAM cuando varios usuarios necesiten un acceso idéntico al clúster

En lugar de crear una entrada para cada usuario de IAM individual, permita que esos usuarios asuman un rol de IAM y asignen ese rol a un grupo RBAC de Kubernetes. Esto será más fácil de mantener, especialmente a medida que aumente el número de usuarios que requieren acceso.

> **Important**
>
> Al acceder al clúster de EKS con la entidad de IAM mapeada por ella `aws-auth` ConfigMap, el nombre de usuario descrito se registra en el campo de usuario del registro de auditoría de Kubernetes. Si utilizas un rol de IAM, los usuarios reales que asumen ese rol no se registran y no se pueden auditar.

Si sigue utilizando `aws-auth` ConfigMap como método de autenticación, al asignar permisos RBAC de K8s a un rol de IAM, debe incluir\ {{}} en su nombre de usuario. SessionName De esta forma, el registro de auditoría registrará el nombre de la sesión para que pueda hacer un seguimiento de quién es el usuario real que asume esta función junto con el registro. CloudTrail

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

> En Kubernetes 1.20 y versiones posteriores, este cambio ya no es necesario, ya que `user.extra.sessionName.0` se agregó al registro de auditoría de Kubernetes.

Emplee el acceso menos privilegiado al crear y RoleBindings ClusterRoleBindings

Como en el punto anterior sobre la concesión de acceso a los recursos de AWS, RoleBindings y solo ClusterRoleBindings debe incluir el conjunto de permisos necesarios para realizar una función específica. Evite `["*"]` utilizarlos en sus funciones y ClusterRoles a menos que sea absolutamente necesario. Si no estás seguro de qué permisos asignar, considera usar una herramienta como audit2rbac para generar automáticamente las funciones y los enlaces en función de las llamadas a la API observadas en el registro de auditoría de Kubernetes.

Crea un clúster mediante un proceso automatizado

Como se ha visto en los pasos anteriores, al crear un clúster de Amazon EKS, si no se utiliza el modo de uso `API_AND_CONFIG_MAP` o `API` autenticación y no se opta por no delegar `cluster-admin` los permisos al creador del clúster, al usuario o rol de la entidad de IAM, como el usuario federado que crea el clúster, se le conceden automáticamente `system:masters` permisos en la configuración RBAC del clúster. A pesar de ser una buena práctica eliminar este permiso, como se describe aquí, si se utiliza el método de `CONFIG_MAP` autenticación, se basa en `aws-auth` ConfigMap él, este acceso no se puede revocar. Por lo tanto, es una buena idea crear el clúster con una canalización de automatización de infraestructuras vinculada a una función específica de IAM, sin permisos que puedan ser asumidos por otros usuarios o entidades, y auditar periódicamente los permisos, las políticas y las personas que tienen acceso a esta función para activar la canalización. Además, esta función no debe usarse para realizar acciones rutinarias en el clúster, sino que debe usarse exclusivamente para acciones a nivel de clúster activadas por la canalización, por ejemplo, mediante cambios en el código de SCM.

Cree el clúster con una función de IAM dedicada

Al crear un clúster de Amazon EKS, al usuario o rol de la entidad de IAM, como un usuario federado que crea el clúster, se le conceden automáticamente `system:masters` permisos en la configuración RBAC del clúster. Este acceso no se puede eliminar y no se administra a través de. `aws-auth` ConfigMap Por lo tanto, es una buena idea crear el clúster con una función de IAM dedicada y auditar periódicamente quién puede asumir esta función. Esta función no se debe utilizar para realizar acciones rutinarias en el clúster, sino que se debe conceder acceso al clúster a otros usuarios a través del mismo con este `aws-auth` ConfigMap fin. Una vez `aws-auth` ConfigMap configurado, el rol debe estar protegido y solo debe usarse en el modo de privilegios elevados o rotura de cristales temporales en situaciones en las que el clúster sea inaccesible de otro modo. Esto puede resultar particularmente útil en clústeres que no tienen configurado el acceso directo de los usuarios.

Audite periódicamente el acceso al clúster

Es probable que las personas que necesitan el acceso cambien con el tiempo. Planifique auditarlo periódicamente `aws-auth` ConfigMap para ver a quién se le ha concedido el acceso y los derechos que se le han asignado. También puedes usar herramientas de código abierto kubectl-who-can, como rbac-lookup, para examinar las funciones vinculadas a una cuenta de servicio, usuario o grupo en particular. Exploraremos este tema con más detalle cuando lleguemos a la sección sobre auditoría. Puede encontrar ideas adicionales en este artículo de NCC Group.

Si confía en **`aws-auth`** ConfigMap, use herramientas para realizar cambios.

Un aws-auth mal formateado ConfigMap puede provocar que pierda el acceso al clúster. Si necesita realizar cambios en, utilice una herramienta. ConfigMap

eksctl La `eksctl` CLI incluye un comando para añadir mapeos de identidad a aws-auth. ConfigMap

Ver la ayuda de CLI:

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

Compruebe las identidades asignadas a su clúster de Amazon EKS.

```
$ eksctl get iamidentitymapping --cluster $CLUSTER_NAME --region $AWS_REGION
```

```
ARN                                                                 USERNAME
               GROUPS                                                ACCOUNT
arn:aws:iam::788355785855:role/kube-system-<SELF_GENERATED_UUID>      system:node:
{{SessionName}}     system:bootstrappers,system:nodes,system:node-proxier
```

Convierta a un rol de IAM en administrador de clústeres:

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

Para obtener más información, consulte los documentos `eksctl`

aws-auth de keikoproj

`aws-auth`de keikoproj incluye una biblioteca cli y una biblioteca go.

Descargue y vea la ayuda de la CLI:

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

Como alternativa, instálalo `aws-auth` con el administrador de complementos krew para kubectl.

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

Consulta la documentación de aws-auth GitHub para obtener más información, incluida la biblioteca go.

AWS IAM Authenticator CLI

El `aws-iam-authenticator` proyecto incluye una CLI para actualizar elConfigMap.

Descargue una versión el GitHub.

Añada permisos de clúster a un rol de IAM:

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

Enfoques alternativos para la administración de la autenticación y el acceso

Si bien la IAM es la forma preferida de autenticar a los usuarios que necesitan acceder a un clúster de EKS, es posible utilizar un proveedor de identidad OIDC, por ejemplo, GitHub mediante un proxy de autenticación y la suplantación de identidad de Kubernetes. Se han publicado artículos sobre dos de estas soluciones en el blog de código abierto de AWS:

- Autenticación en EKS mediante GitHub credenciales con Teleport
- Autenticación OIDC coherente en varios clústeres de EKS mediante kube-oidc-proxy

> **Important**
>
> EKS admite de forma nativa la autenticación OIDC sin usar un proxy. Para obtener más información, lea el blog de lanzamiento, Introducción a la autenticación de proveedores de identidad OIDC para Amazon EKS. Para ver un ejemplo que muestra cómo configurar EKS con Dex, un popular proveedor de OIDC de código abierto con conectores para distintos métodos de autenticación, consulte Uso de Dex y dex-k8s-authenticator para autenticarse en Amazon EKS. Como se describe en los blogs, los usuarios autenticados por un proveedor de OIDC aparecerán en el username/group registro de auditoría de Kubernetes.

También puede usar AWS SSO para federar AWS con un proveedor de identidad externo, por ejemplo, Azure AD. Si decide usarlo, la AWS CLI v2.0 incluye una opción para crear un perfil con nombre que le permite asociar fácilmente una sesión de SSO a su sesión de CLI actual y asumir una función de IAM. Tenga en cuenta que debe asumir una función antes de correr, `kubectl` ya que la función de IAM se utiliza para determinar el grupo RBAC de Kubernetes del usuario.

Identidades y credenciales para los pods de EKS

Algunas aplicaciones que se ejecutan dentro de un clúster de Kubernetes necesitan permiso para llamar a la API de Kubernetes para funcionar correctamente. Por ejemplo, el controlador Load Balancer de AWS debe poder enumerar los puntos de enlace de un servicio. El controlador

también debe poder invocar a AWS para APIs aprovisionar y configurar un ALB. En esta sección analizaremos las prácticas recomendadas para asignar derechos y privilegios a los pods.

Cuentas de servicio de Kubernetes

Una cuenta de servicio es un tipo especial de objeto que permite asignar una función RBAC de Kubernetes a un pod. Se crea automáticamente una cuenta de servicio predeterminada para cada espacio de nombres de un clúster. Al implementar un pod en un espacio de nombres sin hacer referencia a una cuenta de servicio específica, la cuenta de servicio predeterminada de ese espacio de nombres se asignará automáticamente al pod y el secreto, es decir, el token de la cuenta de servicio (JWT) de esa cuenta de servicio, se montará en el pod como un volumen en. `/var/run/secrets/kubernetes.io/serviceaccount` Al decodificar el token de la cuenta de servicio en ese directorio, se mostrarán los siguientes metadatos:

```
{
  "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"
}
```

La cuenta de servicio predeterminada tiene los siguientes permisos para la API de Kubernetes.

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

```
  - /api/*
  - /apis
  - /apis/*
  - /healthz
  - /openapi
  - /openapi/*
  - /version
  - /version/
  verbs:
  - get
```

Esta función autoriza a los usuarios autenticados y no autenticados a leer la información de la API y se considera segura si es de acceso público.

Cuando una aplicación que se ejecuta en un pod llama a Kubernetes APIs, es necesario asignar al pod una cuenta de servicio que le conceda permiso explícito para llamarlos. APIs De forma similar a las directrices para el acceso de los usuarios, el rol ClusterRole o el vínculo a una cuenta de servicio debe restringirse a los recursos y métodos de la API que la aplicación necesita para funcionar, y nada más. Para usar una cuenta de servicio que no sea la predeterminada, simplemente establece en el `spec.serviceAccountName` campo de un pod el nombre de la cuenta de servicio que deseas usar. Para obtener información adicional sobre la creación de cuentas de servicio, consulta https://kubernetes. io/docs/reference/access-authn-authz/rbac/#. service-account-permissions

> **Note**
>
> Antes de la versión 1.24 de Kubernetes, Kubernetes creaba automáticamente un secreto para cada cuenta de servicio. Este secreto se montaba en el pod en/var/run/secrets/kubernetes.io/serviceaccounty el pod lo utilizaba para autenticarse en el servidor API de Kubernetes. En Kubernetes 1.24, un token de cuenta de servicio se genera dinámicamente cuando el pod se ejecuta y, de forma predeterminada, solo es válido durante una hora. No se creará un secreto para la cuenta de servicio. Si tienes una aplicación que se ejecuta fuera del clúster y necesita autenticarse en la API de Kubernetes, como Jenkins, tendrás que crear un tipo de secreto `kubernetes.io/service-account-token` junto con una anotación que haga referencia a la cuenta de servicio, por ejemplo. `metadata.annotations.kubernetes.io/service-account.name: <SERVICE_ACCOUNT_NAME>` Los secretos creados de esta manera no caducan.

Funciones de IAM para cuentas de servicio (IRSA)

El IRSA es una función que permite asignar una función de IAM a una cuenta de servicio de Kubernetes. Funciona aprovechando una función de Kubernetes conocida como proyección del volumen de tokens de cuentas de servicio. Cuando los pods se configuran con una cuenta de servicio que hace referencia a un rol de IAM, el servidor de la API de Kubernetes llamará al punto de conexión público del OIDC para detectar el clúster al iniciarse. El punto final firma criptográficamente el token OIDC emitido por Kubernetes y el token resultante se monta como un volumen. Este token firmado permite al Pod llamar a la función de IAM APIs asociada a AWS. Cuando se invoca una API de AWS, AWS SDKs llama`sts:AssumeRoleWithWebIdentity`. Tras validar la firma del token, IAM cambia el token emitido por Kubernetes por una credencial de rol de AWS temporal.

Al usar IRSA, es importante reutilizar las sesiones del SDK de AWS para evitar llamadas innecesarias a AWS STS.

Al decodificar el token (JWT) para IRSA, se obtendrá un resultado similar al del ejemplo que se muestra a continuación:

```
{
  "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"
}
```

Este token en concreto otorga al Pod privilegios de solo visualización a S3 al asumir una función de IAM. Cuando la aplicación intenta leer desde S3, el token se cambia por un conjunto temporal de credenciales de IAM similar al siguiente:

```
{
    "AssumedRoleUser": {
        "AssumedRoleId": "AROA36C6WWEJULFUYMPB6:abc",
        "Arn": "arn:aws:sts::123456789012:assumed-role/eksctl-winterfell-addon-
iamserviceaccount-de-Role1-1D61LT75JH3MB/abc"
    },
    "Audience": "sts.amazonaws.com",
    "Provider": "arn:aws:iam::123456789012:oidc-provider/oidc.eks.us-
west-2.amazonaws.com/id/D43CF17C27A865933144EA99A26FB128",
    "SubjectFromWebIdentityToken": "system:serviceaccount:default:s3-read-only",
    "Credentials": {
        "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
        "SessionToken": "FwoGZXIvYXdzEGMaDMLxAZkuLpmSwYXShiL9A1S0X87VBC1mHCrRe/
pB2oesl1eXxUYnPJyC9ayOoXMvqXQsomq0xs6OqZ3vaa5Iw1HIyA4Cv1suLaOCoU3hNvOIJ6C94H1vU0siQYk7DIq9Av5RZ
Zv2A6zp5xGz9cWj2f0aD9v66vX4bexOs5t/YYhwuwAvkkJPSIGvxja0xRThnceHyFHKtj0Hbi/
PWAtlI8YJcDX69cM30JAHDdQHltm/4scFptW1hlvMaPWReCAaCrsHrATyka7ttw5YlUyvZ8EPogj6fwHlxmrXM9h1Bqdiko
LgkWsCTGlYcY8z3zkigJMbYn07ewTL5Ss7LazTJJa758I7PZan/
v3xQHd5DEc5WBneiV3iOznDFgup0VAMkIviVjVCkszaPSVEdK2NU7jtrh6Jfm7bU/3P6ZGCkyDLIa8MBn9KPXeJd/
yjTk5IifIwO/mDpGNUribg6TPxhzZ8b/XdZO1kS1gVgqjXyVCM+BRBh6C4H21w/eMzjCtDIpoxt5rGKL6Nu/
IFMipoC4fgx6LIIHwtGYMG7SWQi7OsMAkiwZRg0n68/RqWgLzBt/4pfjSRYuk=",
        "Expiration": "2020-02-20T18:49:50Z",
        "AccessKeyId": "ASIAIOSFODNN7EXAMPLE"
    }
}
```

Un webhook mutante que se ejecuta como parte del plano de control de EKS inyecta el ARN del rol de AWS y la ruta a un archivo de token de identidad web en el pod como variables de entorno. Estos valores también se pueden proporcionar manualmente.

```
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
```

El kubelet rotará automáticamente el token proyectado cuando tenga más del 80% de su TTL total o después de 24 horas. La AWS SDKs es responsable de volver a cargar el token cuando gira. Para obtener más información sobre IRSA, consulte https://docs.aws.amazon.com/eks/ latest/userguide/ iam - -technical-overview.html. roles-for-service-accounts

Pod Identities de EKS

EKS Pod Identities es una función lanzada en re:Invent 2023 que le permite asignar una función de IAM a una cuenta de servicio de Kubernetes, sin necesidad de configurar un proveedor de identidades (IDP) de Open Id Connect (OIDC) para cada clúster de su cuenta de AWS. Para utilizar EKS Pod Identity, debe implementar un agente que se ejecute como un pod en todos los nodos de trabajo aptos. DaemonSet Este agente está disponible como un complemento de EKS y es un requisito previo para utilizar la función EKS Pod Identity. Sus aplicaciones deben usar una versión compatible del SDK de AWS para usar esta función.

Cuando las identidades de pod de EKS estén configuradas para un pod, EKS montará y actualizará un token de identidad de pod en`/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token`. El SDK de AWS utilizará este token para comunicarse con el agente de identidad del pod de EKS, que utiliza el token de identidad del pod y la función de IAM del agente para crear credenciales temporales para los pods mediante una llamada a la AssumeRoleForPodIdentityAPI. El token de identidad del pod que se entrega a sus pods es un JWT emitido desde su clúster de EKS y firmado criptográficamente, junto con las declaraciones de JWT correspondientes para su uso con las identidades de los pods de EKS.

Para obtener más información sobre las identidades de los pods de EKS, consulta este blog.

No es necesario realizar ninguna modificación en el código de la aplicación para utilizar EKS Pod Identities. Las versiones de AWS SDK compatibles descubrirán automáticamente las credenciales disponibles con EKS Pod Identities mediante la cadena de proveedores de credenciales. Al igual que IRSA, las identidades de los pods de EKS establecen variables dentro de los pods para indicarles cómo encontrar las credenciales de AWS.

Trabajando con las funciones de IAM para EKS Pod Identities

- Las identidades de los pods de EKS solo pueden asumir directamente una función de IAM que pertenezca a la misma cuenta de AWS que el clúster de EKS. Para acceder a una función de IAM en otra cuenta de AWS, debe asumir esa función configurando un perfil en la configuración del SDK o en el código de la aplicación.
- Cuando se configuran las identidades de Pod de EKS para las cuentas de servicio, la persona o el proceso que configura la asociación de identidades de pods debe tener los `iam:PassRole` derechos para desempeñar esa función.
- Cada cuenta de servicio solo puede tener un rol de IAM asociado a través de EKS Pod Identities; sin embargo, puedes asociar el mismo rol de IAM a varias cuentas de servicio.

- Las funciones de IAM utilizadas con las identidades de los pods de EKS deben permitir que el director del `pods.eks.amazonaws.com` servicio las asuma y establezca las etiquetas de sesión. A continuación se muestra un ejemplo de una política de confianza de roles que permite a EKS Pod Identities utilizar un rol de IAM:

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

AWS recomienda utilizar claves de condición, por ejemplo, `aws:SourceOrgId` para ayudar a protegerse contra el problema de los diputados confusos entre servicios. En el ejemplo anterior de política de confianza de roles, `ResourceOrgId` es una variable igual al ID de organización de AWS Organizations de la organización de AWS a la que pertenece la cuenta de AWS. EKS transferirá un valor `aws:SourceOrgId` igual a ese valor cuando asuma un rol con EKS Pod Identities.

Identidades de pod ABAC y EKS

Cuando EKS Pod Identities asume una función de IAM, establece las siguientes etiquetas de sesión:

Etiqueta de sesión de EKS Pod Identities	Valor
espacio de nombres de kubernetes	El espacio de nombres en el que se ejecuta el pod asociado a EKS Pod Identities.

Etiqueta de sesión de EKS Pod Identities	Valor
kubernetes-service-account	El nombre de la cuenta de servicio de Kubernetes asociada a EKS Pod Identities
eks-cluster-arn	El ARN del clúster EKS, p. ej. `arn:${Partition}:eks:${Region}:${Account}:cluster/${ClusterName}` El ARN del clúster es único, pero si se elimina un clúster y se vuelve a crear en la misma región con el mismo nombre y dentro de la misma cuenta de AWS, tendrá el mismo ARN.
eks-cluster-name	El nombre del clúster EKS. Tenga en cuenta que los nombres de los clústeres de EKS pueden ser los mismos en su cuenta de AWS y los de los clústeres de EKS en otras cuentas de AWS.
kubernetes-pod-name	El nombre del pod en EKS.
kubernetes-pod-uid	El UID del módulo en EKS.

Estas etiquetas de sesión le permiten usar el control de acceso basado en atributos (ABAC) para conceder acceso a sus recursos de AWS únicamente a cuentas de servicio de Kubernetes específicas. Al hacerlo, es muy importante entender que las cuentas de servicio de kubernetes solo son únicas dentro de un espacio de nombres, y los espacios de nombres de kubernetes solo son únicos dentro de un clúster de EKS. Se puede acceder a estas etiquetas de sesión en las políticas de AWS mediante la clave de condición `aws:PrincipalTag/<tag-key>` global, como `aws:PrincipalTag/eks-cluster-arn`

Por ejemplo, si quisiera conceder acceso solo a una cuenta de servicio específica para acceder a un recurso de AWS de su cuenta con una política de recursos o de IAM, tendría que comprobar `eks-cluster-arn` y `kubernetes-namespace` etiquetar, así como asegurarse de que solo las cuentas de servicio del clúster deseado tengan acceso `kubernetes-service-account` a ese recurso, ya que otros clústeres podrían tener acceso a ese recurso, ya que otros clústeres podrían tener el mismo `kubernetes-service-accounts` y`kubernetes-namespaces`.

Este ejemplo de política de bucket de S3 solo concede acceso a los objetos del bucket de S3 al que está conectado, solo si `kubernetes-service-account` `eks-cluster-arn` todos cumplen los valores esperados, donde el clúster de EKS está alojado en la cuenta de AWS`111122223333`. `kubernetes-namespace`

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

Comparación de las identidades de los pods de EKS con las de IRSA

Tanto las identidades de los pods de EKS como las de IRSA son las formas preferidas de entregar credenciales de AWS temporales a sus pods de EKS. A menos que tenga casos de uso específicos para IRSA, le recomendamos que utilice EKS Pod Identities cuando utilice EKS. Esta tabla ayuda a comparar las dos funciones.

#	Pod Identities de EKS	IRSA
¿Necesita permiso para crear un IDP de OIDC en sus cuentas de AWS?	No	Sí
Requiere una configuración de IDP única por clúster	No	Sí
Establece las etiquetas de sesión relevantes para usarlas con ABAC	Sí	No
Requiere un iam: ¿Comproba r? PassRole	Sí	No
¿Utiliza AWS STS Quota desde su cuenta de AWS?	No	Sí
Puede acceder a otras cuentas de AWS	Indirectamente con el encadenamiento de roles	Directamente con sts: AssumeRoleWithWebIdentity
Compatible con AWS SDKs	Sí	Sí
¿Requiere Pod Identity Agent Daemonset en los nodos?	Sí	No

Identidades y credenciales para los pods de EKS: recomendaciones

Actualice el daemonset de aws-node para usar IRSA

En la actualidad, el daemonset de aws-node está configurado para usar un rol asignado a las instancias y asignarlo a los pods. EC2 IPs Esta función incluye varias políticas gestionadas por AWS, por ejemplo, EC2 ContainerRegistryReadOnly Amazoneks_CNI_Policy, que permiten a todos los pods que se ejecutan en un nodo acceder a direcciones IP o extraer imágenes del ECR. attach/ detach ENIs assign/unassign Dado que esto supone un riesgo para su clúster, se recomienda actualizar el daemonset de aws-node para que utilice IRSA. Encontrará un script para hacerlo en el repositorio de esta guía.

El daemonset de aws-node es compatible con las identidades de EKS Pod en las versiones 1.15.5 y posteriores.

Restrinja el acceso al perfil de instancia asignado al nodo trabajador

Al utilizar las identidades de pod de IRSA o EKS, se actualiza la cadena de credenciales del pod para utilizar primero las identidades de pod de IRSA o EKS. Sin embargo, el pod aún puede heredar los derechos del perfil de instancia asignado al nodo de trabajo. En el caso de los pods que no necesiten estos permisos, puedes bloquear el acceso a los metadatos de la instancia para garantizar que tus aplicaciones solo tengan los permisos que necesitan y no sus nodos.

> **Warning**
>
> Bloquear el acceso a los metadatos de la instancia impedirá que los pods que no usen las identidades de los pods IRSA o EKS hereden la función asignada al nodo de trabajo.

Puedes bloquear el acceso a los metadatos de la instancia exigiendo que la instancia IMDSv2 solo los use y actualizando el recuento de saltos a 1, como se muestra en el siguiente ejemplo. También puedes incluir estos ajustes en la plantilla de lanzamiento del grupo de nodos. No inhabilites los metadatos de la instancia, ya que esto impedirá que componentes como el controlador de terminación de nodos y otros elementos que dependen de los metadatos de la instancia funcionen correctamente.

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

Si utilizas Terraform para crear plantillas de lanzamiento para usarlas con grupos de nodos gestionados, añade el bloque de metadatos para configurar el recuento de saltos, tal y como se muestra en este fragmento de código:

```
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" } …
```

También puedes bloquear el acceso de un pod a EC2 los metadatos manipulando las iptables del nodo. Para obtener más información sobre este método, consulta Limitar el acceso al servicio de metadatos de la instancia.

Si tiene una aplicación que utiliza una versión anterior del SDK de AWS que no es compatible con IRSA o EKS Pod Identities, debe actualizar la versión del SDK.

Limite la política de confianza de los roles de IAM para los roles de IRSA al nombre de la cuenta de servicio, al espacio de nombres y al clúster

La política de confianza se puede aplicar a un espacio de nombres o a una cuenta de servicio específica dentro de un espacio de nombres. Al usar IRSA, es mejor hacer que la política de confianza de roles sea lo más explícita posible e incluir el nombre de la cuenta de servicio. Esto evitará de forma efectiva que otros pods del mismo espacio de nombres asuman esa función. La CLI lo `eksctl` hará automáticamente cuando la utilice para crear accounts/IAM funciones de servicio. Consulte https://eksctl. io/usage/iamserviceaccounts/para obtener más información.

Cuando se trabaja directamente con IAM, se añade una condición a la política de confianza del rol que utiliza condiciones para garantizar que la `:sub` reclamación sea el espacio de nombres y la cuenta de servicio que esperabas. A modo de ejemplo, antes teníamos un token IRSA con la subafirmación «system:serviceaccount:default:s3-read-only». Este es el `s3-read-only` espacio de nombres y la cuenta de servicio. `default` Deberías usar una condición como la siguiente para asegurarte de que solo tu cuenta de servicio en un espacio de nombres determinado del clúster pueda asumir esa función:

```
 "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"
     }
 }
```

Utilice un rol de IAM por aplicación

Tanto con IRSA como con EKS Pod Identity, se recomienda asignar a cada aplicación su propia función de IAM. Esto mejora el aislamiento, ya que puede modificar una aplicación sin que ello afecte a la otra, y le permite aplicar el principio de menor privilegio al conceder únicamente a una aplicación los permisos que necesita.

Al utilizar ABAC con EKS Pod Identity, puede utilizar una función de IAM común en varias cuentas de servicio y confiar en sus atributos de sesión para controlar el acceso. Esto resulta especialmente útil cuando se trabaja a gran escala, ya que ABAC le permite operar con menos funciones de IAM.

Cuando su aplicación necesite acceder al IMDS, utilice IMDSv2 y aumente el límite de saltos en EC2 las instancias a 2

IMDSv2requiere que utilices una solicitud PUT para obtener un token de sesión. La solicitud PUT inicial debe incluir un TTL para el token de sesión. Las versiones más recientes de AWS SDKs se encargarán de esto y de la renovación de dicho token automáticamente. También es importante tener en cuenta que el límite de saltos predeterminado en las EC2 instancias se establece intencionadamente en 1 para evitar el reenvío de IP. Como consecuencia, los pods que soliciten un token de sesión que se ejecuten en EC2 instancias pueden agotar el tiempo de espera y volver a utilizar el flujo de IMDSv1 datos. EKS añade compatibilidad IMDSv2 al habilitar tanto la versión 1 como la versión 2 y cambiar el límite de saltos a 2 en los nodos aprovisionados por eksctl o con las plantillas oficiales. CloudFormation

Desactivar el montaje automático de los tokens de las cuentas de servicio

Si tu aplicación no necesita llamar a la API de Kubernetes, establece el `automountServiceAccountToken` atributo `false` en el de tu aplicación o parchea la cuenta de servicio predeterminada en cada espacio de nombres PodSpec para que deje de montarse automáticamente en los pods. Por ejemplo:

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

Usa cuentas de servicio dedicadas para cada aplicación

Cada aplicación debe tener su propia cuenta de servicio dedicada. Esto se aplica a las cuentas de servicio de la API de Kubernetes, así como a las de IRSA y EKS Pod Identity.

> **Important**
>
> Si utilizas IRSA para blue/green actualizar los clústeres en lugar de realizar una actualización local del clúster, tendrás que actualizar la política de confianza de cada una de las funciones de IAM de IRSA con el punto final OIDC del nuevo clúster. En una actualización de blue/green clúster, se crea un clúster que ejecuta una versión más reciente de Kubernetes junto con el clúster anterior y se utiliza un equilibrador de carga o una malla de servicios para transferir sin problemas el tráfico de los servicios que se ejecutan en el clúster antiguo al clúster nuevo. Al utilizar las actualizaciones de blue/green clústeres con EKS Pod Identity, se crearían asociaciones de identidad de pod entre las funciones de IAM y las cuentas de

servicio del nuevo clúster. Y actualice la política de confianza de los roles de IAM si tiene alguna condición. `sourceArn`

Ejecute la aplicación como usuario no root

Los contenedores se ejecutan como root de forma predeterminada. Si bien esto les permite leer el archivo del token de identidad web, ejecutar un contenedor como root no se considera una práctica recomendada. Como alternativa, considere la posibilidad de añadir el `spec.securityContext.runAsUser` atributo a PodSpec. El valor de `runAsUser` es un valor arbitrario.

En el siguiente ejemplo, todos los procesos del pod se ejecutarán con el ID de usuario especificado en el `runAsUser` campo.

```
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" ]
```

Cuando ejecutas un contenedor como usuario no root, se impide que el contenedor lea el token de la cuenta de servicio de IRSA, ya que al token se le asignan los permisos 0600 [root] de forma predeterminada. Si actualizas el SecurityContext de tu contenedor para que incluya fsgroup=65534 [Nobody], el contenedor podrá leer el token.

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

En Kubernetes 1.19 y versiones posteriores, este cambio ya no es necesario y las aplicaciones pueden leer el token de la cuenta de servicio de IRSA sin añadirlo al grupo Nobody.

Conceda el acceso menos privilegiado a las aplicaciones

Action Hero es una utilidad que puede ejecutar junto con su aplicación para identificar las llamadas a la API de AWS y los permisos de IAM correspondientes que su aplicación necesita para funcionar correctamente. Es similar a IAM Access Advisor en el sentido de que le ayuda a limitar gradualmente el alcance de las funciones de IAM asignadas a las aplicaciones. Consulte la documentación sobre la concesión del acceso menos privilegiado a los recursos de AWS para obtener más información.

Considere la posibilidad de establecer un límite de permisos para las funciones de IAM utilizadas con las identidades de IRSA y Pod. Puede usar el límite de permisos para garantizar que las funciones utilizadas por IRSA o Pod Identities no superen un nivel máximo de permisos. Para ver una guía de ejemplo sobre cómo empezar con los límites de los permisos con un ejemplo de política de límites de permisos, consulta este repositorio de GitHub.

Revisa y revoca el acceso anónimo innecesario a tu clúster de EKS

Lo ideal sería que el acceso anónimo estuviera deshabilitado para todas las acciones de la API. El acceso anónimo se concede mediante la creación de RoleBinding o ClusterRoleBinding para el sistema de usuario integrado de Kubernetes: anonymous. Puedes usar la herramienta rbac-lookup para identificar los permisos que el usuario system:anonymous tiene en tu clúster:

```
./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
```

Cualquier rol que no ClusterRole sea system: no public-info-viewer debe estar vinculado a system:anonymous user ni system:unauthenticated group.

Puede haber algunas razones legítimas para habilitar el acceso anónimo en un lugar específico. APIs Si este es el caso de su clúster, asegúrese de que solo un usuario anónimo pueda acceder a esos grupos específicos. Si APIs se exponen los datos APIs sin autenticación, el clúster no será vulnerable.

Antes de la Kubernetes/EKS versión 1.14, el grupo system:unauthenticated estaba asociado a system:discovery y system:basic-user de forma predeterminada. ClusterRoles Tenga en cuenta que, aunque haya actualizado el clúster a la versión 1.14 o superior, es posible que estos permisos sigan habilitados en el clúster, ya que las actualizaciones del clúster no revocan estos permisos. Para

comprobar cuáles ClusterRoles tienen «system:unauthenticated» excepto system: public-info-viewer puedes ejecutar el siguiente comando (requiere jq util):

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

Además, puedes eliminar «system:unauthenticated» de todas las funciones excepto de «system:» mediante: 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 -
```

También puedes verificarlo y eliminarlo manualmente mediante kubectl describe y kubectl edit. Para comprobar si system:unauthenticated group tiene permisos system:discovery en tu clúster, ejecuta el siguiente comando:

```
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
```

Para comprobar si system:unauthenticated group tiene el permiso system:basic-user en tu clúster, ejecuta el siguiente comando:

```
kubectl describe clusterrolebindings system:basic-user

Name:         system:basic-user
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
```

```
Role:
  Kind:  ClusterRole
  Name:  system:basic-user
Subjects:
  Kind   Name                    Namespace
  ----   ----                    ---------
  Group  system:authenticated
  Group  system:unauthenticated
```

Si system:unauthenticated group está enlazado a system:discovery y/o system:basic-user en tu clúster, debes desasociar estas funciones de system:unauthenticated group. ClusterRoles Edita ClusterRoleBinding system:discovery con el siguiente comando:

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

El comando anterior abrirá la definición actual de system:discovery ClusterRoleBinding en un editor como se muestra a continuación:

```
# 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
```

Elimine la entrada correspondiente al grupo system:unauthenticated de la sección «asuntos» de la pantalla del editor anterior.

Repita los mismos pasos para system:basic-user. ClusterRoleBinding

Reutilice las sesiones del SDK de AWS con IRSA

Cuando utilizas IRSA, las aplicaciones escritas con el SDK de AWS utilizan el token que se entrega en tus pods para llamar `sts:AssumeRoleWithWebIdentity` a fin de generar credenciales de AWS temporales. Esto es diferente de otros servicios de cómputo de AWS, donde el servicio de cómputo entrega credenciales temporales de AWS directamente al recurso de cómputo de AWS, como una función lambda. Esto significa que cada vez que se inicializa una sesión de AWS SDK, se realiza una llamada a AWS `AssumeRoleWithWebIdentity` STS for. Si su aplicación escala rápidamente e inicializa muchas sesiones del SDK de AWS, es posible que se produzca una ralentización por parte de AWS STS, ya que su código tendrá que hacer muchas llamadas. `AssumeRoleWithWebIdentity`

Para evitar esta situación, le recomendamos reutilizar las sesiones del SDK de AWS en su aplicación para evitar realizar `AssumeRoleWithWebIdentity` llamadas innecesarias.

En el siguiente código de ejemplo, se crea una sesión con el SDK de python boto3 y esa misma sesión se utiliza para crear clientes e interactuar con Amazon S3 y Amazon SQS. `AssumeRoleWithWebIdentity`solo se llama una vez y el SDK de AWS actualizará automáticamente las credenciales `my_session` cuando caduquen.

```
import boto3

= Create your own session

my_session = boto3.session.Session()

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

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

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

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

Si va a migrar una aplicación de otro servicio informático de AWS, por ejemplo EC2, a EKS con IRSA, este detalle es especialmente importante. En otros servicios informáticos, la inicialización de una sesión del SDK de AWS no llama a AWS STS a menos que usted le indique que lo haga.

Enfoques alternativos

Si bien las identidades de los pods IRSA y EKS son las formas preferidas de asignar una identidad de AWS a un pod, requieren que incluya una versión reciente de AWS SDKs en su aplicación. Para obtener una lista completa de los SDKs que actualmente son compatibles con IRSA, consulte https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts-minimum-sdk.html, for EKS Pod Identities, see https://docs.aws.amazon.com/eks/latest/userguide/pod: id-minimum-sdk .html. Si tienes una aplicación que no puedes actualizar inmediatamente con un SDK compatible, hay varias soluciones creadas por la comunidad para asignar funciones de IAM a los pods de Kubernetes, como kube2iam y kiam. Si bien AWS no avala, aprueba ni apoya el uso de estas soluciones, la comunidad en general las utiliza con frecuencia para lograr resultados similares a los de las identidades de pod de IRSA y EKS.

Si necesita utilizar una de estas soluciones que no se proporcionan por ley, actúe con la diligencia debida y asegúrese de entender las implicaciones de seguridad que ello implica.

Herramientas y recursos

- Taller de inmersión en seguridad de Amazon EKS: Identity and Access Management
- Patrón de planos Terraform EKS: clúster Amazon EKS completamente privado
- Patrón de planos Terraform EKS: inicio de sesión único de IAM Identity Center para Amazon EKS Cluster
- Patrón de planos de Terraform EKS: inicio de sesión único de Okta para Amazon EKS Cluster
- audit2rbac
- rbac.dev Una lista de recursos adicionales, incluidos blogs y herramientas, para el RBAC de Kubernetes
- Héroe de acción
- kube2iam

- kiam

Pod Security

La especificación del módulo incluye una variedad de atributos diferentes que pueden reforzar o debilitar tu postura general en materia de seguridad. Como profesional de Kubernetes, tu principal preocupación debería ser evitar que un proceso que se ejecuta en un contenedor escape a los límites de aislamiento del tiempo de ejecución del contenedor y acceda al host subyacente.

Capacidades de Linux

Los procesos que se ejecutan dentro de un contenedor se ejecutan en el contexto del usuario root [de Linux] de forma predeterminada. Si bien las acciones de root dentro de un contenedor están limitadas en parte por el conjunto de capacidades de Linux que el motor de ejecución del contenedor asigna a los contenedores, estos privilegios predeterminados podrían permitir a un atacante aumentar sus privilegios .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 y las cápsulas Fargate tienen asignadas las capacidades antes mencionadas de forma predeterminada. Además, las capacidades de Linux solo se pueden eliminar de los pods de Fargate.

Los pods que se ejecutan con privilegios heredan todas las capacidades de Linux asociadas a la raíz del host. Esto debe evitarse en la medida de lo posible.

Autorización de nodos

Todos los nodos de trabajo de Kubernetes utilizan un modo de autorización denominado Autorización de nodos. La autorización de nodos autoriza todas las solicitudes de API que se originan en el kubelet y permite a los nodos realizar las siguientes acciones:

Operaciones de lectura:

- servicios
- puntos de conexión
- nodos
- pods
- secretos, mapas de configuración, notificaciones de volumen persistentes y volúmenes persistentes relacionados con los pods enlazados al nodo del kubelet

Operaciones de escritura:

- nodos y estado de los nodos (habilite el complemento de `NodeRestriction` admisión para limitar que un kubelet modifique su propio nodo)
- pods y estado de los pods (habilite el complemento de `NodeRestriction` admisión para limitar un kubelet y modificar los pods enlazados a sí mismo)
- eventos

Operaciones relacionadas con la autenticación:

- Acceso de lectura y escritura a la API CertificateSigningRequest (CSR) para el arranque de TLS
- la capacidad de crear y realizar comprobaciones delegadas TokenReview SubjectAccessReview authentication/authorization

EKS utiliza el controlador de admisión por restricciones de nodos, que solo permite al nodo modificar un conjunto limitado de atributos del nodo y objetos del módulo que están enlazados al nodo. Sin embargo, un atacante que consiga acceder al host podrá seguir recopilando información confidencial sobre el entorno a través de la API de Kubernetes, lo que le permitirá moverse lateralmente dentro del clúster.

Soluciones de seguridad Pod

Política de seguridad de Pod (PSP)

En el pasado, los recursos de la Política de seguridad de los pods (PSP) se utilizaban para especificar un conjunto de requisitos que los pods tenían que cumplir antes de poder crearlos. A partir de la versión 1.21 de Kubernetes, la PSP quedó obsoleta. Está previsto que se eliminen en la versión 1.25 de Kubernetes.

> **Important**
>
> PSPs están en desuso en la versión 1.21 de Kubernetes. Tendrá hasta la versión 1.25 o aproximadamente 2 años para realizar la transición a una alternativa. En este documento se explica el motivo de esta obsolescencia.

Migración a una nueva solución de seguridad para módulos

Dado que PSPs se eliminaron a partir de la versión 1.25 de Kubernetes, los administradores y operadores de clústeres deben reemplazar esos controles de seguridad. Hay dos soluciones que pueden cubrir esta necesidad:

- Policy-as-code Soluciones (PAC) del ecosistema de Kubernetes
- Estándares de seguridad (PSS) de Kubernetes Pod

Tanto la solución PAC como la PSS pueden coexistir con la PSP; se pueden utilizar en clústeres antes de eliminar la PSP. Esto facilita la adopción al migrar desde PSP. Consulte este documento cuando esté pensando en migrar de PSP a PSS.

Kyverno, una de las soluciones de PAC que se describen a continuación, tiene una guía específica en una entrada de blog para migrar de una solución PSPs a otra, que incluye políticas análogas, comparaciones de características y un procedimiento de migración. Se ha publicado información y orientación adicionales sobre la migración a Kyverno con respecto a la admisión de Pod Security (PSA) en el blog de AWS aquí.

Policy-as-code (PAC)

Policy-as-code Las soluciones (PAC) proporcionan barandillas para guiar a los usuarios del clúster y evitar comportamientos no deseados mediante controles prescritos y automatizados. El PAC utiliza los controladores de admisión dinámica de Kubernetes para interceptar el flujo de solicitudes del servidor API de Kubernetes, mediante una llamada de webhook, y mutar y validar las cargas útiles de las solicitudes, en función de las políticas escritas y almacenadas como código. La mutación y la validación se producen antes de que la solicitud del servidor de API provoque un cambio en el clúster. Las soluciones de PAC utilizan políticas para hacer coincidir las cargas útiles de las solicitudes del servidor de API y actuar en función de ellas, en función de la taxonomía y los valores.

Hay varias soluciones PAC de código abierto disponibles para Kubernetes. Estas soluciones no forman parte del proyecto Kubernetes; provienen del ecosistema de Kubernetes. Algunas soluciones de PAC se enumeran a continuación.

- OPA/Gatekeeper
- Open Policy Agent (OPA)
- Kyverno
- Kubewarden
- Política JS

Para obtener más información sobre las soluciones de PAC y cómo ayudarlo a seleccionar la solución adecuada para sus necesidades, consulte los enlaces que aparecen a continuación.

- Contramedidas basadas en políticas para Kubernetes: primera parte
- Contramedidas basadas en políticas para Kubernetes: parte 2

Estándares de seguridad para Pod (PSS) y Pod Security Admission (PSA)

En respuesta a la obsolescencia de la PSP y a la continua necesidad de controlar la seguridad de los módulos out-of-the-box, el Grupo de Interés Especial sobre Autenticación de Kubernetes creó los Estándares de Seguridad (PSS) y los Sistemas de Admisión de Seguridad para los Pod (PSA). La iniciativa de la PSA incluye un proyecto de webhook para el controlador de admisión que implementa los controles definidos en el PSS. Este enfoque de controlador de admisión se parece al utilizado en las soluciones PAC.

Según la documentación de Kubernetes, el PSS «define tres políticas diferentes para cubrir ampliamente el espectro de la seguridad. Estas políticas son acumulativas y van desde las muy permisivas hasta las muy restrictivas. `»

Estas políticas se definen como:

- Privilegiado: política sin restricciones (insegura) que proporciona el nivel de permisos más amplio posible. Esta política permite escalamientos de privilegios conocidos. Es la ausencia de una política. Esto es bueno para aplicaciones como agentes de registro CNIs, controladores de almacenamiento y otras aplicaciones de todo el sistema que necesitan un acceso privilegiado.
- Base de referencia: política mínimamente restrictiva que impide las escaladas de privilegios conocidas. Permite la configuración de pod predeterminada (especificada mínimamente). La

política básica prohíbe el uso de HostNetwork, HostPID, HostPC, HostPath, HostPort y la imposibilidad de añadir capacidades de Linux, entre otras restricciones.

- Restringido: política muy restringida, que sigue las mejores prácticas actuales de Pod Harding. Esta política se hereda de la base y añade restricciones adicionales, como la imposibilidad de ejecutarse como root o como grupo root. Las políticas restringidas pueden afectar a la capacidad de funcionamiento de una aplicación. Están destinadas principalmente a ejecutar aplicaciones críticas para la seguridad.

Estas políticas definen los perfiles para la ejecución de los módulos, organizados en tres niveles: acceso privilegiado y acceso restringido.

Para implementar los controles definidos por el PSS, el PSA funciona de tres modos:

- hacer cumplir: las infracciones de la política provocarán el rechazo del módulo.
- auditoría: las infracciones de las políticas provocarán la adición de una anotación de auditoría al evento registrado en el registro de auditoría, pero de lo contrario están permitidas.
- advertencia: las infracciones de las políticas generarán una advertencia dirigida al usuario, pero de lo contrario están permitidas.

Estos modos y los niveles de perfil (restricción) se configuran en el nivel del espacio de nombres de Kubernetes, mediante etiquetas, como se muestra en el siguiente ejemplo.

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

Cuando se utilizan de forma independiente, estos modos operativos tienen respuestas diferentes que se traducen en experiencias de usuario diferentes. El modo de aplicación impedirá que se creen pods si los PodSpecs respectivos infringen el nivel de restricción configurado. Sin embargo, en este modo, no se impedirá que los objetos de Kubernetes que no sean pods y que creen pods, como los despliegues, se apliquen al clúster, incluso si el PodSpec incluido en ellos infringe el PSS aplicado. En este caso, se aplicará el despliegue, mientras que no se podrán aplicar los pods.

Se trata de una experiencia de usuario difícil, ya que no hay ningún indicio inmediato de que el objeto de despliegue aplicado correctamente oculte la creación fallida del pod. El PodSpecs

infractor no creará pods. Al inspeccionar el recurso de despliegue con él, `kubectl get deploy <DEPLOYMENT_NAME> -oyaml` se mostrará el mensaje del `.status.conditions` elemento de los pods fallidos, tal y como se muestra a continuación.

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

Tanto en el modo de auditoría como en el de advertencia, las restricciones de los pods no impiden que se creen e inicien los pods infractores. Sin embargo, en estos modos, las anotaciones de auditoría en los eventos del registro de auditoría del servidor API y las advertencias a los clientes del servidor API, como kubectl, se activan, respectivamente, cuando los pods, así como los objetos que los crean, contienen PodSpecs con infracciones. A continuación se muestra un `kubectl` mensaje de advertencia.

```
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
```

Los modos de auditoría y advertencia del PSA son útiles a la hora de introducir el PSS sin afectar negativamente a las operaciones del clúster.

Los modos operativos del PSA no se excluyen mutuamente y se pueden utilizar de forma acumulativa. Como se muestra a continuación, los múltiples modos se pueden configurar en un único espacio de nombres.

```
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
```

En el ejemplo anterior, las advertencias y anotaciones de auditoría fáciles de usar se proporcionan al aplicar las implementaciones, mientras que la aplicación de las infracciones también se proporciona a nivel de grupo. De hecho, varias etiquetas de PSA pueden utilizar diferentes niveles de perfil, como se muestra a continuación.

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

En el ejemplo anterior, el PSA está configurado para permitir la creación de todos los grupos que cumplan el nivel de perfil básico y, a continuación, avisar sobre los grupos (y los objetos que crean grupos) que infrinjan el nivel de perfil restringido. Este es un enfoque útil para determinar los posibles impactos al cambiar del perfil de referencia a uno restringido.

Cápsulas existentes

Si se modifica un espacio de nombres con pods existentes para usar un perfil PSS más restrictivo, los modos de auditoría y advertencia generarán los mensajes adecuados; sin embargo, el modo de aplicación no eliminará los pods. Los mensajes de advertencia se muestran a continuación.

```
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
```

Exenciones

La PSA utiliza las exenciones para excluir la aplicación de infracciones contra grupos que, de otro modo, se habrían aplicado. Estas exenciones se enumeran a continuación.

- Nombres de usuario: se ignoran las solicitudes de los usuarios con un nombre de usuario autenticado (o suplantado) exento.
- RuntimeClassNames: se ignoran los pods y los recursos de carga de trabajo que especifican un nombre de clase de ejecución exento.
- Espacios de nombres: se ignoran los pods y los recursos de carga de trabajo de un espacio de nombres exento.

Estas exenciones se aplican de forma estática en la configuración del controlador de admisión de la PSA como parte de la configuración del servidor API.

En la implementación de Validating Webhook, las exenciones se pueden configurar dentro de un ConfigMaprecurso de Kubernetes que se monta como un volumen en el contenedor. 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"]
```

Como se ve en el ConfigMap YAML anterior, el nivel de PSS predeterminado para todo el clúster se ha establecido como restringido para todos los modos de PSA, auditar, aplicar y advertir. Esto afecta a todos los espacios de nombres, excepto a los exentos:. `namespaces: ["kube-system","policy-test1"]` Además, en el ValidatingWebhookConfigurationrecurso, que se ve a continuación, el espacio de pod-security-webhooknombres también está exento del PSS configurado.

```
...
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**
>
> Las admisiones de Pod Security pasaron a ser estables en Kubernetes v1.25. Si querías usar la función Pod Security Admission antes de activarla de forma predeterminada, tenías que instalar el controlador de admisión dinámico (webhook mutante). Puedes encontrar las instrucciones para instalar y configurar el webhook aquí.

Elegir entre los estándares de seguridad policy-as-code y Pod

Los estándares de seguridad de Pod (PSS) se desarrollaron para sustituir a la política de seguridad de Pod (PSP), proporcionando una solución integrada en Kubernetes y que no requería soluciones del ecosistema de Kubernetes. Dicho esto, las soluciones policy-as-code (PAC) son considerablemente más flexibles.

La siguiente lista de ventajas y desventajas está diseñada para ayudarlo a tomar una decisión más informada sobre la solución de seguridad de su módulo.

Policy-as-code (en comparación con los estándares de seguridad de Pod)

Ventajas:

- Más flexible y detallado (se reduce a los atributos de los recursos si es necesario)
- No solo se centra en los módulos, sino que se puede utilizar con diferentes recursos y acciones
- No solo se aplica a nivel de espacio de nombres
- Más maduros que los estándares de seguridad de Pod
- Las decisiones se pueden basar en cualquier elemento de la carga útil de solicitud del servidor API, así como en los recursos del clúster existentes y en los datos externos (según la solución)
- Admite la mutación de las solicitudes del servidor de API antes de la validación (depende de la solución)
- Puede generar políticas y recursos de Kubernetes complementarios (depende de la solución): a partir de las políticas de pod, Kyverno puede generar automáticamente políticas para controladores de nivel superior, como las implementaciones. Kyverno también puede generar recursos adicionales de Kubernetes «`cuando se crea un nuevo recurso o cuando se actualiza la fuente`» mediante Generate Rules.)
- Se puede usar para desplazarse a la izquierda, hacia las canalizaciones del CICD, antes de realizar llamadas al servidor API de Kubernetes (depende de la solución)
- Se puede usar para implementar comportamientos que no están necesariamente relacionados con la seguridad, como las mejores prácticas, los estándares organizacionales, etc.
- Se puede usar en casos de uso ajenos a Kubernetes (depende de la solución)
- Gracias a la flexibilidad, la experiencia de usuario se puede ajustar a las necesidades de los usuarios

Desventajas:

- No está integrado en Kubernetes
- Más complejo de aprender, configurar y soportar
- La creación de políticas puede requerir nuevas skills/languages/capabilities

Entrada a Pod Security (en comparación policy-as-code con)

Ventajas:

- Integrado en Kubernetes
- Más fácil de configurar
- No hay nuevos idiomas que usar ni políticas que crear
- Si el nivel de admisión predeterminado del clúster está configurado como privilegiado, las etiquetas de los espacios de nombres se pueden utilizar para incluir los espacios de nombres en los perfiles de seguridad del módulo.

Desventajas:

- No es tan flexible o granular como policy-as-code
- Solo 3 niveles de restricciones
- Se centra principalmente en las cápsulas

Resumen

Si actualmente no dispones de una solución de seguridad para cápsulas, aparte de la PSP, y la postura de seguridad que necesitas para tu cápsula se ajusta al modelo definido en las normas de seguridad para cápsulas (PSS), lo más fácil sería adoptar la PSS en lugar de una solución. policy-as-code Sin embargo, si la postura de seguridad de su módulo no se ajusta al modelo de PSS, o si tiene pensado añadir controles adicionales, además de los definidos por el PSS, entonces una policy-as-code solución le parecería más adecuada.

Recomendaciones

Utilice varios modos de admisión segura (PSA) del pod para una mejor experiencia de usuario

Como se mencionó anteriormente, el modo de aplicación del PSA impide que se apliquen cápsulas con infracciones de PSS, pero no detiene a los controladores de nivel superior, como los despliegues. De hecho, la implementación se aplicará correctamente sin ningún indicio de que los pods no se hayan aplicado. Si bien puedes usar kubectl para inspeccionar el objeto de despliegue y descubrir el mensaje de error del pod en la PSA, la experiencia de usuario podría ser mejor. Para mejorar la experiencia del usuario, se deben utilizar varios modos de PSA (auditar, aplicar, advertir).

```
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
```

En el ejemplo anterior, con el modo de aplicación definido, cuando se intenta aplicar al servidor API de Kubernetes un manifiesto de despliegue con infracciones de PSS en el PodSpec correspondiente, el despliegue se aplicará correctamente, pero los pods no. Además, dado que los modos de auditoría y advertencia también están habilitados, el cliente del servidor de API recibirá un mensaje de advertencia y el evento del registro de auditoría del servidor de API también se anotará con un mensaje.

Restrinja los contenedores que pueden ejecutarse como privilegiados

Como se ha mencionado, los contenedores que se ejecutan con privilegios heredan todas las capacidades de Linux asignadas a la raíz del host. Rara vez los contenedores necesitan este tipo de privilegios para funcionar correctamente. Existen varios métodos que se pueden utilizar para restringir los permisos y las capacidades de los contenedores.

> **Important**
>
> Fargate es un tipo de lanzamiento que le permite ejecutar contenedores «sin servidor» en los que los contenedores de un pod se ejecutan en una infraestructura que administra AWS.

Con Fargate, no puedes ejecutar un contenedor privilegiado ni configurar tu pod para que use HostNetwork o HostPort.

No ejecute procesos en contenedores como root

Todos los contenedores se ejecutan como root de forma predeterminada. Esto podría resultar problemático si un atacante es capaz de aprovechar una vulnerabilidad de la aplicación y obtener acceso desde el shell al contenedor en ejecución. Puede mitigar este riesgo de varias maneras. En primer lugar, quitando la carcasa de la imagen del contenedor. En segundo lugar, añade la directiva USER a tu Dockerfile o ejecuta los contenedores del pod como usuario no root. El PodSpec de Kubernetes incluye un conjunto de campos, en, que te permiten especificar el grupo de usuarios and/or en `spec.securityContext` el que se ejecutará la aplicación. Estos campos son y respectivamente. `runAsUser` `runAsGroup`

Para hacer cumplir el uso del `spec.securityContext` PodSpec de Kubernetes y sus elementos asociados, se pueden añadir a los clústeres los estándares de seguridad de Pod policy-as-code o Pod. Estas soluciones le permiten escribir o usar políticas o perfiles que puedan validar las cargas útiles entrantes de solicitudes del servidor API de Kubernetes antes de que se almacenen en etcd. Además, policy-as-code las soluciones pueden modificar las solicitudes entrantes y, en algunos casos, generar nuevas solicitudes.

Nunca ejecute Docker en Docker ni monte el socket en el contenedor

Si bien esto te permite grabar build/run imágenes en contenedores Docker, básicamente estás cediendo el control total del nodo al proceso que se ejecuta en el contenedor. Si necesitas crear imágenes de contenedores en Kubernetes, usa Kaniko, buildah o un servicio de compilación similar. CodeBuild

Note

Los clústeres de Kubernetes que se utilizan para el procesamiento de CICD, como la creación de imágenes de contenedores, deben aislarse de los clústeres que ejecutan cargas de trabajo más generalizadas.

Restrinja el uso de HostPath o, si es necesario, restrinja los prefijos que se pueden usar y configure el volumen como de solo lectura

`hostPath`es un volumen que monta un directorio desde el host directamente al contenedor. Los pods rara vez necesitarán este tipo de acceso, pero si lo necesitan, debes ser consciente de los riesgos. De forma predeterminada, los pods que se ejecutan como root tendrán acceso de escritura al sistema de archivos expuesto por HostPath. Esto podría permitir a un atacante modificar la configuración del kubelet, crear enlaces simbólicos a directorios o archivos no expuestos directamente por HostPath, por ejemplo, /etc/shadow, instalar claves ssh, leer los secretos guardados en el servidor y otras acciones maliciosas. Para mitigar los riesgos de HostPath, configúrelo como, por ejemplo: `spec.containers.volumeMounts` `readOnly`

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

También debe usar policy-as-code soluciones para restringir los directorios que pueden usar los `hostPath` volúmenes o evitar su `hostPath` uso por completo. Puede utilizar las políticas básicas o restringidas de los estándares de seguridad de Pod para evitar el uso de`hostPath`.

Para obtener más información sobre los peligros de la escalada de privilegios, lee el blog de Seth Art Bad Pods: Kubernetes Pod Privilege Escalation.

Establezca solicitudes y límites para cada contenedor para evitar la contención de recursos y los ataques de DoS

En teoría, un pod sin solicitudes ni límites puede consumir todos los recursos disponibles en un host. A medida que se programan más pods en un nodo, es posible que el nodo sufra una presión en la CPU o la memoria, lo que puede provocar que el Kubelet cierre o expulse los pods del nodo. Si bien no se puede evitar que esto suceda por completo, establecer solicitudes y límites ayudará a minimizar la contención de recursos y a mitigar el riesgo que supone el uso de aplicaciones mal escritas que consumen una cantidad excesiva de recursos.

`podSpec`Esto te permite especificar las solicitudes y los límites de la CPU y la memoria. La CPU se considera un recurso comprimible porque puede estar sobresuscrita. La memoria es incompresible, es decir, no se puede compartir entre varios contenedores.

Cuando especificas las solicitudes de CPU o memoria, básicamente estás designando la cantidad de memoria que se garantiza que obtendrán los contenedores. Kubernetes agrega las solicitudes de todos los contenedores de un pod para determinar en qué nodo programar el pod. Si un contenedor supera la cantidad de memoria solicitada, puede ser cancelado si hay una presión de memoria en el nodo.

Los límites son la cantidad máxima de recursos de CPU y memoria que puede consumir un contenedor y corresponden directamente al `memory.limit_in_bytes` valor del cgroup creado para el contenedor. Un contenedor que supere el límite de memoria será eliminado mediante OOM. Si un contenedor supera su límite de CPU, se estrangulará.

> **Note**
>
> Al usar un contenedor`resources.limits`, se recomienda encarecidamente que el uso de los recursos del contenedor (también conocido como huella de recursos) se base en datos y sea preciso, en función de las pruebas de carga. A falta de una huella de recursos precisa y fiable, el contenedor `resources.limits` puede ser acolchado. Por ejemplo, `resources.limits.memory` podría rellenarse entre un 20 y un 30% más que los máximos observables, para tener en cuenta las posibles imprecisiones en los límites de los recursos de memoria.

Kubernetes usa tres clases de calidad de servicio (QoS) para priorizar las cargas de trabajo que se ejecutan en un nodo. Entre ellos se incluyen:

- garantizado
- reventable
- mejor esfuerzo

Si los límites y las solicitudes no están establecidos, el pod se configura como de máximo esfuerzo (prioridad más baja). Los módulos que hacen el mejor esfuerzo son los primeros en morir cuando no hay suficiente memoria. Si los límites están establecidos en todos los contenedores del módulo, o si las solicitudes y los límites se establecen con los mismos valores y no iguales a 0, el módulo se configura como garantizado (máxima prioridad). Los módulos garantizados no se eliminarán a menos que superen los límites de memoria configurados. Si los límites y las solicitudes están configurados con valores diferentes y no iguales a 0, o si un contenedor del pod establece límites y los demás no, o tienen límites establecidos para diferentes recursos, los pods se configuran como burstable

(prioridad media). Estos módulos tienen algunas garantías de recursos, pero se pueden eliminar una vez que superen la memoria solicitada.

> **Important**
>
> Las solicitudes no afectan al `memory_limit_in_bytes` valor del cgroup del contenedor; el límite del cgroup se establece en la cantidad de memoria disponible en el host. Sin embargo, si el valor de las solicitudes es demasiado bajo, el kubelet podría poner fin al pod si la memoria del nodo se ve presionada.

Clase	Priority (Prioridad)	Condición	Condición de muerte
Garantizada	highest	límite = solicitud! = 0	Supere únicament e los límites de memoria
ampliable	medium	límite! = solicitud! = 0	Se puede matar si se supera la memoria solicitada
El mejor esfuerzo	lowest	límite y solicitud no establecidos	El primero en morir cuando no hay suficiente memoria

Para obtener información adicional sobre la QoS de los recursos, consulte la documentación de Kubernetes.

Puedes forzar el uso de las solicitudes y los límites estableciendo una cuota de recursos en un espacio de nombres o creando un rango de límites. Una cuota de recursos te permite especificar la cantidad total de recursos, por ejemplo, la CPU y la RAM, asignados a un espacio de nombres. Cuando se aplica a un espacio de nombres, te obliga a especificar las solicitudes y los límites para todos los contenedores desplegados en ese espacio de nombres. Por el contrario, los rangos de límites ofrecen un control más detallado de la asignación de los recursos. Con los rangos de límites, puede min/max utilizar los recursos de CPU y memoria por pod o por contenedor dentro de un espacio de nombres. También puedes usarlos para establecer los valores de solicitud/límite predeterminados si no se proporciona ninguno.

Policy-as-code Las soluciones se pueden usar para hacer cumplir las solicitudes y los límites, o incluso para crear las cuotas de recursos y los rangos de límites cuando se crean los espacios de nombres.

No permita la escalación privilegiada

La escalada privilegiada permite que un proceso cambie el contexto de seguridad en el que se ejecuta. Sudo es un buen ejemplo de ello, al igual que los binarios con el bit SUID o SGID. La escalada de privilegios es básicamente una forma de que los usuarios ejecuten un archivo con los permisos de otro usuario o grupo. Para evitar que un contenedor utilice la escalación con privilegios, puede implementar una política de policy-as-code mutación que se establezca `allowPrivilegeEscalation` en `false` o establezca `securityContext.allowPrivilegeEscalation` en. `podSpec` Policy-as-code Las políticas también se pueden usar para evitar que las solicitudes del servidor de API se realicen correctamente si se detecta una configuración incorrecta. Los estándares de seguridad de los pods también se pueden usar para evitar que los pods utilicen la escalada de privilegios.

Deshabilita los montajes ServiceAccount de los tokens

En el caso de los pods que no necesiten acceder a la API de Kubernetes, puedes deshabilitar el montaje automático de un ServiceAccount token según las especificaciones de un pod o para todos los pods que usen una determinada especificación. 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
```

Inhabilita la detección de servicios

En el caso de los pods que no necesitan buscar ni llamar a los servicios del clúster, puedes reducir la cantidad de información que se proporciona a un pod. Puede configurar la política de DNS del pod para que no utilice CoreDNS y no exponga los servicios del espacio de nombres del pod como variables de entorno. Consulta los documentos de Kubernetes sobre variables de entorno para obtener más información sobre los enlaces de los servicios. El valor predeterminado de la política de DNS de un pod es «ClusterFirst», que usa el DNS del clúster, mientras que el valor no predeterminado, «Predeterminado», usa la resolución de DNS del nodo subyacente. Consulta los documentos de Kubernetes sobre la política de DNS del pod para obtener más información.

Example

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

Configura tus imágenes con un sistema de archivos raíz de solo lectura

La configuración de las imágenes con un sistema de archivos raíz de solo lectura evita que un atacante sobrescriba un binario del sistema de archivos que utiliza la aplicación. Si la aplicación tiene que escribir en el sistema de archivos, considere la posibilidad de escribir en un directorio temporal o adjuntar y montar un volumen. Para hacer cumplir esto, configure los pods de la SecurityContext siguiente manera:

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

Policy-as-code y los estándares de seguridad del pod se pueden usar para hacer cumplir este comportamiento.

Example

Según Windows, los contenedores de Kubernetes `securityContext.readOnlyRootFilesystem` no se pueden configurar `true` para un

contenedor que se ejecute en Windows, ya que se requiere acceso de escritura para que los procesos del registro y del sistema se ejecuten dentro del contenedor.

Herramientas y recursos

- Taller de inmersión en seguridad de Amazon EKS - Pod Security
- open-policy-agent/gatekeeper-library: La biblioteca de políticas de OPA Gatekeeper es una biblioteca de OPA/Gatekeeper políticas que puede utilizar como sustituta. PSPs
- Biblioteca de políticas de Kyverno
- Una colección de políticas comunes de OPA y Kyverno para EKS.
- Contramedidas basadas en políticas: parte 1
- Contramedidas basadas en políticas: parte 2
- Pod Security Policy Migrator, una herramienta que se convierte PSPs en políticas de OPA/Gatekeeper o Kyverno KubeWarden
- NeuVector de SUSE, una plataforma de seguridad de contenedores de código abierto y de confianza cero, proporciona políticas de procesos y sistemas de archivos, así como reglas de control de admisión.

Aislamiento de inquilinos

Cuando pensamos en la multitenencia, a menudo queremos aislar un usuario o una aplicación de otros usuarios o aplicaciones que se ejecutan en una infraestructura compartida.

Kubernetes es un orquestador de un solo inquilino, es decir, todos los inquilinos de un clúster comparten una única instancia del plano de control. Sin embargo, hay varios objetos de Kubernetes que puedes usar para crear una apariencia de tenencia múltiple. Por ejemplo, los espacios de nombres y los controles de acceso basados en roles (RBAC) se pueden implementar para aislar de forma lógica a los inquilinos unos de otros. Del mismo modo, las cuotas y los rangos de límites se pueden usar para controlar la cantidad de recursos del clúster que puede consumir cada inquilino. Sin embargo, el clúster es la única construcción que proporciona un límite de seguridad sólido. Esto se debe a que un atacante que consigue acceder a un host del clúster puede recuperar todos los secretos y volúmenes que estén instalados en ese host. ConfigMaps También podrían hacerse pasar por el Kubelet, lo que les permitiría manipular los atributos del nodo y and/or moverlos lateralmente dentro del clúster.

En las siguientes secciones, se explica cómo implementar el aislamiento de inquilinos y, al mismo tiempo, mitigar los riesgos de usar un único orquestador de inquilinos, como Kubernetes.

Multitenencia suave

Con la multitenencia flexible, se utilizan estructuras nativas de Kubernetes, por ejemplo, espacios de nombres, funciones y enlaces de funciones y políticas de red, para crear una separación lógica entre los inquilinos. El RBAC, por ejemplo, puede impedir que los inquilinos accedan o manipulen los recursos de los demás. Las cuotas y los rangos límite controlan la cantidad de recursos del clúster que puede consumir cada inquilino, mientras que las políticas de red pueden ayudar a evitar que las aplicaciones implementadas en diferentes espacios de nombres se comuniquen entre sí.

Sin embargo, ninguno de estos controles impide que los pods de distintos inquilinos compartan un nodo. Si se requiere un aislamiento más estricto, puedes usar un selector de nodos, reglas de antiafinidad, restricciones y tolerancias para obligar a and/or los pods de distintos arrendatarios a programarse en nodos separados, lo que suele denominarse nodos de arrendatario único. Esto podría resultar bastante complicado y tener un coste prohibitivo en un entorno con muchos inquilinos.

> **Important**
>
> La multitenencia flexible implementada con los espacios de nombres no permite proporcionar a los inquilinos una lista filtrada de espacios de nombres porque los espacios de nombres son un tipo de ámbito global. Si un inquilino tiene la capacidad de ver un espacio de nombres concreto, puede ver todos los espacios de nombres del clúster.

> **Warning**
>
> Con soft-multi-tenancy, los inquilinos conservan la capacidad de consultar CoredNS para todos los servicios que se ejecutan en el clúster de forma predeterminada. Un atacante podría aprovecharlo ejecutando dig SRV `..svc.cluster.local` desde cualquier pod del clúster. Si necesita restringir el acceso a los registros DNS de los servicios que se ejecutan en sus clústeres, considere la posibilidad de utilizar los complementos Firewall o Policy para CoreDNS. Para obtener información adicional, consulte https://github.com/coredns/kubernetes-metadata-multi-tenancy policy# -policy.

Kiosk es un proyecto de código abierto que puede ayudar a implementar la multitenencia flexible. Se implementa como una serie de controladores CRDs y controladores que proporcionan las siguientes capacidades:

- Cuentas y usuarios de cuentas para separar a los inquilinos en un clúster de Kubernetes compartido
- Aprovisionamiento de espacios de nombres de autoservicio para usuarios de cuentas
- Límites de cuentas para garantizar la calidad del servicio y la imparcialidad al compartir un clúster
- Plantillas de espacios de nombres para el aislamiento seguro de los inquilinos y la inicialización de los espacios de nombres con autoservicio

Loft es una oferta comercial de los mantenedores de Kiosk y que añade las siguientes capacidades: DevSpace

- Acceso a varios clústeres para permitir el acceso a los espacios de diferentes clústeres
- El modo de suspensión reduce las implementaciones en un espacio durante los períodos de inactividad
- Inicio de sesión único con proveedores de autenticación OIDC como GitHub

Hay tres casos de uso principales que se pueden abordar mediante la multitenencia flexible.

Entorno empresarial

La primera es en un entorno empresarial, en el que los «inquilinos» son casi confiables, ya que son empleados, contratistas o están autorizados por la organización por algún otro motivo. Por lo general, cada inquilino se alinea con una división administrativa, como un departamento o un equipo.

En este tipo de entorno, un administrador de clústeres suele ser responsable de crear los espacios de nombres y administrar las políticas. También pueden implementar un modelo de administración delegada en el que determinadas personas supervisen un espacio de nombres, lo que les permite realizar operaciones CRUD para objetos no relacionados con las políticas, como despliegues, servicios, módulos, trabajos, etc.

El aislamiento que proporciona el tiempo de ejecución de un contenedor puede ser aceptable dentro de esta configuración o puede que sea necesario aumentarlo con controles adicionales para garantizar la seguridad de los módulos. También puede ser necesario restringir la comunicación entre los servicios en diferentes espacios de nombres si se requiere un aislamiento más estricto.

Kubernetes como servicio

Por el contrario, la multitenencia flexible se puede utilizar en entornos en los que se quiera ofrecer Kubernetes como servicio (KaaS). Con KaaS, la aplicación se aloja en un clúster compartido junto con un conjunto de controladores CRDs que proporcionan un conjunto de servicios de PaaS. Los inquilinos interactúan directamente con el servidor API de Kubernetes y se les permite realizar operaciones CRUD en objetos ajenos a las políticas. También hay un elemento de autoservicio que permite a los inquilinos crear y administrar sus propios espacios de nombres. En este tipo de entorno, se supone que los inquilinos ejecutan código que no es de confianza.

Para aislar a los inquilinos en este tipo de entorno, es probable que tengas que implementar políticas de red estrictas, así como un entorno aislado. El sandboxing consiste en ejecutar los contenedores de un pod dentro de una micromáquina virtual, como Firecracker, o en un núcleo de espacio de usuario. Hoy en día, puede crear cápsulas con caja de arena con EKS Fargate.

Software como servicio (SaaS)

El caso de uso final de la multitenencia flexible es en un entorno (Software-as-a-ServiceSaaS). En este entorno, cada inquilino está asociado a una instancia concreta de una aplicación que se ejecuta dentro del clúster. Cada instancia suele tener sus propios datos y utiliza controles de acceso independientes que, por lo general, son independientes del RBAC de Kubernetes.

A diferencia de otros casos de uso, el inquilino de un entorno de SaaS no interactúa directamente con la API de Kubernetes. En cambio, la aplicación SaaS es responsable de interactuar con la API de Kubernetes para crear los objetos necesarios para dar soporte a cada inquilino.

Construcciones de Kubernetes

En cada uno de estos casos, se utilizan las siguientes estructuras para aislar a los inquilinos unos de otros:

Espacios de nombres

Los espacios de nombres son fundamentales para implementar la multitenencia flexible. Permiten dividir el clúster en particiones lógicas. Las cuotas, las políticas de red, las cuentas de servicio y otros objetos necesarios para implementar la multitenencia dependen de un espacio de nombres.

Políticas de red

De forma predeterminada, todos los pods de un clúster de Kubernetes pueden comunicarse entre sí. Este comportamiento se puede modificar mediante políticas de red.

Las políticas de red restringen la comunicación entre los pods mediante etiquetas o rangos de direcciones IP. En un entorno multiusuario en el que se requiera un aislamiento estricto de la red entre los usuarios, recomendamos empezar con una regla predeterminada que deniegue la comunicación entre los pods y otra que permita a todos los pods consultar el servidor DNS para obtener la resolución de nombres. Una vez establecido esto, puedes empezar a añadir reglas más permisivas que permitan la comunicación dentro de un espacio de nombres. Esto se puede refinar aún más según sea necesario.

> **Note**
>
> Amazon VPC CNI ahora es compatible con las políticas de red de Kubernetes para crear políticas que puedan aislar las cargas de trabajo confidenciales y protegerlas del acceso no autorizado al ejecutar Kubernetes en AWS. Esto significa que puede utilizar todas las funciones de la API de políticas de red de su clúster de Amazon EKS. Este nivel de control detallado le permite implementar el principio de privilegios mínimos, que garantiza que solo los pods autorizados puedan comunicarse entre sí.

> **Important**
>
> Las políticas de red son necesarias pero no suficientes. La aplicación de las políticas de red requiere un motor de políticas como Calico o Cilium.

Control de acceso basado en roles (RBAC)

Los roles y los enlaces de roles son los objetos de Kubernetes que se utilizan para aplicar el control de acceso basado en roles (RBAC) en Kubernetes. Los roles contienen listas de acciones que se pueden realizar contra los objetos del clúster. Los enlaces de roles especifican las personas o grupos a los que se aplican los roles. En los entornos empresarial y de KaaS, el RBAC se puede utilizar para permitir la administración de objetos por parte de grupos o individuos seleccionados.

Cuotas

Las cuotas se utilizan para definir los límites de las cargas de trabajo alojadas en el clúster. Con las cuotas, puedes especificar la cantidad máxima de CPU y memoria que puede consumir un pod, o puedes limitar la cantidad de recursos que se pueden asignar en un clúster o espacio de nombres.

Los rangos de límites te permiten declarar los valores mínimos, máximos y predeterminados para cada límite.

La sobreasignación de recursos en un clúster compartido suele ser beneficiosa porque permite maximizar los recursos. Sin embargo, el acceso ilimitado a un clúster puede provocar una escasez de recursos, lo que puede provocar una degradación del rendimiento y una pérdida de disponibilidad de las aplicaciones. Si las solicitudes de un pod son demasiado bajas y la utilización real de los recursos supera la capacidad del nodo, el nodo empezará a sufrir una presión de CPU o memoria. Cuando esto sucede, los pods pueden reiniciarse y and/or expulsarse del nodo.

Para evitar que esto suceda, deberías planificar la imposición de cuotas a los espacios de nombres en un entorno con varios inquilinos para obligar a los inquilinos a especificar las solicitudes y los límites al programar sus pods en el clúster. También mitigará una posible denegación de servicio al limitar la cantidad de recursos que puede consumir un pod.

También puedes usar las cuotas para distribuir los recursos del clúster y adaptarlos a los gastos del inquilino. Esto es particularmente útil en el escenario de KaaS.

Prioridad y preferencia de los pods

La prioridad y la prioridad de los pods pueden resultar útiles si quieres dar más importancia a un pod que a otros pods. Por ejemplo, con la prioridad de los pods, puede configurar los pods del cliente A para que se ejecuten con una prioridad mayor que la del cliente B. Cuando no haya suficiente capacidad disponible, el programador desalojará los pods de menor prioridad del cliente B para dar cabida a los pods de mayor prioridad del cliente A. Esto puede resultar especialmente útil en un entorno SaaS en el que los clientes dispuestos a pagar una prima reciben una prioridad más alta.

> **Important**
>
> La prioridad de los pods puede tener un efecto no deseado en otros pods con menor prioridad. Por ejemplo, aunque los pods afectados se cierren correctamente, pero no PodDisruptionBudget está garantizado, lo que podría interrumpir una aplicación de menor prioridad que dependa de un quórum de pods (consulta Limitaciones de la preferencia).

Controles atenuantes

Su principal preocupación como administrador de un entorno multiusuario es evitar que un atacante acceda al host subyacente. Se deben tener en cuenta los siguientes controles para mitigar este riesgo:

Entornos de ejecución aislados para contenedores

El sandboxing es una técnica mediante la cual cada contenedor se ejecuta en su propia máquina virtual aislada. Entre las tecnologías que utilizan el sandboxing en cápsulas se incluyen Firecracker y Firekube de Weave.

Para obtener información adicional sobre el intento de convertir Firecracker en un entorno de ejecución compatible con EKS, consulta el archivo 1238496944684597248.html. https://threadreaderapp.com/thread/

Open Policy Agent (OPA) y Gatekeeper

Gatekeeper es un controlador de admisión de Kubernetes que aplica las políticas creadas con OPA. Con OPA, puedes crear una política que ejecute los pods de los inquilinos en instancias distintas o con una prioridad mayor que la de otros inquilinos. Puedes encontrar una colección de políticas de OPA comunes en el GitHub repositorio de este proyecto.

También hay un complemento OPA experimental para CoreDNS que le permite usar OPA filter/control para los registros devueltos por CoreDNS.

Kyverno

Kyverno es un motor de políticas nativo de Kubernetes que puede validar, mutar y generar configuraciones con políticas como recursos de Kubernetes. Kyverno utiliza superposiciones de estilo Kustomize para la validación, admite el parche JSON y el parche de fusión estratégica para la mutación, y puede clonar recursos en distintos espacios de nombres en función de activadores flexibles.

Puedes usar Kyverno para aislar los espacios de nombres, reforzar la seguridad de los módulos y otras prácticas recomendadas, y generar configuraciones predeterminadas, como políticas de red. En el GitHub repositorio de este proyecto se incluyen varios ejemplos. Muchos otros están incluidos en la biblioteca de políticas del sitio web de Kyverno.

Aislar las cargas de trabajo de los inquilinos en nodos específicos

Se puede restringir las cargas de trabajo arrendatarias para que se ejecuten en nodos específicos para aumentar el aislamiento en el modelo flexible de tenencia múltiple. Con este enfoque, las cargas de trabajo específicas de los inquilinos solo se ejecutan en los nodos aprovisionados para los inquilinos respectivos. Para lograr este aislamiento, se utilizan las propiedades nativas de Kubernetes (afinidad entre los nodos y limitaciones y tolerancias) para programar los pods en nodos específicos e impedir que los pods, procedentes de otros inquilinos, se programen en los nodos específicos de cada inquilino.

Parte 1: Afinidad de nodos

La afinidad de nodos de Kubernetes se utiliza para planificar los nodos según las etiquetas de los nodos. Con las reglas de afinidad de nodos, los pods se sienten atraídos por nodos específicos que coinciden con los términos del selector. En la siguiente especificación de pod, la afinidad de `requiredDuringSchedulingIgnoredDuringExecution` nodos se aplica al pod respectivo. El resultado es que el pod se dirigirá a los nodos que estén etiquetados con la siguiente clave/valor:. `node-restriction.kubernetes.io/tenant: tenants-x`

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

Con esta afinidad de nodos, la etiqueta es obligatoria durante la programación, pero no durante la ejecución; si las etiquetas de los nodos subyacentes cambian, los pods no se desalojarán únicamente por ese cambio de etiqueta. Sin embargo, la programación futura podría verse afectada.

> **Warning**
>
> El prefijo de etiqueta de `node-restriction.kubernetes.io/` tiene un significado especial en Kubernetes. NodeRestrictionque está habilitado para los clústeres de EKS,

`kubelet` impide las adding/removing/updating etiquetas con este prefijo. Los atacantes no pueden usar el `kubelet's credentials to update the node object or modify the system setup to pass these labels into `kubelet` as y `kubelet` no está permitido modificar estas etiquetas. Si se utiliza este prefijo para toda la programación de un pod a un nodo, se evitan situaciones en las que un atacante quiera atraer un conjunto diferente de cargas de trabajo a un nodo modificando las etiquetas de los nodos.

Example

En lugar de la afinidad de nodos, podríamos haber utilizado el selector de nodos. Sin embargo, la afinidad entre nodos es más expresiva y permite tener en cuenta más condiciones a la hora de programar los módulos. Para obtener información adicional sobre las diferencias y opciones de programación más avanzadas, consulte esta entrada del blog de la CNCF sobre la programación avanzada de un pod a un nodo de Kubernetes.

Parte 2: Manchas y tolerancias

Atraer cápsulas a los nodos es solo la primera parte de este enfoque de tres partes. Para que este enfoque funcione, debemos evitar que los pods se programen en nodos para los que los pods no estén autorizados. Para repeler los pods no deseados o no autorizados, Kubernetes utiliza nodos contaminados. Las contaminaciones se utilizan para imponer condiciones en los nodos que impiden la programación de los pods. La siguiente mancha utiliza un par clave-valor de. `tenant: tenants-x`

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

Dado el nodo anterior`taint`, solo los grupos que toleren la contaminación podrán programarse en el nodo. Para poder programar la entrada de cápsulas autorizadas en el nodo, las especificaciones de las cápsulas correspondientes deben incluir una referencia `toleration` a la contaminación, tal y como se muestra a continuación.

```
...
  tolerations:
```

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

No se `toleration` impedirá que los pods con las características anteriores se programen en el nodo, al menos no debido a esa imperfección específica. Kubernetes también utiliza las manchas para detener temporalmente la programación de los pods en determinadas condiciones, como la presión de los recursos del nodo. Gracias a la afinidad entre nodos y a las contaminaciones y tolerancias, podemos atraer eficazmente los grupos deseados a nodos específicos y repeler los grupos no deseados.

> **Important**
>
> Se requieren algunos pods de Kubernetes para ejecutarse en todos los nodos. Algunos ejemplos de estos pods son los iniciados por la interfaz de red de contenedores (CNI) y los daemonsets kube-proxy. Con ese fin, las especificaciones de estos módulos contienen tolerancias muy permisivas para tolerar diferentes tipos de contaminación. Se debe tener cuidado de no cambiar estas tolerancias. El cambio de estas tolerancias podría provocar un funcionamiento incorrecto del clúster. Además, las herramientas de administración de políticas, como OPA/Gatekeeper y Kyverno, se pueden utilizar para redactar políticas de validación que impidan que grupos no autorizados utilicen estas tolerancias permisivas.

Parte 3: Gestión basada en políticas para la selección de nodos

Hay varias herramientas que se pueden utilizar para ayudar a gestionar la afinidad de los nodos y las tolerancias de las especificaciones de los módulos, incluida la aplicación de las normas en los procesos de CICD. Sin embargo, la aplicación del aislamiento también debe hacerse a nivel de clúster de Kubernetes. Para ello, se pueden utilizar herramientas de gestión de políticas para modificar las solicitudes entrantes del servidor API de Kubernetes, en función de la carga útil de las solicitudes, a fin de aplicar las reglas y tolerancias de afinidad de nodos respectivas mencionadas anteriormente.

Por ejemplo, los pods destinados al espacio de nombres tenants-x se pueden marcar con la afinidad y tolerancia de nodos correctas para permitir la programación en los nodos tenants-x. Al utilizar herramientas de administración de políticas configuradas mediante el webhook de admisión mutante de Kubernetes, las políticas se pueden utilizar para modificar las especificaciones de los pods

entrantes. Las mutaciones añaden los elementos necesarios para permitir la programación deseada. A continuación se muestra un ejemplo de OPA/Gatekeeper política que añade una afinidad de nodos.

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

La política anterior se aplica a una solicitud de servidor de API de Kubernetes para aplicar un pod al espacio de nombres tenants-x. La política añade la regla de afinidad de `requiredDuringSchedulingIgnoredDuringExecution` nodos para que los pods se sientan atraídos por los nodos con la etiqueta. `tenant: tenants-x`

Una segunda política, que se muestra a continuación, añade la tolerancia a la misma especificación de pod, utilizando los mismos criterios de coincidencia entre el espacio de nombres y los grupos, tipos y versiones de destino.

```
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"
```

Las políticas anteriores son específicas de los grupos; esto se debe a las rutas hacia los elementos mutados en los elementos de las políticas. `location` Se podrían escribir políticas adicionales para gestionar los recursos que crean pods, como los recursos de Deployment y Job. Las políticas enumeradas y otros ejemplos se pueden ver en el GitHubproyecto complementario de esta guía.

El resultado de estas dos mutaciones es que las vainas son atraídas hacia el nódulo deseado y, al mismo tiempo, no son repelidas por el nudo específico. Para comprobarlo, podemos ver los fragmentos del resultado de dos `kubectl` llamadas para obtener los nodos etiquetados con `tenant=tenants-x` ellos y colocar los pods en el espacio de nombres. `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...
```

Como podemos ver en los resultados anteriores, todos los pods están programados en los nodos etiquetados con. `tenant=tenants-x` En pocas palabras, los pods solo se ejecutarán en los nodos deseados y los otros pods (sin la afinidad y las tolerancias requeridas) no. Las cargas de trabajo de los inquilinos están aisladas de manera efectiva.

A continuación se muestra un ejemplo de especificación de un módulo mutado.

```
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**
>
> Las herramientas de administración de políticas que se integran en el flujo de solicitudes del servidor de la API de Kubernetes, que utilizan webhooks de admisión que mutan y validan, están diseñadas para responder a la solicitud del servidor de la API en un plazo específico. Suele ser de 3 segundos o menos. Si la llamada al webhook no devuelve una respuesta dentro del tiempo configurado, la and/or validación de la mutación de la solicitud entrante del servidor de API puede producirse o no. Este comportamiento se basa en si las configuraciones del webhook de admisión están configuradas en Fail Open o Fail Close.

En los ejemplos anteriores, utilizamos políticas escritas para OPA/Gatekeeper. Sin embargo, hay otras herramientas de administración de políticas que también se encargan de nuestro caso de uso de selección de nodos. Por ejemplo, esta política de Kyverno podría usarse para gestionar la mutación de afinidad nodal.

> **Note**
>
> Si funciona correctamente, la modificación de las políticas efectuará los cambios deseados en las cargas útiles de las solicitudes entrantes del servidor API. Sin embargo, también se deben incluir políticas de validación para verificar que se produzcan los cambios deseados antes de permitir que los cambios persistan. Esto es especialmente importante cuando se utilizan estas políticas para el tenant-to-node aislamiento. También es una buena idea incluir políticas de auditoría para comprobar de forma rutinaria el clúster en busca de configuraciones no deseadas.

Referencias

- k-rail Diseñado para ayudarle a proteger un entorno multiusuario mediante la aplicación de determinadas políticas.
- Prácticas de seguridad para aplicaciones MultiTenant SaaS que utilizan Amazon EKS

Tenencia múltiple dura

La multitenencia estricta se puede implementar mediante el aprovisionamiento de clústeres separados para cada inquilino. Si bien esto proporciona un aislamiento muy fuerte entre los inquilinos, tiene varios inconvenientes.

En primer lugar, cuando tiene muchos inquilinos, este enfoque puede resultar caro rápidamente. No solo tendrá que pagar los costos del plano de control de cada clúster, sino que tampoco podrá compartir los recursos de cómputo entre los clústeres. Con el tiempo, esto provocará una fragmentación, en la que un subconjunto de los clústeres se subutilizará mientras que otros se sobreutilizarán.

En segundo lugar, es probable que necesite comprar o construir herramientas especiales para administrar todos estos clústeres. Con el tiempo, administrar cientos o miles de clústeres puede resultar simplemente demasiado engorroso.

Por último, la creación de un clúster por inquilino será lenta en comparación con la creación de un espacio de nombres. Sin embargo, puede ser necesario un enfoque de tenencia estricta en industrias altamente reguladas o en entornos de SaaS donde se requiere un fuerte aislamiento.

Orientaciones futuras

La comunidad de Kubernetes ha reconocido las deficiencias actuales de la multitenencia flexible y los desafíos que plantea la multitenencia dura. El Multi-Tenancy Special Interest Group (SIG) está intentando abordar estas deficiencias mediante varios proyectos de incubación, como el controlador jerárquico de espacios de nombres (HNC) y el clúster virtual.

La propuesta del HNC (KEP) describe una forma de crear relaciones padre-hijo entre los espacios de nombres mediante la herencia de objetos [normativos], además de la posibilidad de que los administradores arrendatarios creen subespacios de nombres.

La propuesta de clúster virtual describe un mecanismo para crear instancias independientes de los servicios del plano de control, que incluyen el servidor API, el administrador del controlador y el programador, para cada inquilino del clúster (también conocido como «Kubernetes en Kubernetes»).

La propuesta de puntos de referencia para varios arrendatarios proporciona pautas para compartir clústeres mediante espacios de nombres para el aislamiento y la segmentación, y una herramienta de línea de comandos llamada kubectl-mtb para validar el cumplimiento de las directrices.

Herramientas y recursos de administración de varios clústeres

- Banzai Cloud
- Kommander
- Lente
- Nirmata
- Rafay
- Ganadero
- Weave Flux

Auditoría y registro

La recopilación y el análisis de los registros [de auditoría] son útiles por diversas razones. Los registros pueden ayudar a analizar y atribuir la causa raíz, es decir, a atribuir un cambio a un usuario concreto. Cuando se han recopilado suficientes registros, también se pueden utilizar para detectar comportamientos anómalos. En EKS, los registros de auditoría se envían a Amazon Cloudwatch Logs. La política de auditoría de EKS es la siguiente:

```
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"
```

Recomendaciones

Habilite los registros de auditoría

Los registros de auditoría forman parte de los registros del plano de control de Kubernetes gestionados por EKS y gestionados por EKS. Las instrucciones para enabling/disabling los registros del plano de control, que incluyen los registros del servidor API de Kubernetes, el administrador del controlador y el programador, junto con el registro de auditoría, se encuentran aquí, -plane-logs.html# -export. https://docs.aws.amazon.com/eks/ latest/userguide/control enabling-control-plane-log

> **Note**
>
> Si habilita el registro en el plano de control, incurrirá en costes por almacenar los registros. CloudWatch Esto plantea un problema más amplio en relación con el coste continuo de la seguridad. En última instancia, tendrá que sopesar esos costes con el coste de una violación de la seguridad, por ejemplo, una pérdida financiera, un daño a su reputación, etc. Es posible que descubra que puede proteger su entorno de manera adecuada si solo implementa algunas de las recomendaciones de esta guía.

> **Warning**
>
> El tamaño máximo de una entrada de CloudWatch registros es de 1 MB, mientras que el tamaño máximo de una solicitud a la API de Kubernetes es de 1,5 MB. Las entradas de registro de más de 1 MB se truncarán o solo incluirán los metadatos de la solicitud.

Utilice los metadatos de auditoría

Los registros de auditoría de Kubernetes incluyen dos anotaciones que indican si una solicitud se autorizó o no `authorization.k8s.io/decision` y el motivo de la decisión. `authorization.k8s.io/reason` Usa estos atributos para determinar por qué se permitió una llamada a la API en particular.

Cree alarmas para eventos sospechosos

Crea una alarma que te avise automáticamente si hay un aumento de 403 respuestas prohibidas y 401 no autorizadas y, a continuación `hostsourceIPs`, utiliza atributos como, y `k8s_user.username` para averiguar de dónde provienen esas solicitudes.

Analice los registros con Log Insights

Utilice CloudWatch Log Insights para monitorear los cambios en los objetos del RBAC, por ejemplo, los roles RoleBindings ClusterRoles, y. ClusterRoleBindings A continuación, se muestran algunos ejemplos de consultas:

Enumera las actualizaciones de `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
```

Muestra la creación de webhooks de validación nuevos o los cambios en ellos:

```
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
```

Enumera las operaciones de creación, actualización y eliminación de los roles:

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

Las listas crean, actualizan y eliminan operaciones para RoleBindings:

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

Las listas crean, actualizan y eliminan operaciones para ClusterRoles:

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

Las listas crean, actualizan y eliminan operaciones para ClusterRoleBindings:

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

Planifica operaciones de lectura no autorizadas de Secrets:

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| filter objectRef.resource="secrets" and verb in ["get", "watch", "list"] and
 responseStatus.code="401"
```

```
| stats count() by bin(1m)
```

Lista de solicitudes anónimas fallidas:

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

Audite sus CloudTrail registros

Los pods que utilizan funciones de IAM para cuentas de servicio (IRSA) APIs llaman a AWS y se registran automáticamente CloudTrail junto con el nombre de la cuenta de servicio. Si en el registro aparece el nombre de una cuenta de servicio que no estaba autorizada explícitamente a llamar a una API, podría indicar que la política de confianza del rol de IAM estaba mal configurada. En términos generales, Cloudtrail es una excelente forma de atribuir las llamadas a la API de AWS a principios de IAM específicos.

Utilice Insights para descubrir actividades sospechosas CloudTrail

CloudTrail Insights analiza automáticamente los eventos de gestión y escritura de los CloudTrail senderos y te alerta de cualquier actividad inusual. Esto puede ayudarlo a identificar cuándo se produce un aumento en el volumen de llamadas por escritura APIs en su cuenta de AWS, incluso de grupos que utilizan IRSA para asumir una función de IAM. Para obtener más información, consulte Announcing CloudTrail Insights: Identify and Response to Unnusual API Activity.

Recursos adicionales

A medida que aumenta el volumen de registros, analizarlos y filtrarlos con Log Insights u otra herramienta de análisis de registros puede resultar ineficaz. Como alternativa, puede considerar la posibilidad de ejecutar Sysdig Falco y ekscloudwatch. Falco analiza los registros de auditoría y detecta las anomalías o los abusos durante un período prolongado. El proyecto ekscloudwatch reenvía los eventos del registro de auditoría a Falco para su análisis. CloudWatch Falco proporciona un conjunto de reglas de auditoría predeterminadas junto con la posibilidad de añadir las suyas propias.

Otra opción podría ser almacenar los registros de auditoría en S3 y utilizar el algoritmo SageMaker Random Cut Forest para detectar comportamientos anómalos que justifiquen una investigación más profunda.

Herramientas y recursos

Los siguientes proyectos comerciales y de código abierto se pueden utilizar para evaluar la alineación del clúster con las mejores prácticas establecidas:

- Taller de inmersión en seguridad de Amazon EKS - Detective Controls
- kubeaudit
- kube-scan asigna una puntuación de riesgo a las cargas de trabajo que se ejecutan en el clúster de acuerdo con el marco del sistema de puntuación de configuración común de Kubernetes
- kubesec.io
- polaris
- estribor
- Snyk
- Kubescape Kubescape es una herramienta de seguridad de Kubernetes de código abierto que escanea clústeres, archivos YAML y gráficos de Helm. Detecta errores de configuración en función de varios marcos (incluidos NSA-CISA y MITRE ATT&CK®).

Seguridad de la red

La seguridad de la red tiene varias facetas. La primera implica la aplicación de reglas que restringen el flujo del tráfico de red entre los servicios. La segunda implica el cifrado del tráfico mientras está en tránsito. Los mecanismos para implementar estas medidas de seguridad en el EKS son variados, pero a menudo incluyen los siguientes elementos:

Control de tráfico

- Políticas de red
- Grupos de seguridad

Network encryption

- Malla de servicios
- Interfaces de red de contenedores (CNIs)
- Controladores de ingreso y balanceadores de carga

- Instancias Nitro
- ACM Private CA con administrador de certificados

Política de red

Dentro de un clúster de Kubernetes, todas las comunicaciones de un pod a otro están permitidas de forma predeterminada. Si bien esta flexibilidad puede ayudar a fomentar la experimentación, no se considera segura. Las políticas de red de Kubernetes te proporcionan un mecanismo para restringir el tráfico de red entre los Pods (a menudo denominado tráfico Este/Oeste), así como entre los Pods y los servicios externos. Las políticas de red de Kubernetes funcionan en las capas 3 y 4 del modelo OSI. Las políticas de red utilizan etiquetas y selectores de espacios de nombres para identificar los pods de origen y destino, pero también pueden incluir direcciones IP, números de puerto, protocolos o una combinación de ambos. Las políticas de red se pueden aplicar a las conexiones entrantes o salientes del pod, lo que suele denominarse reglas de entrada y salida.

Con el soporte nativo de políticas de red del complemento CNI de Amazon VPC, puede implementar políticas de red para proteger el tráfico de red en los clústeres de Kubernetes. Esto se integra con la API básica de políticas de red de Kubernetes, lo que garantiza la compatibilidad y el cumplimiento de los estándares de Kubernetes. Puede definir políticas mediante distintos identificadores compatibles con la API upstream. De forma predeterminada, todo el tráfico de entrada y salida está permitido en un pod. Cuando se especifica una política de red con un tipo de política de entrada, solo se permiten las conexiones al pod desde el nodo del pod y las permitidas por las reglas de entrada. Lo mismo se aplica a las reglas de salida. Si se definen varias reglas, se tiene en cuenta la unión de todas las reglas al tomar la decisión. Por lo tanto, el orden de evaluación no afecta al resultado de la política.

> **Important**
>
> Cuando aprovisiona un clúster de EKS por primera vez, la funcionalidad de política de red CNI de VPC no está habilitada de forma predeterminada. Asegúrese de implementar la versión compatible del complemento CNI de VPC y establezca la `ENABLE_NETWORK_POLICY` marca `true` en el complemento vpc-cni para habilitarla. Consulte la guía del usuario de Amazon EKS para obtener instrucciones detalladas.

Recomendaciones

Cómo empezar con las políticas de red: siga el principio del mínimo privilegio

Cree una política de denegación predeterminada

Al igual que con las políticas RBAC, se recomienda seguir los principios de acceso con menos privilegios en las políticas de red. Comience por crear una política de denegación total que restrinja todo el tráfico entrante y saliente dentro de un espacio de nombres.

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

default-deny

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

> **Note**
>
> La imagen de arriba fue creada por el visualizador de políticas de red de Tufin.

Cree una regla para permitir las consultas de DNS

Una vez que hayas establecido la regla de denegación de todo predeterminada, puedes empezar a aplicar reglas adicionales, como una regla que permita a los pods consultar CoredNS para la resolución de nombres.

```
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	

Agrega reglas de forma incremental para permitir de forma selectiva el flujo de tráfico entre espacios de nombres o pods

Comprenda los requisitos de la aplicación y cree reglas de entrada y salida detalladas según sea necesario. El siguiente ejemplo muestra cómo restringir el tráfico de entrada del puerto 80 al puerto de origen. `app-one` `client-one` Esto ayuda a minimizar la superficie de ataque y reduce el riesgo de acceso no autorizado.

```
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	

Supervisión del cumplimiento de las políticas de red

- Utilice el editor de políticas de red
  - El editor de políticas de red ayuda con las visualizaciones, la puntuación de seguridad y se genera automáticamente a partir de los registros de flujo de la red
  - Cree políticas de red de forma interactiva
- Registros de auditoría
  - Revise periódicamente los registros de auditoría de su clúster EKS
  - Los registros de auditoría proporcionan abundante información sobre las acciones que se han realizado en su clúster, incluidos los cambios en las políticas de red
  - Utilice esta información para realizar un seguimiento de los cambios en las políticas de red a lo largo del tiempo y detectar cualquier cambio no autorizado o inesperado
- Pruebas automatizadas
  - Implemente pruebas automatizadas creando un entorno de pruebas que refleje su entorno de producción e implemente periódicamente cargas de trabajo que intenten infringir sus políticas de red.
- Monitorización de métricas
  - Configure sus agentes de observabilidad para extraer las métricas de Prometheus de los agentes del nodo CNI de la VPC, lo que permite monitorear el estado del agente y los errores del SDK.

- Audite las políticas de red con regularidad
  - Audite periódicamente sus políticas de red para asegurarse de que cumplen con los requisitos actuales de su aplicación. A medida que su aplicación evoluciona, una auditoría le brinda la oportunidad de eliminar las reglas redundantes de entrada y salida y asegurarse de que sus aplicaciones no tengan permisos excesivos.
- Asegúrese de que existan políticas de red mediante Open Policy Agent (OPA)
  - Utilice la política de OPA como se muestra a continuación para garantizar que la política de red esté siempre vigente antes de incorporar los pods de aplicaciones. Esta política deniega la incorporación de los pods k8s con una etiqueta `k8s-app: sample-app` si no existe la política de red correspondiente.

```
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
}
```

Resolución de problemas

Supervise los registros del agente de nodo vpc-network-policy-controller

Habilite los registros del administrador del controlador del plano de control EKS para diagnosticar la funcionalidad de la política de red. Puede transmitir los registros del plano de control a un grupo de CloudWatch registros y utilizar CloudWatchLog Insights para realizar consultas avanzadas. En los registros, puede ver qué objetos de punto final del pod se han resuelto según una política de red, el estado de conciliación de las políticas y depurar si la política funciona según lo previsto.

Además, Amazon VPC CNI le permite habilitar la recopilación y exportación de registros de aplicación de políticas a Amazon Cloudwatch desde los nodos de trabajo de EKS. Una vez activado, puede utilizar CloudWatchContainer Insights para obtener información sobre su uso en relación con las políticas de red.

Amazon VPC CNI también incluye un SDK que proporciona una interfaz para interactuar con los programas eBPF del nodo. El SDK se instala cuando `aws-node` se implementa en los nodos. Puede encontrar el binario del SDK instalado en el `/opt/cni/bin` directorio del nodo. En el momento del lanzamiento, el SDK ofrece soporte para funcionalidades fundamentales, como la inspección de programas y mapas de eBPF.

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

Registra los metadatos del tráfico de red

Los registros de flujo de VPC de AWS recopilan metadatos sobre el tráfico que fluye a través de una VPC, como la dirección IP y el puerto de origen y destino, junto con los paquetes aceptados o descartados. Esta información podría analizarse para detectar actividades sospechosas o inusuales entre los recursos de la VPC, incluidos los pods. Sin embargo, dado que las direcciones IP de los pods cambian con frecuencia a medida que se sustituyen, es posible que los registros de flujo no sean suficientes por sí solos. Calico Enterprise amplía los registros de flujo con etiquetas de los pods y otros metadatos, lo que facilita el desciframiento de los flujos de tráfico entre los pods.

Grupos de seguridad

EKS utiliza los grupos de seguridad de VPC de AWS (SGs) para controlar el tráfico entre el plano de control de Kubernetes y los nodos de trabajo del clúster. Los grupos de seguridad también se utilizan para controlar el tráfico entre los nodos de trabajo y otros recursos de VPC y las direcciones IP externas. Al aprovisionar un clúster de EKS (con la versión 1.14-eks.3 o superior de Kubernetes), se crea automáticamente un grupo de seguridad de clúster. Este grupo de seguridad permite una comunicación sin restricciones entre el plano de control de EKS y los nodos de los grupos de nodos gestionados. Para simplificar, se recomienda añadir el clúster SG a todos los grupos de nodos, incluidos los grupos de nodos no gestionados.

Antes de la versión 1.14 de Kubernetes y la versión eks.3 de EKS, había grupos de seguridad independientes configurados para el plano de control y los grupos de nodos de EKS. Las reglas mínimas y sugeridas para los grupos de seguridad del plano de control y los grupos de nodos se encuentran en -group-reqs.html. https://docs.aws.amazon.com/eks/ latest/userguide/sec Las reglas mínimas para el grupo de seguridad del plano de control permiten que el puerto 443 entre desde el

nodo de trabajo SG. Esta regla es la que permite a los kubelets comunicarse con el servidor API de Kubernetes. También incluye el puerto 10250 para el tráfico saliente al nodo de trabajo SG; 10250 es el puerto en el que escuchan los kubelets. Del mismo modo, las reglas mínimas del grupo de nodos permiten que el puerto 10250 entre desde el plano de control SG y el 443 que salga desde el plano de control SG. Por último, hay una regla que permite la comunicación sin restricciones entre los nodos de un grupo de nodos.

Si necesita controlar la comunicación entre los servicios que se ejecutan dentro del clúster y los que se ejecutan fuera del clúster, como una base de datos de RDS, considere la posibilidad de utilizar grupos de seguridad para los pods. Con los grupos de seguridad para los pods, puede asignar un grupo de seguridad existente a un conjunto de pods.

> **Warning**
>
> Si haces referencia a un grupo de seguridad que no existía antes de la creación de los pods, estos no se programarán.

Puedes controlar qué pods se asignan a un grupo de seguridad creando un `SecurityGroupPolicy` objeto y especificando un `PodSelector` o un`ServiceAccountSelector`. Si se configuran los selectores en, `{}` se asignará lo SGs referenciado `SecurityGroupPolicy` a todos los pods de un espacio de nombres o a todas las cuentas de servicio de un espacio de nombres. Asegúrese de familiarizarse con todas las consideraciones antes de implementar grupos de seguridad para los pods.

> **Important**
>
> Si los usa SGs para pods, debe crear módulos SGs que permitan la salida del puerto 53 al grupo de seguridad del clúster. Del mismo modo, debe actualizar el grupo de seguridad del clúster para que acepte el tráfico entrante del puerto 53 procedente del grupo de seguridad del pod.

> **Important**
>
> Los límites para los grupos de seguridad se siguen aplicando cuando se utilizan grupos de seguridad para los pods, así que utilícelos con prudencia.

> **Important**
>
> Debes crear reglas para el tráfico entrante desde el grupo de seguridad del clúster (kubelet) para todas las sondas configuradas para el pod.

> **Important**
>
> Los grupos de seguridad para los pods se basan en una función conocida como enlace troncal ENI, que se creó para aumentar la densidad de ENI de una instancia. EC2 Cuando se asigna un pod a un SG, un controlador de VPC asocia un ENI de rama del grupo de nodos al pod. Si no hay suficientes ramas ENIs disponibles en un grupo de nodos en el momento en que el pod está programado, el pod permanecerá en estado pendiente. El número de ramas que puede admitir ENIs una instancia varía según la instancia type/family. See https://docs.aws.amazon.com/eks/latest/userguide/security (groups-for-pods.html# supported-instance-types para obtener más información).

Si bien los grupos de seguridad para pods ofrecen una forma nativa de AWS de controlar el tráfico de red dentro y fuera del clúster sin la sobrecarga de un daemon de políticas, hay otras opciones disponibles. Por ejemplo, el motor de políticas de Cilium le permite hacer referencia a un nombre de DNS en una política de red. Calico Enterprise incluye una opción para asignar políticas de red a los grupos de seguridad de AWS. Si ha implementado una red de servicios como Istio, puede utilizar una pasarela de salida para restringir la salida de la red a dominios o direcciones IP específicos y totalmente cualificados. Para obtener más información sobre esta opción, lea la serie de tres partes sobre el control del tráfico de salida en Istio.

¿Cuándo usar la política de red o el grupo de seguridad para los pods?

¿Cuándo usar la política de red de Kubernetes?

- pod-to-podControlar el tráfico
  - Adecuado para controlar el tráfico de red entre los módulos de un clúster (tráfico de este a oeste)
- Controle el tráfico a nivel de puerto o dirección IP (nivel 3 o 4 de OSI)

Cuándo usar los grupos de seguridad de AWS para los pods (SGP)

- Aproveche las configuraciones de AWS existentes
  - Si ya tiene un conjunto complejo de grupos de EC2 seguridad que administran el acceso a los servicios de AWS y va a migrar aplicaciones de EC2 instancias a EKS, SGPs puede ser una muy buena opción, ya que le permite reutilizar los recursos de los grupos de seguridad y aplicarlos a sus pods.
- Controle el acceso a los servicios de AWS
  - Sus aplicaciones que se ejecutan en un clúster de EKS desean comunicarse con otros servicios de AWS (base de datos RDS) y SGPs utilizarlos como un mecanismo eficaz para controlar el tráfico de los pods a los servicios de AWS.
- Aislamiento del tráfico de pods y nodos
  - Si quieres separar completamente el tráfico del pod del resto del tráfico de nodos, usa el `POD_SECURITY_GROUP_ENFORCING_MODE=strict` modo SGP in.

Prácticas recomendadas para el uso de grupos de seguridad para los pods y de la política de red

- Seguridad por capas
  - Utilice una combinación de la política de red de SGP y Kubernetes para lograr un enfoque de seguridad por capas
  - Se usa SGPs para limitar el acceso a nivel de red a los servicios de AWS que no forman parte de un clúster, mientras que las políticas de red de Kubernetes pueden restringir el tráfico de red entre los pods dentro del clúster.
- Principio de privilegios mínimos
  - Permita únicamente el tráfico necesario entre los pods o los espacios de nombres
- Segmente sus aplicaciones
  - Siempre que sea posible, segmente las aplicaciones según la política de red para reducir el radio de cobertura en caso de que una aplicación se vea comprometida
- Mantenga las políticas simples y claras
  - Las políticas de red de Kubernetes pueden ser bastante detalladas y complejas, por lo que es mejor mantenerlas lo más simples posible para reducir el riesgo de errores de configuración y aliviar la sobrecarga de administración

- Reduzca la superficie de ataque
  - Minimice la superficie de ataque limitando la exposición de sus aplicaciones

> **Important**
>
> Security Groups for pods ofrece dos modos de aplicación: `strict` y`standard`. Debe usar el `standard` modo cuando utilice tanto la política de red como los grupos de seguridad para las funciones de los pods en un clúster de EKS.

Cuando se trata de la seguridad de la red, un enfoque por capas suele ser la solución más eficaz. El uso combinado de la política de red de Kubernetes y el SGP puede proporcionar una defense-in-depth estrategia sólida para las aplicaciones que se ejecutan en EKS.

Aplicación de políticas de Service Mesh o política de red de Kubernetes

A `service mesh` es una capa de infraestructura dedicada que puede añadir a sus aplicaciones. Le permite añadir de forma transparente funciones como la observabilidad, la gestión del tráfico y la seguridad, sin necesidad de añadirlas a su propio código.

Service Mesh aplica las políticas en la capa 7 (aplicación) del modelo OSI, mientras que las políticas de red de Kubernetes funcionan en la capa 3 (red) y la capa 4 (transporte). Hay muchas ofertas en este espacio, como AWSAppMesh, Istio, Linkerd, etc.

¿Cuándo usar Service Mesh para la aplicación de políticas

- ¿Tiene una inversión existente en una malla de servicios
- ¿Necesita capacidades más avanzadas, como la gestión del tráfico, la observabilidad y la seguridad
  - Control de tráfico, balanceo de carga, interrupción de circuitos, limitación de velocidad, tiempos de espera, etc.
  - Información detallada sobre el rendimiento de sus servicios (latencia, tasas de error, solicitudes por segundo, volúmenes de solicitudes, etc.)
  - ¿Desea implementar y aprovechar la malla de servicios para funciones de seguridad como los mTLS

Elija la política de red de Kubernetes para casos de uso más sencillos

- Limite los módulos que pueden comunicarse entre sí
- Las políticas de red requieren menos recursos que una malla de servicios, lo que las convierte en una buena opción para casos de uso más simples o para clústeres más pequeños en los que la sobrecarga de ejecución y administración de una malla de servicios puede no estar justificada

> **Note**
>
> Las políticas de red y la malla de servicios también se pueden usar juntas. Utilice las políticas de red para proporcionar un nivel básico de seguridad y aislamiento entre sus módulos y, a continuación, utilice una malla de servicios para añadir funciones adicionales, como la gestión del tráfico, la observabilidad y la seguridad.

ThirdParty Motores de políticas de red

Considere la posibilidad de utilizar un motor de políticas de red de terceros si tiene requisitos de política avanzados, como políticas de red globales, compatibilidad con reglas basadas en nombres de servidor DNS, reglas de capa 7, reglas ServiceAccount basadas y deny/log acciones explícitas, etc., Calico es un motor de políticas de código abierto de Tigera que funciona bien con EKS. Además de implementar el conjunto completo de funciones de política de red de Kubernetes, Calico admite políticas de red ampliadas con un conjunto más amplio de funciones, incluida la compatibilidad con reglas de capa 7, por ejemplo, HTTP, cuando se integra con Istio. Las políticas de Calico se pueden aplicar a los espacios de nombres, los pods, las cuentas de servicio o a nivel mundial. Cuando las políticas se aplican a una cuenta de servicio, se asocia un conjunto de ingress/egress reglas a esa cuenta de servicio. Con las reglas RBAC adecuadas, puede evitar que los equipos las invaliden, lo que permite a los profesionales de la seguridad de TI delegar de forma segura la administración de los espacios de nombres. Isovalent, empresa encargada del mantenimiento de Cilium, también ha ampliado las políticas de red para incluir una compatibilidad parcial con las reglas de la capa 7, por ejemplo, HTTP. Cilium también admite nombres de host DNS, lo que puede resultar útil para restringir el tráfico entre Kubernetes Services/Pods y los recursos que se ejecutan dentro o fuera de la VPC. Por el contrario, Calico Enterprise incluye una función que le permite asignar una política de red de Kubernetes a un grupo de seguridad de AWS, así como a los nombres de host de DNS.

Puede encontrar una lista de las políticas de red habituales de Kubernetes en. https://github.com/ahmetb/ kubernetes-network-policy-recipes Encontrarás un conjunto similar de reglas para Calico en https://docs.projectcalico. org/security/calico-política de red.

Migración al motor de políticas de red CNI de Amazon VPC

Para mantener la coherencia y evitar un comportamiento inesperado de comunicación entre los pods, se recomienda implementar solo un motor de políticas de red en el clúster. Si quieres migrar del motor de políticas de red CNI de 3P a VPC, te recomendamos convertir tu NetworkPolicy CRDs 3P existente a los recursos de NetworkPolicy Kubernetes antes de habilitar el soporte de políticas de red CNI de VPC. Además, pruebe las políticas migradas en un clúster de prueba independiente antes de aplicarlas en su entorno de producción. Esto le permite identificar y abordar cualquier posible problema o incoherencia en el comportamiento de comunicación de los módulos.

Herramienta de migración

Para ayudarte en el proceso de migración, hemos desarrollado una herramienta llamada K8s Network Policy Migrator que convierte tu política de red existente en políticas CRDs de Calico/ Cilium red nativas de Kubernetes. Tras la conversión, puede probar directamente las políticas de red convertidas en los nuevos clústeres que ejecutan el controlador de políticas de red CNI de VPC. La herramienta está diseñada para ayudarte a agilizar el proceso de migración y garantizar una transición fluida.

> **Important**
>
> La herramienta de migración solo convertirá las políticas de 3P que sean compatibles con la API de políticas de red nativa de Kubernetes. Si utilizas las funciones de política de red avanzadas que ofrecen los complementos de 3P, la herramienta de migración las omitirá e informará de ellas.

Tenga en cuenta que, actualmente, el equipo de ingeniería de políticas de red CNI de AWS VPC no admite la herramienta de migración, sino que se pone a disposición de los clientes haciendo todo lo posible. Le recomendamos que utilice esta herramienta para facilitar el proceso de migración. En caso de que encuentre algún problema o error con la herramienta, le rogamos que cree un GitHubproblema. Sus comentarios son muy valiosos para nosotros y nos ayudarán a mejorar continuamente nuestros servicios.

Recursos adicionales

- Kubernetes y Tigera: políticas de red, seguridad y auditoría
- Calico Enterprise
- Cilium
- NetworkPolicyEditor: un editor de políticas interactivo de Cilium
- El dispositivo Inspektor Gadget aconseja políticas de red Sugiere políticas de red basadas en un análisis del tráfico de la red

Cifrado en tránsito

Es posible que las aplicaciones que deban cumplir con la PCI, la HIPAA u otras normativas necesiten cifrar los datos mientras están en tránsito. Hoy en día, el TLS es la opción de facto para cifrar el tráfico en línea. El TLS, al igual que su predecesor, el SSL, proporciona comunicaciones seguras a través de una red mediante protocolos criptográficos. El TLS utiliza un cifrado simétrico, en el que las claves para cifrar los datos se generan en función de un secreto compartido que se negocia al principio de la sesión. A continuación, se muestran algunas formas de cifrar datos en un entorno de Kubernetes.

Instancias Nitro

El tráfico que se intercambia entre los siguientes tipos de instancias de Nitro, por ejemplo, C5n, G4, I3en, M5dn, M5n, P3dn, R5dn y R5n, se cifra automáticamente de forma predeterminada. Cuando hay un salto intermedio, como una pasarela de tránsito o un balanceador de carga, el tráfico no se cifra. Consulta Cifrado en tránsito para obtener más información sobre el cifrado en tránsito, así como la lista completa de tipos de instancias que admiten el cifrado de red de forma predeterminada.

Interfaces de red de contenedores (CNIs)

WeaveNetse puede configurar para cifrar automáticamente todo el tráfico mediante el NaCl cifrado para el tráfico intermedio y el IPsec ESP para el tráfico de rutas de datos rápidas.

Service Mesh

El cifrado en tránsito también se puede implementar con una malla de servicios como App Mesh, Linkerd v2 e Istio. AppMesh admite mTLs con certificados X.509 o el Servicio de Descubrimiento Secreto (SDS) de Envoy. Tanto Linkerd como Istio admiten mTLS.

El aws-app-mesh-examplesGitHub repositorio proporciona instrucciones para configurar los mTLs mediante certificados X.509 y SPIRE como proveedor de SDS con su contenedor Envoy:

- Configuración de los mTLS mediante certificados X.509
- Configuración de TLS mediante SPIRE (SDS)

App Mesh también admite el cifrado TLS con un certificado privado emitido por AWS Certificate Manager (ACM) o un certificado almacenado en el sistema de archivos local del nodo virtual.

El aws-app-mesh-examplesGitHub repositorio proporciona instrucciones para configurar el TLS mediante certificados emitidos por ACM y certificados incluidos en el contenedor de Envoy:

- Configuración de TLS con certificados TLS proporcionados por un archivo
- Configuración de TLS con AWS Certificate Manager

Controladores de entrada y balanceadores de carga

Los controladores de ingreso son una forma de enrutar de forma inteligente el 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 procesamiento que consume mucha CPU. Por lo tanto, si tiene la flexibilidad necesaria, intente realizar la descarga de SSL en la entrada o en el balanceador de carga.

Utilice el cifrado con los balanceadores de carga elásticos de AWS

Tanto el AWS Application Load Balancer (ALB) como el Network Load Balancer (NLB) admiten el cifrado de transporte (SSL y TLS). La `alb.ingress.kubernetes.io/certificate-arn` anotación del ALB le permite especificar qué certificados desea añadir al ALB. Si omite la anotación, el controlador intentará añadir certificados a los oyentes que lo requieran haciendo coincidir los certificados de AWS Certificate Manager (ACM) disponibles mediante el campo host. A partir de la versión 1.15 de EKS, puede utilizar la `service.beta.kubernetes.io/aws-load-balancer-ssl-cert` anotación con el NLB, tal y como se muestra en el siguiente ejemplo.

```
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
```

Los siguientes son ejemplos adicionales de terminación. SSL/TLS

- Asegurando el ingreso de EKS con Contour y Let's Encrypt The Way GitOps
- ¿Cómo cancelo el tráfico HTTPS en las cargas de trabajo de Amazon EKS con ACM?

> **Important**
>
> Algunos Ingresses, como el controlador LB de AWS, implementan el SSL/TLS uso de anotaciones en lugar de como parte de la especificación de Ingress.

ACM Private CA con administrador de certificados

Puede habilitar TLS y mTLS para proteger las cargas de trabajo de las aplicaciones de EKS en la entrada, en el pod y entre los pods mediante ACM Private Certificate Authority (CA) y cert-manager, un popular complemento de Kubernetes para distribuir, renovar y revocar certificados. ACM Private CA es una entidad de certificación gestionada, segura y de alta disponibilidad, sin los costes iniciales y de mantenimiento que supone gestionar una entidad de certificación propia. Si utiliza la autoridad de certificación predeterminada de Kubernetes, existe la oportunidad de mejorar su seguridad y cumplir los requisitos de conformidad con ACM Private CA. ACM Private CA protege las claves privadas en módulos de seguridad de hardware FIPS 140-2 de nivel 3 (muy seguros), en comparación con la CA predeterminada que almacena las claves codificadas en la memoria (menos segura). Una CA centralizada también le brinda más control y mejora la auditabilidad de los certificados privados, tanto dentro como fuera de un entorno de Kubernetes.

Modo CA de corta duración para un TLS mutuo entre cargas de trabajo

Cuando utilice ACM Private CA para mTLS en EKS, se recomienda utilizar certificados de corta duración con el modo CA de corta duración. Si bien es posible emitir certificados de corta duración en el modo de CA de uso general, utilizar el modo de CA de corta duración resulta más rentable (aproximadamente un 75% más barato que el modo general) en los casos de uso en los que es necesario emitir nuevos certificados con frecuencia. Además, debería intentar ajustar el período de validez de los certificados privados a la vida útil de los módulos de su clúster de EKS. Obtenga más información sobre ACM Private CA y sus ventajas aquí.

Instrucciones de configuración de ACM

Comience por crear una CA privada siguiendo los procedimientos que se proporcionan en los documentos técnicos de ACM Private CA. Una vez que tenga una CA privada, instale cert-manager siguiendo las instrucciones de instalación habituales. Tras instalar el administrador de certificados, instale el complemento Private CA Kubernetes cert-manager siguiendo las instrucciones de configuración que se indican en. GitHub El complemento permite al administrador de certificados solicitar certificados privados de ACM Private CA.

Ahora que tiene una CA privada y un clúster de EKS con el administrador de certificados y el complemento instalados, es el momento de establecer los permisos y crear el emisor. Actualice los permisos de IAM del rol de nodo EKS para permitir el acceso a ACM Private CA. <CA_ARN>Sustitúyalo por el valor de su CA privada:

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

También se pueden utilizar las funciones de servicio para las cuentas de IAM o IRSA. Consulte la sección de recursos adicionales que aparece a continuación para ver ejemplos completos.

Cree un emisor en Amazon EKS mediante la creación de un archivo de definición de recursos personalizado denominado cluster-issuer.yaml con el siguiente texto y <CA_ARN> sustituya la información por su CA privada. <Region>

```
apiVersion: awspca.cert-manager.io/v1beta1
kind: AWSPCAClusterIssuer
metadata:
```

```
          name: demo-test-root-ca
spec:
          arn: <CA_ARN>
          region: <Region>
```

Implemente el emisor que creó.

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

Su clúster de EKS está configurado para solicitar certificados a Private CA. Ahora puede usar el `Certificate` recurso de cert-manager para emitir certificados cambiando los valores del `issuerRef` campo por el emisor de CA privado que creó anteriormente. Para obtener más información sobre cómo especificar y solicitar los recursos de certificados, consulte la guía de recursos de certificados de cert-manager. Consulte algunos ejemplos aquí.

ACM Private CA con Istio y cert-manager

Si ejecuta Istio en su clúster de EKS, puede deshabilitar el plano de control de Istio (específicamente`istiod`) para que no funcione como autoridad de certificación (CA) raíz y configurar ACM Private CA como CA raíz para los mTLs entre cargas de trabajo. Si opta por este enfoque, considere la posibilidad de utilizar el modo CA de corta duración en ACM Private CA. Consulte la sección anterior y esta entrada del blog para obtener más información.

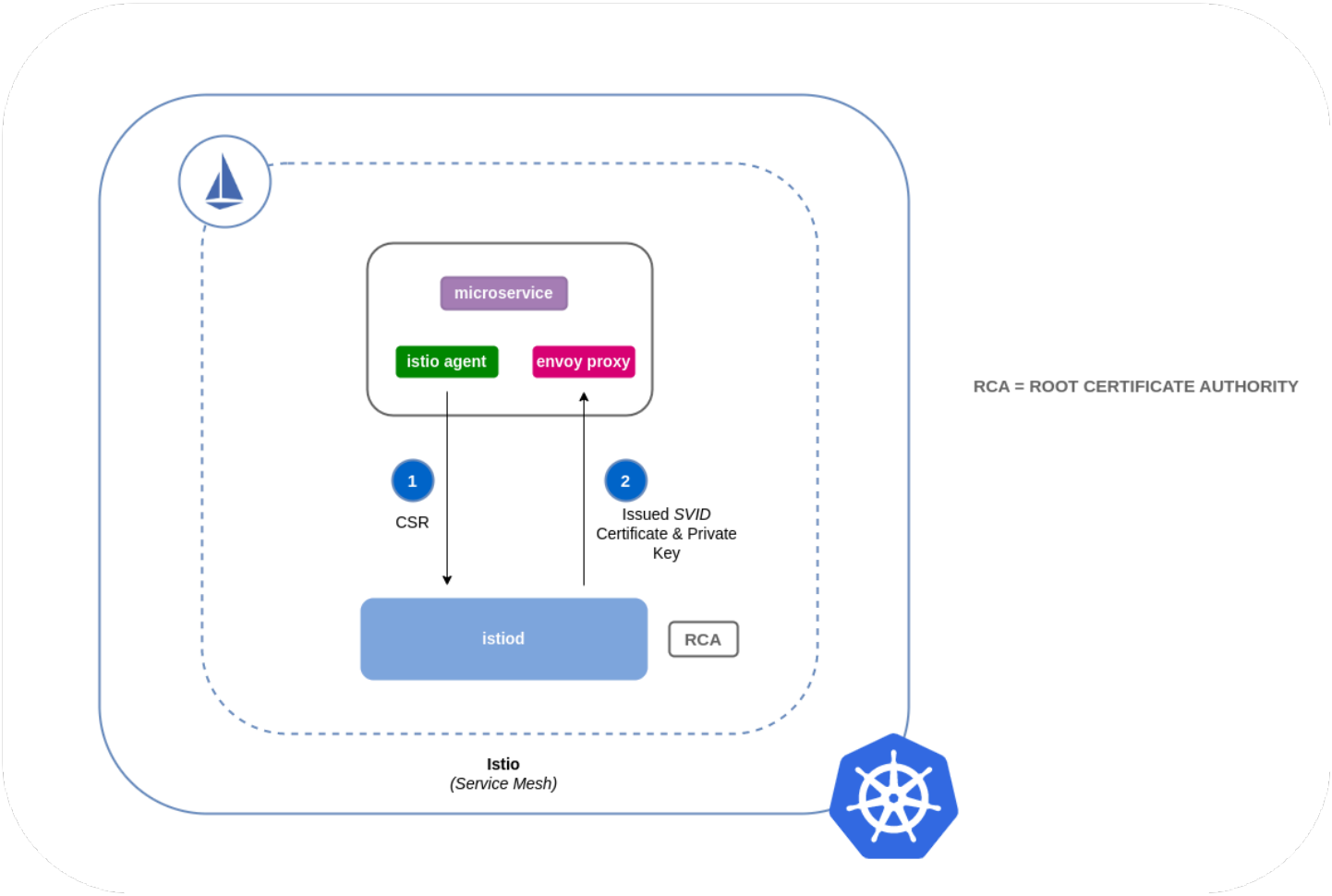
Cómo funciona la firma de certificados en Istio (predeterminado)

Las cargas de trabajo en Kubernetes se identifican mediante cuentas de servicio. Si no especificas una cuenta de servicio, Kubernetes asignará una automáticamente a tu carga de trabajo. Además, las cuentas de servicio montan automáticamente un token asociado. La cuenta de servicio utiliza este token para que las cargas de trabajo se autentiquen en la API de Kubernetes. La cuenta de servicio puede ser suficiente como identidad para Kubernetes, pero Istio tiene su propio sistema de gestión de identidades y su propia entidad de certificación. Cuando una carga de trabajo se inicia con su proxy sidecar, necesita que Istio le asigne una identidad para que se la considere fiable y pueda comunicarse con otros servicios de la red.

Para obtener esta identidad de Istio, `istio-agent` envía una solicitud conocida como solicitud de firma de certificados (o CSR) al plano de control de Istio. Esta CSR contiene el token de la cuenta de servicio para que se pueda verificar la identidad de la carga de trabajo antes de procesarla. Este proceso de verificación lo gestiona `istiod` la autoridad de registro (o RA) y la CA. La RA

actúa como un guardián que se asegura de que solo los CSR verificados lleguen a la CA. Una vez verificada la CSR, se reenviará a la CA, que emitirá un certificado con la identidad de la SPIFFE con la cuenta de servicio. Este certificado se denomina documento de identidad verificable del SPIFFE (o SVID). El SVID se asigna al servicio solicitante con fines de identificación y para cifrar el tráfico en tránsito entre los servicios de comunicación.

Flujo predeterminado para las solicitudes de firma de certificados de Istio:



Cómo funciona la firma de certificados en Istio con ACM Private CA

Puede utilizar un complemento de gestión de certificados denominado agente de solicitud de firma de certificados de Istio (istio-csr) para integrar Istio con ACM Private CA. Este agente permite proteger las cargas de trabajo y los componentes del plano de control de Istio con emisores de gestores de certificados, en este caso ACM Private CA. El agente istio-csr expone el mismo servicio que sirve istiod en la configuración predeterminada de validación de los datos entrantes. CSRs Excepto que, tras la verificación, convertirá las solicitudes en recursos compatibles con el administrador de certificados (es decir, integraciones con emisores de CA externos).

Siempre que haya una CSR derivada de una carga de trabajo, se reenviará a istio-csr, que solicitará los certificados de ACM Private CA. Esta comunicación entre istio-csr y ACM Private CA se habilita mediante el complemento de emisión de AWS Private CA. El administrador de certificados usa este complemento para solicitar certificados TLS a ACM Private CA. El complemento emisor se comunicará con el servicio ACM Private CA para solicitar un certificado firmado para la carga de trabajo. Una vez firmado el certificado, se devolverá a istio-csr, que leerá la solicitud firmada y la devolverá a la carga de trabajo que inició la CSR.

Flujo de solicitudes de firma de certificados de Istio con istio-csr

imagen: istio-csr-with-acm -private-ca.png [Flujo de solicitudes de firma de certificados de Istio con istio-csr]

Instrucciones de configuración de Istio con CA privada

1. Comience por seguir las mismas instrucciones de configuración de esta sección para completar lo siguiente:
2. Crear una entidad de certificación privada
3. Instale cert-manager
4. Instala el complemento emisor
5. Establece los permisos y crea un emisor. El emisor representa a la CA y se utiliza para firmar `istiod` y combinar los certificados de carga de trabajo. Se comunicará con ACM Private CA.
6. Cree un `istio-system` espacio de nombres. Aquí es donde se desplegarán los recursos de Istio `istiod certificate` y otros recursos.
7. Instale Istio CSR configurado con el complemento AWS Private CA Issuer. Puede conservar las solicitudes de firma de certificados para las cargas de trabajo a fin de comprobar que se aprueban y se firman (). `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. Instale Istio con configuraciones personalizadas para reemplazarlo `istiod` `cert-manager istio-csr` como proveedor de certificados para la malla. Este proceso se puede llevar a cabo utilizando el operador Istio.

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

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

          - path: spec.template.spec.containers.[name:discovery].volumeMounts[6]
            value:
              name: cert-manager
```

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

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

9. Implemente el recurso personalizado anterior que creó.

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

10Ahora puede implementar una carga de trabajo en la malla de su clúster de EKS y aplicar las MTL.

Solicitudes de firma de certificados de Istio

imagen: istio-csr-requests .png [Solicitudes de firma de certificados de Istio]

Herramientas y recursos

- Taller de inmersión en seguridad de Amazon EKS: seguridad de redes
- Cómo implementar el administrador de certificados y el complemento ACM Private CA para habilitar el TLS en EKS.
- Configuración del cifrado end-to-end TLS en Amazon EKS con el nuevo AWS Load Balancer Controller y ACM Private CA.
- El complemento privado CA Kubernetes cert-manager está activado. GitHub
- Guía del usuario del complemento privado de gestión de certificados de CA Kubernetes.

- Cómo utilizar el modo de certificación de corta duración de AWS Private Certificate Authority
- egress-operator Un operador y un complemento de DNS para controlar el tráfico de salida de su clúster sin necesidad de inspeccionar el protocolo
- NeuVector de SUSE, una plataforma de seguridad de contenedores de código abierto y de confianza cero, proporciona políticas, normas de red, prevención de pérdidas de datos (DLP), firewall de aplicaciones web (WAF) y firmas de amenazas de red.

Cifrado de datos y gestión de secretos

Cifrado en reposo

Hay tres opciones diferentes de almacenamiento nativo de AWS que puede usar con Kubernetes: EBS, EFS y for Lustre. FSx Las tres ofrecen cifrado en reposo mediante una clave gestionada por el servicio o una clave maestra del cliente (CMK). Para EBS, puede utilizar el controlador de almacenamiento integrado en el árbol o el controlador CSI de EBS. Ambos incluyen parámetros para cifrar volúmenes y suministrar una CMK. Para EFS, puede utilizar el controlador CSI de EFS; sin embargo, a diferencia de EBS, el controlador CSI de EFS no admite el aprovisionamiento dinámico. Si desea utilizar EFS con EKS, tendrá que aprovisionar y configurar el cifrado en reposo para el sistema de archivos antes de crear un PV. Para obtener más información sobre el cifrado de archivos EFS, consulte Cifrado de datos en reposo. Además de ofrecer cifrado en reposo, EFS y FSx for Lustre incluyen una opción para cifrar los datos en tránsito. FSx for Lustre lo hace de forma predeterminada. En el caso de EFS, puede añadir el cifrado de transporte añadiendo el `tls` parámetro a `mountOptions` en su PV, como en este ejemplo:

```
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>
```

El controlador FSx CSI admite el aprovisionamiento dinámico de los sistemas de archivos Lustre. De forma predeterminada, cifra los datos con una clave gestionada por el servicio, aunque existe la opción de proporcionar su propia CMK, como en este ejemplo:

```
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**
>
> A partir del 28 de mayo de 2020, todos los datos escritos en el volumen efímero de los pods de EKS Fargate se cifran de forma predeterminada mediante un algoritmo criptográfico AES-256 estándar del sector. No es necesario realizar ninguna modificación en la aplicación, ya que el servicio gestiona el cifrado y el descifrado sin problemas.

Cifrado de datos en reposo

El cifrado de los datos en reposo se considera una práctica recomendada. Si no estás seguro de si el cifrado es necesario, cifra tus datos.

Gire los suyos periódicamente CMKs

Configure KMS para que rote automáticamente su CMKs. De este modo, las claves se rotarán una vez al año y se guardarán las antiguas indefinidamente para poder seguir descifrando los datos. Para obtener información adicional, consulte Rotación de las claves maestras del cliente

Utilice los puntos de acceso EFS para simplificar el acceso a los conjuntos de datos compartidos

Si ha compartido conjuntos de datos con diferentes permisos de archivos POSIX o desea restringir el acceso a una parte del sistema de archivos compartido mediante la creación de diferentes puntos de montaje, considere la posibilidad de utilizar puntos de acceso EFS. Para obtener más información sobre cómo trabajar con puntos de acceso, consulte https://docs.aws.amazon.com/efs/ latest/ug/ efs -access-points.html. Hoy en día, si quieres usar un punto de acceso (AP), tendrás que hacer referencia al AP en el `volumeHandle` parámetro del PV.

> **Important**
>
> A partir del 23 de marzo de 2021, el controlador CSI de EFS admite el aprovisionamiento dinámico de puntos de acceso EFS. Los puntos de acceso son puntos de entrada específicos de la aplicación a un sistema de archivos EFS que facilitan el uso compartido de un sistema de archivos entre varios pods. Cada sistema de archivos EFS puede tener hasta 120 PVs. Consulte Introducción al aprovisionamiento dinámico de Amazon EFS CSI para obtener más información.

Administración de secretos

Los secretos de Kubernetes se utilizan para almacenar información confidencial, como certificados de usuario, contraseñas o claves de API. Se conservan en etcd como cadenas codificadas en base64. En EKS, los volúmenes de EBS para los nodos etcd se cifran con cifrado EBS. Un pod puede recuperar un objeto secreto de Kubernetes haciendo referencia al secreto de. `podSpec` Estos secretos pueden asignarse a una variable de entorno o montarse como volumen. Para obtener información adicional sobre la creación de secretos, consulte https://kubernetes. io/docs/concepts/ configuration/secret/.

> **Warning**
>
> Todos los pods del espacio de nombres del secreto pueden hacer referencia a los secretos de un espacio de nombres concreto.

> **Warning**
>
> El autorizador de nodos permite al Kubelet leer todos los secretos almacenados en el nodo.

Utilice AWS KMS para el cifrado de sobres de los secretos de Kubernetes

Esto le permite cifrar sus secretos con una clave de cifrado de datos (DEK) única. A continuación, el DEK se cifra con una clave de cifrado clave (KEK) de AWS KMS que se puede rotar automáticamente de forma periódica. Con el complemento KMS para Kubernetes, todos los secretos de Kubernetes se almacenan en etcd en texto cifrado en lugar de texto plano y solo el servidor API de Kubernetes puede descifrarlos. Para obtener más información, consulta el artículo sobre cómo utilizar el soporte del proveedor de cifrado EKS para obtener más información sobre la defensa

Audite el uso de Kubernetes Secrets

En EKS, active el registro de auditoría y cree un filtro de CloudWatch métricas y una alarma que le avisen cuando se utilice un secreto (opcional). El siguiente es un ejemplo de un filtro de métricas para el registro de auditoría de Kubernetes,. `{($.verb="get") && ($.objectRef.resource="secret")}` También puedes usar las siguientes consultas con CloudWatch Log Insights:

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

La consulta anterior mostrará el número de veces que se ha accedido a un secreto en un período de tiempo específico.

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

Esta consulta mostrará el secreto, junto con el espacio de nombres y el nombre de usuario del usuario que intentó acceder al secreto y el código de respuesta.

Cambia tus secretos periódicamente

Kubernetes no rota los secretos automáticamente. Si tiene que ocultar secretos, considere la posibilidad de utilizar un almacén de secretos externo, por ejemplo, Vault o AWS Secrets Manager.

Utilice espacios de nombres separados para aislar los secretos de distintas aplicaciones

Si tiene secretos que no se pueden compartir entre las aplicaciones de un espacio de nombres, cree un espacio de nombres independiente para esas aplicaciones.

Utilice montajes de volumen en lugar de variables de entorno

Los valores de las variables de entorno pueden aparecer involuntariamente en los registros. Los secretos montados como volúmenes se instancian como volúmenes tmpfs (un sistema de archivos respaldado por la RAM) que se eliminan automáticamente del nodo cuando se elimina el pod.

Utilice un proveedor de secretos externo

Existen varias alternativas viables al uso de los secretos de Kubernetes, como AWS Secrets Manager y Hashicorp's Vault. Estos servicios ofrecen funciones como controles de acceso detallados, cifrado seguro y rotación automática de los secretos que no están disponibles en Kubernetes Secrets. Sealed Secrets de Bitnami es otro enfoque que utiliza el cifrado asimétrico para crear «secretos sellados». Se usa una clave pública para cifrar el secreto, mientras que la clave privada que se usa para descifrar el secreto se guarda dentro del clúster, lo que le permite almacenar secretos sellados de forma segura en sistemas de control de código fuente como Git. Para obtener más información, consulta Cómo gestionar el despliegue de secretos en Kubernetes mediante Sealed Secrets.

A medida que ha crecido el uso de almacenes de secretos externos, también lo ha hecho la necesidad de integrarlos con Kubernetes. The Secret Store CSI Driver es un proyecto comunitario que utiliza el modelo de controladores CSI para recuperar secretos de almacenes secretos externos. Actualmente, el controlador es compatible con AWS Secrets Manager, Azure, Vault y GCP. El proveedor de AWS es compatible con AWS Secrets Manager y AWS Parameter Store. También se puede configurar para rotar los secretos cuando caduquen y puede sincronizar los secretos de AWS Secrets Manager con los secretos de Kubernetes. La sincronización de secretos puede resultar útil cuando necesita hacer referencia a un secreto como variable de entorno en lugar de leerlo de un volumen.

> **Note**
>
> Cuando el controlador CSI del almacén de secretos tiene que buscar un secreto, asume la función IRSA asignada al módulo que hace referencia a un secreto. El código de esta operación se encuentra aquí.

Para obtener información adicional sobre el proveedor de secretos y configuración de AWS (ASCP), consulte los siguientes recursos:

- Cómo usar el proveedor de configuración de AWS Secrets con el controlador CSI de Kubernetes Secret Store
- Integración de los secretos de Secrets Manager con el controlador CSI de Kubernetes Secrets Store

external-secrets es otra forma de utilizar un almacén secreto externo con Kubernetes. Al igual que el controlador CSI, external-secrets funciona con una variedad de backends diferentes, incluido AWS Secrets Manager. La diferencia es que, en lugar de recuperar los secretos del almacén secreto externo, external-secrets copia los secretos de estos backends a Kubernetes como secretos. Esto te permite gestionar los secretos con tu tienda de secretos preferida e interactuar con ellos de una forma nativa de Kubernetes.

Herramientas y recursos

- Taller de inmersión en seguridad de Amazon EKS: cifrado de datos y gestión de secretos

Seguridad en tiempo de ejecución

La seguridad en tiempo de ejecución proporciona una protección activa para sus contenedores mientras están en ejecución. La idea es detectar y and/or evitar que se produzcan actividades maliciosas dentro del contenedor. Esto se puede lograr con una serie de mecanismos en el núcleo de Linux o extensiones del núcleo que estén integradas con Kubernetes, como las capacidades de Linux, la computación segura (seccomp) o. AppArmor SELinux También hay opciones como Amazon GuardDuty y herramientas de terceros que pueden ayudar a establecer líneas de base y detectar actividades anómalas con una configuración menos manual de los mecanismos del núcleo de Linux.

> **Important**
>
> Actualmente, Kubernetes no proporciona ningún mecanismo nativo para cargar sec.comp o perfiles en los nodos. AppArmor SELinux Deben cargarse manualmente o instalarse en los nodos cuando se inician. Esto se debe hacer antes de hacer referencia a ellos en tus pods, ya que el programador no sabe qué nodos tienen perfiles. Descubre a continuación cómo herramientas como Security Profiles Operator pueden ayudar a automatizar el aprovisionamiento de perfiles en los nodos.

Contextos de seguridad y controles de Kubernetes integrados

Muchos mecanismos de seguridad en tiempo de ejecución de Linux están estrechamente integrados con Kubernetes y se pueden configurar a través de los contextos de seguridad de Kubernetes. Una de esas opciones es la `privileged` bandera, que viene de forma `false` predeterminada y, si está habilitada, equivale básicamente a rootear en el host. Casi siempre es inapropiado habilitar el modo privilegiado en las cargas de trabajo de producción, pero hay muchos más controles que pueden proporcionar privilegios más detallados a los contenedores, según proceda.

Capacidades de Linux

Las capacidades de Linux permiten conceder determinadas capacidades a un pod o contenedor sin proporcionar todas las capacidades del usuario root. Algunos ejemplos son: `CAP_NET_ADMIN` que permite configurar interfaces de red o firewallsCAP_SYS_TIME, o que permite manipular el reloj del sistema.

Seccomp

Con la informática segura (seccomp), puede impedir que una aplicación contenerizada realice determinadas llamadas al sistema operativo anfitrión subyacente. Si bien el sistema operativo Linux tiene unos cientos de llamadas al sistema, la mayor parte de ellas no son necesarias para ejecutar contenedores. Al restringir las llamadas al sistema que puede realizar un contenedor, puede reducir de manera efectiva la superficie de ataque de su aplicación.

Seccomp intercepta las llamadas al sistema y solo permite el paso de las llamadas permitidas. Docker tiene un perfil seccomp predeterminado que es adecuado para la mayoría de las cargas de trabajo de uso general, y otros tiempos de ejecución de contenedores, como containerd, proporcionan valores predeterminados comparables. Puedes configurar tu contenedor o pod para

que utilice el perfil seccomp predeterminado del motor de ejecución del contenedor añadiendo lo siguiente a la sección de la especificación del pod: `securityContext`

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

A partir de la versión 1.22 (en alfa, estable a partir de la 1.27), lo anterior se `RuntimeDefault` puede usar para todos los pods de un nodo con un único indicador de kubelet. `--seccomp-default` Entonces, el perfil especificado en solo `securityContext` es necesario para otros perfiles.

También es posible crear tus propios perfiles para las cosas que requieren privilegios adicionales. Puede resultar muy tedioso hacerlo manualmente, pero hay herramientas como Inspektor Gadget (también se recomienda en la sección de seguridad de la red para generar políticas de red) y Security Profiles Operator que permiten utilizar herramientas como eBPF o registros para registrar los requisitos de privilegios básicos como perfiles seccomp. Además, Security Profiles Operator permite automatizar el despliegue de los perfiles registrados en los nodos para que los utilicen los pods y los contenedores.

AppArmor y SELinux

AppArmor y SELinux se conocen como sistemas de control de acceso obligatorio o MAC. Su concepto es similar al de los seccomp, pero con capacidades diferentes APIs , que permiten el control de acceso para, por ejemplo, rutas específicas de sistemas de archivos o puertos de red. El soporte para estas herramientas depende de la distribución de Linux, con Debian/Ubuntu soporte AppArmor y soporte para RHEL/CentOS/Bottlerocket/Amazon Linux 2023 SELinux. Consulte también la sección de seguridad de la infraestructura para obtener más información al respecto SELinux.

Ambos AppArmor SELinux están integrados en Kubernetes, pero a partir de la versión 1.28 de Kubernetes los AppArmor perfiles deben especificarse mediante anotaciones, mientras que las SELinux etiquetas se pueden establecer directamente en el campo Opciones del SELinux contexto de seguridad.

Al igual que con los perfiles seccomp, el operador de perfiles de seguridad mencionado anteriormente puede ayudar a implementar los perfiles en los nodos del clúster. (En el futuro, el proyecto también tiene como objetivo generar perfiles para AppArmor y SELinux como lo hace para seccomp).

Recomendaciones

Utilice Amazon GuardDuty para supervisar en tiempo de ejecución y detectar amenazas en sus entornos de EKS

Si actualmente no dispone de una solución para supervisar de forma continua los tiempos de ejecución de EKS, analizar los registros de auditoría de EKS y escanear en busca de malware y otras actividades sospechosas, Amazon recomienda encarecidamente el uso de Amazon GuardDuty a los clientes que deseen proteger sus entornos de AWS de forma sencilla, rápida, segura, escalable y rentable con un solo clic. Amazon GuardDuty es un servicio de supervisión de seguridad que analiza y procesa las fuentes de datos fundamentales, como los eventos de CloudTrail administración de AWS, los registros de eventos de AWS CloudTrail , los registros de flujo de VPC (de instancias de EC2 Amazon), los registros de auditoría de Kubernetes y los registros de DNS. También incluye la supervisión del tiempo de ejecución de EKS. Utiliza fuentes de inteligencia sobre amenazas que se actualizan continuamente, como listas de direcciones IP y dominios maliciosos, y aprendizaje automático para identificar actividades inesperadas, potencialmente no autorizadas y maliciosas en su entorno de AWS. Esto puede incluir problemas como la escalada de privilegios, el uso de credenciales expuestas o la comunicación con direcciones IP o dominios maliciosos, la presencia de malware en las EC2 instancias de Amazon y las cargas de trabajo de contenedores de EKS, o el descubrimiento de actividad sospechosa de la API. GuardDuty le informa del estado de su entorno de AWS mediante resultados de seguridad que puede ver en la GuardDuty consola o a través de Amazon EventBridge. GuardDuty también le ayuda a exportar sus hallazgos a un bucket de Amazon Simple Storage Service (S3) y a integrarlos con otros servicios, como AWS Security Hub y Detective.

Vea esta charla técnica en línea de AWS «Detección mejorada de amenazas para Amazon EKS con Amazon GuardDuty : charlas técnicas en línea de AWS» para ver cómo habilitar estas funciones de seguridad adicionales de EKS step-by-step en cuestión de minutos.

Opcionalmente: utilice una solución de terceros para la supervisión del tiempo de ejecución

Crear y administrar los perfiles seccomp y Apparmor puede resultar difícil si no está familiarizado con la seguridad de Linux. Si no tiene tiempo para adquirir los conocimientos necesarios, considere la posibilidad de utilizar una solución comercial de terceros. Muchos de ellos han ido más allá de los perfiles estáticos, como Apparmor y seccomp, y han empezado a utilizar el aprendizaje automático para bloquear o alertar sobre actividades sospechosas. A continuación, en la sección

de herramientas, encontrará algunas de estas soluciones. Puede encontrar opciones adicionales en AWS Marketplace for Containers.

Tenga en cuenta las capacidades de add/dropping Linux antes de redactar las políticas de seccomp

Las capacidades implican varias comprobaciones de las funciones del núcleo a las que se puede acceder mediante llamadas al sistema. Si la comprobación falla, la llamada al sistema normalmente devuelve un error. La comprobación se puede realizar justo al principio de una llamada al sistema específica o en una zona más profunda del núcleo, en áreas a las que se puede acceder mediante varias llamadas al sistema diferentes (por ejemplo, escribir en un archivo privilegiado específico). Seccomp, por otro lado, es un filtro de llamadas al sistema que se aplica a todas las llamadas al sistema antes de que se ejecuten. Un proceso puede configurar un filtro que le permita revocar su derecho a ejecutar determinadas llamadas al sistema o argumentos específicos para determinadas llamadas al sistema.

Antes de usar seccomp, considere si las capacidades de adding/removing Linux le proporcionan el control que necesita. Consulte Configurar las capacidades de los contenedores para obtener más información.

Comprueba si puedes lograr tus objetivos mediante las políticas de seguridad de Pod (PSPs)

Las políticas de seguridad de Pod ofrecen muchas formas diferentes de mejorar tu postura de seguridad sin introducir una complejidad excesiva. Explore las opciones disponibles PSPs antes de aventurarse a crear perfiles seccomp y Apparmor.

> **Warning**
>
> A partir de la versión 1.25 de Kubernetes, se PSPs han eliminado y sustituido por el controlador de admisión Pod Security. Las alternativas de terceros que existen incluyen OPA/Gatekeeper Kyverno. En el repositorio de la biblioteca de Gatekeeper se PSPs puede encontrar una colección de restricciones y plantillas de restricciones para implementar políticas que se encuentran comúnmente en. GitHub Además, en la biblioteca de políticas de Kyverno se PSPs pueden encontrar muchos sustitutos, incluida la colección completa de normas de seguridad de Pod.

Herramientas y recursos

- 7 cosas que debes saber antes de empezar
- AppArmorCargador
- Configuración de nodos con perfiles
- Security Profiles Operator es una mejora de Kubernetes que tiene como objetivo facilitar a los usuarios su uso SELinux, seccomp y en clústeres de Kubernetes. AppArmor Ofrece funciones tanto para generar perfiles a partir de cargas de trabajo en ejecución como para cargar perfiles en nodos de Kubernetes para usarlos en pods.
- Inspektor Gadget permite inspeccionar, rastrear y perfilar muchos aspectos del comportamiento en tiempo de ejecución en Kubernetes, incluida la asistencia en la generación de perfiles seccomp.
- Aqua
- Qualys
- Stackrox
- Sysdig Secure
- Prisma
- NeuVector de SUSE, plataforma de seguridad de contenedores de código abierto y de confianza cero, proporciona reglas de perfil de procesos y reglas de acceso a archivos.

Protección de la infraestructura (hosts)

Si bien es importante proteger las imágenes de los contenedores, es igual de importante proteger la infraestructura en la que se ejecutan. En esta sección, se analizan diferentes formas de mitigar los riesgos derivados de los ataques lanzados directamente contra el servidor. Estas directrices deben utilizarse junto con las descritas en la sección Seguridad en tiempo de ejecución.

Recomendaciones

Utilice un sistema operativo optimizado para ejecutar contenedores

Considere la posibilidad de usar Flatcar Linux, Project Atomic, RancherOS y Bottlerocket, un sistema operativo especial de AWS diseñado para ejecutar contenedores de Linux. Incluye una superficie de ataque reducida, una imagen de disco que se verifica al arrancar y límites de permisos obligatorios para su uso. SELinux

Como alternativa, utilice la AMI optimizada para EKS para sus nodos de trabajo de Kubernetes. La AMI optimizada para EKS se publica periódicamente y contiene un conjunto mínimo de paquetes de sistema operativo y binarios necesarios para ejecutar sus cargas de trabajo en contenedores.

Consulte la especificación de compilación de Amazon EKS AMI RHEL para ver un ejemplo de script de configuración que se puede utilizar para crear una AMI Amazon EKS personalizada que se ejecute en Red Hat Enterprise Linux mediante Hashicorp Packer. Este script se puede aprovechar aún más para crear EKS personalizados y compatibles con STIG. AMIs

Mantenga actualizado el sistema operativo de su nodo de trabajo

Independientemente de si utiliza un sistema operativo host optimizado para contenedores, como Bottlerocket, o un Amazon Machine Image más grande, pero aún minimalista, como el optimizado para EKS AMIs, se recomienda mantener estas imágenes del sistema operativo anfitrión actualizadas con los últimos parches de seguridad.

Si está optimizado para EKS AMIs, consulte periódicamente el registro de cambios o el canal de notas de la versión y automatice la distribución de las imágenes actualizadas de los nodos de trabajo en su clúster.

Trate su infraestructura como algo inmutable y automatice la sustitución de sus nodos de trabajo

En lugar de realizar actualizaciones in situ, sustituya a sus trabajadores cuando haya un nuevo parche o actualización disponible. Esto puede abordarse de dos maneras. Puede añadir instancias a un grupo de escalado automático existente mediante la AMI más reciente, ya que acordona y vacía los nodos de forma secuencial hasta que todos los nodos del grupo se sustituyan por la AMI más reciente. Como alternativa, puede añadir instancias a un nuevo grupo de nodos y, al mismo tiempo, acordonar y drenar los nodos del grupo de nodos anterior de forma secuencial hasta que se hayan reemplazado todos los nodos. Los grupos de nodos gestionados por EKS utilizan el primer enfoque y mostrarán un mensaje en la consola para actualizar a sus trabajadores cuando haya una nueva AMI disponible. `eksctl`también tiene un mecanismo para crear grupos de nodos con la AMI más reciente y para acordonar y drenar correctamente los pods de los grupos de nodos antes de que se terminen las instancias. Si decide utilizar un método diferente para sustituir los nodos de trabajo, le recomendamos encarecidamente que automatice el proceso para minimizar la supervisión humana, ya que es probable que tenga que sustituir a los trabajadores periódicamente a medida que salgan nuevos nodos y cuando updates/patches se actualice el plano de control.

Con EKS Fargate, AWS actualizará automáticamente la infraestructura subyacente a medida que haya actualizaciones disponibles. A menudo, esto se puede hacer sin problemas, pero puede haber ocasiones en las que una actualización provoque que el pod se reprograme. Por lo tanto, le recomendamos que cree implementaciones con varias réplicas cuando ejecute la aplicación como un pod de Fargate.

Ejecute kube-bench periódicamente para verificar el cumplimiento de los puntos de referencia del CIS para Kubernetes

kube-bench es un proyecto de código abierto de Aqua que evalúa tu clúster comparándolo con los puntos de referencia del CIS para Kubernetes. El punto de referencia describe las mejores prácticas para proteger los clústeres de Kubernetes no gestionados. El índice de referencia de Kubernetes del CIS abarca el plano de control y el plano de datos. Dado que Amazon EKS proporciona un plano de control totalmente gestionado, no se aplican todas las recomendaciones del CIS Kubernetes Benchmark. Para garantizar que este ámbito refleje la forma en que se implementa Amazon EKS, AWS creó el Amazon EKS Benchmark de la CEI. El índice de referencia EKS proviene del Kubernetes Benchmark de CIS, con aportaciones adicionales de la comunidad y consideraciones de configuración específicas para los clústeres de EKS.

Cuando ejecutes kube-bench en un clúster de EKS, sigue estas instrucciones de Aqua Security. Para obtener más información, consulte Presentamos el Amazon EKS Benchmark de la CEI.

Minimice el acceso a los nodos de trabajo

En lugar de habilitar el acceso SSH, utilice el administrador de sesiones SSM cuando necesite acceder de forma remota a un host. A diferencia de las claves SSH, que se pueden perder, copiar o compartir, el administrador de sesiones le permite controlar el acceso a las EC2 instancias mediante IAM. Además, proporciona una pista de auditoría y un registro de los comandos que se ejecutaron en la instancia.

A partir del 19 de agosto de 2020, los grupos de nodos gestionados admiten plantillas personalizadas AMIs y de EC2 lanzamiento. Esto le permite incrustar el agente SSM en la AMI o instalarlo mientras se inicia el nodo de trabajo. Si prefiere no modificar la AMI optimizada ni la plantilla de lanzamiento del ASG, puede instalar el agente SSM con una, DaemonSet como en este ejemplo.

Política de IAM mínima para el acceso SSH basado en SSM

La política administrada por `AmazonSSMManagedInstanceCore` AWS contiene una serie de permisos que no son necesarios para SSM Session Manager/SSM RunCommand si solo

quiere evitar el acceso a SSH. En concreto, son motivo de preocupación los * permisos `ssm:GetParameter(s)` que permitirían al rol acceder a todos los parámetros del almacén de parámetros (incluso SecureStrings con la clave de KMS administrada por AWS configurada).

La siguiente política de IAM contiene el conjunto mínimo de permisos para permitir el acceso a los nodos a través de SSM Systems Manager.

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

Una vez implementada esta política y el complemento Session Manager instalado, puede ejecutar

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

para acceder al nodo.

> **Note**
>
> También puede considerar la posibilidad de añadir permisos para habilitar el registro del administrador de sesiones.

Despliegue a los trabajadores en subredes privadas

Al desplegar a los trabajadores en subredes privadas, se minimiza su exposición a Internet, donde suelen originarse los ataques. A partir del 22 de abril de 2020, la subred en la que se desplieguen controlará la asignación de direcciones IP públicas a los nodos de un grupo de nodos gestionado. Antes de esto, a los nodos de un grupo de nodos gestionado se les asignaba automáticamente una IP pública. Si decide implementar sus nodos de trabajo en subredes públicas, implemente reglas restrictivas para los grupos de seguridad de AWS para limitar su exposición.

Ejecute Amazon Inspector para evaluar la exposición, las vulnerabilidades y las desviaciones de los hosts con respecto a las mejores prácticas

Puede utilizar Amazon Inspector para comprobar si hay accesos no intencionados a la red a sus nodos y si hay vulnerabilidades en las EC2 instancias de Amazon subyacentes.

Amazon Inspector puede proporcionar datos sobre vulnerabilidades y exposiciones comunes (CVE) para sus EC2 instancias de Amazon solo si el agente Amazon EC2 Systems Manager (SSM) está instalado y activado. Este agente viene preinstalado en varias Amazon Machine Images (AMIs), incluido Amazon Linux AMIs optimizado para EKS. Independientemente del estado del agente de SSM, todas tus EC2 instancias de Amazon se escanean para detectar problemas de accesibilidad de la red. Para obtener más información sobre la configuración de escaneos para Amazon EC2, consulta Escanear EC2 instancias de Amazon.

> **Important**
>
> El Inspector no se puede ejecutar en la infraestructura utilizada para ejecutar los pods de Fargate.

Alternativas

¡Corre SELinux

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

SELinux proporciona una capa de seguridad adicional para mantener los contenedores aislados unos de otros y del host. SELinux permite a los administradores aplicar controles de acceso (MAC) obligatorios para cada usuario, aplicación, proceso y archivo. Considérelo como un mecanismo de salvaguardia que restringe las operaciones que se pueden realizar con recursos específicos en función de un conjunto de etiquetas. En EKS, se SELinux puede usar para evitar que los contenedores accedan a los recursos de los demás.

Las SELinux políticas de contenedores se definen en el paquete container-selinux. Docker CE requiere este paquete (junto con sus dependencias) para que los procesos y archivos creados por Docker (u otros entornos de ejecución de contenedores) se ejecuten con un acceso limitado al sistema. Los contenedores utilizan la `container_t` etiqueta a la que sirve de alias. `svirt_lxc_net_t` Estas políticas impiden de forma efectiva que los contenedores accedan a determinadas funciones del servidor.

Cuando se configura SELinux para Docker, Docker etiqueta automáticamente las cargas de trabajo `container_t` como un tipo y asigna a cada contenedor un nivel de MCS único. Esto aislará los contenedores unos de otros. Si necesita restricciones más flexibles, puede crear su propio perfil en el SElinux que se concedan permisos a un contenedor para acceder a áreas específicas del sistema de archivos. Esto es similar al hecho PSPs de que puedes crear diferentes perfiles para distintos contenedores o pods. Por ejemplo, puede tener un perfil para cargas de trabajo generales con un conjunto de controles restrictivos y otro para las cosas que requieren un acceso privilegiado.

SELinux for Containers tiene un conjunto de opciones que se pueden configurar para modificar las restricciones predeterminadas. Los siguientes valores SELinux booleanos se pueden activar o desactivar en función de sus necesidades:

Booleano	Predeterminado	Description (Descripción)
`container_connect_any`	`off`	Permita que los contenedores accedan a los puertos

Booleano	Predeterminado	Description (Descripción)
		privilegiados del host. Por ejemplo, si tiene un contenedor que necesita asignar los puertos 443 u 80 del host.
`container_manage_cgroup`	`off`	Permita que los contenedores administren la configuración del grupo. Por ejemplo, un contenedor que ejecute systemd necesitará que esté habilitado.
`container_use_cephfs`	`off`	Permita que los contenedores usen un sistema de archivos ceph.

De forma predeterminada, los contenedores pueden guardar `/usr` y read/execute leer la mayor parte del `/etc` contenido. Los archivos que están debajo `/var/lib/docker` y `/var/lib/containers` tienen la etiqueta`container_var_lib_t`. Para ver una lista completa de los valores predeterminados, las etiquetas consultan el archivo 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
```

Los archivos etiquetados con `container_file_t` son los únicos archivos en los que los contenedores pueden escribir. Si desea que se pueda escribir en un montaje de volumen, tendrá que especificar `:z` o al `:Z` final.

- **:z** volverá a etiquetar los archivos para que el contenedor pueda leer/escribir
- **:Z** volverá a etiquetar los archivos para que solo el contenedor pueda leer/escribir

```
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
```

En Kubernetes, el reetiquetado es ligeramente diferente. En lugar de hacer que Docker vuelva a etiquetar automáticamente los archivos, puede especificar una etiqueta MCS personalizada para ejecutar el pod. Los volúmenes que admiten el reetiquetado se volverán a etiquetar automáticamente para que sean accesibles. Los módulos que tengan la etiqueta MCS correspondiente podrán acceder al volumen. Si necesitas un aislamiento estricto, establece una etiqueta MCS diferente para cada cápsula.

```
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
```

En este ejemplo, `s0:c144:c154` corresponde a una etiqueta MCS asignada a un archivo al que el contenedor puede acceder.

En EKS, puede crear políticas que permitan la ejecución de contenedores privilegiados, como FluentD, y crear SELinux una política que le permita leer desde /var/log en el host sin necesidad de volver a etiquetar el directorio del host. Los pods con la misma etiqueta podrán acceder a los mismos volúmenes del host.

Hemos implementado un ejemplo AMIs para Amazon EKS que se ha SELinux configurado en Centos 7 y RHEL 7. AMIs Se desarrollaron para mostrar ejemplos de implementaciones que cumplen con los requisitos de los clientes altamente regulados.

> **Warning**
>
> SELinux ignorará los contenedores cuyo tipo no esté confinado.

Herramientas y recursos

- SELinuxPolíticas de seguridad de envío y RBAC de Kubernetes para aplicaciones locales
- Fortalecimiento iterativo de Kubernetes
- Audit 2 Allow
- SEAlert
- Generar SELinux políticas para contenedores con Udica describe una herramienta que analiza las capacidades, los puertos y los puntos de montaje de Linux en los archivos de especificaciones de los contenedores, y genera un conjunto de SELinux reglas que permiten que el contenedor funcione correctamente
- Cuadernos de estrategias de AMI Hardening para reforzar el sistema operativo a fin de cumplir con los diferentes requisitos reglamentarios
- Keiko Upgrade Manager, un proyecto de código abierto de Intuit que organiza la rotación de los nodos trabajadores.
- Sysdig Secure
- eksctl

Conformidad

El cumplimiento es una responsabilidad compartida entre AWS y los consumidores de sus servicios. En términos generales, AWS es responsable de la «seguridad de la nube», mientras que sus usuarios son responsables de la «seguridad en la nube». La línea que define los aspectos de los que son responsables AWS y sus usuarios variará en función del servicio. Por ejemplo, con Fargate, AWS es responsable de administrar la seguridad física de sus centros de datos, el hardware, la infraestructura virtual (Amazon EC2) y el tiempo de ejecución de los contenedores (Docker). Los

usuarios de Fargate son responsables de proteger la imagen del contenedor y su aplicación. Saber quién es responsable de qué es una consideración importante a la hora de ejecutar cargas de trabajo que deben cumplir con las normas de conformidad.

En la siguiente tabla se muestran los programas de conformidad a los que se ajustan los distintos servicios de contenedores.

Programa de cumplimiento	Amazon ECS Orchestrator	Amazon EKS Orchestrator	Fargate ECS	Amazon ECR
PCI DSS Nivel 1	1	1	1	1
Cumplimiento de requisitos de HIPAA	1	1	1	1
SOC I	1	1	1	1
SOC II	1	1	1	1
SOC III	1	1	1	1
ISO 27001:2013	1	1	1	1
ISO 9001:2015	1	1	1	1
ISO 27017:2015	1	1	1	1
ISO 27018:2019	1	1	1	1
IRAP	1	1	1	1
FedRAMP Moderate (East/West)	1	1	0	1
FedRamp High () GovCloud	1	1	0	1
¿CC SRG	1	Reseña de DISA () IL5	0	1
HIPAA BAA	1	1	1	1

Programa de cumplimiento	Amazon ECS Orchestrator	Amazon EKS Orchestrator	Fargate ECS	Amazon ECR
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

El estado de cumplimiento cambia con el tiempo. Para ver el estado más reciente, consulte siempre https://aws.amazon.com/compliance/ services-in-scope /.

Para obtener más información sobre los modelos de acreditación en la nube y las prácticas recomendadas, consulte el documento técnico de AWS, Modelos de acreditación para la adopción segura de la nube

Girando a la izquierda

El concepto de girar a la izquierda implica detectar las infracciones y los errores de las políticas en una fase más temprana del ciclo de vida del desarrollo del software. Desde el punto de vista de la seguridad, esto puede resultar muy beneficioso. Un desarrollador, por ejemplo, puede solucionar los problemas de configuración antes de implementar la aplicación en el clúster. Detectar errores como este antes ayudará a evitar que se implementen configuraciones que infrinjan sus políticas.

La política como código

La política puede considerarse como un conjunto de reglas que rigen los comportamientos, es decir, los comportamientos permitidos o prohibidos. Por ejemplo, puedes tener una política que indique que todos los Dockerfiles deben incluir una directiva USER que haga que el contenedor se ejecute como un usuario no root. Como documento, una política como esta puede ser difícil de descubrir y aplicar. También puede quedar desactualizada a medida que cambien sus requisitos. Con las soluciones Policy as Code (PaC), puede automatizar los controles de seguridad, cumplimiento y privacidad que detectan, previenen, reducen y contrarrestan las amenazas conocidas y persistentes.

Además, le proporcionan un mecanismo para codificar sus políticas y gestionarlas del mismo modo que lo hace con otros artefactos de código. La ventaja de este enfoque es que puede reutilizar sus GitOps estrategias para gestionar DevOps y aplicar políticas de forma coherente en todas las flotas de clústeres de Kubernetes. Consulte Pod Security para obtener información sobre las opciones de PaC y el futuro de PSPs.

Utilice policy-as-code herramientas en proceso para detectar infracciones antes del despliegue

- OPA es un motor de políticas de código abierto que forma parte de la CNCF. Se utiliza para tomar decisiones políticas y se puede gestionar de diferentes maneras, por ejemplo, como una biblioteca de idiomas o un servicio. Las políticas de la OPA están redactadas en un lenguaje específico de dominio (DSL) denominado Rego. Si bien suele ejecutarse como parte de un controlador de admisión dinámico de Kubernetes como el proyecto Gatekeeper, la OPA también se puede incorporar a su proceso. CI/CD Esto permite a los desarrolladores recibir comentarios sobre su configuración al principio del ciclo de lanzamiento, lo que puede ayudarlos a resolver los problemas antes de que pasen a la fase de producción. En el GitHub repositorio de este proyecto se encuentra un conjunto de políticas de OPA comunes.
- Contest se basa en OPA y proporciona una experiencia centrada en los desarrolladores para probar la configuración de Kubernetes.
- Kyverno es un motor de políticas diseñado para Kubernetes. Con Kyverno, las políticas se gestionan como recursos de Kubernetes y no se necesita ningún idioma nuevo para redactarlas. Esto permite utilizar herramientas conocidas como kubectl, git y kustomize para gestionar las políticas. Las políticas de Kyverno pueden validar, mutar y generar recursos de Kubernetes, además de garantizar la seguridad de la cadena de suministro de imágenes de OCI. La CLI de Kyverno se puede utilizar para probar políticas y validar recursos como parte de una CI/CD canalización. Todas las políticas de la comunidad de Kyverno se encuentran en el sitio web de Kyverno y, para ver ejemplos de cómo utilizar la CLI de Kyverno para escribir pruebas en canalizaciones, consulte el repositorio de políticas.

Herramientas y recursos

- Taller de inmersión en seguridad de Amazon EKS: cumplimiento normativo
- kube-bench
- docker-bench-security
- Inspector de AWS

- Revisión de seguridad de Kubernetes Una evaluación de seguridad de Kubernetes 1.13.4 (2019) realizada por una tercera parte
- NeuVector de SUSE, una plataforma de seguridad de contenedores de código abierto y de confianza cero, que proporciona informes de conformidad y comprobaciones de conformidad personalizadas

Respuesta a incidentes y análisis forense

Su capacidad para reaccionar rápidamente ante un incidente puede ayudar a minimizar los daños causados por una infracción. El primer paso de un buen plan de respuesta a incidentes es disponer de un sistema de alerta fiable que pueda advertirle de cualquier comportamiento sospechoso. Cuando se produce un incidente, hay que decidir rápidamente si destruir y sustituir el contenedor afectado o aislarlo e inspeccionarlo. Si decide aislar el contenedor como parte de una investigación forense y un análisis de la causa raíz, debe seguir el siguiente conjunto de actividades:

Ejemplo de plan de respuesta a incidentes

Identifique el pod y el nodo trabajador infractores

Lo primero que debes hacer es aislar el daño. Comience por identificar dónde se produjo la infracción y aísle ese pod y su nodo del resto de la infraestructura.

Identifique los pods y los nodos de trabajo infractores utilizando el nombre de la carga de trabajo

Si conoces el nombre y el espacio de nombres del pod infractor, puedes identificar el nodo de trabajo que ejecuta el pod de la siguiente manera:

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

Si un recurso de carga de trabajo, como una implementación, se ha visto comprometido, es probable que todos los pods que forman parte del recurso de carga de trabajo estén comprometidos. Usa el siguiente comando para enumerar todos los pods del recurso de carga de trabajo y los nodos en los que se ejecutan:

```
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)"'
```

El comando anterior es para despliegues. Puede ejecutar el mismo comando para otros recursos de carga de trabajo, como replicasets, statefulsets, etc.

Identifique los pods y los nodos de trabajo infractores mediante el nombre de la cuenta de servicio

En algunos casos, puedes identificar que una cuenta de servicio está comprometida. Es probable que los pods que utilizan la cuenta de servicio identificada estén comprometidos. Puedes identificar todos los pods mediante la cuenta de servicio y los nodos en los que se ejecutan con el siguiente comando:

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

Identifique los nodos de trabajo y los pods con imágenes vulnerables o comprometidas

En algunos casos, es posible que descubras que la imagen de un contenedor que se está utilizando en los pods de tu clúster es maliciosa o está comprometida. La imagen de un contenedor es maliciosa o está comprometida si se descubre que contiene malware, si se sabe que es una imagen defectuosa o si tiene un CVE que ha sido explotado. Deberías considerar que la imagen de todos los módulos que utilizan el contenedor está comprometida. Puede identificar los pods mediante la imagen y los nodos en los que se ejecutan con el siguiente comando:

```
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)"'
```

Aísle el pod creando una política de red que deniegue todo el tráfico de entrada y salida al pod

Una regla de denegación total de tráfico puede ayudar a detener un ataque que ya está en marcha, ya que corta todas las conexiones al pod. La siguiente política de red se aplicará a un pod con la etiquetaapp=web.

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

> **Important**
>
> Una política de red puede resultar ineficaz si un atacante ha accedido al host subyacente. Si sospecha que eso ha ocurrido, puede usar los grupos de seguridad de AWS para aislar un host comprometido de otros hosts. Al cambiar el grupo de seguridad de un anfitrión, tenga en cuenta que esto afectará a todos los contenedores que se ejecuten en ese host.

Revoca las credenciales de seguridad temporales asignadas al pod o al nodo de trabajo si es necesario

Si al nodo trabajador se le ha asignado una función de IAM que permita a los pods acceder a otros recursos de AWS, elimine esas funciones de la instancia para evitar que el ataque cause más daños. Del mismo modo, si al pod se le ha asignado un rol de IAM, evalúe si puede eliminar de forma segura las políticas de IAM del rol sin que ello afecte a otras cargas de trabajo.

Acordona el nodo trabajador

Al acordonar el nodo trabajador afectado, se informa al planificador para evitar programar módulos en el nodo afectado. Esto le permitirá extraer el nodo para someterlo a un estudio forense sin interrumpir otras cargas de trabajo.

> **Note**
>
> Esta guía no se aplica a Fargate, donde cada pod de Fargate se ejecuta en su propio entorno aislado. En lugar de acordonar, aísle los pods Fargate afectados aplicando una política de red que deniegue todo el tráfico de entrada y salida.

Habilite la protección por despido en el nodo de trabajo afectado

Un atacante puede intentar borrar sus fechorías cerrando un nodo afectado. Habilitar la protección de terminación puede evitar que esto suceda. La protección de escalamiento interno de instancias protegerá al nodo de un evento de escalamiento interno.

> **Warning**
>
> No puede habilitar la protección de terminación en una instancia puntual.

Etiquete el delito Pod/Node con una etiqueta que indique que forma parte de una investigación en curso

Esto servirá como advertencia a los administradores de clústeres para que no manipulen a los afectados Pods/Nodes hasta que se complete la investigación.

Capture los artefactos volátiles en el nodo de trabajo

- Capture la memoria del sistema operativo. Esto capturará el daemon de Docker (u otro entorno de ejecución de contenedor) y sus subprocesos por contenedor. Esto se puede lograr utilizando herramientas como LiME y Volatility, o mediante herramientas de nivel superior, como Automated Forensics Orchestrator para EC2 Amazon, que se basan en ellas.
- Realice un volcado en árbol de netstat de los procesos en ejecución y de los puertos abiertos. Esto capturará el daemon docker y su subproceso por contenedor.
- Ejecute comandos para guardar el estado a nivel del contenedor antes de que se modifique la evidencia. Puede utilizar las funciones del motor de ejecución del contenedor para capturar información sobre los contenedores que se están ejecutando actualmente. Por ejemplo, con Containerd, puede hacer lo siguiente:
  - `crictl ps`para los procesos en ejecución.

- `crictl logs CONTAINER`para registros guardados a nivel de demonio.

  Lo mismo podría lograrse con containerd utilizando la CLI nerdctl, en lugar `docker` de (por ejemplo). `nerdctl inspect` Hay algunos comandos adicionales disponibles según el tiempo de ejecución del contenedor. Por ejemplo, Docker debe `docker diff` ver los cambios en el sistema de archivos del contenedor o `docker checkpoint` guardar todo el estado del contenedor, incluida la memoria volátil (RAM). Consulta esta entrada del blog de Kubernetes para ver información sobre capacidades similares con tiempos de ejecución containerd o CRI-O.
- Detenga el contenedor para realizar una captura forense.
- Realice una instantánea de los volúmenes de EBS de la instancia.

Vuelva a implementar el pod o el recurso de carga de trabajo comprometido

Una vez que haya recopilado los datos para el análisis forense, puede volver a implementar el pod o el recurso de carga de trabajo comprometido.

Primero, solucione la vulnerabilidad que estaba comprometida e inicie nuevos módulos de reemplazo. A continuación, elimine los pods vulnerables.

Si los pods vulnerables están gestionados por un recurso de carga de trabajo de Kubernetes de nivel superior (por ejemplo, una implementación o DaemonSet), al eliminarlos se programarán otros nuevos. Por lo tanto, los pods vulnerables se volverán a lanzar. En ese caso, debería implementar un nuevo recurso de carga de trabajo de reemplazo después de corregir la vulnerabilidad. A continuación, debe eliminar la carga de trabajo vulnerable.

Recomendaciones

Consulte el documento técnico sobre la respuesta a incidentes de seguridad de AWS

Si bien en esta sección se ofrece una breve descripción general junto con algunas recomendaciones para gestionar las presuntas infracciones de seguridad, el tema se trata de forma exhaustiva en el documento técnico AWS Security Incident Response.

Días de práctica de juegos de seguridad

Divida a sus profesionales de la seguridad en 2 equipos: rojo y azul. El equipo rojo se centrará en investigar diferentes sistemas en busca de vulnerabilidades, mientras que el equipo azul se encargará de defenderse de ellas. Si no cuenta con suficientes profesionales de la seguridad para

crear equipos independientes, considere la posibilidad de contratar a una entidad externa que tenga conocimiento de los exploits de Kubernetes.

Kubesploit es un marco de pruebas de penetración CyberArk que puedes utilizar para organizar jornadas de juego. A diferencia de otras herramientas que escanean tu clúster en busca de vulnerabilidades, kubesploit simula un ataque en el mundo real. Esto le da a tu equipo azul la oportunidad de practicar su respuesta a un ataque y evaluar su eficacia.

Realiza pruebas de penetración en tu clúster

Atacar periódicamente tu propio clúster puede ayudarte a descubrir vulnerabilidades y errores de configuración. Antes de empezar, sigue las pautas de las pruebas de penetración antes de realizar una prueba en tu clúster.

Herramientas y recursos

- kube-hunter, una herramienta de pruebas de penetración para Kubernetes.
- Gremlin, un conjunto de herramientas de ingeniería del caos que puede utilizar para simular ataques contra sus aplicaciones e infraestructuras.
- Atacar y defender las instalaciones de Kubernetes
- kubesploit
- NeuVector de SUSE, plataforma de seguridad de contenedores de código abierto y de confianza cero, proporciona informes de vulnerabilidades y riesgos, así como notificaciones de eventos de seguridad
- Amenazas persistentes avanzadas
- Ataque y defensa prácticos de Kubernetes
- Comprometiendo el clúster de Kubernetes al explotar los permisos del RBAC

Seguridad de imagen

Debe considerar la imagen del contenedor como su primera línea de defensa contra un ataque. Una imagen insegura y mal construida puede permitir que un atacante escape de los límites del contenedor y acceder al host. Una vez en el host, un atacante puede acceder a información confidencial o moverse lateralmente dentro del clúster o con su cuenta de AWS. Las siguientes prácticas recomendadas ayudarán a mitigar el riesgo de que esto ocurra.

Recomendaciones

Crea imágenes mínimas

Comience por eliminar todos los archivos binarios extraños de la imagen del contenedor. Si utilizas una imagen desconocida de Dockerhub, inspecciona la imagen con una aplicación como Dive, que puede mostrarte el contenido de cada una de las capas del contenedor. Elimine todos los archivos binarios con los bits SETUID y SETGID, ya que pueden usarse para aumentar los privilegios, y considere la posibilidad de eliminar todos los comandos y utilidades, como nc y curl, que puedan usarse con fines perversos. Puede encontrar los archivos con los bits SETUID y SETGID con el siguiente comando:

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

Para eliminar los permisos especiales de estos archivos, añade la siguiente directiva a la imagen del contenedor:

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

Coloquialmente, esto se conoce como quitar los colmillos de la imagen.

Usa compilaciones de varias etapas

El uso de compilaciones de varias etapas es una forma de crear imágenes mínimas. A menudo, las compilaciones en varias etapas se utilizan para automatizar partes del ciclo de integración continua. Por ejemplo, las compilaciones en varias etapas se pueden usar para borrar el código fuente o realizar un análisis de código estático. Esto brinda a los desarrolladores la oportunidad de obtener comentarios casi inmediatos en lugar de esperar a que se ejecute una canalización. Las compilaciones en varias etapas son atractivas desde el punto de vista de la seguridad porque permiten minimizar el tamaño de la imagen final que se inserta en el registro del contenedor. Las imágenes de contenedores sin herramientas de compilación ni otros binarios desconocidos mejoran la seguridad al reducir la superficie expuesta a ataques de la imagen. Para obtener información adicional sobre las compilaciones de varias etapas, consulta la documentación de compilaciones de varias etapas de Docker.

Cree una lista de materiales de software (SBOMs) para la imagen de su contenedor

Una «lista de materiales de software» (SBOM) es un inventario anidado de los artefactos de software que componen la imagen del contenedor. La SBOM es un elemento fundamental de la seguridad

del software y de la gestión del riesgo de la cadena de suministro de software. Generar y almacenar los SBOMS en un repositorio central y analizarlos en busca SBOMs de vulnerabilidades ayuda a resolver los siguientes problemas:

- Visibilidad: comprenda qué componentes componen la imagen de su contenedor. El almacenamiento en un repositorio central permite SBOMs auditarlo y escanearlo en cualquier momento, incluso después del despliegue, para detectar nuevas vulnerabilidades y responder a ellas, como las de día cero.
- Verificación de procedencia: garantía de que las suposiciones existentes sobre dónde y cómo se origina un artefacto son ciertas y de que el artefacto o los metadatos que lo acompañan no han sido manipulados durante los procesos de construcción o entrega.
- Fiabilidad: garantía de que se puede confiar en que un artefacto determinado y su contenido harán lo que se supone que hacen, es decir, que son adecuados para un propósito. Esto implica juzgar si el código es seguro de ejecutar y tomar decisiones informadas sobre los riesgos asociados a la ejecución del código. La confiabilidad se garantiza mediante la creación de un informe de ejecución de la canalización certificado junto con un informe de escaneo certificado de la SBOM y del CVE acreditado para garantizar a los consumidores de la imagen que, de hecho, la imagen se creó por medios seguros (canalización) con componentes seguros.
- Verificación de la confianza en la dependencia: comprobación recursiva del árbol de dependencias de un artefacto para comprobar la fiabilidad y la procedencia de los artefactos que utiliza. Drift in SBOMs puede ayudar a detectar actividades maliciosas, incluidas las dependencias no autorizadas y no confiables y los intentos de infiltración.

Se pueden utilizar las siguientes herramientas para generar la SBOM:

- Amazon Inspector se puede utilizar para crear y exportar SBOMs.
- Syft de Anchore también se puede utilizar para la generación de SBOM. Para realizar escaneos de vulnerabilidades más rápidos, la SBOM generada para la imagen de un contenedor puede usarse como entrada para escanear. A continuación, la SBOM y el informe de escaneo se certifican y se adjuntan a la imagen antes de enviarla a un repositorio OCI central, como Amazon ECR, para su revisión y auditoría.

Obtenga más información sobre cómo proteger su cadena de suministro de software consultando la guía de mejores prácticas de la cadena de suministro de software de la CNCF.

Escanee las imágenes en busca de vulnerabilidades con regularidad

Al igual que sus homólogas de máquinas virtuales, las imágenes de los contenedores pueden contener archivos binarios y bibliotecas de aplicaciones con vulnerabilidades o desarrollar vulnerabilidades con el tiempo. La mejor forma de protegerse contra las vulnerabilidades es escanear las imágenes con regularidad con un escáner de imágenes. Las imágenes almacenadas en Amazon ECR se pueden escanear de forma automática o bajo demanda (una vez durante un período de 24 horas). Actualmente, el ECR admite dos tipos de digitalización: básica y mejorada. El escaneo básico aprovecha Clair, una solución de escaneo de imágenes de código abierto sin costo alguno. El escaneado mejorado utiliza Amazon Inspector para ofrecer escaneos continuos automáticos con un coste adicional. Una vez escaneada una imagen, los resultados se registran en el flujo de eventos para que se registre el ECR. EventBridge También puede ver los resultados de un escaneo desde la consola de ECR. Las imágenes con una vulnerabilidad ALTA o CRÍTICA deben borrarse o reconstruirse. Si una imagen que se ha implementado presenta una vulnerabilidad, debe sustituirse lo antes posible.

Saber dónde se han desplegado las imágenes con vulnerabilidades es esencial para mantener su entorno seguro. Si bien podría crear una solución de seguimiento de imágenes usted mismo, ya existen varias ofertas comerciales que ofrecen esta y otras funciones avanzadas listas para usar, entre las que se incluyen:

- Grype
- Palo Alto - Prisma Cloud (twistcli)
- Aqua
- Kubei
- Trivy
- Snyk

También se podría utilizar un webhook de validación de Kubernetes para validar que las imágenes estén libres de vulnerabilidades críticas. Los webhooks de validación se invocan antes que la API de Kubernetes. Por lo general, se utilizan para rechazar solicitudes que no cumplen con los criterios de validación definidos en el webhook. Este es un ejemplo de un webhook sin servidor que llama a la API ECR describeImageScan Findings para determinar si un módulo está extrayendo una imagen con vulnerabilidades críticas. Si se encuentran vulnerabilidades, se rechaza el pod y se devuelve un mensaje con la lista de CVEs como evento.

Usa atestaciones para validar la integridad del artefacto

Una certificación es una «declaración» firmada criptográficamente en la que se afirma que algo (un «predicado», por ejemplo, un oleoducto o la SBOM o el informe de análisis de vulnerabilidades) es cierto sobre otra cosa, un «sujeto», es decir, la imagen del contenedor.

Las atestaciones ayudan a los usuarios a validar que un artefacto proviene de una fuente confiable de la cadena de suministro del software. Por ejemplo, podemos usar una imagen de contenedor sin conocer todos los componentes o dependencias del software que se incluyen en esa imagen. Sin embargo, si confiamos en lo que diga el productor de la imagen del contenedor sobre el software presente, podemos usar la certificación del productor para basarnos en ese artefacto. Esto significa que podemos proceder a utilizar el artefacto de forma segura en nuestro flujo de trabajo en lugar de haber realizado el análisis nosotros mismos.

- Las atestaciones se pueden crear mediante el cosignado de AWS Signer o Sigstore.
- Los controladores de admisión de Kubernetes, como Kyverno, se pueden usar para verificar las atestaciones.
- Consulte este taller para obtener más información sobre las prácticas recomendadas de administración de la cadena de suministro de software en AWS mediante el uso de herramientas de código abierto, con temas como la creación y el adjunto de atestaciones a una imagen de contenedor.

Cree políticas de IAM para los repositorios de ECR

Hoy en día, no es raro que una organización tenga varios equipos de desarrollo que operen de forma independiente dentro de una cuenta de AWS compartida. Si estos equipos no necesitan compartir activos, puede crear un conjunto de políticas de IAM que restrinjan el acceso a los repositorios con los que cada equipo puede interactuar. Una buena forma de implementar esto es mediante el uso de espacios de nombres ECR. Los espacios de nombres son una forma de agrupar repositorios similares. Por ejemplo, todos los registros del equipo A pueden ir precedidos del team-a/, mientras que los del equipo B pueden usar el prefijo team-b/. La política para restringir el acceso podría tener el siguiente aspecto:

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

Considere la posibilidad de utilizar puntos finales privados de ECR

La API de ECR tiene un punto final público. En consecuencia, se puede acceder a los registros del ECR desde Internet siempre que la solicitud haya sido autenticada y autorizada por el IAM. Para aquellos que necesiten operar en un entorno aislado en el que la VPC del clúster carece de Internet Gateway (IGW), pueden configurar un punto de conexión privado para ECR. La creación de un punto final privado le permite acceder de forma privada a la API de ECR a través de una dirección IP privada en lugar de enrutar el tráfico a través de Internet. Para obtener información adicional sobre este tema, consulte Puntos de enlace de VPC de la interfaz Amazon ECR.

Implemente políticas de puntos finales para ECR

La política de puntos finales predeterminada permite el acceso a todos los repositorios de ECR de una región. Esto podría permitir attacker/insider a una persona filtrar datos empaquetándolos como una imagen de contenedor y enviándolos a un registro de otra cuenta de AWS. Para mitigar este riesgo, es necesario crear una política de puntos finales que limite el acceso de la API a los repositorios de ECR. Por ejemplo, la siguiente política permite que todos los principios de AWS de su cuenta realicen todas las acciones contra sus repositorios de ECR y solo contra los suyos:

```
{
  "Statement": [
    {
      "Sid": "LimitECRAccess",
      "Principal": "*",
```

```
      "Action": "*",
      "Effect": "Allow",
      "Resource": "arn:aws:ecr:<region>:<account_id>:repository/*"
    }
  ]
}
```

Puede mejorarlo aún más si establece una condición que utilice el nuevo `PrincipalOrgID` atributo, lo que impedirá la aparición pushing/pulling de imágenes según un principio de IAM que no forme parte de su organización de AWS. Consulte aws: PrincipalOrg ID para obtener más información. Recomendamos aplicar la misma política tanto a los puntos de conexión `com.amazonaws.<region>.ecr.dkr` como a los `com.amazonaws.<region>.ecr.api`. Dado que EKS extrae imágenes de kube-proxy, coredns y aws-node del ECR, tendrá que añadir el ID de cuenta del registro, por ejemplo, a la lista de recursos de la política de puntos finales o modificar la política `602401143452.dkr.ecr.us-west-2.amazonaws.com/` para permitir la extracción de datos y restringir los envíos a su ID de cuenta. En la siguiente tabla se muestra el mapeo entre las cuentas de AWS desde las que se venden las imágenes de EKS y la región del clúster.

Account Number (Número de cuenta)	Region
602401143452	Todas las regiones comerciales, excepto las que se indican a continuación
—	—
800184023465	ap-east-1 - Asia-Pacífico (Hong Kong)
558608220178	me-south-1 - Oriente Medio (Baréin)
918309763551	cn-north-1 - China (Pekín)
961992271922	cn-northwest-1 - China (Ningxia)

Para obtener más información sobre el uso de políticas de puntos de conexión, consulte Uso de políticas de puntos de conexión de VPC para controlar el acceso a Amazon ECR.

Implemente políticas de ciclo de vida para la ECR

La Guía de seguridad de contenedores de aplicaciones del NIST advierte sobre el riesgo de que se produzcan «imágenes obsoletas en los registros» y señala que, con el tiempo, las imágenes antiguas con paquetes de out-of-date software vulnerables deberían eliminarse para evitar su despliegue y exposición accidentales. Cada repositorio de ECR puede tener una política de ciclo de vida que establezca reglas sobre cuándo caducan las imágenes. La documentación oficial de AWS describe cómo configurar las reglas de prueba, evaluarlas y luego aplicarlas. Hay varios ejemplos de políticas de ciclo de vida en los documentos oficiales que muestran diferentes formas de filtrar las imágenes de un repositorio:

- Filtrar por antigüedad o recuento de imágenes
- Filtrar por imágenes etiquetadas o sin etiquetar
- Filtrar por etiquetas de imagen, ya sea en varias reglas o en una sola regla

??? + advertencia Si la imagen de una aplicación de ejecución prolongada se purga del ECR, se pueden producir errores al extraer la imagen cuando la aplicación se vuelva a implementar o se escale horizontalmente. Cuando utilices políticas sobre el ciclo de vida de las imágenes, asegúrate de contar con buenas CI/CD prácticas para mantener actualizadas las implementaciones y las imágenes a las que hacen referencia, y crea siempre reglas de caducidad para [imágenes] que tengan en cuenta la frecuencia con la que realizas las versiones o despliegues.

Crear un conjunto de imágenes mantenidas

En lugar de permitir que los desarrolladores creen sus propias imágenes, considere la posibilidad de crear un conjunto de imágenes verificadas para los distintos grupos de aplicaciones de su organización. De este modo, los desarrolladores pueden prescindir de aprender cómo componer Dockerfiles y concentrarse en escribir código. A medida que los cambios se fusionan en Master, una CI/CD canalización puede compilar automáticamente el activo, almacenarlo en un repositorio de artefactos y copiar el artefacto en la imagen correspondiente antes de enviarlo a un registro de Docker, como ECR. Como mínimo, deberías crear un conjunto de imágenes base a partir de las cuales los desarrolladores puedan crear sus propios Dockerfiles. Lo ideal sería evitar extraer imágenes de Dockerhub porque 1/ no siempre se sabe lo que hay en la imagen y 2/ aproximadamente una quinta parte de las 1000 imágenes principales tienen vulnerabilidades. Puedes encontrar una lista de esas imágenes y sus vulnerabilidades aquí.

Añada la directiva USER a sus Dockerfiles para ejecutarlos como un usuario no root

Como se mencionó en la sección de seguridad del pod, debes evitar ejecutar el contenedor como root. Si bien puedes configurarlo como parte del PodSpec, es una buena costumbre usar la `USER` directiva en tus Dockerfiles. La `USER` directiva establece el UID que se utilizará cuando se ejecute `RUN` o la `CMD` instrucción que aparece `ENTRYPOINT` después de la directiva USER.

Borra tus Dockerfiles

Puedes usar Linting para comprobar que tus Dockerfiles cumplen una serie de directrices predefinidas, por ejemplo, la inclusión de la `USER` directiva o el requisito de etiquetar todas las imágenes. dockerfile_lint es un proyecto de código abierto RedHat que verifica las mejores prácticas comunes e incluye un motor de reglas que puedes usar para crear tus propias reglas para lintar Dockerfiles. Se puede incorporar a una canalización de CI, ya que las compilaciones con Dockerfiles que infrinjan una regla fallarán automáticamente.

Cree imágenes desde cero

Reducir la superficie de ataque de las imágenes de sus contenedores debe ser el objetivo principal al crear imágenes. La forma ideal de hacerlo es crear un mínimo de imágenes que no contengan archivos binarios que puedan utilizarse para aprovechar las vulnerabilidades. Afortunadamente, Docker tiene un mecanismo a partir del cual crear imágenes. `scratch` Con lenguajes como Go, puedes crear un binario enlazado estático y hacer referencia a él en tu Dockerfile, como en este ejemplo:

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

Esto crea una imagen de contenedor que consiste en tu aplicación y nada más, lo que la hace extremadamente segura.

Utilice etiquetas inmutables con ECR

Las etiquetas inmutables obligan a actualizar la etiqueta de la imagen cada vez que se envía al repositorio de imágenes. Esto puede impedir que un atacante sobrescriba una imagen con una versión maliciosa sin cambiar las etiquetas de la imagen. Además, permite identificar una imagen de forma sencilla y exclusiva.

Firme las imágenes SBOMs, los procesos de ejecución y los informes de vulnerabilidad

Cuando se introdujo Docker por primera vez, no existía un modelo criptográfico para verificar las imágenes de los contenedores. Con la versión 2, Docker agregó resúmenes al manifiesto de imágenes. Esto permitía aplicar un hash a la configuración de una imagen y utilizar el hash para generar un identificador para la imagen. Cuando la firma de imágenes está habilitada, el motor Docker verifica la firma del manifiesto y se asegura de que el contenido proviene de una fuente confiable y de que no se ha producido ninguna alteración. Tras descargar cada capa, el motor verifica el resumen de la capa y se asegura de que el contenido coincide con el contenido especificado en el manifiesto. La firma de imágenes permite crear de forma eficaz una cadena de suministro segura mediante la verificación de las firmas digitales asociadas a la imagen.

Podemos usar AWS Signer o Sigstore Cosign para firmar imágenes de contenedores, crear atestaciones para SBOMs informes de análisis de vulnerabilidades e informes de ejecución de proyectos. Estas certificaciones garantizan la fiabilidad e integridad de la imagen, que, de hecho, ha sido creada por un canal de confianza sin interferencias ni manipulaciones y que solo contiene los componentes de software que están documentados (en la SBOM) y que el editor de la imagen ha verificado y en los que confía. Estas atestaciones se pueden adjuntar a la imagen del contenedor y enviarse al repositorio.

En la siguiente sección veremos cómo utilizar los artefactos certificados para las auditorías y la verificación del controlador de admisiones.

Verificación de la integridad de la imagen mediante el controlador de admisión de Kubernetes

Podemos verificar las firmas de las imágenes y comprobar los artefactos de forma automática antes de implementar la imagen en el clúster de Kubernetes de destino mediante un controlador de admisión dinámico y admitir los despliegues solo cuando los metadatos de seguridad de los artefactos cumplan con las políticas del controlador de admisión.

Por ejemplo, podemos redactar una política que verifique criptográficamente la firma de una imagen, una SBOM certificada, un informe de ejecución de un proceso certificado o un informe de escaneo CVE certificado. Podemos incluir condiciones en la política para comprobar los datos del informe, por ejemplo, un escaneo CVE no debería contener ningún dato crítico. CVEs Solo se permite el despliegue de imágenes que cumplan estas condiciones y el controlador de admisiones rechazará todos los demás despliegues.

Algunos ejemplos de controladores de admisión son:

- Kyverno
- Guardián de la OPA
- Portieris
- Ratificar
- Kritis
- Tutorial de Grafeas
- Bono

Actualiza las imágenes de los paquetes de tu contenedor

Deberías incluir RUN `apt-get update && apt-get upgrade` en tus Dockerfiles para actualizar los paquetes de tus imágenes. Aunque la actualización requiere que se ejecute como root, esto ocurre durante la fase de creación de la imagen. No es necesario que la aplicación se ejecute como root. Puede instalar las actualizaciones y, a continuación, cambiar a otro usuario con la directiva USER. Si la imagen base se ejecuta como usuario no root, cámbiese a root y viceversa; no confíe únicamente en los responsables del mantenimiento de la imagen base para instalar las actualizaciones de seguridad más recientes.

Ejecute `apt-get clean` para eliminar los archivos del instalador. `/var/cache/apt/archives/` También puede ejecutarlo `rm -rf /var/lib/apt/lists/*` después de instalar los paquetes.

Esto elimina los archivos de índice o las listas de paquetes que están disponibles para su instalación. Tenga en cuenta que estos comandos pueden ser diferentes para cada administrador de paquetes. Por ejemplo:

```
RUN apt-get update && apt-get install -y \
    curl \
    git \
    libsqlite3-dev \
    && apt-get clean && rm -rf /var/lib/apt/lists/*
```

Herramientas y recursos

- Taller de inmersión en seguridad de Amazon EKS: seguridad de imagen
- docker-slim Cree imágenes mínimas y seguras
- dockle verifica que su Dockerfile se alinee con las mejores prácticas para crear imágenes seguras
- dockerfile-lint: filtro basado en reglas para Dockerfiles
- hadolint es un linter inteligente de dockerfile
- Gatekeeper y OPA: un controlador de admisión basado en políticas
- Kyverno Un motor de políticas nativo de Kubernetes
- in-toto Permite al usuario verificar si se pretendía realizar un paso de la cadena de suministro y si el actor adecuado lo realizó
- Notary: un proyecto para firmar imágenes de contenedores
- Notario v2
- Grafeas Una API de metadatos de artefactos abierta para auditar y gobernar su cadena de suministro de software
- NeuVector de SUSE, una plataforma de seguridad de contenedores de código abierto y de confianza cero, que permite escanear contenedores, imágenes y registros para detectar vulnerabilidades, secretos y conformidad.

Estrategia de cuentas múltiples

AWS recomienda utilizar una estrategia de cuentas múltiples y organizaciones de AWS para ayudar a aislar y administrar las aplicaciones y los datos empresariales. El uso de una estrategia de cuentas múltiples tiene muchas ventajas:

- Aumento de las cuotas de servicios de API de AWS. Las cuotas se aplican a las cuentas de AWS y el uso de varias cuentas para sus cargas de trabajo aumenta la cuota total disponible para sus cargas de trabajo.
- Políticas de Identity and Access Management (IAM) más sencillas. Conceder a las cargas de trabajo y a los operadores que las respaldan el acceso únicamente a sus propias cuentas de AWS significa menos tiempo para elaborar políticas de IAM detalladas para lograr el principio de privilegios mínimos.
- Aislamiento mejorado de los recursos de AWS. Por diseño, todos los recursos aprovisionados en una cuenta están aislados de forma lógica de los recursos aprovisionados en otras cuentas. Este límite de aislamiento le permite limitar los riesgos de que se produzcan problemas relacionados con la aplicación, una configuración incorrecta o acciones malintencionadas. Si hay un problema en una cuenta, los impactos en las cargas de trabajo de otras cuentas se pueden reducir o eliminar.
- Más ventajas, tal y como se describe en el documento técnico sobre la estrategia de cuentas múltiples de AWS

En las siguientes secciones se explica cómo implementar una estrategia de cuentas múltiples para sus cargas de trabajo de EKS mediante un enfoque de clúster de EKS centralizado o descentralizado.

Planeación de una estrategia de cuentas con múltiples cargas de trabajo para clústeres con varios inquilinos

En una estrategia de AWS con varias cuentas, los recursos que pertenecen a una carga de trabajo determinada, como los buckets de S3, ElastiCache los clústeres y las tablas de DynamoDB, se crean todos en una cuenta de AWS que contiene todos los recursos de esa carga de trabajo. Se denominan cuentas de carga de trabajo y el clúster EKS se implementa en una cuenta denominada cuenta de clúster. Las cuentas de clúster se analizarán en la siguiente sección. La implementación de recursos en una cuenta de carga de trabajo dedicada es similar a la implementación de recursos de Kubernetes en un espacio de nombres dedicado.

Luego, las cuentas de carga de trabajo se pueden desglosar aún más por ciclo de vida de desarrollo de software u otros requisitos, si corresponde. Por ejemplo, una carga de trabajo determinada puede tener una cuenta de producción, una cuenta de desarrollo o cuentas para alojar instancias de esa carga de trabajo en una región específica. Encontrará más información en este documento técnico de AWS.

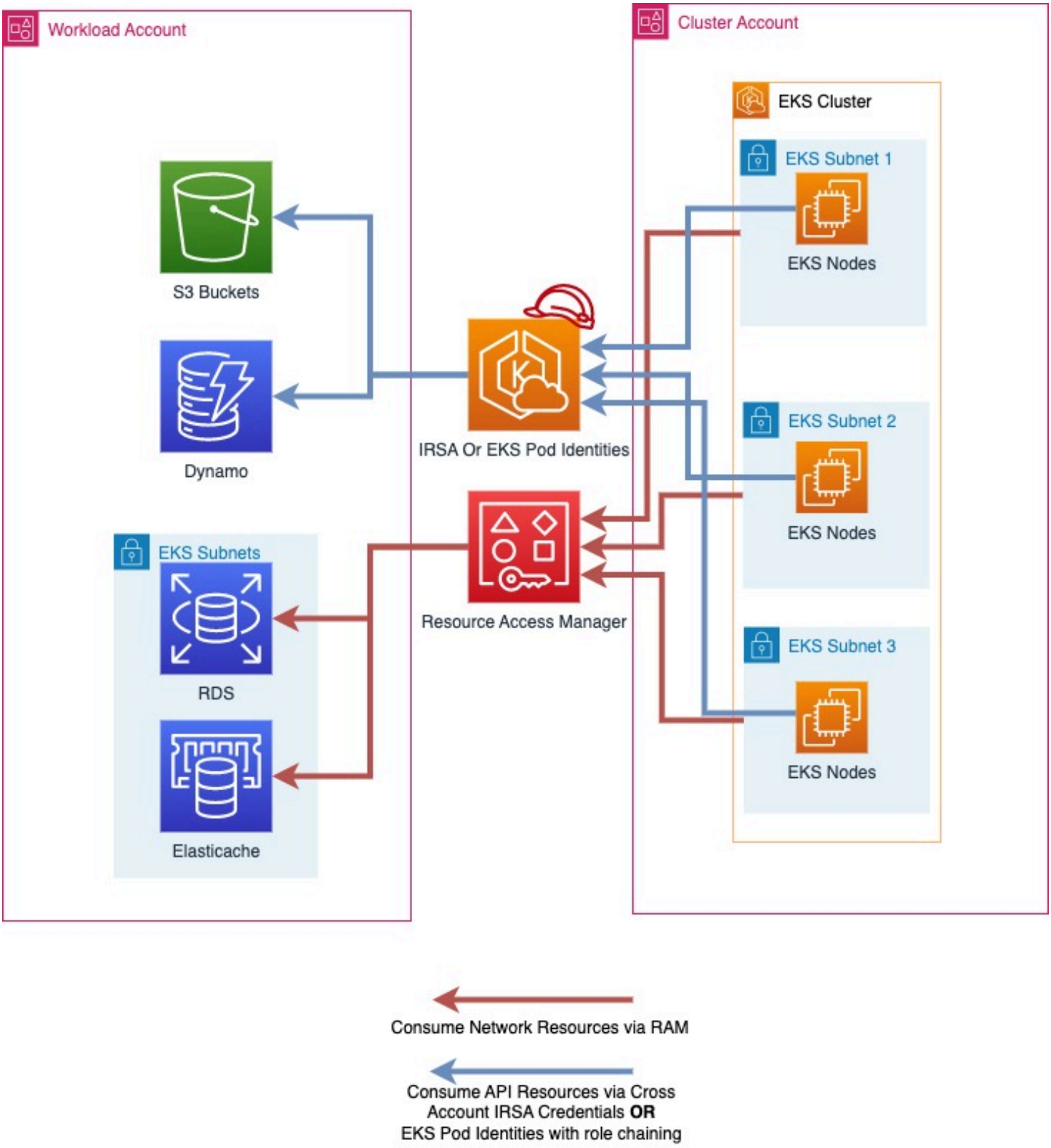
Puede adoptar los siguientes enfoques al implementar la estrategia de cuentas múltiples de EKS:

Clúster EKS centralizado

En este enfoque, su clúster de EKS se implementará en una única cuenta de AWS llamada`Cluster Account`. Al utilizar las funciones de IAM para las cuentas de servicio (IRSA) o las identidades de los pods de EKS para ofrecer credenciales de AWS temporales y AWS Resource Access Manager (RAM) para simplificar el acceso a la red, puede adoptar una estrategia de cuentas múltiples para su clúster de EKS con varios inquilinos. La cuenta de clúster contendrá la VPC, las subredes, el clúster de EKS, los recursos de cómputo de EC2 /Fargate (nodos de trabajo) y cualquier configuración de red adicional necesaria para ejecutar el clúster de EKS.

En una estrategia de cuentas con varias cargas de trabajo para un clúster con varios inquilinos, las cuentas de AWS suelen alinearse con los espacios de nombres de Kubernetes como mecanismo para aislar grupos de recursos. A la hora de implementar una estrategia de cuentas múltiples para clústeres de EKS múltiples, deben seguirse las mejores prácticas de aislamiento de inquilinos dentro de un clúster de EKS de múltiples inquilinos.

Es posible tener varios `Cluster Accounts` en su organización de AWS, y se recomienda tener varios `Cluster Accounts` que se ajusten a las necesidades del ciclo de vida del desarrollo de software. En el caso de las cargas de trabajo que funcionan a gran escala, es posible que necesite varias `Cluster Accounts` para garantizar que haya suficientes cuotas de servicios de Kubernetes y AWS disponibles para todas sus cargas de trabajo.



|En el diagrama anterior, la RAM de AWS se utiliza para compartir subredes de una cuenta de clúster en una cuenta de carga de trabajo. Luego, las cargas de trabajo que se ejecutan en los pods de EKS

utilizan las identidades de los pods de IRSA o EKS y el encadenamiento de roles para asumir un rol en su cuenta de carga de trabajo y acceder a sus recursos de AWS.

Implementación de una estrategia de cuentas de múltiples cargas de trabajo para un clúster con varios inquilinos

Uso compartido de subredes con AWS Resource Access Manager

AWS Resource Access Manager (RAM) le permite compartir recursos entre cuentas de AWS.

Si la RAM está habilitada para su organización de AWS, puede compartir las subredes de VPC de la cuenta del clúster con sus cuentas de carga de trabajo. Esto permitirá que los recursos de AWS que pertenecen a sus cuentas de carga de trabajo, como Amazon ElastiCache Clusters o las bases de datos Amazon Relational Database Service (RDS), se desplieguen en la misma VPC que su clúster de EKS y puedan consumirlos las cargas de trabajo que se ejecutan en su clúster de EKS.

Para compartir un recurso a través de la RAM, abra la RAM en la consola de AWS de la cuenta del clúster y seleccione «Recursos compartidos» y «Crear recurso compartido». Asigne un nombre a su recurso compartido y seleccione las subredes que desee compartir. Vuelva a seleccionar Siguiente e introduzca la cuenta de 12 dígitos IDs para las cuentas de carga de trabajo con las que desee compartir las subredes, vuelva a seleccionar Siguiente y haga clic en Crear recurso compartido para terminar. Tras este paso, la cuenta de carga de trabajo puede implementar recursos en esas subredes.

Los recursos compartidos de RAM también se pueden crear mediante programación o con la infraestructura como código.

Elegir entre EKS Pod Identities e IRSA

En re:Invent 2023, AWS lanzó EKS Pod Identities como una forma más sencilla de entregar credenciales de AWS temporales a sus pods en EKS. Tanto las identidades de los pods de IRSA como las de EKS son métodos válidos para entregar credenciales de AWS temporales a los pods de EKS y seguirán siendo compatibles. Debe considerar qué método de entrega se adapta mejor a sus necesidades.

Al trabajar con un clúster de EKS y varias cuentas de AWS, IRSA puede asumir funciones directamente en las cuentas de AWS distintas de la cuenta en la que está alojado directamente el clúster de EKS, mientras que las identidades de los pods de EKS requieren que configure el encadenamiento de roles. Consulte la documentación de EKS para obtener una comparación detallada.

Acceso a los recursos de API de AWS con funciones de IAM para cuentas de servicio

Las funciones de IAM para cuentas de servicio (IRSA) le permiten entregar credenciales de AWS temporales a sus cargas de trabajo que se ejecutan en EKS. El IRSA se puede utilizar para obtener credenciales temporales para las funciones de IAM en las cuentas de carga de trabajo desde la cuenta del clúster. Esto permite que las cargas de trabajo que se ejecutan en los clústeres de EKS de la cuenta del clúster consuman sin problemas los recursos de la API de AWS, como los buckets de S3 alojados en la cuenta de carga de trabajo, y utilicen la autenticación de IAM para recursos como las bases de datos de Amazon RDS o Amazon EFS. FileSystems

Solo se puede acceder a los recursos de la API de AWS y a otros recursos que utilizan la autenticación de IAM en una cuenta de carga de trabajo mediante las credenciales de las funciones de IAM en esa misma cuenta de carga de trabajo, excepto cuando el acceso entre cuentas es posible y se ha habilitado de forma explícita.

Habilitar IRSA para el acceso entre cuentas

Para permitir que el IRSA para las cargas de trabajo de su cuenta de clúster acceda a los recursos de sus cuentas de carga de trabajo, primero debe crear un proveedor de identidades OIDC de IAM en su cuenta de carga de trabajo. Esto se puede hacer con el mismo procedimiento para configurar el IRSA, con la salvedad de que el proveedor de identidad se creará en la cuenta de carga de trabajo.

A continuación, al configurar IRSA para sus cargas de trabajo en EKS, puede seguir los mismos pasos que en la documentación, pero utilizar el identificador de cuenta de 12 dígitos de la cuenta de carga de trabajo, tal y como se indica en la sección «Ejemplo de creación de un proveedor de identidades a partir del clúster de otra cuenta».

Una vez configurado, la aplicación que se ejecute en EKS podrá usar directamente su cuenta de servicio para asumir una función en la cuenta de carga de trabajo y utilizar los recursos que contiene.

Acceso a los recursos de la API de AWS con las identidades de los pods de EKS

EKS Pod Identities es una nueva forma de entregar credenciales de AWS a las cargas de trabajo que se ejecutan en EKS. Las identidades de los pods de EKS simplifican la configuración de los recursos de AWS, ya que ya no es necesario gestionar las configuraciones de OIDC para entregar las credenciales de AWS a los pods de EKS.

Habilitar las identidades de los pods de EKS para el acceso entre cuentas

A diferencia de IRSA, las identidades de los pods de EKS solo se pueden usar para conceder acceso directo a un rol en la misma cuenta que el clúster de EKS. Para acceder a un rol en otra cuenta de AWS, los pods que usan identidades de pods de EKS deben realizar el encadenamiento de roles.

El encadenamiento de roles se puede configurar en el perfil de una aplicación con su archivo de configuración de AWS mediante el proveedor de credenciales de proceso disponible en varios AWS SDKs. `credential_processse` puede utilizar como fuente de credenciales al configurar un perfil, por ejemplo:

```
# Content of the AWS Config file
[profile account_b_role]
source_profile = account_a_role
role_arn = arn:aws:iam::444455556666:role/account-b-role

[profile account_a_role]
credential_process = /eks-credential-processrole.sh
```

La fuente del script llamado por credential_process:

```
#!/bin/bash
# Content of the eks-credential-processrole.sh
# This will retreive the credential from the pod identities agent,
# and return it to the AWS SDK when referenced in a profile
curl -H "Authorization: $(cat $AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE)"
 $AWS_CONTAINER_CREDENTIALS_FULL_URI | jq -c '{AccessKeyId: .AccessKeyId,
 SecretAccessKey: .SecretAccessKey, SessionToken: .Token, Expiration: .Expiration,
 Version: 1}'
```

Puede crear un archivo de configuración de AWS, como se muestra arriba, con las funciones de cuenta A y B y especificar las variables AWS_CONFIG_FILE y AWS_PROFILE env en las especificaciones de su pod. El webhook de identidad de EKS Pod no se anula si las variables de entorno ya existen en la especificación del pod.

```
# Snippet of the PodSpec
containers:
  - name: container-name
    image: container-image:version
    env:
```

```
    - name: AWS_CONFIG_FILE
      value: path-to-customer-provided-aws-config-file
    - name: AWS_PROFILE
      value: account_b_role
```

Al configurar políticas de confianza de roles para encadenar roles con las identidades de los pods de EKS, puedes hacer referencia a los atributos específicos de EKS como etiquetas de sesión y usar el control de acceso basado en atributos (ABAC) para limitar el acceso a tus roles de IAM únicamente a sesiones de identidad específicas del pod de EKS, como la cuenta de servicio de Kubernetes a la que pertenece un pod.

Tenga en cuenta que algunos de estos atributos pueden no ser universalmente únicos, por ejemplo, dos clústeres de EKS pueden tener espacios de nombres idénticos y un clúster puede tener cuentas de servicio con nombres idénticos en todos los espacios de nombres. Por lo tanto, al conceder el acceso a través de EKS Pod Identities y ABAC, se recomienda tener siempre en cuenta el arn y el espacio de nombres del clúster al conceder el acceso a una cuenta de servicio.

Identidades de ABAC y EKS Pod para el acceso entre cuentas

Al utilizar EKS Pod Identities para asumir funciones (encadenamiento de funciones) en otras cuentas como parte de una estrategia de cuentas múltiples, tiene la opción de asignar una función de IAM única a cada cuenta de servicio que necesite acceder a otra cuenta, o utilizar una función de IAM común en varias cuentas de servicio y usar ABAC para controlar a qué cuentas puede acceder.

Para usar ABAC para controlar qué cuentas de servicio pueden asumir un rol en otra cuenta mediante el encadenamiento de roles, debe crear una declaración de política de confianza de roles que solo permita que una sesión de rol asuma un rol cuando estén presentes los valores esperados. La siguiente política de confianza de roles solo permitirá que un rol de la cuenta del clúster de EKS (ID de cuenta 111122223333) asuma un rol si todas las `kubernetes-service-account` `kubernetes-namespace` etiquetas `eks-cluster-arn` y etiquetas tienen el valor esperado.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "sts:AssumeRole",
```

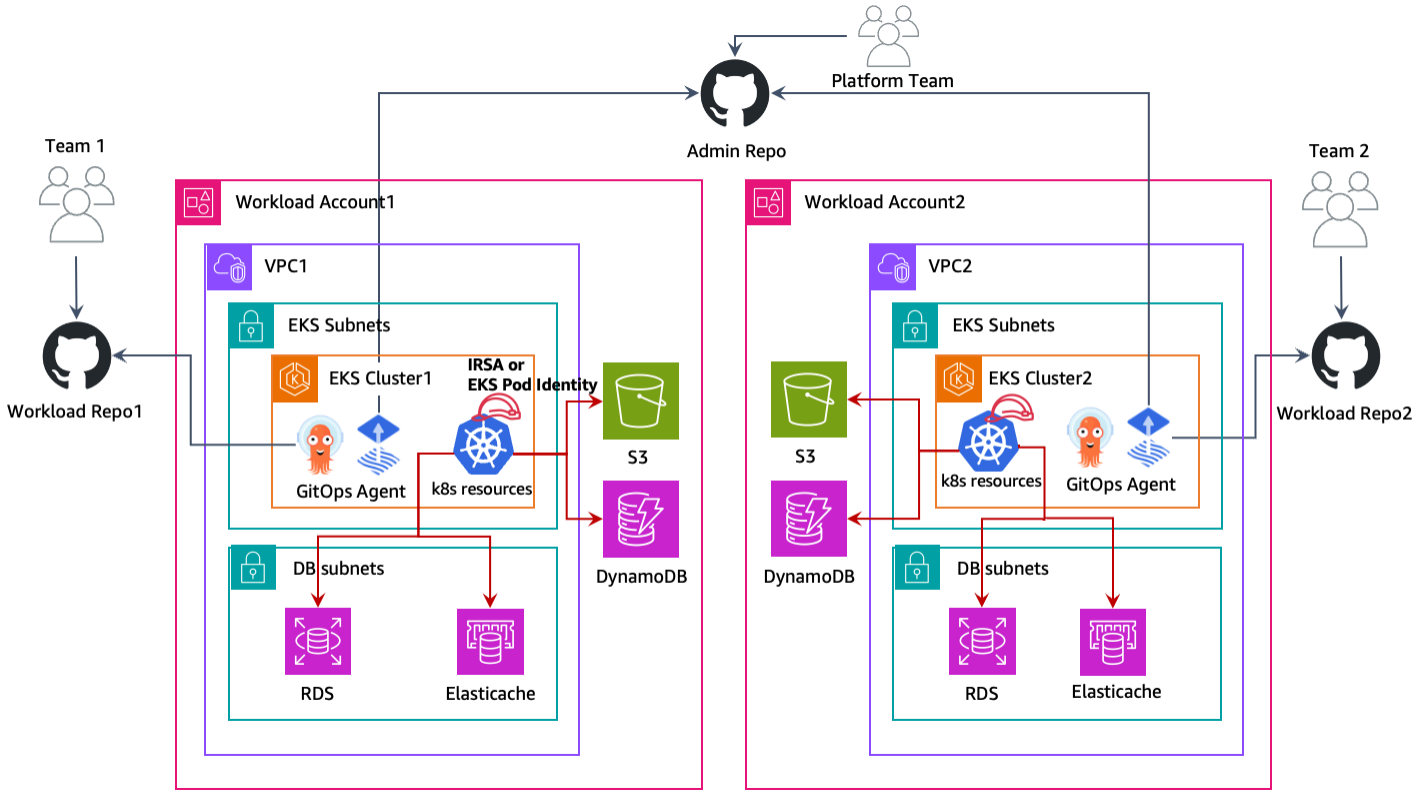
```
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalTag/kubernetes-service-account":
 "PayrollApplication",
                    "aws:PrincipalTag/eks-cluster-arn": "arn:aws:eks:us-
east-1:111122223333:cluster/ProductionCluster",
                    "aws:PrincipalTag/kubernetes-namespace": "PayrollNamespace"
                }
            }
        }
    ]
}
```

Al utilizar esta estrategia, se recomienda garantizar que el rol común de IAM solo tenga `sts:AssumeRole` permisos y ningún otro acceso de AWS.

Al utilizar ABAC, es importante controlar quién puede asignar funciones y usuarios de IAM únicamente a aquellos que tengan una necesidad estricta de hacerlo. Una persona con la capacidad de etiquetar un rol o un usuario de IAM podría establecer etiquetas roles/users idénticas a las que configuraría EKS Pod Identities y podría aumentar sus privilegios. Puede restringir quién tiene acceso para establecer etiquetas, etiquetas y etiquetas para el rol `kubernetes-` y `eks-` los usuarios de IAM mediante la política de IAM o la Política de control de servicios (SCP).

Clústeres EKS descentralizados

Con este enfoque, los clústeres de EKS se implementan en las cuentas de AWS de carga de trabajo respectivas y conviven con otros recursos de AWS, como depósitos de Amazon S3 VPCs, tablas de Amazon DynamoDB, etc. Cada cuenta de carga de trabajo es independiente, autosuficiente y está gestionada por el clúster Unit/Application teams. This model allows the creation of reusable blueprints for various cluster capabilities — AI/ML empresarial correspondiente, procesamiento por lotes, uso general, etc., y venden los clústeres en función de los requisitos del equipo de aplicación. Tanto los equipos de aplicaciones como los de plataformas operan desde sus GitOpsrepositorios respectivos para gestionar las implementaciones en los clústeres de carga de trabajo.



En el diagrama anterior, los clústeres de Amazon EKS y otros recursos de AWS se implementan en las cuentas de carga de trabajo respectivas. Luego, las cargas de trabajo que se ejecutan en los pods de EKS utilizan las identidades de los pods de IRSA o EKS para acceder a sus recursos de AWS.

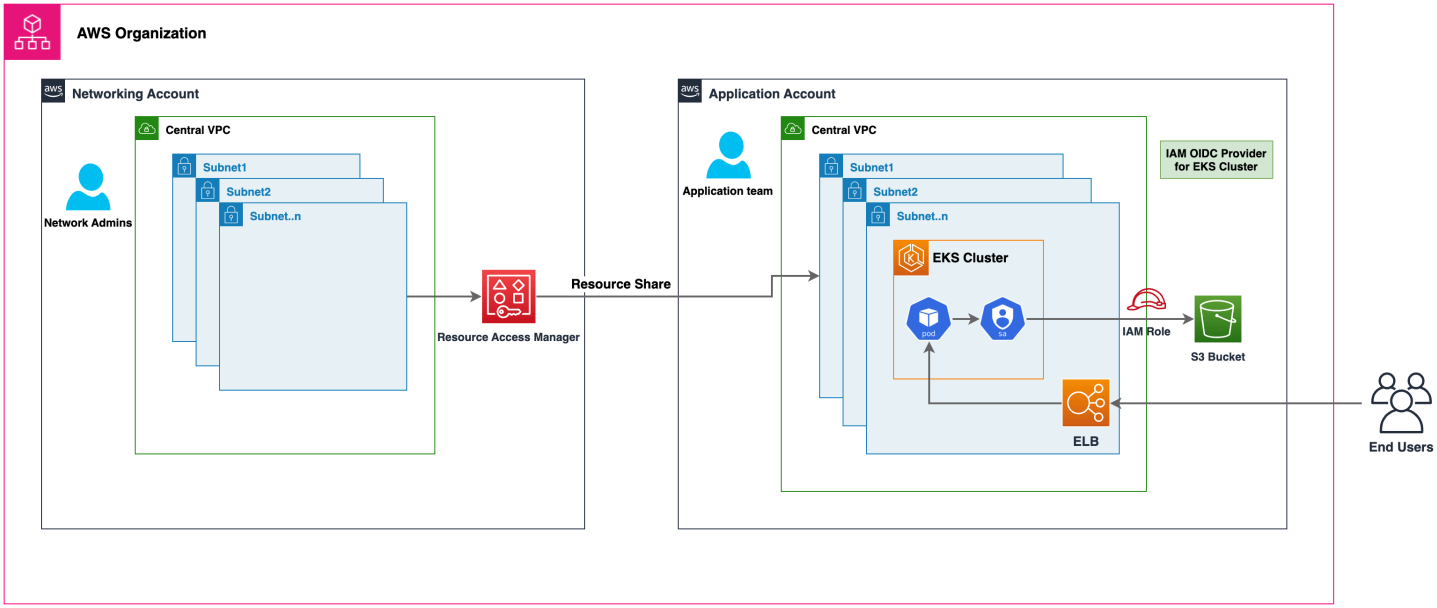
GitOps es una forma de gestionar el despliegue de aplicaciones e infraestructuras de forma que todo el sistema se describa de forma declarativa en un repositorio de Git. Se trata de un modelo operativo que permite gestionar el estado de varios clústeres de Kubernetes mediante las mejores prácticas de control de versiones, artefactos inmutables y automatización. En este modelo de varios clústeres, cada clúster de carga de trabajo se inicia con varios repositorios de Git, lo que permite a cada equipo (aplicación, plataforma, seguridad, etc.) implementar sus cambios respectivos en el clúster.

Debería utilizar las funciones de IAM para las cuentas de servicio (IRSA) o las identidades de los pods de EKS en cada cuenta para permitir que sus cargas de trabajo de EKS obtengan credenciales de AWS temporales para acceder de forma segura a otros recursos de AWS. Las funciones de IAM se crean en las respectivas cuentas de AWS de carga de trabajo y se asignan a las cuentas de servicio de k8s para proporcionar acceso temporal a IAM. Por lo tanto, en este enfoque no se requiere el acceso entre cuentas. Siga la documentación sobre las funciones de IAM para las cuentas de servicio sobre cómo configurar cada carga de trabajo para IRSA y la documentación

sobre las identidades de los pods de EKS sobre cómo configurar las identidades de los pods de EKS en cada cuenta.

Redes centralizadas

También puede utilizar la RAM de AWS para compartir las subredes de la VPC con las cuentas de carga de trabajo y lanzar clústeres de Amazon EKS y otros recursos de AWS en ellas. Esto permite una gestión/administración de la red centralizada, una conectividad de red simplificada y clústeres EKS descentralizados. Consulte este blog de AWS para obtener un recorrido detallado y consideraciones sobre este enfoque.



En el diagrama anterior, la RAM de AWS se utiliza para compartir subredes de una cuenta de red central en una cuenta de carga de trabajo. Luego, el clúster EKS y otros recursos de AWS se lanzan en esas subredes en las cuentas de carga de trabajo respectivas. Los pods de EKS utilizan las identidades de los pods de IRSA o EKS para acceder a sus recursos de AWS.

Clústeres de EKS centralizados o descentralizados

La decisión de utilizar un sistema centralizado o descentralizado dependerá de sus requisitos. En esta tabla se muestran las principales diferencias con cada estrategia.

#	Clúster EKS centralizado	Clústeres EKS descentralizados
Administración de clústeres:	Administrar un único clúster de EKS es más fácil que administrar varios clústeres	Es necesaria una automatización eficiente de la administración de clústeres para reducir la sobrecarga operativa que implica la administración de varios clústeres de EKS
Rentabilidad:	Permite la reutilización de los recursos de clúster y red de EKS, lo que promueve la rentabilidad	Requiere configuraciones de redes y clústeres por carga de trabajo, lo que requiere recursos adicionales
Resiliencia:	Si un clúster se deteriora, es posible que varias cargas de trabajo del clúster centralizado se vean afectadas	Si un clúster se deteriora, los daños se limitan únicamente a las cargas de trabajo que se ejecutan en ese clúster. El resto de las cargas de trabajo no se ven afectadas
Aislamiento y seguridad:	El aislamiento y la multitenencia suave se logran utilizando construcciones nativas de k8s, como. `Namespaces` Las cargas de trabajo pueden compartir los recursos subyacentes, como la CPU, la memoria, etc. Los recursos de AWS están aislados en sus propias cuentas de carga de trabajo a las que, de forma predeterminada, no se puede acceder desde otras cuentas de AWS.	Mayor aislamiento de los recursos informáticos, ya que las cargas de trabajo se ejecutan en clústeres y nodos individuales que no comparten ningún recurso. Los recursos de AWS están aislados en sus propias cuentas de carga de trabajo a las que, de forma predeterminada, no se puede acceder desde otras cuentas de AWS.

#	Clúster EKS centralizado	Clústeres EKS descentralizados
Rendimiento y escalabilidad:	A medida que las cargas de trabajo crecen a escalas muy grandes, es posible que encuentre cuotas de servicio de Kubernetes y AWS en la cuenta del clúster. Puede implementar cuentas de clúster adicionales para escalar aún más	A medida que VPCs hay más clústeres, cada carga de trabajo tiene más k8 disponibles y una cuota de servicio de AWS
Networking:	Se utiliza una sola VPC por clúster, lo que permite una conectividad más sencilla para las aplicaciones de ese clúster	El enrutamiento debe establecerse entre el clúster EKS descentralizado VPCs
Administración de acceso a Kubernetes:	Es necesario mantener muchos roles y usuarios diferentes en el clúster para proporcionar acceso a todos los equipos de carga de trabajo y garantizar que los recursos de Kubernetes estén debidamente segregados	Gestión del acceso simplificada, ya que cada clúster está dedicado a una carga de trabajo o un equipo

#	Clúster EKS centralizado	Clústeres EKS descentralizados
Administración de acceso a AWS:	Los recursos de AWS se implementan en su propia cuenta, a la que solo se puede acceder de forma predeterminada con las funciones de IAM en la cuenta de carga de trabajo. Las funciones de IAM en las cuentas de carga de trabajo se asumen de forma cruzada con las identidades de IRSA o EKS Pod.	Los recursos de AWS se implementan en su propia cuenta, a la que solo se puede acceder de forma predeterminada con las funciones de IAM en la cuenta de carga de trabajo. Las funciones de IAM en las cuentas de carga de trabajo se asignan directamente a los pods con identidades de pod IRSA o EKS

Administración del acceso a los clústeres

La gestión eficaz del acceso es fundamental para mantener la seguridad e integridad de los clústeres de Amazon EKS. En esta guía, se analizan varias opciones para la administración del acceso a EKS, centrándose en el uso de AWS IAM Identity Center (anteriormente, AWS SSO). Compararemos diferentes enfoques, analizaremos sus ventajas y desventajas y destacaremos las limitaciones y consideraciones conocidas.

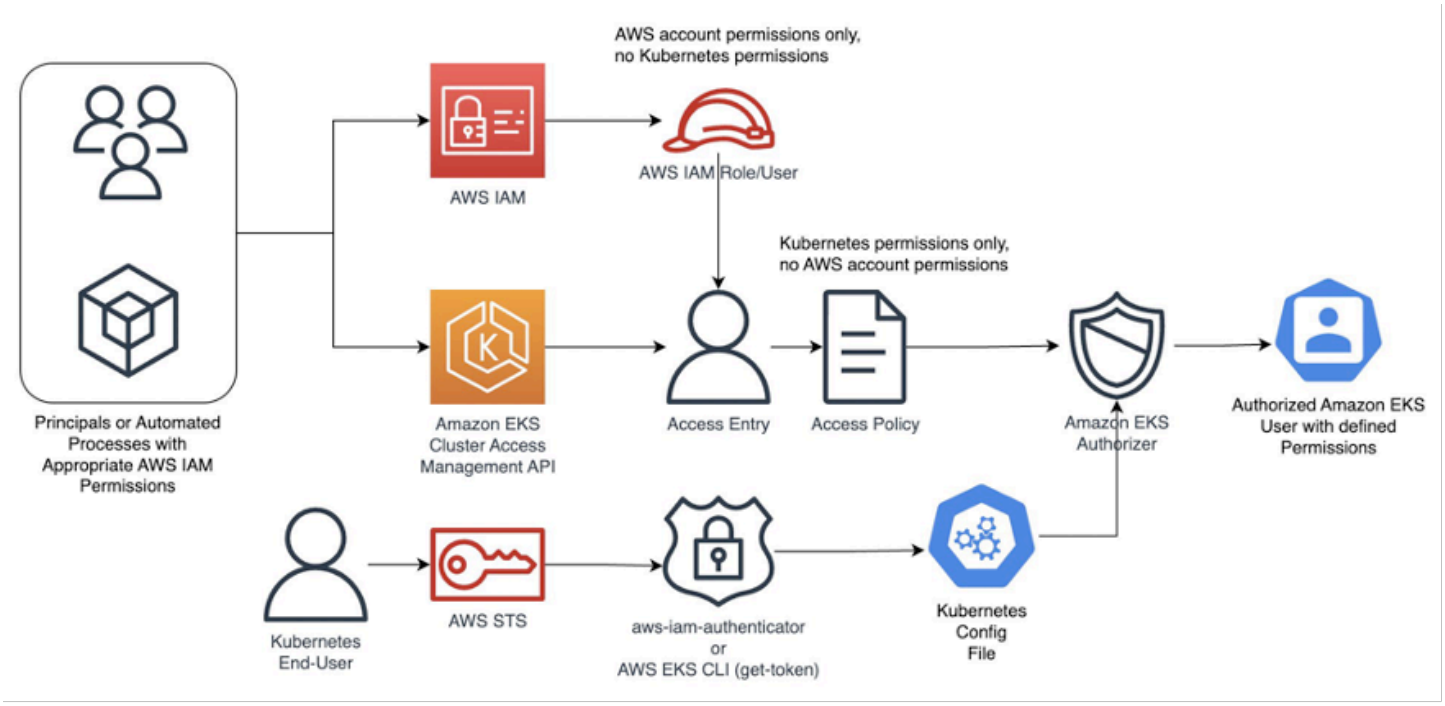
Opciones de administración de acceso de EKS

> **Note**
>
> ConfigMapLa gestión de acceso basada en clústeres (aws-auth ConfigMap) ha quedado obsoleta y se ha sustituido por la API de gestión de acceso a clústeres (CAM). En el caso de los nuevos clústeres de EKS, implemente la API CAM para gestionar el acceso a los clústeres. Para los clústeres existentes que utilizan aws-auth ConfigMap, migre a la API CAM.

Opción 1: AWS IAM Identity Center con API de administración de acceso a clústeres (CAM)

- Administración centralizada de usuarios y permisos
- Integración con los proveedores de identidad existentes (por ejemplo, Microsoft AD, Okta PingId y más)
- La API CAM utiliza entradas de acceso para vincular a los principales (usuarios o roles) de AWS IAM con el clúster de EKS. Estas entradas funcionan con las identidades administradas del IAM Identity Center, lo que permite a los administradores controlar el acceso al clúster para los usuarios y grupos definidos en Identity Center.

Flujo de autenticación del clúster EKS:



1. Los directores (usuarios humanos) o los procesos automatizados se autentican mediante AWS IAM presentando los permisos de cuenta de AWS correspondientes. En este paso, se asignan al principal de AWS IAM correspondiente (rol o usuario).
2. A continuación, una entrada de acceso a EKS asigna este principio de IAM a un principal RBAC de Kubernetes (usuario o grupo) mediante la definición de la política de acceso adecuada, que solo contiene los permisos de Kubernetes.

3. Cuando un usuario final de Kubernetes intenta acceder a un clúster, su solicitud de autenticación es procesada por la CLI de aws-iam-authenticator AWS EKS y validada en el contexto del clúster en el archivo kubeconfig.
4. Por último, el autorizador de EKS verifica los permisos asociados a la entrada de acceso del usuario autenticado y concede o deniega el acceso en consecuencia.
    - La API utiliza políticas de acceso específicas de Amazon EKS para definir el nivel de autorización de cada entrada de acceso. Estas políticas se pueden asignar a las funciones y permisos configurados en el Centro de identidad de IAM, lo que garantiza un control de acceso uniforme en todos los servicios de AWS y los clústeres de EKS.

Ventajas en comparación con la gestión de acceso ConfigMap basada en bases:

1. Menor riesgo de errores de configuración: la administración directa basada en API elimina los errores comunes asociados con la edición manual ConfigMap . Esto ayuda a evitar eliminaciones accidentales o errores de sintaxis que podrían impedir que los usuarios accedan al clúster.
2. Principio de privilegios mínimos mejorado: elimina la necesidad del permiso de administrador del clúster de la identidad del creador del clúster y permite una asignación de permisos más detallada y adecuada. Puede optar por añadir este permiso para los casos de uso de breakglass.
3. Modelo de seguridad mejorado: proporciona una validación integrada de las entradas de acceso antes de que se apliquen. Además, ofrece una integración más estrecha con AWS IAM para la autenticación.
4. Operaciones simplificadas: ofrece una forma más intuitiva de administrar los permisos mediante herramientas nativas de AWS.

Prácticas recomendadas:

1. Utilice AWS Organizations para administrar varias cuentas y aplicar políticas de control de servicios (SCPs).
2. Implemente el principio de privilegios mínimos mediante la creación de conjuntos de permisos específicos para diferentes funciones de EKS (p. ej., administrador, desarrollador o de solo lectura).
3. Utilice el control de acceso basado en atributos (ABAC) para asignar dinámicamente los permisos a los pods en función de los atributos del usuario.
4. Audite y revise periódicamente los permisos de acceso.

Consideraciones y limitaciones:

1. Los roles ARNs generados por Identity Center tienen sufijos aleatorios, lo que dificulta su uso en configuraciones estáticas.
2. Soporte limitado para permisos detallados a nivel de recursos de Kubernetes. Se requiere una configuración adicional para las funciones RBAC personalizadas de Kubernetes. Además del RBAC nativo de Kubernetes, considera usar Kyverno para la administración avanzada de permisos en los clústeres de EKS.

Opción 2: AWS IAM Users/Roles mapeado a grupos de Kubernetes

Ventajas:

1. Control detallado de los permisos de IAM.
2. Función estática y predecible ARNs

Desventajas:

1. Aumento de la sobrecarga de administración de las cuentas de usuario
2. Falta de administración de identidad centralizada
3. Potencial de proliferación de entidades de IAM

Prácticas recomendadas:

1. Utilice funciones de IAM en lugar de usuarios de IAM para mejorar la seguridad y la capacidad de administración
2. Implemente una convención de nomenclatura para las funciones a fin de garantizar la coherencia y la facilidad de administración
3. Utilice las condiciones de la política de IAM para restringir el acceso en función de las etiquetas u otros atributos.
4. Cambie las claves de acceso y revise los permisos con regularidad.

Consideraciones y limitaciones:

1. Problemas de escalabilidad al administrar una gran cantidad de usuarios o roles

2. No hay capacidades de inicio de sesión único integradas

Opción 3: proveedores de OIDC

Ventajas:

1. Integración con los sistemas de gestión de identidad existentes
2. Reducción de la sobrecarga de administración de las cuentas de usuario

Desventajas:

1. Complejidad de configuración adicional
2. Posibilidad de aumentar la latencia durante la autenticación
3. Dependencia de un proveedor de identidad externo

Mejores prácticas:

1. Configure cuidadosamente el proveedor del OIDC para garantizar la validación segura del token.
2. Utilice tokens de corta duración e implemente mecanismos de actualización de los mismos.
3. Audite y actualice periódicamente las configuraciones del OIDC.

Consulte esta guía para obtener una implementación de referencia sobre la integración de proveedores de inicio de sesión único externos con Amazon EKS.

Consideraciones y limitaciones:

1. Integración nativa limitada con los servicios de AWS en comparación con IAM.
2. La URL del emisor del proveedor del OIDC debe ser de acceso público para que EKS pueda descubrir las claves de firma.

AWS EKS Pod Identity frente a IRSA para cargas de trabajo

Amazon EKS ofrece dos formas de conceder permisos de IAM de AWS a las cargas de trabajo que se ejecutan en los clústeres de Amazon EKS: funciones de IAM para cuentas de servicio (IRSA) e identidades de pods de EKS.

Si bien las identidades de pod IRSA y EKS ofrecen las ventajas del acceso con privilegios mínimos, el aislamiento de credenciales y la auditabilidad, la forma recomendada de conceder permisos a las cargas de trabajo es EKS Pod Identity.

Para obtener una guía detallada sobre la identidad y las credenciales de los pods de EKS, consulte la sección Identidades y credenciales de las mejores prácticas de seguridad.

Recomendación

Combine el centro de identidad de IAM con la API de CAM

- Administración simplificada: al usar la API de administración de acceso a clústeres junto con IAM Identity Center, los administradores pueden administrar el acceso a los clústeres de EKS junto con otros servicios de AWS, lo que reduce la necesidad de cambiar entre diferentes interfaces o editar ConfigMaps manualmente.
- Utilice las entradas de acceso para administrar los permisos de Kubernetes de las entidades principales de IAM ajenos al clúster. Puede añadir y administrar el acceso al clúster mediante la API de EKS, la interfaz de línea de comandos de AWS, AWS SDKs CloudFormation, AWS y la consola de administración de AWS. Esto significa que puede administrar los usuarios con las mismas herramientas con las que creó el clúster.
- Los permisos granulares de Kubernetes se pueden aplicar mapeando los usuarios o grupos de Kubernetes con los principios de IAM asociados a las identidades de SSO mediante entradas de acceso y políticas de acceso.
- Para empezar, siga Cambiar el modo de autenticación para usar entradas de acceso y, a continuación, Migrar las entradas de aws-auth existentes a entradas de acceso. ConfigMap

Prácticas recomendadas para el escalado automático de clústeres

Esta guía proporciona consejos sobre el escalado automático de clústeres, incluida información sobre el modo automático, Karpenter y el escalador automático de clústeres de Kubernetes.

Temas

- Modo automático de EKS
- Karpenter
- Escalador automático de clústeres

Modo automático de EKS

> **Tip**
>
> ¿Le interesa adquirir experiencia práctica con el modo automático Amazon EKS? Inscríbase en AWS Connected Community para participar en un próximo taller virtual de Kubernetes dirigido por expertos de AWS.

Amazon EKS Auto Mode representa una evolución significativa en la administración de la infraestructura de Kubernetes, ya que combina una infraestructura de clústeres segura y escalable con las capacidades de Kubernetes integradas administradas por AWS. El servicio proporciona operaciones de nodos de trabajo totalmente gestionadas, lo que elimina la necesidad de que los clientes tengan que configurar grupos o grupos de nodos gestionados. AutoScaling

La principal diferencia arquitectónica es que el modo automático de EKS utiliza un sistema basado en Karpenter que aprovisiona automáticamente las EC2 instancias en respuesta a las solicitudes de los módulos. Estas instancias se ejecutan en Bottlerocket AMIs con complementos preinstalados, como los controladores CSI de EBS, lo que hace que la infraestructura sea realmente gestionada por AWS. A diferencia de los métodos de escalado tradicionales:

- El escalador automático de clústeres (CAS) tradicional requiere la administración manual de los grupos de nodos y solo puede crear nodos con un único tipo de instancia por grupo de nodos

- El Karpenter autogestionado ofrece más flexibilidad al trabajar con la API EC2 Fleet y puede aprovisionar distintos tipos de instancias, pero requiere la gestión de los clientes
- El modo automático de EKS gestiona todas las operaciones de escalado de forma automática mediante operaciones gestionadas y NodePools NodeClasses

El nuevo sistema introduce varias mejoras operativas:

- Escalado automático basado en cápsulas sin configuración manual de grupos de nodos
- Controladores balanceadores de carga gestionados integrados que crean automáticamente ALB/NLB en función de los recursos de Ingress
- Funciones de seguridad integradas con una identidad de pod preconfigurada
- Tiempo de ejecución máximo del nodo de 21 días con reemplazo automático

Desde el punto de vista de los costes, el modo automático de EKS mantiene EC2 los precios estándar y añade una cuota de gestión únicamente para los nodos gestionados por el modo automático. Lo que es más importante, los clientes pueden seguir combinando nodos gestionados en modo automático con nodos autogestionados en el mismo clúster.

Si bien AWS se encarga de la mayoría de los aspectos operativos, los clientes siguen siendo responsables de la administración de las versiones del clúster y pueden realizar actualizaciones controladas que activen actualizaciones continuas de los nodos de trabajo.

Razones para usar el modo automático

El modo automático está dirigido a los usuarios que desean disfrutar de las ventajas de Kubernetes y EKS, pero que necesitan minimizar la carga operativa en torno a Kubernetes, como las actualizaciones, y las partes installation/maintenance fundamentales de la plataforma, como el autoscalamiento, el equilibrio de carga y el almacenamiento. El modo automático lleva a EKS un paso más allá al minimizar el trabajo pesado indiferenciado que conlleva el mantenimiento de Kubernetes

Preguntas frecuentes

¿Cuál es la diferencia entre el modo automático EKS y el Karpenter de código abierto?

El modo automático EKS es un amplio conjunto de funciones que simplifican el funcionamiento de Kubernetes aptos para producción. Una de estas características son las ventajas del

autoscalamiento de Karpenter, totalmente gestionado. Desde el punto de vista de las operaciones, la única diferencia es que en el modo automático EKS no es necesario gestionar el despliegue, el escalado y la actualización de los módulos Karpenter por sí mismos. Todas las demás operaciones, como las NodeClasses gestionadas, NodePools funcionan igual que con Karpenter de código abierto.

¿Puedo ejecutar grupos de nodos gestionados junto con nodos gestionados en modo automático?

Sí, puedes ejecutar nodos estáticos a través de un grupo de nodos gestionado junto con los nodos de escalado automático que proporciona el modo automático

¿Puedo migrar un clúster del modo EKS estándar al modo automático de EKS?

Sí, las instrucciones para activar el modo automático de EKS en un clúster existente se encuentran en la documentación oficial de AWS

Aspectos a tener en cuenta: 1. Tras activar el modo automático, querrá desinstalar todos los componentes que haya instalado y que ahora se administren mediante el modo automático, como Karpenter o el AWS Load Balancer Controller 2. Debe asegurarse de que los complementos instalados lo estén. up-to-date Consulte la documentación.

¿Cómo se configura NodePools en el modo automático EKS?

Un nuevo clúster vendrá preconfigurado con dos NodePools

de uso general

```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
```

Esto NodePool indica a Karpenter que lance nodos con las siguientes características:

1. Tipo de capacidad de «On Demand»
2. Tipos de instancia de C, M o R
3. Generación de instancias de 4
4. Arquitectura AMD
5. SO Linux

También define qué es la lógica de reducción al declarar que solo el 10% de todos los nodos pueden estar en un estado interrumpido en un momento dado y que la consolidación solo debe producirse cuando los nodos están vacíos o infrautilizados.

sistema

```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 similar al «de uso general», excepto por las siguientes diferencias:

1. Admite nodos con la arquitectura ARM y la arquitectura AMD
2. Contamina estos nodos con un NoSchedule a menos que haya una tolerancia para ««»CriticalAddonsOnly. Esto es para uso interno de los complementos de EKS

personalizado

Puede crear su propia personalización en NodePools función de sus necesidades. Para obtener más informaciónNodePools , consulte la documentación de Karpenter.

¿Puedo personalizar la AMI que usa el modo automático cuando se lanzan nuevos nodos?

No, actualmente los únicos compatibles AMIs son los Bottlerocket proporcionados por Amazon

¿Cómo puedo instalar herramientas o agentes personalizados en mis hosts de Kubernetes?

Como no se admite la personalización de la AMI, si necesita un software a nivel de host para tareas como el escaneo de seguridad, debería implementar la carga de trabajo como un Kubernetes. DaemonSet

¿Qué componentes se ejecutan en el plano de datos de mi clúster cuando aprovisiono un nuevo clúster de modo automático de EKS?

Si crea un clúster con eksctl o la consola de AWS, los únicos pods que se ejecutan en un clúster de EKS Auto Mode son los pods de Kubernetes Metrics Server. Los demás componentes del modo automático de EKS, como Karpenter, el controlador Load Balancer de AWS y el controlador CSI de EBS, se ejecutan y administran fuera del clúster.

¿Qué componentes gestionados se están ejecutando para dar soporte a mi nuevo clúster EKS Auto Mode?

El modo automático de EKS automatiza por completo el despliegue de la mayoría de las piezas de un plano de datos necesarias para un Kubernetes apto para producción. Esto incluye:

- Karpenter, para autoescalar el cómputo de su clúster

- Controlador de AWS Load Balancer que le permite exponer fácilmente los servicios de Kubernetes mediante la integración automatizada de Elastic Load Balancer
- EBS CSI
- PVC CON
- Agente de Pod Identity de EKS

¿Cómo soluciono los problemas de los componentes del modo automático que solían funcionar como pods en mi clúster?

Con el modo automático de EKS, muchos de los componentes, como el controlador de Load Balancer de AWS y Karpenter, se administran fuera del clúster, por lo que no tendrá la misma visibilidad de los registros a la que está acostumbrado cuando se autoadministra. Si se encuentra en una situación en la que necesita solucionar problemas relacionados con la funcionalidad de una funcionalidad del modo automático, cree un ticket de AWS Support.

Karpenter

Karpenter es un proyecto de código abierto diseñado para mejorar la administración del ciclo de vida de los nodos en los clústeres de Kubernetes. Automatiza el aprovisionamiento y el desaprovisionamiento de los nodos en función de las necesidades de programación específicas de los pods, lo que permite un escalado eficiente y una optimización de costes. Sus funciones principales son:

- Supervise los pods que el programador de Kubernetes no puede programar debido a limitaciones de recursos.
- Evalúe los requisitos de programación (solicitudes de recursos, selectores de nodos, afinidades, tolerancias, etc.) de los módulos no programables.
- Aprovisione nuevos nodos que cumplan con los requisitos de esos módulos.
- Elimine los nodos cuando ya no los necesite.

Con Karpenter, puede definir NodePools con restricciones el aprovisionamiento de nodos, como las restricciones, las etiquetas, los requisitos (tipos de instancias, zonas, etc.) y los límites del total de recursos aprovisionados. Al implementar cargas de trabajo, puede especificar varias restricciones de programación en las especificaciones del módulo, como las requests/limits, node selectors, node/pod

afinidades de los recursos, las tolerancias y las restricciones de distribución de la topología. Luego, Karpenter aprovisionará nodos del tamaño correcto en función de estas especificaciones.

Razones para usar Karpenter

Antes del lanzamiento de Karpenter, los usuarios de Kubernetes dependían principalmente de los grupos de Amazon EC2 Auto Scaling y del escalador automático de clústeres (CAS) de Kubernetes para ajustar dinámicamente la capacidad de procesamiento de sus clústeres. Con Karpenter, no es necesario crear docenas de grupos de nodos para lograr la flexibilidad y la diversidad que ofrece Karpenter. A diferencia de CAS, Karpenter no está tan estrechamente vinculado a las versiones de Kubernetes y no requiere que cambies entre AWS y Kubernetes. APIs

Karpenter consolida las responsabilidades de organización de instancias en un único sistema, que es más simple, estable y compatible con los clústeres. Karpenter se diseñó para superar algunos de los desafíos que presentaba Cluster Autoscaler al proporcionar formas simplificadas de:

- Aprovisione los nodos en función de los requisitos de carga de trabajo.
- Cree diversas configuraciones de nodos por tipo de instancia mediante NodePool opciones flexibles. En lugar de administrar muchos grupos de nodos personalizados específicos, Karpenter podría permitirle administrar diversas capacidades de carga de trabajo con una sola carga de trabajo flexible. NodePool
- Logre una mejor programación de los pods a escala mediante el lanzamiento rápido de nodos y la programación de los pods.

Para obtener información y documentación sobre el uso de Karpenter, visite el sitio karpenter.sh.

Recomendaciones

Las mejores prácticas se dividen en secciones sobre Karpenter en sí y sobre la programación de módulos NodePools.

Mejores prácticas de Karpenter

Las siguientes mejores prácticas abarcan temas relacionados con el propio Karpenter.

Concéntrese AMIs en los clústeres de producción

Le recomendamos encarecidamente que fije las conocidas Amazon Machine Images (AMIs) utilizadas por Karpenter para los clústeres de producción. Si se utiliza `amiSelector` un alias

definido como`@latest`, o si se utiliza algún otro método que dé como resultado una implementación que no se haya probado a AMIs medida que se publiquen, se corre el riesgo de que se produzcan fallos en la carga de trabajo y se produzcan tiempos de inactividad en los clústeres de producción. Por ello, recomendamos encarecidamente fijar las versiones probadas que funcionen AMIs para sus clústeres de producción y, al mismo tiempo, probar las versiones más recientes en clústeres que no sean de producción. Por ejemplo, puedes establecer un alias en tu cuenta de la NodeClass siguiente manera:

```
amiSelectorTerms
  - alias: al2023@v20240807
```

Para obtener información sobre cómo administrar y fijar datos AMIs en Karpenter, consulte Administración AMIs en la documentación de Karpenter.

Utilice Karpenter para cargas de trabajo con necesidades de capacidad cambiantes

Karpenter acerca la gestión del escalado a la versión nativa de Kubernetes que a los grupos de escalado automático () y a los grupos APIs de nodos gestionados (). ASGs MNGs ASGs y MNGs son abstracciones nativas de AWS en las que el escalado se activa en función de las métricas de nivel de AWS, como EC2 la carga de la CPU. El escalador automático de clústeres une las abstracciones de Kubernetes con las abstracciones de AWS, pero pierde algo de flexibilidad debido a eso, por ejemplo, la programación para una zona de disponibilidad específica.

Karpenter elimina una capa de abstracción de AWS para aportar parte de la flexibilidad directamente a Kubernetes. Karpenter se utiliza mejor para clústeres con cargas de trabajo que se enfrentan a períodos de alta demanda o que presentan diversos requisitos de procesamiento. MNGs y ASGs son ideales para clústeres que ejecutan cargas de trabajo que tienden a ser más estáticas y consistentes. Puede utilizar una combinación de nodos gestionados de forma dinámica y estática, en función de sus necesidades.

Considere otros proyectos de escalado automático cuando...

Necesita funciones que aún se estén desarrollando en Karpenter. Debido a que Karpenter es un proyecto relativamente nuevo, considere la posibilidad de realizar otros proyectos de escalado automático por el momento si necesita funciones que aún no forman parte de Karpenter.

Ejecute el controlador Karpenter en EKS Fargate o en un nodo de trabajo que pertenezca a un grupo de nodos.

Karpenter se instala mediante un gráfico de Helm. El diagrama de Helm instala el controlador Karpenter y un módulo de webhook como una implementación que debe ejecutarse antes de poder utilizar el controlador para escalar el clúster. Recomendamos un grupo de nodos pequeño como mínimo con al menos un nodo trabajador. Como alternativa, puedes ejecutar estos pods en EKS Fargate creando un perfil de Fargate para el espacio de nombres. `karpenter` Si lo hace, todos los pods desplegados en este espacio de nombres se ejecutarán en EKS Fargate. No ejecute Karpenter en un nodo gestionado por Karpenter.

Karpenter no admite plantillas de lanzamiento personalizadas

La versión 1 no admite plantillas de lanzamiento personalizadas. APIs Puede utilizar los datos de usuario personalizados and/or directamente especificando la AMIs personalización en el EC2NodeClass. Encontrará más información sobre cómo hacerlo en NodeClasses.

Excluya los tipos de instancias que no se ajusten a su carga de trabajo

Considera la posibilidad de excluir tipos de instancias específicos con la `node.kubernetes.io/instance-type` clave si las cargas de trabajo que se ejecutan en tu clúster no los requieren.

En el siguiente ejemplo, se muestra cómo evitar el aprovisionamiento de instancias de Graviton de gran tamaño.

```
- key: node.kubernetes.io/instance-type
  operator: NotIn
  values:
  - m6g.16xlarge
  - m6gd.16xlarge
  - r6g.16xlarge
  - r6gd.16xlarge
  - c6g.16xlarge
```

Habilite la gestión de interrupciones al usar Spot

Karpenter admite la gestión nativa de las interrupciones y puede gestionar los eventos de interrupción involuntarios, como las interrupciones de instancias puntuales, los eventos de mantenimiento programado y los eventos de terminación o detención de instancias que podrían

interrumpir sus cargas de trabajo. Cuando Karpenter detecta este tipo de eventos en los nodos, contamina, agota y cierra automáticamente los nodos afectados con antelación para empezar a limpiar las cargas de trabajo sin problemas antes de que se produzcan interrupciones. En el caso de las interrupciones detectadas con 2 minutos de antelación, Karpenter inicia rápidamente un nuevo nodo para poder mover los pods antes de recuperar la instancia. Para habilitar la gestión de interrupciones, configure el argumento `--interruption-queue` CLI con el nombre de la cola de SQS aprovisionada para este fin. No se recomienda utilizar el sistema de gestión de interrupciones de Karpenter junto con el gestor de terminaciones de nodos, tal y como se explica aquí.

Los módulos que requieran puntos de control u otras formas de drenaje correcto (es decir, 2 minutos antes de apagarlos) deberían permitir a Karpenter gestionar las interrupciones en sus clústeres.

Clúster privado Amazon EKS sin acceso a Internet saliente

Al aprovisionar un clúster de EKS en una VPC sin ruta a Internet, debe asegurarse de haber configurado su entorno de acuerdo con los requisitos de clúster privado que aparecen en la documentación de EKS. Además, debe asegurarse de haber creado un punto de enlace regional de VPC de STS en su VPC. De lo contrario, verá errores similares a los que aparecen a continuación.

```
{"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"}
```

Estos cambios son necesarios en un clúster privado porque el controlador de Karpenter usa las funciones de IAM para las cuentas de servicio (IRSA). Los pods configurados con IRSA adquieren credenciales mediante una llamada a la API AWS Security Token Service (AWS STS). Si no hay acceso saliente a Internet, debe crear y usar un punto de enlace de VPC de AWS STS en su VPC.

Los clústeres privados también requieren que crees un punto de conexión de VPC para SSM. Cuando Karpenter intenta aprovisionar un nodo nuevo, consulta las configuraciones de la plantilla de Launch y un parámetro de SSM. Si no tiene un punto final de VPC de SSM en su VPC, se producirá el siguiente error:

```
{"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"}
```

No hay ningún punto final de VPC para la API de consulta de listas de precios. Como resultado, los datos de precios se quedarán obsoletos con el paso del tiempo. Karpenter soluciona este problema al incluir los datos de precios bajo demanda en su binario, pero solo actualiza esos datos cuando Karpenter se actualiza. Las solicitudes fallidas de datos de precios generarán los siguientes mensajes de error:

```
{"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"}
```

Consulte esta documentación para usar Karpenter en clústeres EKS completamente privados y para saber qué puntos finales de VPC se deben crear.

Creando NodePools

Las siguientes prácticas recomendadas abarcan temas relacionados con la creación NodePools.

Crea varios NodePools cuando...

Cuando distintos equipos compartan un clúster y necesiten ejecutar sus cargas de trabajo en distintos nodos de trabajo o tengan distintos requisitos de sistema operativo o tipo de instancia, cree varios NodePools. Por ejemplo, un equipo puede querer usar Bottlerocket, mientras que otro puede querer usar Amazon Linux. Del mismo modo, un equipo podría tener acceso a un hardware de GPU caro que otro equipo no necesitaría. El uso de varias unidades NodePools garantiza que cada equipo disponga de los recursos más adecuados.

Cree NodePools que sean mutuamente excluyentes o ponderados

Se recomienda crear programas NodePools que se excluyan mutuamente o que estén ponderados para proporcionar un comportamiento de programación coherente. Si no lo son y varios NodePools

coinciden, Karpenter elegirá aleatoriamente cuál usar, lo que provocará resultados inesperados. Entre los ejemplos útiles para crear varios se NodePools incluyen los siguientes:

Crear una NodePool con GPU y permitir que solo se ejecuten cargas de trabajo especiales en estos nodos (caros):

```
# 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"
```

Despliegue con tolerancia a la contaminación:

```
# 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"
```

En el caso de un despliegue general para otro equipo, la NodePool especificación podría incluir NodeAffinity. Entonces, un despliegue podría ser igual. 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
```

Despliegue mediante 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"]
```

Utilice temporizadores (TTL) para eliminar automáticamente los nodos del clúster

Puede usar temporizadores en los nodos aprovisionados para establecer cuándo eliminar los nodos que no tienen módulos de carga de trabajo o que han alcanzado su fecha de caducidad. La caducidad de los nodos se puede utilizar como medio de actualización, de modo que los

nodos se retiren y se sustituyan por versiones actualizadas. Consulte la sección Vencimiento en la documentación de Karpenter para obtener información sobre cómo configurar la caducidad de `spec.template.spec` los nodos.

Evite restringir demasiado los tipos de instancias que Karpenter puede aprovisionar, especialmente cuando utilice Spot

Al utilizar Spot, Karpenter utiliza la estrategia de asignación optimizada de precio y capacidad para aprovisionar las instancias. EC2 Esta estrategia indica EC2 que hay que aprovisionar las instancias de los grupos más profundos según el número de instancias que vaya a lanzar y que tengan el menor riesgo de interrupción. EC2 A continuación, Fleet solicita instancias puntuales de los grupos con el precio más bajo. Cuantos más tipos de instancias permita que utilice Karpenter, mejor EC2 podrá optimizar el tiempo de ejecución de su instancia puntual. De forma predeterminada, Karpenter utilizará todos los tipos de instancias que EC2 ofrezca la región y las zonas de disponibilidad en las que esté desplegado el clúster. Karpenter selecciona de forma inteligente entre todos los tipos de instancias en función de los pods pendientes para asegurarse de que los pods estén programados en instancias del tamaño y el equipamiento adecuados. Por ejemplo, si su pod no requiere una GPU, Karpenter no programará su pod para un tipo de instancia que admita una EC2 GPU. Cuando no estés seguro de qué tipos de instancias usar, puedes ejecutar Amazon ec2-instance-selector para generar una lista de tipos de instancias que coincidan con tus requisitos de procesamiento. Por ejemplo, la CLI toma la memoria, la vCPU, la arquitectura y la región como parámetros de entrada y le proporciona una lista de EC2 instancias que cumplen con esas restricciones.

```
$ 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
```

No debes imponer demasiadas restricciones a Karpenter cuando utilices instancias puntuales, ya que hacerlo puede afectar a la disponibilidad de tus aplicaciones. Supongamos, por ejemplo, que se recuperan todas las instancias de un tipo concreto y no hay alternativas adecuadas disponibles para sustituirlas. Los pods permanecerán en estado pendiente hasta que se reponga la capacidad puntual para los tipos de instancias configurados. Puedes reducir el riesgo de que se produzcan errores de

capacidad insuficiente distribuyendo las instancias entre distintas zonas de disponibilidad, ya que los grupos de puntos son diferentes entre AZs sí. Dicho esto, la mejor práctica general es permitir que Karpenter utilice un conjunto diverso de tipos de instancias al usar Spot.

¿Programar pods?

Las siguientes prácticas recomendadas se refieren a la implementación de pods en un clúster mediante Karpenter para el aprovisionamiento de nodos.

Siga las prácticas recomendadas de EKS para obtener una alta disponibilidad

Si necesita ejecutar aplicaciones de alta disponibilidad, siga las recomendaciones generales de mejores prácticas de EKS. Consulte Topology Spread en la documentación de Karpenter para obtener detalles sobre cómo distribuir los pods entre nodos y zonas. Usa los presupuestos de disrupción para establecer el número mínimo de pods disponibles que deben mantenerse, en caso de que se intente desalojar o eliminar los pods.

Utilice restricciones en capas para restringir las funciones informáticas disponibles en su proveedor de servicios en la nube

El modelo de restricciones en capas de Karpenter le permite crear un conjunto complejo de restricciones de despliegue de módulos NodePool y módulos para obtener las mejores adaptaciones posibles a la programación de los módulos. Entre los ejemplos de restricciones que puede solicitar una especificación de pod se incluyen los siguientes:

- Debe ejecutarse en zonas de disponibilidad en las que solo estén disponibles determinadas aplicaciones. Supongamos, por ejemplo, que tiene un pod que tiene que comunicarse con otra aplicación que se ejecuta en una EC2 instancia que reside en una zona de disponibilidad concreta. Si su objetivo es reducir el tráfico entre zonas de disponibilidad en su VPC, puede que desee ubicar los pods en la zona de disponibilidad donde se encuentra EC2 la instancia. Este tipo de segmentación suele lograrse mediante selectores de nodos. Para obtener información adicional sobre los selectores de nodos, consulta la documentación de Kubernetes.
- Requiere ciertos tipos de procesadores u otro hardware. Consulta la sección sobre aceleradores de los documentos de Karpenter para ver un ejemplo de especificaciones de un pod que requiere que el pod funcione en una GPU.

Cree alarmas de facturación para monitorizar su gasto informático

Cuando configure su clúster para que escale automáticamente, debe crear alarmas de facturación que le avisen cuando su gasto haya superado un umbral y añadir límites de recursos a su configuración de Karpenter. Establecer límites de recursos con Karpenter es similar a establecer la capacidad máxima de un grupo de escalado automático de AWS, ya que representa la cantidad máxima de recursos informáticos que puede instanciar un Karpenter. NodePool

> **Note**
>
> No es posible establecer un límite global para todo el clúster. Los límites se aplican a determinados grupos NodePools.

El siguiente fragmento le indica a Karpenter que solo aprovisione un máximo de 1000 núcleos de CPU y 1000 Gi de memoria. Karpenter dejará de añadir capacidad solo cuando se alcance o supere el límite. Cuando se supera un límite, el controlador Karpenter `memory resource usage of 1001 exceeds limit of 1000` escribirá un mensaje similar en los registros del controlador. Si estás redireccionando los registros de tu contenedor a CloudWatch registros, puedes crear un filtro de métricas para buscar patrones o términos específicos en tus registros y, a continuación, crear una CloudWatchalarma que te avise cuando se supere el umbral de métricas configurado.

Para obtener más información sobre el uso de límites con Karpenter, consulta Cómo establecer límites de recursos en la documentación de Karpenter.

```
spec:
  limits:
    cpu: 1000
    memory: 1000Gi
```

Si no utilizas límites ni restringes los tipos de instancias que Karpenter puede aprovisionar, Karpenter seguirá añadiendo capacidad de cómputo a tu clúster según sea necesario. Si bien configurar Karpenter de esta manera permite que el clúster se escale libremente, también puede tener importantes implicaciones en cuanto a los costos. Por este motivo, recomendamos configurar las alarmas de facturación. Las alarmas de facturación te permiten recibir alertas y notificaciones proactivas cuando los cargos estimados calculados en tu (s) cuenta (s) superen un umbral definido. Consulta Cómo configurar una alarma de CloudWatch facturación de Amazon para supervisar de forma proactiva los cargos estimados para obtener más información.

Es posible que también desee habilitar la detección de anomalías en los costos, que es una función de administración de costos de AWS que utiliza el aprendizaje automático para monitorear continuamente sus costos y su uso a fin de detectar gastos inusuales. Puede encontrar más información en la guía de introducción a AWS Cost Anomaly Detection. Si ha llegado a crear un presupuesto en AWS Budgets, también puede configurar una acción para que le notifique cuando se supere un umbral específico. Con las medidas presupuestarias, puede enviar un correo electrónico, publicar un mensaje en un tema de SNS o enviar un mensaje a un chatbot como Slack. Para obtener más información, consulte Configuración de las acciones de AWS Budgets.

Utilice la do-not-disrupt anotación karpenter.sh/ para evitar que Karpenter desaprovisione un nodo

Si ejecutas una aplicación crítica en un nodo aprovisionado por Karpenter, como un trabajo por lotes de larga duración o una aplicación con estado, y el TTL del nodo ha caducado, la aplicación se interrumpirá cuando se cierre la instancia. Al añadir una `karpenter.sh/do-not-disrupt` anotación al pod, le indica a Karpenter que conserve el nodo hasta que se cierre el pod o se elimine la anotación. `karpenter.sh/do-not-disrupt` Consulte la documentación de distribución para obtener más información.

Si los únicos módulos que no son de Daemonset que quedan en un nodo son los que están asociados a las tareas, Karpenter puede atacar y terminar esos nodos siempre que el estado de la tarea sea correcto o fallido.

Configure requests=limits para todos los recursos que no sean de CPU cuando utilice la consolidación

La consolidación y la programación, en general, funcionan comparando las solicitudes de recursos de los pods con la cantidad de recursos asignables en un nodo. No se tienen en cuenta los límites de recursos. Por ejemplo, los pods que tienen un límite de memoria superior a la solicitud de memoria pueden superar la solicitud. Si varios pods del mismo nodo se rompen al mismo tiempo, es posible que algunos de ellos se cierren debido a una falta de memoria (OOM). La consolidación puede aumentar las probabilidades de que esto ocurra, ya que se suele empaquetar los pods en los nodos teniendo en cuenta únicamente sus solicitudes.

Se utiliza LimitRanges para configurar los valores predeterminados de las solicitudes y los límites de recursos

Como Kubernetes no establece solicitudes ni límites predeterminados, el consumo de recursos del host, la CPU y la memoria subyacentes por parte de un contenedor no está limitado. El programador

de Kubernetes analiza el total de solicitudes de un pod (cuanto mayor sea el total de solicitudes de los contenedores del pod o el total de recursos de los contenedores de inicio del pod) para determinar en qué nodo de trabajo programar el pod. Del mismo modo, Karpenter considera las solicitudes de un pod para determinar qué tipo de instancia aprovisiona. Puedes usar un rango de límites para aplicar un valor predeterminado razonable a un espacio de nombres, en caso de que algunos pods no especifiquen las solicitudes de recursos.

Consulte Configurar las solicitudes y los límites de memoria predeterminados para un espacio de nombres

Aplique solicitudes de recursos precisas a todas las cargas de trabajo

Karpenter puede lanzar los nodos que mejor se adapten a sus cargas de trabajo cuando la información sobre sus requisitos de carga de trabajo es precisa. Esto es particularmente importante si se utiliza la función de consolidación de Karpenter.

Consulte Configurar y dimensionar los recursos Requests/Limits para todas las cargas de trabajo

Recomendaciones de CoredNS

Actualice la configuración de CoredNS para mantener la confiabilidad

Al implementar pods de CoredNS en nodos gestionados por Karpenter, dada la naturaleza dinámica de Karpenter en nodos nuevos terminating/creating rápidamente para adaptarse a la demanda, es recomendable seguir las siguientes prácticas recomendadas:

Duración lamentable de CoredNS

Sonda de preparación de CoredNS

Esto garantizará que las consultas de DNS no se dirijan a un pod de CoreDNS que aún no esté listo o que haya finalizado.

Planos de Karpenter

Dado que Karpenter adopta un enfoque centrado en las aplicaciones para aprovisionar la capacidad de cómputo en el plano de datos de Kubernetes, hay situaciones de carga de trabajo habituales en las que quizás se pregunte cómo configurarlas correctamente. Karpenter Blueprints es un repositorio que incluye una lista de escenarios de carga de trabajo comunes siguiendo las mejores prácticas

que se describen aquí. Dispondrá de todos los recursos que necesita incluso para crear un clúster de EKS con Karpenter configurado y probar cada uno de los planos incluidos en el repositorio. Puede combinar diferentes planos para crear finalmente el que necesite para sus cargas de trabajo.

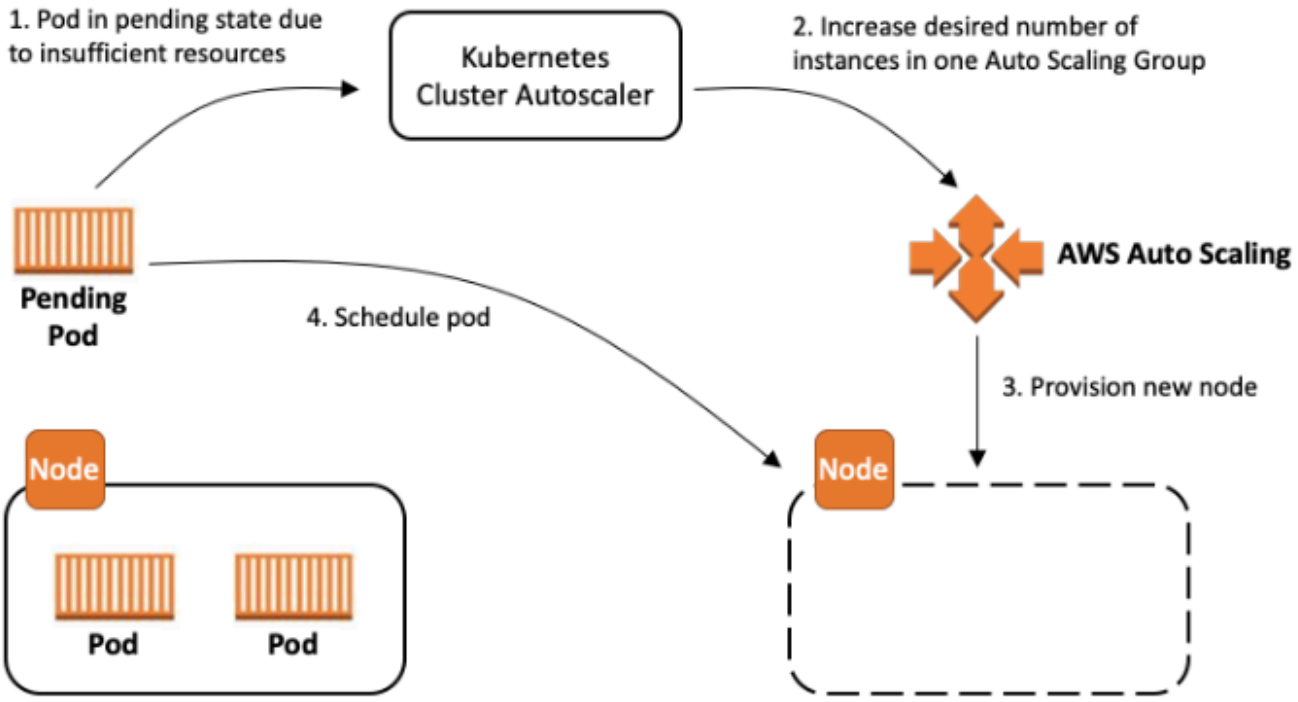
Recursos adicionales

- Taller de un día de inmersión en Karpenter
- Taller de optimización de costos de Karpenter
- Taller de EKS - Karpenter
- Karpenter contra Cluster Autoscaler
- Sesión de Karpenter en re:Invent 2023
- Tutorial: Ejecute clústeres de Kubernetes por menos dinero con Amazon EC2 Spot y Karpenter

Escalador automático de clústeres

Descripción general de

El escalador automático de clústeres de Kubernetes es una popular solución de escalado automático de clústeres mantenida por SIG Autoscaler. Es responsable de garantizar que el clúster tenga suficientes nodos para programar los pods sin desperdiciar recursos. Presta atención a los pods que no se programan y a los nodos que están infrautilizados. A continuación, simula la adición o eliminación de nodos antes de aplicar el cambio al clúster. La implementación de AWS Cloud Provider en Cluster AutoScaler controla el `.DesiredReplicas` campo de sus grupos de EC2 Auto Scaling.



Esta guía proporcionará un modelo mental para configurar el escalador automático de clústeres y elegir el mejor conjunto de ventajas y desventajas para cumplir con los requisitos de su organización. Si bien no existe una configuración única que sea la mejor, hay un conjunto de opciones de configuración que le permiten equilibrar el rendimiento, la escalabilidad, el costo y la disponibilidad. Además, en esta guía se ofrecen consejos y prácticas recomendadas para optimizar la configuración de AWS.

Glosario

La siguiente terminología se utilizará con frecuencia a lo largo de este documento. Estos términos pueden tener un significado amplio, pero se limitan a las siguientes definiciones para los fines de este documento.

La escalabilidad se refiere al rendimiento del escalador automático de clústeres a medida que el clúster de Kubernetes aumenta en número de pods y nodos. A medida que se alcanzan los límites de escalabilidad, el rendimiento y la funcionalidad del escalador automático de clústeres se degradan. A medida que el escalador automático de clústeres supere sus límites de escalabilidad, ya no podrá añadir ni eliminar nodos del clúster.

El rendimiento se refiere a la rapidez con la que el escalador automático de clústeres puede tomar y ejecutar decisiones de escalado. Un escalador automático de clústeres que funcione perfectamente

tomaría una decisión al instante y activaría una acción de escalado en respuesta a estímulos, como la imposibilidad de programar un módulo.

La disponibilidad significa que los módulos se pueden programar rápidamente y sin interrupciones. Esto incluye cuándo es necesario programar los pods recién creados y cuándo un nodo reducido cierra los pods restantes programados para él.

El costo viene determinado por la decisión que hay detrás de la ampliación y la escalabilidad en función de los eventos. Los recursos se desperdician si un nodo existente está infrautilizado o si se agrega un nodo nuevo que es demasiado grande para los pods entrantes. Según el caso de uso, la cancelación prematura de los módulos puede conllevar costes debido a una decisión agresiva de reducción.

Los grupos de nodos son un concepto abstracto de Kubernetes para un grupo de nodos dentro de un clúster. No es un verdadero recurso de Kubernetes, sino que existe como una abstracción en el escalador automático de clústeres, la API de clústeres y otros componentes. Los nodos de un grupo de nodos comparten propiedades, como etiquetas y defectos, pero pueden constar de varias zonas de disponibilidad o tipos de instancias.

EC2 Los grupos de Auto Scaling se pueden utilizar como una implementación de los grupos de nodos en EC2. EC2 Los grupos de Auto Scaling están configurados para lanzar instancias que se unen automáticamente a sus clústeres de Kubernetes y aplican etiquetas y contaminantes a su recurso de nodo correspondiente en la API de Kubernetes.

EC2 Los grupos de nodos gestionados son otra implementación de los grupos de nodos existentes. EC2 Eliminan la complejidad al configurar manualmente los grupos de EC2 escalado automático y proporcionan funciones de administración adicionales, como la actualización de la versión de los nodos y la terminación sencilla de los nodos.

Uso del escalador automático de clústeres

Normalmente, el Cluster Autoscaler se instala como una Implementación en su clúster. Utiliza la elección del líder para garantizar una alta disponibilidad, pero el trabajo lo realiza una sola réplica a la vez. No es escalable horizontalmente. Para las configuraciones básicas, la configuración predeterminada debería funcionar de inmediato siguiendo las instrucciones de instalación proporcionadas, pero hay algunas cosas a tener en cuenta.

Asegúrese de que:

- La versión del escalador automático de clústeres coincide con la versión del clúster. La compatibilidad entre versiones no está probada ni admitida.
- La detección automática está habilitada, a menos que haya casos de uso avanzados específicos que impidan el uso de este modo.

Emplee el acceso menos privilegiado a la función de IAM

Cuando utilice el Auto Discovery, le recomendamos encarecidamente que emplee el acceso con privilegios mínimos limitando las acciones `autoscaling:SetDesiredCapacity` y `autoscaling:TerminateInstanceInAutoScalingGroup` los grupos de Auto Scaling que están dentro del clúster actual.

Esto evitará que un escalador automático de clústeres que se ejecute en un clúster modifique los grupos de nodos de otro clúster, incluso si el `--node-group-auto-discovery` argumento no se ha limitado a los grupos de nodos del clúster mediante etiquetas (por ejemplo). `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": "*"
        }
    ]
}
```

Configuración de los grupos de nodos

Un escalado automático eficaz comienza con la configuración correcta de un conjunto de grupos de nodos para el clúster. Seleccionar el conjunto correcto de grupos de nodos es clave para maximizar la disponibilidad y reducir los costes de las cargas de trabajo. AWS implementa grupos de nodos mediante grupos de EC2 Auto Scaling, que son flexibles para un gran número de casos de uso. Sin embargo, el escalador automático de clústeres hace algunas suposiciones sobre los grupos de nodos. Mantener la coherencia de las configuraciones de EC2 Auto Scaling Group con estas suposiciones minimizará el comportamiento no deseado.

Asegúrese de que:

- Cada nodo de un grupo de nodos tiene propiedades de programación idénticas, como etiquetas, manchas y recursos.
  - Por MixedInstancePolicies lo tanto, los tipos de instancia deben tener la misma forma para la CPU, la memoria y la GPU
  - El primer tipo de instancia especificado en la política se utilizará para simular la programación.
  - Si su política tiene tipos de instancias adicionales con más recursos, es posible que los recursos se desperdicien después de ampliarlos.
  - Si su política tiene tipos de instancias adicionales con menos recursos, es posible que los pods no puedan programar las instancias.
- Se prefieren los grupos de nodos con muchos nodos a los grupos de nodos con menos nodos. Esto tendrá el mayor impacto en la escalabilidad.
- Siempre que sea posible, prefiera EC2 las funciones cuando ambos sistemas sean compatibles (por ejemplo, MixedInstancePolicy las regiones)

> **Note**
>
> Recomendamos utilizar grupos de nodos gestionados por EKS. Los grupos de nodos gestionados vienen con potentes funciones de administración, que incluyen funciones para Cluster AutoScaler, como la detección automática de grupos de EC2 Auto Scaling y la elegante terminación de nodos.

Optimización del rendimiento y la escalabilidad

Comprender la complejidad del tiempo de ejecución del algoritmo de escalado automático le ayudará a ajustar el escalador automático de clústeres para que siga funcionando sin problemas en clústeres grandes con más de 1000 nodos.

Los principales requisitos para ajustar la escalabilidad del escalador automático de clústeres son los recursos que se proporcionan al proceso, el intervalo de escaneo del algoritmo y el número de grupos de nodos del clúster. Hay otros factores que intervienen en la verdadera complejidad del tiempo de ejecución de este algoritmo, como la complejidad de los complementos de programación y el número de pods. Se consideran parámetros no configurables, ya que son inherentes a la carga de trabajo del clúster y no se pueden ajustar fácilmente.

El escalador automático de clústeres carga todo el estado del clúster en la memoria, incluidos los pods, los nodos y los grupos de nodos. En cada intervalo de escaneo, el algoritmo identifica los módulos no programables y simula la programación para cada grupo de nodos. El ajuste de estos factores conlleva diferentes ventajas y desventajas que se deben considerar cuidadosamente para cada caso de uso.

Escalado automático vertical del escalador automático del clúster

La forma más sencilla de escalar el escalador automático de clústeres a clústeres más grandes es aumentar las solicitudes de recursos para su implementación. Tanto la memoria como la CPU deben aumentarse para los clústeres grandes, aunque esto varía considerablemente según el tamaño del clúster. El algoritmo de escalado automático almacena todos los pods y nodos en la memoria, lo que puede provocar un consumo de memoria superior a un gigabyte en algunos casos. Por lo general, el aumento de los recursos se realiza de forma manual. Si considera que el ajuste constante de los recursos supone una carga operativa, considere la posibilidad de utilizar el Addon Resizer o el escalador automático Vertical Pod.

Reducir el número de grupos de nodos

Minimizar la cantidad de grupos de nodos es una forma de garantizar que el escalador automático de clústeres siga funcionando bien en clústeres grandes. Esto puede resultar difícil para algunas organizaciones que estructuran sus grupos de nodos por equipo o por aplicación. Si bien esto es totalmente compatible con la API de Kubernetes, se considera que va en contra del patrón del escalador automático de clústeres, lo que repercute en la escalabilidad. Existen muchas razones para usar varios grupos de nodos (por ejemplo, Spot o GPUs), pero en muchos casos existen diseños alternativos que logran el mismo efecto y utilizan un número reducido de grupos.

Asegúrese de que:

- El aislamiento de los pods se realiza mediante espacios de nombres en lugar de grupos de nodos.
  - Es posible que esto no sea posible en clústeres multiusuario de baja confianza.
  - El pod ResourceRequests y el pod ResourceLimits están configurados correctamente para evitar la contención de recursos.
  - Los tipos de instancias más grandes darán como resultado un empaquetado de contenedores más óptimo y reducirán la sobrecarga de los módulos del sistema.
- NodeTaints o se NodeSelectors utilizan para programar los pods como excepción, no como regla.
- Los recursos regionales se definen como un único grupo de EC2 Auto Scaling con múltiples zonas de disponibilidad.

Reducir el intervalo de escaneo

Un intervalo de escaneo bajo (por ejemplo, 10 segundos) garantizará que el escalador automático del clúster responda lo más rápido posible cuando los módulos no se puedan programar. Sin embargo, cada escaneo genera muchas llamadas de API a la API de Kubernetes y a EC2 Auto Scaling Group o EKS Managed Node Group. APIs Estas llamadas a la API pueden provocar una limitación de la velocidad o incluso la falta de disponibilidad del servicio para su plano de control de Kubernetes.

El intervalo de escaneo predeterminado es de 10 segundos, pero en AWS, el lanzamiento de un nodo tarda mucho más en lanzar una nueva instancia. Esto significa que es posible aumentar el intervalo sin aumentar significativamente el tiempo de escalado vertical general. Por ejemplo, si se tarda 2 minutos en lanzar un nodo, si se cambia el intervalo a 1 minuto, se compensará con una reducción de 6 veces en las llamadas a la API y con ampliaciones un 38% más lentas.

Fragmentación entre grupos de nodos

El escalador automático de clústeres se puede configurar para que funcione en un conjunto específico de grupos de nodos. Con esta funcionalidad, es posible implementar varias instancias del escalador automático de clústeres, cada una configurada para funcionar en un conjunto diferente de grupos de nodos. Esta estrategia le permite utilizar un número arbitrario de grupos de nodos, lo que supone un coste menor que el de la escalabilidad. Solo recomendamos usar esto como último recurso para mejorar el rendimiento.

El escalador automático de clústeres no se diseñó originalmente para esta configuración, por lo que tiene algunos efectos secundarios. Como los fragmentos no se comunican, es posible que varios escaladores automáticos intenten programar un pod no programable. Esto puede provocar una ampliación innecesaria de varios grupos de nodos. Estos nodos adicionales volverán a reducirse después de`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
```

Asegúrese de que:

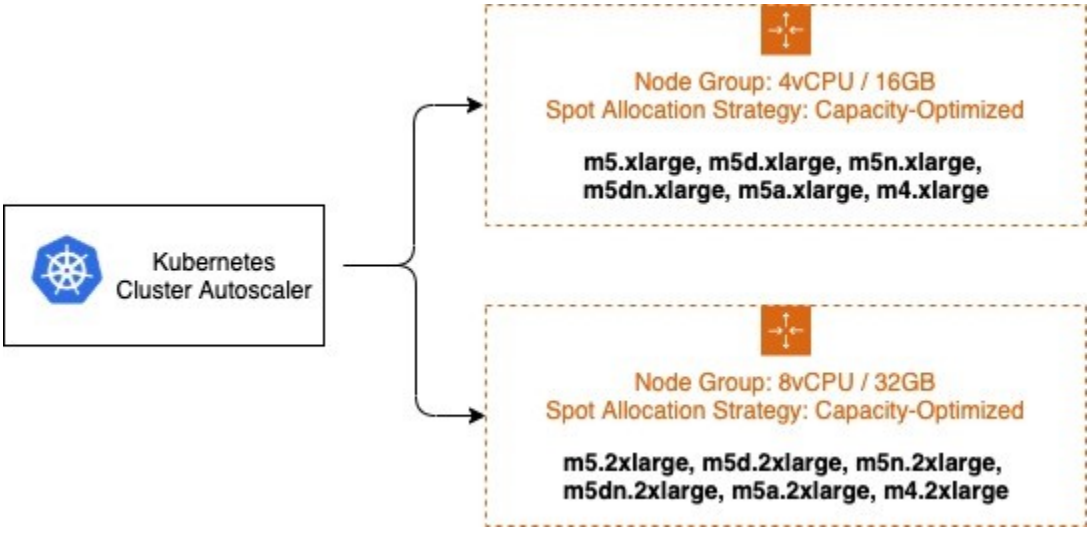
- Cada fragmento está configurado para apuntar a un conjunto único de grupos de EC2 Auto Scaling.
- Cada fragmento se despliega en un espacio de nombres diferente para evitar conflictos en las elecciones de los líderes

Optimización del costo y la disponibilidad

Spot Instances

Puede utilizar las instancias puntuales en sus grupos de nodos y ahorrar hasta un 90% con respecto al precio bajo demanda, con la ventaja de que las instancias puntuales pueden interrumpirse en cualquier momento cuando EC2 necesite recuperar la capacidad. Se producirán errores de capacidad insuficiente cuando su grupo de EC2 Auto Scaling no pueda ampliarse debido a la falta de capacidad disponible. Aprovechar al máximo la diversidad mediante la selección de varias familias de instancias puede aumentar sus probabilidades de alcanzar la escala deseada al aprovechar muchos grupos de capacidad puntuales y reducir el impacto de las interrupciones de las instancias puntuales en la disponibilidad del clúster. Las políticas de instancias mixtas con instancias de spot son una excelente manera de aumentar la diversidad sin aumentar el número de grupos de nodos. Tenga en cuenta que, si necesita recursos garantizados, utilice instancias bajo demanda en lugar de instancias puntuales.

Es fundamental que todos los tipos de instancias tengan una capacidad de recursos similar al configurar políticas de instancias mixtas. El simulador de programación del escalador automático utiliza el primero InstanceType del. MixedInstancePolicy Si los siguientes tipos de instancias son más grandes, es posible que se desperdicien recursos después de ampliarlos. Si son más pequeños, es posible que los pods no se programen en las nuevas instancias debido a una capacidad insuficiente. Por ejemplo, todas las instancias M4, M5, M5a y M5n tienen cantidades similares de CPU y memoria y son excelentes candidatas para una. MixedInstancePolicy La herramienta de selección de EC2 instancias puede ayudarte a identificar tipos de instancias similares.



Se recomienda aislar la capacidad bajo demanda y la puntual en grupos independientes de EC2 Auto Scaling. Es preferible utilizar esta opción en lugar de utilizar una estrategia de capacidad base, ya que las propiedades de programación son fundamentalmente diferentes. Como las instancias puntuales se interrumpen en cualquier momento (cuando es EC2 necesario recuperar la capacidad), los usuarios suelen contaminar sus nodos prevenibles, por lo que es necesario que el pod tolere explícitamente el comportamiento de prevención. Estas contaminaciones dan como resultado propiedades de programación diferentes para los nodos, por lo que deben separarse en varios grupos de EC2 Auto Scaling.

El escalador automático de clústeres tiene un concepto de expansores, que proporciona diferentes estrategias para seleccionar el grupo de nodos que se va a escalar. Esta estrategia `--expander=least-waste` es una buena opción de uso general y, si va a utilizar varios grupos de nodos para diversificar las instancias puntuales (como se describe en la imagen anterior), podría ayudar a optimizar aún más los costes de los grupos de nodos al escalar el grupo, lo que sería mejor utilizar después de la actividad de escalado.

Priorizar un grupo de nodos o un ASG

También puede configurar el escalado automático basado en prioridades mediante el expansor de prioridad. `--expander=priority`permite que su clúster priorice un grupo de nodos o un ASG y, si no puede escalarse por algún motivo, elegirá el siguiente grupo de nodos de la lista priorizada. Esto resulta útil en situaciones en las que, por ejemplo, quieres usar tipos de instancias P3 porque su GPU proporciona un rendimiento óptimo para tu carga de trabajo, pero como segunda opción también puedes usar tipos de instancias P2.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-autoscaler-priority-expander
  namespace: kube-system
data:
  priorities: |-
    10:
      - .*p2-node-group.*
    50:
      - .*p3-node-group.*
```

Cluster AutoScaler intentará escalar el grupo EC2 Auto Scaling que coincida con el nombre p3-node-group. Si esta operación no se realiza correctamente`--max-node-provision-time`,

intentará escalar un grupo de EC2 Auto Scaling que coincida con el nombre p2-node-group. Este valor predeterminado es 15 minutos y se puede reducir para seleccionar un grupo de nodos con mayor capacidad de respuesta, aunque si el valor es demasiado bajo, puede provocar escalamientos innecesarios.

Sobreaprovisionamiento

El escalador automático de clústeres minimiza los costos al garantizar que los nodos solo se agreguen al clúster cuando sea necesario y se eliminen cuando no se utilicen. Esto afecta considerablemente a la latencia de implementación, ya que muchos pods se ven obligados a esperar a que el nodo se amplíe antes de poder programarlos. Los nodos pueden tardar varios minutos en estar disponibles, lo que puede aumentar la latencia de programación de pods en un orden de magnitud.

Esto se puede mitigar al utilizar el sobreaprovisionamiento, que cambia el costo de la latencia de programación. El sobreaprovisionamiento se implementa mediante módulos temporales con prioridad negativa, que ocupan espacio en el clúster. Cuando los módulos recién creados no se puedan programar y tengan una prioridad más alta, se sustituirá por los módulos temporales para liberar espacio. A continuación, los módulos temporales dejan de poder programarse, lo que activa el escalador automático de clústeres para escalar los nuevos nodos sobreaprovisionados.

El sobreaprovisionamiento tiene otras ventajas menos obvias. Sin el sobreaprovisionamiento, uno de los efectos secundarios de un clúster muy utilizado es que los pods tomarán decisiones de programación menos óptimas si se sigue la `preferredDuringSchedulingIgnoredDuringExecution` regla de afinidad entre los nodos y los nodos. Un caso de uso común es separar los pods para una aplicación de alta disponibilidad en las distintas zonas de disponibilidad. AntiAffinity El sobreaprovisionamiento puede aumentar considerablemente las probabilidades de que un nodo de la zona correcta esté disponible.

La cantidad de capacidad sobreaprovisionada es una decisión empresarial cuidadosa para su organización. En esencia, se trata de una compensación entre rendimiento y coste. Una forma de tomar esta decisión consiste en determinar la frecuencia media de ampliación y dividirla entre el tiempo que se tarda en escalar un nodo nuevo. Por ejemplo, si en promedio se necesita un nodo nuevo cada 30 segundos y EC2 se tarda 30 segundos en aprovisionar un nodo nuevo, un solo nodo de sobreaprovisionamiento garantizará que siempre haya un nodo adicional disponible, lo que reducirá la latencia de programación en 30 segundos a costa de una sola instancia adicional EC2 . Para mejorar las decisiones de programación zonal, aprovisione en exceso un número de nodos igual al número de zonas de disponibilidad de su grupo de EC2 Auto Scaling para garantizar que el planificador pueda seleccionar la mejor zona para los pods entrantes.

Evite los desalojos a escala reducida

Algunas cargas de trabajo son costosas de expulsar. El análisis de macrodatos, las tareas de aprendizaje automático y las pruebas se completarán con el tiempo, pero deberán reiniciarse si se interrumpen. El escalador automático de clústeres intentará reducir la escala de cualquier nodo situado debajo del nodo scale-down-utilization-threshold, lo que interrumpirá los pods que queden en el nodo. Esto se puede evitar asegurándose de que los pods cuyo desalojo resulta caro estén protegidos por una etiqueta reconocida por el escalador automático de clústeres.

Asegúrese de que:

- Los módulos caros de desalojar tienen la anotación `cluster-autoscaler.kubernetes.io/safe-to-evict=false`

Casos de uso avanzados

Volúmenes de EBS

El almacenamiento persistente es fundamental para crear aplicaciones con estado, como bases de datos o cachés distribuidas. Los volúmenes de EBS permiten este caso de uso en Kubernetes, pero están limitados a una zona específica. Estas aplicaciones pueden tener una alta disponibilidad si se dividen en varias zonas AZs mediante un volumen de EBS independiente para cada zona de disponibilidad. El escalador automático de clústeres puede entonces equilibrar el escalado de los grupos de escalado automático. EC2

Asegúrese de que:

- El equilibrio del grupo de nodos se habilita al establecer `balance-similar-node-groups=true`.
- Los grupos de nodos se configuran con los mismos ajustes, excepto para zonas de disponibilidad y volúmenes de EBS diferentes.

Programación conjunta

Los trabajos de entrenamiento distribuidos de machine learning se benefician significativamente de la latencia minimizada de las configuraciones de nodos de la misma zona. Estas cargas de trabajo implementan varios módulos en una zona específica. Esto se puede lograr configurando

la afinidad de los pods para todos los pods coprogramados o bien utilizando la afinidad de nodos. `topologyKey: failure-domain.beta.kubernetes.io/zone` A continuación, el escalador automático de clústeres ampliará una zona específica para adaptarla a las demandas. Es posible que desee asignar varios grupos de EC2 Auto Scaling, uno por zona de disponibilidad, para permitir la conmutación por error para toda la carga de trabajo coprogramada.

Asegúrese de que:

- El equilibrio del grupo de nodos se habilita al establecer `balance-similar-node-groups=false`
- La afinidad de nodos o la preferencia de nodos se utilizan cuando los clústeres incluyen grupos de nodos regionales y zonales.
  - Usa la afinidad de nodos para forzar o fomentar que los grupos regionales eviten los grupos de nodos zonales y viceversa.
  - Si los pods zonales se programan en grupos de nodos regionales, esto provocará un desequilibrio en la capacidad de los pods regionales.
  - Si tus cargas de trabajo zonales toleran las interrupciones y la reubicación, configura Pod Preemption para permitir que los grupos a escala regional impongan la prevención y la reprogramación en una zona menos concurrida.

Aceleradores

Algunos clústeres utilizan aceleradores de hardware especializados, como la GPU. Al ampliarse, el complemento del dispositivo acelerador puede tardar varios minutos en anunciar el recurso en el clúster. El escalador automático del clúster ha simulado que este nodo tendrá el acelerador, pero hasta que el acelerador esté listo y actualice los recursos disponibles en el nodo, no se podrán programar los módulos pendientes en el nodo. Esto puede generar un escalado horizontal innecesario repetidas veces.

Además, no se considerará la posibilidad de reducir la capacidad de los nodos con aceleradores y un alto uso de la CPU o la memoria, incluso si el acelerador no se utiliza. Este comportamiento puede resultar caro debido al coste relativo de los aceleradores. En cambio, el escalador automático de clústeres puede aplicar reglas especiales para considerar la posibilidad de reducir la escala de los nodos si tienen aceleradores desocupados.

Para garantizar el comportamiento correcto en estos casos, puedes configurar el kubelet de los nodos del acelerador para etiquetar el nodo antes de que se una al clúster. El escalador automático

de clústeres utilizará este selector de etiquetas para activar el comportamiento optimizado del acelerador.

Asegúrese de que:

- El Kubelet para nodos de GPU está configurado con `--node-labels k8s.amazonaws.com/accelerator=$ACCELERATOR_TYPE`
- Los nodos con aceleradores se rigen por la misma regla de propiedades de programación indicada anteriormente.

Escalando desde 0

El escalador automático de clústeres es capaz de escalar los grupos de nodos desde y hacia cero, lo que puede generar importantes ahorros de costos. Detecta los recursos de CPU, memoria y GPU de un grupo de Auto Scaling al inspeccionar lo InstanceType especificado en su LaunchConfiguration o LaunchTemplate. Algunos pods requieren recursos adicionales, como `WindowsENI` o específicos, `PrivateIPv4Address` NodeSelectors o Taints, que no se pueden detectar en el. LaunchConfiguration El Auto Scaler de clústeres puede tener en cuenta estos factores descubriéndolos a partir de las etiquetas del Grupo EC2 Auto Scaling. Por ejemplo:

```
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**
>
> Tenga en cuenta que, al escalar a cero, su capacidad vuelve a estar disponible EC2 y es posible que no esté disponible en el futuro.

Parámetros adicionales

Existen muchas opciones de configuración que se pueden utilizar para ajustar el comportamiento y el rendimiento del Cluster Autoscaler. Encontrará una lista completa de parámetros en GitHub.

Parámetro	Description (Descripción)	Predeterminado
intervalo de escaneo	¿Con qué frecuencia se reevalúa el clúster para ampliarlo o reducirlo	10 segundos
max-empty-bulk-delete	Número máximo de nodos vacíos que se pueden eliminar al mismo tiempo.	10
scale-down-delay-after-añadir	¿Cuánto tiempo después de la ampliación se reanuda la evaluación de la reducción	10 minutos
scale-down-delay-after-eliminar	¿Cuánto tiempo después de la eliminación del nodo se reanuda la evaluación de escala reducida? El valor predeterminado es scan-inte rval	intervalo de escaneo
scale-down-delay-after-fallo	¿Cuánto tiempo después del fracaso de la reducción se reanuda la evaluación	3 minutos
scale-down-unneeded-time	¿Cuánto tiempo debería no ser necesario un nodo antes de que sea apto para la reducción	10 minutos
scale-down-unready-time	¿Cuánto tiempo debería no ser necesario un nodo que no esté preparado para ser apto para la reducción	20 minutos
scale-down-utilization-threshold	Nivel de utilización del nodo, definido como la suma de los recursos solicitados dividida	0,5

Parámetro	Description (Descripción)	Predeterminado
	por la capacidad, por debajo del cual se puede considera r la posibilidad de reducir la escala de un nodo	
scale-down-non-empty-los candidatos cuentan	Número máximo de nodos no vacíos considerados en una iteración como candidatos a ser reducidos con drenaje. Un valor más bajo significa una mejor capacidad de respuesta de la CA, pero es posible que la latencia de reducción sea más lenta. Un valor más alto puede afectar al rendimien to de la CA con clústeres grandes (cientos de nodos). Configúrelo en un valor no positivo para desactivar esta heurística; CA no limitará la cantidad de nodos que considera. »	30

Parámetro	Description (Descripción)	Predeterminado
scale-down-candidates-pool-proporción	Proporción de nodos que se consideran candidatos adicionales y no vacíos para la reducción cuando algunos candidatos de la iteración anterior ya no son válidos. Un valor más bajo significa una mejor capacidad de respuesta de CA, pero es posible que la latencia de reducción sea más lenta. Un valor más alto puede afectar al rendimien to de la CA con clústeres grandes (cientos de nodos). Si se establece en 1.0 para desactivar esta heurística, CA considerará todos los nodos como candidatos adicionales.	0.1
scale-down-candidates-poolC uenta de -minutos	Número mínimo de nodos que se consideran candidato s adicionales no vacíos para la reducción cuando algunos candidatos de la iteración anterior ya no son válidos. Al calcular el tamaño del grupo para los candidatos adicional es, tomamos `max(#nodes * scale-down-candida tes-pool-ratio, scale-down-candida tes-pool-min-count)`	50

Recursos adicionales

Esta página contiene una lista de presentaciones y demostraciones de Cluster AutoScaler. Si quieres añadir una presentación o una demostración aquí, envía una solicitud de cambios.

Presentación/demostración	Presentadores
Escalado automático y optimización de costes en Kubernetes: de 0 a 100	Guy Templeton, Skyscanner y Jiaxin Shan, Amazon
Análisis profundo de SIG con escalado automático	Maciek Pytel y Marcin Wielgus

Referencias

- 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/selector de instancias amazon-ec2
- https://github.com/aws/aws-node-termination-handler

Mejores prácticas de confiabilidad

Esta sección proporciona orientación sobre cómo hacer que las cargas de trabajo que se ejecutan en EKS sean resilientes y estén altamente disponibles

Cómo usar esta guía

Esta guía está destinada a desarrolladores y arquitectos que desean desarrollar y operar servicios de alta disponibilidad y tolerantes a fallos en EKS. La guía está organizada en diferentes áreas temáticas para facilitar su consumo. Cada tema comienza con una breve descripción, seguida de una lista de recomendaciones y mejores prácticas para garantizar la fiabilidad de los clústeres de EKS.

Introducción

Las prácticas recomendadas de fiabilidad de EKS se han agrupado en los siguientes temas:

- Aplicaciones
- Plano de control
- Plano de datos

¿Qué hace que un sistema sea fiable? Si un sistema puede funcionar de manera consistente y satisfacer las demandas a pesar de los cambios en su entorno durante un período de tiempo, se lo puede considerar confiable. Para lograrlo, el sistema debe detectar las fallas, corregirlas automáticamente y tener la capacidad de escalar en función de la demanda.

Los clientes pueden usar Kubernetes como base para operar aplicaciones y servicios de misión crítica de manera confiable. Pero además de incorporar principios de diseño de aplicaciones basadas en contenedores, ejecutar cargas de trabajo de manera confiable también requiere una infraestructura confiable. En Kubernetes, la infraestructura comprende el plano de control y el plano de datos.

EKS proporciona un plano de control Kubernetes apto para producción que está diseñado para ofrecer una alta disponibilidad y tolerancia a los fallos.

En EKS, AWS es responsable de la fiabilidad del plano de control de Kubernetes. EKS ejecuta el plano de control de Kubernetes en tres zonas de disponibilidad de una región de AWS. Administra

automáticamente la disponibilidad y la escalabilidad de los servidores API de Kubernetes y del clúster etcd.

La responsabilidad de la confiabilidad del plano de datos la comparten usted, el cliente y AWS. EKS ofrece cuatro opciones de nodos de trabajo para implementar el plano de datos de Kubernetes.

El modo automático de EKS, que es la opción más gestionada, gestiona el aprovisionamiento, el escalado y las actualizaciones del plano de datos, además de proporcionar capacidades gestionadas de computación, redes y almacenamiento. El modo automático AMIs se publica con frecuencia y los clústeres se actualizan automáticamente a la AMI más reciente para implementar correcciones y parches de seguridad de CVE. Puede controlar cuándo esto ocurre configurando los controles de interrupción en el modo automático. NodePools

Fargate gestiona el aprovisionamiento y el escalado del plano de datos mediante la ejecución de un pod por nodo. La tercera opción, los grupos de nodos gestionados, gestiona el aprovisionamiento y las actualizaciones del plano de datos. Por último, los nodos autogestionados son la opción menos gestionada para el plano de datos. Cuanto más plano de datos gestionado por AWS utilice, menos responsabilidad tendrá.

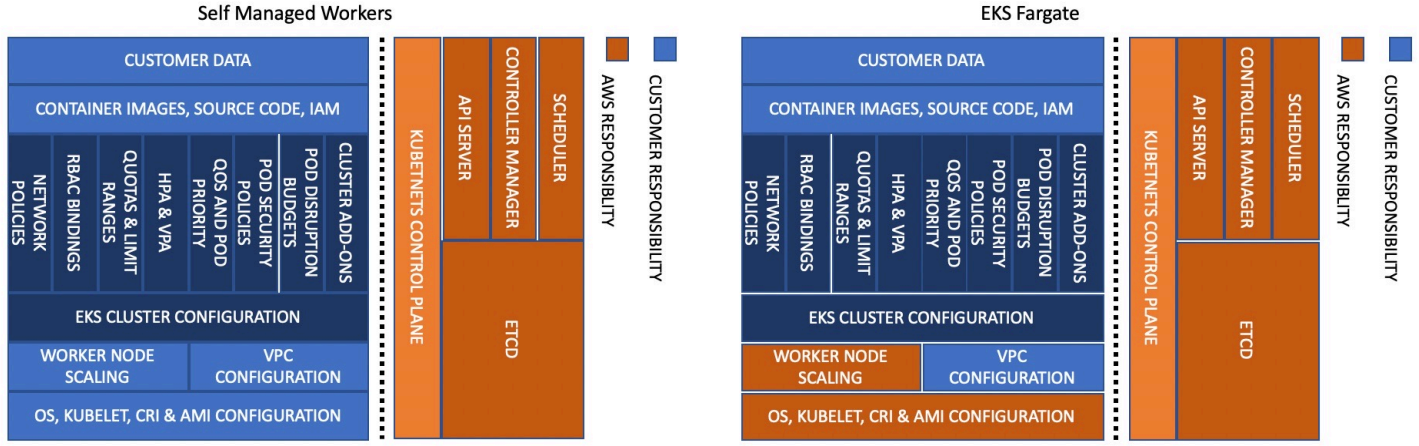
Los grupos de nodos administrados automatizan el aprovisionamiento y la administración del ciclo de vida de los nodos. EC2 Puede usar la API de EKS (mediante la consola de EKS, la API de AWSCloudFormation, la CLI de AWS o Terraform`eksctl`) para crear, escalar y actualizar los nodos administrados. Los nodos gestionados ejecutan EC2 instancias de Amazon Linux 2 optimizadas para EKS en su cuenta, y puede instalar paquetes de software personalizados habilitando el acceso SSH. Cuando aprovisiona nodos administrados, se ejecutan como parte de un grupo de Auto Scaling administrado por EKS que puede abarcar varias zonas de disponibilidad; esto se controla a través de las subredes que proporciona al crear los nodos administrados. EKS también etiqueta automáticamente los nodos gestionados para que puedan usarse con Cluster AutoScaler.

Amazon EKS sigue el modelo de responsabilidad compartida CVEs y los parches de seguridad en los grupos de nodos gestionados. Dado que los nodos gestionados ejecutan Amazon EKS optimizado, AMIs Amazon EKS es responsable de crear versiones parcheadas de estos AMIs cuando se corrijan errores. Sin embargo, es responsable de implementar estas versiones de AMI parcheadas en los grupos de nodos administrados.

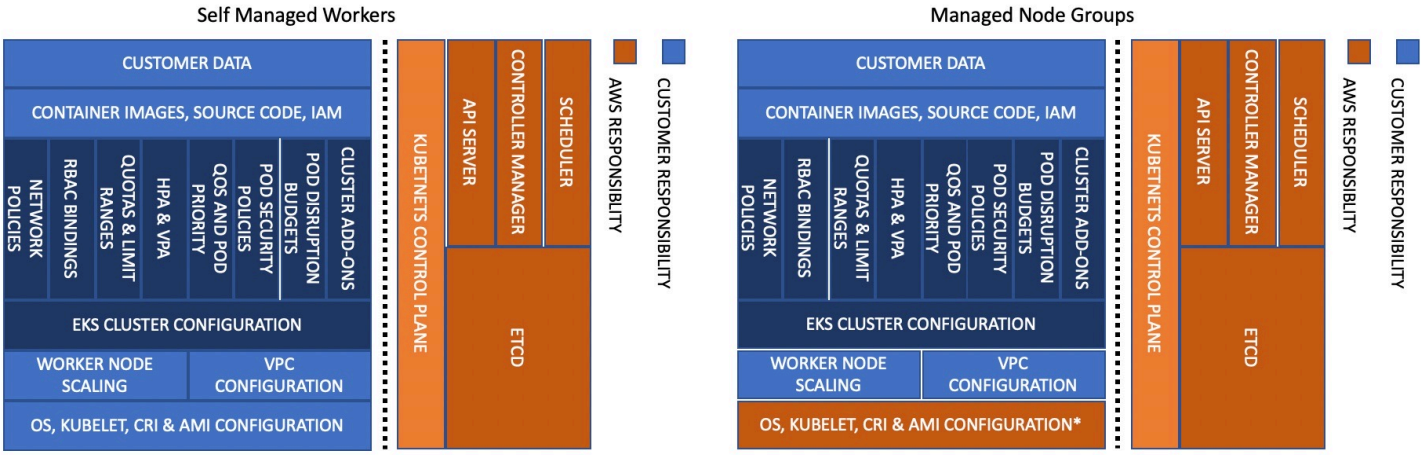
EKS también gestiona la actualización de los nodos, aunque usted debe iniciar el proceso de actualización. El proceso de actualización del nodo gestionado se explica en la documentación de EKS.

Si ejecuta nodos autogestionados, puede utilizar la AMI de Linux optimizada para Amazon EKS para crear nodos de trabajo. Usted es responsable de parchear y actualizar la AMI y los nodos. Se recomienda utilizar o utilizar `eksctl` la infraestructura como herramientas de código para aprovisionar los nodos autogestionados CloudFormation, ya que esto le facilitará la actualización de los nodos autogestionados. Considere la posibilidad de migrar a nuevos nodos cuando actualice los nodos de trabajo, ya que el proceso de migración contamina el grupo de nodos anterior **`NoSchedule`** y agota los nodos una vez que una nueva pila esté lista para aceptar la carga de trabajo del módulo existente. Sin embargo, también puedes realizar una actualización local de los nodos autogestionados.

Modelo de responsabilidad compartida - Fargate



Modelo de responsabilidad compartida - MNG



Esta guía incluye un conjunto de recomendaciones que puede utilizar para mejorar la confiabilidad de su plano de datos de EKS, los componentes principales de Kubernetes y sus aplicaciones.

Comentarios

Esta guía se publica GitHub para recopilar comentarios y sugerencias directos de la comunidad en general. EKS/Kubernetes Si tienes alguna buena práctica que consideres que deberíamos incluir en la guía, presenta un problema o envía un PR al GitHub repositorio. Tenemos la intención de actualizar la guía periódicamente a medida que se añadan nuevas funciones al servicio o cuando surja una nueva práctica recomendada.

Ejecutar aplicaciones de alta disponibilidad

Sus clientes esperan que su aplicación esté siempre disponible, incluso cuando realice cambios y, especialmente, durante los picos de tráfico. Una arquitectura escalable y resiliente permite que sus aplicaciones y servicios funcionen sin interrupciones, lo que hace que sus usuarios estén satisfechos. Una infraestructura escalable crece y se reduce en función de las necesidades de la empresa. Eliminar los puntos únicos de fallo es un paso fundamental para mejorar la disponibilidad de una aplicación y hacerla resiliente.

Con Kubernetes, puede operar sus aplicaciones y ejecutarlas de una manera resiliente y de alta disponibilidad. Su gestión declarativa garantiza que, una vez que hayas configurado la aplicación, Kubernetes intentará continuamente hacer coincidir el estado actual con el estado deseado.

Recomendaciones

Configure los presupuestos de Pod Disrup

Los presupuestos de interrupción de los pods se utilizan para limitar la cantidad de interrupciones simultáneas que sufrirá una aplicación. Deben configurarse para las cargas de trabajo si es importante tener siempre disponible una parte de esa carga de trabajo. EKS Auto Mode, Karpenter y Cluster Autoscaler conocen y respetan los presupuestos establecidos en materia de disrupción de cápsulas a la hora de reducir las emisiones. EKS Auto Mode, Karpenter y Managed Node Groups también respetan los presupuestos de interrupción de los módulos al actualizar los nodos

Evita usar pods individuales

Si toda la aplicación se ejecuta en un solo pod, la aplicación no estará disponible si se cierra ese pod. En lugar de implementar aplicaciones mediante pods individuales, cree implementaciones. Si un pod creado por una implementación falla o se cierra, el controlador de implementación iniciará un nuevo pod para garantizar que siempre se ejecute el número especificado de pods de réplica.

Ejecuta varias réplicas

La ejecución de varios pods de réplicas de una aplicación mediante una implementación ayuda a que funcione con una alta disponibilidad. Si una réplica falla, las réplicas restantes seguirán funcionando, aunque con una capacidad reducida, hasta que Kubernetes cree otro pod para compensar la pérdida. Además, puede usar el escalador automático de módulos horizontales para escalar las réplicas automáticamente en función de la demanda de carga de trabajo.

Programe réplicas en todos los nodos

Ejecutar varias réplicas no será muy útil si todas las réplicas se ejecutan en el mismo nodo y el nodo deja de estar disponible. Considere la posibilidad de utilizar restricciones de antiafinidad o dispersión de topología de pods para distribuir las réplicas de una implementación en varios nodos de trabajo.

Puede mejorar aún más la confiabilidad de una aplicación típica ejecutándola en varios. AZs

Uso de las reglas de antiafinidad de Pod

El siguiente manifiesto indica al programador de Kubernetes que prefiera colocar los pods en nodos separados y. AZs No requiere nodos distintos ni zonas de disponibilidad, ya que, en ese caso, Kubernetes no podrá programar ningún módulo una vez que haya un módulo en ejecución en cada zona de disponibilidad. Si tu aplicación solo requiere tres réplicas, puedes utilizarla `requiredDuringSchedulingIgnoredDuringExecution` para`topologyKey: topology.kubernetes.io/zone`, y el programador de Kubernetes no programará dos pods en la misma zona de disponibilidad.

```
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
```

Uso de restricciones de dispersión de la topología de Pod

Al igual que las reglas de antiafinidad de los pods, las restricciones de dispersión de la topología de los pods le permiten hacer que su aplicación esté disponible en diferentes dominios de errores (o topología), como hosts o. AZs Este enfoque funciona muy bien cuando se intenta garantizar la tolerancia a los errores y la disponibilidad, ya que dispone de varias réplicas en cada uno de los distintos dominios de topología. Las reglas de antiafinidad de los pods, por otro lado, pueden producir fácilmente un resultado cuando se tiene una única réplica en un dominio de topología, ya que los pods que tienen una antiafinidad entre sí tienen un efecto repelente. En esos casos, una réplica única en un nodo dedicado no es ideal para la tolerancia a errores ni supone un buen uso de los recursos. Con las restricciones de dispersión de la topología, tiene más control sobre la dispersión o distribución que el programador debería intentar aplicar en los dominios de la topología. Estas son algunas propiedades importantes que se pueden utilizar en este enfoque:

1. `maxSkew`Se usa para controlar o determinar el punto máximo en el que las cosas pueden ser desiguales en los dominios de la topología. Por ejemplo, si una aplicación tiene 10 réplicas y se

despliega en 3 AZs, no se puede obtener una distribución uniforme, pero sí se puede influir en el grado de desigualdad de la distribución. En este caso, `maxSkew` puede estar entre 1 y 10. Un valor de 1 significa que puedes terminar con un diferencial igual `4,3,3` `3,4,3` o `3,3,4` superior al 3 AZs. Por el contrario, un valor de 10 significa que puedes terminar con un diferencial igual `10,0,0` `0,10,0` o superior `0,0,10` a 3 AZs.

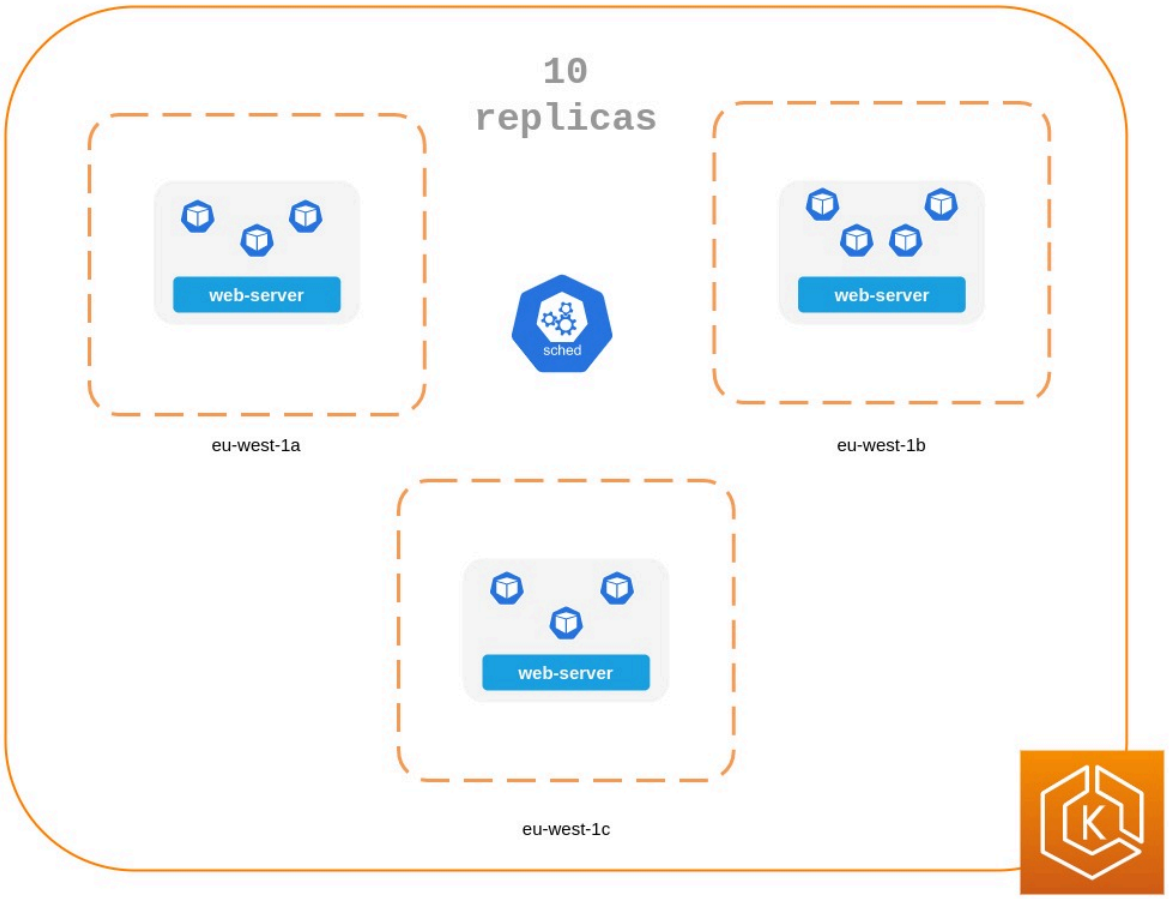
2. `topologyKey`Es una clave para una de las etiquetas de los nodos y define el tipo de dominio de topología que se debe usar para la distribución del módulo. Por ejemplo, una dispersión zonal tendría el siguiente par clave-valor:

   ```
   topologyKey: "topology.kubernetes.io/zone"
   ```

3. La `whenUnsatisfiable` propiedad se usa para determinar cómo desea que responda el planificador si no se pueden cumplir las restricciones deseadas.
4. `labelSelector`Se utiliza para encontrar los módulos que coincidan, de modo que el planificador pueda conocerlos a la hora de decidir dónde colocar los módulos de acuerdo con las restricciones que se especifiquen.

Además de los anteriores, hay otros campos sobre los que puede obtener más información en la documentación de Kubernetes.

La topología del pod distribuye las restricciones en 3 AZs




```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spread-host-az
  labels:
    app: web-server
spec:
  replicas: 10
  selector:
    matchLabels:
      app: web-server
  template:
    metadata:
      labels:
        app: web-server
    spec:
      topologySpreadConstraints:
```

```
    - maxSkew: 1
      topologyKey: "topology.kubernetes.io/zone"
      whenUnsatisfiable: ScheduleAnyway
      labelSelector:
        matchLabels:
          app: express-test
  containers:
  - name: web-app
    image: nginx:1.16-alpine
```

Ejecute Kubernetes Metrics Server

Instala el servidor de métricas de Kubernetes para ayudarte a escalar tus aplicaciones. Los complementos de escalado automático de Kubernetes, como HPA y VPA, necesitan rastrear las métricas de las aplicaciones para escalarlas. El servidor de métricas recopila métricas de recursos que se pueden utilizar para tomar decisiones de escalado. Las métricas se recopilan de los kubelets y se muestran en el formato de la API de métricas.

El servidor de métricas no retiene ningún dato y no es una solución de monitorización. Su objetivo es exponer las métricas de uso de la CPU y la memoria a otros sistemas. Si quieres hacer un seguimiento del estado de tu aplicación a lo largo del tiempo, necesitas una herramienta de monitorización como Prometheus o Amazon. CloudWatch

Siga la documentación de EKS para instalar metrics-server en su clúster de EKS.

Escalador automático de pod horizontal (HPA)

HPA puede escalar automáticamente su aplicación en respuesta a la demanda y ayudarle a evitar que sus clientes se vean afectados durante los picos de tráfico. Se implementa como un bucle de control en Kubernetes que consulta periódicamente las métricas que proporcionan métricas de APIs recursos.

HPA puede recuperar métricas de lo siguiente: 1. APIs `metrics.k8s.io`también conocida como API de métricas de recursos: proporciona el uso de CPU y memoria para los pods 2. `custom.metrics.k8s.io` — Proporciona métricas de otros recopiladores de métricas, como Prometheus; estas métricas son internas de su clúster de Kubernetes. 3. `external.metrics.k8s.io` — Proporciona métricas externas a su clúster de Kubernetes (por ejemplo, profundidad de cola de SQS, latencia de ELB).

Debe usar una de estas tres APIs para proporcionar la métrica para escalar su aplicación.

Escalar las aplicaciones en función de métricas personalizadas o externas

Puede usar métricas personalizadas o externas para escalar su aplicación en función de métricas distintas de la utilización de la CPU o la memoria. Los servidores de la API de métricas personalizadas proporcionan la `custom-metrics.k8s.io` API que HPA puede usar para escalar automáticamente las aplicaciones.

Puede utilizar el adaptador de Prometheus para métricas de Kubernetes para recopilar métricas de Prometheus y utilizarlas APIs con la HPA. En este caso, el adaptador de Prometheus expondrá las métricas de Prometheus en el formato de la API de métricas.

Una vez que haya implementado el adaptador Prometheus, podrá consultar métricas personalizadas mediante kubectl. `kubectl get –raw /apis/custom.metrics.k8s.io/v1beta1/`

Las métricas externas, como su nombre indica, proporcionan al escalador automático del pod horizontal la capacidad de escalar las implementaciones mediante métricas externas al clúster de Kubernetes. Por ejemplo, en las cargas de trabajo de procesamiento por lotes, es habitual escalar el número de réplicas en función del número de trabajos en curso en una cola de SQS.

Para escalar automáticamente las cargas de trabajo de Kubernetes, puedes usar KEDA (escalado automático basado en eventos de Kubernetes), un proyecto de código abierto que puede impulsar el escalado de contenedores en función de una serie de eventos personalizados. En este blog de AWS se describe cómo utilizar Amazon Managed Service for Prometheus for Kubernetes y el autoscalamiento de las cargas de trabajo.

Escalador automático de cápsulas verticales (VPA)

El VPA ajusta automáticamente la reserva de CPU y memoria de tus Pods para ayudarte a ajustar el tamaño correcto a tus aplicaciones. En el caso de las aplicaciones que deban escalarse verticalmente (lo que se consigue aumentando la asignación de recursos), puedes usar el VPA para escalar automáticamente las réplicas de los pods o proporcionar recomendaciones de escalado.

Es posible que tu aplicación deje de estar disponible temporalmente si VPA necesita escalarla, ya que la implementación actual del VPA no realiza ajustes in situ en los pods, sino que recrea el pod que necesita escalarse.

La documentación de EKS incluye un tutorial para configurar el VPA.

El proyecto Goldilocks de Fairwinds proporciona un panel de control para visualizar las recomendaciones de VPA para las solicitudes y los límites de CPU y memoria. Su modo de

actualización de VPA te permite escalar automáticamente los pods según las recomendaciones del VPA.

Actualización de aplicaciones

Las aplicaciones modernas requieren una innovación rápida con un alto grado de estabilidad y disponibilidad. Kubernetes le proporciona las herramientas para actualizar sus aplicaciones de forma continua sin interrumpir a sus clientes.

Veamos algunas de las mejores prácticas que permiten implementar cambios rápidamente sin sacrificar la disponibilidad.

Disponga de un mecanismo para realizar reversiones

Tener un botón de deshacer puede evitar desastres. Se recomienda probar las implementaciones en un entorno inferior independiente (entorno de prueba o desarrollo) antes de actualizar el clúster de producción. El uso de una canalización de CI/CD puede ayudarle a automatizar y probar las implementaciones. Con un proceso de implementación continuo, puede volver rápidamente a la versión anterior si la actualización resulta defectuosa.

Puede usar Deployments para actualizar una aplicación en ejecución. Por lo general, esto se hace actualizando la imagen del contenedor. Puedes usarlo `kubectl` para actualizar una implementación como esta:

```
kubectl --record deployment.apps/nginx-deployment set image nginx-deployment
 nginx=nginx:1.16.1
```

El `--record` argumento registra los cambios en la implementación y le ayuda si necesita realizar una reversión. `kubectl rollout history deployment`muestra los cambios registrados en las implementaciones de su clúster. Puede deshacer un cambio mediante. `kubectl rollout undo deployment <DEPLOYMENT_NAME>`

De forma predeterminada, cuando actualizas una implementación que requiere una recreación de los pods, Deployment realizará una actualización progresiva. En otras palabras, Kubernetes solo actualizará una parte de los pods en ejecución de una implementación y no todos los pods a la vez. Puedes controlar la forma en que Kubernetes realiza las actualizaciones sucesivas a través de la propiedad. `RollingUpdateStrategy`

Al realizar una actualización sucesiva de una implementación, puedes usar la `Max Unavailable`propiedad para especificar el número máximo de pods que no estarán disponibles

durante la actualización. La `Max Surge` propiedad de despliegue te permite establecer el número máximo de pods que se pueden crear en lugar del número deseado de pods.

Considere la posibilidad `max unavailable` de realizar ajustes para garantizar que una implementación no cause molestias a sus clientes. Por ejemplo, Kubernetes establece un 25% de forma `max unavailable` predeterminada, lo que significa que si tienes 100 pods, es posible que solo tengas 75 pods funcionando activamente durante el lanzamiento. Si tu aplicación necesita un mínimo de 80 pods, esta implementación puede ser perjudicial. En su lugar, puedes `max unavailable` configurarlo en un 20% para asegurarte de que haya al menos 80 pods funcionales durante la implementación.

Usa despliegues blue/green

Los cambios son intrínsecamente riesgosos, pero los cambios que no se pueden deshacer pueden ser potencialmente catastróficos. Los procedimientos de cambio que permiten retroceder en el tiempo de forma eficaz mediante una reversión hacen que las mejoras y la experimentación sean más seguras. Blue/green las implementaciones le proporcionan un método para retirar rápidamente los cambios si las cosas van mal. En esta estrategia de despliegue, se crea un entorno para la nueva versión. Este entorno es idéntico a la versión actual de la aplicación que se está actualizando. Una vez que se aprovisiona el nuevo entorno, el tráfico se enruta al nuevo entorno. Si la nueva versión produce los resultados deseados sin generar errores, se cierra el entorno anterior. De lo contrario, el tráfico se restablece a la versión anterior.

Puedes realizar blue/green despliegues en Kubernetes creando un nuevo despliegue que sea idéntico al despliegue de la versión existente. Una vez que compruebes que los pods de la nueva implementación funcionan sin errores, puedes empezar a enviar tráfico a la nueva implementación cambiando la `selector` especificación del servicio que dirige el tráfico a los pods de tu aplicación.

Muchas herramientas de integración continua, como Flux, Jenkins y Spinnaker, permiten automatizar las implementaciones. blue/green El blog de AWS Containers incluye un tutorial sobre el uso del controlador de equilibrio de carga de AWS: uso del controlador de equilibrio de carga de AWS blue/green para la implementación, la implementación y las pruebas A/B

Utilice las implementaciones de Canary

Las implementaciones de Canary son una variante de las blue/green implementaciones que pueden eliminar significativamente el riesgo derivado de los cambios. En esta estrategia de despliegue, se crea un despliegue nuevo con menos módulos junto con el despliegue anterior y se desvía un

pequeño porcentaje del tráfico al nuevo despliegue. Si las métricas indican que la nueva versión funciona igual o mejor que la versión existente, debe aumentar progresivamente el tráfico a la nueva implementación y, al mismo tiempo, ampliarla hasta que todo el tráfico se desvíe a la nueva implementación. Si hay algún problema, puede dirigir todo el tráfico a la implementación anterior y dejar de enviar tráfico a la nueva implementación.

Aunque Kubernetes no ofrece una forma nativa de realizar despliegues canarios, puedes usar herramientas como Flagger con Istio.

Health chequeos y autocuración

Ningún software está libre de errores, pero Kubernetes puede ayudarte a minimizar el impacto de los fallos de software. En el pasado, si una aplicación fallaba, alguien tenía que solucionar la situación reiniciándola manualmente. Kubernetes te permite detectar errores de software en tus Pods y sustituirlos automáticamente por réplicas nuevas. Con Kubernetes, puedes supervisar el estado de tus aplicaciones y reemplazar automáticamente las instancias en mal estado.

Kubernetes admite tres tipos de controles de estado:

1. Sonda de vivacidad
2. Sonda de inicio (compatible con la versión 1.16+ de Kubernetes)
3. Sonda de preparación

Kubelet, el agente de Kubernetes, es responsable de realizar todas las comprobaciones mencionadas anteriormente. Kubelet puede comprobar el estado de un Pods de tres maneras: puede ejecutar un comando de shell dentro del contenedor de un Pod, enviar una solicitud HTTP GET a su contenedor o abrir un conector TCP en un puerto específico.

Si eliges un sondeo *exec* basado, que ejecute un script de shell dentro de un contenedor, asegúrate de que el comando shell se cierre antes de que caduque el valor. `timeoutSeconds` De lo contrario, el nodo tendrá `<defunct>` procesos, lo que provocará una falla en el nodo.

Recomendaciones

Utilice Liveness Probe para eliminar las cápsulas insalubres

La sonda Liveness puede detectar situaciones de bloqueo en las que el proceso continúa ejecutándose, pero la aplicación deja de responder. Por ejemplo, si ejecutas un servicio web que

escucha en el puerto 80, puedes configurar una sonda Liveness para que envíe una solicitud HTTP GET al puerto 80 del Pod. Kubelet envía periódicamente una solicitud GET al Pod y espera una respuesta; si el Pod responde entre 200 y 399, el kubelet considera que el Pod está en buen estado; de lo contrario, el Pod se marcará como en mal estado. Si un pod no pasa las comprobaciones de estado continuas, el kubelet lo cancelará.

Puedes usarlo `initialDelaySeconds` para retrasar la primera sonda.

Cuando utilices el Liveness Probe, asegúrate de que tu aplicación no se encuentre en una situación en la que todos los pods no apaguen simultáneamente el Liveness Probe, ya que Kubernetes intentará reemplazar todos los pods, lo que hará que la aplicación quede desconectada. Además, Kubernetes seguirá creando nuevos pods que también fallarán con los Liveness Probes, lo que supondrá una carga innecesaria para el plano de control. Evita configurar el Liveness Probe para que dependa de un factor externo a tu Pod, por ejemplo, una base de datos externa. En otras palabras, una external-to-your-Pod base de datos que no responda no debería hacer que tus Pods no superen sus Liveness Probes.

El libro posterior a LIVENESS PROBES ARE DANGEROUS, de Sandor Szücs, describe los problemas que pueden provocar las sondas mal configuradas.

Utilice Startup Probe para las aplicaciones que tardan más en iniciarse

Cuando tu aplicación necesite más tiempo para iniciarse, puedes usar la sonda de inicio para retrasar la sonda de vitalidad y preparación. Por ejemplo, una aplicación Java que necesita hidratar la memoria caché de una base de datos puede tardar hasta dos minutos en ser completamente funcional. Cualquier sonda de vitalidad o preparación hasta que esté completamente operativa podría fallar. La configuración de una sonda de inicio permitirá que la aplicación Java se mantenga en buen estado antes de que se ejecuten Liveness o Readiness Probe.

Hasta que la sonda de inicio tenga éxito, todas las demás sondas se desactivarán. Puedes definir el tiempo máximo que debe esperar Kubernetes para que se inicie la aplicación. Si, transcurrido el tiempo máximo configurado, el pod sigue fallando en las pruebas de inicio, se cerrará y se creará un nuevo pod.

El Startup Probe es similar al Liveness Probe: si fallan, se vuelve a crear el Pod. Como explica Ricardo A. en su artículo Fantastic Probes And How To Configure Them, los Startup Probes se deben utilizar cuando el tiempo de inicio de una aplicación sea impredecible. Si sabe que su aplicación necesita diez segundos para iniciarse, debería utilizar Liveness/Readiness Probe with en su lugar. `initialDelaySeconds`

Utilice Readiness Probe para detectar una falta de disponibilidad parcial

Mientras que la sonda Liveness detecta errores en una aplicación y se resuelven cerrando el pod (es decir, reiniciando la aplicación), Readiness Probe detecta situaciones en las que la aplicación puede no estar disponible temporalmente. En estas situaciones, es posible que la aplicación deje de responder temporalmente; sin embargo, se espera que vuelva a estar en buen estado una vez finalizada la operación.

Por ejemplo, durante I/O operaciones de disco intensas, es posible que las aplicaciones no estén disponibles temporalmente para gestionar las solicitudes. En este caso, cerrar el Pod de la aplicación no es una solución; al mismo tiempo, las solicitudes adicionales que se envíen al Pod pueden fallar.

Puedes usar la sonda de preparación para detectar la falta de disponibilidad temporal en tu aplicación y dejar de enviar solicitudes a su pod hasta que vuelva a funcionar. A diferencia de Liveness Probe, en el que un error provocaría la recreación del Pod, un error en el Readiness Probe implicaría que el Pod no recibirá tráfico del servicio de Kubernetes. Cuando la sonda de preparación se ejecute correctamente, el Pod volverá a recibir tráfico del servicio.

Al igual que la sonda Liveness Probe, evita configurar las sondas de preparación que dependan de un recurso externo al pod (como una base de datos). Esta es una situación en la que una configuración incorrecta de Readiness puede hacer que la aplicación deje de funcionar: si la sonda de preparación de un pod falla cuando no se puede acceder a la base de datos de la aplicación, otras réplicas del pod también fallarán simultáneamente, ya que comparten los mismos criterios de comprobación de estado. Si configuras la sonda de esta forma, se garantizará que, cuando la base de datos no esté disponible, las sondas de preparación del Pod fallarán y Kubernetes dejará de enviar tráfico a todos los pods.

Un efecto secundario del uso de las sondas de preparación es que pueden aumentar el tiempo que se tarda en actualizar las implementaciones. Las réplicas nuevas no recibirán tráfico a menos que las pruebas de preparación se realicen correctamente; hasta entonces, las réplicas antiguas seguirán recibiendo tráfico.

Hacer frente a las interrupciones

Los pods tienen una vida útil limitada. Incluso si tienes pods de larga duración, es prudente asegurarte de que los pods terminen correctamente cuando llegue el momento. En función de tu estrategia de actualización, las actualizaciones de los clústeres de Kubernetes pueden requerir la creación de nuevos nodos de trabajo, lo que requiere que todos los pods se vuelvan a crear en los

nodos más nuevos. Una gestión adecuada de las terminaciones y los presupuestos de interrupción de los pods pueden ayudarte a evitar interrupciones en el servicio, ya que los pods se retiran de los nodos más antiguos y se vuelven a crear en los nodos más nuevos.

La forma preferida de actualizar los nodos de trabajo es crear nuevos nodos de trabajo y cerrar los antiguos. Antes de cerrar los nodos de trabajo, debería `drain` hacerlo. Cuando se vacía un nodo trabajador, todos sus módulos se desalojan de forma segura. En este caso, la palabra clave es «seguridad»; cuando se desalojan las cápsulas de un trabajador, no se les envía simplemente una `SIGKILL` señal. En su lugar, se envía una `SIGTERM` señal al proceso principal (PID 1) de cada contenedor de las cápsulas que se va a desalojar. Una vez enviada la `SIGTERM` señal, Kubernetes concederá al proceso un `SIGKILL` tiempo (período de gracia) antes de que se envíe la señal. Este período de gracia es de 30 segundos de forma predeterminada; puedes anular el predeterminado usando `grace-period` flag en kubectl o declare en tu Podspec. `terminationGracePeriodSeconds`

`kubectl delete pod <pod name> –grace-period=<seconds>`

Es habitual tener contenedores en los que el proceso principal no tenga el PID 1. Considera este contenedor de muestras basado en Python:

```
$ kubectl exec python-app -it ps
 PID USER TIME COMMAND
 1   root 0:00 {script.sh} /bin/sh ./script.sh
 5   root 0:00 python app.py
```

En este ejemplo, el script shell recibe`SIGTERM`, el proceso principal, que resulta ser una aplicación de Python en este ejemplo, no recibe ninguna `SIGTERM` señal. Cuando se cierre el Pod, la aplicación Python se cerrará abruptamente. Esto se puede solucionar cambiando el `ENTRYPOINT`contenedor para iniciar la aplicación Python. Como alternativa, puedes usar una herramienta como dumb-init para asegurarte de que tu aplicación pueda gestionar las señales.

También puedes usar Container Hooks para ejecutar un script o una solicitud HTTP al iniciar o detener el contenedor. La acción de `PreStop` enganche se ejecuta antes de que el contenedor reciba una `SIGTERM` señal y debe completarse antes de que se envíe esta señal. El `terminationGracePeriodSeconds` valor se aplica desde el momento en que la acción de `PreStop` enganche comienza a ejecutarse, no cuando se envía la `SIGTERM` señal.

Recomendaciones

Proteja la carga de trabajo crítica con Pod Disruption Budgets

Pod Disruption Budget o PDB pueden detener temporalmente el proceso de desalojo si el número de réplicas de una aplicación cae por debajo del umbral declarado. El proceso de desalojo continuará una vez que el número de réplicas disponibles supere el umbral. Puede utilizar el PDB para declarar el `maxUnavailable` número `minAvailable` y el número de réplicas. Por ejemplo, si quieres que estén disponibles al menos tres copias de tu aplicación, puedes crear una PDB.

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: my-svc-pdb
spec:
  minAvailable: 3
  selector:
    matchLabels:
      app: my-svc
```

La política de PDB anterior indica a Kubernetes que detenga el proceso de desalojo hasta que haya tres o más réplicas disponibles. El `PodDisruptionBudgets` agotamiento de los nodos se respeta. Durante una actualización de un grupo de nodos gestionada por EKS, los nodos se agotan con un tiempo de espera de quince minutos. Transcurridos quince minutos, si la actualización no es forzada (la opción se denomina actualización progresiva en la consola de EKS), la actualización no se realizará correctamente. Si la actualización es forzada, se eliminan los pods.

En el caso de los nodos autogestionados, también puede utilizar herramientas como AWS Node Termination Handler, que garantiza que el plano de control de Kubernetes responda adecuadamente a los eventos que pueden provocar que la EC2 instancia deje de estar disponible, como los eventos de EC2mantenimiento y las interrupciones puntuales. EC2 Utiliza la API de Kubernetes para acordonar el nodo y garantizar que no se programen nuevos pods. A continuación, lo agota y cierra los pods que estén en ejecución.

Puedes usar la antiafinidad de los pods para programar los pods de una implementación en diferentes nodos y evitar demoras relacionadas con la PDB durante las actualizaciones de los nodos.

Practica la ingeniería del caos

La ingeniería del caos es la disciplina que consiste en experimentar con un sistema distribuido para generar confianza en la capacidad del sistema para soportar condiciones turbulentas durante la producción.

En su blog, Dominik Tornow explica que Kubernetes es un sistema declarativo en el que «el usuario proporciona al sistema una representación del estado deseado del sistema». Luego, el sistema considera el estado actual y el estado deseado para determinar la secuencia de comandos para pasar del estado actual al estado deseado. » Esto significa que Kubernetes siempre almacena el estado deseado y, si el sistema se desvía, Kubernetes tomará medidas para restablecer el estado. Por ejemplo, si un nodo de trabajo deja de estar disponible, Kubernetes reprogramará los Pods en otro nodo de trabajo. Del mismo modo, si uno `replica` se bloquea, el controlador de despliegue creará uno nuevo. `replica` De esta forma, los controladores de Kubernetes corrigen automáticamente los fallos.

Las herramientas de ingeniería del caos, como Gremlin, le ayudan a probar la resiliencia de su clúster de Kubernetes e identificar los puntos únicos de fallo. Las herramientas que crean un caos artificial en tu clúster (y más allá) pueden descubrir las debilidades sistémicas, ofrecer la oportunidad de identificar los cuellos de botella y los errores de configuración y corregir los problemas en un entorno controlado. La filosofía de Chaos Engineering aboga por romper las cosas a propósito y realizar pruebas de stress en la infraestructura para minimizar los tiempos de inactividad imprevistos.

Utilice una malla de servicios

Puede utilizar una malla de servicios para mejorar la resiliencia de su aplicación. Las mallas de servicio permiten service-to-service la comunicación y aumentan la observabilidad de su red de microservicios. La mayoría de los productos de malla de servicios funcionan con un pequeño proxy de red junto a cada servicio que intercepta e inspecciona el tráfico de red de la aplicación. Puede colocar la aplicación en una malla sin necesidad de modificarla. Con las funciones integradas del proxy de servicio, puede hacer que genere estadísticas de red, cree registros de acceso y añada encabezados HTTP a las solicitudes salientes para el rastreo distribuido.

Una malla de servicios puede ayudarle a hacer que sus microservicios sean más resilientes con funciones como los reintentos automáticos de solicitudes, los tiempos de espera, la interrupción de circuitos y la limitación de velocidad.

Si opera varios clústeres, puede usar una malla de servicios para habilitar la comunicación entre clústeres. service-to-service

Mallas de servicios

- Istio
- LinkerD
- Cónsul

Observabilidad

La observabilidad es un término general que incluye el monitoreo, el registro y el rastreo. Las aplicaciones basadas en microservicios se distribuyen por naturaleza. A diferencia de las aplicaciones monolíticas, en las que basta con monitorear un solo sistema, en una arquitectura de aplicaciones distribuidas, es necesario monitorear el rendimiento de cada componente. Puede usar sistemas de monitoreo, registro y rastreo distribuido a nivel de clúster para identificar los problemas en su clúster antes de que afecten a sus clientes.

Las herramientas integradas de Kubernetes para la resolución de problemas y la supervisión son limitadas. El servidor de métricas recopila las métricas de los recursos y las almacena en la memoria, pero no las conserva. Puedes ver los registros de un pod con kubectl, pero Kubernetes no conserva los registros automáticamente. Además, la implementación del rastreo distribuido se realiza a nivel de código de la aplicación o mediante mallas de servicios.

La extensibilidad de Kubernetes brilla aquí. Kubernetes le permite ofrecer la solución centralizada de monitoreo, registro y rastreo que prefiera.

Recomendaciones

Supervise sus aplicaciones

La cantidad de métricas que necesita monitorear en las aplicaciones modernas crece continuamente. Es útil disponer de una forma automatizada de realizar el seguimiento de las aplicaciones, de modo que pueda centrarse en resolver los desafíos de sus clientes. Las herramientas de monitoreo para todo el clúster, como Prometheus o CloudWatchContainer Insights, pueden monitorear su clúster y su carga de trabajo y proporcionarle señales cuando, o preferiblemente, antes de que las cosas salgan mal.

Las herramientas de monitoreo le permiten crear alertas a las que su equipo de operaciones puede suscribirse. Considere la posibilidad de establecer reglas para activar las alarmas en caso

de eventos que, si se agravan, pueden provocar una interrupción o afectar al rendimiento de las aplicaciones.

Si no tienes claro qué métricas debes monitorear, puedes inspirarte en estos métodos:

- Método RED. Representa las solicitudes, los errores y la duración.
- Método USE. Representa la utilización, la saturación y los errores.

El post de Sysdig sobre las mejores prácticas para emitir alertas en Kubernetes incluye una lista completa de componentes que pueden afectar a la disponibilidad de sus aplicaciones.

Utilice la biblioteca cliente de Prometheus para exponer las métricas de la aplicación

Además de supervisar el estado de la aplicación y agregar métricas estándar, también puede usar la biblioteca cliente de Prometheus para exponer métricas personalizadas específicas de la aplicación y mejorar la observabilidad de la aplicación.

Utilice herramientas de registro centralizadas para recopilar y conservar los registros

El registro en EKS se divide en dos categorías: registros del plano de control y registros de aplicaciones. El registro del plano de control de EKS proporciona registros de auditoría y diagnóstico directamente desde el plano de control a CloudWatch los registros de su cuenta. Los registros de aplicaciones son registros generados por los pods que se ejecutan dentro de tu clúster. Los registros de aplicaciones incluyen los registros producidos por los pods que ejecutan las aplicaciones de lógica empresarial y los componentes del sistema de Kubernetes, como CoredNS, Cluster Autoscaler, Prometheus, etc.

Los EKS proporcionan cinco tipos de registros del plano de control:

1. Registros de los componentes del servidor de la API de Kubernetes
2. Auditoría
3. Autenticador
4. Administrador de controladores
5. Programador

Los registros del administrador del controlador y del planificador pueden ayudar a diagnosticar problemas en el plano de control, como los cuellos de botella y los errores. De forma

predeterminada, los registros del plano de control de EKS no se envían a Logs. CloudWatch Puede habilitar el registro del plano de control y seleccionar los tipos de registros del plano de control de EKS que desea capturar para cada clúster de su cuenta

Para recopilar los registros de las aplicaciones, es necesario instalar en el clúster una herramienta de agregación de registros como Fluent Bit, Fluentd o CloudWatchContainer Insights.

Las herramientas de agregación de registros de Kubernetes se ejecutan como DaemonSets registros de contenedores y los extraen de los nodos. Luego, los registros de las aplicaciones se envían a un destino centralizado para su almacenamiento. Por ejemplo, CloudWatch Container Insights puede usar FluentBit o Fluentd para recopilar registros y enviarlos a CloudWatch Logs para su almacenamiento. Fluent Bit y Fluentd son compatibles con muchos de los sistemas de análisis de registros más populares, como Elasticsearch e InfluxDB, lo que le permite cambiar el backend de almacenamiento de sus registros modificando Fluentbit o la configuración de registros de Fluentd.

Utilice un sistema de rastreo distribuido para identificar los cuellos de botella

Una aplicación moderna típica tiene componentes distribuidos por la red y su confiabilidad depende del correcto funcionamiento de cada uno de los componentes que componen la aplicación. Puede utilizar una solución de rastreo distribuido para comprender cómo fluyen las solicitudes y cómo se comunican los sistemas. Los rastreos pueden mostrarle dónde existen cuellos de botella en la red de aplicaciones y evitar problemas que puedan provocar errores en cascada.

Tiene dos opciones para implementar el rastreo en sus aplicaciones: puede implementar el rastreo distribuido a nivel de código mediante bibliotecas compartidas o usar una malla de servicios.

Implementar el rastreo a nivel de código puede ser desventajoso. En este método, debe realizar cambios en el código. Esto se complica aún más si tiene aplicaciones políglotas. También es responsable del mantenimiento de otra biblioteca en todos sus servicios.

Las mallas de servicio, como LinkerD e Istio, se pueden utilizar para implementar el rastreo distribuido en su aplicación con cambios mínimos en el código de la aplicación. Puede usar Service Mesh para estandarizar la generación, el registro y el rastreo de métricas.

Las herramientas de rastreo, como AWS X-Ray, de Jaeger son compatibles con las implementaciones de bibliotecas compartidas y redes de servicios.

Considere la posibilidad de utilizar una herramienta de rastreo, como AWS X-Ray o Jaeger, que sea compatible con ambas implementaciones (biblioteca compartida y malla de servicios), de modo que no tenga que cambiar de herramienta si más adelante adopta la malla de servicios.

Plano de control EKS

Amazon Elastic Kubernetes Service (EKS) es un servicio de Kubernetes administrado que le facilita la ejecución de Kubernetes en AWS sin necesidad de instalar, operar y mantener su propio plano de control de Kubernetes o nodos de trabajo. Funciona con Kubernetes en sentido ascendente y está certificado como compatible con Kubernetes. Esta conformidad garantiza que EKS sea compatible con Kubernetes, al igual que la versión comunitaria de código abierto que se puede instalar de forma APIs local o local. EC2 Las aplicaciones existentes que se ejecutan en Kubernetes upstream son compatibles con Amazon EKS.

EKS administra automáticamente la disponibilidad y la escalabilidad de los nodos del plano de control de Kubernetes y reemplaza automáticamente los nodos del plano de control en mal estado.

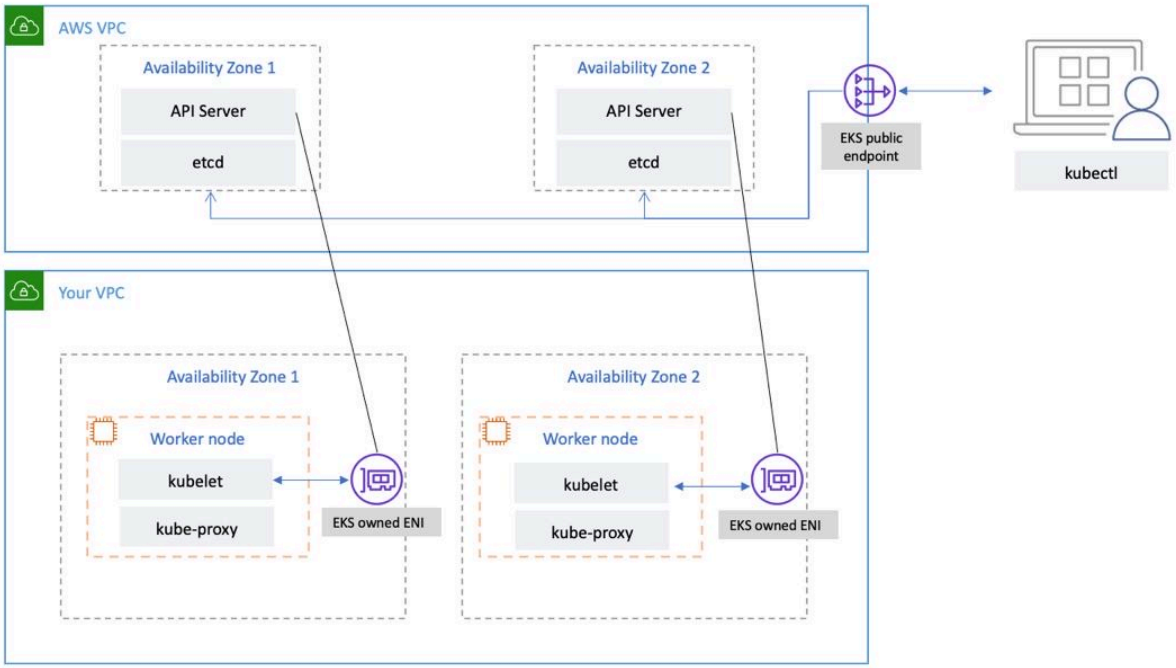
Arquitectura EKS

La arquitectura EKS está diseñada para eliminar cualquier punto único de fallo que pueda comprometer la disponibilidad y la durabilidad del plano de control de Kubernetes.

El plano de control de Kubernetes gestionado por EKS se ejecuta dentro de una VPC gestionada por EKS. El plano de control de EKS comprende la API de Kubernetes, los nodos del servidor, etc., el clúster. Nodos del servidor de API de Kubernetes que ejecutan componentes como el servidor de API o el programador, y que se ejecutan `kube-controller-manager` en un grupo de autoscalamiento. EKS ejecuta un mínimo de dos nodos de servidor de API en distintas zonas de disponibilidad (AZs) dentro de la región de AWS. Del mismo modo, para mayor durabilidad, los nodos del servidor etcd también se ejecutan en un grupo de autoscalamiento que abarca tres nodos. AZs EKS ejecuta una puerta de enlace NAT en cada zona de disponibilidad, y los servidores API y etcd se ejecutan en una subred privada. Esta arquitectura garantiza que un evento en una única zona de disponibilidad no afecte a la disponibilidad del clúster EKS.

Cuando crea un clúster nuevo, Amazon EKS crea un punto de enlace de alta disponibilidad para el servidor API de Kubernetes administrado que utiliza para comunicarse con el clúster (mediante herramientas como). `kubectl` El punto final administrado usa NLB para equilibrar la carga de los servidores de la API de Kubernetes. EKS también proporciona dos ENI diferentes AZs para facilitar la comunicación con sus nodos de trabajo.

Plano de datos EKS: conectividad de red



Puedes configurar si se puede acceder al servidor de API de tu clúster de Kubernetes desde la Internet pública (mediante el punto de conexión público) o a través de tu VPC (mediante el gestionado por EKS) o desde ambos. ENIs

Tanto si los usuarios como los nodos trabajadores se conectan al servidor de API mediante el punto final público o el ENI gestionado por EKS, existen rutas de conexión redundantes.

Recomendaciones

Revise las siguientes recomendaciones.

Supervise las métricas del plano de control

La supervisión de las métricas de la API de Kubernetes puede proporcionarte información sobre el rendimiento del plano de control e identificar problemas. Un plano de control defectuoso puede comprometer la disponibilidad de las cargas de trabajo que se ejecutan dentro del clúster. Por ejemplo, los controladores mal escritos pueden sobrecargar los servidores de la API y afectar a la disponibilidad de la aplicación.

Kubernetes expone las métricas del plano de control en el punto final. `/metrics`

Puede ver las métricas expuestas mediante: `kubectl`

```
kubectl get --raw /metrics
```

Estas métricas se representan en formato de texto de Prometheus.

Puedes usar Prometheus para recopilar y almacenar estas métricas. En mayo de 2020, CloudWatch se agregó soporte para monitorear las métricas CloudWatch de Prometheus en Container Insights. Por lo tanto, también puede usar Amazon CloudWatch para monitorear el plano de control del EKS. Puede utilizar el tutorial para añadir un nuevo Prometheus Scrape Target: Prometheus KPI Server Metrics para recopilar métricas CloudWatch y crear un panel de control para supervisar el plano de control de su clúster.

Puedes encontrar las métricas del servidor API de Kubernetes aquí. Por ejemplo, `apiserver_request_duration_seconds` puede indicar cuánto tardan en ejecutarse las solicitudes de API.

Considere la posibilidad de monitorizar estas métricas del plano de control:

Servidor de API

Métrica	Description (Descripción)
`apiserver_request_total`	Contador de solicitudes de apiserver desglosadas por verbo, valor de ensayo, grupo, versión, recurso, ámbito, componente y código de respuesta HTTP.
`apiserver_request_duration_seconds*`	Histograma de latencia de respuesta en segundos para cada verbo, valor de simulacro, grupo, versión, recurso, subrecurso, ámbito y componente.
`apiserver_admission_controller_admission_duration_seconds*`	Histograma de latencia del controlador de admisión en segundos, identificado por su nombre y desglosado para cada operación y recurso y tipo de API (validar o admitir).

Métrica	Description (Descripción)
`apiserver_admission_webhook_rejection_count`	Recuento de rechazos de webhooks de admisión. Se identifica por nombre, operación, código de rechazo, tipo (validando o admitiendo), tipo de error (calling_webhook_error, apiserver_internal_error, no_error)
`rest_client_request_duration_seconds*`	Solicita el histograma de latencia en segundos. Desglosado por verbo y URL.
`rest_client_requests_total`	Número de solicitudes HTTP, fragmentadas por código de estado, método y host.

- Las métricas del histograma incluyen los sufijos _bucket, _sum y _count.

etc.d.

Métrica	Descri on (Descr ión)
`etcd_request_duration_seconds*`	Etcd solicita el histogr a de latenci en segun para cada operac y

Métrica	Descri on (Descr ión)
	tipo de objeto
`apiserver_storage_db_total_size_in_bytes` o `apiserver_storage_size_bytes` (a partir de EKS v1.28)	Tamañ de la base de datos Etcd.

- Las métricas del histograma incluyen los sufijos _bucket, _sum y _count.

Considere la posibilidad de utilizar el panel de información general sobre la supervisión de Kubernetes para visualizar y supervisar las solicitudes del servidor de la API de Kubernetes y las métricas de latencia, etc.d.

> **Important**
>
> Cuando se supera el límite de tamaño de la base de datos, etcd emite una alarma de falta de espacio y deja de recibir más solicitudes de escritura. En otras palabras, el clúster pasa a ser de solo lectura y el servidor de API del clúster rechazará todas las solicitudes para mutar objetos, como la creación de nuevos pods, el escalado de las implementaciones, etc.

Autenticación de

Actualmente, EKS admite dos tipos de autenticación: los tokens de cuentas portadoras/de servicio y la autenticación de IAM, que utiliza la autenticación con token de webhook. Cuando los usuarios llaman a la API de Kubernetes, un webhook pasa a IAM un token de autenticación incluido en la

solicitud. El token, una URL firmada de base 64, lo genera la interfaz de línea de comandos de AWS (AWS CLI).

El usuario o rol de IAM que crea el clúster de EKS obtiene automáticamente acceso total al clúster. Puede administrar el acceso al clúster de EKS editando el mapa de configuración de aws-auth.

Si configura mal el `aws-auth` mapa de configuración y pierde el acceso al clúster, puede seguir utilizando el usuario o el rol del creador del clúster para acceder a su clúster de EKS.

En el improbable caso de que no pueda usar el servicio de IAM en la región de AWS, también puede usar el token portador de la cuenta de servicio de Kubernetes para administrar el clúster.

Cree una `super-admin` cuenta que pueda realizar todas las acciones del clúster:

```
kubectl -n kube-system create serviceaccount super-admin
```

Cree un enlace de roles que asigne al superadministrador el rol de administrador del clúster:

```
kubectl create clusterrolebinding super-admin-rb --clusterrole=cluster-admin --
serviceaccount=kube-system:super-admin
```

Obtén el secreto de la cuenta de servicio:

```
SECRET_NAME=`kubectl -n kube-system get serviceaccount/super-admin -o
 jsonpath='{.secrets[0].name}'`
```

Obtenga el token asociado al secreto:

```
TOKEN=`kubectl -n kube-system get secret $SECRET_NAME -o jsonpath='{.data.token}'|
 base64 --decode`
```

Agrega la cuenta de servicio y el token a`kubeconfig`:

```
kubectl config set-credentials super-admin --token=$TOKEN
```

Defina el contexto actual para usar la cuenta `kubeconfig` de superadministrador:

```
kubectl config set-context --current --user=super-admin
```

La versión final `kubeconfig` debería tener este aspecto:

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data:<REDACTED>
    server: https://<CLUSTER>.gr7.us-west-2.eks.amazonaws.com
  name: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
contexts:
- context:
    cluster: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
    user: super-admin
  name: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
current-context: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
kind: Config
preferences: {}
users:
#- name: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
#  user:
#    exec:
#      apiVersion: client.authentication.k8s.io/v1beta1
#      args:
#      - --region
#      - us-west-2
#      - eks
#      - get-token
#      - --cluster-name
#      - <<cluster name>>
#      command: aws
#      env: null
- name: super-admin
  user:
    token: <<super-admin sa's secret>>
```

Webhooks de admisión

Kubernetes tiene dos tipos de webhooks de admisión: los webhooks de admisión validan y los webhooks de admisión mutantes. Estos permiten al usuario ampliar la API de kubernetes y validar o mutar los objetos antes de que la API los acepte. Una mala configuración de estos webhooks puede desestabilizar el plano de control de EKS al bloquear las operaciones críticas del clúster.

Para evitar afectar a las operaciones críticas del clúster, evite configurar webhooks «generales» como los siguientes:

```
- name: "pod-policy.example.com"
  rules:
  - apiGroups:   ["*"]
    apiVersions: ["*"]
    operations:  ["*"]
    resources:   ["*"]
    scope: "*"
```

O bien, asegúrate de que el webhook tenga una política de apertura por error con un tiempo de espera inferior a 30 segundos para asegurarte de que, si tu webhook no está disponible, no afectará a las cargas de trabajo críticas del clúster.

Bloquee los pods con unsafe **`sysctls`**

`Sysctl`es una utilidad de Linux que permite a los usuarios modificar los parámetros del núcleo durante el tiempo de ejecución. Estos parámetros del núcleo controlan varios aspectos del comportamiento del sistema operativo, como la red, el sistema de archivos, la memoria virtual y la administración de procesos.

Kubernetes permite asignar `sysctl` perfiles a los pods. Kubernetes se clasifica como seguro e inseguro. `systcls` Los espacios de nombres del contenedor o del pod `sysctls` son seguros, y configurarlos no afecta a los demás pods del nodo ni al nodo en sí. Por el contrario, los sysctls no seguros están deshabilitados de forma predeterminada, ya que pueden interrumpir otros pods o hacer que el nodo sea inestable.

Como `sysctls` los inseguros están deshabilitados de forma predeterminada, el kubelet no creará un pod con un perfil inseguro. `sysctl` Si creas un pod de este tipo, el programador los asignará repetidamente a los nodos, mientras que el nodo no podrá lanzarlo. En última instancia, este bucle infinito sobrecarga el plano de control del clúster, lo que hace que el clúster sea inestable.

Considera usar OPA Gatekeeper o Kyverno para rechazar los Pods que no sean seguros. `sysctls`

Gestión de las actualizaciones de clústeres

Desde abril de 2021, el ciclo de lanzamiento de Kubernetes ha cambiado de cuatro versiones al año (una vez por trimestre) a tres versiones al año. Una nueva versión secundaria (como la 1. 21 o 1. 22) se publica aproximadamente cada quince semanas. A partir de la versión 1.19 de Kubernetes, cada versión secundaria recibe soporte durante aproximadamente doce meses después de su lanzamiento por primera vez. Con la llegada de la versión 1.28 de Kubernetes, el sesgo de

compatibilidad entre el plano de control y los nodos de trabajo ha pasado de las versiones n-2 a las n-3 secundarias. Para obtener más información, consulte Prácticas recomendadas para las actualizaciones de clústeres.

Conectividad de terminales de clúster

Al trabajar con Amazon EKS (Elastic Kubernetes Service), es posible que se produzcan errores o tiempos de espera de conexión durante eventos como el escalado del plano de control de Kubernetes o la aplicación de parches. Estos eventos pueden provocar el reemplazo de las instancias de kube-apiserver, lo que podría provocar que se devuelvan direcciones IP diferentes al resolver el FQDN. En este documento, se describen las mejores prácticas para que los consumidores de las API de Kubernetes mantengan una conectividad fiable.

> **Note**
>
> La implementación de estas mejores prácticas puede requerir actualizaciones en las configuraciones o los scripts de los clientes para gestionar las nuevas estrategias de reresolución y reintento del DNS de forma eficaz.

El principal problema se debe al almacenamiento en caché del DNS en el lado del cliente y a la posibilidad de que las direcciones IP de los terminales EKS queden obsoletas: NLB público para los terminales públicos o X-ENI para los terminales privados. Cuando se sustituyen las instancias de kube-apiserver, el nombre de dominio completo (FQDN) puede convertirse en nuevas direcciones IP. Sin embargo, debido a la configuración del tiempo de vida (TTL) del DNS, que se establece en 60 segundos en la zona Route 53 gestionada por AWS, los clientes pueden seguir utilizando direcciones IP anticuadas durante un breve período de tiempo.

Para mitigar estos problemas, los usuarios de las API de Kubernetes (como kubectl, CI/CD pipelines y aplicaciones personalizadas) deberían implementar las siguientes prácticas recomendadas:

- Implemente la reresolución de DNS
- Implemente los reintentos con Backoff y Jitter. Por ejemplo, consulte este artículo titulado Los fallos ocurren
- Implemente los tiempos de espera de los clientes. Establezca los tiempos de espera adecuados para evitar que las solicitudes de larga duración bloqueen su aplicación. Tenga en cuenta que es posible que algunas bibliotecas cliente de Kubernetes, especialmente las generadas

por los generadores de OpenAPI, no permitan configurar fácilmente los tiempos de espera personalizados.

- Ejemplo 1 con kubectl:

```
kubectl get pods --request-timeout 10s # default: no timeout
```

- Ejemplo 2 con Python: el cliente de Kubernetes proporciona un parámetro _request_timeout

Al implementar estas mejores prácticas, puedes mejorar significativamente la confiabilidad y la resiliencia de tus aplicaciones al interactuar con la API de Kubernetes. Recuerda probar estas implementaciones exhaustivamente, especialmente en condiciones de fallo simuladas, para asegurarte de que se comportan como se espera durante los eventos reales de escalado o aplicación de parches.

Ejecutar clústeres de gran tamaño

EKS monitorea activamente la carga en las instancias del plano de control y las escala automáticamente para garantizar un alto rendimiento. Sin embargo, debe tener en cuenta los posibles problemas y límites de rendimiento en Kubernetes y las cuotas en los servicios de AWS cuando ejecute clústeres grandes.

- Según las pruebas realizadas por el equipo, los clústeres con más de 1000 servicios pueden experimentar una latencia de red si se utilizan `kube-proxy` en `iptables` modo in. ProjectCalico La solución es cambiar al `ipvs`modo `kube-proxy` de ejecución.
- También es posible que se limiten las solicitudes de EC2 API si el CNI necesita solicitar direcciones IP para los pods o si necesitas crear nuevas EC2 instancias con frecuencia. Puedes reducir la EC2 API de llamadas configurando el CNI para que almacene en caché las direcciones IP. Puede usar tipos de EC2 instancias más grandes para reducir los eventos de EC2 escalado.

Recursos adicionales:

- Desmitificando las redes de clústeres para los nodos de trabajadores de Amazon EKS
- Control de acceso al punto final del clúster Amazon EKS
- AWS re:Invent 2019: Amazon EKS bajo el capó (-R1) CON421

Plano de datos EKS

Para operar aplicaciones resilientes y de alta disponibilidad, necesita un plano de datos resiliente y de alta disponibilidad. Un plano de datos elástico garantiza que Kubernetes pueda escalar y reparar sus aplicaciones automáticamente. Un plano de datos resiliente consta de dos o más nodos de trabajo, puede crecer y reducirse con la carga de trabajo y recuperarse automáticamente de los fallos.

Con EKS, tiene varias opciones para los nodos de trabajo: nodos gestionados en modo automático de EKS, EC2 instancias y Fargate.

El modo automático de EKS ofrece la ruta más sencilla hacia un plano de datos resiliente. El modo automático extiende la administración de los clústeres de Kubernetes en AWS más allá del propio clúster, para permitir que AWS también configure y administre la infraestructura que permite el buen funcionamiento de sus cargas de trabajo. El modo automático escala automáticamente el plano de datos hacia arriba o hacia abajo a medida que Kubernetes amplía los pods y garantiza continuamente que los nodos del clúster tengan el tamaño adecuado y rentable para las cargas de trabajo que se estén ejecutando actualmente.

Si eliges EC2 instancias, puedes gestionar los nodos de trabajo tú mismo o utilizar los grupos de nodos gestionados por EKS. Puede tener un clúster con una combinación de nodos de trabajo gestionados y autogestionados y Fargate.

Fargate ejecuta cada pod en un entorno informático aislado. Cada pod que se ejecuta en Fargate tiene su propio nodo de trabajo. Fargate escala automáticamente el plano de datos a medida que Kubernetes escala los pods. Puede escalar tanto el plano de datos como su carga de trabajo mediante el escalador automático de cápsulas horizontales.

La forma preferida de escalar los EC2 nodos de trabajo (si no se utiliza el modo automático de EKS, donde AWS lo realiza automáticamente) es mediante grupos de Karpenter, Kubernetes Cluster Autoscaler o Auto Scaling. EC2

Recomendaciones

Distribuya los nodos de trabajo y las cargas de trabajo entre varios AZs

Puede proteger sus cargas de trabajo de los errores en una zona de disponibilidad individual ejecutando nodos de trabajo y pods en varios nodos. AZs Puede controlar la zona de disponibilidad en la que se crean los nodos de trabajo mediante las subredes en las que crea los nodos.

El método recomendado para distribuir los pods AZs es utilizar restricciones de dispersión topológica para los pods. Las funciones de escalado automático, como el modo automático de EKS y Karpenter, conocen las restricciones de dispersión de la topología y lanzan automáticamente los nodos en la dirección correcta AZs para poder cumplir con las restricciones.

La siguiente implementación distribuye los módulos entre sí, AZs si es posible, y permite que esos módulos se ejecuten de todos modos si no es así:

```
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`solo conoce los dominios de topología a través de los nodos que existen con esas etiquetas. Si la implementación anterior se implementa en un clúster con nodos solo en una zona, todos los módulos se programarán en esos nodos, al `kube-scheduler` igual que en las demás zonas. Para que esta distribución de topología funcione según lo

previsto con el programador, los nodos deben existir ya en todas las zonas. La `minDomains` propiedad de las restricciones de dispersión de una topología se utiliza para informar al planificador del número de dominios aptos, incluso si hay un nodo funcionando allí para evitar este problema.

Warning

Si `DoNotSchedule` se establece `whenUnsatisfiable` en, los pods no se podrán programar si no se cumple la restricción de dispersión de la topología. Solo debe configurarse si es preferible que los pods no se ejecuten en lugar de infringir la restricción de dispersión de la topología.

En las versiones anteriores de Kubernetes, puedes usar las reglas de antiafinidad de los pods para programar varios pods en varios. AZs En el siguiente manifiesto se indica al programador de Kubernetes que prefiera programar los pods por separado. 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**
>
> No es necesario que los pods se programen de AZs forma distinta; de lo contrario, la cantidad de pods en una implementación nunca superará la cantidad de. AZs

Asegúrese de poder lanzar nodos en cada zona de disponibilidad cuando utilice volúmenes de EBS

Si utilizas Amazon EBS para proporcionar volúmenes persistentes, debes asegurarte de que los pods y el volumen de EBS asociado estén ubicados en la misma zona de disponibilidad. Un pod no puede acceder a los volúmenes persistentes respaldados por EBS ubicados en una zona de disponibilidad diferente. El programador de Kubernetes sabe en qué zona de disponibilidad se encuentra un nodo trabajador a partir de las etiquetas que aparecen en el nodo y siempre programa un pod que requiera un volumen de EBS en la misma zona de disponibilidad que el volumen. Sin embargo, si no hay nodos de trabajo disponibles en la zona de disponibilidad donde se encuentra el volumen, no se puede programar el pod.

Si utiliza el modo automático EKS o Karpenter, tendrá que asegurarse de seleccionar las subredes en NodeClass cada zona de disponibilidad. Si utiliza grupos de nodos gestionados, debe asegurarse de tener un grupo de nodos en cada zona de disponibilidad.

El modo automático de EKS incluye una capacidad de almacenamiento de EBS, pero si utiliza Karpenter o grupos de nodos gestionados, también será necesario instalar la CSI de EBS.

Utilice el modo automático de EKS para gestionar los nodos de trabajo

El modo automático de EKS agiliza la administración de EKS al proporcionar clústeres listos para la producción con una sobrecarga operativa mínima. El modo automático se encarga de aumentar o reducir el número de nodos en función de los pods que se ejecuten en el clúster. Los nodos

se actualizan automáticamente con parches y correcciones de software, y las actualizaciones se realizan de acuerdo con los ajustes de NodePoolinterrupción configurados y los presupuestos de interrupción de los módulos.

Ejecute el agente de supervisión de nodos

El agente de supervisión de nodos supervisa los problemas de estado de los nodos y reacciona ante ellos publicando los eventos de Kubernetes y actualizando el estado de los nodos. El agente de monitoreo de nodos se incluye en los nodos de modo automático de EKS y se puede instalar como un complemento de EKS para los nodos que no se administran mediante el modo automático.

El modo automático de EKS, los grupos de nodos gestionados y Karpenter tienen la capacidad de detectar las condiciones graves de los nodos notificadas por el agente de supervisión de nodos y reparar esos nodos automáticamente cuando se producen esas condiciones.

Implemente QoS

En el caso de las aplicaciones críticas, considere la posibilidad de definir `requests` = `limits` para el contenedor del pod. Esto garantizará que el contenedor no se destruya si otro pod solicita recursos.

Se recomienda implementar límites de CPU y memoria para todos los contenedores, ya que así se evita que un contenedor consuma recursos del sistema de forma inadvertida y afecte a la disponibilidad de otros procesos ubicados en el mismo lugar.

Configure y dimensione los recursos para todas las cargas de trabajo Requests/Limits

Se pueden aplicar algunas pautas generales para dimensionar las solicitudes de recursos y los límites de las cargas de trabajo:

- No especifique los límites de recursos de la CPU. Si no hay límites, la solicitud actúa como un indicador del tiempo relativo de CPU que ocupan los contenedores. Esto permite que sus cargas de trabajo utilicen toda la CPU sin límites artificiales ni privaciones de recursos.
- Para los recursos que no son de CPU, configuration `requests` = `limits` proporciona el comportamiento más predecible. ¡Si! `requests` `=limits`, también se ha reducido la QOS del contenedor, de garantizada a explotable, por lo que es más probable que se desaloje en caso de presión en el nodo.
- En el caso de los recursos que no sean de la CPU, no especifique un límite que sea mucho mayor que el solicitado. Cuanto mayor sea `limits` la configuración relativa `arequests`, mayor será la

probabilidad de que los nodos estén sobrecargados, lo que aumentará las probabilidades de que se interrumpa la carga de trabajo.

- Las solicitudes del tamaño correcto son particularmente importantes cuando se utiliza una solución de autoescalado de nodos como Karpenter o Cluster. AutoScaler Estas herramientas analizan las solicitudes de carga de trabajo para determinar la cantidad y el tamaño de los nodos que se van a aprovisionar. Si tus solicitudes son demasiado pequeñas y los límites son mayores, es posible que tus cargas de trabajo se desalojen o que la OOM se cancele si se encuentran agrupadas en un nodo.

Determinar las solicitudes de recursos puede resultar difícil, pero herramientas como el escalador automático Vertical Pod pueden ayudarte a «ajustar el tamaño» de las solicitudes al observar el uso de los recursos del contenedor durante el tiempo de ejecución. Otras herramientas que pueden resultar útiles para determinar el tamaño de las solicitudes son:

- Ricitos de oro
- Parca
- Prodfiler
- rsg

Configure las cuotas de recursos para los espacios de nombres

Los espacios de nombres se utilizan en entornos con muchos usuarios distribuidos en varios equipos o proyectos. Proporcionan un margen para los nombres y son una forma de dividir los recursos del clúster entre varios equipos, proyectos y cargas de trabajo. Puedes limitar el consumo total de recursos en un espacio de nombres. El `ResourceQuota`objeto puede limitar por tipo la cantidad de objetos que se pueden crear en un espacio de nombres, así como la cantidad total de recursos informáticos que pueden consumir los recursos de ese proyecto. Puede limitar la suma total de los recursos and/or informáticos de almacenamiento (CPU y memoria) que se pueden solicitar en un espacio de nombres determinado.

Si la cuota de recursos está habilitada para un espacio de nombres para los recursos informáticos, como la CPU y la memoria, los usuarios deben especificar las solicitudes o los límites para cada contenedor de ese espacio de nombres.

Considere la posibilidad de configurar cuotas para cada espacio de nombres. Considere la posibilidad de `LimitRanges` aplicar automáticamente límites preconfigurados a los contenedores dentro de un espacio de nombres.

Limite el uso de los recursos del contenedor dentro de un espacio de nombres

Las cuotas de recursos ayudan a limitar la cantidad de recursos que puede usar un espacio de nombres. El `LimitRange`objeto puede ayudarte a implementar los recursos mínimos y máximos que un contenedor puede solicitar. Al usarlo, `LimitRange` puede establecer una solicitud y límites predeterminados para los contenedores, lo que resulta útil si establecer límites de recursos informáticos no es una práctica estándar en su organización. Como su nombre indica, `LimitRange` puede imponer un uso mínimo y máximo de los recursos de cómputo por pod o contenedor en un espacio de nombres. Además, impone una solicitud de almacenamiento mínima y máxima por cada espacio de PersistentVolumeClaim nombres.

Considere la posibilidad `LimitRange` de usarlo junto con `ResourceQuota` para hacer cumplir los límites tanto a nivel de contenedor como de espacio de nombres. Al establecer estos límites, se garantizará que un contenedor o un espacio de nombres no afecte a los recursos utilizados por otros inquilinos del clúster.

Utilice NodeLocal DNSCache

Puede mejorar el rendimiento del DNS del clúster ejecutando NodeLocalDNSCache. Esta función ejecuta un agente de almacenamiento en caché de DNS en los nodos del clúster como unDaemonSet. Todos los pods utilizan el agente de almacenamiento en caché de DNS que se ejecuta en el nodo para la resolución de nombres en lugar de utilizar `kube-dns` el Servicio. Esta función se incluye automáticamente en el modo automático EKS.

Configurar CoredNS con autoescalado

Otro método para mejorar el rendimiento del DNS del clúster consiste en habilitar el autoscalamiento integrado de los pods de CoreDNS.

Esta función supervisa de forma continua el estado del clúster, incluida la cantidad de nodos y núcleos de CPU. En función de esa información, el controlador adaptará dinámicamente el número de réplicas de la implementación de CoreDNS en un clúster de Amazon EKS.

Prácticas recomendadas para la creación de redes

Es fundamental comprender las redes de Kubernetes para operar el clúster y las aplicaciones de manera eficiente. Las redes de pods, también denominadas redes de clústeres, son el centro de las redes de Kubernetes. Kubernetes admite los complementos de interfaz de red de contenedores (CNI) para redes de clústeres.

Amazon EKS es compatible oficialmente con el complemento CNI Amazon Virtual Private Cloud (VPC) para implementar la red Kubernetes Pod. La VPC CNI proporciona una integración nativa con la VPC de AWS y funciona en modo subyacente. En el modo subyacente, los pods y los hosts se encuentran en la misma capa de red y comparten el espacio de nombres de la red. La dirección IP del pod es coherente desde la perspectiva del clúster y la VPC.

Esta guía presenta la interfaz de red de contenedores de Amazon VPC (VPC CNI) en el contexto de las redes de clústeres de Kubernetes. El CNI de VPC es el complemento de red predeterminado compatible con EKS y, por lo tanto, es el tema central de la guía. La CNI de VPC es altamente configurable para admitir diferentes casos de uso. Además, esta guía incluye secciones dedicadas a diferentes casos de uso, modos de funcionamiento y subcomponentes de la CNI de VPC, seguidas de recomendaciones.

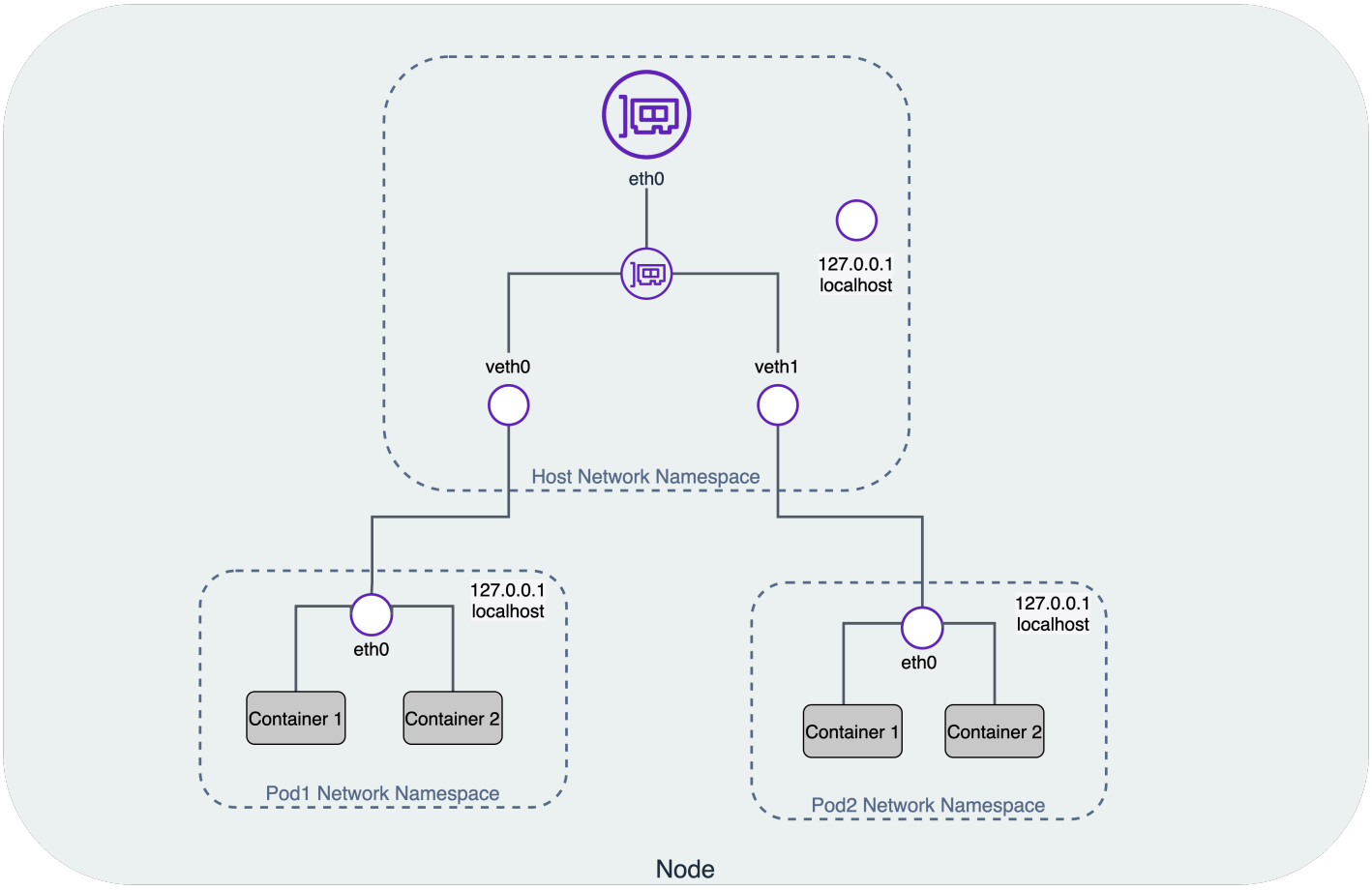
Amazon EKS ejecuta Kubernetes en sentido ascendente y cuenta con la certificación de conformidad con Kubernetes. Si bien puede utilizar complementos CNI alternativos, esta guía no ofrece recomendaciones para administrar otros complementos. CNIs Consulte la documentación del CNI alternativo de EKS para obtener una lista de socios y recursos para gestionar las alternativas CNIs de forma eficaz.

Modelo de red de Kubernetes

Kubernetes establece los siguientes requisitos para las redes de clústeres:

- Los pods programados en el mismo nodo deben poder comunicarse con otros pods sin utilizar la NAT (traducción de direcciones de red).
- Todos los daemons del sistema (procesos en segundo plano, por ejemplo, kubelet) que se ejecutan en un nodo concreto pueden comunicarse con los pods que se ejecutan en el mismo nodo.
- Los pods que usan la red host deben poder contactar con todos los demás pods de todos los demás nodos sin usar NAT.

Consulta el modelo de red de Kubernetes para obtener más información sobre lo que Kubernetes espera de las implementaciones de redes compatibles. En la siguiente figura se ilustra la relación entre los espacios de nombres de la red del pod y el espacio de nombres de la red del host.



Interfaz de red de contenedores (CNI)

Kubernetes admite las especificaciones y los complementos de la CNI para implementar el modelo de red de Kubernetes. Un CNI consta de una especificación (versión actual 1.0.0) y bibliotecas para escribir complementos para configurar las interfaces de red en contenedores, además de una serie de complementos compatibles. La CNI se ocupa únicamente de la conectividad de red de los contenedores y de eliminar los recursos asignados cuando se elimina el contenedor.

El complemento CNI se activa pasando a kubelet la opción de línea de comandos. `--network-plugin=cni` Kubelet lee un archivo desde `--cni-conf-dir` (predeterminado etc/cni/net /.d) y usa la configuración CNI de ese archivo para configurar la red de cada Pod. El archivo de configuración del CNI debe coincidir con la especificación del CNI (versión 0.4.0 como mínimo) y todos los complementos de CNI necesarios a los que haga referencia la configuración deben estar

presentes en el directorio (predeterminado/). `--cni-bin-dir` opt/cni/bin Si hay varios archivos de configuración del CNI en el directorio, el kubelet utiliza el archivo de configuración que aparece primero por su nombre en orden lexicográfico.

Amazon Virtual Private Cloud (VPC) CNI

La VPC CNI proporcionada por AWS es el complemento de red predeterminado para los clústeres de EKS. El complemento CNI de VPC se instala de forma predeterminada al aprovisionar clústeres de EKS. La CNI de VPC se ejecuta en los nodos de trabajo de Kubernetes. El complemento CNI de VPC consta del binario CNI y del complemento de administración de direcciones IP (ipamd). La CNI asigna una dirección IP de la red de VPC a un pod. El ipamd administra las interfaces de red elásticas de AWS (ENIs) en cada nodo de Kubernetes y mantiene la piscina caliente de. IPs La CNI de VPC ofrece opciones de configuración para la preasignación de direcciones IP ENIs y para acelerar los tiempos de inicio del pod. Consulte el CNI de Amazon VPC para obtener información sobre las prácticas recomendadas de administración de complementos.

Amazon EKS recomienda especificar subredes en al menos dos zonas de disponibilidad al crear un clúster. La CNI de Amazon VPC asigna direcciones IP a los pods desde las subredes de los nodos. Recomendamos encarecidamente comprobar las direcciones IP disponibles en las subredes. Tenga en cuenta las recomendaciones de VPC y subred antes de implementar clústeres de EKS.

La CNI de Amazon VPC asigna un conjunto caliente de direcciones IP secundarias ENIs y de la subred conectada al ENI principal del nodo. Este modo de CNI de VPC se denomina modo IP secundario. El número de direcciones IP y, por lo tanto, el número de pods (densidad de pods) se define por el número ENIs y la dirección IP por ENI (límites), según lo definido por el tipo de instancia. El modo secundario es el predeterminado y funciona bien para clústeres pequeños con tipos de instancias más pequeños. Considere la posibilidad de utilizar el modo de prefijo si tiene problemas con la densidad de los pods. También puedes aumentar las direcciones IP disponibles en el nodo para los pods asignando prefijos a. ENIs

El CNI de Amazon VPC se integra de forma nativa con AWS VPC y permite a los usuarios aplicar las prácticas recomendadas de seguridad y redes de VPC de AWS existentes para crear clústeres de Kubernetes. Esto incluye la capacidad de usar registros de flujo de VPC, políticas de enrutamiento de VPC y grupos de seguridad para aislar el tráfico de la red. De forma predeterminada, la CNI de Amazon VPC aplica a los pods el grupo de seguridad asociado al ENI principal del nodo. Considere la posibilidad de habilitar grupos de seguridad para los pods cuando desee asignar reglas de red diferentes a un pod.

De forma predeterminada, la CNI de VPC asigna direcciones IP a los pods de la subred asignada al ENI principal de un nodo. Es habitual experimentar una escasez de IPv4 direcciones cuando se ejecutan clústeres grandes con miles de cargas de trabajo. AWS VPC le permite ampliar la disponibilidad IPs mediante la asignación de una secundaria CIDRs para evitar el agotamiento de los bloques de IPv4 CIDR. El CNI de VPC de AWS le permite usar un rango CIDR de subred diferente para los pods. Esta función de la VPC CNI se denomina red personalizada. Podría considerar la posibilidad de utilizar redes personalizadas para utilizar `100.64.0.0/10` y `198.19.0.0/16` CIDRs (CG-NAT) con EKS. De hecho, esto le permite crear un entorno en el que los pods ya no consuman ninguna RFC1918 dirección IP de su VPC.

Las redes personalizadas son una opción para solucionar el problema del agotamiento de las IPv4 direcciones, pero requieren una sobrecarga operativa. Para resolver este problema, recomendamos utilizar IPv6 clústeres en lugar de redes personalizadas. En concreto, recomendamos migrar a IPv6 clústeres si ha agotado por completo todo el espacio de IPv4 direcciones disponible para su VPC. Evalúe los planes de soporte IPv6 de su organización y considere si invertir en ellos IPv6 puede tener un mayor valor a largo plazo.

El apoyo de EKS IPv6 se centra en resolver el problema de agotamiento de la IP provocado por un espacio de IPv4 direcciones limitado. En respuesta a los problemas de IPv4 agotamiento de los clientes, EKS ha priorizado los pods IPv6 exclusivos en lugar de los pods de doble pila. Es decir, los pods pueden acceder a IPv4 los recursos, pero no se les asigna una IPv4 dirección del rango CIDR de la VPC. La CNI de VPC asigna IPv6 direcciones a los pods del bloque CIDR de VPC administrado por AWS. IPv6

Calculadora de subredes

Este proyecto incluye un documento de Excel sobre la calculadora de subredes. Este documento de calculadora simula el consumo de direcciones IP de una carga de trabajo específica en diferentes opciones de configuración de ENI, como `WARM_IP_TARGET` y. `WARM_ENI_TARGET` El documento incluye dos hojas, una primera para el modo ENI cálido y una segunda para el modo IP cálido. Consulte la guía del CNI de VPC para obtener más información sobre estos modos.

Entradas:

- Tamaño CIDR de subred
- Objetivo ENI cálido o objetivo IP cálido
- Lista de instancias
  - tipo, número y cantidad de módulos de carga de trabajo programados por instancia

Salidas:

- Número total de pods alojados
- Número de subred consumida IPs
- Número de subred restante IPs
- Detalles a nivel de instancia
  - Número de Warm IPs/ENIs por instancia
  - Número de activos IPs/ENIs por instancia

Consideraciones sobre la VPC y la subred

El funcionamiento de un clúster de EKS requiere conocimientos sobre las redes de VPC de AWS, además de las redes de Kubernetes.

Le recomendamos que comprenda los mecanismos de comunicación del plano de control de EKS antes de empezar a diseñar su VPC o a implementar clústeres en los existentes. VPCs

Consulte las consideraciones sobre la VPC en clúster y las consideraciones sobre el grupo de seguridad de Amazon EKS al diseñar una VPC y subredes para utilizarlas con EKS.

Descripción general de

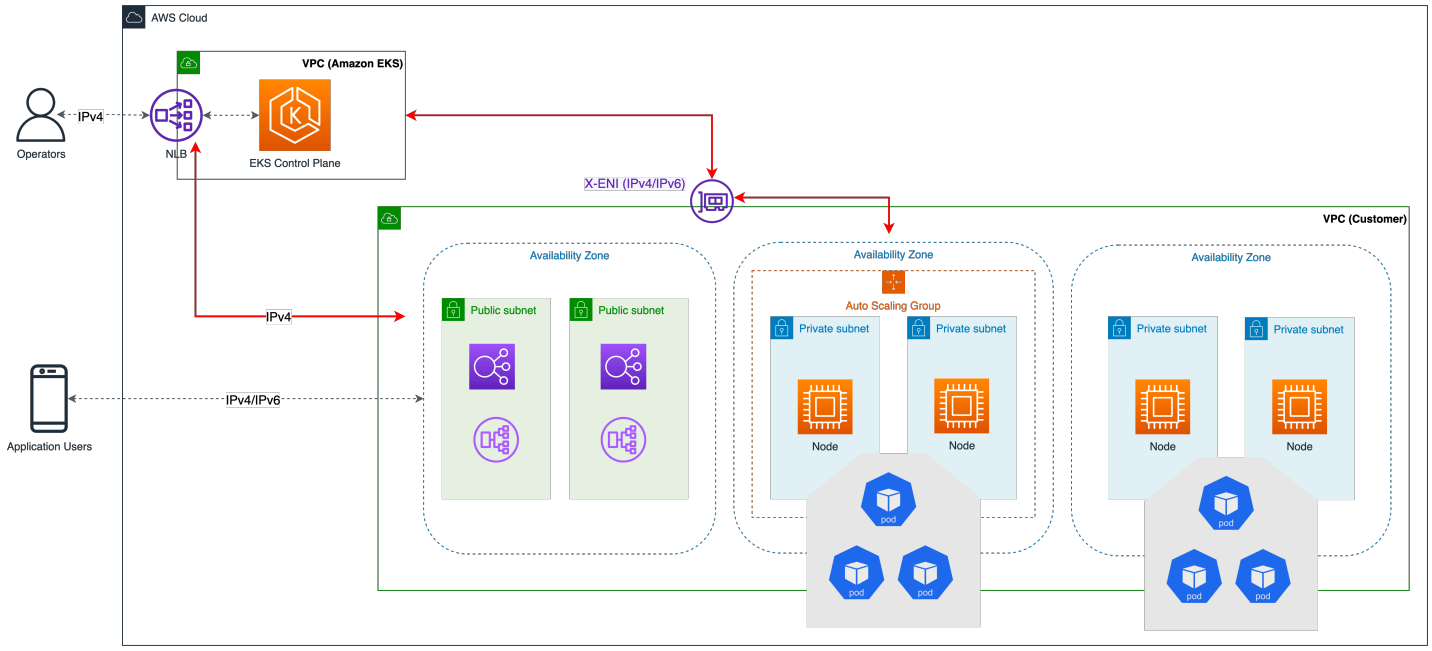
Arquitectura de clústeres de EKS

Un clúster EKS consta de dos VPCs:

- Una VPC gestionada por AWS que aloja el plano de control de Kubernetes. Esta VPC no aparece en la cuenta del cliente.
- Una VPC gestionada por el cliente que aloja los nodos de Kubernetes. Aquí es donde se ejecutan los contenedores, así como otras infraestructuras de AWS administradas por el cliente, como los balanceadores de carga utilizados por el clúster. Esta VPC aparece en la cuenta del cliente. Debe crear una VPC gestionada por el cliente antes de crear un clúster. El eksctl crea una VPC si no la proporciona.

Los nodos de la VPC del cliente necesitan la capacidad de conectarse al punto final del servidor de API administrado en la VPC de AWS. Esto permite que los nodos se registren en el plano de control de Kubernetes y reciban solicitudes para ejecutar pods de aplicaciones.

Los nodos se conectan al plano de control de EKS a través de (a) un terminal público de EKS o (b) una interfaz de red elástica entre cuentas (X-ENI) gestionada por EKS. Cuando se crea un clúster, debes especificar al menos dos subredes de VPC. EKS coloca un X-ENI en cada subred especificada durante la creación del clúster (también denominadas subredes de clúster). El servidor API de Kubernetes usa estas cuentas cruzadas ENIs para comunicarse con los nodos implementados en las subredes de VPC del clúster administradas por el cliente.



Cuando se inicia el nodo, se ejecuta el script de arranque de EKS y se instalan los archivos de configuración del nodo de Kubernetes. Como parte del proceso de arranque de cada instancia, se lanzan los agentes de ejecución del contenedor, kubelet y los agentes de nodo de Kubernetes.

Para registrar un nodo, Kubelet contacta con el punto final del clúster de Kubernetes. Establece una conexión con el punto final público fuera de la VPC o con el punto final privado dentro de la VPC. Kubelet recibe instrucciones de la API y proporciona actualizaciones de estado y latidos al punto final de forma periódica.

Comunicación en el plano de control EKS

EKS tiene dos formas de controlar el acceso al punto final del clúster. El control de acceso al punto final le permite elegir si se puede acceder al punto final desde la Internet pública o solo a través de su VPC. Puede activar el punto final público (que es el predeterminado), el punto final privado o ambos a la vez.

La configuración del punto final de la API del clúster determina la ruta que siguen los nodos para comunicarse con el plano de control. Tenga en cuenta que estos ajustes de punto final se pueden cambiar en cualquier momento a través de la consola o la API de EKS.

Punto final público

Este es el comportamiento predeterminado para los clústeres de Amazon EKS nuevos. Cuando solo está habilitado el punto final público del clúster, las solicitudes de API de Kubernetes que se originan en la VPC del clúster (como la comunicación entre el nodo de trabajo y el plano de control) salen de la VPC, pero no de la red de Amazon. Para que los nodos se conecten al plano de control, deben tener una dirección IP pública y una ruta a una puerta de enlace de Internet o una ruta a una puerta de enlace de NAT, donde puedan usar la dirección IP pública de la puerta de enlace de NAT.

Punto final público y privado

Cuando los puntos de enlace públicos y privados están habilitados, las solicitudes de API de Kubernetes desde la VPC se comunican con el plano de control a través de la X- dentro de la VPC. ENIs Se puede acceder al servidor de la API del clúster desde Internet.

Punto de conexión privado

No hay acceso público a su servidor de API desde Internet cuando solo está habilitado el punto final privado. Todo el tráfico al servidor de la API del clúster debe proceder desde dentro de la VPC de su clúster o de una red conectada. Los nodos se comunican con el servidor API a través de X- ENIs dentro de su VPC. Tenga en cuenta que las herramientas de administración de clústeres deben tener acceso al punto final privado. Obtenga más información sobre cómo conectarse a un punto final de clúster privado de Amazon EKS desde fuera de la VPC de Amazon.

Tenga en cuenta que los servidores DNS públicos resuelven el punto final del servidor API del clúster en una dirección IP privada de la VPC. En el pasado, el punto de conexión se podía resolver desde dentro de la VPC.

Configuraciones de VPC

Soporte y direccionamiento de Amazon VPC IPv4 . IPv6 Amazon EKS es compatible IPv4 de forma predeterminada. Una VPC debe tener un bloque IPv4 CIDR asociado. Si lo desea, puede asociar varios bloques de enrutamiento entre dominios IPv4 sin clase (CIDR) y varios bloques IPv6 CIDR a su VPC. Al crear una VPC, debe especificar un bloque IPv4 CIDR para la VPC de los rangos de IPv4 direcciones privadas que se especifican en el RFC 1918. El tamaño de bloque permitido está comprendido entre un `/16` prefijo (65 536 direcciones IP) y `/28` un prefijo (16 direcciones IP).

Al crear una nueva VPC, puede adjuntar un único bloque de IPv6 CIDR y hasta cinco al cambiar una VPC existente. La longitud del prefijo del tamaño del bloque IPv6 CIDR puede estar entre /44 y /60 y, para las IPv6 subredes, entre /44/ y /64. Puedes solicitar un bloqueo IPv6 CIDR del conjunto de IPv6 direcciones que mantiene Amazon. Consulte la sección de bloques CIDR de VPC de la Guía del usuario de VPC para obtener más información.

Los clústeres de Amazon EKS admiten IPv4 tanto como IPv6. De forma predeterminada, los clústeres EKS utilizan IPv4 IP. Si se especifica IPv6 en el momento de la creación del clúster, se habilitará el uso de IPv6 clústeres. IPv6 los clústeres requieren doble pila VPCs y subredes.

Amazon EKS recomienda utilizar al menos dos subredes que se encuentren en distintas zonas de disponibilidad durante la creación del clúster. Las subredes que se transfieren durante la creación del clúster se conocen como subredes de clúster. Al crear un clúster, Amazon EKS crea hasta 4 cuentas cruzadas (cuenta x o x-ENIs) ENIs en las subredes que especifique. Las x- siempre ENIs se implementan y se utilizan para el tráfico de administración de clústeres, como la entrega de registros, la ejecución y el proxy. Consulte la guía del usuario de EKS para obtener información completa sobre los requisitos de VPC y subred.

Los nodos de trabajo de Kubernetes se pueden ejecutar en las subredes del clúster, pero no se recomienda hacerlo. Durante las actualizaciones del clúster, Amazon EKS aprovisiona más ENIs en las subredes del clúster. Cuando el clúster se amplía, los nodos y pods de trabajo pueden consumir lo disponible IPs en la subred del clúster. Por lo tanto, para asegurarse de que hay suficientes disponibles IPs , puede considerar la posibilidad de utilizar subredes de clúster dedicadas con /28 netmask.

Los nodos de trabajo de Kubernetes pueden ejecutarse en una subred pública o privada. El hecho de que una subred sea pública o privada se refiere a si el tráfico dentro de la subred se enruta a través de una puerta de enlace de Internet. Las subredes públicas tienen una entrada en la tabla de enrutamiento a Internet a través de la puerta de enlace de Internet, pero las subredes privadas no.

El tráfico que se origina en otro lugar y llega a los nodos se denomina entrada. El tráfico que se origina en los nodos y sale de la red se denomina de salida. Los nodos con direcciones IP públicas o elásticas (EIPs) dentro de una subred configurada con una puerta de enlace de Internet permiten la entrada desde fuera de la VPC. Las subredes privadas suelen tener un enrutamiento a una puerta de enlace NAT, que no permite la entrada de tráfico a los nodos de las subredes desde fuera de la VPC y, al mismo tiempo, permite que el tráfico de los nodos salga de la VPC (salida).

En el IPv6 mundo, todas las direcciones se pueden enrutar por Internet. Las IPv6 direcciones asociadas a los nodos y los pods son públicas. Las subredes privadas se admiten mediante la

implementación de una puerta de enlace de Internet de solo salida (EIGW) en una VPC, lo que permite el tráfico saliente y bloquea todo el tráfico entrante. Las prácticas recomendadas para implementar IPv6 subredes se encuentran en la guía del usuario de VPC.

Puede configurar la VPC y las subredes de tres maneras diferentes:

Usar solo subredes públicas

En las mismas subredes públicas, se crean tanto los nodos como los recursos de entrada (como los balanceadores de carga). Etiquete la subred pública con el `kubernetes.io/role/elb`fin de crear balanceadores de carga orientados a Internet. En esta configuración, el punto final del clúster se puede configurar para que sea público, privado o ambos (público y privado).

Uso de subredes públicas y privadas

Los nodos se crean en subredes privadas, mientras que los recursos de Ingress se instancian en subredes públicas. Puede habilitar el acceso público, privado o ambos (público y privado) al punto final del clúster. Según la configuración del punto final del clúster, el tráfico de los nodos ingresará a través de la puerta de enlace NAT o el ENI.

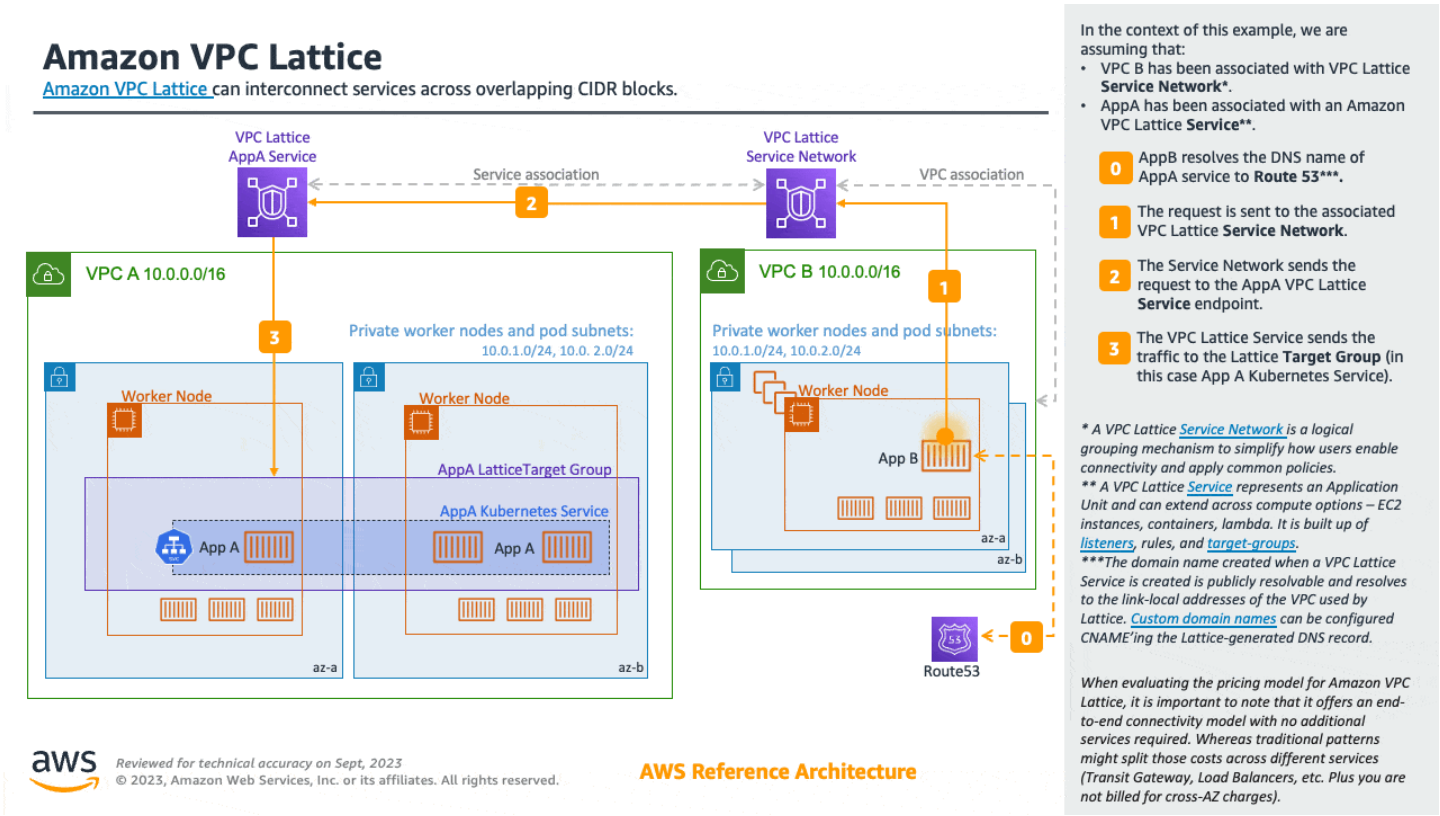
Se utilizarán únicamente subredes privadas

Tanto los nodos como la entrada se crean en subredes privadas. Uso de la etiqueta de `kubernetes.io/role/internal-elb`subred para construir balanceadores de carga internos. Para acceder al punto final del clúster, se necesitará una conexión VPN. Debe activar AWS PrivateLink para EC2 y todos los repositorios de Amazon ECR y S3. Solo debe estar habilitado el punto final privado del clúster. Le sugerimos que consulte los requisitos de los clústeres privados de EKS antes de aprovisionar los clústeres privados.

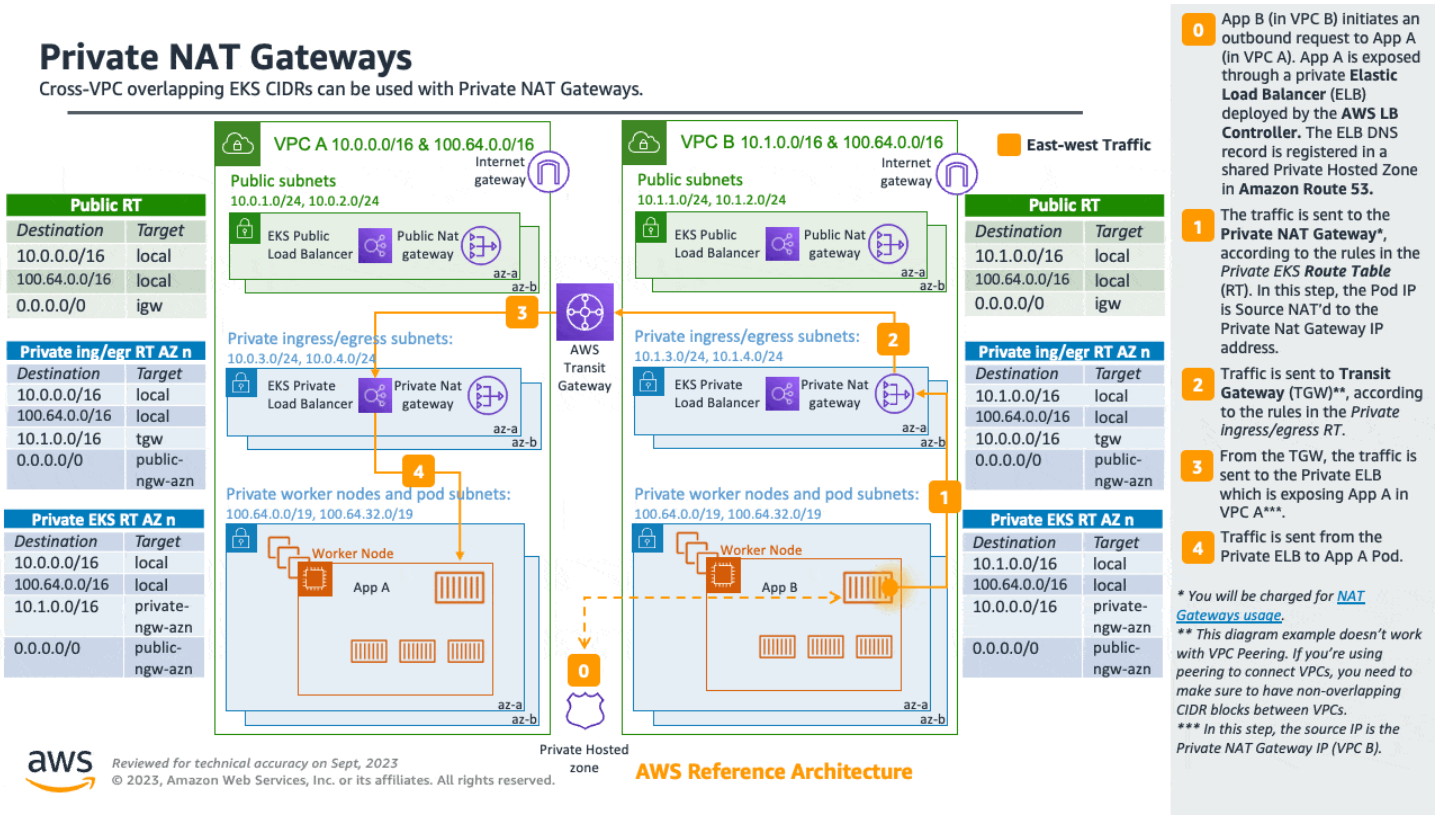
Comunicación entre VPCs

Hay muchos escenarios en los que es necesario implementar clústeres EKS múltiples VPCs e independientes en ellos VPCs.

Puede utilizar Amazon VPC Lattice para conectar servicios de forma coherente y segura entre varias cuentas VPCs AND (sin necesidad de que servicios como VPC peering, AWS o AWS PrivateLink Transit Gateway proporcionen conectividad adicional). Obtenga más información aquí.



Amazon VPC Lattice funciona en el espacio de direcciones locales del enlace IPv4 y IPv6 proporciona conectividad entre los servicios que pueden tener direcciones superpuestas. IPv4 Para lograr una mayor eficiencia operativa, recomendamos encarecidamente implementar clústeres y nodos de EKS en rangos de IP que no se superpongan. En caso de que su infraestructura incluya VPCs rangos de IP superpuestos, necesitará diseñar su red en consecuencia. Sugerimos una puerta de enlace NAT privada o CNI de VPC en modo de red personalizado junto con una puerta de enlace de tránsito para integrar las cargas de trabajo en EKS y resolver los desafíos de CIDR superpuestos y, al mismo tiempo, conservar las direcciones IP enrutables. RFC1918

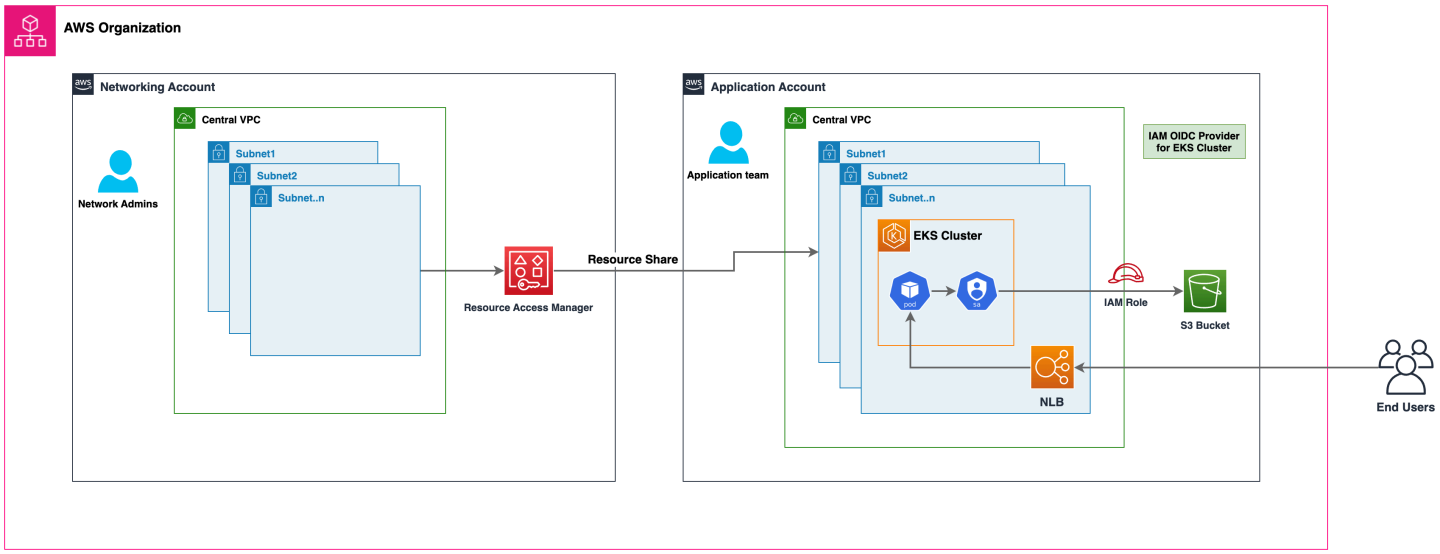


Considere la posibilidad de utilizar AWS PrivateLink, también conocido como servicio de punto final, si usted es el proveedor de servicios y quiere compartir su servicio y entrada de Kubernetes (ya sea ALB o NLB) con la VPC de su cliente en cuentas separadas.

Compartir la VPC en varias cuentas

Muchas empresas adoptaron Amazon compartido VPCs como un medio para agilizar la administración de la red, reducir los costos y mejorar la seguridad en las múltiples cuentas de AWS de una organización de AWS. Utilizan AWS Resource Access Manager (RAM) para compartir de forma segura los recursos de AWS compatibles con cuentas de AWS individuales, unidades organizativas (OUs) o con toda la organización de AWS.

Puede implementar clústeres de Amazon EKS, grupos de nodos gestionados y otros recursos de AWS de apoyo (como grupos de seguridad LoadBalancers, puntos de conexión, etc.) en subredes de VPC compartidas desde otra cuenta de AWS mediante la RAM de AWS. La siguiente figura muestra un ejemplo de arquitectura de alto nivel. Esto permite a los equipos de redes centrales controlar las estructuras de red VPCs, como las subredes, etc., al tiempo que permite a los equipos de aplicaciones o plataformas implementar clústeres de Amazon EKS en sus respectivas cuentas de AWS. Encontrará un recorrido completo de este escenario en este repositorio de GitHub.



Consideraciones a la hora de utilizar subredes compartidas

- Los clústeres y los nodos de trabajo de Amazon EKS se pueden crear en subredes compartidas que forman parte de la misma VPC. Amazon EKS no admite la creación de clústeres en varios VPCs.
- Amazon EKS utiliza los grupos de seguridad de VPC de AWS (SGs) para controlar el tráfico entre el plano de control de Kubernetes y los nodos de trabajo del clúster. Los grupos de seguridad también se utilizan para controlar el tráfico entre los nodos de trabajo y otros recursos de VPC y las direcciones IP externas. Debes crear estos grupos de seguridad en la application/participant cuenta. Asegúrese de que los grupos de seguridad que piensa usar para sus pods también estén ubicados en la cuenta del participante. Puede configurar las reglas de entrada y salida de sus grupos de seguridad para permitir el tráfico necesario hacia y desde los grupos de seguridad ubicados en la cuenta de VPC central.
- Cree las funciones de IAM y las políticas asociadas en la cuenta del participante en la que reside su clúster de Amazon EKS. Estas funciones y políticas de IAM son esenciales para conceder los permisos necesarios a los clústeres de Kubernetes administrados por Amazon EKS, así como a los nodos y pods que se ejecutan en Fargate. Los permisos permiten a Amazon EKS realizar llamadas a otros servicios de AWS en su nombre.
- Puede seguir los siguientes enfoques para permitir el acceso multicuenta a los recursos de AWS, como los buckets de Amazon S3, las tablas de Dynamodb, etc., desde los pods de k8s:
  - Enfoque de políticas basado en recursos: si el servicio de AWS admite políticas de recursos, puede añadir la política basada en recursos adecuada para permitir el acceso entre cuentas a las funciones de IAM asignadas a los pods de kubernetes. En este escenario, el proveedor del

OIDC, las funciones de IAM y las políticas de permisos existen en la cuenta de la aplicación. Para encontrar los servicios de AWS que admiten políticas basadas en recursos, consulte los servicios de AWS que funcionan con IAM y busque los servicios que digan Sí en la columna Basado en recursos.

  - Enfoque de proveedor de OIDC: los recursos de IAM, como el proveedor de OIDC, las funciones de IAM, los permisos y las políticas de confianza, se crearán en la cuenta de AWS de otro participante donde existan los recursos. Estas funciones se asignarán a los pods de Kubernetes en la cuenta de la aplicación para que puedan acceder a los recursos de varias cuentas. Consulte el blog Funciones de IAM entre cuentas para cuentas de servicio de Kubernetes para obtener una descripción completa de este enfoque.

- Puede implementar los recursos de Amazon Elastic Loadbalancer (ELB) (ALB o NLB) para enrutar el tráfico a los pods de k8s, ya sea en las cuentas de aplicaciones o de redes centrales. Consulte el tutorial Expose Amazon EKS Pods Through Cross-Account Load Balancer para obtener instrucciones detalladas sobre cómo implementar los recursos del ELB en la cuenta de red central. Esta opción ofrece una mayor flexibilidad, ya que otorga a la cuenta de Central Networking el control total sobre la configuración de seguridad de los recursos del Load Balancer.
- Al utilizar el CNI `custom networking feature` de Amazon VPC, debe utilizar las asignaciones de ID de zona de disponibilidad (AZ) que figuran en la cuenta de red central para crear cada una de ellas. `ENIConfig` Esto se debe a la asignación aleatoria de los nombres físicos AZs a los de las zonas de disponibilidad de cada cuenta de AWS.

Grupos de seguridad

Un grupo de seguridad controla el tráfico al que se permite llegar y dejar los recursos a los que está asociado. Amazon EKS usa grupos de seguridad para administrar la comunicación entre el plano de control y los nodos. Al crear un clúster, Amazon EKS crea un grupo de seguridad denominado `eks-cluster-sg-my-cluster-uniqueID`. EKS asocia estos grupos de seguridad a los nodos gestionados ENIs y a los nodos. Las reglas predeterminadas permiten que todo el tráfico fluya libremente entre el clúster y los nodos, y permite que todo el tráfico saliente llegue a cualquier destino.

Al crear un clúster, puede especificar sus propios grupos de seguridad. Consulte la recomendación de grupos de seguridad cuando especifique sus propios grupos de seguridad.

Recomendaciones

Considere la posibilidad de implementar Multi-AZ

Las regiones de AWS proporcionan varias zonas de disponibilidad (AZ) físicamente separadas y aisladas, que se conectan mediante redes de baja latencia, alto rendimiento y alta redundancia. Con las zonas de disponibilidad, puede diseñar y operar aplicaciones que conmuten automáticamente entre zonas de disponibilidad sin interrupciones. Amazon EKS recomienda encarecidamente implementar clústeres de EKS en varias zonas de disponibilidad. Tenga en cuenta la posibilidad de especificar subredes en al menos dos zonas de disponibilidad al crear el clúster.

Kubelet, que se ejecuta en los nodos, añade automáticamente etiquetas al objeto del nodo, por ejemplo. `topology.kubernetes.io/region=us-west-2` Recomendamos usar etiquetas de nodos junto con las restricciones de dispersión de la topología de los pods para controlar la distribución de los pods en las zonas. Estas sugerencias permiten al programador de Kubernetes colocar los pods según la disponibilidad prevista, lo que reduce el riesgo de que una falla correlacionada afecte a toda la carga de trabajo. Consulta Cómo asignar nodos a los pods para ver ejemplos del selector de nodos y de las restricciones de dispersión de zonas de disponibilidad.

Puedes definir las subredes o las zonas de disponibilidad al crear los nodos. Los nodos se colocan en subredes de clúster si no hay subredes configuradas. El soporte de EKS para los grupos de nodos gestionados distribuye automáticamente los nodos entre varias zonas de disponibilidad según la capacidad disponible. Karpenter respetará la ubicación de la distribución de la zona de disponibilidad mediante el escalado de los nodos para especificar AZs si las cargas de trabajo definen los límites de dispersión de la topología.

Los AWS Elastic Load Balancer Controller administran los balanceadores de carga de AWS para un clúster de Kubernetes. Proporciona un Application Load Balancer (ALB) para los recursos de entrada de Kubernetes y un Network Load Balancer (NLB) para los servicios de Kubernetes del tipo Loadbalancer. El controlador Elastic Load Balancer utiliza etiquetas para detectar las subredes. El controlador ELB requiere un mínimo de dos zonas de disponibilidad (AZs) para aprovisionar correctamente el recurso de entrada. Considere configurar las subredes en al menos dos AZs para aprovechar la seguridad y la confiabilidad de la redundancia geográfica.

Implemente nodos en subredes privadas

Una VPC que incluya subredes públicas y privadas es el método ideal para implementar cargas de trabajo de Kubernetes en EKS. Considere la posibilidad de establecer un mínimo de dos subredes

públicas y dos subredes privadas en dos zonas de disponibilidad distintas. La tabla de rutas relacionada de una subred pública contiene una ruta a una puerta de enlace a Internet. Los pods pueden interactuar con Internet a través de una puerta de enlace NAT. Las subredes privadas son compatibles con las pasarelas de Internet del entorno (EIGW) que solo permiten la IPv6 salida.

La creación de instancias de nodos en subredes privadas ofrece el máximo control sobre el tráfico hacia los nodos y resulta eficaz para la gran mayoría de las aplicaciones de Kubernetes. Los recursos de entrada (como los balanceadores de carga) se instancian en las subredes públicas y dirigen el tráfico a los pods que funcionan en subredes privadas.

Considera el modo solo privado si exiges una seguridad y un aislamiento estrictos de la red. En esta configuración, se implementan tres subredes privadas en distintas zonas de disponibilidad dentro de la VPC de la región de AWS. Los recursos implementados en las subredes no pueden acceder a Internet, ni Internet puede acceder a los recursos de las subredes. Para que su aplicación de Kubernetes pueda acceder a otros servicios de AWS, debe configurar los puntos de enlace de PrivateLink las interfaces and/or . Puede configurar balanceadores de carga internos para redirigir el tráfico a los pods mediante AWS Load Balancer Controller. Las subredes privadas deben estar etiquetadas (`kubernetes.io/role/internal-elb: 1`) para que el controlador pueda aprovisionar los balanceadores de carga. Para que los nodos se registren en el clúster, el punto final del clúster debe estar configurado en modo privado. Consulte la guía de clústeres privados para conocer todos los requisitos y consideraciones.

Considere el modo público y privado para el punto final del clúster

Amazon EKS ofrece modos de punto de public-and-private conexión de clúster solo públicos y privados. El modo predeterminado es solo público, sin embargo, recomendamos configurar el punto final del clúster en modo público y privado. Esta opción permite que las llamadas a la API de Kubernetes dentro de la VPC del clúster (como la node-to-control-plane comunicación) utilicen el punto de enlace de la VPC privada y el tráfico permanezca dentro de la VPC del clúster. Por otro lado, puedes acceder al servidor de API de tu clúster desde Internet. Sin embargo, recomendamos encarecidamente limitar los bloques de CIDR que pueden utilizar el punto final público. Aprenda a configurar el acceso a los terminales públicos y privados, incluida la limitación de los bloques de CIDR.

Le sugerimos un terminal exclusivamente privado cuando necesite seguridad y aislamiento de la red. Recomendamos utilizar cualquiera de las opciones que figuran en la guía del usuario de EKS para conectarse a un servidor API de forma privada.

Configure los grupos de seguridad con cuidado

Amazon EKS admite el uso de grupos de seguridad personalizados. Todos los grupos de seguridad personalizados deben permitir la comunicación entre los nodos y el plano de control de Kubernetes. Compruebe los requisitos de los puertos y configure las reglas manualmente cuando su organización no permita la comunicación abierta.

EKS aplica los grupos de seguridad personalizados que usted proporciona durante la creación del clúster a las interfaces administradas (X-ENIs). Sin embargo, no los asocia inmediatamente a los nodos. Al crear grupos de nodos, se recomienda encarecidamente asociar los grupos de seguridad personalizados de forma manual. Considere la posibilidad de activar securityGroupSelectorlos Términos para permitir la detección de grupos de seguridad personalizados mediante plantillas de nodos de Karpenter durante el escalado automático de los nodos.

Recomendamos encarecidamente crear un grupo de seguridad para permitir todo el tráfico de comunicación entre nodos. Durante el proceso de arranque, los nodos requieren conectividad a Internet saliente para acceder al punto final del clúster. Evalúe los requisitos de acceso saliente, como la conexión local y el acceso al registro del contenedor, y establezca las reglas adecuadas. Antes de poner los cambios en producción, le recomendamos encarecidamente que compruebe detenidamente las conexiones en su entorno de desarrollo.

Implemente pasarelas NAT en cada zona de disponibilidad

Si implementa nodos en subredes privadas (IPv4 y IPv6), considere la posibilidad de crear una puerta de enlace NAT en cada zona de disponibilidad (AZ) para garantizar una arquitectura independiente de la zona y reducir los gastos entre zonas de disponibilidad. Cada puerta de enlace NAT de una zona de disponibilidad se implementa con redundancia.

CNI de Amazon VPC

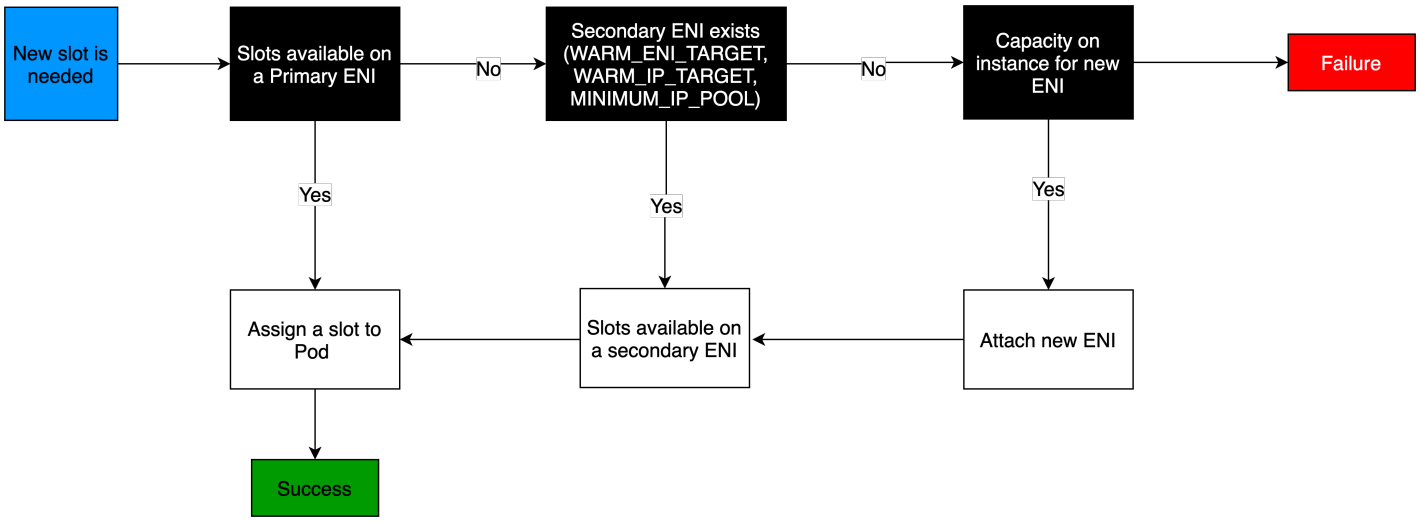
Amazon EKS implementa redes de clústeres mediante el complemento Amazon VPC Container Network Interface, también conocido como VPC CNI. El complemento CNI permite que los pods de Kubernetes tengan la misma dirección IP que tienen en la red de VPC. Más específicamente, todos los contenedores del pod comparten un espacio de nombres de red y pueden comunicarse entre sí mediante puertos locales.

El CNI de Amazon VPC tiene dos componentes:

- CNI Binary, que configurará la red Pod para permitir la comunicación. Pod-to-Pod El binario CNI se ejecuta en el sistema de archivos raíz de un nodo y el kubelet lo invoca cuando se añade un nuevo pod o se elimina un pod existente del nodo.
- ipamd, un daemon de administración de direcciones IP (IPAM) local de nodos que lleva mucho tiempo funcionando y es responsable de:
  - administrar en un nodo, y ENIs
  - mantener un conjunto cálido de direcciones IP o prefijos disponibles

Cuando se crea una instancia, EC2 crea y adjunta un ENI principal asociado a una subred principal. La subred principal puede ser pública o privada. Los pods que se ejecutan en el modo HostNetwork utilizan la dirección IP principal asignada al ENI principal del nodo y comparten el mismo espacio de nombres de red que el host.

El complemento CNI administra las interfaces de red elásticas (ENI) en el nodo. Cuando se aprovisiona un nodo, el complemento CNI asigna automáticamente un conjunto de ranuras (IPs o prefijos) de la subred del nodo a la ENI principal. Este grupo se conoce como grupo cálido y su tamaño viene determinado por el tipo de instancia del nodo. Según la configuración del CNI, una ranura puede ser una dirección IP o un prefijo. Cuando se ha asignado una ranura a un ENI, el CNI puede conectar ranuras adicionales ENIs con un conjunto cálido de ranuras a los nodos. Estos adicionales ENIs se denominan secundarios ENIs. Cada ENI solo puede admitir un número determinado de ranuras, según el tipo de instancia. El CNI asocia más ENIs a las instancias en función de la cantidad de ranuras necesarias, que normalmente corresponde a la cantidad de pods. Este proceso continúa hasta que el nodo ya no pueda admitir más ENI. El CNI también preasigna los espacios «calientes» ENIs y los espacios para que los Pod se inicien más rápidamente. Ten en cuenta que cada tipo de instancia tiene un número máximo de instancias ENIs que se pueden adjuntar. Esta es una de las limitaciones de la densidad de pods (número de pods por nodo), además de los recursos de cómputo.



La cantidad máxima de interfaces de red y la cantidad máxima de ranuras que puede utilizar varían según el tipo de instancia. EC2 Como cada pod consume una dirección IP en una ranura, la cantidad de pods que puede ejecutar en una EC2 instancia concreta depende del número de pods que se le ENIs puedan conectar y del número de ranuras que admita cada ENI. Te sugerimos configurar el número máximo de pods por cada guía del usuario de EKS para evitar que se agoten los recursos de CPU y memoria de la instancia. El uso de pods no `hostNetwork` se incluye en este cálculo. Puedes considerar usar un script llamado max-pod-calculator.sh para calcular el número máximo de pods recomendado por EKS para un tipo de instancia determinado.

Descripción general de

El modo IP secundario es el modo predeterminado para la CNI de VPC. Esta guía proporciona una descripción general genérica del comportamiento de la CNI de la VPC cuando el modo IP secundaria está habilitado. La funcionalidad de ipamd (asignación de direcciones IP) puede variar en función de los ajustes de configuración de la CNI de la VPC, the section called “Modo de prefijo para Linux” comothe section called “Grupos de seguridad por pod”, y. the section called “Redes personalizadas”

La CNI de Amazon VPC se implementa como un Daemonset de Kubernetes denominado aws-node en los nodos de trabajo. Cuando se aprovisiona un nodo de trabajo, se le adjunta un ENI predeterminado, denominado ENI principal. El CNI asigna un conjunto caliente de direcciones IP secundarias ENIs y de la subred conectada al ENI principal del nodo. De forma predeterminada, ipamd intenta asignar un ENI adicional al nodo. El IPAMD asigna un ENI adicional cuando se programa un solo pod y se le asigna una dirección IP secundaria del ENI principal. Este ENI «cálido» permite una conexión en red de Pod más rápida. A medida que se agota el conjunto de direcciones IP secundarias, el CNI añade otro ENI para asignar más.

El número ENIs y las direcciones IP de un grupo se configuran mediante variables de entorno denominadas WARM_ENI_TARGET, WARM_IP_TARGET y MINIMUM_IP_TARGET. El Daemonset comprobará periódicamente si hay un número suficiente de ellas conectadas. `aws-node` ENIs Se adjuntará un número suficiente de ellos cuando se cumplan `WARM_ENI_TARGET` todas `WARM_IP_TARGET` las `MINIMUM_IP_TARGET` condiciones o. ENIs Si no hay suficientes ENIs adjuntos, el CNI realizará una llamada a la API para adjuntar más EC2 datos hasta que se alcance `MAX_ENI` el límite.

- `WARM_ENI_TARGET`- Número entero, los valores superiores a 0 indican que el requisito está activado
  - El número de puntos calientes ENIs que se deben mantener. Un ENI está «caliente» cuando está conectado como ENI secundario a un nodo, pero ningún Pod lo utiliza. Más específicamente, no se ha asociado ninguna dirección IP del ENI a un pod.
  - Ejemplo: considere una instancia con 2 direcciones IP ENIs, cada ENI admite 5 direcciones IP. WARM_ENI_TARGET está establecido en 1. Si hay exactamente 5 direcciones IP asociadas a la instancia, el CNI mantiene 2 ENIs conectadas a la instancia. El primer ENI está en uso y se utilizan las 5 direcciones IP posibles de este ENI. El segundo ENI es «cálido» y contiene las 5 direcciones IP agrupadas. Si se lanza otro pod en la instancia, se necesitará una sexta dirección IP. El CNI asignará a este sexto pod una dirección IP del segundo ENI y una dirección IP 5 IPs del grupo. El segundo ENI ya está en uso y ya no está en estado «caliente». La CNI asignará una tercera ENI para mantener al menos una ENI cálida.

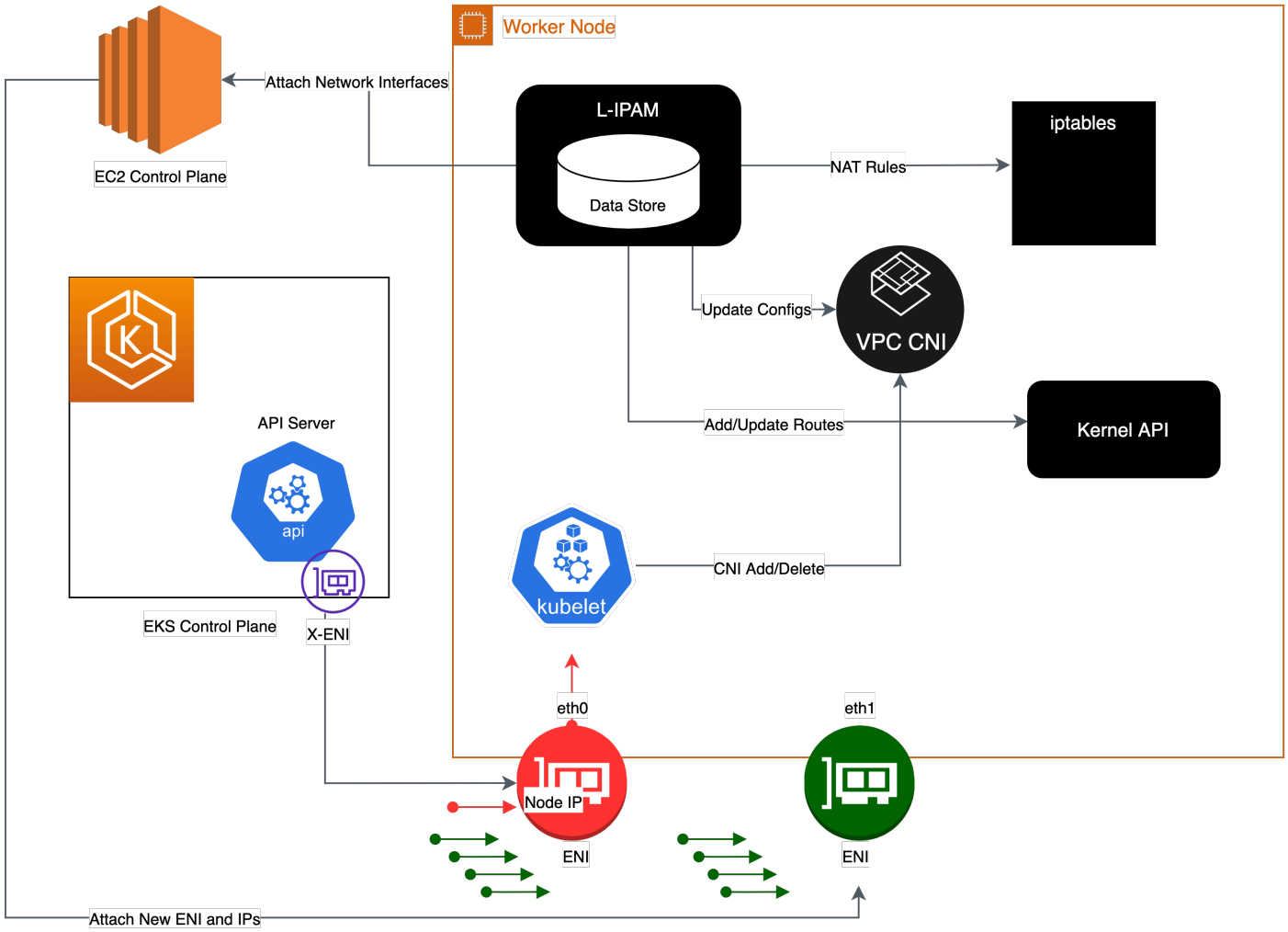
> **Note**
>
> El warm ENIs sigue consumiendo direcciones IP del CIDR de su VPC. Las direcciones IP están «no utilizadas» o «en estado caliente» hasta que se asocian a una carga de trabajo, como un pod.

- `WARM_IP_TARGET`, Entero, los valores superiores a 0 indican que el requisito está activado
  - El número de direcciones IP cálidas que se van a mantener. Hay una IP cálida disponible en un ENI conectado activamente, pero no se ha asignado a un pod. En otras palabras, el número de Warm IPs disponible es el número IPs que se puede asignar a un Pod sin necesidad de un ENI adicional.
- Ejemplo: considere una instancia con 1 ENI, cada ENI admite 20 direcciones IP. WARM_IP_TARGET está establecido en 5. WARM_ENI_TARGET está establecido en 0. Solo

se adjuntará 1 ENI hasta que se necesite una decimosexta dirección IP. A continuación, el CNI adjuntará un segundo ENI, que consumirá 20 direcciones posibles de la subred CIDR.

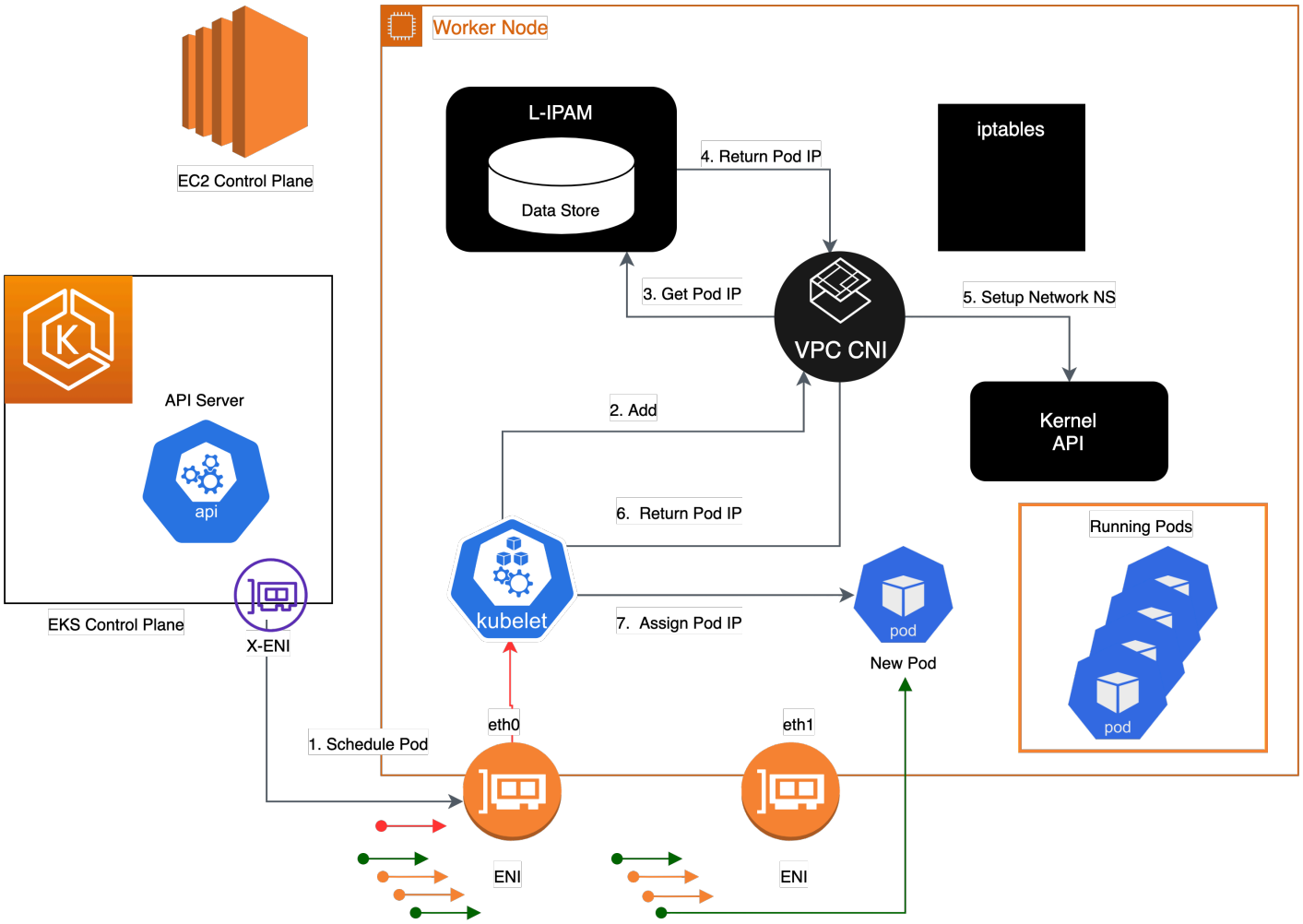
- `MINIMUM_IP_TARGET`, Entero, los valores superiores a 0 indican que el requisito está activado
  - El número mínimo de direcciones IP que se van a asignar en cualquier momento. Por lo general, se usa para anticipar la asignación de varias ENIs al lanzar la instancia.
  - Ejemplo: pensemos en una instancia recién lanzada. Tiene 1 ENI y cada ENI admite 10 direcciones IP. MINIMUM_IP_TARGET está establecido en 100. El ENI adjunta inmediatamente 9 más ENIs para un total de 100 direcciones. Esto ocurre independientemente de los valores WARM_IP_TARGET o WARM_ENI_TARGET.

Este proyecto incluye un documento de Excel con una calculadora de subredes. Este documento de calculadora simula el consumo de direcciones IP de una carga de trabajo específica en diferentes opciones de configuración de ENI, como `WARM_IP_TARGET` y. `WARM_ENI_TARGET`

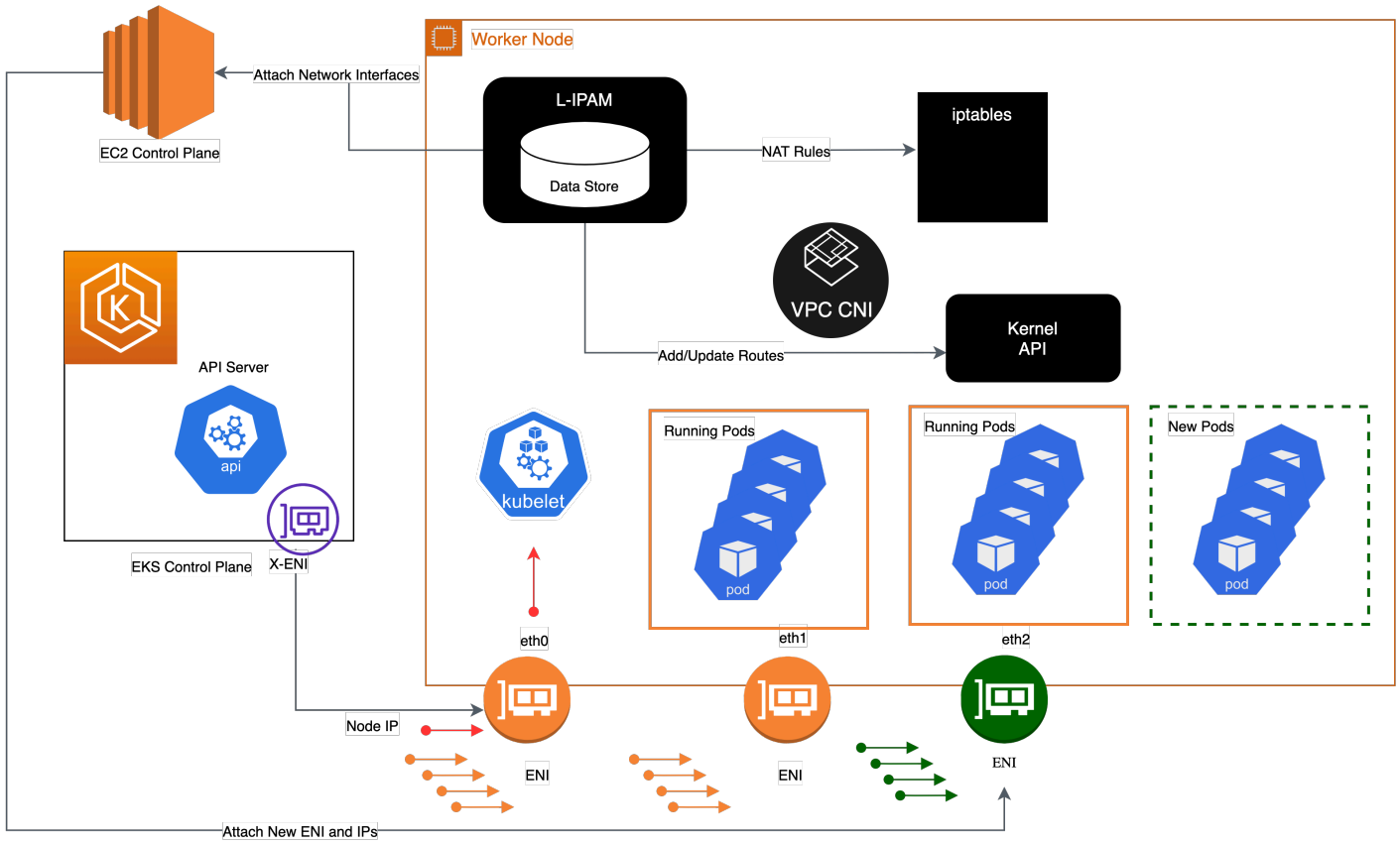


Cuando Kubelet recibe una solicitud para añadir un pod, el binario CNI consulta a ipamd para obtener una dirección IP disponible, que luego ipamd proporciona al pod. El binario CNI conecta el host y la red del Pod.

Los pods desplegados en un nodo se asignan, de forma predeterminada, a los mismos grupos de seguridad que el ENI principal. Como alternativa, los pods se pueden configurar con diferentes grupos de seguridad.



A medida que se agota el grupo de direcciones IP, el complemento asocia automáticamente otra interfaz de red elástica a la instancia y asigna otro conjunto de direcciones IP secundarias a esa interfaz. Este proceso continúa hasta que el nodo ya no puede admitir interfaces de red elásticas adicionales.



Cuando se elimina un pod, la CNI de la VPC coloca la dirección IP del pod en una memoria caché de enfriamiento de 30 segundos. La caché IPs de enfriamiento no se asigna a los nuevos pods. Cuando finaliza el período de enfriamiento, VPC CNI devuelve el Pod IP a la piscina caliente. El período de reflexión evita que las direcciones IP de los pods se reciclen prematuramente y permite que kube-proxy termine de actualizar las reglas de iptables en todos los nodos del clúster. Cuando el número de configuraciones de piscina caliente IPs o lo ENIs supera, el complemento ipamd regresa ENIs a IPs la VPC.

Como se describió anteriormente en el modo de IP secundaria, cada pod recibe una dirección IP privada secundaria de una de las direcciones ENIs conectadas a una instancia. Como cada pod usa una dirección IP, la cantidad de pods que puedes ejecutar en una EC2 instancia concreta depende del número de pods que se le ENIs puedan conectar y del número de direcciones IP que admita. La CNI de la VPC comprueba el archivo de límites para averiguar cuántas direcciones IP ENIs y cuántas direcciones IP están permitidas para cada tipo de instancia.

Puedes usar la siguiente fórmula para determinar la cantidad máxima de pods que puedes implementar en un nodo.

```
(Number of network interfaces for the instance type * (the number of IP addresses per
 network interface - 1)) + 2
```

El +2 indica los pods que requieren una red de host, como kube-proxy y VPC CNI. Amazon EKS requiere que kube-proxy y VPC CNI funcionen en cada nodo, y estos requisitos se tienen en cuenta en el valor de max-pods. Si desea ejecutar pods de red de host adicionales, considere actualizar el valor de los max-pods. Puede especificarlos `--kubelet-extra-args "–max-pods=110"` como datos de usuario en la plantilla de lanzamiento.

Por ejemplo, en un clúster con 3 nodos c5.large (3 ENIs y un máximo de 10 IPs por ENI), cuando el clúster se inicie y tenga 2 pods de CoredNS, el CNI consumirá 50 direcciones IP y mantendrá 43 en una piscina caliente. IPs La piscina caliente permite que los Pod se inicien más rápidamente cuando se implementa la aplicación.

Nodo 1 (con pod CoredNS): ENIs 2, 20 asignados IPs

Nodo 2 (con pod CoredNS): ENIs 2, 20 asignados IPs

Nodo 3 (sin pod): 1 ENI. 10 IPs asignados.

Para el nodo 1 y el nodo 2 (configuración idéntica):

- 2 ENIs × 10 IPs por ENI = 20 en IPs total
- Reste 2 primarios IPs (1 por ENI) = 18 IPs
- Reste 1 IP para el pod de CoredNS = 17 disponibles IPs
- Por lo tanto, cada uno de estos nodos tiene 17 IPs en una piscina caliente

Para el nodo 3:

- 1 ENI × 10 IPs = 10 en IPs total
- Reste 1 IP principal = 9 IPs disponibles en piscina caliente

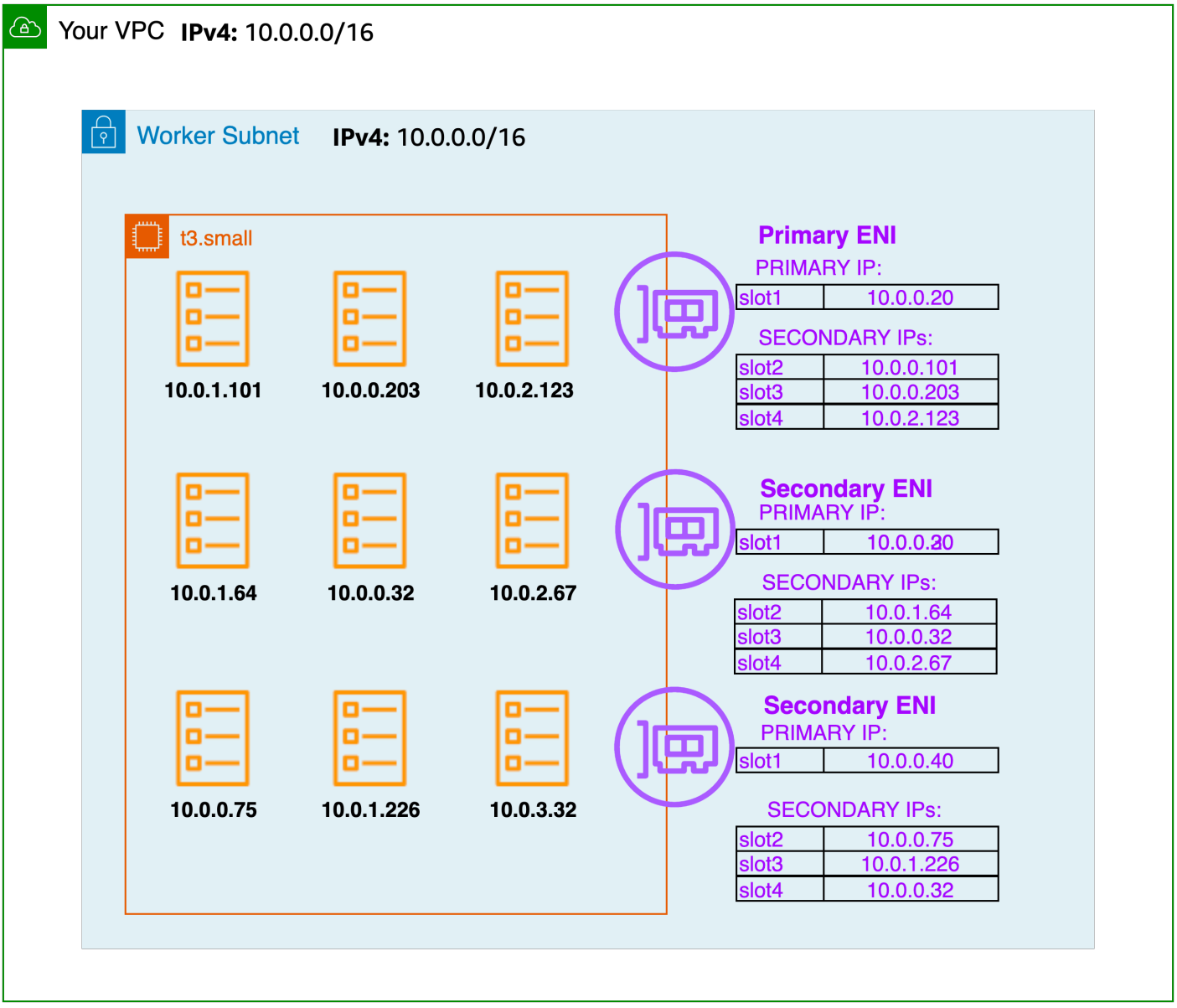
Cálculo de la piscina caliente total: - 17 (nodo 1) + 17 (nodo 2) + 9 (nodo 3) = 43 IPs

Tenga en cuenta que cada uno de los módulos de infraestructura, que suelen funcionar como conjuntos de demonios, contribuye al número máximo de pods. Estos pueden incluir:

- CoreDNS

- Amazon Elastic LoadBalancer
- Módulos operativos para el servidor de métricas

Le sugerimos que planifique su infraestructura combinando las capacidades de estos Pods. Para ver una lista del número máximo de pods compatibles con cada tipo de instancia, consulta el archivo eni-max-Pods.txt en GitHub.



Recomendaciones

Implemente el clúster EKS con el modo automático

Cuando usa el modo automático de EKS para crear un clúster, AWS administra la configuración de la interfaz de red de contenedores (CNI) de VPC para su clúster. Con el modo automático de Amazon EKS, no necesita instalar ni actualizar los complementos de red. Sin embargo, asegúrese de que sus cargas de trabajo sean compatibles con la configuración CNI de VPC gestionada.

Implemente el complemento gestionado por VPC CNI

Al aprovisionar un clúster, Amazon EKS instala la CNI de VPC automáticamente. No obstante, Amazon EKS admite complementos administrados que permiten que el clúster interactúe con los recursos subyacentes de AWS, como la informática, el almacenamiento y las redes. Le recomendamos encarecidamente que implemente clústeres con complementos gestionados, incluido el CNI de VPC.

El complemento gestionado de Amazon EKS ofrece la instalación y administración de CNI de VPC para clústeres de Amazon EKS. Los complementos de Amazon EKS incluyen los últimos parches de seguridad y correcciones de errores, y AWS los valida para que funcionen con Amazon EKS. El complemento CNI de VPC le permite garantizar de forma continua la seguridad y la estabilidad de sus clústeres de Amazon EKS y reducir el esfuerzo necesario para instalar, configurar y actualizar los complementos. Además, se puede añadir, actualizar o eliminar un complemento gestionado mediante la API de Amazon EKS, la consola de administración de AWS, la CLI de AWS y eksctl.

Puede encontrar los campos gestionados del CNI de la VPC utilizando `--show-managed-fields` flag con el comando. `kubectl get`

```
kubectl get daemonset aws-node --show-managed-fields -n kube-system -o yaml
```

Los complementos administrados evitan la desviación de la configuración al sobrescribir automáticamente las configuraciones cada 15 minutos. Esto significa que cualquier cambio en los complementos gestionados que se realice a través de la API de Kubernetes tras la creación del complemento se sobrescribirá mediante el proceso automatizado de prevención de desviaciones y también se establecerá en los valores predeterminados durante el proceso de actualización del complemento.

Los campos gestionados por EKS aparecen en ManagedFields y el administrador es EKS. Los campos gestionados por EKS incluyen la cuenta de servicio, la imagen, la URL de la imagen, la

sonda de actividad, la sonda de disponibilidad, las etiquetas, los volúmenes y los montajes de volúmenes.

> **Note**
>
> Los campos que se utilizan con más frecuencia, como WARM_ENI_TARGET, WARM_IP_TARGET y MINIMUM_IP_TARGET, no se administran y no se conciliarán. Los cambios en estos campos se mantendrán al actualizar el complemento.

Te sugerimos que pruebes el comportamiento del complemento en los clústeres que no son de producción para una configuración específica antes de actualizar los clústeres de producción. Además, siga los pasos de la guía del usuario de EKS para las configuraciones complementarias.

Migre a Managed Add-On

Administrará la compatibilidad de las versiones y actualizará los parches de seguridad de la CNI de VPC autogestionada. Para actualizar un complemento autogestionado, debe utilizar Kubernetes APIs y las instrucciones descritas en la guía del usuario de EKS. Le recomendamos migrar a un complemento gestionado para los clústeres de EKS existentes y le recomendamos encarecidamente que cree una copia de seguridad de su configuración actual de CNI antes de la migración. Para configurar los complementos administrados, puede utilizar la API Amazon EKS, la consola de administración de AWS o la interfaz de línea de comandos de AWS.

```
kubectl apply view-last-applied daemonset aws-node -n kube-system  aws-k8s-cni-old.yaml
```

Amazon EKS sustituirá los ajustes de configuración del CNI si el campo aparece como gestionado con los ajustes predeterminados. Le recomendamos que no modifique los campos administrados. El complemento no reconcilia los campos de configuración, como las variables de entorno cálido y los modos CNI. Los pods y las aplicaciones seguirán ejecutándose mientras se migra a una CNI gestionada.

Backup de la configuración del CNI antes de la actualización

La CNI de VPC se ejecuta en el plano de datos del cliente (nodos) y, por lo tanto, Amazon EKS no actualiza automáticamente el complemento (gestionado y autogestionado) cuando se publican nuevas versiones o después de actualizar el clúster a una nueva versión secundaria de Kubernetes. Para actualizar el complemento para un clúster existente, debe activar una actualización mediante la API update-addon o haciendo clic en el enlace actualizar ahora de la consola de EKS para ver

los complementos. Si ha implementado un complemento autogestionado, siga los pasos que se mencionan en la sección Actualización del complemento CNI de VPC autogestionado.

Le recomendamos encarecidamente que actualice las versiones secundarias de una en una. Por ejemplo, si su versión secundaria actual es `1.9` y desea actualizarla a `1.11`, primero debe actualizarse a la versión más reciente de revisión de `1.10` y, luego, actualizarse la última versión de revisión de `1.11`.

Realice una inspección del Daemonset de aws-node antes de actualizar el CNI de Amazon VPC. Realice una copia de seguridad de la configuración existente. Si utiliza un complemento gestionado, confirme que no ha actualizado ninguna configuración que Amazon EKS pueda anular. Le recomendamos incluir un enlace posterior a la actualización en su flujo de trabajo de automatización o realizar un paso de aplicación manual después de una actualización del complemento.

```
kubectl apply view-last-applied daemonset aws-node -n kube-system  aws-k8s-cni-old.yaml
```

Si se trata de un complemento autogestionado, compara la copia de seguridad con `releases` otra GitHub para ver las versiones disponibles y familiarizarte con los cambios en la versión a la que quieres actualizar. Te recomendamos que utilices Helm para gestionar los complementos autogestionados y utilizar los archivos de valores para aplicar la configuración. Cualquier operación de actualización que implique la eliminación de Daemonset provocará un tiempo de inactividad de la aplicación y debe evitarse.

Comprenda el contexto de seguridad

Le recomendamos encarecidamente que comprenda los contextos de seguridad configurados para administrar la CNI de la VPC de manera eficiente. El CNI de Amazon VPC tiene dos componentes: CNI binario e ipamd (aws-node) Daemonset. El CNI se ejecuta como un binario en un nodo y tiene acceso al sistema de archivos raíz del nodo. También tiene acceso privilegiado, ya que gestiona iptables a nivel de nodo. El kubelet invoca el binario CNI cuando se añaden o quitan Pods.

El Daemonset de aws-node es un proceso de larga duración responsable de la administración de las direcciones IP a nivel de nodo. El aws-node se ejecuta en `hostNetwork` modo y permite el acceso al dispositivo de bucle invertido y a la actividad de red de otros módulos del mismo nodo. El contenedor de inicio aws-node se ejecuta en modo privilegiado y monta el socket CRI, lo que permite al Daemonset monitorear el uso de IP por parte de los pods que se ejecutan en el nodo. Amazon EKS está trabajando para eliminar el requisito de privilegios del contenedor de inicio aws-node. Además, el aws-node necesita actualizar las entradas NAT y cargar los módulos de iptables y, por lo tanto, funciona con los privilegios de NET_ADMIN.

Amazon EKS recomienda implementar las políticas de seguridad definidas en el manifiesto de aws-node para la administración de IP de los pods y la configuración de red. Considere la posibilidad de actualizar la CNI de VPC a la última versión. Además, considere la posibilidad de abrir una GitHub edición si tiene un requisito de seguridad específico.

Utilice un rol de IAM diferente para el CNI

La CNI de VPC de AWS requiere permisos de AWS Identity and Access Management (IAM). La política de CNI debe configurarse antes de poder utilizar la función de IAM. Puede usar `AmazonEKS_CNI_Policy`, que es una política administrada por AWS para IPv4 clústeres. La política gestionada por CNI de AmazoneKS solo tiene permisos para los clústeres. IPv4 Debe crear una política de IAM independiente para los IPv6 clústeres con los permisos que se indican aquí.

De forma predeterminada, la CNI de VPC hereda la función de IAM del nodo Amazon EKS (grupos de nodos gestionados y autogestionados).

Se recomienda encarecidamente configurar un rol de IAM independiente con las políticas pertinentes para el CNI de Amazon VPC. De lo contrario, los pods de Amazon VPC CNI obtienen el permiso asignado a la función de IAM del nodo y tienen acceso al perfil de instancia asignado al nodo.

El complemento CNI de VPC crea y configura una cuenta de servicio denominada aws-node. De forma predeterminada, la cuenta de servicio se vincula a la función de IAM del nodo Amazon EKS con la política CNI de Amazon EKS adjunta. Para utilizar la función de IAM independiente, le recomendamos que cree una nueva cuenta de servicio con la política CNI de Amazon EKS adjunta. Para usar una nueva cuenta de servicio, debe volver a implementar los pods de CNI. Considere la posibilidad de especificar un complemento gestionado `--service-account-role-arn` por CNI para VPC al crear nuevos clústeres. Asegúrese de eliminar la política CNI de Amazon EKS tanto IPv4 para el rol de nodo IPv6 de Amazon EKS como para el del mismo.

Se recomienda bloquear los metadatos de la instancia de acceso para minimizar el alcance de las brechas de seguridad.

Gestione los fallos Liveness/Readiness de la sonda

Recomendamos aumentar los valores de tiempo de espera de la sonda de actividad y preparación (predeterminados`timeoutSeconds: 10`) para los clústeres EKS 1.20 y versiones posteriores para evitar que los fallos de la sonda provoquen que el pod de la aplicación quede atascado en un estado de creación de contenedores. Este problema se ha observado en clústeres de procesamiento por lotes y con uso intensivo de datos. El uso elevado de la CPU provoca fallos en el estado de las

sondas de aws-node, lo que provoca que no se atiendan las solicitudes de CPU del pod. Además de modificar el tiempo de espera de la sonda, asegúrese de que las solicitudes de recursos de CPU (predeterminadas`CPU: 25m`) para aws-node estén configuradas correctamente. No sugerimos actualizar la configuración a menos que su nodo tenga problemas.

Le recomendamos encarecidamente que ejecute sudo `bash /opt/cni/bin/aws-cni-support.sh` en un nodo mientras contrata el soporte de Amazon EKS. El script le ayudará a evaluar los registros de kubelet y el uso de la memoria en el nodo. Considere instalar el agente SSM en los nodos de trabajo de Amazon EKS para ejecutar el script.

Configurar la política IPTables de reenvío en instancias AMI no optimizadas para EKS

Si utilizas una AMI personalizada, asegúrate de configurar la política de reenvío de iptables en ACCEPT en kubelet.service. Muchos sistemas configuran la política de reenvío de iptables en DROP. Puede crear una AMI personalizada mediante HashiCorp Packer y una especificación de compilación con recursos y scripts de configuración del repositorio de AMI de Amazon EKS en AWS GitHub. Puede actualizar kubelet.service y seguir las instrucciones especificadas aquí para crear una AMI personalizada.

Actualice rutinariamente la versión CNI

El CNI de VPC es compatible con versiones anteriores. La última versión funciona con todas las versiones de Kubernetes compatibles con Amazon EKS. Además, el VPC CNI se ofrece como un complemento de EKS (consulte la sección «Implementación del complemento gestionado por VPC CNI» más arriba). Si bien los complementos de EKS organizan las actualizaciones de los complementos, no los actualizan automáticamente como el CNI, porque se ejecutan en el plano de datos. Usted es responsable de actualizar el complemento CNI de la VPC tras las actualizaciones de los nodos de trabajo gestionadas y autogestionadas.

Optimización de la utilización de direcciones IP

Gracias a la modernización de las aplicaciones, la escala de los entornos en contenedores está creciendo a un ritmo acelerado. Esto significa que se están desplegando cada vez más nodos y módulos de trabajo.

El complemento CNI de Amazon VPC asigna a cada pod una dirección IP de los CIDR de la VPC. Este enfoque proporciona una visibilidad total de las direcciones de los pods con herramientas como los registros de flujo de VPC y otras soluciones de monitoreo. Según el tipo de carga de trabajo, esto puede provocar que los pods consuman una cantidad considerable de direcciones IP.

Al diseñar la arquitectura de red de AWS, es importante optimizar el consumo de IP de Amazon EKS en la VPC y a nivel de nodo. Esto le ayudará a mitigar los problemas de agotamiento de la IP y a aumentar la densidad de los pods por nodo.

En esta sección, analizaremos las técnicas que pueden ayudarlo a alcanzar estos objetivos.

Optimice el consumo de IP a nivel de nodo

La delegación de prefijos es una función de Amazon Virtual Private Cloud (Amazon VPC) que le permite IPv4 asignar IPv6 o prefijos a sus instancias de Amazon Elastic Compute Cloud (Amazon EC2). Aumenta las direcciones IP por interfaz de red (ENI), lo que aumenta la densidad de pods por nodo y mejora la eficiencia informática. Las redes personalizadas también admiten la delegación de prefijos.

Para obtener información detallada, consulte las secciones Delegación de prefijos con nodos de Linux y Delegación de prefijos con nodos de Windows.

Mitigación del riesgo de agotamiento de la IP

Para evitar que sus clústeres consuman todas las direcciones IP disponibles, le recomendamos encarecidamente que ajuste el tamaño de sus redes VPCs y de sus subredes teniendo en cuenta el crecimiento.

IPv6La adopción es una excelente manera de evitar estos problemas desde el principio. Sin embargo, para las organizaciones cuyas necesidades de escalabilidad superan la planificación inicial y no pueden adoptarlas IPv6, la respuesta recomendada al agotamiento de las direcciones IP es mejorar el diseño de la VPC. La técnica más utilizada entre los clientes de Amazon EKS consiste en añadir una dirección secundaria no enrutable CIDRs a la VPC y configurar la CNI de la VPC para que utilice este espacio IP adicional al asignar direcciones IP a los pods. Esto se conoce comúnmente como redes personalizadas.

Analizaremos qué variables del CNI de Amazon VPC puede utilizar para optimizar el pool caliente IPs asignado a sus nodos. Cerraremos esta sección con algunos otros patrones arquitectónicos que no son intrínsecos a Amazon EKS, pero que pueden ayudar a mitigar el agotamiento de la IP.

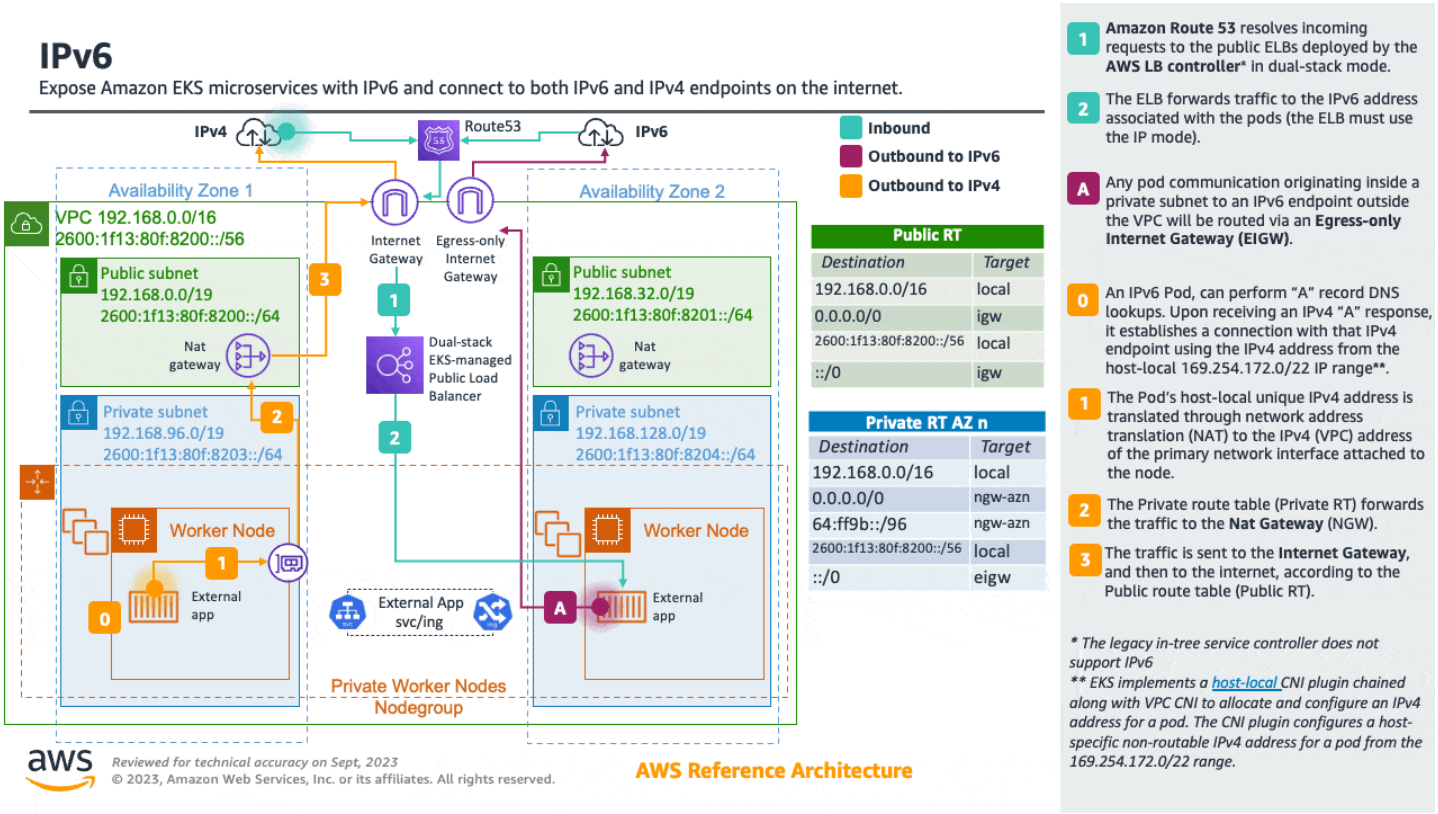
Uso IPv6 (recomendado)

La adopción IPv6 es la forma más sencilla de evitar las RFC1918 limitaciones; le recomendamos encarecidamente que considere la adopción IPv6 como primera opción a la hora de elegir una arquitectura de red. IPv6 proporciona un espacio total de direcciones IP considerablemente mayor, y

los administradores de clústeres pueden centrarse en migrar y escalar las aplicaciones sin tener que esforzarse en evitar IPv4 los límites.

Los clústeres de Amazon EKS admiten IPv4 tanto como IPv6. De forma predeterminada, los clústeres EKS utilizan el espacio de IPv4 direcciones. Si se especifica un espacio de direcciones IPv6 basado en el momento de la creación del clúster, se permitirá el uso de IPv6. En un clúster de IPv6 EKS, los módulos y los servicios reciben IPv6 direcciones y, al mismo tiempo, mantienen la capacidad de los IPv4 puntos finales heredados de conectarse a los servicios que se ejecutan en IPv6 los clústeres y viceversa. Toda la pod-to-pod comunicación dentro de un clúster siempre se produce de forma interrumpida. IPv6 Dentro de una VPC (/56), el tamaño del bloque IPv6 CIDR para las IPv6 subredes se fija en /64. Esto proporciona 2^64 direcciones (aproximadamente 18 trillones), lo que permite escalar sus despliegues en EKS. IPv6

Para obtener información detallada, consulte la sección Cómo ejecutar clústeres de IPv6 EKS y, para obtener experiencia práctica, consulte la sección Comprensión IPv6 de Amazon EKS del taller Póngase manos a la obra. IPv6



Optimice el consumo de IP en IPv4 los clústeres

Esta sección está dedicada a los clientes que ejecutan aplicaciones antiguas y no and/or están preparados para migrar a ellas IPv6. Si bien animamos a todas las organizaciones a migrar a ellas

lo antes IPv6 posible, reconocemos que es posible que algunas aún necesiten buscar enfoques alternativos para escalar sus cargas de trabajo en contenedores. IPv4 Por este motivo, también le mostraremos los patrones arquitectónicos para optimizar IPv4 (RFC1918) abordar el consumo de espacio con los clústeres de Amazon EKS.

Planifique su crecimiento

Como primera línea de defensa contra el agotamiento de la IP, recomendamos encarecidamente dimensionar las subredes IPv4 VPCs y las de sus subredes teniendo en cuenta el crecimiento, a fin de evitar que los clústeres consuman todas las direcciones IP disponibles. No podrás crear nuevos pods o nodos si las subredes no tienen suficientes direcciones IP disponibles.

Antes de crear la VPC y las subredes, se recomienda trabajar hacia atrás a partir de la escala de carga de trabajo requerida. Por ejemplo, cuando los clústeres se crean con eksctl (una sencilla herramienta CLI para crear y administrar clústeres en EKS), se crean 19 subredes de forma predeterminada. Una máscara de red de /19 es adecuada para la mayoría de los tipos de carga de trabajo, ya que permite asignar más de 8000 direcciones.

> **Important**
>
> Al ajustar el tamaño VPCs y las subredes, es posible que haya varios elementos (distintos de los pods y los nodos) que puedan consumir direcciones IP, por ejemplo, los balanceadores de carga, las bases de datos de RDS y otros servicios integrados en la vpc.

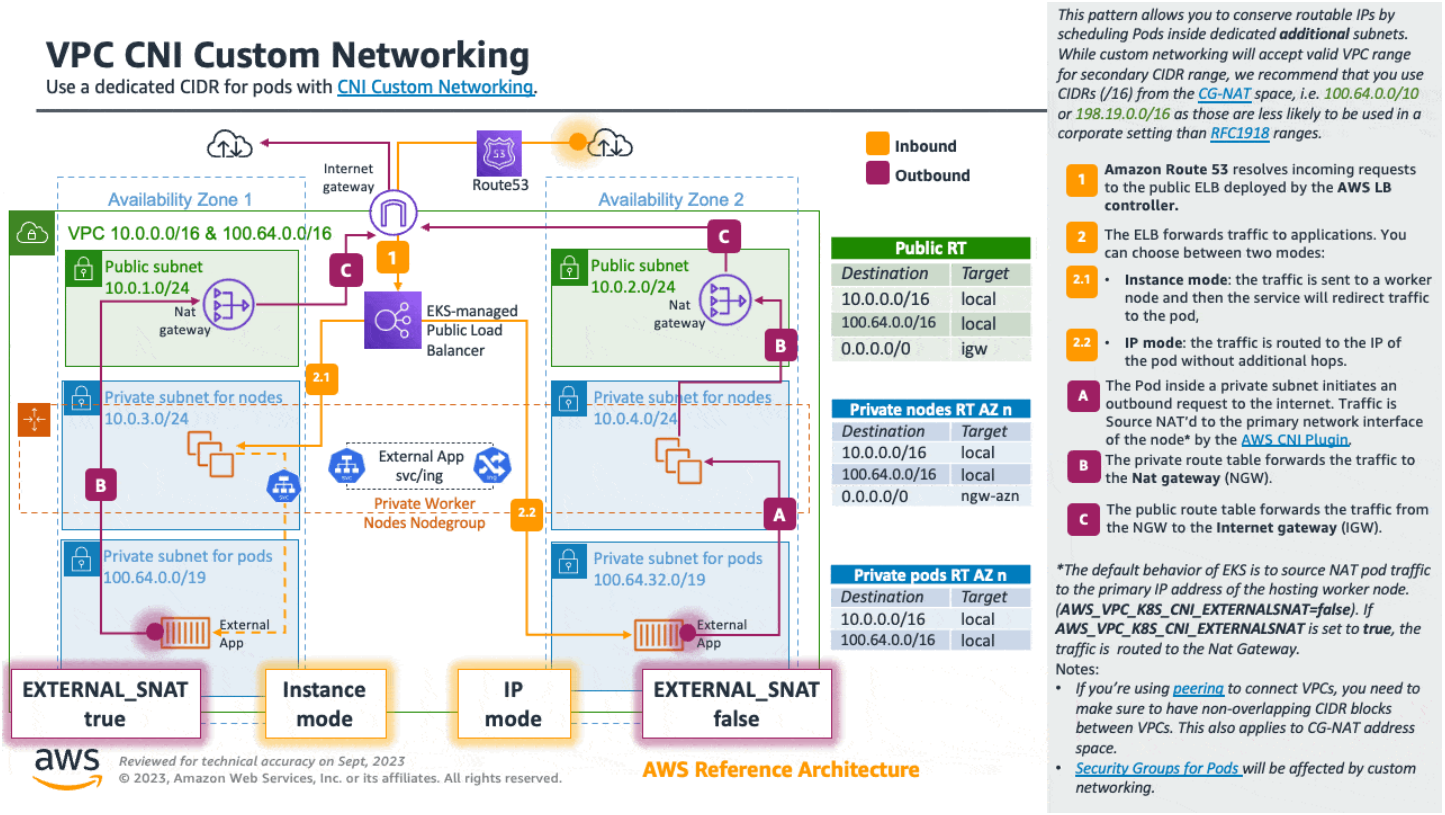
Además, Amazon EKS puede crear hasta 4 interfaces de red elásticas (X-ENI) necesarias para permitir la comunicación con el plano de control (más información aquí). Durante las actualizaciones de clústeres, Amazon EKS crea una nueva X ENIs y elimina las antiguas cuando la actualización se realiza correctamente. Por este motivo, recomendamos una máscara de red de al menos /28 (16 direcciones IP) para las subredes asociadas a un clúster de EKS.

Puede utilizar el ejemplo de la hoja de cálculo de la calculadora de subredes EKS para planificar su red. La hoja de cálculo calcula el uso de IP en función de las cargas de trabajo y la configuración ENI de la VPC. El uso de IP se compara con el de una IPv4 subred para determinar si la configuración y el tamaño de la subred son suficientes para la carga de trabajo. Tenga en cuenta que, si las subredes de su VPC se quedan sin direcciones IP disponibles, le sugerimos que cree una nueva subred con los bloques CIDR originales de la VPC. Observe que ahora Amazon EKS permite la modificación de subredes de clústeres y grupos de seguridad.

Redes personalizadas

Si estás a punto de agotar el espacio RFC1918 IP, puedes usar el patrón de redes personalizadas para conservar la capacidad de enrutamiento IPs programando los pods dentro de subredes adicionales dedicadas. Si bien las redes personalizadas aceptan un rango de VPC válido como rango de CIDR secundario, le recomendamos que utilice CIDRs (/16) desde el espacio CG-NAT, es decir`100.64.0.0/10`, `198.19.0.0/16` ya que es menos probable que se usen en un entorno corporativo que los rangos. RFC1918

Para obtener información detallada, consulte la sección dedicada a las redes personalizadas.



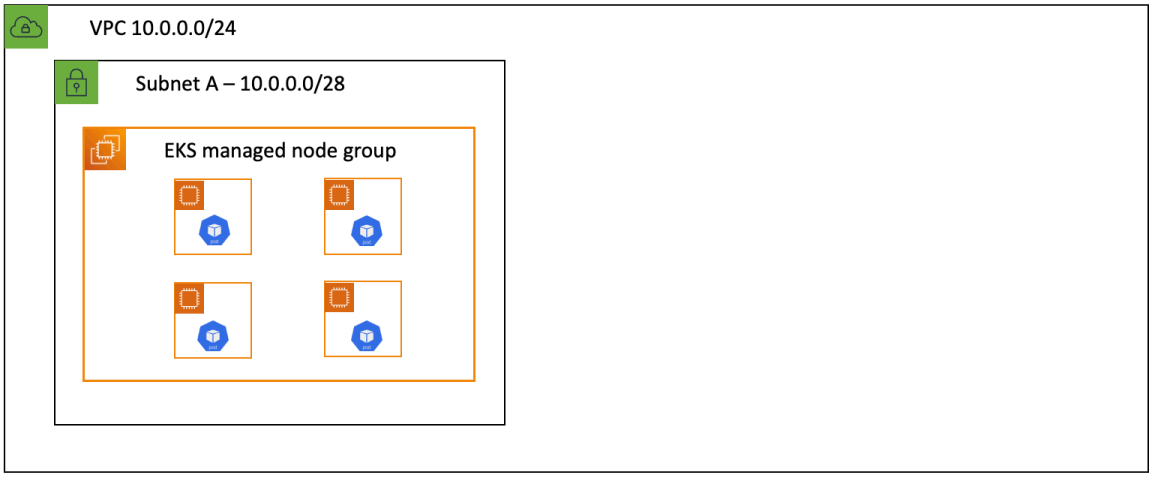
Descubrimiento de subredes mejorado

La detección de subredes mejorada ofrece una alternativa de configuración de red simplificada para el agotamiento de la IP, ya que etiqueta las nuevas subredes para que el CNI de Amazon VPC las pueda detectar. Con la detección mejorada de subredes, las cargas de trabajo actuales pueden seguir ejecutándose en las mismas subredes y Amazon Elastic Kubernetes Service (Amazon EKS) ahora puede programar pods adicionales en las nuevas «subredes utilizables».

Si las subredes actuales de su clúster se están quedando sin direcciones IP, simplemente puede añadir subredes adicionales a su clúster de Amazon EKS de la siguiente manera:

1. Asocie un nuevo bloque CIDR a su VPC.
2. Cree una nueva subred en el nuevo bloque CIDR y etiquétela con «kubernetes». io/role/cni» = «1".
3. Active la configuración ENABLE_SUBNET_DISCOVERY del complemento CNI de Amazon VPC en «true» (predeterminado desde la versión 1.18.0).

Una vez que la detección mejorada de subredes esté habilitada en los clústeres de VPC y Amazon EKS, se conectarán nuevas interfaces de red elásticas ENIs () a los nodos de Amazon EKS, tal como se describe en el siguiente diagrama:



Para obtener más información, consulte Amazon VPC CNI presenta la detección mejorada de subredes en el blog sobre contenedores de AWS.

Optimice la piscina caliente IPs

Con la configuración predeterminada, el CNI de VPC mantiene un ENI completo (y el asociado IPs) en la piscina caliente. Esto puede consumir una gran cantidad de IPs, especialmente en los tipos de instancias más grandes.

Si la subred de su clúster tiene un número limitado de direcciones IP disponibles, analice estas variables del entorno de configuración de la CNI de la VPC:

- `WARM_IP_TARGET`
- `MINIMUM_IP_TARGET`
- `WARM_ENI_TARGET`

Puedes configurar el valor de `MINIMUM_IP_TARGET` para que coincida estrechamente con la cantidad de pods que esperas ejecutar en tus nodos. Si lo haces, te asegurarás de que, a medida que se creen los pods, el CNI pueda asignar direcciones IP desde el pool caliente sin necesidad de llamar a la EC2 API.

Ten en cuenta que si estableces un valor `WARM_IP_TARGET` demasiado bajo, se generarán más llamadas a la EC2 API y se podrían limitar las solicitudes. En el caso de clústeres grandes, úsalo junto con `MINIMUM_IP_TARGET` para evitar la limitación de las solicitudes.

Para configurar estas opciones, puedes descargar el `aws-k8s-cni.yaml` manifiesto y configurar las variables de entorno. En el momento de escribir este artículo, la última versión se encuentra aquí. Compruebe que la versión del valor de configuración coincida con la versión CNI de la VPC instalada.

> **Warning**
>
> Estos ajustes se restablecerán a sus valores predeterminados cuando actualice la CNI. Realice una copia de seguridad del CNI antes de actualizarlo. Revise los ajustes de configuración para determinar si necesita volver a aplicarlos una vez que la actualización se haya realizado correctamente.

Puede ajustar los parámetros de la CNI sobre la marcha sin tiempo de inactividad para sus aplicaciones existentes, pero debe elegir valores que se adapten a sus necesidades de escalabilidad. Por ejemplo, si trabajas con cargas de trabajo por lotes, te recomendamos que actualices los valores predeterminados `WARM_ENI_TARGET` para que se ajusten a las necesidades de la báscula de Pod. Si `WARM_ENI_TARGET` se establece un valor alto, siempre se mantiene el conjunto de IP caliente necesario para ejecutar grandes cargas de trabajo por lotes y, por lo tanto, se evitan retrasos en el procesamiento de datos.

> **Warning**
>
> Mejorar el diseño de la VPC es la respuesta recomendada ante el agotamiento de las direcciones IP. Considere soluciones como IPv6 la secundaria CIDRs. Ajustar estos valores

para minimizar el número de temperaturas cálidas IPs debería ser una solución temporal una vez que se hayan excluido otras opciones. La configuración incorrecta de estos valores puede interferir con el funcionamiento del clúster. Antes de realizar cualquier cambio en un sistema de producción, asegúrese de revisar las consideraciones de esta página.

Supervise el inventario de direcciones IP

Además de las soluciones descritas anteriormente, también es importante tener visibilidad sobre la utilización de IP. Puede supervisar el inventario de direcciones IP de las subredes mediante CNI Metrics Helper. Algunas de las métricas disponibles son:

- la cantidad máxima que puede admitir ENIs el clúster
- número de ENIs ya asignados
- número de direcciones IP actualmente asignadas a los pods
- número total y máximo de direcciones IP disponibles

También puede configurar CloudWatch alarmas para recibir notificaciones si una subred se está quedando sin direcciones IP.

> **Warning**
>
> Asegúrese de que la `DISABLE_METRICS` variable del CNI de VPC esté establecida en false.

Consideraciones adicionales

Existen otros patrones arquitectónicos que no son intrínsecos a Amazon EKS y que pueden ayudar a reducir la IP. Por ejemplo, puede optimizar la comunicación VPCs o compartir una VPC entre varias cuentas para limitar la asignación de IPv4 direcciones.

Obtenga más información sobre estos patrones aquí:

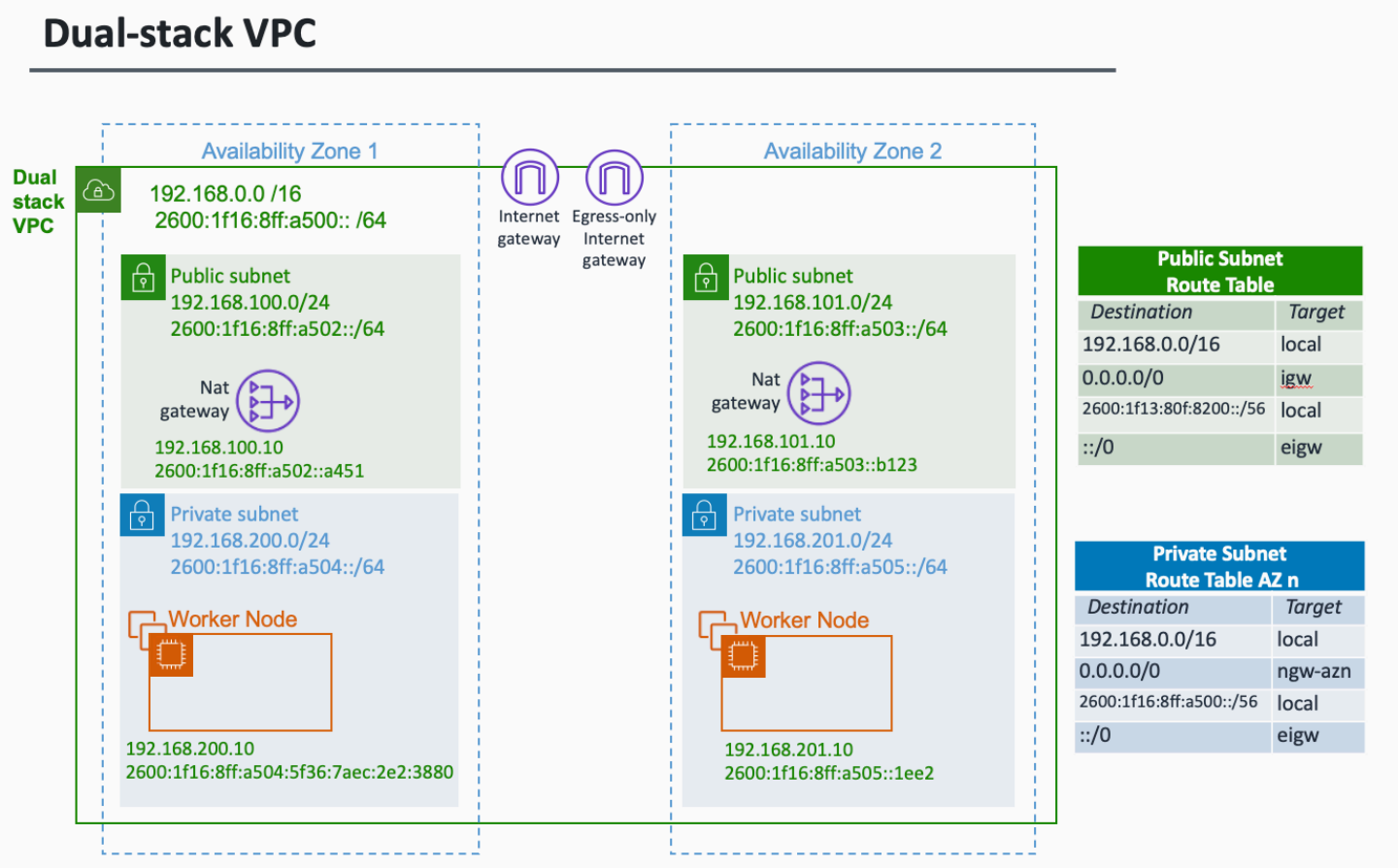
- Diseño de redes Amazon VPC de hiperescala,
- Cree una conectividad segura entre varias cuentas y varias VPC con Amazon VPC Lattice.

IPv6 Ejecución de clústeres EKS

El IPv6 modo EKS resuelve el problema de IPv4 agotamiento que suele presentarse en los clústeres de EKS a gran escala. El apoyo de EKS IPv6 se centra en resolver el problema de IPv4 agotamiento, que se debe al tamaño limitado del espacio de IPv4 direcciones. Esta es una preocupación importante planteada por varios de nuestros clientes y es distinta de la función de doble pila de Kubernetes IPv4. IPv6 EKS/ también IPv6 proporcionará la flexibilidad necesaria para interconectar los límites de la red y, por lo tanto, minimizar las posibilidades de que se produzca una superposición de CIDR y, IPv6 CIDRs por lo tanto, resolverá un problema doble (dentro del clúster, entre clústeres). Cuando se despliegan clústeres EKS en IPv6 modo (--ip-family ipv6), la acción no es reversible. En pocas palabras, la IPv6 compatibilidad con EKS está habilitada durante toda la vida útil del clúster.

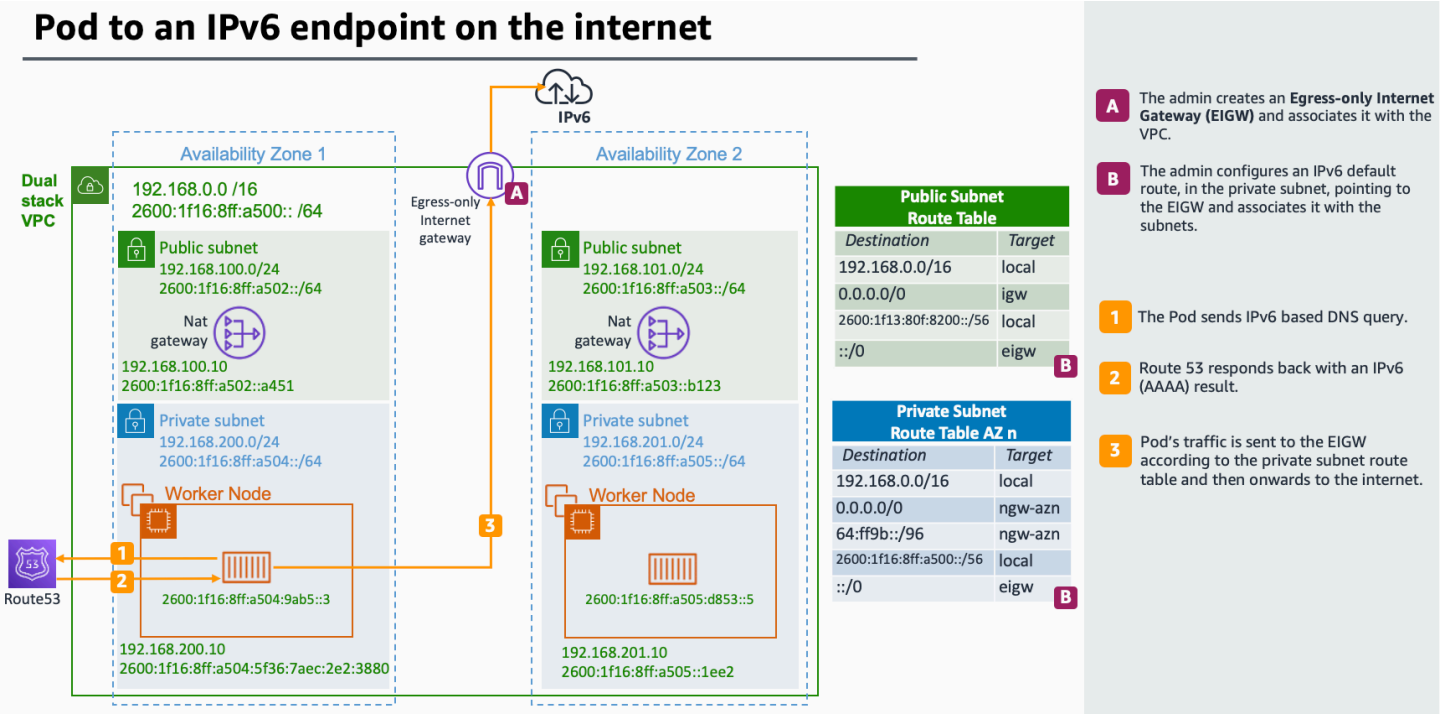
En un clúster de IPv6 EKS, los pods y los servicios recibirán IPv6 direcciones y, al mismo tiempo, mantendrán la compatibilidad con los IPv4 puntos finales antiguos. Esto incluye la posibilidad de que los IPv4 puntos finales externos accedan a los servicios del clúster y los pods accedan a los puntos finales externos. IPv4

El IPv6 soporte de Amazon EKS aprovecha las capacidades de IPv6 VPC nativas. A cada VPC se le asigna un prefijo de IPv4 dirección (el tamaño del bloque CIDR puede oscilar entre /16 y /28) y un prefijo de dirección /56 único (fijo) desde la GUA (IPv6 dirección de unidifusión global) de Amazon; puede asignar un prefijo de dirección /64 a cada subred de la VPC. IPv4 las funciones, como las tablas de enrutamiento, las listas de control de acceso a la red, el emparejamiento y la resolución de DNS, funcionan de la misma manera en una VPC IPv6 habilitada. En ese caso, la VPC se denomina VPC de doble pila. En el siguiente diagrama se muestra el patrón básico de la IPV4 IPv6 VPC que admite clústeres basados en EKS/: IPv6



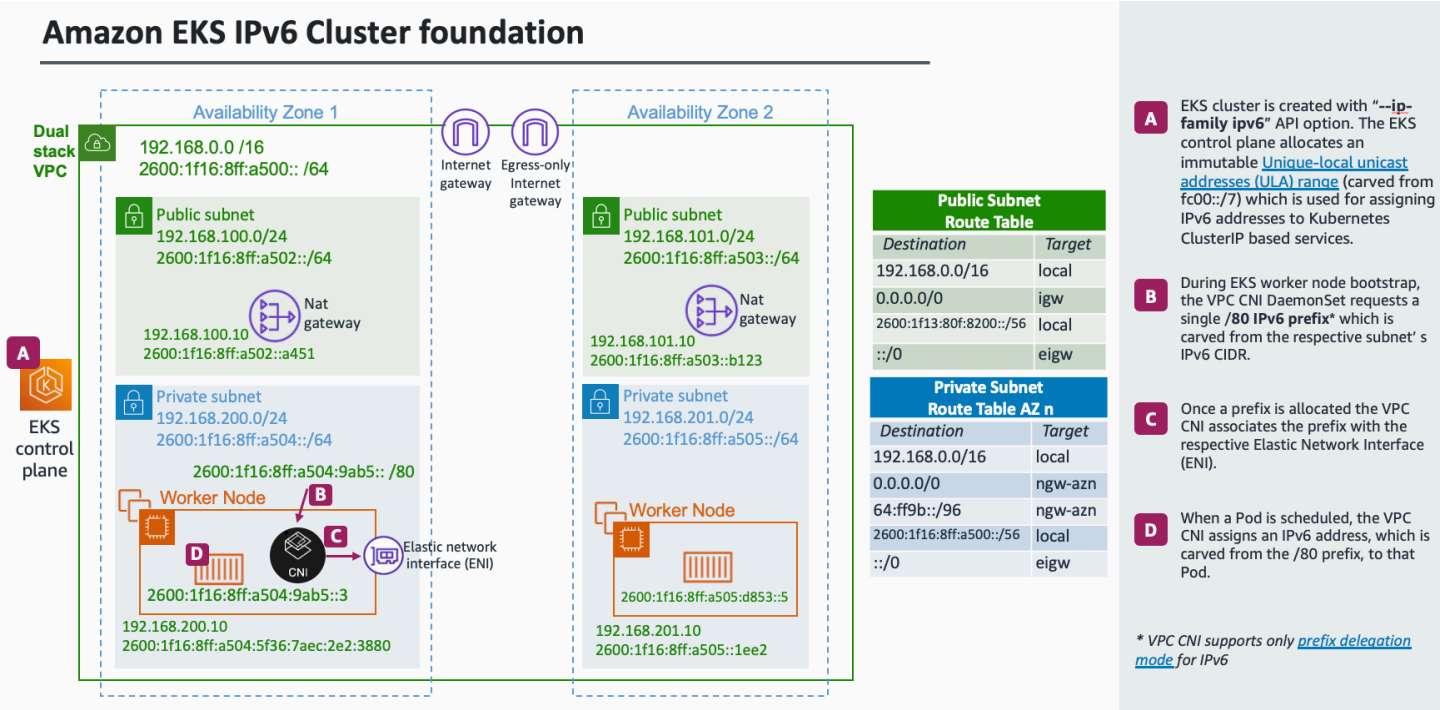
En el IPv6 mundo, todas las direcciones se pueden enrutar por Internet. De forma predeterminada, la VPC asigna el IPv6 CIDR del rango GUA público. Sin embargo, desde agosto de 2024, también puede usar IPv6 direcciones privadas VPCs y subredes con Amazon VPC IP Address Manager (IPAM). Consulte esta entrada del blog sobre redes de AWS y la documentación de VPC para obtener más información.

El siguiente diagrama muestra el flujo de salida de IPv6 Internet de un pod dentro de un clúster EKS/: IPv6



Las prácticas recomendadas para implementar IPv6 subredes se encuentran en la guía del usuario de VPC.

En un clúster de IPv6 EKS, los nodos y los pods reciben direcciones públicas IPv6 . EKS asigna IPv6 direcciones a los servicios en función de las direcciones de IPv6 unidifusión locales únicas (ULA). El CIDR del servicio ULA para un IPv6 clúster se asigna automáticamente durante la etapa de creación del clúster y no se puede especificar, a diferencia de lo que ocurre. IPv4 El siguiente diagrama muestra un patrón básico de un plan de datos del plano de control de clústeres IPv6 basado en EKS/:



Descripción general de

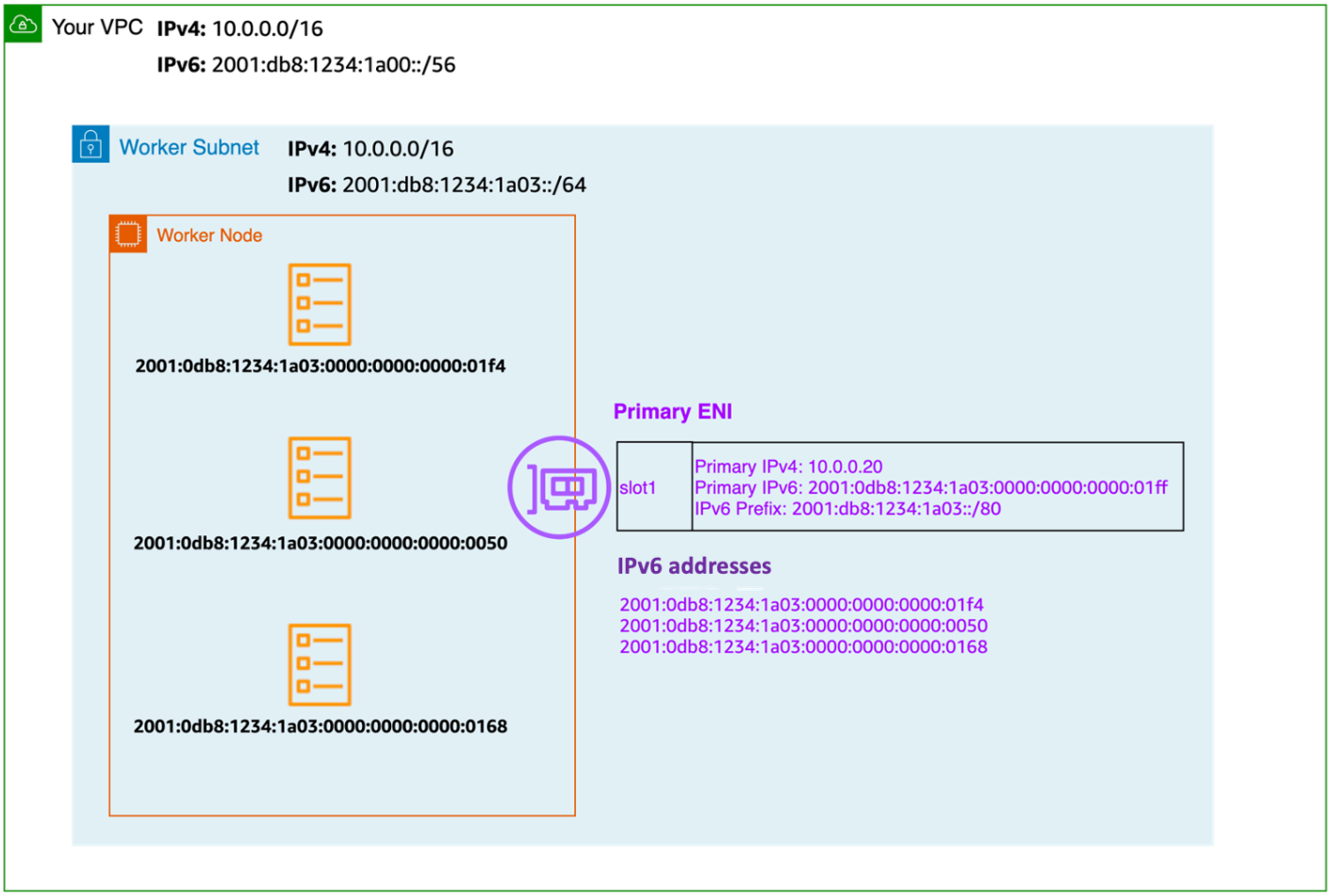
EKS/ solo IPv6 se admite en el modo de prefijo (modo de asignación de IP ENI del complemento VPC-CNI). Más información sobre el modo de prefijo.

La asignación de prefijos solo funciona en EC2 instancias basadas en Nitro, por lo que EKS/ solo IPv6 se admite cuando el plano de datos del clúster utiliza instancias basadas en Nitro. EC2

En pocas palabras, un IPv6 prefijo /80 (por nodo trabajador) generará aproximadamente 10^14 IPv6 direcciones. El factor limitante ya no será sino la densidad de los pods (en términos de recursos). IPs

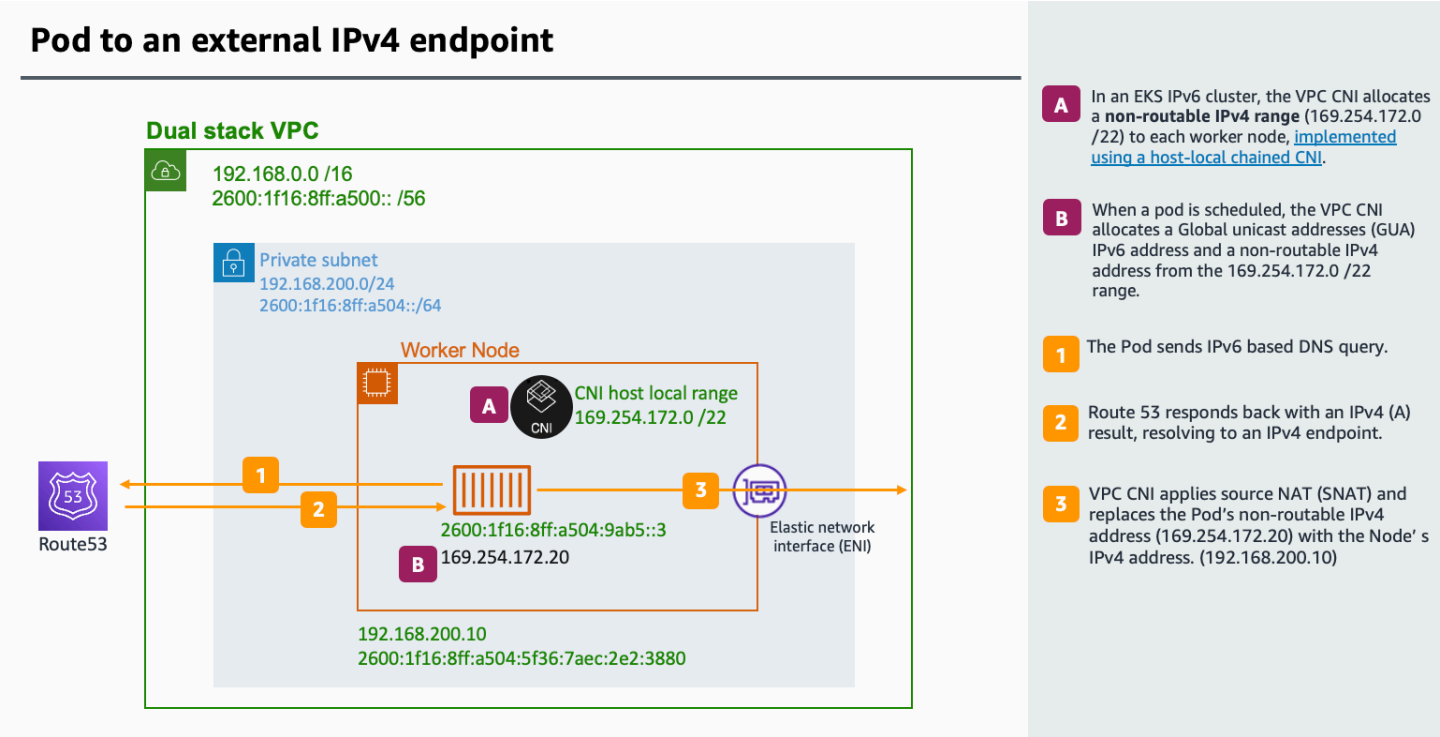
IPv6 la asignación de prefijos solo se produce en el momento del arranque del nodo de trabajo de EKS. Se sabe que este comportamiento mitiga los escenarios en los que los IPv4 clústeres EKS/ con una alta tasa de abandono de pods suelen retrasarse en la programación de los pods debido a la limitación de las llamadas a la API generadas por el complemento CNI de la VPC (ipamd) con el objetivo de asignar direcciones privadas de manera oportuna. IPv4 También se sabe que los botones avanzados del complemento VPC-CNI ajustan WARM_IP/ENI y MINIMUM_IP innecesariamente.

El siguiente diagrama amplía una interfaz de red elástica (ENI) de nodo de trabajo: IPv6



A cada nodo de trabajo de EKS se le asignan IPv6 direcciones IPv4 y direcciones, junto con las entradas de DNS correspondientes. Para un nodo de trabajo determinado, solo se consume una IPv4 dirección de la subred de doble pila. El soporte de EKS IPv6 le permite comunicarse con los IPv4 puntos de enlace (AWS, locales, Internet) a través de un modelo de salida altamente obstinado. IPv4 EKS implementa un complemento CNI local del host, secundario al complemento CNI de VPC, que asigna y configura una dirección para un pod. IPv4 El complemento CNI configura una dirección no enrutable específica del host para un pod del rango 169.254.172.0/22. IPv4 La IPv4 dirección asignada al pod es exclusiva del nodo de trabajo y no se anuncia fuera del nodo de trabajo. 169.254.172.0/22 proporciona hasta 1024 direcciones únicas que pueden admitir tipos de instancias grandes. IPv4

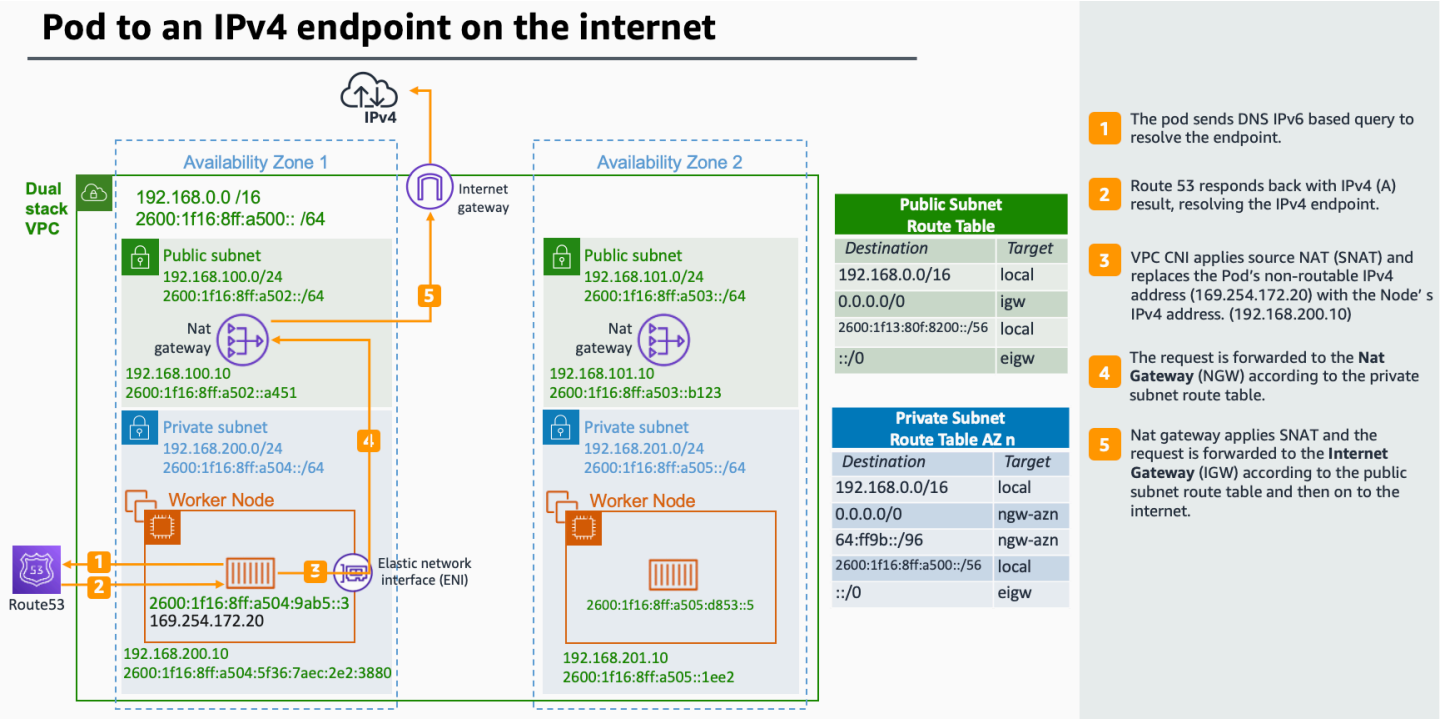
El siguiente diagrama muestra el flujo de un IPv6 pod que se conecta a un punto final situado fuera del límite del clúster (sin conexión a Internet): IPv4



En el diagrama anterior, los Pods realizarán una búsqueda de DNS para el punto final y, al recibir una respuesta IPv4 «A», la IPv4 dirección única del nodo del Pod se traduce mediante la traducción de direcciones de red de origen (SNAT) a la dirección privada (IPv4 VPC) de la interfaz de red principal conectada al nodo de trabajo. EC2

> **Note**
>
> El patrón anterior requiere que se deshabilite en las subredes en las DNS64 que se ejecutan EKS/ Pods. IPv6 Cuando DNS64 está activado, el solucionador de DNS devuelve una IPv6 dirección sintetizada para los puntos finales IPv4 exclusivos junto con una dirección. IPv4 Como resultado, el tráfico se enruta a través de la NAT64 funcionalidad de la puerta de enlace NAT (si está incluida en la arquitectura) en lugar de permanecer dentro de la VPC, como se muestra en el patrón anterior. Esto puede provocar un uso inesperado de la puerta de enlace NAT y los costes asociados.

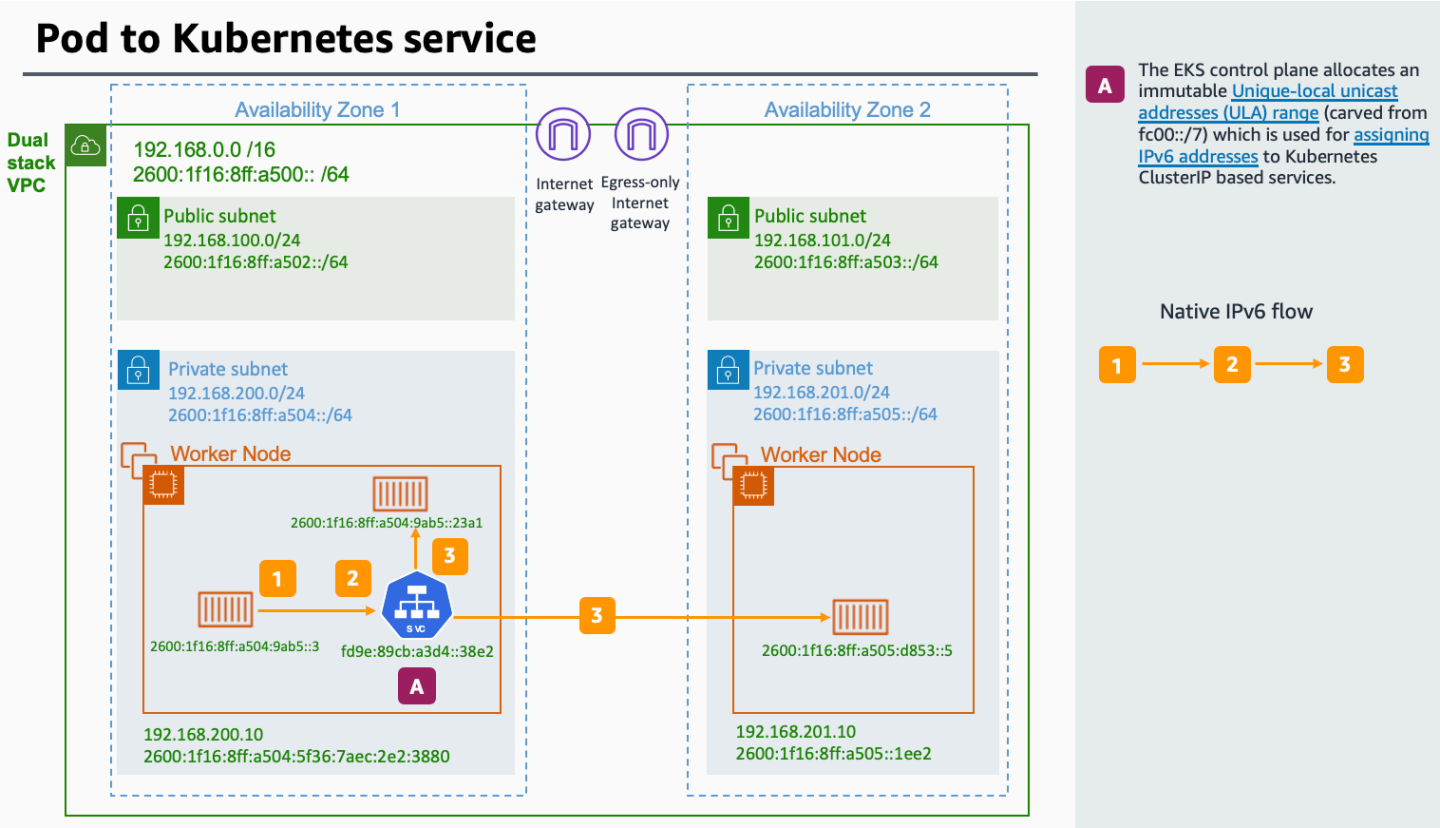
Los IPv6 EKS/Pods también deberán conectarse a los IPv4 puntos finales a través de Internet mediante IPv4 direcciones públicas para lograr que exista un flujo similar. El siguiente diagrama muestra el flujo de un IPv6 pod que se conecta a un IPv4 punto final situado fuera del límite del clúster (se puede enrutar por Internet):



En el diagrama anterior, los Pods realizarán una búsqueda de DNS para el punto final y, al recibir una respuesta IPv4 «A», la IPv4 dirección única del nodo del Pod se traduce mediante la traducción de direcciones de red de origen (SNAT) a la dirección privada (IPv4 VPC) de la interfaz de red principal conectada al nodo de trabajo. EC2 La IPv4 dirección del pod (fuente IPv4: IP EC2 principal) se enruta luego a la puerta de enlace IPv4 NAT, donde la IP EC2 principal se traduce (SNAT) en una dirección IP pública enrutable por Internet válida (IP IPv4 pública asignada a la puerta de enlace NAT).

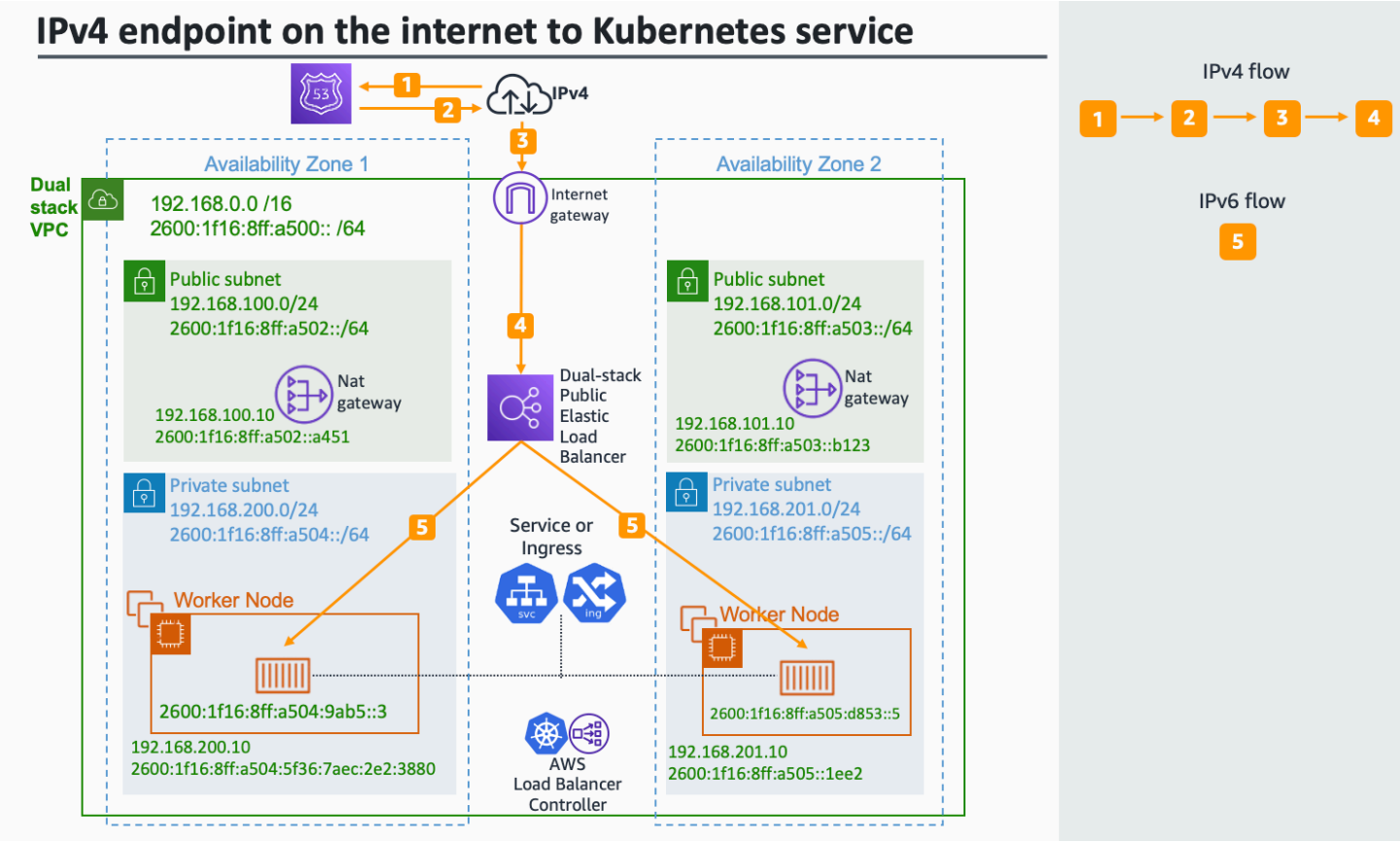
Cualquier Pod-to-Pod comunicación entre los nodos siempre utiliza una dirección. IPv6 La VPC CNI configura iptables para que gestione y bloquee cualquier conexión. IPv6 IPv4

Los servicios de Kubernetes solo recibirán direcciones IPv6 (ClusteriP) de direcciones unidifusión locales únicas (ULA). IPv6 El CIDR del servicio ULA para un IPv6 clúster se asigna automáticamente durante la etapa de creación del clúster de EKS y no se puede modificar. El siguiente diagrama muestra el flujo del servicio Pod a Kubernetes:



Los servicios se exponen a Internet mediante un balanceador de cargas de AWS. El balanceador de cargas recibe IPv6 direcciones IPv4 y direcciones públicas, lo que se conoce como balanceador de cargas de doble pila. En el caso de IPv4 los clientes que acceden a los servicios de Kubernetes en IPv6 clústeres, el balanceador de cargas se encarga de la traducción. IPv4 IPv6

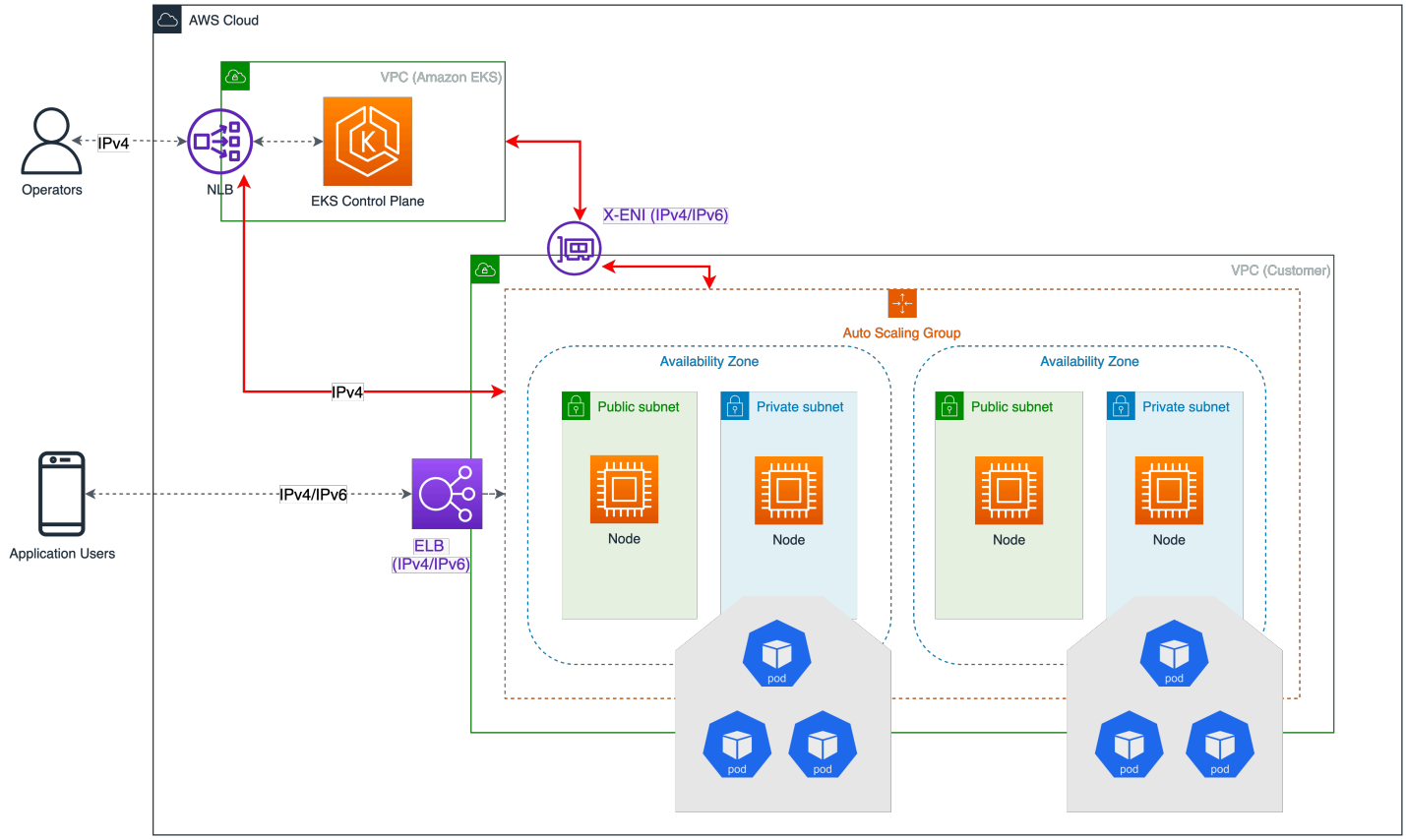
Amazon EKS recomienda ejecutar nodos de trabajo y pods en subredes privadas. Puede crear balanceadores de carga públicos en las subredes públicas que equilibren la carga del tráfico hacia los pods que se ejecutan en nodos que se encuentran en subredes privadas. El siguiente diagrama muestra a un IPv4 usuario de Internet que accede a un servicio basado en IPv6 EKS/Ingress:



> **Note**
>
> El patrón anterior requiere implementar la versión más reciente del controlador de balanceador de carga de AWS

Comunicación con el plano de datos del plano de control EKS

EKS suministrará Cross-Account ENIs (X-ENIs) en modo de doble pila (IPv4/IPv6). Los componentes del nodo de Kubernetes, como kubelet y kube-proxy, están configurados para admitir la doble pila. Kubelet y kube-proxy se ejecutan en modo HostNetwork y se enlazan a ambos y a las direcciones conectadas a la interfaz de red principal de un nodo. IPv4 IPv6 El servidor api de Kubernetes se comunica con los pods y los componentes del nodo a través de la interfaz X-. ENIs IPv6 Los pods se comunican con los servidores API a través de la XENIs, y la comunicación entre el pod y el servidor API siempre utiliza el modo. IPv6



Recomendaciones

Programación basada en recursos informáticos

Un solo IPv6 prefijo es suficiente para ejecutar varios pods en un solo nodo. Esto también elimina eficazmente las limitaciones de ENI e IP en cuanto al número máximo de pods en un nodo. Si bien IPv6 elimina la dependencia directa de los max-pods, al usar adjuntos de prefijo con tipos de instancias más pequeños, como la m5.large, es probable que se agoten los recursos de CPU y memoria de la instancia mucho antes de agotar sus direcciones IP. Debe establecer manualmente el valor máximo de pod recomendado por EKS si utiliza grupos de nodos autogestionados o un grupo de nodos gestionado con un ID de AMI personalizado.

Puedes usar la siguiente fórmula para determinar la cantidad máxima de pods que puedes implementar en un nodo de un clúster de IPv6 EKS.

```
((Number of network interfaces for instance type (number of prefixes per network
 interface-1)* 16) + 2
```

```
((3 ENIs)_((10 secondary IPs per ENI-1)_ 16)) + 2 = 460 (real)
```

Los grupos de nodos gestionados calculan automáticamente la cantidad máxima de pods para ti. Evita cambiar el valor recomendado por EKS para la cantidad máxima de pods para evitar errores en la programación de los pods debido a limitaciones de recursos.

Evalúe el propósito de las redes personalizadas existentes

Si las redes personalizadas están habilitadas actualmente, Amazon EKS recomienda volver a evaluar su necesidad con IPv6. Si optó por utilizar una red personalizada para solucionar el problema del IPv4 agotamiento, ya no será necesaria. IPv6 Si utiliza una red personalizada para cumplir con un requisito de seguridad, como una red independiente para nodos y pods, le recomendamos que envíe una solicitud de hoja de ruta de EKS.

Cápsulas Fargate en EKS/Cluster IPv6

EKS es compatible con IPv6 los pods que se ejecutan en Fargate. Los pods que se ejecutan en Fargate consumirán direcciones privadas enrutables de IPv6 VPC extraídas de los rangos CIDR IPv4 de la VPC (). IPv4 IPv6 En pocas palabras, la densidad de su clúster de EKS/Fargate Pods se limitará a las direcciones y direcciones disponibles. IPv4 IPv6 Se recomienda dimensionar su pila doble subnets/VPC CIDRs para un futuro crecimiento. No podrás programar nuevos Fargate Pods si la subred subyacente no contiene una dirección disponible, independientemente de IPv6 las IPv4 direcciones disponibles.

Implemente el controlador Load Balancer Controller (LBC) de AWS

El controlador de servicios de Kubernetes integrado en el árbol original no es compatible. IPv6 Recomendamos utilizar la versión más reciente del complemento AWS Load Balancer Controller. El LBC solo implementará un NLB de doble pila o un ALB de doble pila si consume la definición de kubernetes correspondiente anotada con: y service/ingress `"alb.ingress.kubernetes.io/ip-address-type: dualstack"` `"alb.ingress.kubernetes.io/target-type: ip"`

Redes personalizadas

De forma predeterminada, el CNI de Amazon VPC asignará a los pods una dirección IP seleccionada de la subred principal. La subred principal es la subred CIDR a la que está conectado el ENI principal, normalmente la subred del nodo/host.

Si el CIDR de la subred es demasiado pequeño, es posible que el CNI no pueda adquirir suficientes direcciones IP secundarias para asignarlas a tus pods. Este es un desafío común para los clústeres de EKS. IPv4

Las redes personalizadas son una solución a este problema.

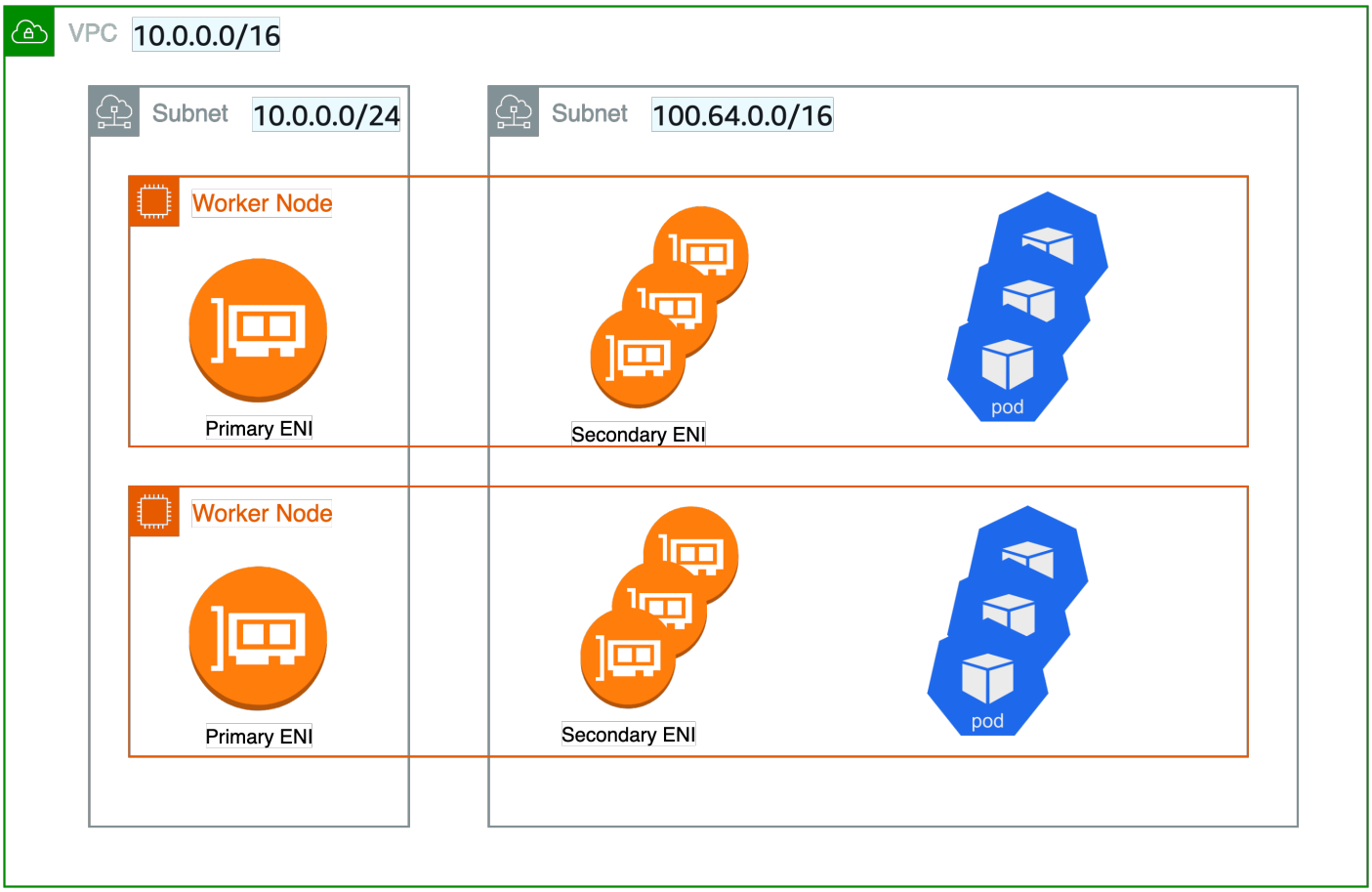
Las redes personalizadas abordan el problema del agotamiento de la IP mediante la asignación del nodo y el pod IPs desde los espacios de direcciones de VPC secundarios (CIDR). El soporte de redes personalizadas admite ENIConfig recursos personalizados. ENIConfig Incluye un rango CIDR de subred alternativo (extraído de un CIDR de VPC secundario), junto con los grupos de seguridad a los que pertenecerán los pods. Cuando se habilitan las redes personalizadas, la CNI de la VPC crea una secundaria en la subred definida ENIs en. ENIConfig El CNI asigna a los pods direcciones IP de un rango de CIDR definido en un CRD. ENIConfig

Como las redes personalizadas no utilizan el ENI principal, la cantidad máxima de pods que puede ejecutar en un nodo es inferior. Los pods de la red host siguen utilizando la dirección IP asignada al ENI principal. Además, el ENI principal se utiliza para gestionar la traducción de la red de origen y enrutar el tráfico de los Pods fuera del nodo.

Configuración de ejemplo

Si bien las redes personalizadas aceptan un rango de VPC válido como rango de CIDR secundario, le recomendamos que utilice CIDRs (/16) del espacio CG-NAT, es decir, 100.64.0.0/10 o 198.19.0.0/16, ya que es menos probable que se usen en un entorno corporativo que en otros rangos. RFC1918 Para obtener información adicional sobre las asociaciones de bloques de CIDR permitidas y restringidas que puede usar con su VPC, IPv4 consulte Restricciones de asociación de bloques de CIDR en la sección Tamaño de subredes y VPC de la documentación de la VPC.

Como se muestra en el siguiente diagrama, la interfaz de red elástica (ENI) principal del nodo de trabajo sigue utilizando ENIs el rango CIDR de VPC principal (en este caso 10.0.0.0/16), pero la secundaria usa el rango CIDR de VPC secundario (en este caso 100.64.0.0/16). Ahora, para que los pods usen el rango CIDR de 100.64.0.0/16, debes configurar el complemento CNI para que utilice redes personalizadas. Puedes seguir los pasos que se detallan aquí.



Si desea que el CNI utilice una red personalizada, defina la variable de `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG` entorno en. `true`

```
kubectl set env daemonset aws-node -n kube-system
 AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

Cuando`AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true`, el CNI asignará la dirección IP del pod desde una subred definida en. `ENIConfig` El recurso `ENIConfig` personalizado se usa para definir la subred en la que se programarán los pods.

```
apiVersion : crd.k8s.amazonaws.com/v1alpha1
kind : ENIConfig
metadata:
  name: us-west-2a
spec:
  securityGroups:
    - sg-0dff111a1d11c1c11
```

```
  subnet: subnet-011b111c1f11fdf11
```

Al crear los recursos `ENIconfig` personalizados, tendrás que crear nuevos nodos de trabajo y agotar los nodos existentes. Los nodos de trabajo y los pods existentes no se verán afectados.

Recomendaciones

Utilice redes personalizadas cuando

Le recomendamos que considere la posibilidad de utilizar una red personalizada si está IPv4 agotada y IPv6 aún no puede utilizarla. El soporte de Amazon EKS para RFC6598el espacio le permite escalar los pods más allá de RFC1918abordar los desafíos de agotamiento. Considere la posibilidad de utilizar la delegación de prefijos con redes personalizadas para aumentar la densidad de pods en un nodo.

Si tienes requisitos de seguridad para ejecutar los Pods en una red diferente con requisitos de grupo de seguridad diferentes, podrías considerar la posibilidad de utilizar redes personalizadas. Cuando están habilitadas las redes personalizadas, los pods utilizan subredes o grupos de seguridad diferentes, tal como se definen ENIConfig en la interfaz de red principal del nodo.

De hecho, las redes personalizadas son una opción ideal para implementar varios clústeres y aplicaciones de EKS para conectar los servicios del centro de datos local. Puede aumentar la cantidad de direcciones privadas (RFC1918) a las que EKS puede acceder en su VPC para servicios como Amazon Elastic Load Balancing y NAT-GW, y, al mismo tiempo, utilizar un espacio CG-NAT no enrutable para sus pods en varios clústeres. Las redes personalizadas con la puerta de enlace de tránsito y una VPC de servicios compartidos (incluidas las puertas de enlace NAT en varias zonas de disponibilidad para una alta disponibilidad) le permiten ofrecer flujos de tráfico escalables y predecibles. Esta entrada de blog describe un patrón arquitectónico que es una de las formas más recomendadas de conectar los EKS Pods a una red de centro de datos mediante redes personalizadas.

Evite las redes personalizadas cuando

¿Listo para implementar IPv6

Las redes personalizadas pueden mitigar los problemas de agotamiento de la IP, pero requieren una sobrecarga operativa adicional. Si actualmente está implementando una VPC de doble pila (IPv4/IPv6) o si su plan IPv6 incluye soporte, le recomendamos IPv6 implementar clústeres en su

lugar. Puede configurar clústeres de IPv6 EKS y migrar sus aplicaciones. En un clúster de IPv6 EKS, tanto Kubernetes como los Pods obtienen una IPv6 dirección y pueden comunicarse de entrada y salida con ambos IPv4 puntos de conexión. IPv6 Consulte las prácticas recomendadas para ejecutar clústeres de EKS. IPv6

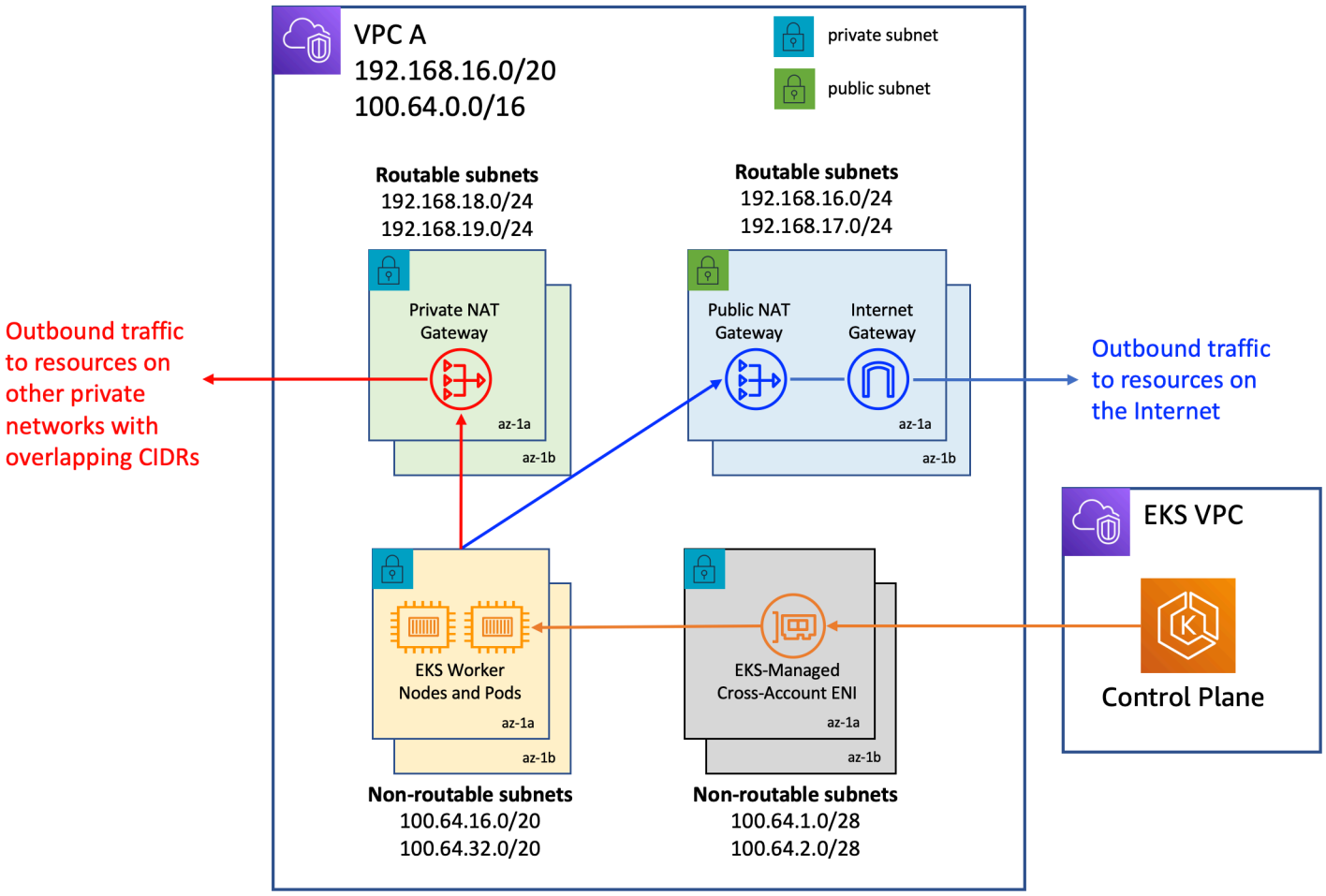
Espacio CG-NAT agotado

Además, si actualmente utiliza CIDRs el espacio CG-NAT o no puede vincular un CIDR secundario a la VPC de su clúster, es posible que deba explorar otras opciones, como usar una CNI alternativa. Le recomendamos encarecidamente que busque soporte comercial o que cuente con los conocimientos técnicos necesarios para depurar y enviar parches al proyecto de complemento CNI de código abierto. Consulte la guía de usuario de complementos CNI alternativos para obtener más información.

Utilice una puerta de enlace NAT privada

Amazon VPC ahora ofrece capacidades de puerta de enlace NAT privada. La puerta de enlace NAT privada de Amazon permite que las instancias de subredes privadas se conecten a otras redes VPCs y a redes locales superpuestas. CIDRs Considere la posibilidad de utilizar el método descrito en esta entrada de blog para emplear una puerta de enlace NAT privada para superar los problemas de comunicación de las cargas de trabajo de EKS causados por la superposición CIDRs, una queja importante que han expresado nuestros clientes. Las redes personalizadas no pueden resolver por sí solas las dificultades que se superponen entre sí en el CIDR y, además, agravan los desafíos de configuración.

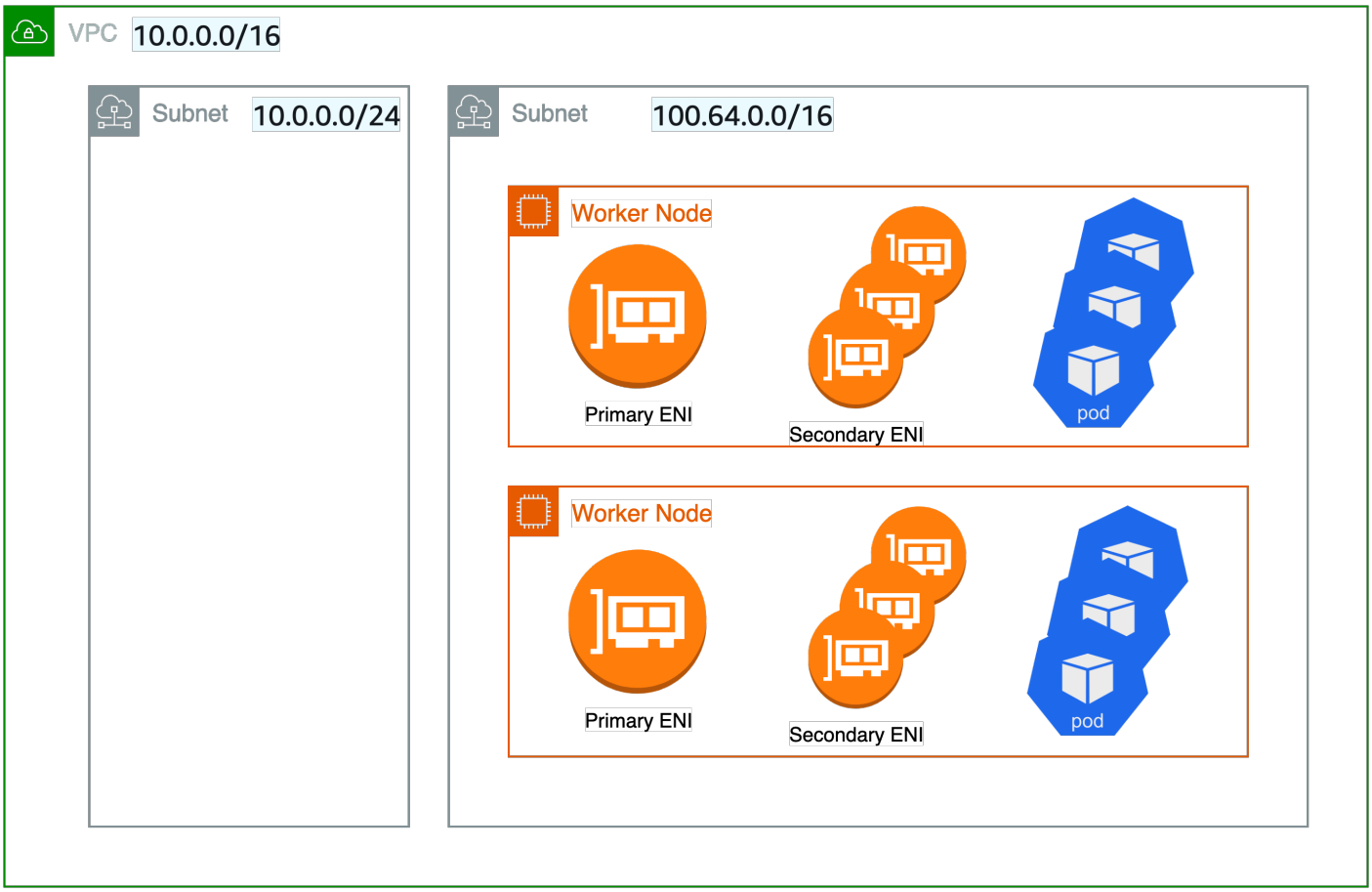
La arquitectura de red utilizada en la implementación de esta entrada de blog sigue las recomendaciones de la documentación de Amazon VPC sobre Habilitar la comunicación entre redes superpuestas. Como se muestra en esta entrada de blog, puede ampliar el uso de la puerta de enlace NAT privada junto con RFC6598 las direcciones para gestionar los problemas de agotamiento de la IP privada de los clientes. Los clústeres EKS y los nodos de trabajo se implementan en el rango CIDR secundario no enrutable de 100.64.0.0/16 VPC, mientras que la puerta de enlace NAT privada y la puerta de enlace NAT se implementan en los rangos CIDR enrutables. RFC1918 El blog explica cómo se utiliza una puerta de enlace de tránsito para conectarse a fin de facilitar la comunicación entre rangos de CIDR superpuestos que no se VPCs pueden enrutar. VPCs Para los casos de uso en los que los recursos de EKS del rango de direcciones no enrutables de una VPC necesitan comunicarse con otros VPCs que no tienen rangos de direcciones superpuestos, los clientes tienen la opción de usar el emparejamiento de VPC para interconectarlos. VPCs Este método podría suponer un posible ahorro de costes, ya que todo el tránsito de datos dentro de una zona de disponibilidad a través de una conexión de interconexión de VPC ahora es gratuito.



Red única para nodos y pods

Si necesitas aislar tus nodos y pods en una red específica por motivos de seguridad, te recomendamos que despliegues los nodos y los pods en una subred desde un bloque CIDR secundario más grande (por ejemplo, 100.64.0.0/8). Tras instalar el nuevo CIDR en su VPC, puede implementar otro grupo de nodos mediante el CIDR secundario y drenar los nodos originales para volver a implementar automáticamente los pods en los nuevos nodos de trabajo. Para obtener más información sobre cómo implementar esto, consulte esta entrada de blog.

Las redes personalizadas no se utilizan en la configuración que se muestra en el siguiente diagrama. Por el contrario, los nodos de trabajo de Kubernetes se implementan en subredes del rango CIDR de VPC secundario de la VPC, como 100.64.0.0/10. Puede mantener el clúster de EKS en funcionamiento (el plano de control permanecerá en el original). subnet/s), but the nodes and Pods will be moved to a secondary subnet/s Esta es otra técnica, aunque poco convencional, para mitigar el peligro de agotamiento de la propiedad intelectual en una VPC. Proponemos vaciar los nodos antiguos antes de volver a desplegarlos en los nuevos nodos de trabajo.



Automatice la configuración con etiquetas de zona de disponibilidad

Puede permitir que Kubernetes aplique automáticamente la zona de disponibilidad (AZ) ENIConfig correspondiente al nodo de trabajo.

Kubernetes añade automáticamente la etiqueta a tus nodos de trabajo. `topology.kubernetes.io/zone` Amazon EKS recomienda usar la zona de disponibilidad como nombre de configuración de ENI cuando solo tenga una subred secundaria (CIDR alternativo) por zona de disponibilidad. A continuación, puede establecer la etiqueta utilizada para descubrir el nombre de la configuración de ENI en. `topology.kubernetes.io/zone` Tenga en cuenta que la etiqueta `failure-domain.beta.kubernetes.io/zone` está obsoleta y se ha reemplazado por la etiqueta`topology.kubernetes.io/zone`.

1. Establezca el `name` campo en la zona de disponibilidad de su VPC.
2. Active la configuración automática mediante el siguiente comando
3. Establezca la etiqueta de configuración mediante el siguiente comando

```
kubectl set env daemonset aws-node -n kube-system
 "AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true"
kubectl set env daemonset aws-node -n kube-system
 "ENI_CONFIG_LABEL_DEF=topology.kubernetes.io/zone"
```

Si tiene varias subredes secundarias por zona de disponibilidad, debe crear una específica`ENI_CONFIG_LABEL_DEF`. Podría considerar la posibilidad de configurar `ENI_CONFIG_LABEL_DEF` `k8s.amazonaws.com/eniConfig`y etiquetar los nodos con nombres de ENIConfig personalizados, como `k8s.amazonaws.com/eniConfig=us-west-2a-subnet-1`y. `k8s.amazonaws.com/eniConfig=us-west-2a-subnet-2`

Sustituya los pods al configurar una red secundaria

La activación de redes personalizadas no modifica los nodos existentes. Las redes personalizadas son una acción disruptiva. En lugar de sustituir progresivamente todos los nodos de trabajo del clúster después de habilitar las redes personalizadas, le sugerimos que actualice la CloudFormation plantilla de AWS de la Guía de introducción de EKS con un recurso personalizado que llame a una función Lambda para actualizar el `aws-node` Daemonset con la variable de entorno a fin de habilitar las redes personalizadas antes de aprovisionar los nodos de trabajo.

Si tenías algún nodo en tu clúster con pods en ejecución antes de cambiarte a la función de red CNI personalizada, debes acordonar y drenar los nodos para cerrar los pods sin problemas y, a continuación, terminar los nodos. Solo los nodos nuevos que coincidan con la ENIConfig etiqueta o las anotaciones utilizan redes personalizadas y, por lo tanto, a los pods programados en estos nuevos nodos se les puede asignar una IP desde el CIDR secundario.

Calcule el número máximo de pods por nodo

Como el ENI principal del nodo ya no se usa para asignar direcciones IP de pods, se ha reducido el número de pods que se pueden ejecutar en un tipo de EC2 instancia determinado. Para evitar esta limitación, puedes usar la asignación de prefijos con redes personalizadas. Con la asignación de prefijos, cada IP secundaria se sustituye por un prefijo /28 en la secundaria. ENIs

Tenga en cuenta la cantidad máxima de pods para una instancia m5.large con redes personalizadas.

La cantidad máxima de pods que puedes ejecutar sin la asignación de prefijos es 29

- `3 ENIs - 1) * (10 secondary IPs per ENI - 1 + 2 = 20`

Al habilitar los adjuntos con prefijos, se aumenta la cantidad de pods a 290.

- `(3 ENIs - 1) * ((10 secondary IPs per ENI - 1) * 16 + 2 = 290`

Sin embargo, te sugerimos configurar el número máximo de pods en 110 en lugar de 290, ya que la instancia tiene un número bastante pequeño de dispositivos virtuales. CPUs En los casos más grandes, EKS recomienda un valor máximo de 250 pods. Al utilizar adjuntos de prefijo con tipos de instancias más pequeños (por ejemplo, m5.large), es posible que agote los recursos de CPU y memoria de la instancia mucho antes que sus direcciones IP.

> **Note**
>
> Cuando el prefijo CNI asigna un prefijo /28 a un ENI, tiene que ser un bloque contiguo de direcciones IP. Si la subred desde la que se genera el prefijo está muy fragmentada, es posible que no se pueda adjuntar el prefijo. Puede evitar que esto suceda creando una nueva VPC dedicada para el clúster o reservando en la subred un conjunto de CIDR exclusivamente para los adjuntos de prefijos. Visite la sección de reservas de CIDR de la subred para obtener más información sobre este tema.

Identifique el uso actual del espacio CG-NAT

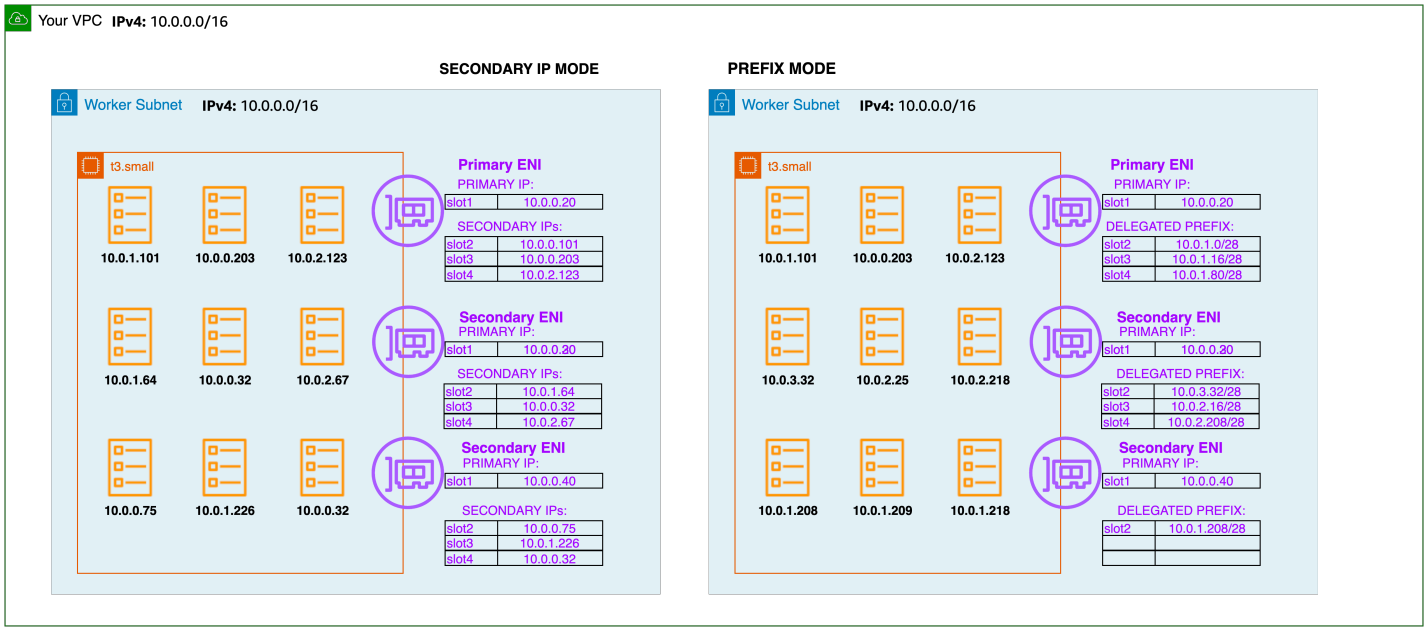
Las redes personalizadas le permiten mitigar el problema del agotamiento de la IP, pero no pueden resolver todos los desafíos. Si ya usa el espacio CG-NAT para su clúster, o simplemente no tiene la capacidad de asociar un CIDR secundario a la VPC de su clúster, le sugerimos que explore otras opciones, como usar una CNI alternativa o pasar a clústeres. IPv6

Modo de prefijo para Linux

El CNI de Amazon VPC asigna prefijos de red a las interfaces de EC2 red de Amazon para aumentar el número de direcciones IP disponibles para los nodos y aumentar la densidad de pods por nodo. Puede configurar la versión 1.9.0 o posterior del complemento CNI de Amazon VPC para IPv4 asignar IPv6 CIDRs y, en lugar de asignar, direcciones IP secundarias individuales a las interfaces de red.

El modo de prefijo está activado de forma predeterminada en los IPv6 clústeres y es la única opción admitida. La CNI de VPC asigna un IPv6 prefijo /80 a una ranura de un ENI. Consulte la IPv6 sección de esta guía para obtener más información.

Con el modo de asignación de prefijos, el número máximo de interfaces de red elásticas por tipo de instancia sigue siendo el mismo, pero ahora puede configurar Amazon VPC CNI para que asigne prefijos de direcciones /28 (16 direcciones IP IPv4), en lugar de asignar direcciones IPv4 individuales a las ranuras de las interfaces de red. Cuando `ENABLE_PREFIX_DELEGATION` se establece en VPC verdadera, el CNI asigna una dirección IP a un pod a partir del prefijo asignado a un ENI. Siga las instrucciones que se mencionan en la guía del usuario de EKS para activar el modo IP de prefijo.



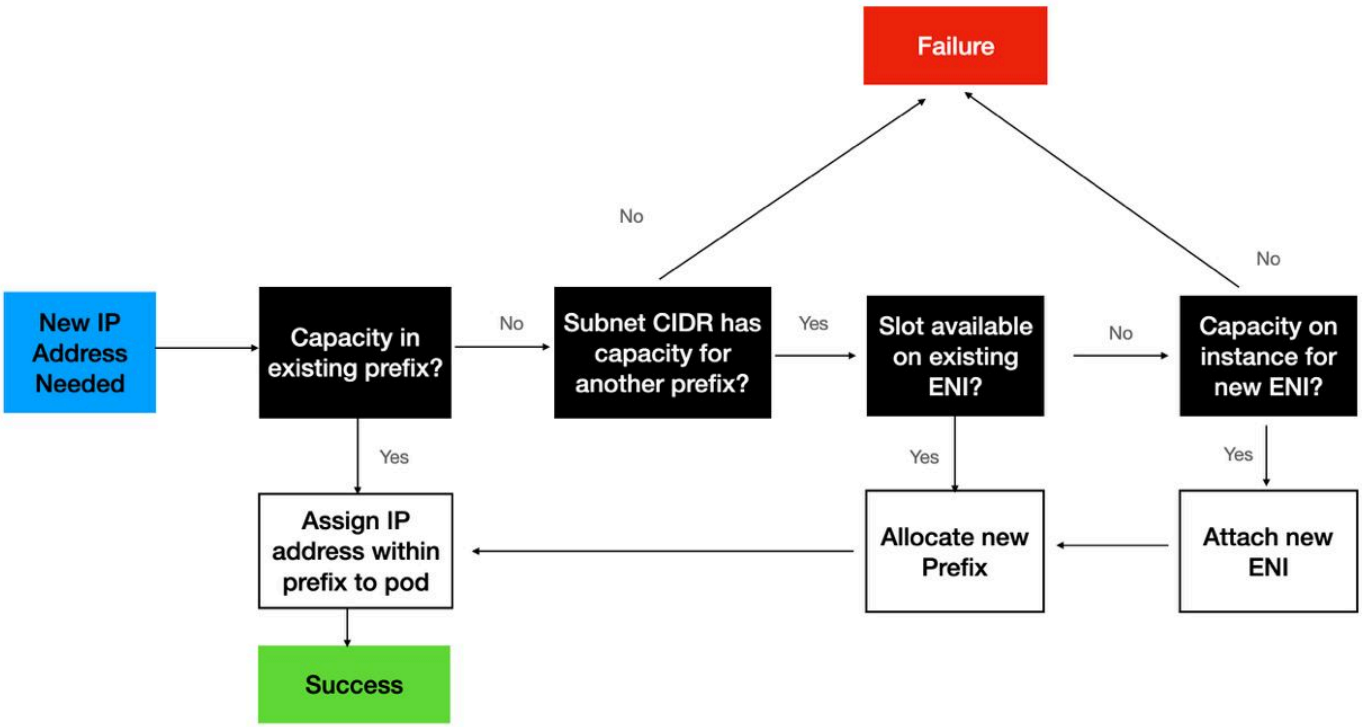
El número máximo de direcciones IP que puede asignar a una interfaz de red depende del tipo de instancia. Cada prefijo que asigna a una interfaz de red cuenta como una dirección IP. Por ejemplo, una `c5.large` instancia tiene un límite de `10` IPv4 direcciones por interfaz de red. Cada interfaz de red de esta instancia tiene una IPv4 dirección principal. Si una interfaz de red no tiene IPv4 direcciones secundarias, puede asignar hasta 9 prefijos a la interfaz de red. Por cada IPv4 dirección adicional que asigne a una interfaz de red, puede asignar un prefijo menos a la interfaz de red. Consulte la EC2 documentación de AWS sobre las direcciones IP por interfaz de red y por tipo de instancia y la asignación de prefijos a las interfaces de red.

Durante la inicialización del nodo de trabajo, la CNI de la VPC asigna uno o más prefijos al ENI principal. El CNI asigna previamente un prefijo para acelerar el arranque del pod al mantener una piscina caliente. El número de prefijos que deben guardarse en una piscina caliente se puede controlar configurando variables de entorno.

- `WARM_PREFIX_TARGET`, el número de prefijos que se asignarán por encima de las necesidades actuales.

- `WARM_IP_TARGET`, el número de direcciones IP que se van a asignar por encima de las necesidades actuales.
- `MINIMUM_IP_TARGET`, el número mínimo de direcciones IP que estarán disponibles en cualquier momento.
- `WARM_IP_TARGET`y `MINIMUM_IP_TARGET` si se establece, se `WARM_PREFIX_TARGET` anulará.

A medida que se programen más Pods, se solicitarán prefijos adicionales para el ENI actual. En primer lugar, la CNI de VPC intenta asignar un nuevo prefijo a una ENI existente. Si la ENI está al máximo de su capacidad, la CNI de la VPC intenta asignar una nueva ENI al nodo. Se ENIs adjuntará una nueva hasta que se alcance el límite máximo de ENI (definido por el tipo de instancia). Cuando se adjunta un nuevo ENI, ipamd asignará uno o más prefijos necesarios para mantener la `WARM_PREFIX_TARGET` configuración `WARM_IP_TARGET` y`MINIMUM_IP_TARGET`.



Recomendaciones

Utilice el modo de prefijo cuando

Utilice el modo de prefijo si tiene un problema de densidad de pods en los nodos de trabajo. Para evitar errores de CNI en la VPC, se recomienda examinar las subredes en busca de bloques de

direcciones contiguos para el prefijo /28 antes de migrar al modo de prefijo. Consulte la sección «Utilizar reservas de subred para evitar la fragmentación de subredes ()» para obtener información sobre las reservas de subredes. IPv4

Para garantizar la compatibilidad con versiones anteriores, el límite máximo de pods está configurado para admitir el modo IP secundario. Para aumentar la densidad de los módulos, especifique el `max-pods` valor de Kubelet y `--use-max-pods=false` los datos de usuario de los nodos. Puedes considerar usar el script max-pod-calculator.sh para calcular la cantidad máxima de pods recomendada por EKS para un tipo de instancia determinado. Consulte la guía del usuario de EKS para ver, por ejemplo, los datos de usuario.

```
./max-pods-calculator.sh --instance-type m5.large --cni-version ``1.9``.0 --cni-prefix-delegation-enabled
```

El modo de asignación de prefijos es especialmente relevante para los usuarios de redes CNI personalizadas, donde el ENI principal no se utiliza para los pods. Con la asignación de prefijos, puedes adjuntar más IPs en casi todos los tipos de instancias de Nitro, incluso sin el ENI principal que se utiliza en los pods.

Evita el modo de prefijo cuando

Si la subred está muy fragmentada y no tiene suficientes direcciones IP disponibles para crear prefijos /28, evite usar el modo de prefijo. El prefijo adjunto puede fallar si la subred desde la que se produce el prefijo está fragmentada (una subred muy utilizada con direcciones IP secundarias dispersas). Este problema se puede evitar creando una nueva subred y reservando un prefijo.

En el modo de prefijo, los pods comparten el grupo de seguridad asignado a los nodos de trabajo. Considere la posibilidad de usar grupos de seguridad para los pods si tiene un requisito de seguridad para cumplir con los requisitos de seguridad de la red mediante la ejecución de aplicaciones con diferentes requisitos de seguridad de red en recursos informáticos compartidos.

Usa tipos de instancias similares en el mismo grupo de nodos

El grupo de nodos puede contener instancias de muchos tipos. Si una instancia tiene un número máximo de pods bajo, ese valor se aplica a todos los nodos del grupo de nodos. Considera usar tipos de instancias similares en un grupo de nodos para maximizar el uso de los nodos. Si utilizas Karpenter para el escalado automatizado de nodos, te recomendamos configurar node.kubernetes.io/instance-type en la parte de requisitos de la API del aprovisionador.

> **Warning**
>
> El número máximo de pods para todos los nodos de un grupo de nodos concreto viene definido por el número máximo de pods más bajo de todos los tipos de instancias individuales del grupo de nodos.

`WARM_PREFIX_TARGET`Configúrelo para conservar IPv4 las direcciones

El valor predeterminado del manifiesto de instalación `WARM_PREFIX_TARGET` es 1. En la mayoría de los casos, el valor recomendado de 1 para `WARM_PREFIX_TARGET` proporcionará una buena combinación de tiempos de lanzamiento rápidos del pod y, al mismo tiempo, minimizará las direcciones IP no utilizadas asignadas a la instancia.

Si necesitas conservar aún más el uso `WARM_IP_TARGET` y la `MINIMUM_IP_TARGET` configuración de IPv4 las direcciones por nodo, que se anulan `WARM_PREFIX_TARGET` cuando se configuran. Al establecer un valor inferior `WARM_IP_TARGET` a 16, puede evitar que el CNI mantenga un prefijo sobrante completo adjunto.

Prefiere asignar nuevos prefijos en lugar de adjuntar un nuevo ENI

Asignar un prefijo adicional a un ENI existente es una operación de EC2 API más rápida que crear y adjuntar un nuevo ENI a la instancia. El uso de prefijos mejora el rendimiento y, al mismo tiempo, reduce la asignación de direcciones. IPv4 Por lo general, adjuntar un prefijo se completa en menos de un segundo, mientras que adjuntar un nuevo ENI puede tardar hasta 10 segundos. En la mayoría de los casos de uso, el CNI solo necesitará un ENI por nodo trabajador cuando se ejecute en modo de prefijo. Si puede permitirse (en el peor de los casos) hasta 15 sin usar IPs por nodo, le recomendamos encarecidamente que utilice el nuevo modo de red con asignación de prefijos y aproveche las mejoras de rendimiento y eficiencia que conlleva.

Utilice las reservas de subred para evitar la fragmentación de subredes () IPv4

Al EC2 asignar un IPv4 prefijo /28 a un ENI, debe ser un bloque contiguo de direcciones IP de la subred. Si la subred desde la que se genera el prefijo está fragmentada (una subred muy utilizada con direcciones IP secundarias dispersas), es posible que se produzca un error en el adjunto del prefijo y que aparezca el siguiente mensaje de error en los registros de la CNI de la VPC:

```
failed to allocate a private IP/Prefix address: InsufficientCidrBlocks: There are not
 enough free cidr blocks in the specified subnet to satisfy the request.
```

Para evitar la fragmentación y disponer de suficiente espacio contiguo para crear prefijos, puede utilizar las reservas CIDR de subred de VPC para reservar espacio IP dentro de una subred para uso exclusivo de los prefijos. Una vez que haya creado una reserva, el complemento CNI de VPC lo llamará EC2 APIs para asignar prefijos que se asignan automáticamente desde el espacio reservado.

Se recomienda crear una nueva subred, reservar espacio para los prefijos y habilitar la asignación de prefijos con la CNI de VPC para los nodos de trabajo que se ejecutan en esa subred. Si la nueva subred está dedicada únicamente a los pods que se ejecutan en tu clúster de EKS con la asignación de prefijos CNI de VPC habilitada, puedes saltarte el paso de reserva de prefijos.

Evite degradar el CNI de VPC

El modo de prefijo funciona con la versión 1.9.0 y posteriores del CNI de VPC. Debe evitarse la degradación del complemento CNI de Amazon VPC a una versión anterior a la 1.9.0 una vez que se habilite el modo de prefijo y se asignen los prefijos. ENIs Debe eliminar y volver a crear los nodos si decide degradar la CNI de la VPC.

Sustituya todos los nodos durante la transición a la delegación de prefijos

Se recomienda encarecidamente crear nuevos grupos de nodos para aumentar el número de direcciones IP disponibles, en lugar de sustituir de forma sucesiva los nodos de trabajo existentes. Acordona y vacía todos los nodos existentes para desalojar de forma segura todos los pods existentes. Para evitar interrupciones en el servicio, te sugerimos implementar los presupuestos de interrupción de módulos en tus clústeres de producción para cargas de trabajo críticas. A los pods de los nodos nuevos se les asignará una IP a partir de un prefijo asignado a un ENI. Tras confirmar que los pods se están ejecutando, puedes eliminar los nodos y grupos de nodos antiguos. Si utilizas grupos de nodos gestionados, sigue los pasos que se mencionan aquí para eliminar un grupo de nodos de forma segura.

Modo de prefijo para Windows

En Amazon EKS, el controlador de recursos de la VPC asigna de forma predeterminada una dirección IP secundaria a cada pod que se ejecuta en un host de Windows. Esta dirección IP es una dirección enrutable por VPC que se asigna desde la subred del host. En Linux, cada ENI adjunto a la instancia tiene varias ranuras que se pueden rellenar con una dirección IP secundaria o un CIDR /28 (un prefijo). Sin embargo, los hosts Windows solo admiten un único ENI y las ranuras disponibles. El uso exclusivo de direcciones IP secundarias puede limitar artificialmente la cantidad de pods que se pueden ejecutar en un host de Windows, incluso cuando haya una gran cantidad de direcciones IP disponibles para su asignación.

Para aumentar la densidad de los pods en los hosts de Windows, especialmente cuando se utilizan tipos de instancias más pequeños, puede habilitar la delegación de prefijos para los nodos de Windows. Cuando la delegación de prefijos está habilitada, los IPv4 prefijos /28 se asignan a las ranuras ENI en lugar de a las direcciones IP secundarias. La delegación de prefijos se puede habilitar añadiendo la `enable-windows-prefix-delegation: "true"` entrada al mapa de configuración. `amazon-vpc-cni` Este es el mismo mapa de configuración en el que debe establecer la `enable-windows-ipam: "true"` entrada para habilitar la compatibilidad con Windows.

Siga las instrucciones que se mencionan en la guía del usuario de EKS para habilitar el modo de delegación de prefijos en los nodos de Windows.

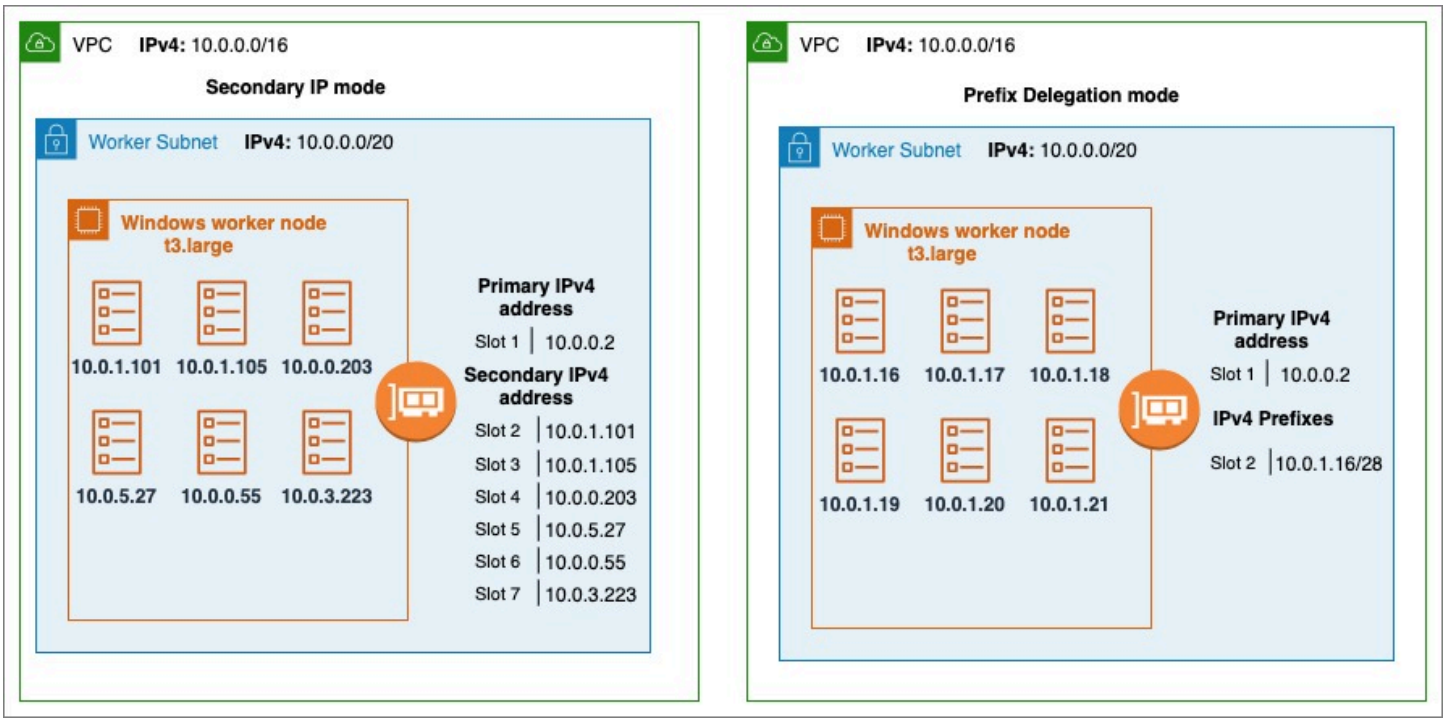


Figura: Comparación del modo de IP secundaria con el modo de delegación de prefijos

La cantidad máxima de direcciones IP que puede asignar a una interfaz de red depende del tipo de instancia y de su tamaño. Cada prefijo asignado a una interfaz de red ocupa una ranura disponible. Por ejemplo, una `c5.large` instancia tiene un límite de `10` ranuras por interfaz de red. La primera ranura de una interfaz de red siempre la ocupa la dirección IP principal de la interfaz, lo que deja 9 ranuras para prefijos y direcciones IP and/or secundarias. Si a estas ranuras se les asignan prefijos, el nodo puede admitir 144 direcciones IP (9 x 16), mientras que si se les asignan direcciones IP secundarias, solo puede admitir 9 direcciones IP. Consulte la documentación sobre las direcciones IP por interfaz de red por tipo de instancia y la asignación de prefijos a las interfaces de red para obtener más información.

Durante la inicialización del nodo de trabajo, el controlador de recursos de la VPC asigna uno o más prefijos al ENI principal para acelerar el inicio del pod al mantener un conjunto caliente de direcciones IP. El número de prefijos que se guardarán en una piscina caliente se puede controlar configurando los siguientes parámetros de configuración en el mapa de configuración. `amazon-vpc-cni`

- `warm-prefix-target`, el número de prefijos que se van a asignar por encima de las necesidades actuales.
- `warm-ip-target`, el número de direcciones IP que se van a asignar por encima de las necesidades actuales.
- `minimum-ip-target`, el número mínimo de direcciones IP que estarán disponibles en cualquier momento.
- `warm-ip-target` and/or `minimum-ip-target`si está establecido, se `warm-prefix-target` anulará.

A medida que haya más pods programados en el nodo, se solicitarán prefijos adicionales para el ENI existente. Cuando se programa un pod en el nodo, el controlador de recursos de VPC primero intenta asignar una IPv4 dirección a partir de los prefijos existentes en el nodo. Si eso no es posible, se solicitará un nuevo IPv4 prefijo siempre que la subred tenga la capacidad requerida.

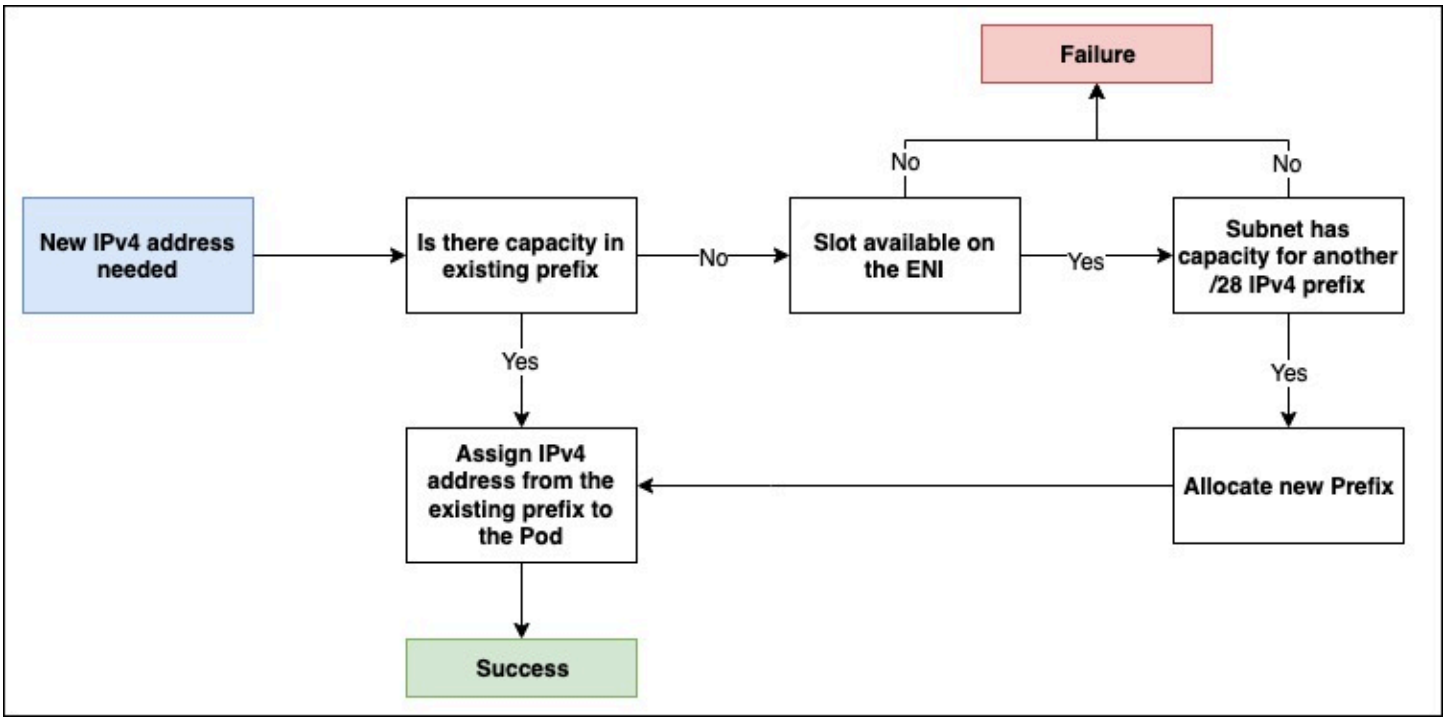


Figura: Flujo de trabajo durante la asignación de IPv4 direcciones al pod

Recomendaciones

Utilice la delegación de prefijos cuando

Utilice la delegación de prefijos si tiene problemas de densidad de pods en los nodos de trabajo. Para evitar errores, recomendamos examinar las subredes en busca de bloques de direcciones contiguos para el prefijo /28 antes de migrar al modo de prefijo. Consulte la sección «Utilizar reservas de subred para evitar la fragmentación de subredes ()» para obtener más información sobre las reservas de subredes. IPv4

De forma predeterminada, los nodos de Windows están configurados `max-pods` en. `110` Para la gran mayoría de los tipos de instancias, esto debería ser suficiente. Si quieres aumentar o disminuir este límite, añade lo siguiente al comando bootstrap en tus datos de usuario:

```
-KubeletExtraArgs '--max-pods=example-value'
```

Para obtener más información sobre los parámetros de configuración del bootstrap para los nodos de Windows, consulte la documentación aquí.

Evite la delegación de prefijos cuando

Si la subred está muy fragmentada y no tiene suficientes direcciones IP disponibles para crear prefijos /28, evite usar el modo de prefijo. El prefijo adjunto puede fallar si la subred desde la que se produce el prefijo está fragmentada (una subred muy utilizada con direcciones IP secundarias dispersas). Este problema se puede evitar creando una nueva subred y reservando un prefijo.

Configure los parámetros para la delegación de prefijos a fin de conservar las direcciones IPv4

`warm-prefix-targetwarm-ip-target`, y se `minimum-ip-target` puede utilizar para ajustar con precisión el comportamiento del escalado previo y el escalado dinámico con prefijos. De forma predeterminada, se utilizan los siguientes valores:

```
warm-ip-target: "1"
minimum-ip-target: "3"
```

Al ajustar estos parámetros de configuración, puedes lograr un equilibrio óptimo entre conservar las direcciones IP y garantizar una menor latencia del pod debido a la asignación de la dirección IP. Para obtener más información sobre estos parámetros de configuración, consulta la documentación aquí.

Utilice las reservas de subred para evitar la fragmentación de subredes () IPv4

Al EC2 asignar un IPv4 prefijo /28 a un ENI, debe ser un bloque contiguo de direcciones IP de la subred. Si la subred desde la que se genera el prefijo está fragmentada (una subred muy utilizada con direcciones IP secundarias dispersas), es posible que se produzca un error en el adjunto del prefijo y se produzca el siguiente evento de nodo:

```
InsufficientCidrBlocks: The specified subnet does not have enough free cidr blocks to
 satisfy the request
```

Para evitar la fragmentación y disponer de suficiente espacio contiguo para crear prefijos, utilice las reservas CIDR de subred de VPC para reservar espacio IP dentro de una subred para el uso exclusivo de los prefijos. Una vez que haya creado una reserva, las direcciones IP de los bloques reservados no se asignarán a otros recursos. De esta forma, el controlador de recursos de VPC podrá obtener los prefijos disponibles durante la llamada de asignación al nodo ENI.

Se recomienda crear una nueva subred, reservar espacio para los prefijos y habilitar la asignación de prefijos para los nodos de trabajo que se ejecutan en esa subred. Si la nueva subred está dedicada únicamente a los pods que se ejecutan en tu clúster de EKS con la delegación de prefijos habilitada, puedes saltarte el paso de reserva de prefijos.

Sustituya todos los nodos al migrar del modo de IP secundaria al modo de delegación de prefijos o viceversa

Se recomienda encarecidamente crear nuevos grupos de nodos para aumentar el número de direcciones IP disponibles en lugar de sustituir los nodos de trabajo existentes de forma sucesiva.

Si se utilizan grupos de nodos autogestionados, los pasos para la transición serían los siguientes:

- Aumente la capacidad de su clúster para que los nuevos nodos puedan acomodar sus cargas de trabajo
- Activa/desactiva la función de delegación de prefijos para Windows
- Acordona y vacía todos los nodos existentes para desalojar de forma segura todos tus pods existentes. Para evitar interrupciones en el servicio, te sugerimos implementar los presupuestos de interrupción de módulos en tus clústeres de producción para cargas de trabajo críticas.
- Tras confirmar que los pods se están ejecutando, puedes eliminar los nodos y grupos de nodos antiguos. A los pods de los nodos nuevos se les asignará una IPv4 dirección a partir de un prefijo asignado al nodo ENI.

Al utilizar grupos de nodos gestionados, los pasos para la transición serían:

- Habilitar/deshabilitar la función de delegación de prefijos para Windows
- Actualice el grupo de nodos siguiendo los pasos que se mencionan aquí. Realiza pasos similares a los anteriores, pero los gestiona EKS.

> **Warning**
>
> Ejecuta todos los pods de un nodo en el mismo modo

En el caso de Windows, te recomendamos que evites ejecutar los pods tanto en el modo de IP secundaria como en el modo de delegación de prefijos al mismo tiempo. Esta situación puede surgir al migrar del modo de IP secundaria al modo de delegación de prefijo o viceversa con cargas de trabajo de Windows en ejecución.

Si bien esto no afectará a los pods en ejecución, puede haber inconsistencias con respecto a la capacidad de direcciones IP del nodo. Por ejemplo, pensemos en un nodo t3.xlarge que tiene 14 ranuras para direcciones secundarias. IPv4 Si utiliza 10 pods, las direcciones IP secundarias consumirán 10 ranuras del ENI. Tras activar la delegación de prefijos, la capacidad anunciada en el servidor kube-api sería de 244 (14 ranuras * 16 direcciones IP por prefijo), pero la capacidad real en ese momento sería de 64 (4 ranuras restantes* 16 direcciones por prefijo). Esta incoherencia entre la cantidad de capacidad anunciada y la cantidad real de capacidad (ranuras restantes) puede provocar problemas si ejecutas más pods de las direcciones IP disponibles para su asignación.

Dicho esto, puedes usar la estrategia de migración descrita anteriormente para hacer una transición segura de tus Pods de una dirección IP secundaria a direcciones obtenidas a partir de prefijos. Al cambiar de un modo a otro, los Pods seguirán funcionando con normalidad y:

- Al cambiar del modo IP secundario al modo de delegación de prefijos, no se publicarán las direcciones IP secundarias asignadas a los pods en ejecución. Se asignarán prefijos a los espacios libres. Una vez que se cierre un módulo, se liberarán la IP secundaria y la ranura que estaba utilizando.
- Al cambiar del modo de delegación de prefijos al modo de IP secundaria, se liberará un prefijo cuando todos los que estén IPs dentro de su rango ya no estén asignados a los pods. Si se asigna alguna IP del prefijo a un pod, ese prefijo se conservará hasta que los pods terminen.

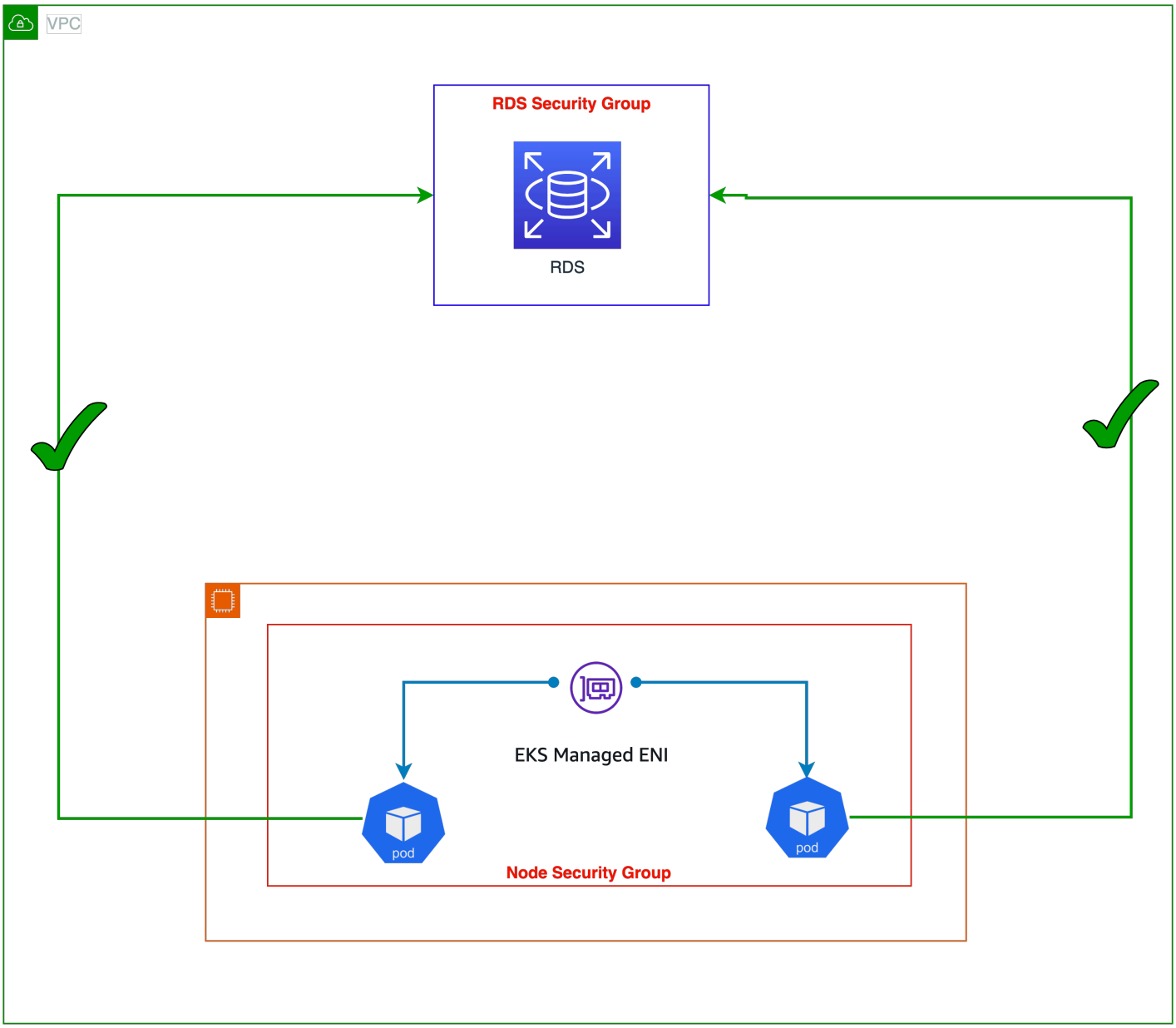
Problemas de depuración con la delegación de prefijos

Puede utilizar nuestra guía de depuración aquí para analizar en profundidad el problema al que se enfrenta con la delegación de prefijos en Windows.

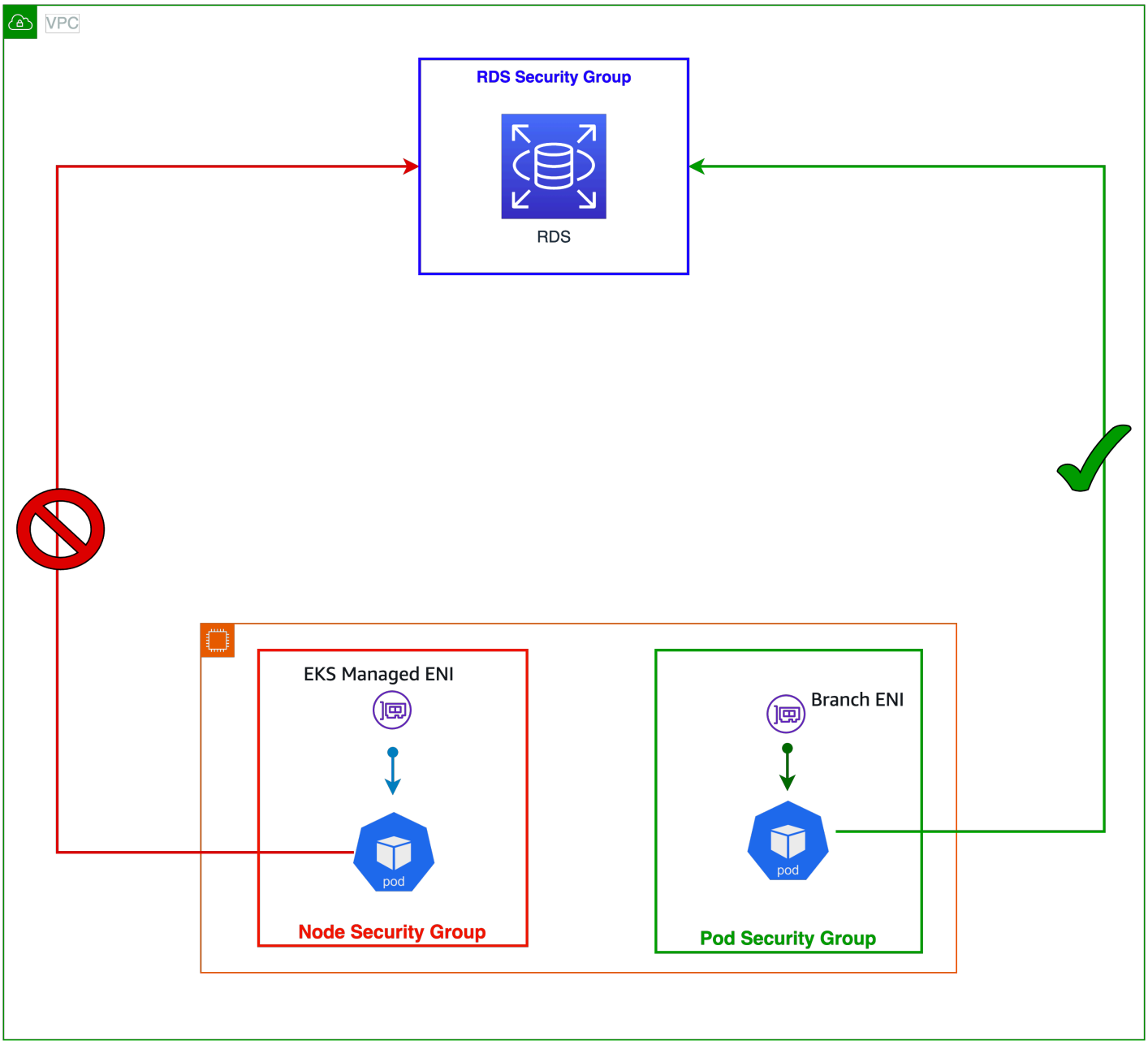
Grupos de seguridad por pod

Un grupo de seguridad de AWS actúa como un firewall virtual para que EC2 las instancias controlen el tráfico entrante y saliente. De forma predeterminada, la CNI de Amazon VPC utilizará grupos de seguridad asociados al ENI principal del nodo. Más específicamente, todos los ENI asociados a la instancia tendrán los mismos grupos de EC2 seguridad. Por lo tanto, cada pod de un nodo comparte los mismos grupos de seguridad que el nodo en el que se ejecuta.

Como se muestra en la imagen siguiente, todos los pods de aplicaciones que funcionen en los nodos de trabajo tendrán acceso al servicio de base de datos de RDS (teniendo en cuenta que el RDS inbound permite el grupo de seguridad de los nodos). Los grupos de seguridad son demasiado generales porque se aplican a todos los pods que se ejecutan en un nodo. Los grupos de seguridad para Pods permiten segmentar la red de las cargas de trabajo, lo que constituye una parte esencial de una buena estrategia de defensa exhaustiva.



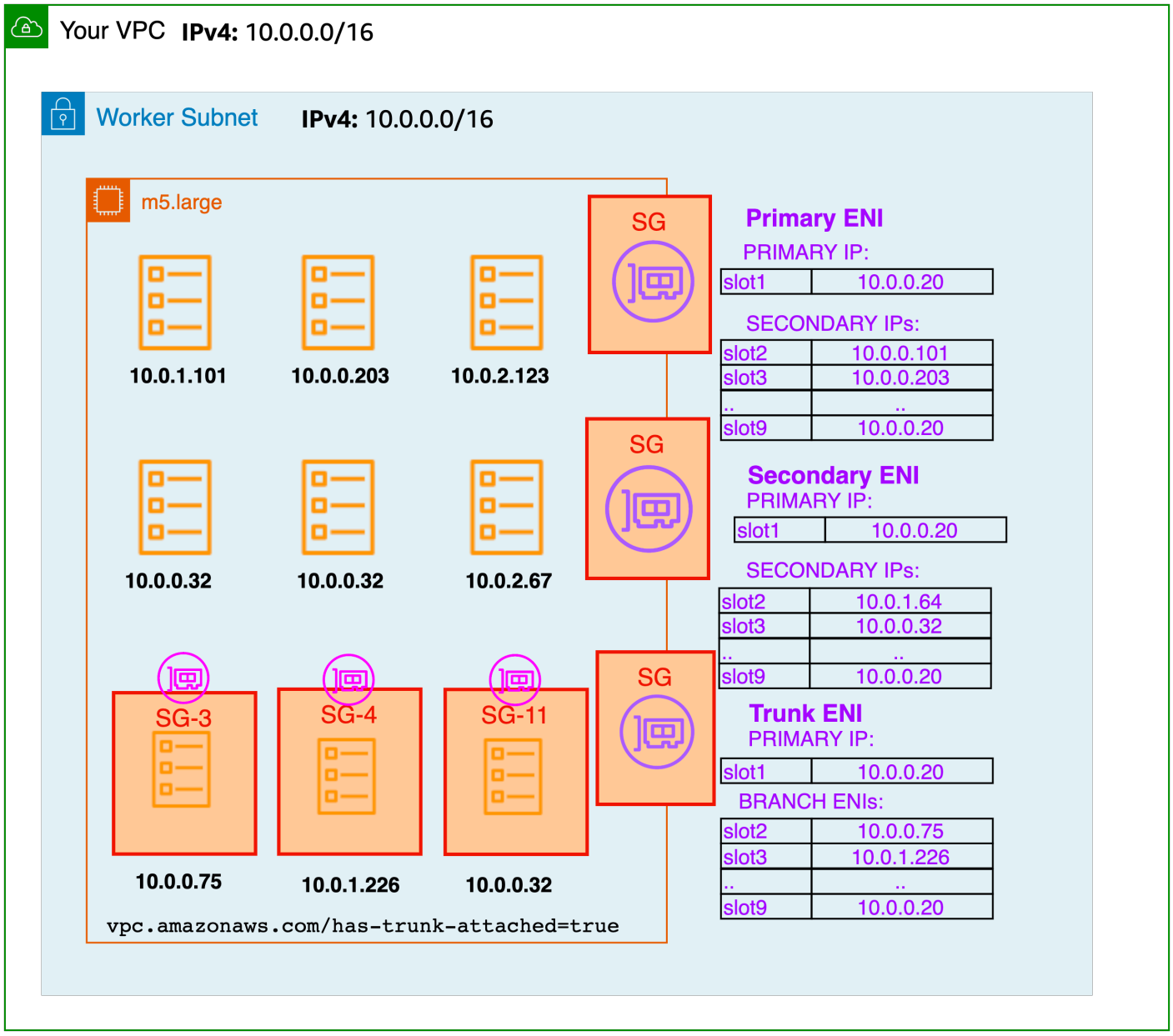
Con los grupos de seguridad para los pods, puede mejorar la eficiencia informática al ejecutar aplicaciones con diferentes requisitos de seguridad de red en recursos informáticos compartidos. Se pueden definir varios tipos de reglas de seguridad, como Pod-to-Pod los servicios de Pod-to-External AWS, en un solo lugar con grupos de EC2 seguridad y aplicarlos a las cargas de trabajo con Kubernetes nativo. APIs La siguiente imagen muestra los grupos de seguridad aplicados a nivel de pod y cómo simplifican la implementación de la aplicación y la arquitectura de nodos. El pod ahora puede acceder a la base de datos de Amazon RDS.



Puedes habilitar los grupos de seguridad para los pods configurando el `ENABLE_POD_ENI=true` CNI de VPC. Una vez activado, el controlador de recursos de VPC que se ejecuta en el plano de control (gestionado por EKS) crea y conecta una interfaz troncal denominada «`aws-k8 «`s-trunk-enial nodo. La interfaz troncal actúa como una interfaz de red estándar conectada a la instancia. Para administrar las interfaces troncales, debe añadir la política `AmazonEKSVPCResourceController` administrada al rol de clúster que acompaña a su clúster de Amazon EKS.

El controlador también crea interfaces de rama denominadas «aws-k8s-branch-eni» y las asocia a la interfaz troncal. A los pods se les asigna un grupo de seguridad mediante el recurso

SecurityGroupPolicypersonalizado y se asocian a una interfaz de sucursal. Como los grupos de seguridad se especifican con interfaces de red, ahora podemos programar pods que requieran grupos de seguridad específicos en estas interfaces de red adicionales. Consulte la sección de la guía del usuario de EKS sobre grupos de seguridad para pods, incluidos los requisitos previos de implementación.



La capacidad de la interfaz de la sucursal se suma a los límites de tipos de instancia existentes para las direcciones IP secundarias. Los pods que usan grupos de seguridad no se tienen en cuenta en la fórmula de max-pods y, si usas un grupo de seguridad para los pods, debes considerar la posibilidad

de aumentar el valor de max-pods o estar de acuerdo con ejecutar menos pods de los que el nodo realmente puede admitir.

Un m5.large puede tener hasta 9 interfaces de red de sucursales y hasta 27 direcciones IP secundarias asignadas a sus interfaces de red estándar. Como se muestra en el siguiente ejemplo, el número máximo de pods predeterminado para un m5.large es 29, y EKS cuenta los pods que utilizan grupos de seguridad para calcular el número máximo de pods. Consulta la guía del usuario de EKS para obtener instrucciones sobre cómo cambiar los max-pods de los nodos.

Cuando los grupos de seguridad para los pods se utilizan en combinación con redes personalizadas, se utiliza el grupo de seguridad definido en los grupos de seguridad para los pods en lugar del grupo de seguridad especificado en el. ENIConfig Como resultado, cuando las redes personalizadas estén habilitadas, evalúe cuidadosamente el orden de los grupos de seguridad y utilice los grupos de seguridad por pod.

Recomendaciones

Deshabilite TCP Early Demux para Liveness Probe

Si utilizas sondas activas o de preparación, también tendrás que deshabilitar la demultiplexación temprana de TCP para que el kubelet pueda conectarse a los pods de las interfaces de red de las sucursales a través de TCP. Esto solo es necesario en modo estricto. Para ello, ejecute el siguiente comando:

```
kubectl edit daemonset aws-node -n kube-system
```

En la `initContainer` sección, cambie el valor de `DISABLE_TCP_EARLY_DEMUX` a `true`.

Utilice Security Group For Pods para aprovechar la inversión existente en configuración de AWS.

Los grupos de seguridad facilitan la restricción del acceso de la red a los recursos de la VPC, como las bases de datos o las instancias de RDS. EC2 Una clara ventaja de los grupos de seguridad por pod es la oportunidad de reutilizar los recursos de grupos de seguridad de AWS existentes. Si utiliza grupos de seguridad como firewall de red para limitar el acceso a sus servicios de AWS, le sugerimos que aplique grupos de seguridad a los pods mediante Branch ENIs. Considere la posibilidad de utilizar grupos de seguridad para los pods si va a transferir aplicaciones de EC2 instancias a EKS y limitar el acceso a otros servicios de AWS con grupos de seguridad.

Configure el modo de aplicación del grupo de seguridad de Pod

La versión 1.11 del complemento CNI de Amazon VPC agregó una nueva configuración denominada `POD_SECURITY_GROUP_ENFORCING_MODE` («modo de aplicación»). El modo de aplicación controla qué grupos de seguridad se aplican al pod y si la NAT de origen está habilitada. Puede especificar el modo de aplicación como estricto o estándar. El valor predeterminado es estricto, lo que refleja el comportamiento anterior del CNI de la VPC con `ENABLE_POD_ENI` el valor establecido en. `true`

En el modo estricto, solo se aplican los grupos de seguridad ENI de la sucursal. La NAT de origen también está desactivada.

En el modo estándar, se aplican los grupos de seguridad asociados tanto al ENI principal como al ENI de sucursal (asociado al módulo). El tráfico de red debe cumplir con ambos grupos de seguridad.

> **Warning**
>
> Cualquier cambio de modo solo afectará a los pods recién lanzados. Los pods existentes utilizarán el modo que se configuró cuando se creó el pod. Los clientes deberán reciclar los pods existentes con grupos de seguridad si quieren cambiar el comportamiento del tráfico.

Modo obligatorio: utilice el modo estricto para aislar el tráfico de nodos y pods:

De forma predeterminada, los grupos de seguridad de los pods están configurados en «modo estricto». Usa esta configuración si debes separar completamente el tráfico del pod del resto del tráfico del nodo. En modo estricto, la NAT de origen está desactivada para poder utilizar los grupos de seguridad salientes del ENI de la sucursal.

> **Warning**
>
> Cuando se habilita el modo estricto, todo el tráfico saliente de un pod abandonará el nodo y entrará en la red de VPC. El tráfico entre los pods del mismo nodo pasará por la VPC. Esto aumenta el tráfico de VPC y limita las funciones basadas en nodos. No NodeLocal DNSCache es compatible con el modo estricto.

Modo obligatorio: utilice el modo estándar en las siguientes situaciones

La IP de origen del cliente es visible para los contenedores del pod

Si necesitas mantener la IP de origen del cliente visible para los contenedores del pod, considera `POD_SECURITY_GROUP_ENFORCING_MODE` configurarla `enstandard`. Los servicios de Kubernetes admiten externalTrafficPolicy =local para permitir la conservación de la IP de origen del cliente (tipo clúster de tipo predeterminado). Ahora puedes ejecutar servicios de Kubernetes de este tipo NodePort y LoadBalancer usar instancias de destino externalTrafficPolicy configuradas en Local en el modo estándar. `Local`conserva la IP de origen del cliente y evita un segundo salto de tipo Servicios LoadBalancer . NodePort

Despliegue NodeLocal DNSCache

Cuando utilices grupos de seguridad para los pods, configura el modo estándar para que sea compatible con los pods que los usen NodeLocal DNSCache. NodeLocal DNSCache mejora el rendimiento del DNS del clúster al ejecutar un agente de almacenamiento en caché de DNS en los nodos del clúster como un DaemonSet. Esto ayudará a los pods que tienen los requisitos de QPS de DNS más altos a consultar los Kube-DNS/CoreDNS locales que tienen una caché local, lo que mejorará la latencia.

NodeLocal DNSCache no se admite en modo estricto, ya que todo el tráfico de red, incluso el del nodo, entra en la VPC.

Compatible con la política de red de Kubernetes

Recomendamos utilizar el modo de aplicación estándar cuando se utilice la política de red con los pods que tengan grupos de seguridad asociados.

Recomendamos encarecidamente utilizar grupos de seguridad para los pods a fin de limitar el acceso a nivel de red a los servicios de AWS que no forman parte de un clúster. Tenga en cuenta las políticas de red para restringir el tráfico de red entre los pods de un clúster, lo que suele East/West denominarse tráfico.

Identifique las incompatibilidades con los grupos de seguridad por pod

Las instancias basadas en Windows y las que no son de Nitro no admiten grupos de seguridad para los pods. Para utilizar grupos de seguridad con los pods, las instancias deben estar etiquetadas con. isTrunkingEnabled Utilice políticas de red para administrar el acceso entre pods en lugar de grupos de seguridad si sus pods no dependen de ningún servicio de AWS dentro o fuera de su VPC.

Utilice grupos de seguridad por pod para controlar de forma eficaz el tráfico a los servicios de AWS

Si una aplicación que se ejecuta en el clúster de EKS tiene que comunicarse con otro recurso de la VPC, por ejemplo, una base de datos de RDS, considere la posibilidad de SGs utilizarla para pods. Si bien hay motores de políticas que permiten especificar un CIDR o un nombre de DNS, son una opción menos óptima cuando se comunica con los servicios de AWS que tienen puntos de enlace que residen dentro de una VPC.

Por el contrario, las políticas de red de Kubernetes proporcionan un mecanismo para controlar el tráfico de entrada y salida tanto dentro como fuera del clúster. Se deben tener en cuenta las políticas de red de Kubernetes si su aplicación tiene dependencias limitadas de otros servicios de AWS. Puede configurar políticas de red que especifiquen reglas de salida en función de los rangos de CIDR para limitar el acceso a los servicios de AWS, a diferencia de la semántica nativa de AWS, por ejemplo. SGs Puede usar las políticas de red de Kubernetes para controlar el tráfico de red entre los pods (lo que suele denominarse East/West tráfico) y entre los pods y los servicios externos. Las políticas de red de Kubernetes se implementan en los niveles OSI 3 y 4.

Amazon EKS le permite utilizar motores de políticas de red como Calico y Cilium. De forma predeterminada, los motores de políticas de red no están instalados. Consulte las guías de instalación correspondientes para obtener instrucciones sobre cómo configurarlos. Para obtener más información sobre cómo utilizar la política de red, consulte las mejores prácticas de seguridad de EKS. La función de nombres de host DNS está disponible en las versiones empresariales de los motores de políticas de red, lo que podría resultar útil para controlar el tráfico entre Kubernetes Services/Pods y los recursos que se ejecutan fuera de AWS. Además, puede considerar la compatibilidad con nombres de host DNS para los servicios de AWS que no admiten grupos de seguridad de forma predeterminada.

Etiquete un solo grupo de seguridad para usar AWS Loadbalancer Controller

Cuando se asignan muchos grupos de seguridad a un pod, Amazon EKS recomienda etiquetar un único grupo de seguridad como `kubernetes.io/cluster/$name`compartido o propio. La etiqueta permite al controlador Loadbalancer de AWS actualizar las reglas de los grupos de seguridad para enrutar el tráfico a los pods. Si solo se asigna un grupo de seguridad a un pod, la asignación de una etiqueta es opcional. Los permisos establecidos en un grupo de seguridad son acumulativos, por lo que basta con etiquetar un único grupo de seguridad para que el controlador del balanceador de cargas localice y concilie las reglas. También ayuda a cumplir las cuotas predeterminadas definidas por los grupos de seguridad.

Configure la NAT para el tráfico saliente

La NAT de origen está deshabilitada para el tráfico saliente de los pods a los que se les han asignado grupos de seguridad. En el caso de los pods que utilizan grupos de seguridad que requieren acceso a Internet, lance nodos de trabajo en subredes privadas configuradas con una instancia o puerta de enlace de NAT y active la SNAT externa en la CNI.

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

Implemente pods con grupos de seguridad en subredes privadas

Los pods a los que se les asignan grupos de seguridad deben ejecutarse en nodos que estén desplegados en subredes privadas. Ten en cuenta que los pods con grupos de seguridad asignados implementados en subredes públicas no podrán acceder a Internet.

Verifica los terminationGracePeriodsegundos en el archivo de especificaciones del pod

Asegúrese de que no `terminationGracePeriodSeconds` sea cero en el archivo de especificaciones del pod (30 segundos por defecto). Esto es esencial para que Amazon VPC CNI elimine la red Pod del nodo de trabajo. Cuando se establece en cero, el complemento CNI no elimina la red Pod del host y el ENI de la sucursal no se limpia de manera efectiva.

Uso de grupos de seguridad para cápsulas con Fargate

Los grupos de seguridad para los pods que se ejecutan en Fargate funcionan de forma muy similar a los pods que se ejecutan en los nodos de EC2 trabajo. Por ejemplo, debe crear el grupo de seguridad antes de hacer referencia a él en el SecurityGroupPolicy que asocie a su Fargate Pod. De forma predeterminada, el grupo de seguridad del clúster se asigna a todos los Fargate Pods cuando no se asigna explícitamente SecurityGroupPolicy un a un Fargate Pod. Por motivos de simplicidad, es posible que desees añadir el grupo de seguridad del clúster al de un Fagate Pod SecurityGroupPolicy ; de lo contrario, tendrás que añadir las reglas mínimas del grupo de seguridad a tu grupo de seguridad. Puedes encontrar el grupo de seguridad del clúster mediante la API describe-cluster.

```
aws eks describe-cluster --name CLUSTER_NAME --query
'cluster.resourcesVpcConfig.clusterSecurityGroupId'
```

```
cat >my-fargate-sg-policy.yaml <<EOF
```

```
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
  name: my-fargate-sg-policy
  namespace: my-fargate-namespace
spec:
  podSelector:
    matchLabels:
      role: my-fargate-role
  securityGroups:
    groupIds:
      - cluster_security_group_id
      - my_fargate_pod_security_group_id
EOF
```

Las reglas mínimas de los grupos de seguridad se muestran aquí. Estas reglas permiten que los Fargate Pods se comuniquen con servicios integrados en el clúster, como kube-apiserver, kubelet y CoredNS. También necesitas añadir reglas para permitir las conexiones entrantes y salientes desde y hacia tu Fargate Pod. Esto permitirá que tu pod se comunique con otros pods o recursos de tu VPC. Además, debes incluir reglas para que Fargate extraiga imágenes de contenedores de Amazon ECR u otros registros de contenedores, como. DockerHub Para obtener más información, consulte los rangos de direcciones IP de AWS en la Referencia general de AWS.

Puedes usar los siguientes comandos para buscar los grupos de seguridad aplicados a un Fargate Pod.

```
kubectl get pod FARGATE_POD -o jsonpath='{.metadata.annotations.fargate\.amazonaws
\.com/pod-sg}{"\n"}'
```

Anota el eNIID del comando anterior.

```
aws ec2 describe-network-interfaces --network-interface-ids ENI_ID --query
 'NetworkInterfaces[*].Groups[*]'
```

Los pods Fargate existentes se deben eliminar y volver a crear para poder aplicar los nuevos grupos de seguridad. Por ejemplo, el siguiente comando inicia la implementación de la aplicación de ejemplo. Para actualizar pods específicos, puedes cambiar el espacio de nombres y el nombre de la implementación en el siguiente comando.

```
kubectl rollout restart -n example-ns deployment example-pod
```

Equilibrio de carga

Los balanceadores de carga reciben el tráfico entrante y lo distribuyen entre los destinos de la aplicación prevista alojada en un clúster de EKS. Esto mejora la resiliencia de la aplicación. Cuando se implementa en un clúster de EKS, el controlador de AWS Load Balancer creará y administrará los balanceadores de carga elásticos de AWS para ese clúster. Cuando se crea un servicio de Kubernetes de este tipoLoadBalancer , el controlador Load Balancer de AWS crea un Network Load Balancer (NLB) que equilibra la carga del tráfico recibido en la capa 4 del modelo OSI. Mientras que cuando se crea un objeto Ingress de Kubernetes, el controlador Load Balancer de AWS crea un Application Load Balancer (ALB) que equilibra la carga del tráfico en la capa 7 del modelo OSI.

Elegir el tipo de Load Balancer

La gama de AWS Elastic Load Balancing (ELB) admite los siguientes balanceadores de carga: balanceadores de carga de aplicaciones (ALB), balanceadores de carga de red (NLB), balanceadores de carga de puerta de enlace (GWLB) y balanceadores de carga clásicos (CLB). Esta sección de mejores prácticas se centrará en el ALB y el NLB, que son los dos más relevantes para los clústeres de EKS.

La consideración principal a la hora de elegir el tipo de balanceador de cargas son los requisitos de carga de trabajo.

Para obtener información más detallada y como referencia para todos los balanceadores de carga de AWS, consulte Comparaciones de productos

Elija el Application Load Balancer (ALB) si su carga de trabajo es HTTP/HTTPS

Si una carga de trabajo requiere el equilibrio de carga en la capa 7 del modelo OSI, se puede utilizar el controlador Load Balancer de AWS para aprovisionar un ALB; abordaremos el aprovisionamiento en la siguiente sección. El ALB se controla y configura mediante el recurso de Ingress mencionado anteriormente y enruta el tráfico HTTP o HTTPS a diferentes pods del clúster. El ALB ofrece a los clientes la flexibilidad de cambiar el algoritmo de enrutamiento del tráfico de la aplicación; el algoritmo de enrutamiento predeterminado es por turnos, y el algoritmo de enrutamiento de solicitudes menos pendientes también es una alternativa.

Elija el Network Load Balancer (NLB) si su carga de trabajo es TCP o si su carga de trabajo requiere la preservación de la IP de origen de los clientes

Un Network Load Balancer funciona en la cuarta capa (transporte) del modelo de interconexión de sistemas abiertos (OSI). Es adecuado para cargas de trabajo basadas en TCP y UDP. De forma predeterminada, Network Load Balancer también conserva la IP de origen de la dirección de los clientes al presentar el tráfico al pod.

Elija el Network Load Balancer (NLB) si su carga de trabajo no puede utilizar DNS

Otra razón clave para usar el NLB es si sus clientes no pueden utilizar el DNS. En este caso, es posible que el NLB se adapte mejor a su carga de trabajo, ya que los IPs de un Network Load Balancer son estáticos. Si bien se recomienda a los clientes utilizar el DNS para convertir nombres de dominio en direcciones IP cuando se conecten a los balanceadores de carga, si la aplicación de un cliente no admite la resolución de DNS y solo acepta códigos IPs rígidos, un NLB es más adecuado, ya que IPs son estáticos y permanecen iguales durante toda la vida del NLB.

Aprovisionamiento de balanceadores de carga

Tras determinar cuál es el Load Balancer más adecuado para sus cargas de trabajo, los clientes disponen de varias opciones para aprovisionar un balanceador de carga.

Aprovisione los balanceadores de carga mediante la implementación del controlador de balanceadores de carga de AWS

Existen dos métodos clave para aprovisionar los balanceadores de carga dentro de un clúster de EKS.

- Aprovechamiento del controlador de servicios en el proveedor de nube de AWS (antiguo)
- Aprovechar el controlador Load Balancer de AWS (recomendado)

De forma predeterminada, el controlador de servicio de Kubernetes, también conocido como controlador integrado en el árbol, reconcilia el tipo de recurso del servicio de Kubernetes. LoadBalancer Este controlador está integrado en el componente AWS Cloud Provider, que funciona como Kubernetes Cloud Controller Manager.

La configuración del Elastic Load Balancer aprovisionado se controla mediante anotaciones que se deben añadir al manifiesto del servicio de Kubernetes. Las anotaciones que utilizan el controlador de servicio y el controlador de AWS Load Balancer son diferentes.

El controlador de servicio es una versión antigua y actualmente solo recibe correcciones de errores críticos. Cuando crea un servicio de Kubernetes de este tipo LoadBalancer, el controlador de servicio crea un CLB de AWS de forma predeterminada, pero también puede crear un AWS NLB si utiliza la anotación correcta. Vale la pena señalar que Service Controller no admite los recursos de Kubernetes Ingress y tampoco los admite. IPv6

Recomendamos utilizar el controlador Load Balancer de AWS en los clústeres de EKS para conciliar los recursos de Kubernetes Service e Ingress. Debe usar las anotaciones correctas en el servicio de Kubernetes o en el manifiesto de Ingress para que AWS Load Balancer Controller sea el propietario del proceso de reconciliación. (en lugar de Service Controller)

Si utiliza el modo automático EKS, se le proporciona automáticamente el controlador Load Balancer de AWS; no es necesaria ninguna instalación.

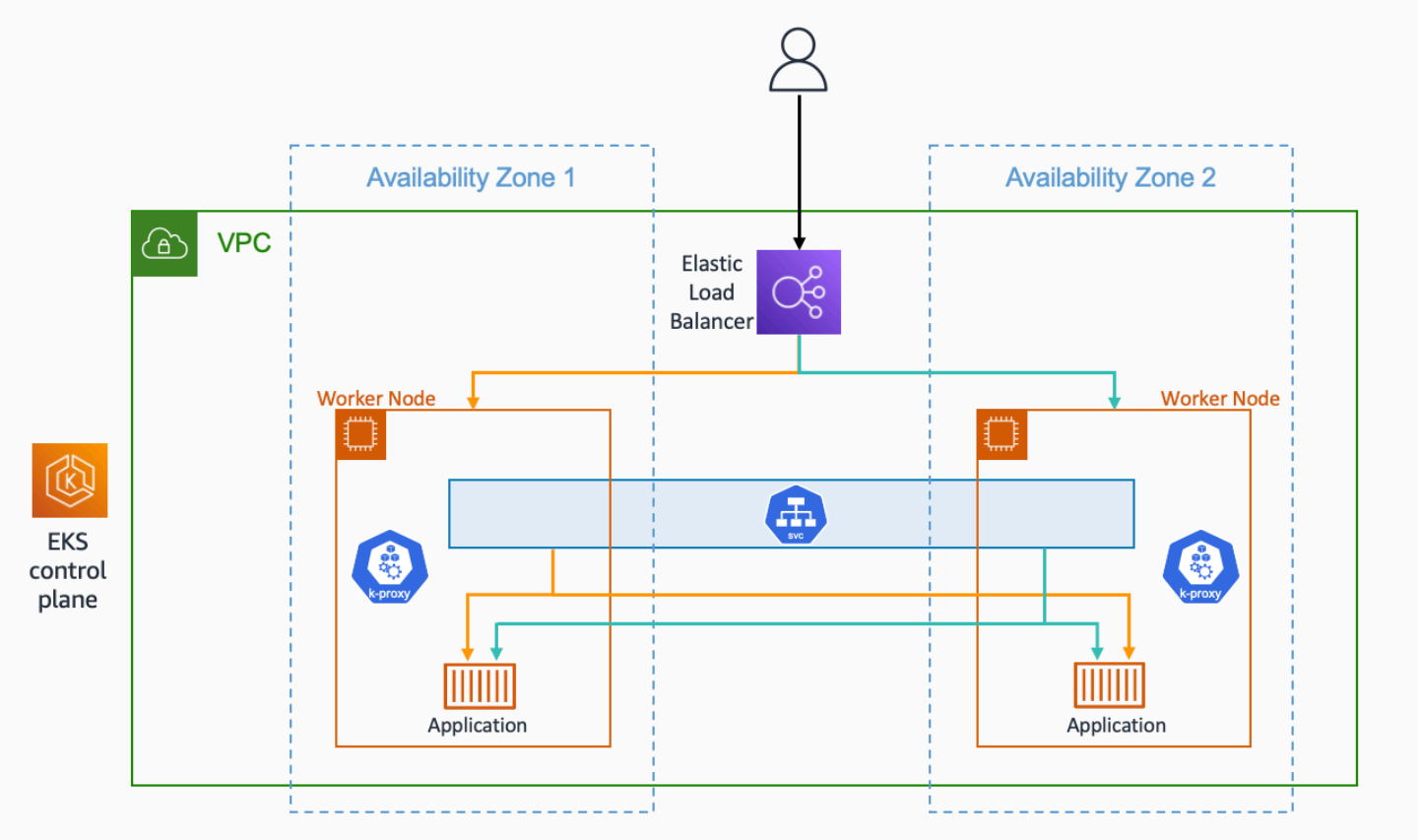
Elegir el tipo de destino del Load Balancer

Registre los pods como objetivos mediante IP Target-Type

Un AWS Elastic Load Balancer: Network & Application envía el tráfico recibido a los objetivos registrados de un grupo objetivo. Para un clúster EKS, hay dos tipos de objetivos que puedes registrar en el grupo de destino: instancia e IP. El tipo de objetivo que se utilice tiene implicaciones en lo que se registra y en la forma en que se enruta el tráfico desde el Load Balancer al pod. De forma predeterminada, el controlador Load Balancer de AWS registrará los objetivos mediante el tipo «Instancia» y este objetivo será la IP del nodo de trabajo yNodePort, entre sus implicaciones, se incluyen:

- El tráfico del Load Balancer se reenviará al nodo de trabajo del NodePort, se procesa mediante las reglas de iptables (configuradas por kube-proxy que se ejecuta en el nodo) y se reenvía al Servicio en su ClusterIP (aún en el nodo). Finalmente, el Servicio selecciona aleatoriamente un pod registrado en él y le reenvía el tráfico. Este flujo implica varios saltos y se puede incurrir en una latencia adicional, especialmente porque el Servicio a veces selecciona un pod que se ejecuta en otro nodo de trabajo que también podría estar en otra zona de disponibilidad.
- Como el Load Balancer registra el nodo de trabajo como su objetivo, esto significa que la comprobación de estado que se envíe al destino no la recibirá directamente el pod, sino el nodo de trabajo de su parte, NodePort y el tráfico de comprobación de estado seguirá la misma ruta descrita anteriormente.
- La supervisión y la solución de problemas son más complejas, ya que el tráfico reenviado por el Load Balancer no se envía directamente a los pods y tendrías que correlacionar cuidadosamente

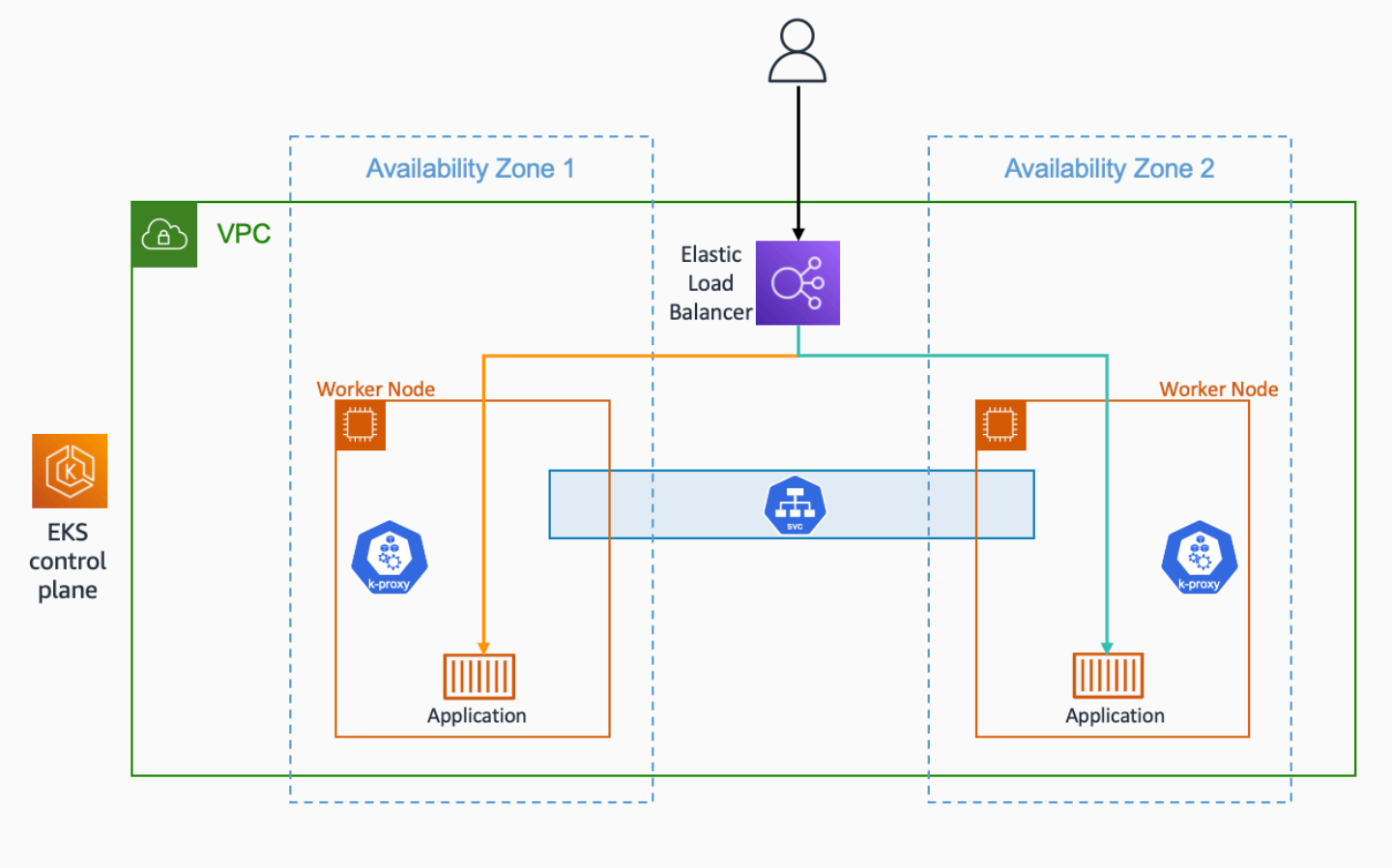
el paquete recibido en el nodo de trabajo con el ClusterIP de servicio y, finalmente, con el pod para tener una end-to-end visibilidad completa de la ruta del paquete y poder solucionar problemas adecuadamente.



Por el contrario, si configuras el tipo de destino como «IP», como recomendamos, la implicación será la siguiente:

- El tráfico del Load Balancer se reenviará directamente al pod, lo que simplifica la ruta de red, ya que evita los saltos adicionales anteriores de los nodos de trabajo y la IP del clúster de servicios, reduce la latencia en la que se habría incurrido si el servicio hubiera reenviado el tráfico a un pod en otra AZ y, por último, elimina la sobrecarga de procesamiento de las reglas de iptables en los nodos de trabajo.
- El módulo recibe y responde directamente a la comprobación de estado del balanceador de carga, lo que significa que el estado objetivo «saludable» o «insalubre» es una representación directa del estado de salud del módulo.

- La supervisión y la solución de problemas son más fáciles y cualquier herramienta que se utilice para capturar la dirección IP del paquete revelará directamente el tráfico bidireccional entre el Load Balancer y el pod en sus campos de origen y destino.



Para crear un AWS Elastic Load Balancing que utilice objetivos IP, añada:

- `alb.ingress.kubernetes.io/target-type: ip`anotación en tu manifiesto de Ingress al configurar tu Kubernetes Ingress (Application Load Balancer)
- `service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip`anotación en el manifiesto de su servicio al configurar el tipo de servicio de Kubernetes (Network LoadBalancer Load Balancer).

Configuración de las comprobaciones de estado de Load Balancer

Si bien Kubernetes proporciona sus propios mecanismos de control de estado (que se detallan en la siguiente sección), recomendamos implementar los controles de estado del ELB como una medida

de seguridad complementaria que funciona fuera del plano de control de Kubernetes. Esta capa independiente sigue supervisando la aplicación incluso durante:

- Interrupciones en el plano de control de Kubernetes
- Retrasos en la ejecución de
- Particiones de red entre kubelet y pod

Para las cargas de trabajo críticas que requieren la máxima disponibilidad y una recuperación acelerada en los escenarios mencionados anteriormente, los controles de estado del ELB proporcionan una red de seguridad esencial que funciona junto con los mecanismos nativos de Kubernetes, y no los reemplaza.

Para configurar y ajustar las comprobaciones de estado de su ELB, debe usar anotaciones en su servicio de Kubernetes o en el manifiesto de Ingress que el controlador de servicio o el controlador de AWS Load Balancer Controller conciliarían.

Disponibilidad y ciclo de vida del pod

Durante la actualización de una aplicación, debe asegurarse de que la aplicación esté siempre disponible para procesar las solicitudes, de modo que los usuarios no sufran ningún tiempo de inactividad. Un desafío habitual en este escenario es sincronizar el estado de disponibilidad de las cargas de trabajo entre la capa de Kubernetes y la infraestructura, por ejemplo, los balanceadores de carga externos. En las siguientes secciones, se destacan las mejores prácticas para abordar estos escenarios.

> **Note**
>
> Las siguientes explicaciones se basan en lo que se recomienda sustituir a los puntos de conexión en Kubernetes. EndpointSlices Las diferencias entre ambos son insignificantes en el contexto de los escenarios que se describen a continuación. El controlador Load Balancer de AWS consume puntos de enlace de forma predeterminada; puede activarlos EndpointSlices activándolos enable-endpoint-sliceflagen el controlador.

Utilice controles de estado

De forma predeterminada, Kubernetes ejecuta la comprobación del estado del proceso, en la que el proceso de kubelet del nodo verifica si el proceso principal del contenedor se está ejecutando

o no. Si no es así, de forma predeterminada, reinicia ese contenedor. Sin embargo, también puedes configurar las sondas de Kubernetes para identificar cuándo un proceso contenedor se está ejecutando pero se encuentra en un estado de bloqueo, o si una aplicación se ha iniciado correctamente o no. Las sondas se pueden basar en los mecanismos exec, grpc, HttpGet y TcpSocket. Según el tipo y el resultado de la sonda, se puede reiniciar el contenedor.

Consulta la sección sobre creación de pods en el apéndice que aparece a continuación para revisar la secuencia de eventos del proceso de creación de pods.

Usa sondas de preparación

De forma predeterminada, cuando todos los contenedores de un pod están en ejecución, el estado del pod se considera «Listo». Sin embargo, es posible que la aplicación aún no pueda procesar las solicitudes de los clientes. Por ejemplo, es posible que la aplicación necesite extraer algunos datos o configuraciones de un recurso externo para poder procesar las solicitudes. En tal estado, no querrá cerrar la aplicación ni reenviarle ninguna solicitud. La sonda de preparación te permite asegurarte de que el Pod no se considera «listo», lo que significa que no se añadirá al EndpointSlice objeto hasta que no se obtenga el resultado de la sonda`success`. Por otro lado, si la sonda falla más adelante en la línea, el Pod se retira del EndpointSlice objeto. Puedes configurar una sonda de preparación en el manifiesto del Pod para cada contenedor. `kubelet`Un proceso en cada nodo ejecuta la sonda de preparación en los contenedores de ese nodo.

Utilice las puertas de preparación de los Pod

Un aspecto de la sonda de preparación es el hecho de que no contiene ningún mecanismo de retroalimentación/influencia externo: un proceso de kubelet en el nodo ejecuta la sonda y define su estado. Esto no tiene ningún impacto en las solicitudes entre los microservicios en sí mismos en la capa de Kubernetes (tráfico este-oeste), ya que el EndpointSlice controlador mantiene la lista de puntos finales (pods) siempre actualizada. Entonces, ¿por qué y cuándo necesitarías un mecanismo externo?

Cuando expongas tus aplicaciones mediante el tipo de servicio de Kubernetes Load Balancer o Kubernetes Ingress (para el tráfico norte-sur), la lista de pods del servicio de Kubernetes correspondiente debe propagarse al balanceador de cargas de infraestructura externo IPs para que el balanceador de cargas también tenga una lista de objetivos actualizada. El controlador Load Balancer de AWS cierra esta brecha. Cuando utiliza el controlador de equilibrio de carga de AWS y lo aprovecha`target group: IP`, del mismo modo que el controlador de equilibrio de carga de `kube-proxy` AWS también recibe una actualización (`watch`mediante) y, a continuación, se comunica con la API del ELB para configurar y empezar a registrar la IP del pod como destino en el ELB.

Al realizar una actualización sucesiva de una implementación, se crean nuevos pods y, tan pronto como el estado de un nuevo pod es «Listo», se cierra el pod. old/existing Durante este proceso, el EndpointSlice objeto de Kubernetes se actualiza más rápido que el tiempo que tarda el ELB en registrar los nuevos pods como objetivos (véase registro de objetivos). Durante un breve período de tiempo, podría producirse una discordancia de estados entre la capa de Kubernetes y la capa de infraestructura, lo que podría provocar que se descartaran las solicitudes de los clientes. Durante este período, dentro de la capa de Kubernetes, los nuevos pods estarán listos para procesar las solicitudes, pero desde el punto de vista del ELB, no lo están.

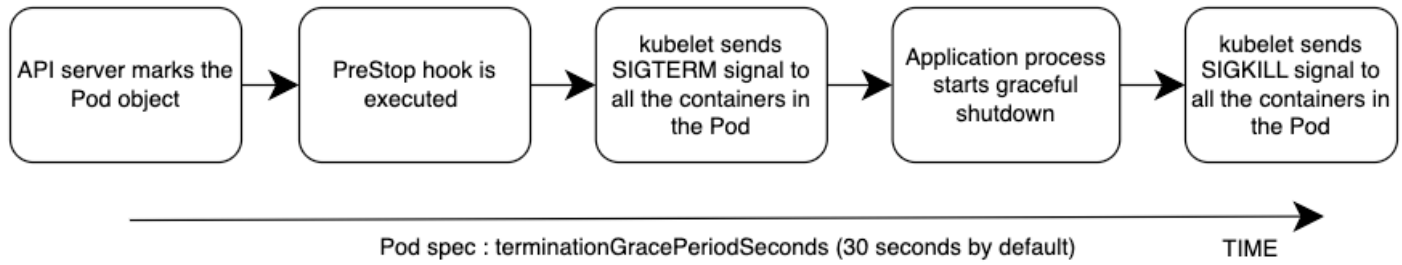
El Pod Readiness Gates te permite definir requisitos adicionales que deben cumplirse antes de que el estado del pod se considere «listo». En el caso de AWS ELB, el controlador de carga de AWS supervisa el estado del objetivo (el pod) en el ELB de AWS y, una vez que se completa el registro del objetivo y su estado pasa a ser «Correcto», el controlador actualiza el estado del pod a «Listo». Con este enfoque, usted influye en el estado del pod en función del estado de la red externa, que es el estado objetivo en el AWS ELB. Pod Readiness Gates es crucial en los escenarios de actualizaciones sucesivas, ya que le permite evitar que la actualización continua de una implementación termine con los pods antiguos hasta que el estado objetivo de los pods recién creados pase a ser «En buen estado» en el ELB de AWS.

Cierre las aplicaciones sin problemas

Su aplicación debería responder a una señal SIGTERM iniciando su cierre correcto para que los clientes no sufran ningún tiempo de inactividad. Esto significa que tu aplicación debe ejecutar procedimientos de limpieza, como guardar datos, cerrar los descriptores de los archivos, cerrar las conexiones a las bases de datos, completar correctamente las solicitudes en curso y salir de forma puntual para cumplir con la solicitud de cierre del Pod. Deberías establecer el período de gracia como para que la limpieza pueda finalizar. Para saber cómo responder a la señal SIGTERM, puede consultar los recursos del lenguaje de programación correspondiente que utilice para su aplicación.

Si su aplicación no puede cerrarse correctamente al recibir una señal SIGTERM o si ignora o no recibe la señal, puede aprovechar PreStopHook para iniciar un cierre correcto de la aplicación. El enlace Prestop se ejecuta inmediatamente antes de que se envíe la señal SIGTERM y puede realizar operaciones arbitrarias sin tener que implementarlas en el propio código de la aplicación.

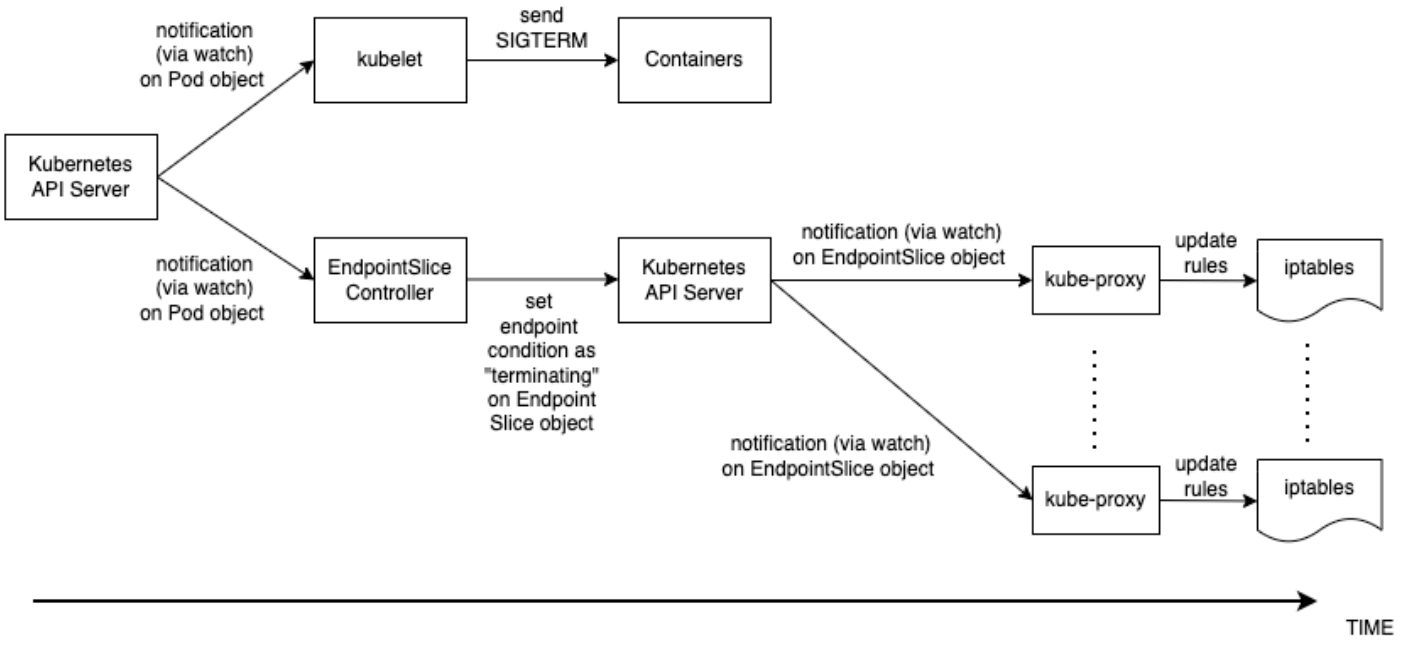
La secuencia general de eventos se muestra en el siguiente diagrama. Nota: independientemente del resultado de un proceso de cierre correcto de la aplicación o del resultado del bloqueo, los PreStop contenedores de la aplicación finalmente se cierran al final del período de gracia mediante SIGKILL.



Consulte la sección sobre la eliminación del pod en la sección del apéndice que aparece a continuación para revisar la secuencia de eventos del proceso de eliminación del pod.

Gestione con elegancia las solicitudes de los clientes

La secuencia de eventos en la eliminación del pod es diferente a la de la creación del pod. Cuando se crea un pod, se `kubelet` actualiza la IP del pod en la API de Kubernetes y solo entonces se actualiza el EndpointSlice objeto. Por otro lado, cuando se cierra un pod, la API de Kubernetes notifica al kubelet y al controlador al mismo tiempo. EndpointSlice Inspeccione cuidadosamente el siguiente diagrama, que muestra la secuencia de eventos.



La forma en que el estado se propaga desde el servidor API hasta las reglas de iptables en los nodos explicados anteriormente crea una condición de carrera interesante. Como existe una alta probabilidad de que el contenedor reciba la señal SIGKILL mucho antes que el kube-proxy de cada

nodo, se actualizan las reglas locales de iptables. En tal caso, dos escenarios que vale la pena mencionar son:

- Si su solicitud descarta de forma inmediata y sin rodeos las solicitudes de vuelo y las conexiones al recibir SIGTERM, significa que los clientes verían errores 50 veces multiplicados por todas partes.
- Aunque tu solicitud se asegure de que todas las solicitudes y conexiones en vuelo se procesen por completo al recibir SIGTERM, durante el período de gracia, las solicitudes de nuevos clientes seguirán enviándose al contenedor de aplicaciones, ya que es posible que las reglas de iptables aún no se hayan actualizado. Hasta que el procedimiento de limpieza cierre el socket del servidor del contenedor, esas nuevas solicitudes generarán nuevas conexiones. Cuando finaliza el período de gracia, esas conexiones, que se establecen después del SIGTERM, se cancelan incondicionalmente desde que se envía SIGKILL.

Definir el período de gracia en las especificaciones del Pod durante un tiempo suficiente puede solucionar este problema, pero en función del retraso de propagación y del número de solicitudes reales de los clientes, es difícil anticipar el tiempo que tardará la aplicación en cerrar las conexiones correctamente. Por lo tanto, el enfoque no tan perfecto pero más factible en este caso es utilizar un PreStop gancho para retrasar la señal SIGTERM hasta que se actualicen las reglas de iptables para garantizar que no se envíen nuevas solicitudes de clientes a la aplicación, sino que solo continúen las conexiones existentes. PreStop hook puede ser un simple controlador de Exec, como. `sleep 10`

El comportamiento y la recomendación mencionados anteriormente se aplicarían igualmente cuando exponga sus aplicaciones con el tipo de servicio de Kubernetes Load Balancer o Kubernetes Ingress (para el tráfico norte-sur) con AWS Load Balancer Controller y apalancamiento. `target group: IP` Porque, `kube-proxy` al igual que el controlador Load Balancer de AWS, también recibe una actualización (mediante un reloj) del EndpointSlice objeto y, a continuación, se comunica con la API del ELB para empezar a anular el registro de la IP del pod en el ELB. Sin embargo, dependiendo de la carga en la API de Kubernetes o la API de ELB, esto también puede llevar tiempo y es posible que el SIGTERM ya se haya enviado a la aplicación hace mucho tiempo. Una vez que el ELB comienza a anular el registro del destino, deja de enviar solicitudes a ese destino, por lo que la aplicación no recibe ninguna nueva solicitud y el ELB también inicia un retraso de anulación del registro, que es de 300 segundos de forma predeterminada. Durante el proceso de cancelación del registro, el objetivo es básicamente el `draining` lugar donde el ELB espera a que se agoten las conexiones en vuelo con ese objetivo. requests/existing Una vez transcurrido el plazo de cancelación del registro, el objetivo no se utiliza y cualquier solicitud en vuelo dirigida a ese objetivo se rechaza por la fuerza.

Usa el presupuesto de Pod Disruption

Configure un presupuesto de interrupción de cápsulas (PDB) para sus aplicaciones. PDBlimits el número de pods de una aplicación replicada que están inactivos simultáneamente debido a interrupciones voluntarias. Garantiza que haya un número o porcentaje mínimo de pods disponibles en una implementación StatefulSet . Por ejemplo, una aplicación basada en quórum debe garantizar que el número de réplicas en ejecución nunca sea inferior al número necesario para un quórum. O bien, una interfaz web podría garantizar que la cantidad de réplicas que se cargan nunca caiga por debajo de un porcentaje determinado del total. La PDB protegerá la aplicación contra acciones como el agotamiento de los nodos o el lanzamiento de nuevas versiones de Deployments. Tenga en cuenta que las PDB no protegerán la aplicación contra interrupciones involuntarias, como un fallo del sistema operativo del nodo o la pérdida de conectividad de la red. Para obtener más información, consulte la documentación sobre cómo especificar un presupuesto de interrupciones para su aplicación en Kubernetes.

Referencias

- KubeCon Sesión de Europa 2019: ¿Preparado? Una inmersión profunda en Pod Readiness Gates for Service Health
- Libro: Kubernetes en acción
- Blog de AWS: Cómo escalar rápidamente su aplicación con ALB en EKS (sin perder tráfico)

Apéndice

Creación de pods

Es imprescindible entender cuál es la secuencia de eventos en un escenario en el que se despliega un pod healthy/ready para luego recibir y procesar las solicitudes de los clientes. Hablemos de la secuencia de eventos.

1. Un pod se crea en el plano de control de Kubernetes (es decir, mediante un comando kubectl, una actualización de despliegue o una acción de escalado).
2. `kube-scheduler`asigna el pod a un nodo del clúster.
3. El proceso de kubelet que se ejecuta en el nodo asignado recibe la actualización (a través de`watch`) y se comunica con el motor de ejecución del contenedor para iniciar los contenedores definidos en la especificación del Pod.

4. Cuando los contenedores comienzan a ejecutarse, el kubelet actualiza la condición del Pod como `Ready` en el objeto Pod de la API de Kubernetes.
5. El EndpointSlicecontrolador recibe la actualización del estado del Pod (a través de`watch`) y añade el Pod IP/Port como un nuevo punto final al EndpointSliceobjeto (lista de Pod IPs) del servicio de Kubernetes correspondiente.
6. El proceso kube-proxy de cada nodo recibe la actualización (vía`watch`) del EndpointSlice objeto y, a continuación, actualiza las reglas de iptables en cada nodo, con la nueva IP/puerto del pod.

Eliminación del pod

Al igual que en la creación de un pod, es imprescindible entender cuál es la secuencia de eventos durante la eliminación del pod. Hablemos de la secuencia de eventos.

1. Se envía una solicitud de eliminación de un pod al servidor API de Kubernetes (es decir, mediante un `kubectl` comando, una actualización de implementación o una acción de escalado).
2. El servidor de la API de Kubernetes inicia un período de gracia, que es de 30 segundos de forma predeterminada, configurando el campo DeletionTimestamp en el objeto Pod. (El período de gracia se puede configurar hasta en las especificaciones del Pod) `terminationGracePeriodSeconds`
3. El `kubelet` proceso que se ejecuta en el nodo recibe la actualización (a través del reloj) del objeto Pod y envía una señal SIGTERM al identificador de proceso 1 (PID 1) que se encuentra dentro de cada contenedor de ese pod. A continuación, observa el`terminationGracePeriodSeconds`.
4. El EndpointSlicecontrolador también recibe la actualización (mediante`watch`) del paso 2 y establece la condición del punto final como «terminado» en el EndpointSliceobjeto (lista de pods IPs) del correspondiente servicio de Kubernetes.
5. El proceso kube-proxy de cada nodo recibe la actualización (vía`watch`) del EndpointSlice objeto y, a continuación, kube-proxy actualiza las reglas de iptables de cada nodo para dejar de reenviar las solicitudes de los clientes al pod.
6. Cuando `terminationGracePeriodSeconds` expira, `kubelet` envía una señal SIGKILL al proceso principal de cada contenedor del pod y lo cierra por la fuerza.
7. TheEndpointSliceEl controlador elimina el punto final del objeto. EndpointSlice
8. El servidor API elimina el objeto Pod.

Supervisión de las cargas de trabajo de EKS para detectar problemas de rendimiento de la red

Supervisión del tráfico de CoredNS para detectar problemas de limitación de DNS

La ejecución de cargas de trabajo con un uso intensivo de DNS a veces puede provocar fallos intermitentes de CoreDNS debido a la limitación del DNS, lo que puede afectar a las aplicaciones, en las que se pueden producir errores ocasionales. UnknownHostException

La implementación de CoreDNS tiene una política de antiafinidad que indica al programador de Kubernetes que ejecute instancias de CoreDNS en nodos de trabajo independientes del clúster, es decir, debe evitar la ubicación conjunta de réplicas en el mismo nodo de trabajo. Esto reduce de forma efectiva el número de consultas de DNS por interfaz de red, ya que el tráfico de cada réplica se enruta a través de un ENI diferente. Si observa que las consultas de DNS se están limitando debido al límite de 1024 paquetes por segundo, puede 1) intentar aumentar el número de réplicas de CoredNS o 2) implementarlas. NodeLocal DNSCache Consulte Supervisar las métricas de CoredNS para obtener más información.

Challenge

- La caída de paquetes se produce en cuestión de segundos y puede resultar complicado para nosotros monitorear adecuadamente estos patrones para determinar si realmente se está produciendo una limitación del DNS.
- Las consultas de DNS se limitan en el nivel de la interface de red elástica. Por lo tanto, las consultas restringidas no aparecen en el registro de consultas.
- Los logs de flujo no capturan todo el tráfico IP. Por ejemplo: Tráfico generado por instancias al contactar con el servidor DNS de Amazon. Si utilizas tu propio servidor DNS, se registrará todo el tráfico dirigido a ese servidor DNS

Solución

Una forma sencilla de identificar los problemas de limitación del DNS en los nodos de trabajo consiste en capturar `linklocal_allowance_exceeded` las métricas. El valor linklocal_allowance_exceeded es el número de paquetes descartados debido a que el PPS del tráfico dirigido a los servicios proxy locales superó el máximo de la interfaz de red. Esto afecta al tráfico

hacia el servicio de DNS, el servicio de metadatos de instancia y el Amazon Time Sync Service. En lugar de realizar un seguimiento de este evento en tiempo real, también podemos transmitir esta métrica a Amazon Managed Service for Prometheus y visualizarla en Amazon Managed Grafana

Supervisión de los retrasos en las consultas de DNS mediante las métricas de Conntrack

Otra métrica que puede ayudar a monitorear la limitación o el retraso de las consultas de CoredNS es y. `conntrack_allowance_available` `conntrack_allowance_exceeded` Los fallos de conectividad causados por superar los límites permitidos por Connections Tracked pueden tener un impacto mayor que los que se producen al superar otros permisos. Cuando se depende del TCP para transferir datos, los paquetes que se ponen en cola o se descartan debido a que se superan los límites de la red de la EC2 instancia, como el ancho de banda, el PPS, etc., suelen gestionarse sin problemas gracias a las capacidades de control de la congestión del TCP. Los flujos afectados se ralentizarán y los paquetes perdidos se retransmitirán. Sin embargo, cuando una instancia supera su límite de conexiones rastreadas, no se pueden establecer nuevas conexiones hasta que se cierren algunas de las existentes para dejar espacio a nuevas conexiones.

`conntrack_allowance_availabley` `conntrack_allowance_exceeded` ayuda a los clientes a controlar el límite de conexiones rastreadas, que varía en cada caso. Estas métricas de rendimiento de la red ofrecen a los clientes visibilidad de la cantidad de paquetes en cola o descartados cuando se superan las asignaciones de red de una instancia, como el ancho de banda de la red Packets-Per-Second (PPS), las conexiones rastreadas y el acceso a los servicios de enlace local (Amazon DNS, Instance Metadata Service, Amazon Time Sync).

`conntrack_allowance_availablees` el número de conexiones rastreadas que puede establecer la instancia antes de alcanzar el límite de conexiones rastreadas de ese tipo de instancia (solo se admite en instancias basadas en Nitro). `conntrack_allowance_exceededes` la cantidad de paquetes descartados porque el seguimiento de conexiones superó el máximo de la instancia y no se pudieron establecer nuevas conexiones.

Otras métricas importantes de rendimiento de la red

Otras métricas importantes del rendimiento de la red incluyen:

`bw_in_allowance_exceeded`(el valor ideal de la métrica debe ser cero) es la cantidad de paquetes en cola and/or descartados porque el ancho de banda agregado entrante superó el máximo de la instancia

`bw_out_allowance_exceeded`(el valor ideal de la métrica debe ser cero) es la cantidad de paquetes en cola and/or descartados porque el ancho de banda agregado saliente superó el máximo de la instancia

`pps_allowance_exceeded`(el valor ideal de la métrica debe ser cero) es la cantidad de paquetes en cola and/or descartados porque el PPS bidireccional superó el máximo de la instancia

Capturar las métricas para monitorear las cargas de trabajo en busca de problemas de rendimiento de la red

El controlador Elastic Network Adapter (ENA) publica las métricas de rendimiento de la red descritas anteriormente en las instancias en las que están habilitadas. Todas las métricas de rendimiento de la red se pueden publicar CloudWatch mediante el CloudWatch agente. Consulte el blog para obtener más información.

Ahora capturemos las métricas comentadas anteriormente, las almacenaremos en Amazon Managed Service for Prometheus y las visualicemos con Amazon Managed Grafana

Requisitos previos

- ethtool: asegúrese de que los nodos de trabajo tengan instalado ethtool
- Un espacio de trabajo de AMP configurado en su cuenta de AWS. Para obtener instrucciones, consulte Crear un espacio de trabajo en la Guía del usuario de AMP.
- Espacio de trabajo de Grafana gestionado por Amazon

Implementación del Prometheus Node Exporter

Para recopilar las métricas de ethtool para los nodos trabajadores, necesitamos implementar un recopilador que recopile las métricas y las exponga en formato Prometheus. Los siguientes comandos añadirán el repositorio de `prometheus-community` Helm e instalarán el `prometheus-node-exporter` gráfico con el recopilador activado. `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'"
```

Implemente el recopilador ADOT para recopilar las métricas de ethtool y almacenarlas en el espacio de trabajo de Amazon Managed Service for Prometheus

Cada clúster en el que instale AWS Distro for OpenTelemetry (ADOT) debe tener esta función para conceder a su cuenta de servicio de AWS permisos para almacenar métricas en Amazon Managed Service for Prometheus. Siga estos pasos para crear y asociar su rol de IAM a su cuenta de servicio de Amazon EKS mediante IRSA:

```
eksctl create iamserviceaccount --name adot-collector --namespace default
 --cluster <CLUSTER_NAME> --attach-policy-arn arn:aws:iam::aws:policy/
AmazonPrometheusRemoteWriteAccess --attach-policy-arn arn:aws:iam::aws:policy/
AWSXrayWriteOnlyAccess --attach-policy-arn arn:aws:iam::aws:policy/
CloudWatchAgentServerPolicy --region <REGION> --approve  --override-existing-
serviceaccounts
```

Implementemos el recopilador ADOT para extraer las métricas del exportador de ethtool de Prometheus y almacenarlas en Amazon Managed Service para Prometheus.

En el siguiente procedimiento, se utiliza un ejemplo de archivo YAML con la implementación como valor de modo. Este es el modo predeterminado e implementa ADOT Collector de forma similar a una aplicación independiente. Esta configuración recibe métricas de OTLP de la aplicación de muestra y métricas de Amazon Managed Service for Prometheus extraídas de los pods del clúster.

```
curl -o collector-config-amp.yaml https://raw.githubusercontent.com/aws-observability/
aws-otel-community/master/sample-configs/operator/collector-config-amp.yaml
```

En collector-config-amp .yaml, sustituye lo siguiente por tus propios valores:

- modo: despliegue
- Cuenta de servicio: adot-collector
- punto final: <YOUR_REMOTE_WRITE_ENDPOINT>
- región: <YOUR_ >AWS_REGION
- nombre: adot-collector

```
kubectl apply -f collector-config-amp.yaml
```

Una vez implementado el recopilador de adot, las métricas se almacenarán correctamente en Amazon Prometheus.

Configurar el administrador de alertas en Amazon Managed Service para que Prometheus envíe notificaciones

Puedes usar el administrador de alertas de Amazon Managed Service for Prometheus para configurar reglas de alerta para las alertas críticas y, a continuación, enviar notificaciones a un tema de Amazon SNS. Vamos a configurar las reglas de registro y las reglas de alerta para comprobar las métricas comentadas hasta ahora.

Utilizaremos el controlador ACK para Amazon Managed Service for Prometheus para proporcionar las reglas de alerta y grabación.

Implementemos el controlador ACL para el servicio Amazon Managed Service for Prometheus:

```
export SERVICE=prometheusservice
export RELEASE_VERSION=`curl -sL https://api.github.com/repos/aws-controllers-k8s/
$SERVICE-controller/releases/latest | grep '"tag_name":' | cut -d'"' -f4`
export ACK_SYSTEM_NAMESPACE=ack-system
export AWS_REGION=us-east-1
aws ecr-public get-login-password --region us-east-1 | helm registry login --username
 AWS --password-stdin public.ecr.aws
helm install --create-namespace -n $ACK_SYSTEM_NAMESPACE ack-$SERVICE-controller \
oci://public.ecr.aws/aws-controllers-k8s/$SERVICE-chart --version=$RELEASE_VERSION --
set=aws.region=$AWS_REGION
```

Ejecute el comando y, después de unos momentos, debería ver el siguiente mensaje:

```
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.
```

Vamos a crear ahora un archivo yaml para aprovisionar los grupos de reglas y definiciones del administrador de alertas. Guarde el siguiente archivo como `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"
```

Reemplace su ID de espacio de trabajo por el ID de espacio de trabajo del espacio de trabajo que está utilizando.

Vamos a configurar ahora la definición del administrador de alertas. Guarde el siguiente archivo como `alertmanager.yaml`

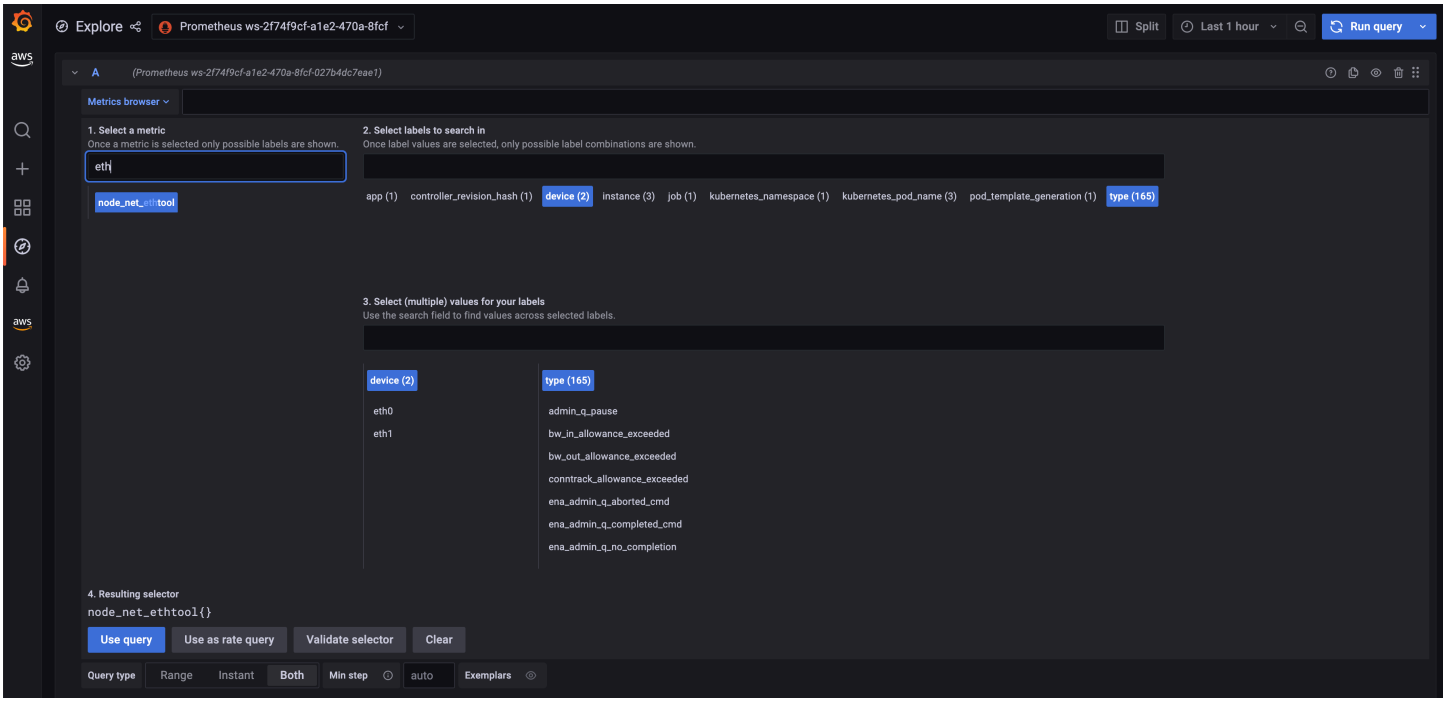
```
apiVersion: prometheusservice.services.k8s.aws/v1alpha1
kind: AlertManagerDefinition
metadata:
  name: alert-manager
spec:
  workspaceID: <Your WORKSPACE-ID >
  configuration: |
    alertmanager_config: |
      route:
         receiver: default_receiver
       receivers:
       - name: default_receiver
          sns_configs:
          - topic_arn: TOPIC-ARN
            sigv4:
              region: REGION
            message: |
              alert_type: {{ .CommonLabels.alertname }}
              event_type: {{ .CommonLabels.event_type }}
```

Sustituya su WORKSPACE-ID por el ID de espacio de trabajo del nuevo espacio de trabajo, TOPIC-ARN por el ARN de un tema de Amazon Simple Notification Service al que desee enviar las alertas y REGION por la región actual de la carga de trabajo. Asegúrese de que su espacio de trabajo tenga permisos para enviar mensajes a Amazon SNS.

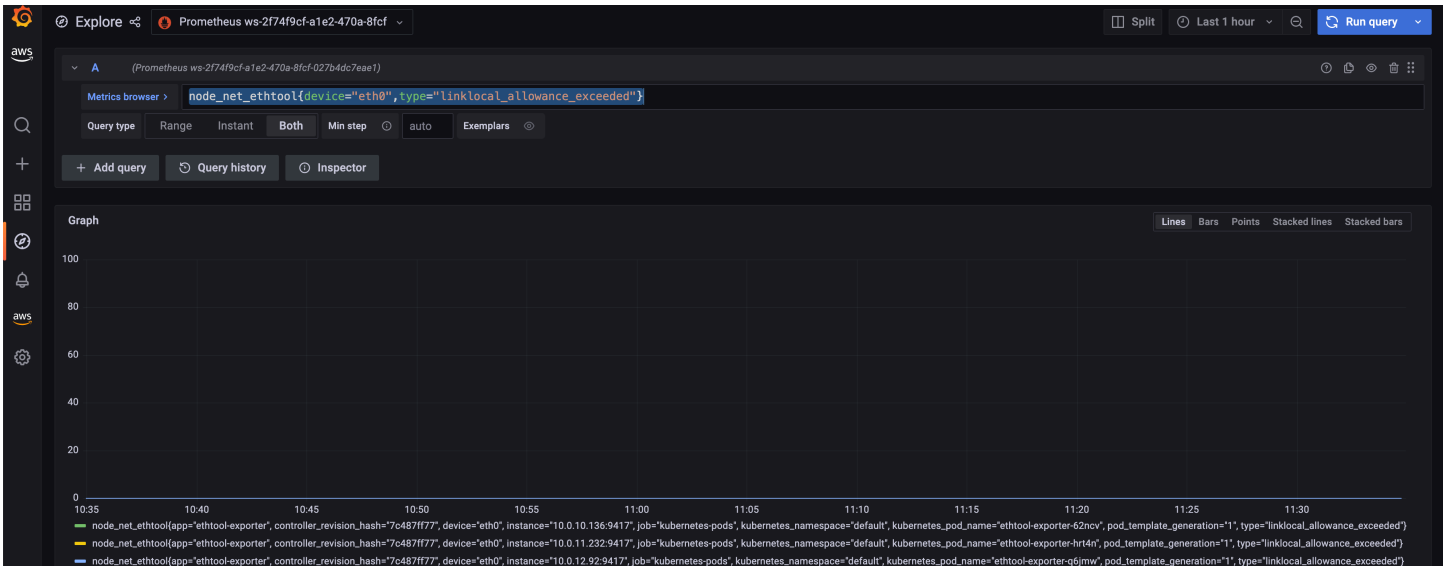
Visualice las métricas de ethtool en Amazon Managed Grafana

Visualicemos las métricas de la Grafana gestionada por Amazon y creemos un panel de control. Configure el Amazon Managed Service para Prometheus como fuente de datos dentro de la consola de Amazon Managed Grafana. Para obtener instrucciones, consulte Añadir Amazon Prometheus como fuente de datos

Analicemos ahora las métricas de Amazon Managed Grafana: haz clic en el botón de exploración y busca ethtool:



Vamos a crear un panel para la métrica linklocal_allowance_exceeded mediante la consulta. `rate(node_net_ethtool{device="eth0",type="linklocal_allowance_exceeded"}[30s])` El resultado será el siguiente panel.

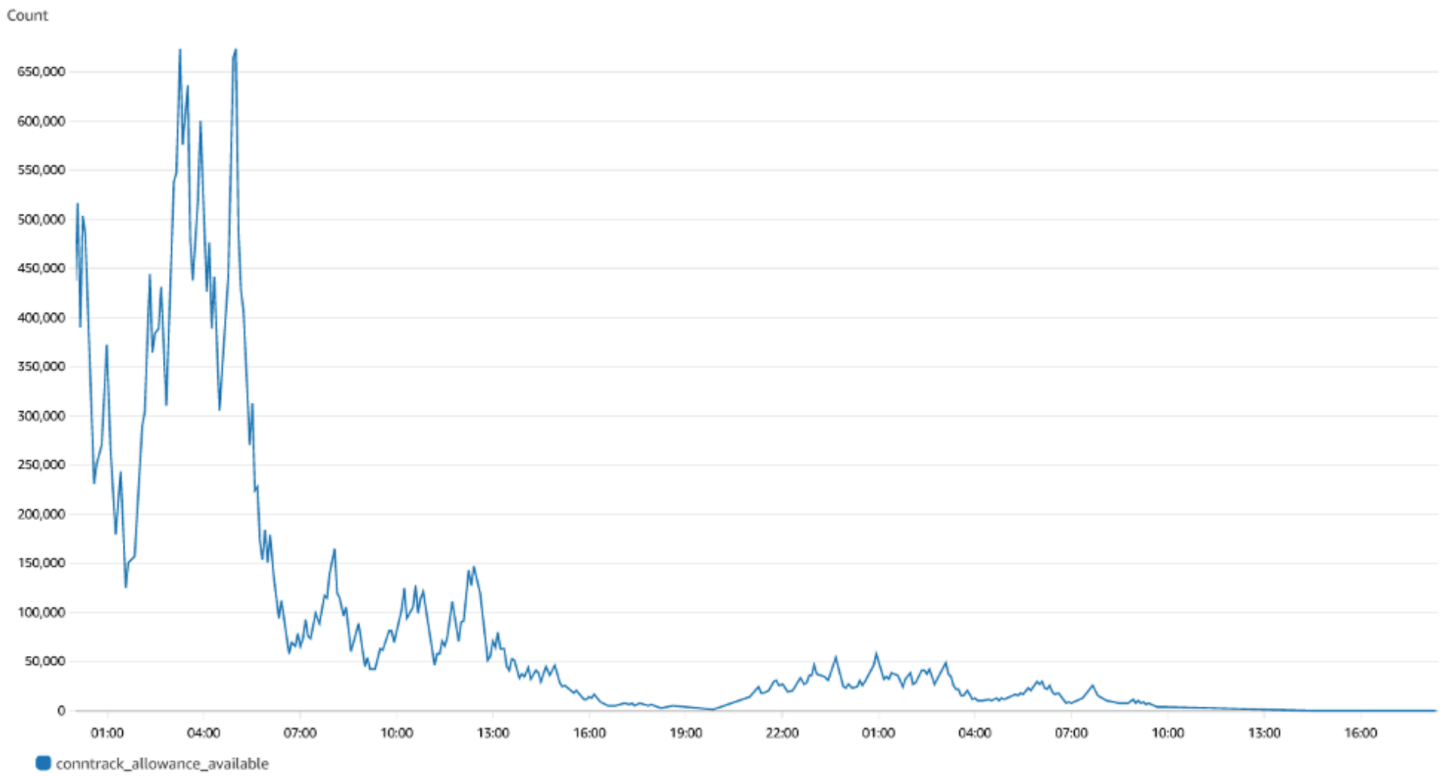


Podemos ver claramente que no se descartó ningún paquete ya que el valor es cero.

Vamos a crear un panel para la métrica conntrack_allowance_exceeded mediante la consulta. `rate(node_net_ethtool{device="eth0",type="conntrack_allowance_exceeded"}[30s])` El resultado será el siguiente panel.



La métrica se `conntrack_allowance_exceeded` puede visualizar en CloudWatch, siempre que ejecute un agente de Cloudwatch como se describe aquí. El panel resultante CloudWatch tendrá el siguiente aspecto:



Podemos ver claramente que no se descartó ningún paquete ya que el valor es cero. Si utilizas instancias basadas en Nitro, puedes crear un panel similar para `conntrack_allowance_available` monitorizar de forma proactiva las conexiones de tu EC2

instancia. Puedes ampliarlo aún más configurando alertas en Amazon Managed Grafana para enviar notificaciones a Slack, SNS, Pagerduty, etc.

Ejecutando kube-proxy en modo IPVS

EKS en el modo de servidor virtual IP (IPVS) resuelve el problema de latencia de la red que suele producirse cuando se ejecutan clústeres grandes con más de 1000 servicios con kube-proxy ejecutándose en el modo iptables antiguo. Este problema de rendimiento es el resultado del procesamiento secuencial de las reglas de filtrado de paquetes de iptables para cada paquete. Este problema de latencia se solucionó en nftables, el sucesor de iptables. Sin embargo, en el momento de escribir este artículo, kube-proxy aún estaba en desarrollo para utilizar nftables. Para solucionar este problema, puedes configurar tu clúster para que se ejecute en modo IPVS. `kube-proxy`

Descripción general de

IPVS, que ha estado en formato GA desde la versión 1.11 de Kubernetes, utiliza tablas hash en lugar de búsquedas lineales para procesar los paquetes, lo que proporciona eficiencia a los clústeres con miles de nodos y servicios. IPVS se diseñó para equilibrar la carga, lo que lo convierte en una solución adecuada para los problemas de rendimiento de las redes de Kubernetes.

IPVS ofrece varias opciones para distribuir el tráfico a los pods de back-end. Puede encontrar información detallada sobre cada opción en la documentación oficial de Kubernetes, pero a continuación se muestra una lista sencilla. Round Robin y Least Connections se encuentran entre las opciones de equilibrio de carga de IPVS más populares en 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)
```

Implementación

Solo se requieren unos pocos pasos para habilitar IPVS en su clúster de EKS. Lo primero que debe hacer es asegurarse de que las imágenes del nodo de trabajo de EKS tengan instalado el `ipvsadm`

paquete de administración del servidor virtual de Linux. Para instalar este paquete en una imagen basada en Fedora, como Amazon Linux 2023, puede ejecutar el siguiente comando en la instancia del nodo trabajador.

```
sudo dnf install -y ipvsadm
```

En una imagen basada en Debian, como Ubuntu, el comando de instalación tendría este aspecto.

```
sudo apt-get install ipvsadm
```

A continuación, debe cargar los módulos del núcleo para las opciones de configuración de IPVS enumeradas anteriormente. Recomendamos escribir estos módulos en un archivo dentro del `/etc/modules-load.d/` directorio para que sobrevivan a un reinicio.

```
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'
```

Puede ejecutar el siguiente comando para cargar estos módulos en una máquina que ya esté funcionando.

```
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**
>
> Se recomienda encarecidamente ejecutar estos pasos del nodo de trabajo como parte del proceso de arranque del nodo de trabajo mediante un script de datos de usuario o en cualquier script de compilación que se ejecute para crear una AMI de nodo de trabajo personalizada.

A continuación, configurará los clústeres para que se `kube-proxy` DaemonSet ejecuten en modo IPVS. Esto se hace configurando una de `kube-proxy` `mode` `ipvs` las `ipvs scheduler` opciones de equilibrio de carga enumeradas anteriormente, por ejemplo: `rr` para Round Robin.

> **Warning**
>
> Se trata de un cambio disruptivo y debe realizarse fuera del horario laboral. Recomendamos realizar estos cambios durante la creación inicial del clúster de EKS para minimizar los impactos.

Puede emitir un comando de la AWS CLI para habilitar IPVS actualizando el complemento `kube-proxy` EKS.

```
aws eks update-addon --cluster-name $CLUSTER_NAME --addon-name kube-proxy \
  --configuration-values '{"ipvs": {"scheduler": "rr"}, "mode": "ipvs"}' \
  --resolve-conflicts OVERWRITE
```

O puede hacerlo modificando el `kube-proxy-config` ConfigMap de su clúster.

```
kubectl -n kube-system edit cm kube-proxy-config
```

Busque la `scheduler` configuración en `ipvs` y establezca el valor en una de las opciones de equilibrio de carga de ipvs enumeradas anteriormente, por ejemplo: `rr` para Round Robin. Busca la `mode` configuración, cuyo valor predeterminado es`iptables`, y cambia el valor a. `ipvs` El resultado de cualquiera de las opciones debería ser similar al de la siguiente configuración.

```
  iptables:
    masqueradeAll: false
    masqueradeBit: 14
    minSyncPeriod: 0s
    syncPeriod: 30s
  ipvs:
    excludeCIDRs: null
    minSyncPeriod: 0s
    scheduler: "rr"
    syncPeriod: 30s
  kind: KubeProxyConfiguration
  metricsBindAddress: 0.0.0.0:10249
  mode: "ipvs"
  nodePortAddresses: null
  oomScoreAdj: -998
  portRange: ""
  udpIdleTimeout: 250ms
```

Si los nodos de trabajo se unieron al clúster antes de realizar estos cambios, tendrás que reiniciar el DaemonSet kube-proxy.

```
kubectl -n kube-system rollout restart ds kube-proxy
```

Validación

Para validar que el clúster y los nodos de trabajo se estén ejecutando en modo IPVS, ejecute el siguiente comando en uno de sus nodos de trabajo.

```
sudo ipvsadm -L
```

Como mínimo, deberías ver un resultado similar al siguiente, que muestre las entradas del servicio de servidor de API de Kubernetes en y `10.100.0.1` el servicio de CoredNS en. `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**
>
> El resultado de este ejemplo proviene de un clúster EKS con un rango de direcciones IP del servicio de. `10.100.0.0/16`

Mejores prácticas de escalabilidad de EKS

Esta guía proporciona consejos para escalar los clústeres de EKS. El objetivo de escalar un clúster de EKS es maximizar la cantidad de trabajo que puede realizar un solo clúster. El uso de un único clúster de EKS de gran tamaño puede reducir la carga operativa en comparación con el uso de varios clústeres, pero tiene ventajas y desventajas en relación con aspectos como las implementaciones en varias regiones, el aislamiento de los inquilinos y las actualizaciones de los clústeres. En este documento, nos centraremos en cómo lograr la máxima escalabilidad con un solo clúster.

Cómo usar esta guía

Esta guía está destinada a los desarrolladores y administradores responsables de crear y administrar clústeres de EKS en AWS. Se centra en algunas prácticas genéricas de escalado de Kubernetes, pero no tiene especificaciones específicas para los clústeres de Kubernetes autogestionados o los clústeres que se ejecutan fuera de una región de AWS con EKS Anywhere.

Cada tema incluye una breve descripción general, seguida de recomendaciones y prácticas recomendadas para operar los clústeres de EKS a escala. No es necesario leer los temas en un orden determinado y las recomendaciones no se deben aplicar sin probar y verificar su funcionamiento en los clústeres.

Comprender las dimensiones de escalado

La escalabilidad es diferente del rendimiento y la fiabilidad, y estos tres aspectos deben tenerse en cuenta a la hora de planificar las necesidades de clúster y carga de trabajo. A medida que los clústeres se escalan, es necesario monitorearlos, pero esta guía no abordará las mejores prácticas de monitoreo. EKS puede ampliarse a tamaños grandes, pero tendrás que planificar cómo vas a escalar un clúster a más de 300 nodos o 5000 pods. No se trata de cifras absolutas, pero provienen de la colaboración en esta guía con varios usuarios, ingenieros y profesionales de soporte.

El escalado en Kubernetes es multidimensional y no hay ajustes ni recomendaciones específicos que funcionen en todas las situaciones. Las áreas principales en las que podemos proporcionar orientación para el escalado incluyen:

El plano de control de Kubernetes en un clúster de EKS incluye todos los servicios que AWS ejecuta y escala automáticamente (por ejemplo, el servidor API de Kubernetes). La ampliación del plano de

control es responsabilidad de AWS, pero su uso responsable del plano de control es responsabilidad suya.

El escalado del plano de datos de Kubernetes se ocupa de los recursos de AWS necesarios para el clúster y las cargas de trabajo, pero están fuera del plano de control de EKS. Los recursos, incluidas EC2 las instancias, el kubelet y el almacenamiento, deben escalarse a medida que crece el clúster.

Los servicios de clúster son controladores y aplicaciones de Kubernetes que se ejecutan dentro del clúster y proporcionan funcionalidad al clúster y a las cargas de trabajo. Pueden ser complementos de EKS y también otros servicios o gráficos de Helm que instales para garantizar la conformidad y las integraciones. Las cargas de trabajo suelen depender de estos servicios y, a medida que las cargas de trabajo se amplíen, los servicios del clúster deberán adaptarse a ellos.

Las cargas de trabajo son la razón por la que tiene un clúster y deben escalarse horizontalmente con el clúster. Hay integraciones y configuraciones que las cargas de trabajo tienen en Kubernetes que pueden ayudar a que el clúster se amplíe. Las abstracciones de Kubernetes también tienen en cuenta aspectos arquitectónicos, como los espacios de nombres y los servicios.

Escalado extra grande

Si vas a escalar un solo clúster a más de 1000 nodos o 50 000 pods, estaremos encantados de hablar contigo. Te recomendamos que te pongas en contacto con tu equipo de soporte o con tu gestor técnico de cuentas para ponerte en contacto con especialistas que puedan ayudarte a planificar y ampliar tu negocio más allá de la información proporcionada en esta guía. Amazon EKS puede admitir hasta 100 000 nodos en un solo clúster si lo selecciona para la incorporación.

Plano de control de Kubernetes

El plano de control de Kubernetes consta del servidor API de Kubernetes, el administrador del controlador de Kubernetes, el programador y otros componentes necesarios para que Kubernetes funcione. Los límites de escalabilidad de estos componentes varían según lo que se ejecute en el clúster, pero las áreas que tienen un mayor impacto en la escalabilidad incluyen la versión de Kubernetes, la utilización y el escalado de nodos individuales.

Limite la carga de trabajo y la sobrecarga de nodos

> **Important**
>
> Para evitar alcanzar los límites de las API en el plano de control, debe limitar los picos de escalado que aumentan el tamaño del clúster en porcentajes de dos dígitos a la vez (por ejemplo, de 1000 nodos a 1100 nodos o de 4000 a 4500 pods a la vez).

El plano de control de EKS se ampliará automáticamente a medida que el clúster crezca, pero hay límites en cuanto a la rapidez con la que se escalará. Al crear un clúster de EKS por primera vez, el plano de control no podrá ampliarse inmediatamente a cientos de nodos o miles de módulos. Para obtener más información sobre cómo EKS ha realizado mejoras de escalado, consulte esta entrada de blog.

El escalado de aplicaciones de gran tamaño requiere que la infraestructura se adapte y esté completamente preparada (por ejemplo, calentar los balanceadores de carga). Para controlar la velocidad de escalado, asegúrese de escalarlo en función de las métricas correctas para su aplicación. Es posible que el escalado de la CPU y la memoria no prediga con precisión las limitaciones de la aplicación, por lo que utilizar métricas personalizadas (por ejemplo, las solicitudes por segundo) en el escalador automático de módulos horizontales (HPA) de Kubernetes puede ser una mejor opción de escalado.

Para usar una métrica personalizada, consulta los ejemplos de la documentación de Kubernetes. Si tiene necesidades de escalado más avanzadas o necesita escalar en función de fuentes externas (por ejemplo, AWS SQS Queue), utilice KEDA para el escalado de carga de trabajo basado en eventos.

Amplíe los nodos y los módulos de forma segura

Sustituya las instancias de ejecución prolongada

La sustitución periódica de los nodos mantiene el clúster en buen estado, ya que evita desviaciones en la configuración y problemas que solo se producen después de un tiempo de actividad prolongado (por ejemplo, pérdidas lentas de memoria). El reemplazo automatizado le proporcionará buenos procesos y prácticas para actualizar los nodos y aplicar parches de seguridad. Si todos los nodos del clúster se sustituyen con regularidad, se necesitará menos esfuerzo para mantener procesos separados para el mantenimiento continuo.

Usa la configuración de tiempo de vida (TTL) de Karpenter para reemplazar las instancias después de que hayan estado ejecutándose durante un período de tiempo específico. Los grupos de nodos autogestionados pueden usar esta `max-instance-lifetime` configuración para realizar ciclos de nodos automáticamente. Los grupos de nodos gestionados no disponen actualmente de esta función, pero puedes realizar un seguimiento de la solicitud aquí GitHub.

Elimine los nodos infrautilizados

Puede eliminar los nodos cuando no tengan cargas de trabajo en ejecución utilizando el umbral de reducción del escalador automático de clústeres de Kubernetes con la configuración del aprovisionador `--scale-down-utilization-threshold`o, en Karpenter, puede utilizar la configuración del aprovisionador. `ttlSecondsAfterEmpty`

Utilice los presupuestos para la interrupción de los módulos y el cierre seguro de los nodos

Para eliminar los pods y los nodos de un clúster de Kubernetes, es necesario que los controladores realicen actualizaciones en varios recursos (por ejemplo). EndpointSlices Hacerlo con frecuencia o demasiado rápido puede provocar una ralentización del servidor de API e interrupciones en las aplicaciones a medida que los cambios se propaguen a los controladores. Los presupuestos relacionados con la disrupción de los módulos son una buena práctica para reducir la rotación y proteger la disponibilidad de la carga de trabajo a medida que los nodos se eliminan o se reprograman en un clúster.

Usa la caché del lado del cliente cuando ejecutes Kubectl

El uso ineficiente del comando kubectl puede añadir carga adicional al servidor API de Kubernetes. Debes evitar ejecutar scripts o automatizaciones que usen kubectl de forma repetida (por ejemplo, en un bucle for) o ejecutar comandos sin una caché local.

`kubectl`tiene una caché del lado del cliente que almacena en caché la información de descubrimiento del clúster para reducir la cantidad de llamadas a la API necesarias. La caché está habilitada de forma predeterminada y se actualiza cada 10 minutos.

Si ejecutas kubectl desde un contenedor o sin una caché del lado del cliente, es posible que tengas problemas de limitación de la API. Se recomienda conservar la caché del clúster montando el para evitar realizar llamadas innecesarias a la `--cache-dir` API.

Inhabilita la compresión kubectl

Si deshabilitas la compresión kubectl en tu archivo kubeconfig, puedes reducir el uso de la API y la CPU del cliente. De forma predeterminada, el servidor comprimirá los datos enviados al cliente para optimizar el ancho de banda de la red. Esto añade una carga de CPU al cliente y al servidor por cada solicitud y, si se dispone de un ancho de banda adecuado, se pueden reducir la sobrecarga y la latencia si se dispone de un ancho de banda adecuado. Para deshabilitar la compresión, puedes usar el `--disable-compression=true` indicador o configurarlo `disable-compression: true` en tu archivo kubeconfig.

```
apiVersion: v1
clusters:
- cluster:
    server: serverURL
    disable-compression: true
  name: cluster
```

Escalador automático de clústeres compartidos

El escalador automático de clústeres de Kubernetes se ha probado para escalar hasta 1000 nodos. En un clúster grande con más de 1000 nodos, se recomienda ejecutar varias instancias del escalador automático de clústeres en modo fragmentado. Cada instancia de Cluster AutoScaler está configurada para escalar un conjunto de grupos de nodos. El siguiente ejemplo muestra 2 configuraciones de escalado automático de clústeres configuradas para escalar 4 grupos de nodos cada una.

ClusterAutoscaler-1

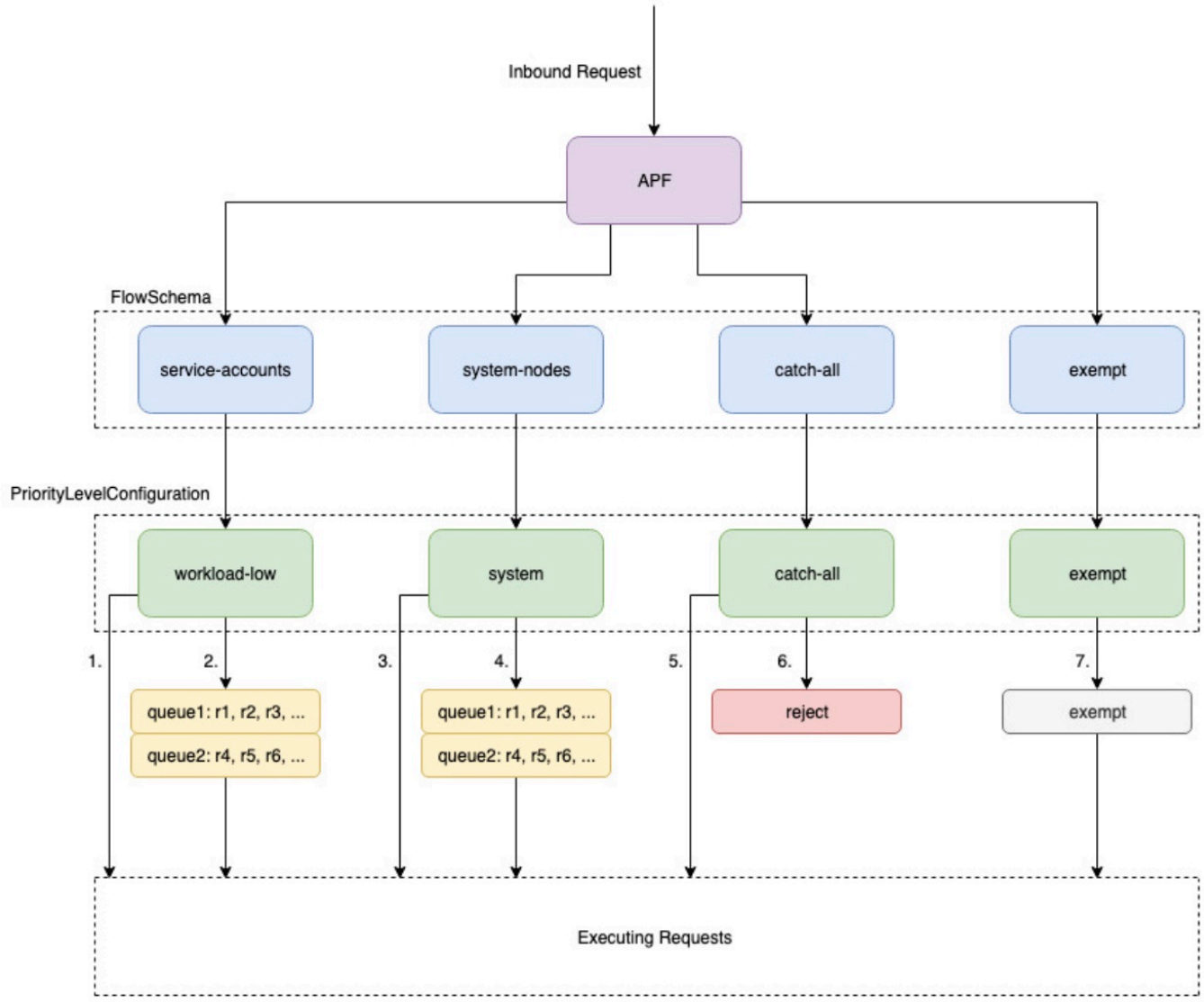
```
autoscalingGroups:
- name: eks-core-node-grp-20220823190924690000000011-80c1660e-030d-476d-cb0d-
d04d585a8fcb
  maxSize: 50
  minSize: 2
- name: eks-data_m1-20220824130553925600000011-5ec167fa-ca93-8ca4-53a5-003e1ed8d306
  maxSize: 450
  minSize: 2
- name: eks-data_m2-20220824130733258600000015-aac167fb-8bf7-429d-d032-e195af4e25f5
  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
```

Prioridad e imparcialidad de la API



1. If workload-low has not reached its concurrency limit, the request will be dispatched immediately.
2. If workload-low has reached its concurrency limit, the request will be queued prior to dispatch.
3. If system has not reached its concurrency limit, the request will be dispatched immediately.
4. If system has reached its concurrency limit, the request will be queued prior to dispatch.
5. If catch-all has not reached its concurrency limit, the request will be dispatched immediately.
6. If catch-all has reached its concurrency limit, the request will be dropped.
7. Requests mapping to exempt will always be dispatched.

Descripción general de

Para evitar sobrecargarse durante los períodos de aumento de solicitudes, el servidor de API limita el número de solicitudes a bordo que puede tener pendientes en un momento dado. Cuando se supere

este límite, el servidor de API empezará a rechazar las solicitudes y devolverá a los clientes un código de respuesta HTTP 429 para indicar «Demasiadas solicitudes». Es preferible que el servidor descarte las solicitudes y haga que los clientes lo intenten más tarde a no establecer límites en el número de solicitudes por parte del servidor y sobrecargar el plano de control, lo que podría provocar una disminución del rendimiento o una falta de disponibilidad.

El mecanismo que utiliza Kubernetes para configurar cómo se dividen estas solicitudes entrantes entre los distintos tipos de solicitudes se denomina API Priority and Fairness. El servidor API configura el número total de solicitudes en vuelo que puede aceptar sumando los valores especificados en los indicadores y. `--max-requests-inflight` `--max-mutating-requests-inflight` EKS utiliza los valores predeterminados de 400 y 200 solicitudes para estos indicadores, lo que permite enviar un total de 600 solicitudes en un momento dado. Sin embargo, a medida que amplía el plano de control para adaptarlo a tamaños más grandes en respuesta al aumento de la utilización y la pérdida de carga de trabajo, también aumenta la cuota de solicitudes durante el vuelo hasta 2000 (sujeto a cambios). La APF especifica cómo se subdivide esta cuota de solicitudes durante el vuelo entre los distintos tipos de solicitudes. Tenga en cuenta que los planos de control EKS tienen una alta disponibilidad y cuentan con al menos 2 servidores API registrados en cada clúster. Esto significa que el número total de solicitudes en vuelo que tu clúster puede gestionar es el doble (o superior si se amplía aún más horizontalmente) de la cuota de vuelo establecida por kube-apiserver. Esto equivale a varios miles en los clústeres de EKS más grandes. requests/second

Hay dos tipos de objetos de Kubernetes, llamados PriorityLevelConfigurations y FlowSchemas configuran cómo se divide el número total de solicitudes entre los distintos tipos de solicitudes. El servidor API mantiene estos objetos automáticamente y EKS usa la configuración predeterminada de estos objetos para la versión secundaria de Kubernetes en cuestión. PriorityLevelConfigurations representan una fracción del número total de solicitudes permitidas. Por ejemplo, a la carga de trabajo máxima PriorityLevelConfiguration se le asignan 98 de un total de 600 solicitudes. La suma de las solicitudes asignadas a todas PriorityLevelConfigurations será igual a 600 (o ligeramente superior a 600, ya que el servidor API redondeará si a un nivel determinado se le concede una fracción de la solicitud). Para comprobar PriorityLevelConfigurations el número de solicitudes asignadas a cada uno de tu clúster, puedes ejecutar el siguiente comando. Estos son los valores predeterminados en 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
```

El segundo tipo de objeto es. FlowSchemas Las solicitudes del servidor API con un conjunto determinado de propiedades se clasifican en la misma categoría FlowSchema. Estas propiedades incluyen el usuario autenticado o los atributos de la solicitud, como el grupo de API, el espacio de nombres o el recurso. A FlowSchema también especifica a qué se debe asignar PriorityLevelConfiguration este tipo de solicitud. Los dos objetos juntos dicen: «Quiero que este tipo de solicitudes se tengan en cuenta para este porcentaje de solicitudes a bordo». Cuando una solicitud llegue al servidor de API, comprobará cada una de ellas FlowSchemas hasta que encuentre una que coincida con todas las propiedades requeridas. Si varios FlowSchemas coinciden con una solicitud, el servidor de API elegirá la FlowSchema que tenga la menor prioridad coincidente, que se especifique como propiedad en el objeto.

El mapeo de FlowSchemas a se PriorityLevelConfigurations puede ver con este comando:

```
$ 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 puede tener un tipo de cola, rechazo o exención. Para los tipos Queue y Reject, se aplica un límite al número máximo de solicitudes en vuelo para ese nivel de prioridad; sin embargo, el comportamiento varía cuando se alcanza ese límite. Por ejemplo, Workload-high PriorityLevelConfiguration usa el tipo Queue y tiene 98 solicitudes disponibles para el controlador-administrador, el controlador de punto final, el programador, los controladores relacionados con eks y desde los pods que se ejecutan en el espacio de nombres del sistema kube. Como se utiliza el tipo Queue, el servidor de API intentará mantener las solicitudes en la memoria y esperará que el número de solicitudes en vuelo baje a menos de 98 antes de que se agote el tiempo de espera. Si se agota el tiempo de espera de una solicitud determinada o si ya hay demasiadas solicitudes en cola, el servidor API no tiene más remedio que descartar la solicitud y devolver al cliente un 429. Ten en cuenta que la puesta en cola puede impedir que una solicitud reciba un 429, pero esto conlleva el inconveniente de aumentar la end-to-end latencia de la solicitud.

Ahora consideremos el catch-all FlowSchema que se asigna al catch-all con el tipo Reject PriorityLevelConfiguration . Si los clientes alcanzan el límite de 13 solicitudes en vuelo, el servidor de API no hará cola y las descartará al instante con un código de respuesta de 429. Por último, las solicitudes asignadas a un PriorityLevelConfiguration tipo Exento nunca recibirán un 429 y siempre se enviarán de forma inmediata. Se usa para solicitudes de alta prioridad, como las solicitudes healthz o las solicitudes que provienen del grupo system:masters.

Supervisión de las APF y de las solicitudes descartadas

Para confirmar si se está descartando alguna solicitud debido a la APF, se `apiserver_flowcontrol_rejected_requests_total` pueden monitorear las métricas del servidor de API para comprobar qué solicitudes se ven afectadas FlowSchemas y. PriorityLevelConfigurations Por ejemplo, esta métrica muestra que se FlowSchema descartaron 100 solicitudes de las cuentas de servicio debido a que se agotó el tiempo de espera de las solicitudes en las colas con poca carga de trabajo:

```
% 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
```

Para comprobar qué tan cerca PriorityLevelConfiguration está una determinada persona de recibir 429 segundos o de experimentar un aumento de la latencia debido a las colas, puedes comparar la diferencia entre el límite de simultaneidad y la simultaneidad en uso. En este ejemplo, tenemos un búfer de 100 solicitudes.

```
% 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
```

Para comprobar si un determinado número de solicitudes PriorityLevelConfiguration está en cola pero no necesariamente se han cancelado, `apiserver_flowcontrol_current_inqueue_requests` puedes consultar la métrica de:

```
% 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
```

Otras métricas útiles de Prometheus incluyen:

- apiserver_flowcontrol_dispatched_requests_total
- apiserver_flowcontrol_request_execution_seconds
- apiserver_flowcontrol_request_wait_duration_seconds

Consulte la documentación preliminar para obtener una lista completa de las métricas de la APF.

Prevenir las solicitudes rechazadas

Evite los 429 modificando su carga de trabajo

Cuando la APF cancela solicitudes por PriorityLevelConfiguration superar el número máximo permitido de solicitudes durante el vuelo, los clientes de la zona afectada FlowSchemas pueden reducir el número de solicitudes que se están ejecutando en un momento dado. Esto se puede lograr reduciendo el número total de solicitudes realizadas durante el período en el que se están produciendo 429 solicitudes. Tenga en cuenta que las solicitudes de larga duración, como las costosas llamadas a listas, son especialmente problemáticas porque se consideran solicitudes en vuelo durante todo el tiempo que se estén ejecutando. Reducir el número de estas costosas solicitudes u optimizar la latencia de estas llamadas de lista (por ejemplo, reduciendo la cantidad de objetos recuperados por solicitud o pasando a utilizar una solicitud de vigilancia) puede ayudar a reducir la simultaneidad total que requiere la carga de trabajo determinada.

Evita los 429 segundos cambiando la configuración de tu APF

> **Warning**
>
> Cambie la configuración de APF predeterminada únicamente si sabe lo que está haciendo. Los ajustes de APF mal configurados pueden provocar la interrupción de las solicitudes del servidor API y provocar importantes interrupciones en la carga de trabajo.

Otro enfoque para evitar que se rechacen las solicitudes consiste en cambiar la configuración predeterminada FlowSchemas o PriorityLevelConfigurations instalada en los clústeres de EKS. EKS instala la configuración inicial predeterminada para FlowSchemas y PriorityLevelConfigurations para la versión secundaria de Kubernetes en cuestión. El servidor API reconciliará automáticamente estos objetos con sus valores predeterminados si se modifican, a menos que la siguiente anotación en los objetos esté configurada como falsa:

```
metadata:
  annotations:
    apf.kubernetes.io/autoupdate-spec: "false"
```

A un nivel alto, la configuración del APF se puede modificar de la siguiente manera:

- Asigne más capacidad a bordo a las solicitudes que le interesen.

- Aísle las solicitudes costosas o que no sean esenciales y que puedan agotar la capacidad de otros tipos de solicitudes.

Esto se puede lograr cambiando el valor predeterminado FlowSchemas PriorityLevelConfigurations o creando nuevos objetos de estos tipos. Los operadores pueden aumentar los valores de los PriorityLevelConfigurations objetos relevantes assuredConcurrencyShares para aumentar la fracción de solicitudes durante el vuelo que se les asignan. Además, el número de solicitudes que se pueden poner en cola en un momento dado también se puede aumentar si la aplicación puede gestionar la latencia adicional provocada por las solicitudes que se ponen en cola antes de enviarse.

Como alternativa, se pueden crear PriorityLevelConfigurations objetos FlowSchema y objetos nuevos que sean específicos de la carga de trabajo del cliente. Tenga en cuenta que si se asigna más assuredConcurrencyShares a los ya existentes PriorityLevelConfigurations o a los nuevos PriorityLevelConfigurations , se reducirá el número de solicitudes que pueden gestionar otros grupos, ya que el límite general se mantendrá en 600 en vuelo por servidor API.

Al realizar cambios en los valores predeterminados de APF, estas métricas deben supervisarse en un clúster que no sea de producción para garantizar que los cambios en la configuración no provoquen 429 errores imprevistos:

1. Todos `apiserver_flowcontrol_rejected_requests_total` deberían monitorizar la métrica para garantizar que ningún FlowSchemas grupo comience a descartar solicitudes.
2. Los valores correspondientes a `apiserver_flowcontrol_nominal_limit_seats` y `apiserver_flowcontrol_current_executing_seats` deben compararse para garantizar que la simultaneidad utilizada no corra el riesgo de superar el límite de ese nivel de prioridad.

Un caso de uso común para definir una nueva FlowSchema y PriorityLevelConfiguration es el aislamiento. Supongamos que queremos aislar las llamadas a eventos de listas de larga duración procedentes de los pods para incluirlas en su propia cuota de solicitudes. Esto evitará que las solicitudes importantes de los pods que utilizan las cuentas FlowSchema de servicio existentes reciban 429 y se queden sin capacidad de solicitudes. Recuerde que el número total de solicitudes a bordo es limitado; sin embargo, en este ejemplo se muestra que la configuración del APF se puede modificar para dividir mejor la capacidad de solicitudes para una carga de trabajo determinada:

Ejemplo de FlowSchema objeto para aislar las solicitudes de eventos de la lista:

```
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
```

- Esto FlowSchema captura todas las llamadas a eventos de la lista realizadas por las cuentas de servicio en el espacio de nombres predeterminado.
- La prioridad coincidente 8000 es inferior al valor de 9000 utilizado por las cuentas de servicio existentes, por FlowSchema lo que estas llamadas a eventos de la lista coincidirán con list-events-default-service las cuentas de servicio y no con las cuentas de servicio.
- Estamos utilizando el catch-all PriorityLevelConfiguration para aislar estas solicitudes. Este grupo solo permite que estas llamadas a eventos de lista de larga duración utilicen 13 solicitudes en vuelo. Los pods empezarán a recibir 429 en cuanto intenten emitir más de 13 de estas solicitudes de forma simultánea.

Recuperando recursos en el servidor de API

Obtener información del servidor de API es un comportamiento esperado para clústeres de cualquier tamaño. A medida que se amplía la cantidad de recursos del clúster, la frecuencia de las solicitudes y el volumen de datos pueden convertirse rápidamente en un obstáculo para el plano de control

y provocar latencia y lentitud de la API. Dependiendo de la gravedad de la latencia, si no se tiene cuidado, se producirán tiempos de inactividad inesperados.

Saber qué es lo que se solicita y con qué frecuencia es el primer paso para evitar este tipo de problemas. Esta es una guía para limitar el volumen de consultas en función de las mejores prácticas de escalado. Las sugerencias de esta sección se proporcionan por orden, empezando por las opciones que mejor escalan.

Utilice informadores compartidos

Al crear controladores y sistemas de automatización que se integren con la API de Kubernetes, a menudo necesitará obtener información de los recursos de Kubernetes. Si buscas estos recursos con regularidad, se puede producir una carga significativa en el servidor de la API.

Si utilizas un informador de la biblioteca Client-Go, tendrás la ventaja de estar pendiente de los cambios en los recursos en función de los eventos, en lugar de realizar sondeos para detectar cambios. Los informadores reducen aún más la carga al utilizar una caché compartida para los eventos y los cambios, de forma que varios controladores vigilan los mismos recursos no supongan una carga adicional.

Los controladores deben evitar sondear los recursos de todo el clúster sin etiquetas ni selectores de campo, especialmente en clústeres grandes. Cada encuesta sin filtrar requiere que etcd envíe una gran cantidad de datos innecesarios a través del servidor API para que el cliente los filtre. Al filtrar en función de las etiquetas y los espacios de nombres, puedes reducir la cantidad de trabajo que el servidor de API debe realizar para cumplir con las solicitudes y los datos enviados al cliente.

Optimiza el uso de la API de Kubernetes

Cuando llames a la API de Kubernetes con controladores personalizados o automatizados, es importante que limites las llamadas solo a los recursos que necesitas. Sin límites, puedes provocar una carga innecesaria en el servidor de la API, etc.

Se recomienda utilizar el argumento watch siempre que sea posible. Sin argumentos, el comportamiento predeterminado es enumerar los objetos. Para usar watch en lugar de list, puedes añadirlo `?watch=true` al final de tu solicitud de API. Por ejemplo, para colocar todos los pods en el espacio de nombres predeterminado con un reloj, usa lo siguiente:

```
/api/v1/namespaces/default/pods?watch=true
```

Si publicas objetos, debes limitar el alcance de lo que publicas y la cantidad de datos devueltos. Puedes limitar los datos devueltos añadiendo `limit=500` argumentos a las solicitudes. El `fieldSelector` argumento y la `/namespace/` ruta pueden ser útiles para asegurarte de que tus listas tengan un alcance tan limitado como sea necesario. Por ejemplo, para enumerar solo los pods en ejecución en el espacio de nombres predeterminado, usa la siguiente ruta de API y los argumentos siguientes.

```
/api/v1/namespaces/default/pods?fieldSelector=status.phase=Running&limit=500
```

O bien, haz una lista de todos los pods que se ejecutan con:

```
/api/v1/pods?fieldSelector=status.phase=Running&limit=500
```

Otra opción para limitar las llamadas de vigilancia o los objetos de la lista es utilizarla `resourceVersions`, que puedes consultar en la documentación de Kubernetes. Sin ningún `resourceVersion` argumento, recibirá la versión más reciente disponible, que requiere una lectura de quórum etcd, que es la lectura más cara y lenta de la base de datos. La ResourceVersion depende de los recursos que esté intentando consultar y se encuentra en el campo. `metadata.resourseVersion` Esto también se recomienda en caso de utilizar llamadas de vigilancia y no solo llamadas de lista

Hay una opción especial `resourceVersion=0` disponible que devolverá los resultados de la caché del servidor de la API. Esto puede reducir la carga de etcd, pero no admite la paginación.

```
/api/v1/namespaces/default/pods?resourceVersion=0
```

Se recomienda usar watch con una ResourceVersion configurada como el valor conocido más reciente recibido de la lista o visualización anterior. Esto se gestiona automáticamente en client-go. Sin embargo, se sugiere verificarlo dos veces si está utilizando un cliente k8s en otros idiomas.

```
/api/v1/namespaces/default/pods?watch=true&resourceVersion=362812295
```

Si llamas a la API sin ningún argumento, será la que requiera más recursos para el servidor de la API y etcd. Esta llamada incluirá todos los pods de todos los espacios de nombres sin paginar ni limitar el alcance y requerirá que se lea quórum desde etcd.

```
/api/v1/pods
```

Evita el estruendo de las manadas DaemonSet

A DaemonSet garantiza que todos los nodos (o algunos) ejecuten una copia de un pod. A medida que los nodos se unen al clúster, el daemonset-controller crea pods para esos nodos. A medida que los nodos abandonan el clúster, esos pods se recolectan como basura. Al eliminar un DaemonSet , se limpiarán los pods que haya creado.

Algunos usos típicos de a DaemonSet son:

- Ejecutar un daemon de almacenamiento en clúster en cada nodo
- Ejecutar un daemon de recopilación de registros en cada nodo
- Ejecutar un daemon de monitoreo de nodos en cada nodo

En clústeres con miles de nodos, la creación de uno nuevo DaemonSet, la actualización de uno o el aumento del número de nodos puede provocar una carga elevada en el plano de control. DaemonSet Si los DaemonSet pods emiten costosas solicitudes de servidor API al arrancar el pod, pueden provocar un uso elevado de los recursos en el plano de control debido a la gran cantidad de solicitudes simultáneas.

En condiciones normales de funcionamiento, puedes usar `RollingUpdate` a para garantizar el despliegue gradual de nuevos DaemonSet pods. Con una estrategia de `RollingUpdate` actualización, tras actualizar una DaemonSet plantilla, el mando elimina las DaemonSet cápsulas antiguas y crea nuevas DaemonSet cápsulas automáticamente y de forma controlada. Se DaemonSet ejecutará como máximo un pod en cada nodo durante todo el proceso de actualización. Puede realizar una implementación gradual `maxUnavailable` configurando 1, `maxSurge` 0 y `minReadySeconds` 60. Si no especificas una estrategia de actualización, Kubernetes utilizará de forma predeterminada la creación de a `RollingUpdate` con `maxUnavailable` 1, `maxSurge` 0 y 0. `minReadySeconds`

```
minReadySeconds: 60
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 0
    maxUnavailable: 1
```

A `RollingUpdate` garantiza el despliegue gradual de los nuevos DaemonSet pods si el ya DaemonSet está creado y tiene el número esperado de `Ready` pods en todos los nodos. En

determinadas condiciones, las estrategias no contemplan problemas de rebaño estrepitosos. `RollingUpdate`

Evita que las manadas atronen sobre la creación DaemonSet

De forma predeterminada, independientemente de la `RollingUpdate` configuración, el daemonset-controlador del nodo kube-controller-manager creará cápsulas para todos los nodos coincidentes de forma simultánea al crear uno nuevo. DaemonSet Para forzar el despliegue gradual de los pods después de crear un DaemonSet, puedes usar una o. `NodeSelector` `NodeAffinity` Esto creará un nodo DaemonSet que no coincida con ningún nodo y, a continuación, podrás actualizarlos gradualmente para que puedan ejecutar un pod desde allí DaemonSet a una velocidad controlada. Puedes seguir este enfoque:

- Añada una etiqueta a todos los nodos para`run-daemonset=false`.

```
kubectl label nodes --all run-daemonset=false
```

- Crea la tuya DaemonSet con una `NodeAffinity` configuración que coincida con cualquier nodo sin `run-daemonset=false` etiqueta. Al principio, esto hará DaemonSet que no tengas los pods correspondientes.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: run-daemonset
          operator: NotIn
          values:
          - "false"
```

- Quita la `run-daemonset=false` etiqueta de tus nodos a una velocidad controlada. Puedes usar este script bash como ejemplo:

```
#!/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
```

- Si lo desea, elimine la `NodeAffinity` configuración de su DaemonSet objeto. Ten en cuenta que esto también activará `RollingUpdate` y reemplazará gradualmente todos los DaemonSet módulos existentes a medida que la DaemonSet plantilla haya cambiado.

Evite el estruendo de las manadas al escalar los nodos

Al igual que en DaemonSet la creación, la creación de nuevos nodos a un ritmo rápido puede provocar el inicio simultáneo de un gran número de DaemonSet módulos. Deberías crear nuevos nodos a una velocidad controlada para que el controlador cree nodos DaemonSet a la misma velocidad. Si esto no es posible, puedes hacer que los nuevos nodos inicialmente no sean aptos para los existentes DaemonSet utilizando`NodeAffinity`. A continuación, podéis añadir gradualmente una etiqueta a los nuevos nodos para que el daemonset-controlador cree cápsulas a una velocidad controlada. Puedes seguir este enfoque:

- Añada una etiqueta a todos los nodos existentes para `run-daemonset=true`

```
kubectl label nodes --all run-daemonset=true
```

- Actualiza tu DaemonSet `NodeAffinity` configuración para que coincida con cualquier nodo que tenga una `run-daemonset=true` etiqueta. Ten en cuenta que esto también activará `RollingUpdate` y reemplazará gradualmente todos los DaemonSet pods existentes debido a un cambio en la DaemonSet plantilla. Debes esperar `RollingUpdate` a que se complete antes de continuar con el siguiente paso.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: run-daemonset
```

```
          operator: In
          values:
          - "true"
```

- Cree nuevos nodos en el clúster. Tenga en cuenta que estos nodos no tendrán la `run-daemonset=true` etiqueta, por lo que no coincidirán con esos nodos. DaemonSet
- Añada la `run-daemonset=true` etiqueta a sus nuevos nodos (que actualmente no tienen la `run-daemonset` etiqueta) a una velocidad controlada. Puedes usar este script bash como ejemplo:

```
#!/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
```

- Si lo desea, elimine la `NodeAffinity` configuración de su DaemonSet objeto y elimine la `run-daemonset` etiqueta de todos los nodos.

Evita que las actualizaciones se apoderen de las manadas DaemonSet

Una `RollingUpdate` política solo respetará la `maxUnavailable` configuración de los DaemonSet pods que estén configurados. `Ready` Si a DaemonSet tiene solo `NotReady` pods o un gran porcentaje de `NotReady` ellos y actualizas su plantilla, el daemonset-controlador creará nuevos pods simultáneamente para todos los pods. `NotReady` Esto puede provocar problemas con el ruido de las manadas si hay un número importante de ellas, por ejemplo, si las `NotReady` cápsulas se bloquean continuamente o no obtienen imágenes.

Para forzar el despliegue gradual de los pods cuando actualizas a DaemonSet y los hay, puedes cambiar temporalmente la estrategia de actualización de los `NotReady` pods de a. DaemonSet `RollingUpdate` `OnDelete` De este modo`OnDelete`, después de actualizar una DaemonSet plantilla, el mando crea nuevos pods tras eliminar manualmente los antiguos para que puedas controlar el lanzamiento de nuevos pods. Puedes seguir este enfoque:

- Compruebe si tiene `NotReady` cápsulas en su DaemonSet.
- Si no, puedes actualizar la DaemonSet plantilla de forma segura y la `RollingUpdate` estrategia garantizará una implementación gradual.
- En caso afirmativo, primero debes actualizar la tuya DaemonSet para usar la `OnDelete` estrategia.

```
updateStrategy:
  type: OnDelete
```

- A continuación, actualiza tu DaemonSet plantilla con los cambios necesarios.
- Tras esta actualización, puedes eliminar los DaemonSet pods antiguos emitiendo solicitudes de eliminación de pods a un ritmo controlado. Puedes usar este script bash como ejemplo en el que el DaemonSet nombre es fluentd-elasticsearch en el espacio de nombres del sistema kube:

```
#!/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
```

- Por último, puedes volver a la estrategia anterior. DaemonSet `RollingUpdate`

Plano de datos de Kubernetes

La selección de los tipos de EC2 instancias es posiblemente una de las decisiones más difíciles a las que se enfrentan los clientes cuando se trata de clústeres con múltiples cargas de trabajo. No existe una solución one-size-fits definitiva. Estos son algunos consejos que le ayudarán a evitar los errores más comunes relacionados con el escalado de la computación.

Escalado automático de nodos

Te recomendamos que utilices el escalado automático de nodos, ya que reduce el esfuerzo y se integra perfectamente con Kubernetes. Se recomiendan los grupos de nodos gestionados y Karpenter para clústeres de gran escala.

Los grupos de nodos gestionados le ofrecerán la flexibilidad de los grupos de Amazon EC2 Auto Scaling con ventajas adicionales para la configuración y las actualizaciones gestionadas. Se puede escalar con el escalador automático de clústeres de Kubernetes y es una opción común para los clústeres que tienen diversas necesidades informáticas.

Karpenter es un escalador automático de nodos nativo de carga de trabajo de código abierto creado por AWS. Escala los nodos de un clúster en función de los requisitos de carga de trabajo en materia de recursos (p. ej., la GPU) y de las limitaciones y tolerancias (p. ej., la dispersión por zonas) sin gestionar los grupos de nodos. Los nodos se crean directamente, EC2 lo que evita las cuotas predeterminadas de los grupos de nodos (450 nodos por grupo) y proporciona una mayor flexibilidad de selección de instancias con una menor sobrecarga operativa. Recomendamos a los clientes que utilicen Karpenter siempre que sea posible.

Usa muchos tipos de instancias diferentes EC2

Cada región de AWS tiene un número limitado de instancias disponibles por tipo de instancia. Si crea un clúster que usa solo un tipo de instancia y escala el número de nodos más allá de la capacidad de la región, recibirá un error que indica que no hay instancias disponibles. Para evitar este problema, no debes limitar arbitrariamente el tipo de instancias que se pueden usar en tu clúster.

Karpenter utilizará un amplio conjunto de tipos de instancias compatibles de forma predeterminada y seleccionará una instancia en el momento del aprovisionamiento en función de los requisitos de carga de trabajo pendientes, la disponibilidad y el coste. Puede ampliar la lista de tipos de instancias utilizados en la clave de. `karpenter.k8s.aws/instance-category` NodePools

El escalador automático de clústeres de Kubernetes requiere que los grupos de nodos tengan un tamaño similar para poder escalarlos de forma coherente. Debes crear varios grupos en función del tamaño de la CPU y la memoria y escalarlos de forma independiente. Utilice el ec2-instance-selector para identificar las instancias que tienen un tamaño similar para sus grupos de nodos.

```
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
```

Prefiera nodos más grandes para reducir la carga del servidor de API

A la hora de decidir qué tipos de instancias usar, un menor número de nodos grandes supondrá una menor carga en el plano de control de Kubernetes, ya que habrá menos kubelets y en ejecución. DaemonSets Sin embargo, es posible que los nodos grandes no se utilicen plenamente como los nodos más pequeños. Los tamaños de los nodos deben evaluarse en función de la disponibilidad de la carga de trabajo y los requisitos de escala.

Un clúster con tres instancias u-24tb1.metal (24 TB de memoria y 448 núcleos) tiene 3 kubelets y, de forma predeterminada, estaría limitado a 110 pods por nodo. Si sus pods utilizan 4 núcleos cada uno, es de esperar (4 núcleos x 110 = 440 núcleos por nodo). Con un clúster de 3 nodos, tu capacidad para gestionar un incidente en una instancia sería baja, ya que la interrupción de una instancia podría afectar a un tercio del clúster. Debes especificar los requisitos de los nodos y la distribución de los módulos en tus cargas de trabajo para que el programador de Kubernetes pueda colocar las cargas de trabajo correctamente.

Las cargas de trabajo deben definir los recursos que necesitan y la disponibilidad requerida mediante restricciones, tolerancias y. PodTopologySpread Deberían preferir los nodos más grandes que se puedan utilizar plenamente y cumplir los objetivos de disponibilidad para reducir la carga del plano de control, reducir las operaciones y reducir los costes.

El programador de Kubernetes intentará distribuir automáticamente las cargas de trabajo entre las zonas de disponibilidad y los hosts si hay recursos disponibles. Si no hay capacidad disponible, el escalador automático de clústeres de Kubernetes intentará añadir nodos en cada zona de disponibilidad de manera uniforme. Karpenter intentará añadir nodos de la forma más rápida y económica posible, a menos que la carga de trabajo especifique otros requisitos.

Para forzar que las cargas de trabajo se distribuyan con el programador y se creen nuevos nodos entre las zonas de disponibilidad, debe utilizar: 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
```

Utilice nodos de tamaños similares para lograr un rendimiento uniforme de las cargas de trabajo

Las cargas de trabajo deben definir el tamaño de los nodos en los que deben ejecutarse para permitir un rendimiento uniforme y un escalado predecible. Una carga de trabajo que requiera 500 millones de CPU tendrá un rendimiento diferente en una instancia de 4 núcleos que en una de 16 núcleos. Evita los tipos de instancias que utilizan ráfagas, CPUs como las instancias de la serie T.

Para garantizar que tus cargas de trabajo tengan un rendimiento uniforme, una carga de trabajo puede utilizar las etiquetas de Karpenter compatibles para segmentar tamaños de instancias específicos.

```
kind: deployment
...
spec:
  template:
    spec:
    containers:
    nodeSelector:
      karpenter.k8s.aws/instance-size: 8xlarge
```

Las cargas de trabajo que se programen en un clúster con el escalador automático de clústeres de Kubernetes deben hacer coincidir un selector de nodos con los grupos de nodos según la coincidencia de etiquetas.

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/nodegroup
            operator: In
            values:
            - 8-core-node-group    # match your node group name
```

Utilice los recursos informáticos de forma eficiente

Los recursos informáticos incluyen EC2 instancias y zonas de disponibilidad. El uso eficaz de los recursos informáticos aumentará la escalabilidad, la disponibilidad y el rendimiento y reducirá el coste total. El uso eficiente de los recursos es extremadamente difícil de predecir en un entorno de escalado automático con múltiples aplicaciones. Karpenter se creó para aprovisionar instancias bajo demanda en función de las necesidades de carga de trabajo a fin de maximizar la utilización y la flexibilidad.

Karpenter permite a las cargas de trabajo declarar el tipo de recursos informáticos que necesitan sin tener que crear primero grupos de nodos ni configurar etiquetas para nodos específicos. Consulte las prácticas recomendadas de Karpenter para obtener más información. Considere habilitar la consolidación en su aprovisionador de Karpenter para reemplazar los nodos que están infrautilizados.

Automatice las actualizaciones de Amazon Machine Image (AMI)

Si mantiene actualizados los componentes de los nodos de trabajo, garantizará que disponga de los últimos parches de seguridad y funciones compatibles con la API de Kubernetes. La actualización del kubelet es el componente más importante de la funcionalidad de Kubernetes, pero la automatización de los parches del sistema operativo, del kernel y de las aplicaciones instaladas localmente reducirá el mantenimiento a medida que se vaya escalando.

Se recomienda utilizar la última AMI de Amazon Linux 2 optimizada para Amazon EKS o Bottlerocket optimizada para Amazon EKS para la imagen de nodo. Karpenter utilizará automáticamente la última AMI disponible para aprovisionar nuevos nodos en el clúster. Los grupos de nodos gestionados actualizarán la AMI durante una actualización del grupo de nodos, pero no actualizarán el ID de la AMI en el momento del aprovisionamiento de nodos.

Para los grupos de nodos gestionados, debe actualizar la plantilla de lanzamiento del Grupo Auto Scaling (ASG) con la nueva AMI IDs cuando estén disponibles para las versiones de parches. Las versiones secundarias de AMI (por ejemplo, 1.23.5 a 1.24.3) estarán disponibles en la consola y la API de EKS como actualizaciones para el grupo de nodos. Las versiones de parches (p. ej., de la 1.23.5 a la 1.23.6) no se presentarán como actualizaciones para los grupos de nodos. Si desea mantener su grupo de nodos actualizado con las versiones de parches de la AMI, debe crear una nueva versión de la plantilla de lanzamiento y dejar que el grupo de nodos sustituya las instancias por la nueva versión de la AMI.

Puede encontrar la AMI más reciente disponible en esta página o utilizar la AWS CLI.

```
aws ssm get-parameter \
  --name /aws/service/eks/optimized-ami/1.24/amazon-linux-2/recommended/image_id \
  --query "Parameter.Value" \
  --output text
```

Utilice varios volúmenes de EBS como contenedores

Los volúmenes de EBS tienen una cuota input/output (de E/S) basada en el tipo de volumen (por ejemplo, gp3) y el tamaño del disco. Si sus aplicaciones comparten un único volumen raíz de EBS con el host, esto puede agotar la cuota de disco de todo el host y provocar que otras aplicaciones esperen hasta alcanzar la capacidad disponible. Las aplicaciones escriben en el disco si escriben archivos en su partición superpuesta, montan un volumen local desde el host y también cuando se desconectan de forma estándar (STDOUT), según el agente de registro utilizado.

Para evitar el I/O agotamiento del disco, debe montar un segundo volumen en la carpeta de estado del contenedor (por ejemplo, /run/containerd), usar volúmenes EBS independientes para el almacenamiento de la carga de trabajo y deshabilitar el registro local innecesario.

Para montar un segundo volumen en sus EC2 instancias mediante eksctl, puede utilizar un grupo de nodos con la siguiente configuración:

```
managedNodeGroups:
  - name: al2-workers
    amiFamily: AmazonLinux2
    desiredCapacity: 2
    volumeSize: 80
    additionalVolumes:
      - volumeName: '/dev/sdz'
```

```
        volumeSize: 100
    preBootstrapCommands:
    - |
      "systemctl stop containerd"
      "mkfs -t ext4 /dev/nvme1n1"
      "rm -rf /var/lib/containerd/*"
      "mount /dev/nvme1n1 /var/lib/containerd/"
      "systemctl start containerd"
```

Si utilizas terraform para aprovisionar tus grupos de nodos, consulta algunos ejemplos en EKS Blueprints for terraform. Si utiliza Karpenter para aprovisionar nodos, puede utilizar los datos de usuario de los nodos para `blockDeviceMappings`añadir volúmenes adicionales.

Para montar un volumen de EBS directamente en su pod, debe utilizar el controlador CSI de AWS EBS y consumir un volumen con una clase de almacenamiento.

```
---
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
```

Evite las instancias con límites de conexión de EBS bajos si las cargas de trabajo utilizan volúmenes de EBS

EBS es una de las formas más fáciles de que las cargas de trabajo tengan almacenamiento persistente, pero también tiene limitaciones de escalabilidad. Cada tipo de instancia tiene un número máximo de volúmenes de EBS que se pueden adjuntar. Las cargas de trabajo deben declarar los tipos de instancias en los que deben ejecutarse y limitar el número de réplicas en una sola instancia contaminada por Kubernetes.

Deshabilita los registros innecesarios en el disco

Evite el registro local innecesario al no ejecutar las aplicaciones mediante la depuración del registro en producción y la desactivación del registro que lee y escribe en el disco con frecuencia. Journald es el servicio de registro local que mantiene un búfer de registro en la memoria y lo vacía en el disco periódicamente. Se prefiere Journald a syslog, que registra cada línea inmediatamente en el disco. Al deshabilitar syslog, también se reduce la cantidad total de almacenamiento que se necesita y se evitan complicadas reglas de rotación de registros. Para deshabilitar syslog, puedes añadir el siguiente fragmento a tu configuración de cloud-init:

```
runcmd:
  - [ systemctl, disable, --now, syslog.service ]
```

Las instancias de parcheo se implementan cuando la velocidad de actualización del sistema operativo es necesaria

> **Important**
>
> Solo se deben aplicar parches a las instancias instaladas cuando sea necesario. Amazon recomienda tratar la infraestructura como algo inmutable y probar minuciosamente las

actualizaciones que se promocionan en entornos inferiores de la misma manera que lo hacen las aplicaciones. Esta sección se aplica cuando eso no es posible.

Se tarda unos segundos en instalar un paquete en un host Linux existente sin interrumpir las cargas de trabajo en contenedores. El paquete se puede instalar y validar sin acordonar, agotar ni reemplazar la instancia.

Para reemplazar una instancia, primero debe crear, validar y distribuir una nueva. AMIs Se debe crear un sustituto para la instancia, y se debe acordonar y vaciar la instancia anterior. Luego, las cargas de trabajo deben crearse en la nueva instancia, verificarse y repetirse para todas las instancias que necesiten ser parcheadas. Reemplazar las instancias de forma segura sin interrumpir las cargas de trabajo lleva horas, días o semanas.

Amazon recomienda utilizar una infraestructura inmutable que se cree, pruebe y promueva a partir de un sistema declarativo automatizado, pero si tiene la necesidad de parchar los sistemas rápidamente, tendrá que parchear los sistemas y sustituirlos a medida que haya nuevos AMIs disponibles. Debido a la gran diferencia de tiempo entre la aplicación de parches y la sustitución de los sistemas, recomendamos utilizar AWS Systems Manager Patch Manager para automatizar la aplicación de parches a los nodos cuando sea necesario.

Los nodos de parches le permitirán implementar rápidamente las actualizaciones de seguridad y reemplazar las instancias de forma regular una vez que se haya actualizado la AMI. Si utiliza un sistema operativo con un sistema de archivos raíz de solo lectura, como Flatcar Container Linux o Bottlerocket OS, le recomendamos que utilice los operadores de actualización que funcionan con esos sistemas operativos. El operador de actualización de Flatcar Linux y el operador de actualización de Bottlerocket reiniciarán las instancias para mantener los nodos actualizados automáticamente.

Servicios de clúster

Los servicios de clúster se ejecutan dentro de un clúster de EKS, pero no son cargas de trabajo de los usuarios. Si tiene un servidor Linux, a menudo necesita ejecutar servicios como NTP, syslog y un motor de ejecución de contenedores para soportar sus cargas de trabajo. Los servicios de clúster son servicios de soporte similares que le ayudan a automatizar y operar su clúster. En Kubernetes, normalmente se ejecutan en el espacio de nombres del sistema kube y algunos se ejecutan como. DaemonSets

Se espera que los servicios de clúster tengan un alto tiempo de actividad y, a menudo, son fundamentales durante las interrupciones y para la solución de problemas. Si un servicio de clúster principal no está disponible, puede perder el acceso a los datos que pueden ayudar a recuperar o evitar una interrupción (por ejemplo, un uso elevado del disco). Deben ejecutarse en instancias informáticas dedicadas, como un grupo de nodos independiente o AWS Fargate. Esto garantizará que los servicios de clúster no se vean afectados en las instancias compartidas por cargas de trabajo que puedan estar aumentando o consumiendo más recursos.

CoredNS de escala

El escalado de CoredNS tiene dos mecanismos principales. Reducir el número de llamadas al servicio CoreDNS y aumentar el número de réplicas.

Reduzca las consultas externas reduciendo los puntos

La configuración ndots especifica cuántos puntos (también conocidos como «puntos») de un nombre de dominio se consideran suficientes para evitar consultar el DNS. Si su aplicación tiene una configuración de ndots de 5 (predeterminada) y solicita recursos de un dominio externo como api.example.com (2 puntos), se consultará CoreDNS para cada dominio de búsqueda definido en /etc/resolv.conf para un dominio más específico. De forma predeterminada, se buscarán los siguientes dominios antes de realizar una solicitud externa.

```
api.example.<namespace>.svc.cluster.local
api.example.svc.cluster.local
api.example.cluster.local
api.example.<region>.compute.internal
```

`region`Los valores `namespace` y se sustituirán por el espacio de nombres de las cargas de trabajo y la región de procesamiento. Es posible que tengas dominios de búsqueda adicionales en función de la configuración del clúster.

Puedes reducir la cantidad de solicitudes a CoreDNS reduciendo la opción ndots de tu carga de trabajo o calificando completamente tus solicitudes de dominio al incluir un seguimiento. (por ejemplo). `api.example.com.` Si tu carga de trabajo se conecta a servicios externos a través de DNS, te recomendamos configurar ndots en 2 para que las cargas de trabajo no hagan innecesarias las consultas de DNS en clústeres dentro del clúster. Puedes configurar un servidor DNS y un dominio de búsqueda diferentes si la carga de trabajo no requiere acceso a los servicios del clúster.

```
spec:
```

```
  dnsPolicy: "None"
  dnsConfig:
    options:
      - name: ndots
        value: "2"
      - name: edns0
```

Si reduce los puntos a un valor demasiado bajo o si los dominios a los que se va a conectar no incluyen suficiente especificidad (incluso al final), es posible que se produzcan errores en las búsquedas de DNS. Asegúrese de probar cómo afectará esta configuración a sus cargas de trabajo.

Escalar coredNS horizontalmente

Las instancias de CoredNS se pueden escalar añadiendo réplicas adicionales a la implementación. Se recomienda utilizar NodeLocal DNS o el escalador automático proporcional del clúster para escalar CoredNS.

NodeLocal El DNS requerirá ejecutar una instancia por nodo, lo DaemonSet que requiere más recursos de procesamiento en el clúster, pero evitará solicitudes de DNS fallidas y reducirá el tiempo de respuesta de las consultas de DNS en el clúster. El escalador automático proporcional del clúster escalará los CoredNS en función del número de nodos o núcleos del clúster. No se trata de una correlación directa con las consultas de solicitud, pero puede resultar útil en función de las cargas de trabajo y del tamaño del clúster. La escala proporcional predeterminada consiste en añadir una réplica adicional por cada 256 núcleos o 16 nodos del clúster, lo que ocurra primero.

Si utilizas el complemento CoredNS EKS, considera habilitar la opción de escalado automático.
El escalador automático de CoreDNS ajusta dinámicamente el número de réplicas de CoreDNS mediante la supervisión del número de nodos y los núcleos de la CPU, mediante una fórmula que toma el máximo de (nodos ÷ 16) o (núcleos de CPU), ampliándose inmediatamente cuando es necesario y reduciéndose gradualmente para mantener la estabilidad.

Escale el servidor Kubernetes Metrics de forma vertical

El servidor de métricas de Kubernetes admite el escalado horizontal y vertical. Al escalar horizontalmente el servidor de métricas, tendrá una alta disponibilidad, pero no se escalará horizontalmente para gestionar más métricas de clústeres. Deberás escalar verticalmente el servidor de métricas según sus recomendaciones a medida que los nodos y las métricas recopiladas se agreguen al clúster.

El servidor de métricas guarda en la memoria los datos que recopila, agrega y almacena. A medida que crece un clúster, aumenta la cantidad de datos que almacena el servidor de métricas. En clústeres grandes, el servidor de métricas requerirá más recursos informáticos que la reserva de memoria y CPU especificada en la instalación predeterminada. Puedes usar el escalador automático Vertical Pod (VPA) o el Addon Resizer para escalar el servidor de métricas. El Addon Resizer se escala verticalmente en proporción a los nodos de trabajo y el VPA se escala en función del uso de la CPU y la memoria.

Duración lamentable de CoredNS

Los pods utilizan el `kube-dns` Servicio para la resolución de nombres. Kubernetes usa la NAT de destino (DNAT) para redirigir el `kube-dns` tráfico de los nodos a los pods de backend de CoredNS. A medida que escala la implementación de CoreDNS`kube-proxy`, actualiza las reglas y cadenas de iptables en los nodos para redirigir el tráfico de DNS a los pods de CoreDNS. La propagación de nuevos puntos finales al aumentar la escala y eliminar las reglas al reducir la escala de CoredNS puede tardar entre 1 y 10 segundos, según el tamaño del clúster.

Este retraso de propagación puede provocar errores en la búsqueda de DNS cuando se termina un pod de CoreDNS y las reglas de iptables del nodo no se han actualizado. En este escenario, el nodo puede seguir enviando consultas de DNS a un pod de CoreDNS terminado.

Puedes reducir los errores de búsqueda de DNS configurando una duración mínima en tus pods de CoreDNS. Mientras esté en modo lameduck, CoreDNS seguirá respondiendo a las solicitudes durante el vuelo. Si se establece una duración mínima, se retrasará el proceso de cierre de CoreDNS, lo que dará a los nodos el tiempo que necesitan para actualizar sus reglas y cadenas de iptables.

Recomendamos establecer la duración del lameduck de CoredNS en 30 segundos.

Sonda de preparación de CoredNS

Recomendamos utilizarla `/ready` en lugar de utilizarla `/health` para la sonda de preparación de CoreDNS.

De acuerdo con la recomendación anterior de establecer la duración del lameduck en 30 segundos, lo que proporciona tiempo suficiente para que las reglas de iptables del nodo se actualicen antes de la finalización del pod, si se emplea, `/ready` en lugar de para `/health` la sonda de preparación de CoreDNS, se garantiza que el pod de CoreDNS esté completamente preparado al inicio para responder rápidamente a las solicitudes de DNS.

```
readinessProbe:
  httpGet:
    path: /ready
    port: 8181
    scheme: HTTP
```

Para obtener más información sobre el complemento CoredNS Ready, consulte https://coredns. io/plugins/ready/

Agentes de registro y monitoreo

Los agentes de registro y supervisión pueden añadir una carga significativa al plano de control del clúster, ya que consultan el servidor de API para enriquecer los registros y las métricas con metadatos de la carga de trabajo. El agente de un nodo solo tiene acceso a los recursos del nodo local para ver datos como el contenedor y el nombre del proceso. Al consultar el servidor de API, puede añadir más detalles, como el nombre y las etiquetas de la implementación de Kubernetes. Esto puede ser extremadamente útil para la solución de problemas, pero perjudicial para el escalado.

Como hay tantas opciones diferentes para el registro y la supervisión, no podemos mostrar ejemplos para todos los proveedores. Con fluentbit, recomendamos habilitar Use_Kubelet para obtener metadatos del kubelet local en lugar del servidor API de Kubernetes y `Kube_Meta_Cache_TTL` establecerlo en un número que reduzca las llamadas repetidas cuando los datos se puedan almacenar en caché (por ejemplo, 60).

Escalar la supervisión y el registro tiene dos opciones generales:

- Desactivar las integraciones
- Muestreo y filtrado

La desactivación de las integraciones no suele ser una opción porque se pierden los metadatos del registro. Esto elimina el problema de escalado de la API, pero generará otros problemas al no disponer de los metadatos necesarios cuando se necesiten.

El muestreo y el filtrado reducen la cantidad de métricas y registros que se recopilan. Esto reducirá la cantidad de solicitudes a la API de Kubernetes y reducirá la cantidad de almacenamiento necesaria para las métricas y los registros que se recopilan. Al reducir los costes de almacenamiento, se reducirán los costes del sistema en general.

La capacidad de configurar el muestreo depende del software del agente y se puede implementar en diferentes puntos de ingesta. Es importante añadir el muestreo lo más cerca posible del agente, ya que es probablemente allí donde se producen las llamadas al servidor de la API. Póngase en contacto con su proveedor para obtener más información sobre el soporte de muestreo.

Si utiliza CloudWatch y CloudWatch Logs, puede añadir el filtrado de agentes mediante los patrones que se describen en la documentación.

Para evitar perder registros y métricas, debe enviar sus datos a un sistema que pueda almacenar datos en búfer en caso de que se produzca una interrupción en el terminal receptor. Con fluentbit, puede utilizar Amazon Kinesis Data Firehose para conservar temporalmente los datos, lo que reduce la posibilidad de sobrecargar la ubicación final de almacenamiento de datos.

Cargas de trabajo

Las cargas de trabajo influyen en el tamaño que puede escalar su clúster. Las cargas de trabajo que utilizan APIs mucho Kubernetes limitarán la cantidad total de cargas de trabajo que puedes tener en un solo clúster, pero hay algunos valores predeterminados que puedes cambiar para reducir la carga.

Las cargas de trabajo de un clúster de Kubernetes tienen acceso a funciones que se integran con la API de Kubernetes (por ejemplo, Secrets y ServiceAccounts), pero estas funciones no siempre son obligatorias y se deben deshabilitar si no se utilizan. Limitar el acceso a las cargas de trabajo y la dependencia del plano de control de Kubernetes aumentará la cantidad de cargas de trabajo que puede ejecutar en el clúster y mejorará la seguridad de los clústeres al eliminar el acceso innecesario a las cargas de trabajo e implementar prácticas de privilegios mínimos. Lea las prácticas recomendadas de seguridad para obtener más información.

Úselo IPv6 para redes de pods

No se puede hacer la transición de una VPC IPv4 a otra, IPv6 por lo que es importante habilitarla IPv6 antes de aprovisionar un clúster. Si la habilitas IPv6 en una VPC, no significa que tengas que usarla y, si tus pods y servicios la utilizan, IPv6 puedes seguir enrutando el tráfico hacia y desde IPv4 las direcciones. Consulte las prácticas recomendadas de redes de EKS para obtener más información.

El uso IPv6 en su clúster evita algunos de los límites de escalado de clústeres y cargas de trabajo más comunes. IPv6 evita el agotamiento de las direcciones IP cuando no se pueden crear pods y

nodos porque no hay ninguna dirección IP disponible. También presenta mejoras en el rendimiento por nodo, ya que los pods reciben las direcciones IP más rápido al reducir la cantidad de adjuntos ENI por nodo. Puede lograr un rendimiento de nodo similar mediante el modo de IPv4 prefijo en la CNI de la VPC, pero aun así debe asegurarse de tener suficientes direcciones IP disponibles en la VPC.

Limite la cantidad de servicios por espacio de nombres

La cantidad máxima de servicios en un espacio de nombres es de 5000 y la cantidad máxima de servicios en un clúster es de 10 000. Para ayudar a organizar las cargas de trabajo y los servicios, aumentar el rendimiento y evitar el impacto en cascada de los recursos incluidos en el ámbito de los espacios de nombres, recomendamos limitar la cantidad de servicios por espacio de nombres a 500.

La cantidad de reglas de tablas de IP que se crean por nodo con kube-proxy aumenta con la cantidad total de servicios del clúster. La generación de miles de reglas de tablas IP y el enrutamiento de paquetes mediante esas reglas repercute en el rendimiento de los nodos y añade latencia a la red.

Cree espacios de nombres de Kubernetes que abarquen un único entorno de aplicaciones siempre que el número de servicios por espacio de nombres sea inferior a 500. Esto hará que la detección de servicios sea lo suficientemente pequeña como para evitar los límites de detección de servicios y también puede ayudarte a evitar colisiones en los nombres de los servicios. Los entornos de aplicaciones (p. ej., dev, test, prod) deben utilizar clústeres de EKS independientes en lugar de espacios de nombres.

Comprenda las cuotas de Elastic Load Balancer

Al crear sus servicios, tenga en cuenta qué tipo de equilibrio de carga utilizará (por ejemplo, Network Load Balancer (NLB) o Application Load Balancer (ALB)). Cada tipo de balanceador de carga proporciona una funcionalidad diferente y tiene cuotas diferentes. Algunas de las cuotas predeterminadas se pueden ajustar, pero hay algunas cuotas máximas que no se pueden cambiar. Para ver las cuotas y el uso de su cuenta, consulte el panel Service Quotas en la consola de AWS.

Por ejemplo, el objetivo predeterminado de ALB es 1000. Si tienes un servicio con más de 1000 puntos de conexión, tendrás que aumentar la cuota, dividir el servicio en varios ALBs o utilizar Kubernetes Ingress. Los objetivos predeterminados de NLB son 3000, pero están limitados a 500 objetivos por zona de disponibilidad. Si tu clúster ejecuta más de 500 pods para un servicio NLB, tendrás que usar varios AZs o solicitar un aumento del límite de cuota.

Una alternativa a usar un balanceador de carga acoplado a un servicio es usar un controlador de ingreso. El controlador AWS Load Balancer puede crear recursos ALBs para la entrada, pero puede considerar la posibilidad de ejecutar un controlador dedicado en su clúster. Un controlador de entrada integrado en el clúster le permite exponer varios servicios de Kubernetes desde un único balanceador de cargas mediante la ejecución de un proxy inverso dentro del clúster. Los controladores tienen diferentes funciones, como la compatibilidad con la API Gateway, que pueden tener ventajas según el número y el tamaño de las cargas de trabajo.

Utilice Route 53, Global Accelerator o CloudFront

Para que un servicio que utilice varios balanceadores de carga esté disponible como un único punto de enlace, debe usar Amazon CloudFront, AWS Global Accelerator o Amazon Route 53 para exponer todos los balanceadores de carga como un único punto de enlace orientado al cliente. Cada opción tiene diferentes ventajas y se puede utilizar por separado o en conjunto, según sus necesidades.

Route 53 puede exponer varios balanceadores de carga con un nombre común y puede enviar tráfico a cada uno de ellos en función del peso asignado. Puede obtener más información sobre los pesos de DNS en la documentación y cómo implementarlos con el controlador de DNS externo de Kubernetes en la documentación del controlador Load Balancer de AWS.

Global Accelerator puede enrutar las cargas de trabajo a la región más cercana en función de la dirección IP solicitada. Esto puede resultar útil para las cargas de trabajo que se implementan en varias regiones, pero no mejora el enrutamiento a un solo clúster en una sola región. El uso de Route 53 en combinación con Global Accelerator ofrece beneficios adicionales, como la comprobación del estado y la conmutación automática por error si no hay una zona de disponibilidad disponible. Puedes ver un ejemplo del uso de Global Accelerator con Route 53 en esta entrada de blog.

CloudFront se puede usar con Route 53 y Global Accelerator o por sí solo para dirigir el tráfico a varios destinos. CloudFront almacena en caché los activos que se suministran desde las fuentes de origen, lo que puede reducir los requisitos de ancho de banda en función de lo que se esté sirviendo.

Úselo EndpointSlices en lugar de puntos finales

Cuando descubras pods que coincidan con una etiqueta de servicio, deberías usarlos EndpointSlicesen lugar de Endpoints. Los endpoints eran una forma sencilla de exponer los servicios a pequeña escala, pero los servicios grandes que se escalan automáticamente o se actualizan generan mucho tráfico en el plano de control de Kubernetes. EndpointSlices tienen una agrupación automática que permite cosas como sugerencias que tengan en cuenta la topología.

No todos los controladores lo utilizan de forma EndpointSlices predeterminada. Debe verificar la configuración del mando y habilitarlo si es necesario. En el caso del controlador Load Balancer de AWS, debe activar el indicador `--enable-endpoint-slices` opcional que desee utilizar. EndpointSlices

Si es posible, utilice secretos externos e inmutables

El kubelet guarda en caché las claves y valores actuales de los secretos que se utilizan en los volúmenes de los pods de ese nodo. El kubelet vigila los Secretos para detectar cambios. A medida que el clúster se amplía, el creciente número de relojes puede afectar negativamente al rendimiento del servidor API.

Existen dos estrategias para reducir el número de reproducciones en Secrets:

- Para las aplicaciones que no necesitan acceso a los recursos de Kubernetes, puedes deshabilitar el montaje automático de los secretos de las cuentas de servicio configurando Token: false automountServiceAccount
- Si los secretos de su aplicación son estáticos y no se modificarán en el futuro, márquelos como inmutables. El kubelet no mantiene una API que vigile los secretos inmutables.

Para deshabilitar el montaje automático de una cuenta de servicio en los pods, puedes usar la siguiente configuración en tu carga de trabajo. Puedes anular esta configuración si determinadas cargas de trabajo necesitan una cuenta de servicio.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app
automountServiceAccountToken: true
```

Supervisa la cantidad de secretos del clúster antes de que supere el límite de 10 000. Puede ver el recuento total de secretos de un clúster con el siguiente comando. Debe supervisar este límite a través de las herramientas de supervisión de clústeres.

```
kubectl get secrets -A | wc -l
```

Debe configurar la supervisión para alertar al administrador del clúster antes de que se alcance este límite. Considere la posibilidad de utilizar opciones de administración de secretos externas, como

AWS Key Management Service (AWS KMS) o Hashicorp Vault con el controlador CSI de Secrets Store.

Limite el historial de despliegues

Los pods pueden tardar en crearse, actualizarse o eliminarse porque los objetos antiguos se siguen rastreando en el clúster. Puedes reducir el número `revisionHistoryLimit` de despliegues para limpiar los objetos antiguos, ReplicaSets lo que reducirá la cantidad total de objetos rastreados por el Kubernetes Controller Manager. El límite de historial predeterminado para las implementaciones es de 10.

Si su clúster crea muchos objetos de trabajo mediante CronJobs u otros mecanismos, debe usar la `ttlSecondsAfterFinished`configuración para limpiar automáticamente los módulos antiguos del clúster. Esto eliminará los trabajos ejecutados correctamente del historial de trabajos después de un período de tiempo específico.

Desactivar enableServiceLinks de forma predeterminada

Cuando un pod se ejecuta en un nodo, el kubelet añade un conjunto de variables de entorno para cada servicio activo. Los procesos de Linux tienen un tamaño máximo para su entorno, que se puede alcanzar si tienes demasiados servicios en tu espacio de nombres. La cantidad de servicios por espacio de nombres no debe superar los 5000. Después, el número de variables del entorno de servicio supera los límites del shell, lo que provoca que los pods se bloqueen al iniciarse.

Existen otros motivos por los que los pods no deberían usar variables de entorno de servicio para la detección de servicios. Algunos ejemplos son los conflictos en los nombres de las variables de entorno, la filtración de nombres de servicios y el tamaño total del entorno. Debe usar CoredNS para descubrir los puntos finales del servicio.

Limite los webhooks de admisión dinámica por recurso

Los webhooks de admisión dinámica incluyen los webhooks de admisión y los webhooks mutantes. Son puntos finales de la API que no forman parte del plano de control de Kubernetes y que se invocan en secuencia cuando se envía un recurso a la API de Kubernetes. Cada webhook tiene un tiempo de espera predeterminado de 10 segundos y puede aumentar el tiempo que tarda una solicitud de API si tienes varios webhooks o si se agota el tiempo de espera de alguno de ellos.

Asegúrate de que tus webhooks tengan una alta disponibilidad, especialmente durante un incidente de zona de disponibilidad, y de que la Política de errores esté configurada correctamente para

rechazar el recurso o ignorar el error. No llames a los webhooks cuando no los necesites, ya que permite que los comandos --dry-run kubectl pasen por alto el webhook.

```
apiVersion: admission.k8s.io/v1
kind: AdmissionReview
request:
  dryRun: False
```

Los webhooks mutantes pueden modificar los recursos de forma sucesiva y frecuente. Si tienes 5 webhooks mutantes y despliegas 50 recursos, etcd almacenará todas las versiones de cada recurso hasta que se realice la compactación (cada 5 minutos) para eliminar las versiones antiguas de los recursos modificados. En este escenario, cuando etcd elimine los recursos sustituidos, se eliminará una versión de 200 recursos de etcd y, según el tamaño de los recursos, puede ocupar un espacio considerable en el host de etcd hasta que la desfragmentación se ejecute cada 15 minutos.

Esta desfragmentación puede provocar pausas en etcd, lo que podría tener otros efectos en la API y los controladores de Kubernetes. Debes evitar la modificación frecuente de recursos de gran tamaño o la modificación de cientos de recursos en rápida sucesión.

Compare las cargas de trabajo en varios clústeres

Si tiene dos clústeres que deberían tener un rendimiento similar pero no lo tienen, intente comparar las métricas para identificar el motivo.

Por ejemplo, comparar la latencia de los clústeres es un problema habitual. Esto suele deberse a una diferencia en el volumen de solicitudes de API. Puedes ejecutar la siguiente CloudWatch LogInsight consulta para entender la diferencia.

```
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
```

Puedes añadir filtros adicionales para acotarlo, por ejemplo, centrándote en todas las listas solicitadas`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
```

Teoría de escalado de Kubernetes

Los nodos frente a la tasa de abandono

A menudo, cuando hablamos de la escalabilidad de Kubernetes, lo hacemos en términos del número de nodos que hay en un solo clúster. Curiosamente, esta métrica rara vez es la más útil para entender la escalabilidad. Por ejemplo, un clúster de 5000 nodos con un número grande pero fijo de pods no pondría mucho stress en el plano de control después de la configuración inicial. Sin embargo, si cogiéramos un clúster de 1000 nodos e intentáramos crear 10 000 puestos de trabajo de corta duración en menos de un minuto, se ejercería una gran presión sostenida sobre el plano de control.

El simple hecho de utilizar el número de nodos para entender el escalamiento puede resultar engañoso. Es mejor pensar en términos de la tasa de cambio que se produce dentro de un período de tiempo específico (usemos un intervalo de 5 minutos para esta discusión, ya que esto es lo que las consultas de Prometheus suelen utilizar de forma predeterminada). Exploremos por qué encuadrar el problema en términos de la tasa de cambio puede darnos una mejor idea de lo que debemos ajustar para lograr la escala deseada.

Pensando en consultas por segundo

Kubernetes tiene varios mecanismos de protección para cada componente (el Kubelet, el Scheduler, el Kube Controller Manager y el servidor API) para evitar sobrecargar al siguiente eslabón de la cadena de Kubernetes. Por ejemplo, el Kubelet tiene una bandera para limitar las llamadas al servidor API a una velocidad determinada. Estos mecanismos de protección se expresan generalmente, aunque no siempre, en términos de consultas permitidas por segundo o QPS.

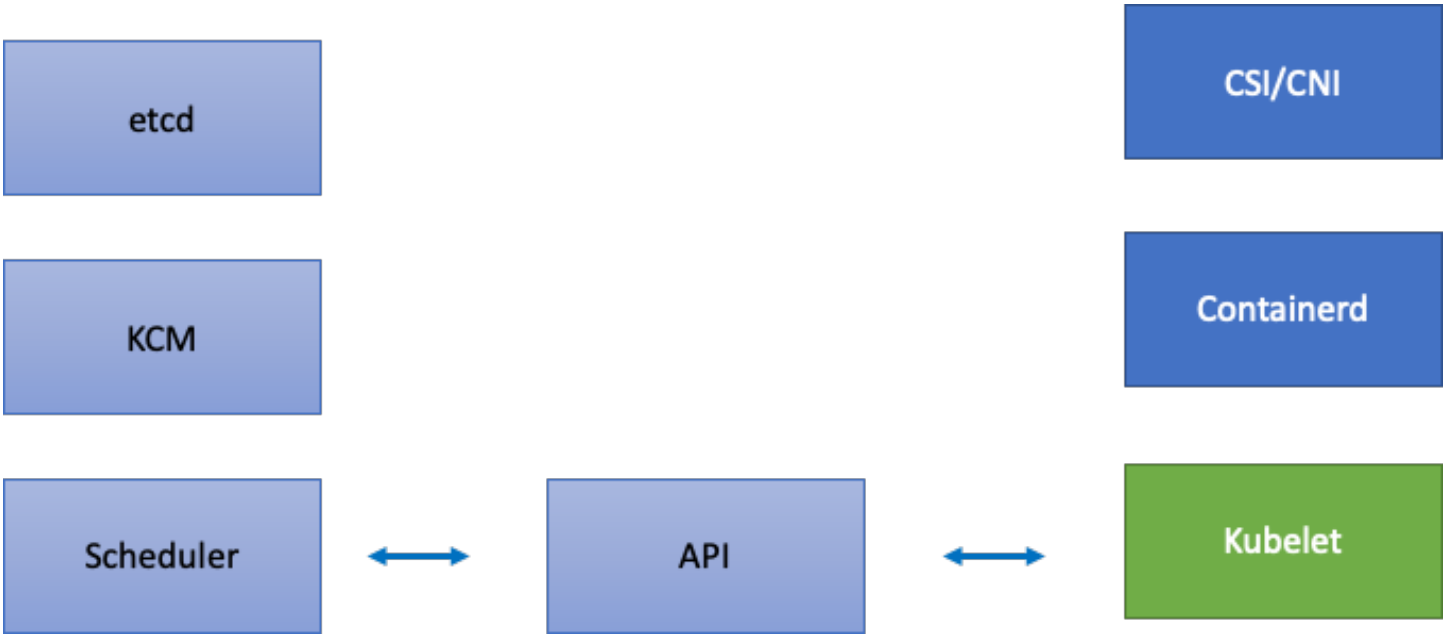
Se debe tener mucho cuidado al cambiar estos ajustes de QPS. Eliminar un obstáculo, como las consultas por segundo en un Kubelet, tendrá un impacto en otros componentes posteriores. Esto puede sobrecargar al sistema y lo hará a una velocidad superior a determinada, por lo que comprender y monitorear cada parte de la cadena de servicios es clave para escalar correctamente las cargas de trabajo en Kubernetes.

> **Note**
>
> El servidor de API tiene un sistema más complejo, con la introducción de la prioridad y la equidad de la API, que analizaremos por separado.

> **Note**
>
> Precaución: algunas métricas parecen ser las adecuadas, pero de hecho miden otra cosa. Por ejemplo, `kubelet_http_inflight_requests` se refiere solo al servidor de métricas de Kubelet, no al número de solicitudes de Kubelet a apiserver. Esto podría provocar que configuremos mal el indicador QPS en el Kubelet. Una consulta en los registros de auditoría de un Kubelet concreto sería una forma más fiable de comprobar las métricas.

Escalar los componentes distribuidos

Como EKS es un servicio gestionado, dividamos los componentes de Kubernetes en dos categorías: componentes gestionados por AWS, que incluyen etcd, Kube Controller Manager y Scheduler (en la parte izquierda del diagrama), y componentes configurables por el cliente, como Kubelet, Container Runtime y los distintos operadores que llaman a AWS, APIs como los controladores de redes y almacenamiento (en la parte derecha del diagrama). Dejamos el servidor de API en el medio aunque esté gestionado por AWS, ya que los clientes pueden configurar los ajustes de Prioridad y Equidad de la API.



Cuellos de botella ascendentes y descendentes

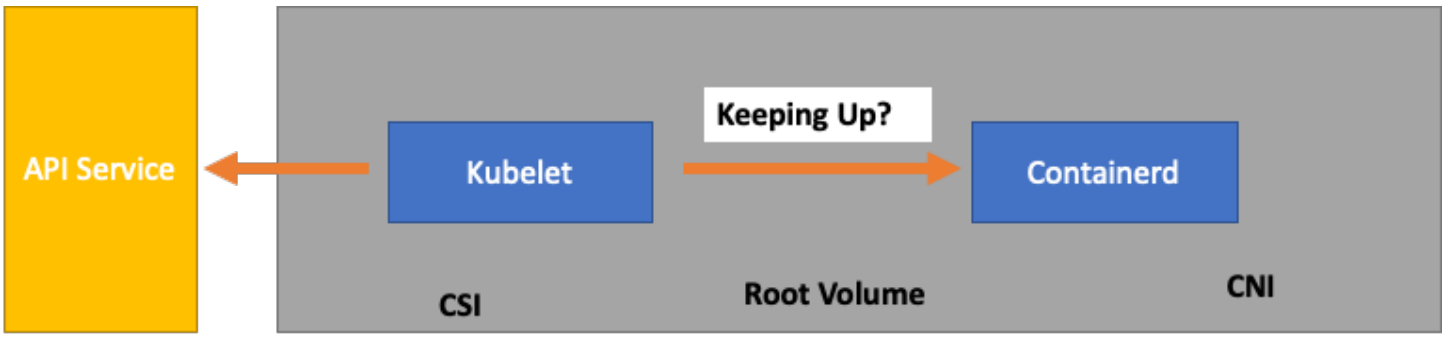
A medida que supervisamos cada servicio, es importante analizar las métricas en ambas direcciones para detectar cuellos de botella. Aprendamos cómo hacerlo usando Kubelet como ejemplo. Kubelet habla tanto con el servidor API como con el tiempo de ejecución del contenedor. ¿Cómo y qué debemos monitorear para detectar si alguno de los componentes tiene algún problema?

¿Cuántos pods hay por nodo

Si analizamos los números de escalado, como el número de pods que se pueden ejecutar en un nodo, podríamos tomar al pie de la letra los 110 pods por nodo que admite upstream.

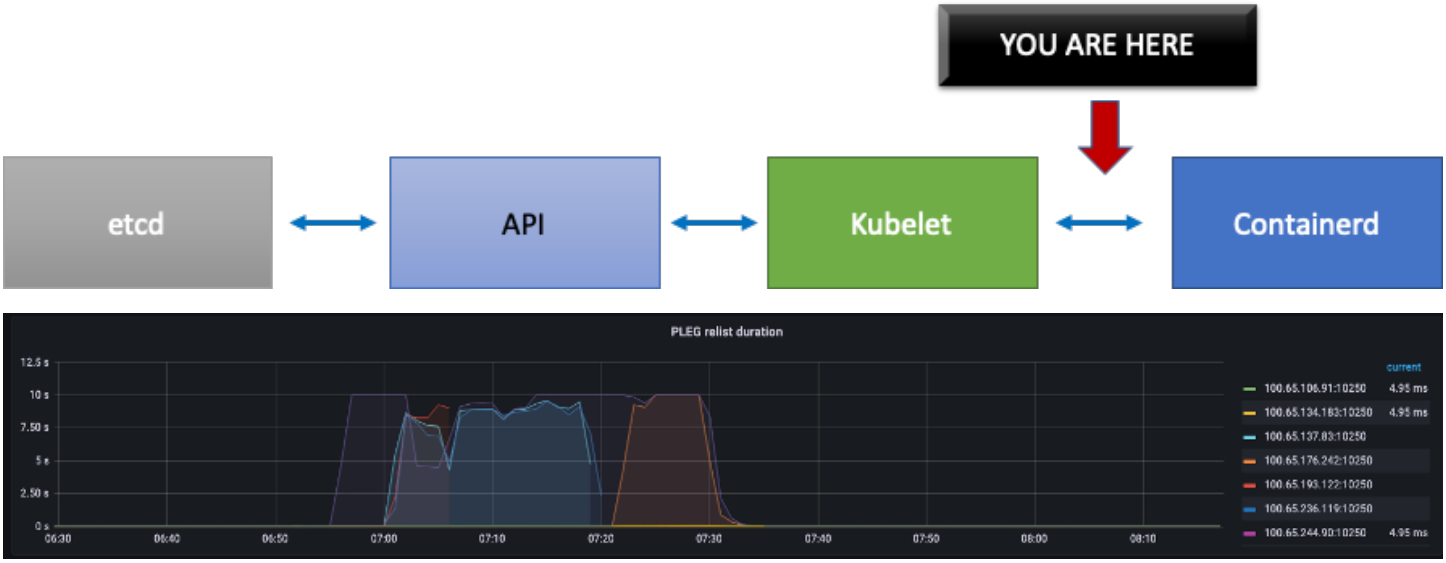
> **Note**
>
> https://kubernetes. io/docs/setup/best-practices/cluster-grande/

Sin embargo, es probable que su carga de trabajo sea más compleja de lo que se probó en una prueba de escalabilidad en Upstream. Para asegurarnos de que podemos dar servicio a la cantidad de módulos que queremos que funcionen en producción, asegurémonos de que el Kubelet está «a la altura» del tiempo de ejecución de Containerd.



Para simplificar demasiado, el Kubelet obtiene el estado de los pods a partir del tiempo de ejecución del contenedor (en nuestro caso, Containerd). ¿Qué pasaría si muchos pods cambiaran de estado demasiado rápido? Si la tasa de cambio es demasiado alta, las solicitudes [al entorno de ejecución del contenedor] pueden agotarse.

> **Note**
>
> Kubernetes está en constante evolución y, actualmente, este subsistema está experimentando cambios. https://github.com/kubernetes/mejoras/problemas/3386



En el gráfico anterior, vemos una línea plana que indica que acabamos de alcanzar el valor de tiempo de espera de la métrica de duración de la generación de eventos del ciclo de vida del pod. Si quieres ver esto en tu propio clúster, puedes usar la siguiente sintaxis de ProMQL.

```
increase(kubelet_pleg_relist_duration_seconds_bucket{instance="$instance"}
[$__rate_interval])
```

Si observamos este comportamiento de tiempo de espera, sabremos que hemos llevado al nodo por encima del límite que era capaz de soportar. Tenemos que corregir la causa del tiempo de espera antes de continuar. Esto puede lograrse reduciendo el número de módulos por nodo o buscando errores que puedan estar provocando un gran volumen de reintentos (lo que repercutiría en la tasa de abandono). La conclusión importante es que las métricas son la mejor manera de saber si un nodo es capaz de gestionar la tasa de abandono de los pods asignados en lugar de utilizar un número fijo.

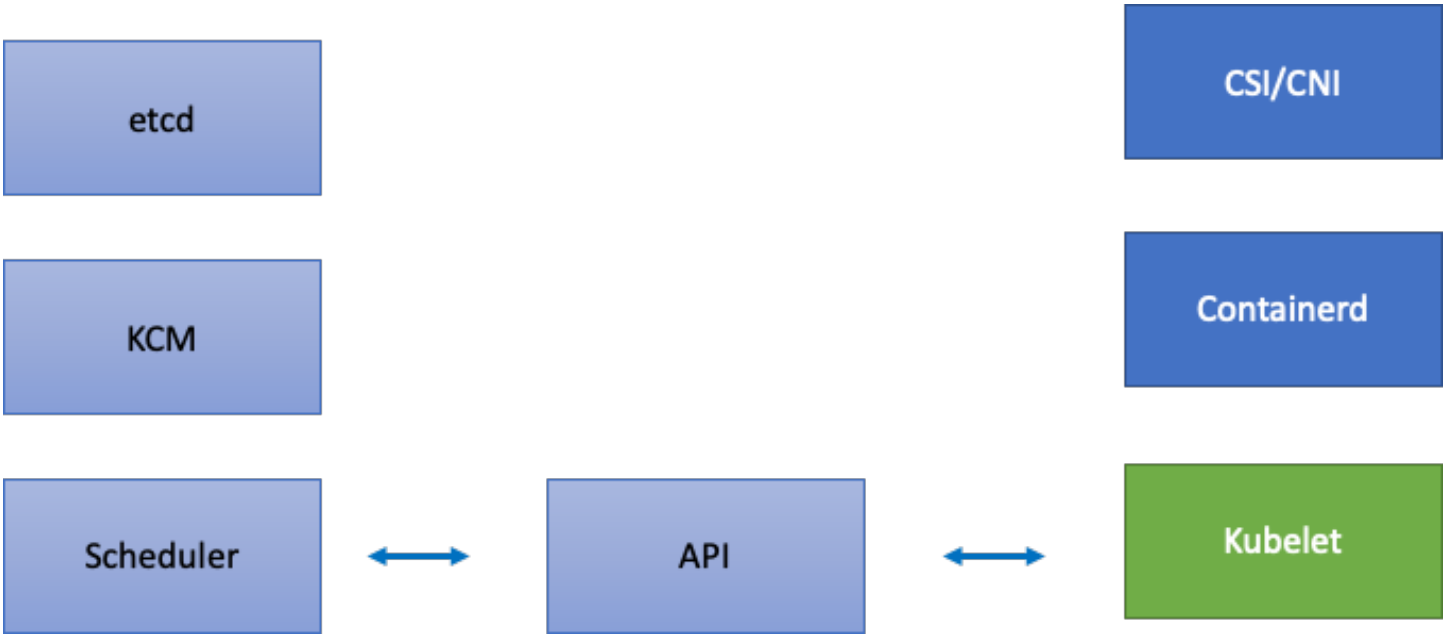
Escale por métricas

Si bien el concepto de usar métricas para optimizar los sistemas es antiguo, a menudo se pasa por alto cuando las personas comienzan su viaje a Kubernetes. En lugar de centrarnos en números específicos (es decir, 110 módulos por nodo), centramos nuestros esfuerzos en encontrar las métricas que nos ayudan a detectar los cuellos de botella en nuestro sistema. Comprender los umbrales correctos para estas métricas puede darnos un alto grado de confianza en que nuestro sistema está configurado de manera óptima.

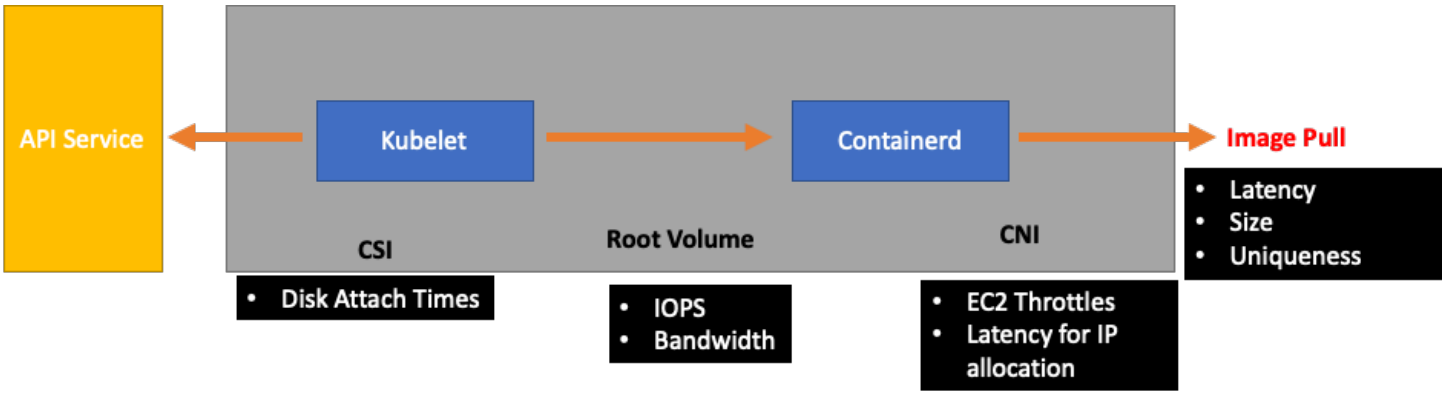
El impacto de los cambios

Un patrón común que podría causarnos problemas es centrarnos en el primer error métrico o logarítmico que parezca sospechoso. Cuando vimos que el Kubelet estaba agotando el tiempo de espera, podríamos probar cosas al azar, como aumentar la velocidad por segundo que el Kubelet puede enviar, etc. Sin embargo, es aconsejable analizar primero el panorama completo de todas las fases posteriores al error que encontremos. Realice cada cambio con un propósito y respaldado por datos.

Tras el Kubelet, estaría el tiempo de ejecución de Containerd (errores del pod), DaemonSets como el controlador de almacenamiento (CSI) y el controlador de red (CNI) que se comunican con la API, etc. EC2



Continuemos con el ejemplo anterior en el que el Kubelet no está a la altura del tiempo de ejecución. Hay varios puntos en los que podríamos empaquetar un nodo de forma tan densa que provocara errores.



Al diseñar el tamaño de nodo adecuado para nuestras cargas de trabajo, se trata de easy-to-overlook señales que podrían estar ejerciendo una presión innecesaria sobre el sistema, lo que limitaría tanto nuestra escala como nuestro rendimiento.

El coste de los errores innecesarios

Los controladores Kubernetes se destacan a la hora de volver a intentarlo cuando se presentan condiciones de error, pero esto tiene un coste. Estos reintentos pueden aumentar la presión sobre componentes como el Kube Controller Manager. Supervisar este tipo de errores es un elemento importante de las pruebas de escala.

Cuando se producen menos errores, es más fácil detectar los problemas en el sistema. Al asegurarnos periódicamente de que nuestros clústeres estén libres de errores antes de realizar operaciones importantes (como las actualizaciones), podemos simplificar los registros de solución de problemas cuando se producen imprevistos.

Ampliando nuestra visión

En los clústeres a gran escala con miles de nodos, no queremos buscar cuellos de botella de forma individual. En ProMQL podemos encontrar los valores más altos de un conjunto de datos mediante una función llamada topk; siendo K una variable, colocamos el número de elementos que queremos. Aquí utilizamos tres nodos para hacernos una idea de si todos los Kubelets del clúster están saturados. Hemos estado analizando la latencia hasta este punto, ahora veamos si el Kubelet descarta los eventos.

```
topk(3, increase(kubelet_pleg_discard_events{}[$__rate_interval]))
```

Desglosando esta afirmación.

- Usamos la variable Grafana `$__rate_interval` para asegurarnos de que obtiene las cuatro muestras que necesita. Esto evita un tema complejo de monitoreo con una variable simple.
- `topk`nos da solo los mejores resultados y el número 3 limita esos resultados a tres. Esta es una función útil para las métricas de todo el clúster.
- `{}`Si nos dice que no hay filtros, normalmente pondría el nombre del trabajo, sea cual sea la regla de extracción. Sin embargo, dado que estos nombres varían, lo dejaremos en blanco.

Dividir el problema por la mitad

Para solucionar un cuello de botella en el sistema, adoptaremos el enfoque de encontrar una métrica que muestre que hay un problema en sentido ascendente o descendente, ya que esto nos permite dividir el problema por la mitad. También será un principio fundamental de la forma en que mostraremos los datos de nuestras métricas.

Un buen punto de partida para este proceso es el servidor de API, ya que nos permite ver si hay algún problema con una aplicación cliente o con el plano de control.

Monitorización del plano de control

Servidor de API

Al analizar nuestro servidor de API, es importante recordar que una de sus funciones es limitar las solicitudes entrantes para evitar sobrecargar el plano de control. Lo que puede parecer un cuello de botella a nivel de servidor de API podría estar protegiéndolo de problemas más graves. Debemos tener en cuenta los pros y los contras de aumentar el volumen de solicitudes que se envían a través del sistema. Para determinar si se deben aumentar los valores del servidor API, he aquí una pequeña muestra de los aspectos que debemos tener en cuenta:

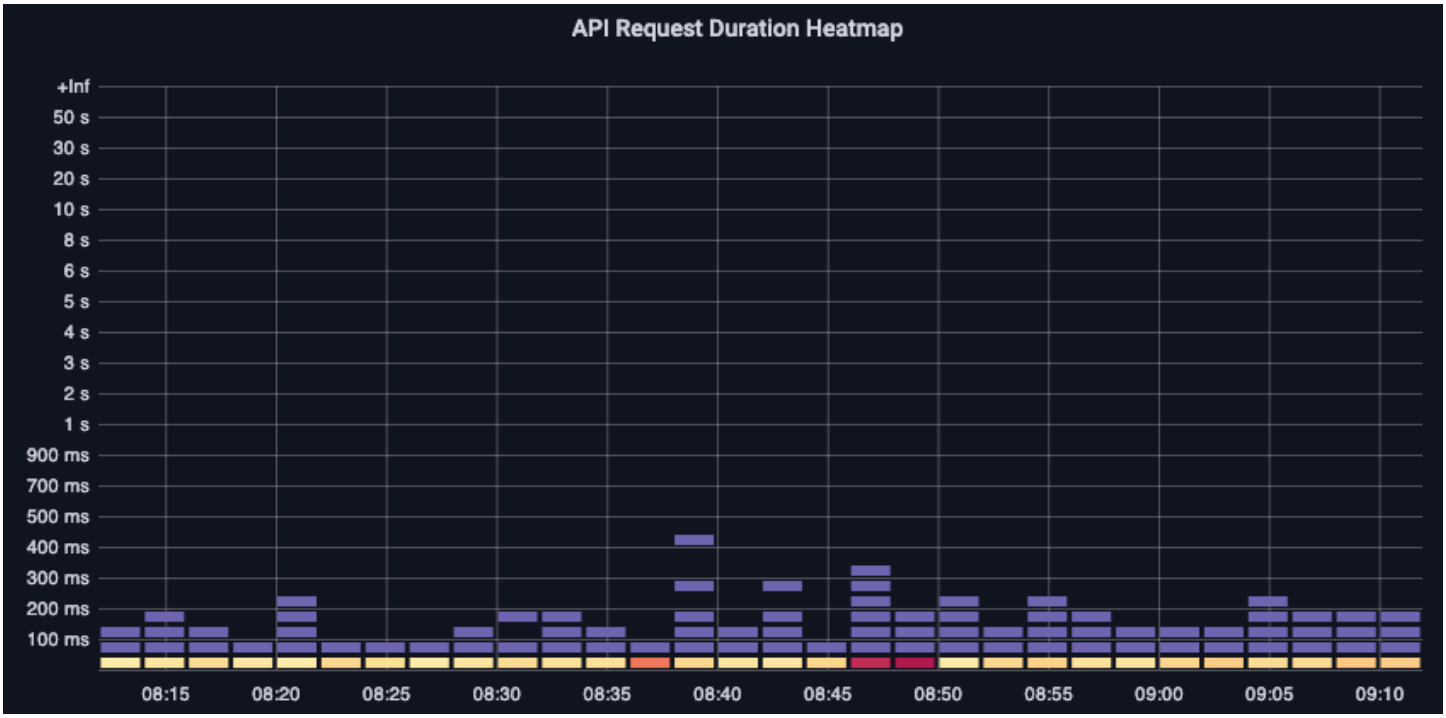
1. ¿Cuál es la latencia de las solicitudes que circulan por el sistema?
2. ¿Esa latencia es del propio servidor de API o es algo «descendente», como etcd?
3. ¿La profundidad de la cola del servidor API es un factor en esta latencia?
4. ¿Las colas de prioridad y equidad (APF) de la API están configuradas correctamente para los patrones de llamadas a la API que queremos?

¿Dónde está el problema?

Para empezar, podemos usar la métrica de latencia de la API para obtener una idea del tiempo que tarda el servidor de API en atender las solicitudes. Usemos el siguiente mapa de calor de ProMQL y Grafana para mostrar estos datos.

```
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**
>
> Para obtener información detallada sobre cómo monitorear el servidor de API con el panel de API utilizado en este artículo, consulta el siguiente blog



Todas estas solicitudes tardan menos de un segundo, lo que indica que el plano de control está gestionando las solicitudes a tiempo. Pero, ¿y si ese no fuera el caso?

El formato que utilizamos en la sección anterior sobre la duración de la solicitud de API es un mapa de calor. Lo bueno del formato de mapa de calor es que nos indica el valor de tiempo de espera de la API de forma predeterminada (60 segundos). Sin embargo, lo que realmente necesitamos saber es en qué umbral debería preocupar este valor antes de alcanzar el umbral de tiempo de espera. Para tener una idea aproximada de cuáles son los umbrales aceptables, podemos utilizar el SLO previo de Kubernetes, que puede consultarse aquí

> **Note**
>
> ¿Observa la función max en esta afirmación? Cuando se utilizan métricas que agregan varios servidores (de forma predeterminada, dos servidores API en EKS), es importante no calcular el promedio de esos servidores.

Patrones de tráfico asimétricos

¿Qué pasaría si un servidor API [pod] tuviera una carga ligera y el otro estuviera muy cargado? Si calculamos el promedio de esos dos números, podríamos malinterpretar lo que estaba sucediendo. Por ejemplo, aquí tenemos tres servidores API, pero toda la carga está en uno de estos servidores

API. Como regla general, todo lo que tenga varios servidores, como los servidores etcd y API, debe desglosarse a la hora de invertir en problemas de escala y rendimiento.



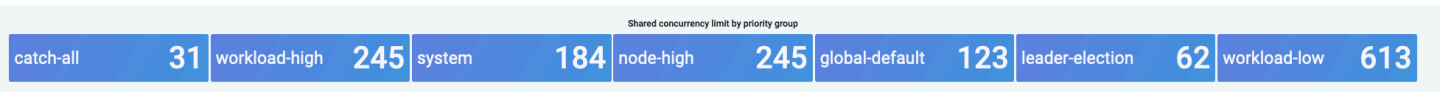
Con el cambio a API Priority and Fairness, el número total de solicitudes en el sistema es solo un factor para comprobar si el servidor de API tiene un exceso de suscripciones. Como el sistema ahora funciona a partir de una serie de colas, debemos comprobar si alguna de estas colas está llena y si el tráfico de esa cola se está reduciendo.

Veamos estas colas con la siguiente consulta:

```
max without(instance)(apiserver_flowcontrol_nominal_limit_seats{})
```

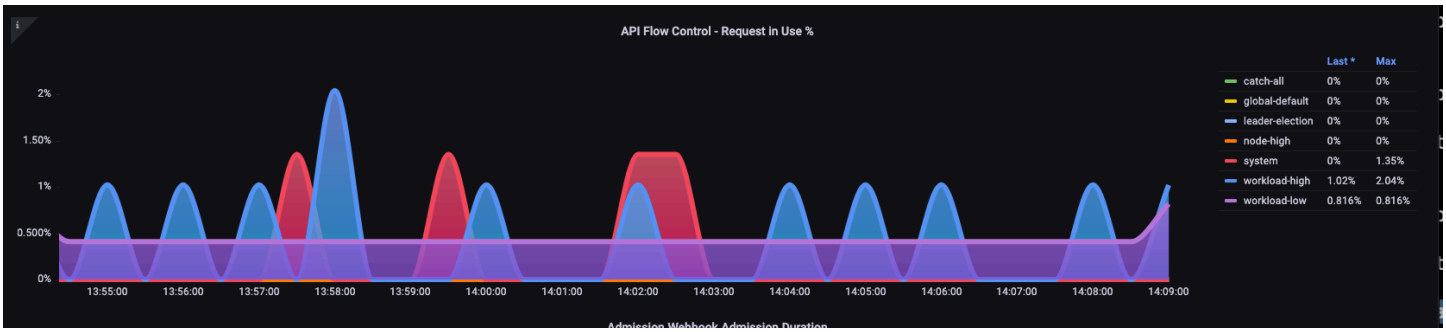
> **Note**
>
> Para obtener más información sobre cómo funciona la API A&F, consulte la siguiente guía de prácticas recomendadas

Aquí vemos los siete grupos de prioridades diferentes que vienen de forma predeterminada en el clúster



A continuación, queremos ver qué porcentaje de ese grupo de prioridades se está utilizando, de modo que podamos entender si un determinado nivel de prioridad se está saturando. Podría ser conveniente limitar las solicitudes en el nivel de carga de trabajo más bajo, pero no lo sería reducir las solicitudes en el nivel de elección de un líder.

El sistema API Priority and Fairness (APF) tiene varias opciones complejas, algunas de las cuales pueden tener consecuencias imprevistas. Un problema habitual que observamos sobre el terreno es el aumento de la profundidad de la cola hasta el punto de empezar a añadir una latencia innecesaria. Podemos monitorizar este problema mediante la `apiserver_flowcontrol_current_inqueue_request` métrica. Podemos comprobar si hay caídas utilizando el`apiserver_flowcontrol_rejected_requests_total`. Estas métricas tendrán un valor distinto de cero si algún grupo supera su simultaneidad.



Aumentar la profundidad de la cola puede convertir al servidor API en una fuente importante de latencia y debe hacerse con cuidado. Recomendamos ser prudentes con el número de colas que se crean. Por ejemplo, el número de comparticiones en un sistema EKS es de 600. Si creamos demasiadas colas, esto puede reducir el número de veces que se comparten las colas importantes que necesitan más rendimiento, como la cola para las elecciones de líderes o la cola del sistema. Si se crean demasiadas colas adicionales, puede resultar más difícil dimensionarlas correctamente.

Para centrarnos en un cambio simple e impactante que puedas realizar en APF, simplemente cogemos acciones de los grupos infrautilizados y aumentamos el tamaño de los grupos que están al máximo de su uso. Al redistribuir de forma inteligente las acciones entre estos grupos, puede reducir las probabilidades de que se produzcan caídas.

Para obtener más información, consulte la configuración de prioridad y equidad de la API en la Guía de mejores prácticas de EKS.

Latencia de API frente a latencia de etcd

¿Cómo podemos utilizar el servidor metrics/logs de API para determinar si hay un problema con el servidor de API, un problema del servidor upstream/downstream de API o una combinación de ambos? Para entenderlo mejor, veamos cómo se pueden relacionar el servidor API y etcd, y lo fácil que puede ser solucionar problemas en un sistema incorrecto.

En el siguiente gráfico, vemos la latencia del servidor API, pero también vemos que gran parte de esta latencia está correlacionada con el servidor etcd debido a que las barras del gráfico muestran

la mayor parte de la latencia a nivel etcd. Si hay 15 segundos de latencia etcd al mismo tiempo que hay 20 segundos de latencia del servidor API, entonces la mayor parte de la latencia se encuentra realmente en el nivel etcd.

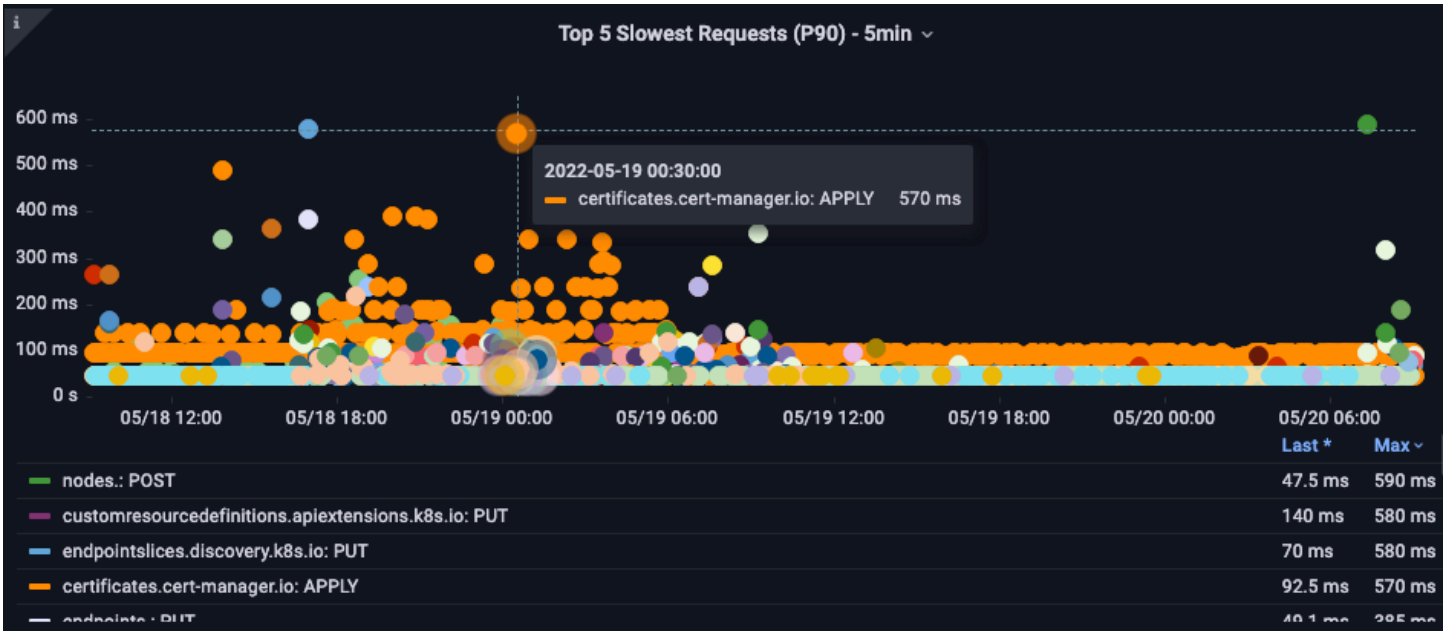
Al analizar todo el flujo, vemos que es aconsejable no centrarse únicamente en el servidor API, sino también buscar señales que indiquen que etcd está bajo presión (es decir, que los contadores de aplicaciones lentas aumentan). Lo que hace que un panel de control sea potente es poder pasar rápidamente al área problemática correcta con solo un vistazo.

> **Note**
>
> El panel de control de la sección se encuentra en https://github.com/RiskyAdventure/Solución de problemas- Dashboards/blob/main/api -troubleshooter.json



Problemas entre el plano de control y el lado del cliente

En este gráfico, buscamos las llamadas a la API que más tiempo tardaron en completarse durante ese período. En este caso, vemos que un recurso personalizado (CRD) llama a una función APPLY que es la llamada más latente durante el intervalo de tiempo de las 05:40.



Con estos datos, podemos utilizar una consulta Ad-Hoc ProMQL o CloudWatch Insights para extraer las solicitudes LIST del registro de auditoría durante ese período de tiempo y ver de qué aplicación se trata.

Encontrando la fuente con CloudWatch

La mejor forma de utilizar las métricas es encontrar el área problemática que queremos analizar y reducir tanto el marco temporal como los parámetros de búsqueda del problema. Una vez que tengamos estos datos, queremos pasar a los registros para obtener información más detallada sobre los tiempos y los errores. Para ello, convertiremos nuestros registros en métricas con CloudWatch Logs Insights.

Por ejemplo, para investigar el problema anterior, utilizaremos la siguiente consulta de CloudWatch Logs Insights para obtener el UserAgent y el RequestUri y poder determinar qué aplicación está causando esta latencia.

> **Note**
>
> Es necesario utilizar un contador adecuado para evitar que un reloj muestre un List/Resync comportamiento normal.

```
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
```

Al usar esta consulta, encontramos dos agentes diferentes que ejecutan una gran cantidad de operaciones de listas de alta latencia. Splunk y su CloudWatch agente. Con los datos, podemos tomar la decisión de eliminar, actualizar o reemplazar este controlador por otro proyecto.

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**
>
> Para obtener más información sobre este tema, consulte el siguiente blog

Programador

Como las instancias del plano de control de EKS se ejecutan en una cuenta de AWS independiente, no podremos recopilar esos componentes para obtener métricas (el servidor de API es la excepción). Sin embargo, dado que tenemos acceso a los registros de auditoría de estos componentes, podemos convertirlos en métricas para ver si alguno de los subsistemas está causando un cuello de botella de escalado. Usemos CloudWatch Logs Insights para ver cuántos pods no programados hay en la cola del programador.

Los pods no programados están en el registro del programador

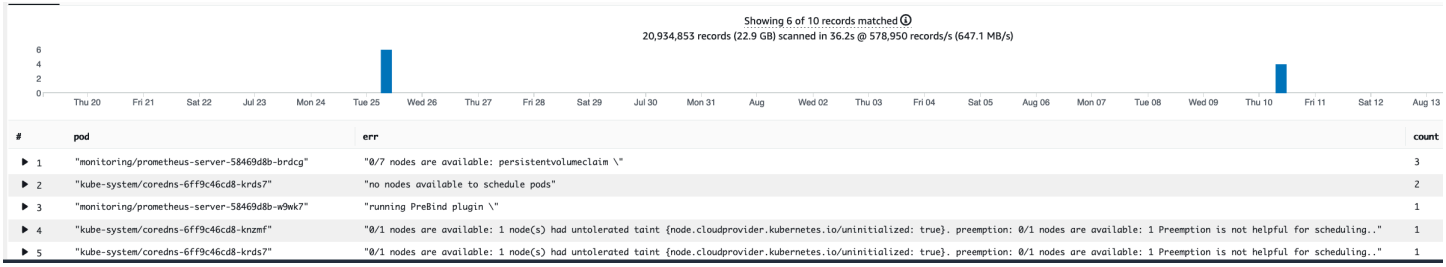
Si tuviéramos acceso a recopilar las métricas del programador directamente en un Kubernetes autogestionado (como Kops), utilizaríamos el siguiente ProMQL para entender el atraso acumulado del programador.

```
max without(instance)(scheduler_pending_pods)
```

Como no tenemos acceso a la métrica anterior en EKS, utilizaremos la siguiente consulta de CloudWatch Logs Insights para ver los retrasos acumulados y comprobar cuántos pods no se pudieron programar de forma no programada durante un período de tiempo determinado. Luego, podríamos analizar más a fondo los mensajes en las horas de mayor actividad para entender la naturaleza del cuello de botella. Por ejemplo, los nodos no giran lo suficientemente rápido o el limitador de velocidad está en el propio programador.

```
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
```

Aquí vemos los errores del programador que indican que el pod no se desplegó porque el PVC de almacenamiento no estaba disponible.



#	pod	err	count
▶ 1	"monitoring/prometheus-server-58469d8b-brdcg"	"0/7 nodes are available: persistentvolumeclaim \"	3
▶ 2	"kube-system/coredns-6ff9c46cd8-krds7"	"no nodes available to schedule pods"	2
▶ 3	"monitoring/prometheus-server-58469d8b-w9wk7"	"running PreBind plugin \"	1
▶ 4	"kube-system/coredns-6ff9c46cd8-knzmf"	"0/1 nodes are available: 1 node(s) had untolerated taint {node.cloudprovider.kubernetes.io/uninitialized: true}. preemption: 0/1 nodes are available: 1 Preemption is not helpful for scheduling.."	1
▶ 5	"kube-system/coredns-6ff9c46cd8-krds7"	"0/1 nodes are available: 1 node(s) had untolerated taint {node.cloudprovider.kubernetes.io/uninitialized: true}. preemption: 0/1 nodes are available: 1 Preemption is not helpful for scheduling.."	1

> **Note**
>
> El registro de auditoría debe estar activado en el plano de control para habilitar esta función. También es una buena práctica limitar la retención de registros para no aumentar los costes con el tiempo de forma innecesaria. A continuación se muestra un ejemplo de cómo activar todas las funciones de registro con la herramienta EKSCTL.

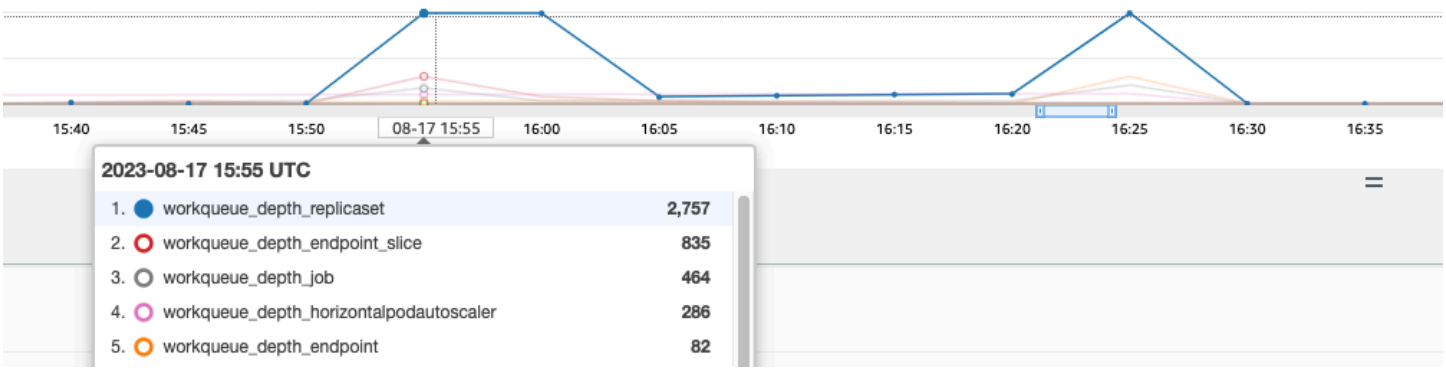
```
cloudWatch:
  clusterLogging:
    enableTypes: ["*"]
    logRetentionInDays: 10
```

Kube Controller Manager

Kube Controller Manager, como todos los demás controladores, tiene límites en cuanto al número de operaciones que puede realizar a la vez. Repasemos cuáles son algunos de esos indicadores observando una configuración de KOPS en la que podemos establecer estos parámetros.

```
kubeControllerManager:
  concurrentEndpointSyncs: 5
  concurrentReplicasetSyncs: 5
  concurrentNamespaceSyncs: 10
  concurrentServiceaccountTokenSyncs: 5
  concurrentServiceSyncs: 5
  concurrentResourceQuotaSyncs: 5
  concurrentGcSyncs: 20
  kubeAPIBurst: 30
  kubeAPIQPS: "20"
```

Estos controladores tienen colas que se llenan en momentos de alta rotación en un clúster. En este caso, vemos que el controlador del conjunto de réplicas tiene una gran cantidad de trabajo pendiente.



Tenemos dos maneras diferentes de abordar esta situación. Si se ejecuta de forma autogestionada, podríamos simplemente aumentar las rutinas simultáneas. Sin embargo, esto repercutiría en el etcd al procesar más datos en el KCM. La otra opción sería reducir la cantidad de objetos del conjunto de réplicas que se utilizan `.spec.revisionHistoryLimit` en la implementación para reducir la cantidad de objetos del conjunto de réplicas que podemos revertir, reduciendo así la presión sobre este controlador.

```
spec:
```

```
  revisionHistoryLimit: 2
```

Otras funciones de Kubernetes se pueden ajustar o desactivar para reducir la presión en los sistemas con una alta tasa de rotación. Por ejemplo, si la aplicación de nuestros pods no necesita comunicarse directamente con la API k8s, desactivar el secreto proyectado en esos pods disminuiría la carga. ServiceaccountTokenSyncs Si es posible, esta es la forma más conveniente de abordar estos problemas.

```
kind: Pod
spec:
  automountServiceAccountToken: false
```

En los sistemas en los que no podemos acceder a las métricas, podemos volver a revisar los registros para detectar posibles discrepancias. Si quisiéramos ver el número de solicitudes que se están procesando por controlador o a nivel agregado, utilizaríamos la siguiente consulta de CloudWatch Logs Insights.

Volumen total procesado por la KCM

```
# Query to count API qps coming from kube-controller-manager, split by controller type.
# If you're seeing values close to 20/sec for any particular controller, it's most
 likely seeing client-side API throttling.
fields @timestamp, @logStream, @message
| filter @logStream like /kube-apiserver-audit/
| filter userAgent like /kube-controller-manager/
# Exclude lease-related calls (not counted under kcm qps)
| filter requestURI not like "apis/coordination.k8s.io/v1/namespaces/kube-system/
leases/kube-controller-manager"
# Exclude API discovery calls (not counted under kcm qps)
| filter requestURI not like "?timeout=32s"
# Exclude watch calls (not counted under kcm qps)
| filter verb != "watch"
# If you want to get counts of API calls coming from a specific controller, uncomment
 the appropriate line below:
# | filter user.username like "system:serviceaccount:kube-system:job-controller"
# | filter user.username like "system:serviceaccount:kube-system:cronjob-controller"
# | filter user.username like "system:serviceaccount:kube-system:deployment-controller"
# | filter user.username like "system:serviceaccount:kube-system:replicaset-controller"
# | filter user.username like "system:serviceaccount:kube-system:horizontal-pod-
autoscaler"
```

```
# | filter user.username like "system:serviceaccount:kube-system:persistent-volume-
binder"
# | filter user.username like "system:serviceaccount:kube-system:endpointslice-
controller"
# | filter user.username like "system:serviceaccount:kube-system:endpoint-controller"
# | filter user.username like "system:serviceaccount:kube-system:generic-garbage-
controller"
| stats count(*) as count by user.username
| sort count desc
```

Lo más importante es analizar los problemas de escalabilidad y analizar cada paso del proceso (API, programador, KCM, etc.) antes de pasar a la fase detallada de solución de problemas. A menudo, en la fase de producción, es necesario realizar ajustes en más de una parte de Kubernetes para permitir que el sistema funcione al máximo. Es fácil solucionar problemas de forma inadvertida, lo que no es más que un síntoma (como el tiempo de espera de un nodo) de un cuello de botella mucho más grande.

ETC.D.

etcd utiliza un archivo mapeado en memoria para almacenar pares clave-valor de manera eficiente. Existe un mecanismo de protección para establecer el tamaño de este espacio de memoria disponible, que normalmente se establece en los límites de 2, 4 y 8 GB. Si hay menos objetos en la base de datos, es menos necesario limpiar etcd cuando se actualizan los objetos y se deben limpiar las versiones anteriores. Este proceso de limpieza de las versiones antiguas de un objeto se denomina compactación. Tras una serie de operaciones de compactación, hay un proceso posterior que recupera el espacio útil, denominado desfragmentación, que se produce por encima de un determinado umbral o siguiendo un calendario fijo.

Hay un par de medidas relacionadas con el usuario que podemos utilizar para limitar el número de objetos en Kubernetes y reducir así el impacto del proceso de compactación y desfragmentación. Por ejemplo, Helm mantiene un nivel alto. `revisionHistoryLimit` Esto mantiene los objetos más antiguos, como ReplicaSets los del sistema, para poder revertirlos. Al establecer los límites del historial en 2, podemos reducir la cantidad de objetos (por ejemplo ReplicaSets) de diez a dos, lo que a su vez supondría una menor carga para el sistema.

```
apiVersion: apps/v1
kind: Deployment
spec:
  revisionHistoryLimit: 2
```

Desde el punto de vista de la supervisión, si los picos de latencia del sistema se producen siguiendo un patrón establecido separados por horas, puede resultar útil comprobar si este proceso de desfragmentación es la causa. Esto lo podemos comprobar mediante CloudWatch los registros.

Si quieres ver los start/end tiempos de desfragmentación, usa la siguiente consulta:

```
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

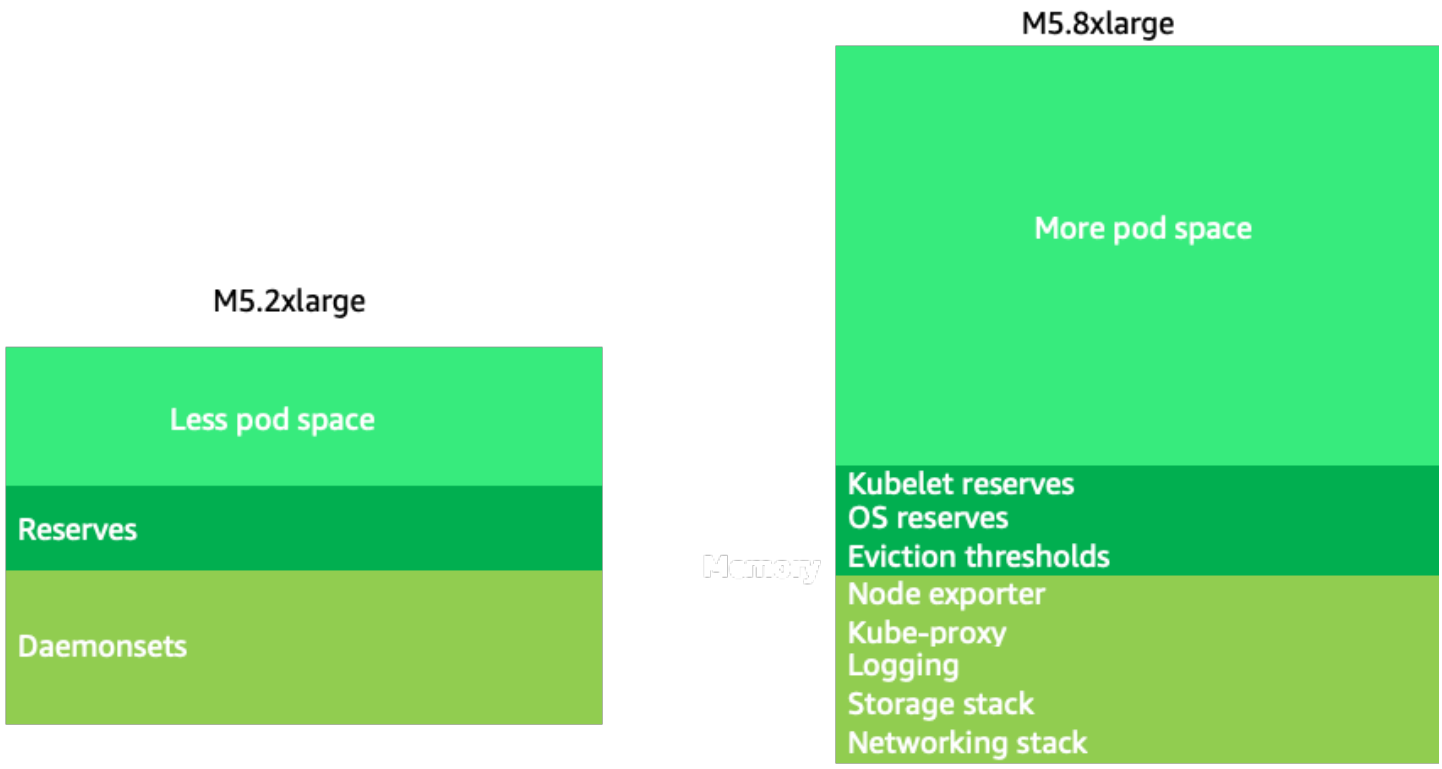
Eficiencia de nodos y cargas de trabajo

Ser eficientes con nuestras cargas de trabajo y nodos reduce al complexity/cost mismo tiempo que aumenta el rendimiento y la escalabilidad. Hay muchos factores que se deben tener en cuenta a la hora de planificar esta eficiencia, y es más fácil pensar en términos de compensaciones en lugar de establecer una configuración de mejores prácticas para cada función. Analicemos estas ventajas y desventajas en profundidad en la siguiente sección.

Selección de nodos

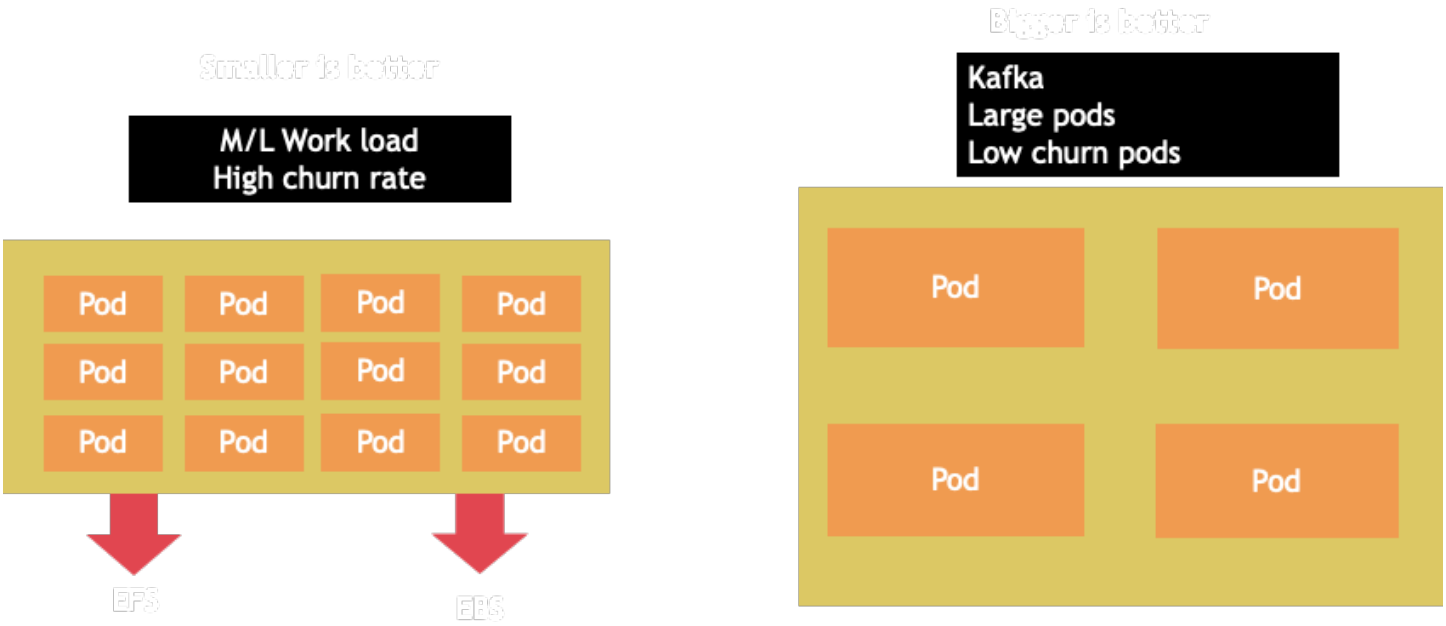
Si utilizamos tamaños de nodo un poco más grandes (de 4 a 12 veces más), se aumenta el espacio disponible para ejecutar los pods, ya que reduce el porcentaje del nodo que se utiliza para «sobrecargas», por ejemplo, DaemonSetsy se reserva para los componentes del sistema. En el siguiente diagrama, vemos la diferencia entre el espacio útil de un sistema de 2 x grandes y el de uno de 8 x, con un número moderado de. DaemonSets

> **Note**
>
> Dado que, por regla general, el k8s escala horizontalmente, para la mayoría de las aplicaciones no tiene sentido considerar el impacto en el rendimiento del NUMA como tamaño de los nodos, por lo que se recomienda un rango inferior a ese tamaño de nodo.



Los tamaños de nodos grandes nos permiten tener un mayor porcentaje de espacio utilizable por nodo. Sin embargo, este modelo se puede llevar al extremo si se empaqueta el nodo con tantos pods que provoque errores o sature el nodo. Supervisar la saturación de los nodos es clave para utilizar correctamente nodos de mayor tamaño.

La selección de nodos rara vez es una one-size-fits-all propuesta. A menudo, lo mejor es dividir las cargas de trabajo con tasas de rotación muy diferentes en diferentes grupos de nodos. Las cargas de trabajo de lotes pequeños con una alta tasa de abandono se gestionarían mejor con la familia de instancias 4xlarge, mientras que una aplicación a gran escala, como Kafka, que ocupa 8 vCPU y tiene una baja tasa de abandono, se gestionaría mejor con la familia 12xlarge.



> **Note**
>
> Otro factor a tener en cuenta cuando los nodos son muy grandes es que los CGROUPS no ocultan la cantidad total de vCPU de la aplicación contenerizada. Los tiempos de ejecución dinámicos suelen generar un número no intencionado de subprocesos del sistema operativo, lo que crea una latencia difícil de solucionar. Para estas aplicaciones, se recomienda fijar la CPU. Para una exploración más profunda del tema, consulte el siguiente https:// www.youtube.com video/watch? v= _ NqtfDy KAqg

Empaquetado en contenedores de nodos

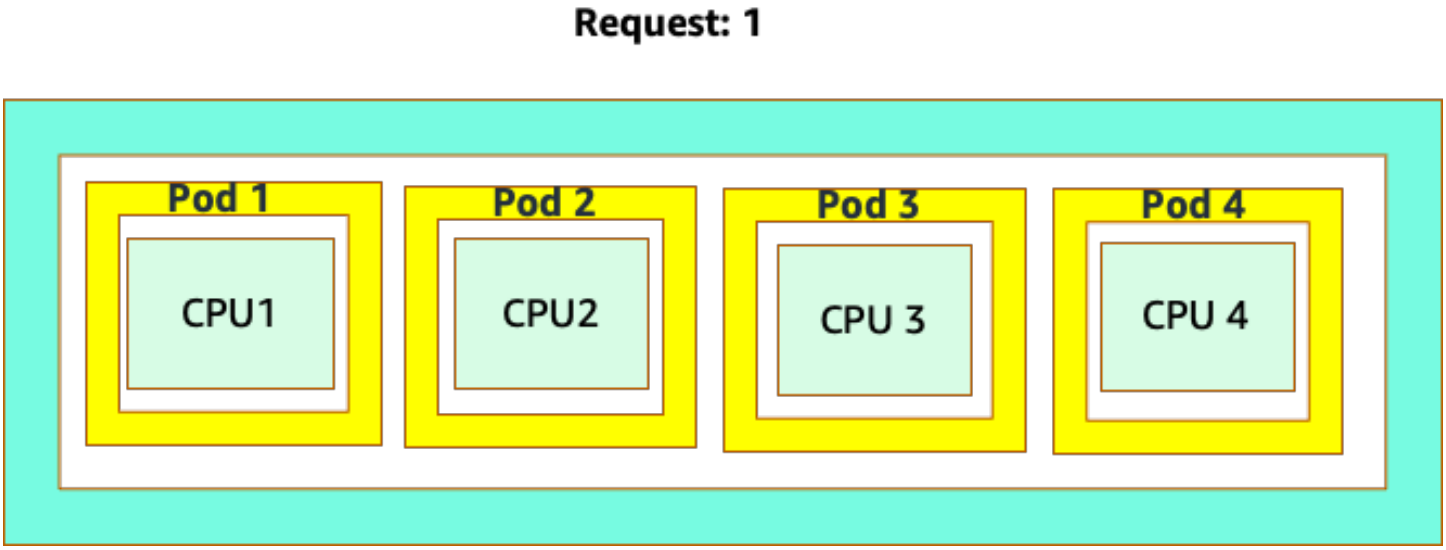
Reglas de Kubernetes frente a Linux

Hay dos conjuntos de reglas que debemos tener en cuenta al tratar las cargas de trabajo en Kubernetes. Las reglas del programador de Kubernetes, que usa el valor de la solicitud para programar los pods en un nodo, y luego lo que ocurre una vez programado el pod, que es el ámbito de Linux, no de Kubernetes.

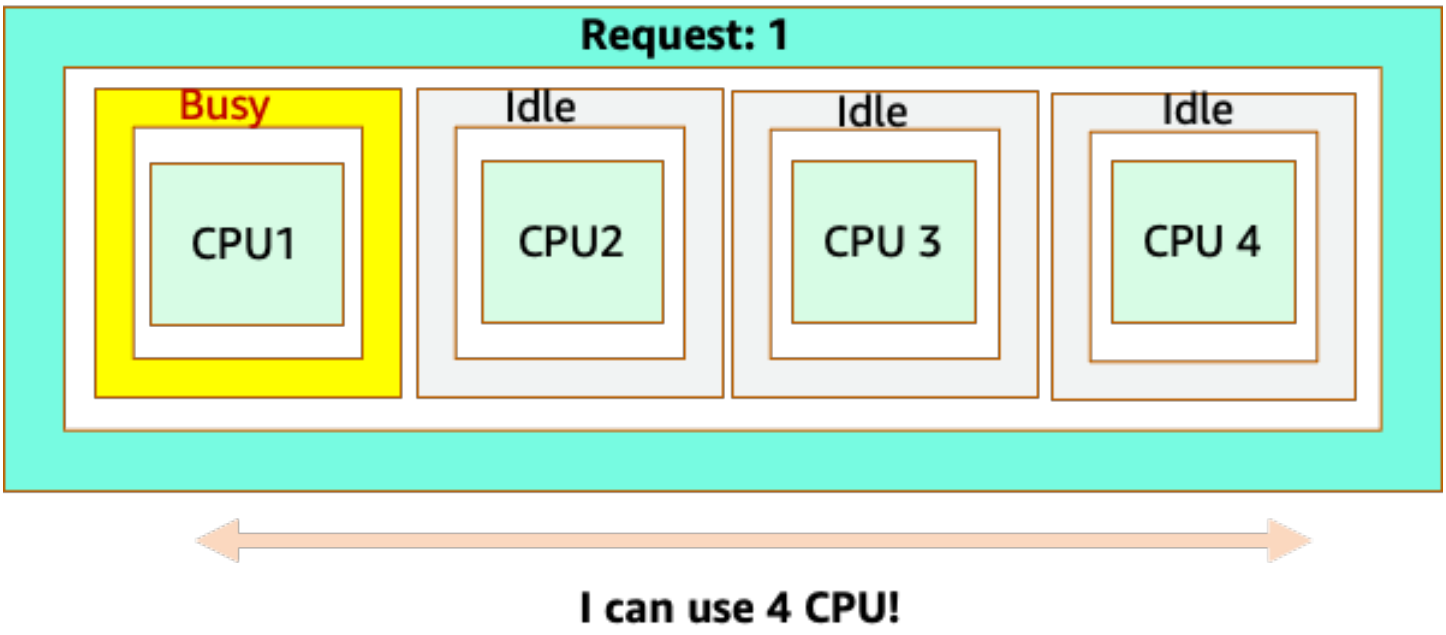
Una vez finalizado el programador de Kubernetes, se aplica un nuevo conjunto de reglas: el Programador Completamente Justo de Linux (CFS). La conclusión clave es que Linux CFS no tiene el concepto de núcleo. Analizaremos por qué pensar en núcleos puede provocar problemas importantes a la hora de optimizar las cargas de trabajo para adaptarlas a mayor escala.

Pensando en núcleos

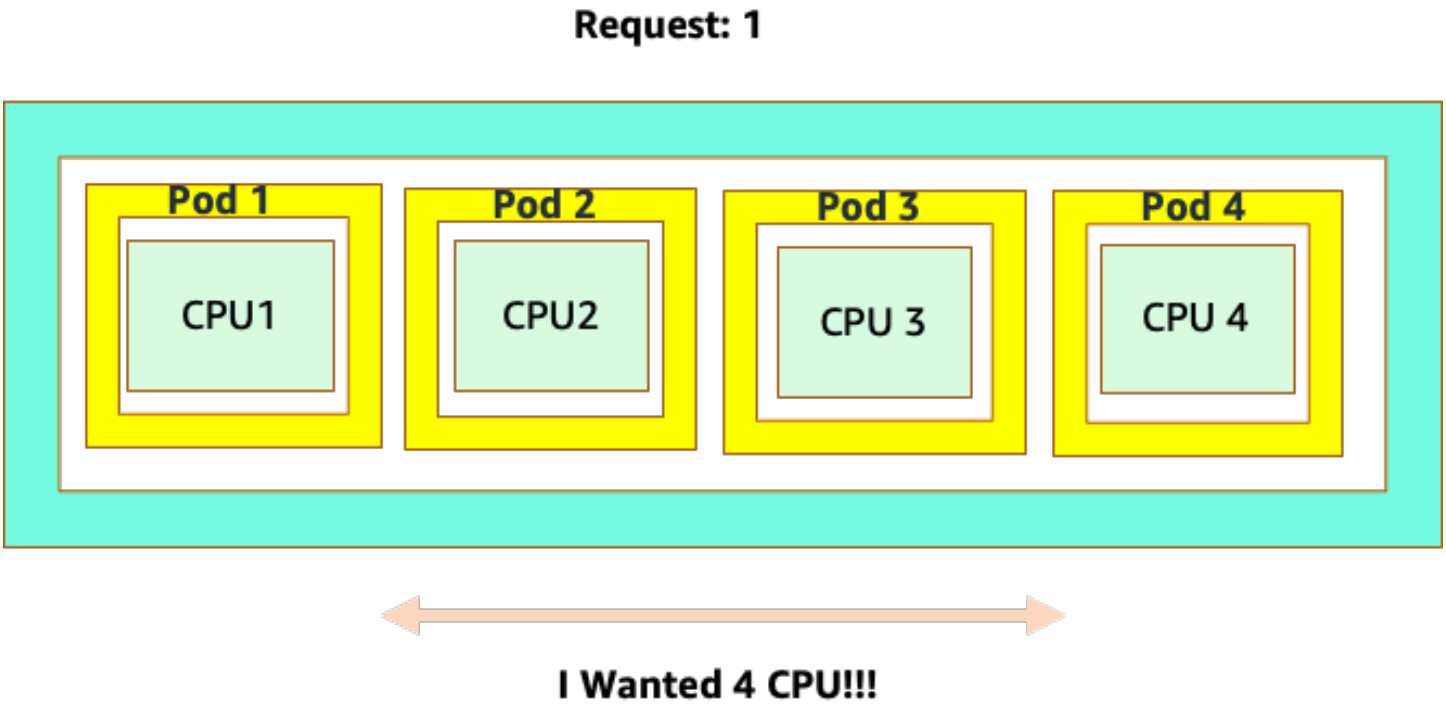
La confusión comienza porque el programador de Kubernetes tiene el concepto de núcleos. Desde la perspectiva de un programador de Kubernetes, si analizáramos un nodo con 4 pods NGINX, cada uno con una solicitud de un conjunto de núcleos, el nodo tendría este aspecto.



Sin embargo, hagamos un experimento mental sobre qué tan diferente se ve esto desde la perspectiva del CFS de Linux. Lo más importante que hay que recordar al utilizar el sistema CFS de Linux es que los contenedores ocupados (CGROUPS) son los únicos contenedores que se tienen en cuenta para el sistema compartido. En este caso, solo el primer contenedor está ocupado, por lo que se permite usar los 4 núcleos del nodo.



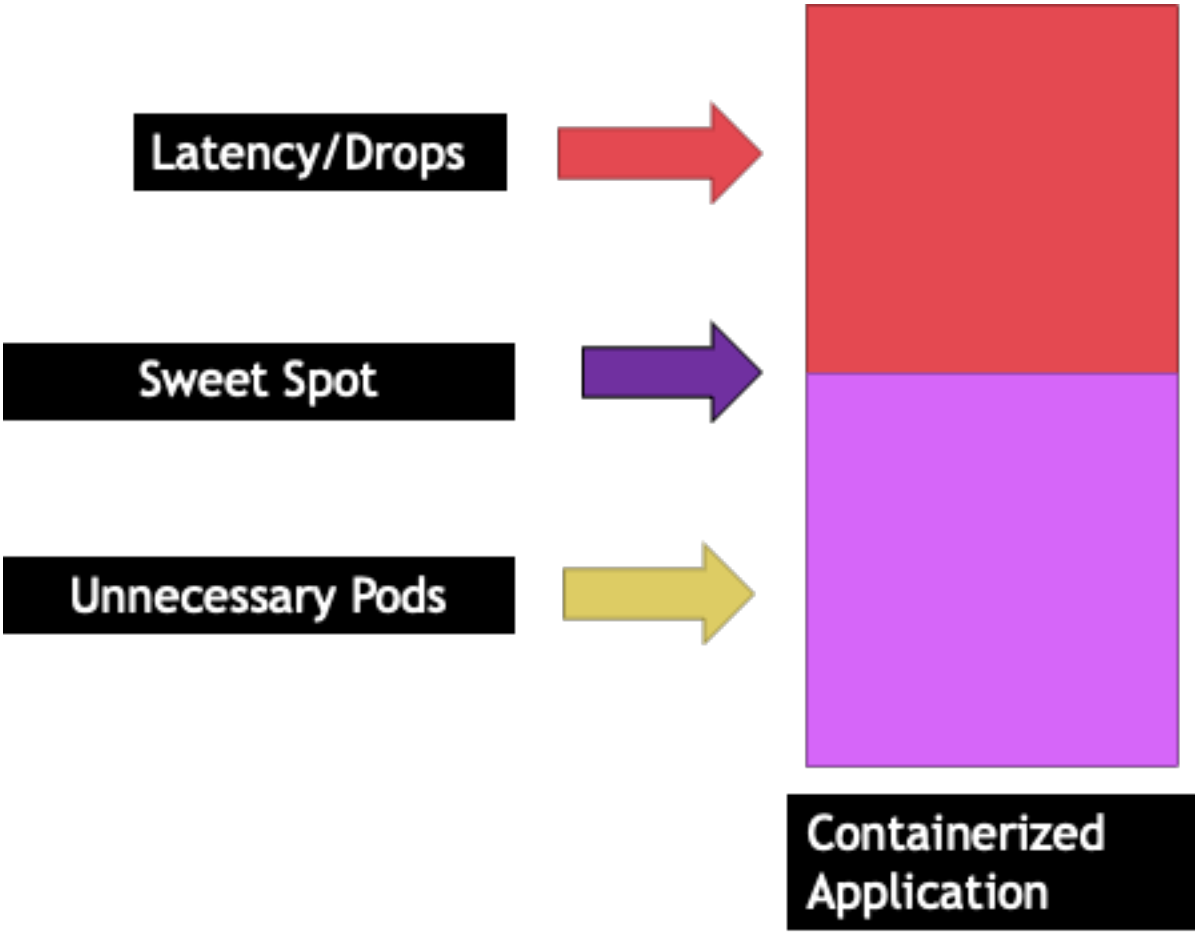
¿Por qué importa esto? Supongamos que realizamos nuestras pruebas de rendimiento en un clúster de desarrollo en el que una aplicación NGINX era el único contenedor ocupado de ese nodo. Cuando pasamos la aplicación a producción, ocurre lo siguiente: la aplicación NGINX necesita 4 vCPU de recursos; sin embargo, dado que todos los demás pods del nodo están ocupados, el rendimiento de nuestra aplicación se ve limitado.



Esta situación nos llevaría a añadir más contenedores de forma innecesaria porque no permitíamos que nuestras aplicaciones se adaptaran a su «punto óptimo». Exploremos este importante concepto de a con un poco más de `"sweet spot"` detalle.

Aplicación: tamaño correcto

Cada aplicación tiene un punto determinado en el que no puede soportar más tráfico. Superar este punto puede aumentar los tiempos de procesamiento e incluso reducir el tráfico si se supera con creces este punto. Esto se conoce como punto de saturación de la aplicación. Para evitar problemas de escalado, debemos intentar escalar la aplicación antes de que alcance su punto de saturación. Llamemos a este punto el punto óptimo.



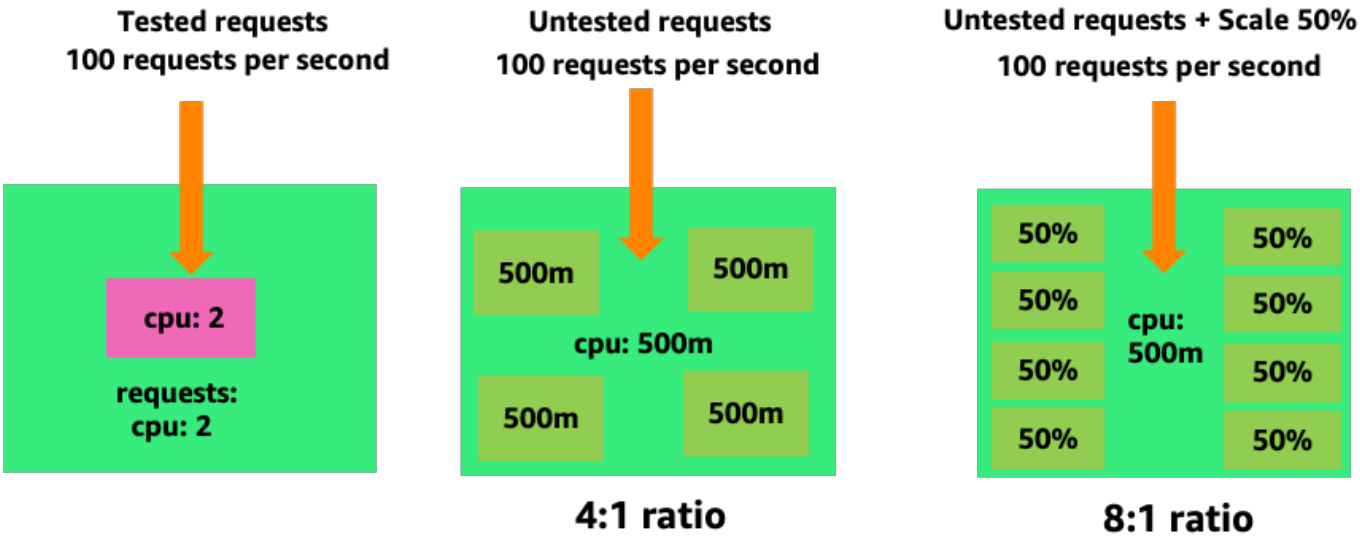
Tenemos que probar cada una de nuestras aplicaciones para entender su punto óptimo. En este sentido, no habrá una guía universal, ya que cada aplicación es diferente. Durante estas pruebas, intentamos entender cuál es la mejor métrica que muestre el punto de saturación de nuestras aplicaciones. A menudo, las métricas de uso se utilizan para indicar que una aplicación está saturada, pero esto puede provocar rápidamente problemas de escalado (analizaremos este tema en detalle en una sección posterior). Una vez que tengamos este «punto óptimo», podemos usarlo para escalar nuestras cargas de trabajo de manera eficiente.

Por el contrario, ¿qué pasaría si ampliáramos la escala mucho antes del punto óptimo y creáramos cápsulas innecesarias? Exploremos eso en la siguiente sección.

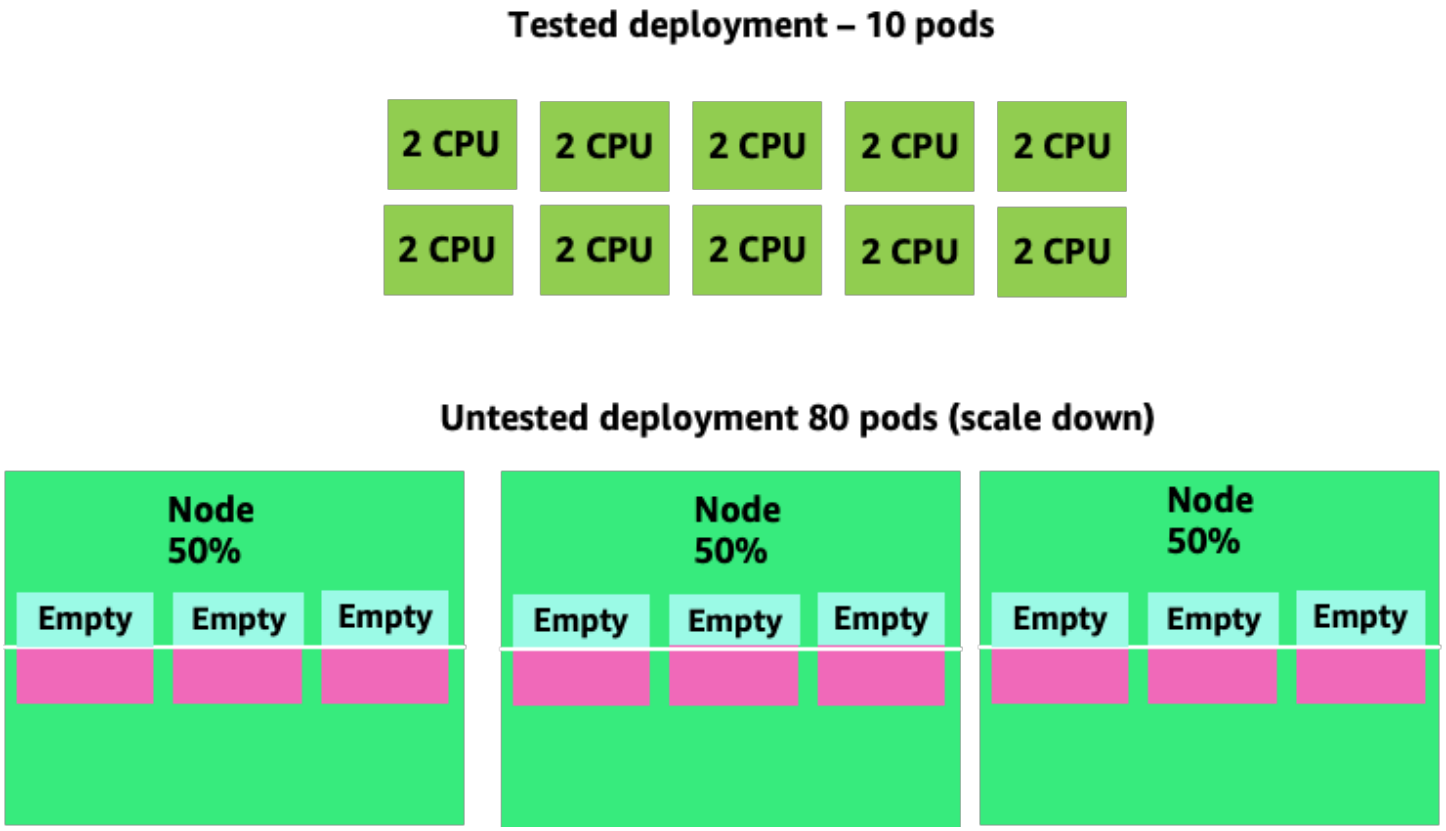
Expansión de vainas

Para ver cómo la creación de cápsulas innecesarias puede salirse de control rápidamente, veamos el primer ejemplo de la izquierda. La escala vertical correcta de este contenedor requiere aproximadamente dos V de CPUs uso cuando se gestionan 100 solicitudes por segundo. Sin embargo, si aprovisionáramos menos del valor de las solicitudes y las estableciéramos en medio

núcleo, ahora necesitaríamos 4 módulos por cada uno que realmente necesitáramos. Para agravar aún más este problema, si nuestro HPA estuviera configurado de forma predeterminada con un 50% de CPU, esos módulos se escalarían medio vacíos, lo que generaría una proporción de 8:1.

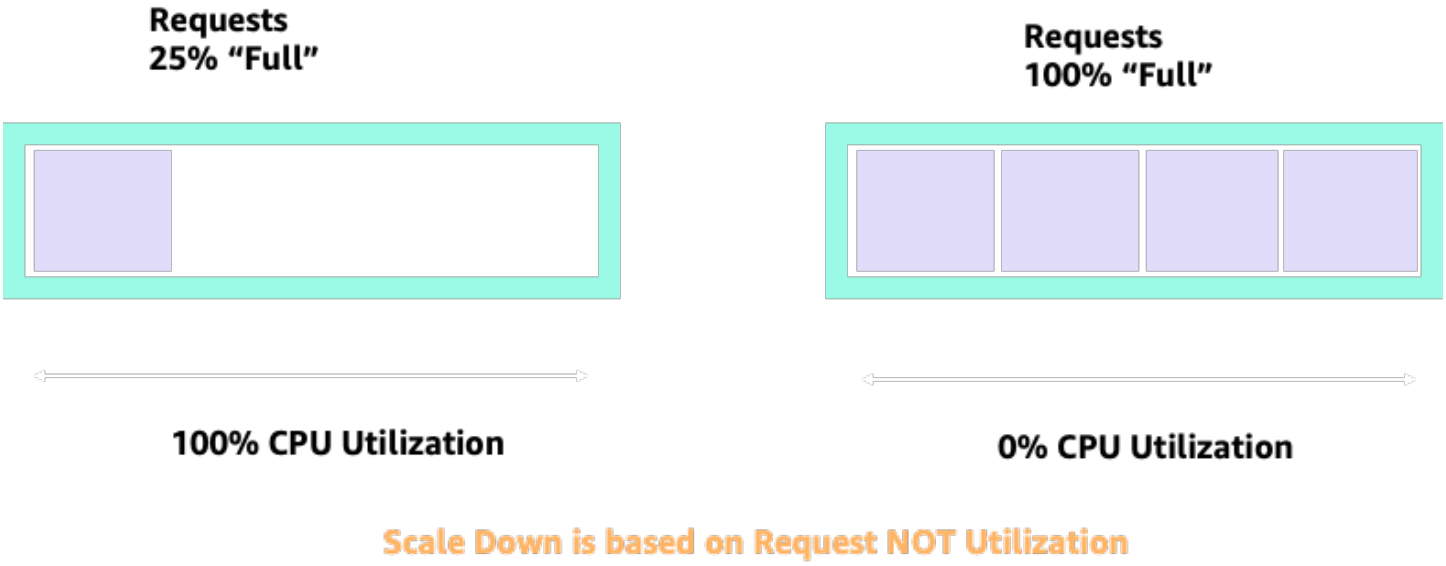


Al ampliar este problema, podemos ver rápidamente cómo se nos puede ir de las manos. Un despliegue de diez módulos con una configuración incorrecta podría llegar rápidamente a 80 módulos y disponer de la infraestructura adicional necesaria para su funcionamiento.



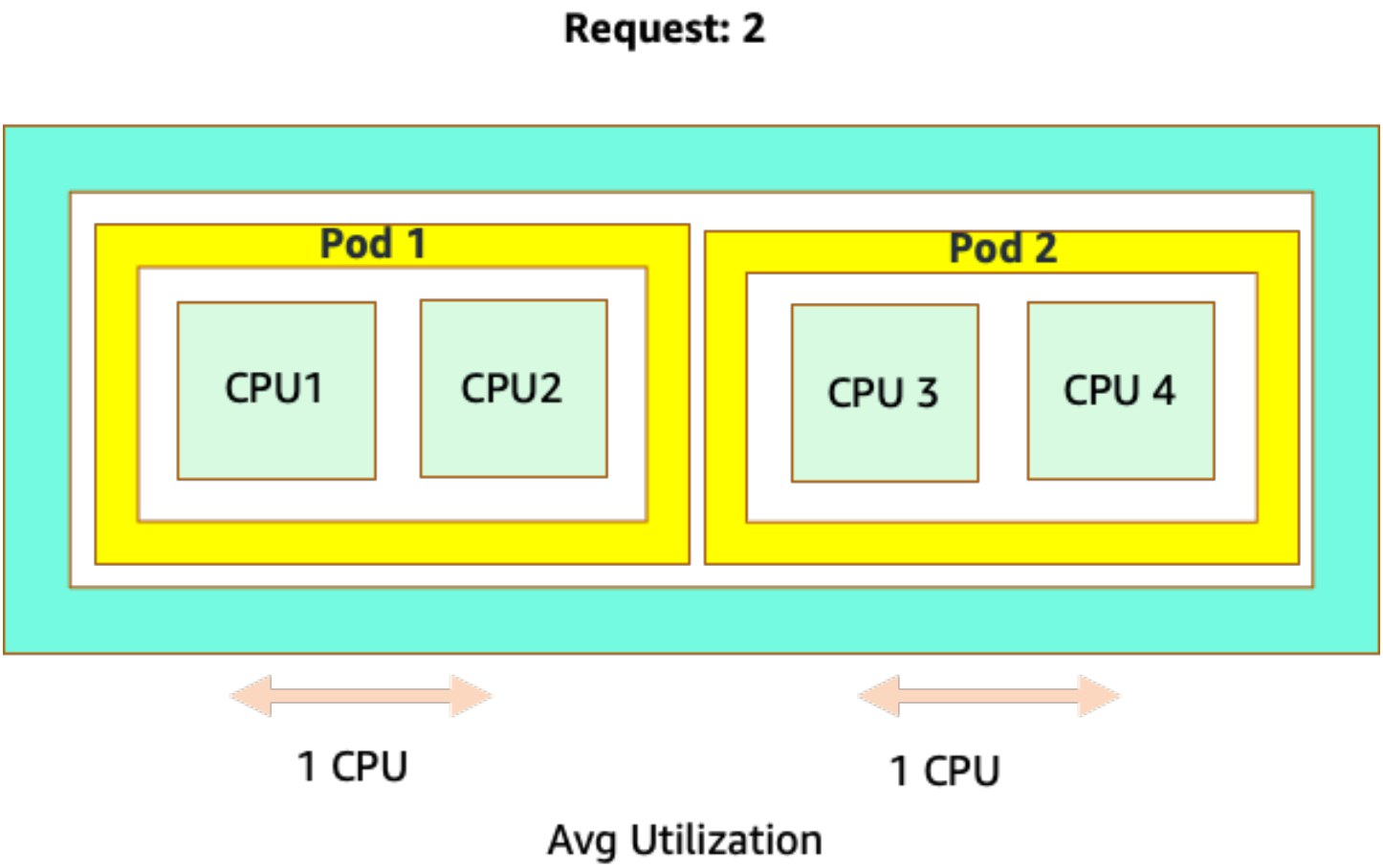
Ahora que entendemos el impacto de no permitir que las aplicaciones funcionen en su punto óptimo, volvamos al nivel de los nodos y preguntemos por qué es tan importante esta diferencia entre el programador de Kubernetes y el CFS de Linux.

Al escalar hacia arriba y hacia abajo con HPA, podemos tener un escenario en el que tengamos mucho espacio para asignar más módulos. Sería una mala decisión porque el nodo que se muestra a la izquierda ya utiliza el 100% de la CPU. En un escenario poco realista pero teóricamente posible, podríamos llegar al otro extremo, en el que nuestro nodo esté completamente lleno y, sin embargo, nuestra utilización de la CPU sea nula.

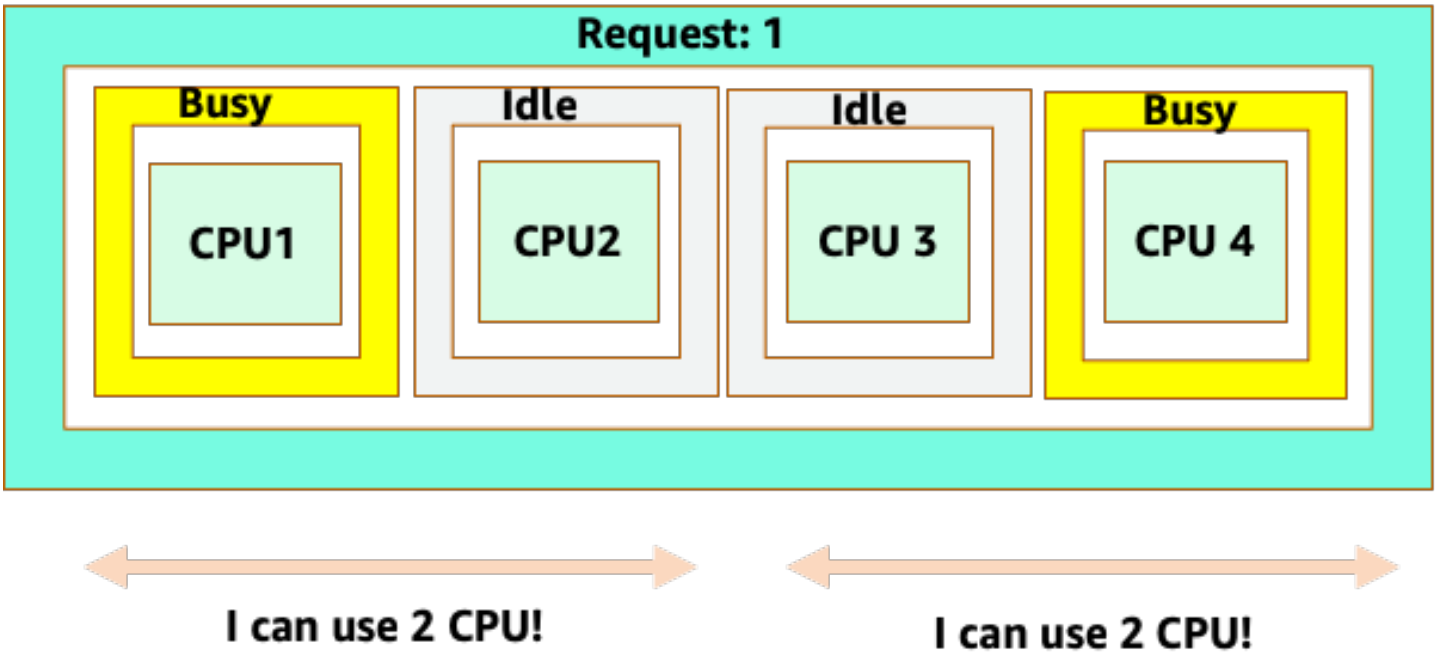


Configuración de solicitudes

Sería tentador fijar la solicitud en el valor ideal para esa aplicación, pero esto provocaría ineficiencias, como se muestra en el siguiente diagrama. En este caso, hemos establecido el valor de la solicitud en 2 vCPU; sin embargo, la utilización media de estos pods solo ejecuta 1 CPU la mayoría de las veces. Esta configuración provocaría que desperdiciáramos el 50% de nuestros ciclos de CPU, lo que sería inaceptable.



Esto nos lleva a la compleja respuesta al problema. La utilización de los contenedores no se puede pensar en el vacío; hay que tener en cuenta las demás aplicaciones que se ejecutan en el nodo. En el siguiente ejemplo, los contenedores que son de naturaleza explosiva se mezclan con dos contenedores de bajo uso de la CPU que pueden tener limitaciones de memoria. De esta forma, permitimos que los contenedores lleguen a su punto óptimo sin sobrecargar el nodo.



Lo importante que debemos extraer de todo esto es que utilizar el concepto de núcleos del programador de Kubernetes para comprender el rendimiento de los contenedores de Linux puede llevar a una mala toma de decisiones, ya que no están relacionados entre sí.

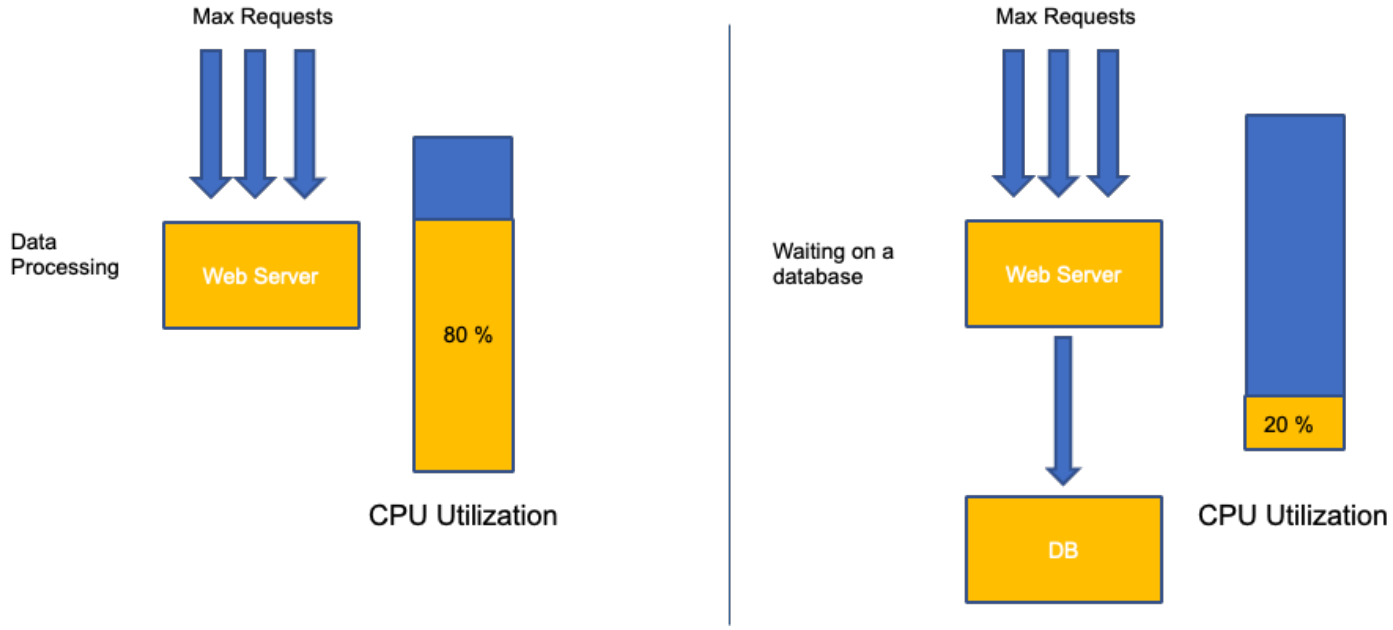
> **Note**
>
> El CFS de Linux tiene sus puntos fuertes. Esto es especialmente cierto en el caso de las cargas de trabajo I/O basadas. Sin embargo, si su aplicación utiliza núcleos completos sin sidecars y no tiene I/O requisitos, la fijación de la CPU puede eliminar una gran parte de la complejidad de este proceso, por lo que se recomienda tener en cuenta estas advertencias.

Utilización frente a saturación

Un error común en el escalado de aplicaciones es utilizar únicamente el uso de la CPU para la métrica de escalado. En aplicaciones complejas, esto casi siempre es un mal indicador de que una aplicación está realmente saturada de solicitudes. En el ejemplo de la izquierda, vemos que todas nuestras solicitudes están llegando al servidor web, por lo que el uso de la CPU va registrando un buen ritmo con la saturación.

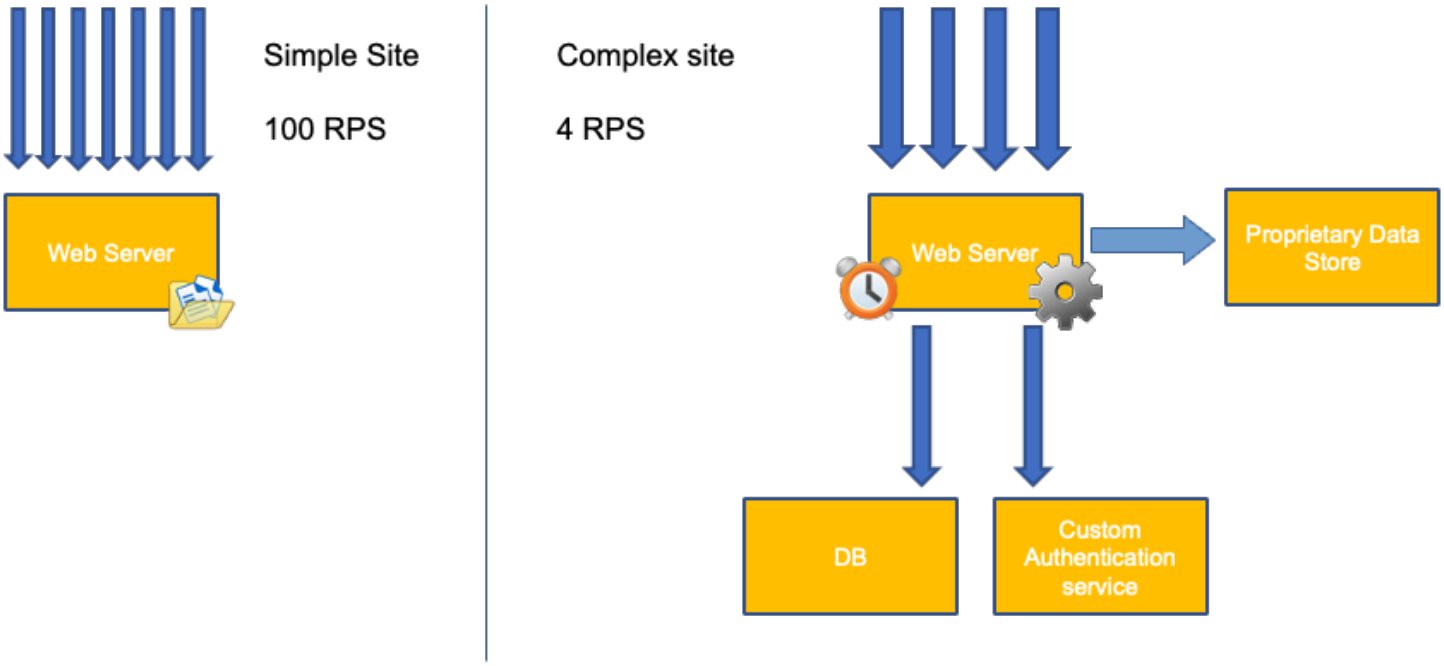
En las aplicaciones del mundo real, es probable que algunas de esas solicitudes sean atendidas por una capa de base de datos o una capa de autenticación, etc. En este caso más común, observe que

la CPU no realiza el seguimiento debido a la saturación, ya que la solicitud está siendo atendida por otras entidades. En este caso, la CPU es un indicador muy pobre de saturación.



El uso de una métrica incorrecta en el rendimiento de las aplicaciones es la principal razón por la que se produce un escalado innecesario e impredecible en Kubernetes. Debes tener mucho cuidado al elegir la métrica de saturación correcta para el tipo de aplicación que utilices. Es importante tener en cuenta que no existe una recomendación única que se pueda dar a todos. Según el idioma utilizado y el tipo de aplicación en cuestión, existe un conjunto diverso de métricas de saturación.

Podríamos pensar que este problema se debe únicamente a la utilización de la CPU, sin embargo, otras métricas comunes, como la solicitud por segundo, también pueden tener exactamente el mismo problema que el descrito anteriormente. Tenga en cuenta que la solicitud también puede ir a capas de bases de datos, capas de autenticación, y no ser atendida directamente por nuestro servidor web, por lo que no es una métrica adecuada para medir la verdadera saturación del propio servidor web.



Lamentablemente, no hay respuestas fáciles a la hora de elegir la métrica de saturación correcta. Estas son algunas pautas a tener en cuenta:

- Conozca el entorno de ejecución de su idioma: los lenguajes con varios subprocesos del sistema operativo reaccionarán de forma diferente que las aplicaciones con un solo subproceso, lo que afectará de forma diferente al nodo.
- Conozca la escala vertical correcta: ¿cuánto búfer desea en la escala vertical de sus aplicaciones antes de escalar un nuevo pod?
- ¿Qué métricas reflejan realmente la saturación de su aplicación? La métrica de saturación de un productor de Kafka sería muy diferente a la de una aplicación web compleja.
- ¿Cómo se afectan entre sí todas las demás aplicaciones del nodo? El rendimiento de las aplicaciones no se realiza de forma aislada, sino que las demás cargas de trabajo del nodo tienen un impacto importante.

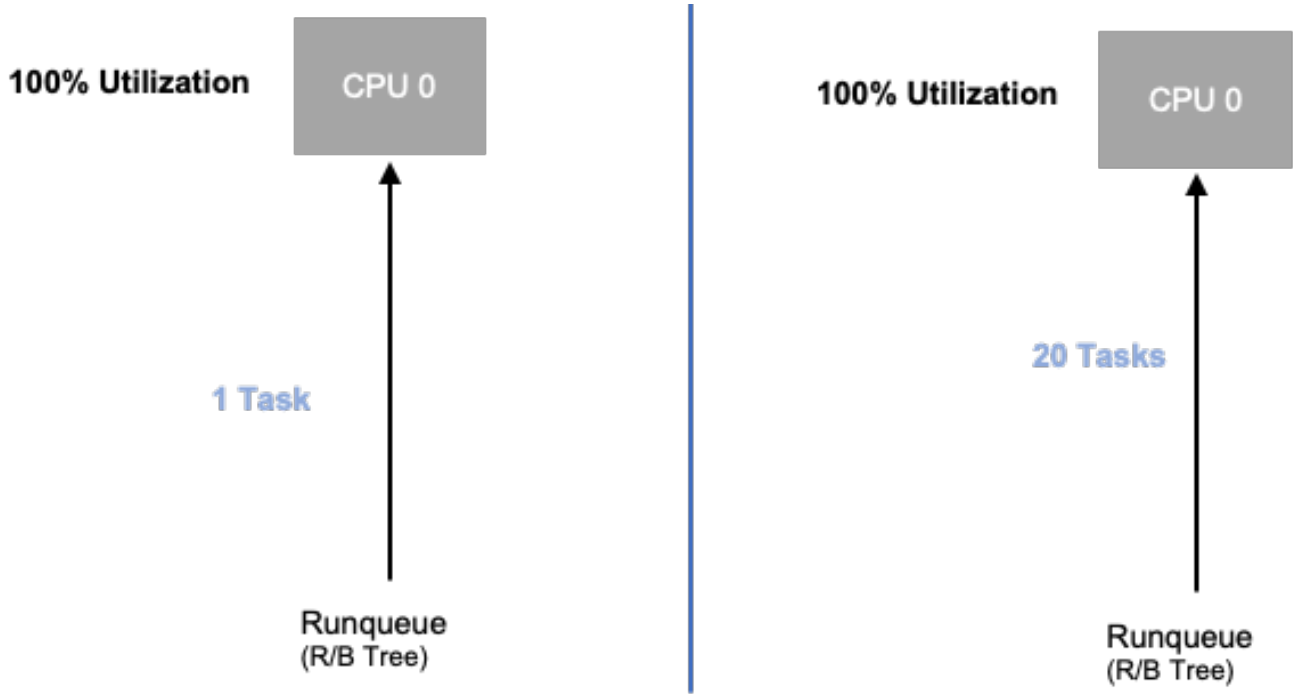
Para cerrar esta sección, sería fácil descartar lo anterior por ser demasiado complejo e innecesario. A menudo puede darse el caso de que tengamos un problema, pero desconozcamos la verdadera naturaleza del problema porque estamos utilizando métricas incorrectas. En la siguiente sección veremos cómo podría ocurrir eso.

Saturación de nodos

Ahora que hemos explorado la saturación de aplicaciones, veamos este mismo concepto desde el punto de vista de los nodos. Tomemos dos CPUs que se utilizan al 100% para ver la diferencia entre la utilización y la saturación.

La vCPU de la izquierda se utiliza al 100%, sin embargo, no hay otras tareas esperando a ejecutarse en esta vCPU, por lo que, en un sentido puramente teórico, esto es bastante eficiente. Mientras tanto, en el segundo ejemplo, tenemos 20 aplicaciones de un solo subproceso esperando ser procesadas por una vCPU. Las 20 aplicaciones ahora experimentarán algún tipo de latencia mientras esperan su turno para que la vCPU las procese. En otras palabras, la vCPU de la derecha está saturada.

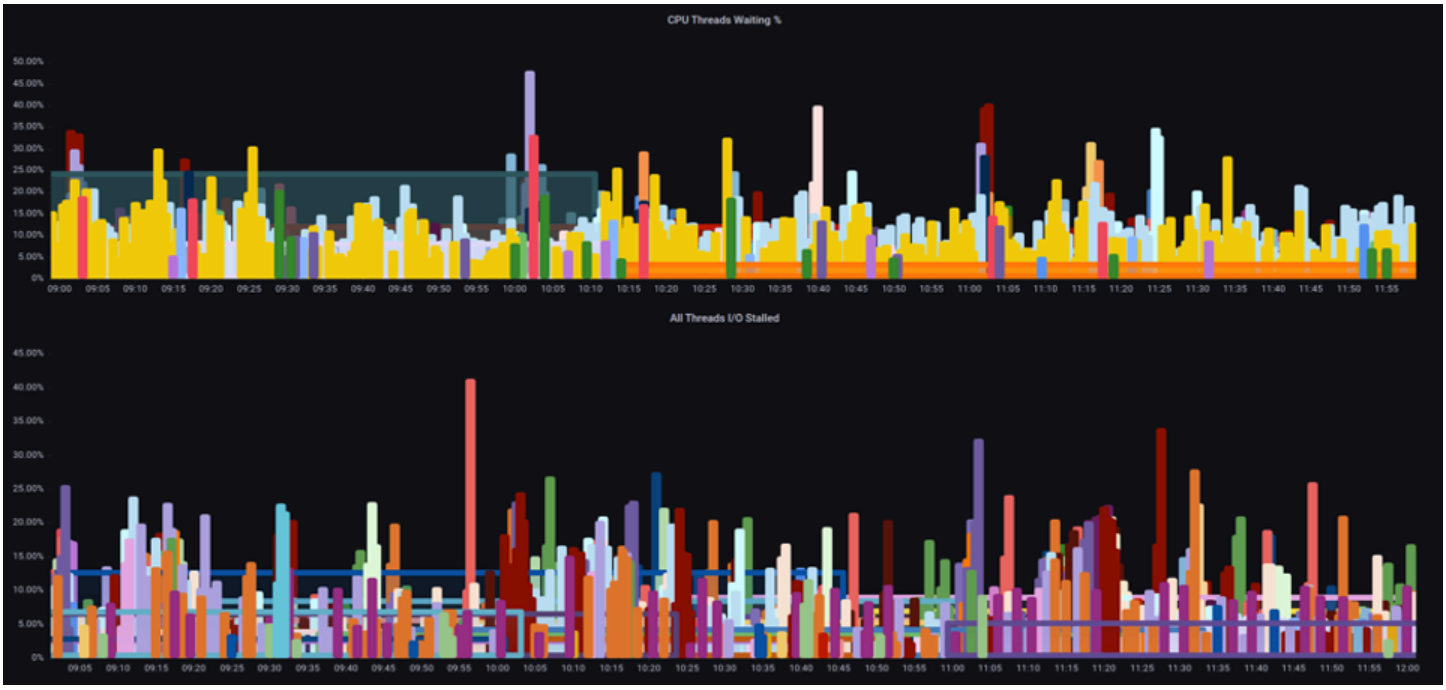
No solo no veríamos este problema si solo analizáramos la utilización, sino que podríamos atribuir esta latencia a algo no relacionado, como las redes, lo que nos llevaría por el camino equivocado.



Al aumentar el número total de pods que se ejecutan en un nodo en un momento dado, es importante ver las métricas de saturación, no solo las métricas de uso, ya que podemos pasar por alto fácilmente el hecho de que hemos sobresaturado un nodo. Para esta tarea, podemos utilizar las métricas de información sobre el estancamiento de presión, como se muestra en el siguiente gráfico.

ProMQL: E/S estancada

```
topk(3, ((irate(node_pressure_io_stalled_seconds_total[1m])) * 100))
```



> **Note**
>
> Para obtener más información sobre las métricas de bloqueo de presión, consulte https:// facebookmicrosites.github. io/psi/docs/overview*

Con estas métricas, podemos saber si los subprocesos están esperando en la CPU o incluso si todos los subprocesos de la caja están estancados esperando un recurso como la memoria o I/O. For example, we could see what percentage every thread on the instance was stalled waiting on I/O durante un período de 1 minuto.

```
topk(3, ((irate(node_pressure_io_stalled_seconds_total[1m])) * 100))
```

Con esta métrica, podemos ver en el gráfico anterior que todos los hilos de la caja estaban parados el 45% del tiempo esperando I/O en el punto más alto, lo que significa que estábamos desperdiciando todos esos ciclos de CPU en ese minuto. Entender lo que está sucediendo puede ayudarnos a recuperar una cantidad significativa de tiempo de vCPU, lo que hace que el escalado sea más eficiente.

HPA V2

Se recomienda utilizar la versión autoscaling/v2 de la API HPA. Las versiones anteriores de la API HPA podían tener problemas de escalado en algunos casos extremos. También se limitaba a que los módulos solo se duplicaran durante cada paso de escalado, lo que generaba problemas en las implementaciones pequeñas que necesitaban escalarse rápidamente.

AutoScaling/v2 nos permite tener más flexibilidad a la hora de incluir varios criterios a partir de los cuales escalar y, además, una gran flexibilidad a la hora de utilizar métricas personalizadas y externas (distintas de las métricas de K8).

Por ejemplo, podemos escalar en función del mayor de los tres valores (ver más abajo). Escalamos si la utilización media de todos los módulos supera el 50%, si las métricas personalizadas indican que los paquetes por segundo de entrada superan la media de 1000 o si el objeto de entrada supera las 10 000 solicitudes por segundo.

> **Note**
>
> Esto es solo para demostrar la flexibilidad de la API de autoscalamiento. Recomendamos evitar las reglas demasiado complejas que pueden ser difíciles de solucionar en producción.

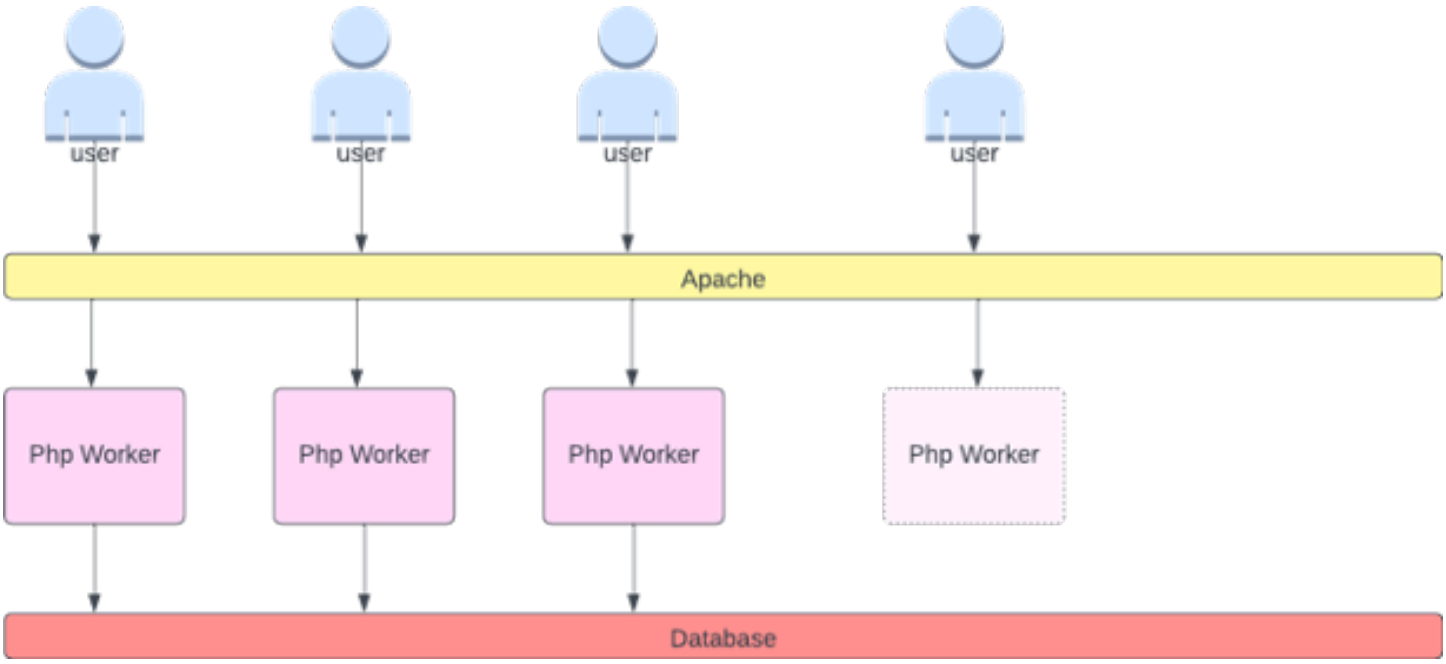
```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: 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
```

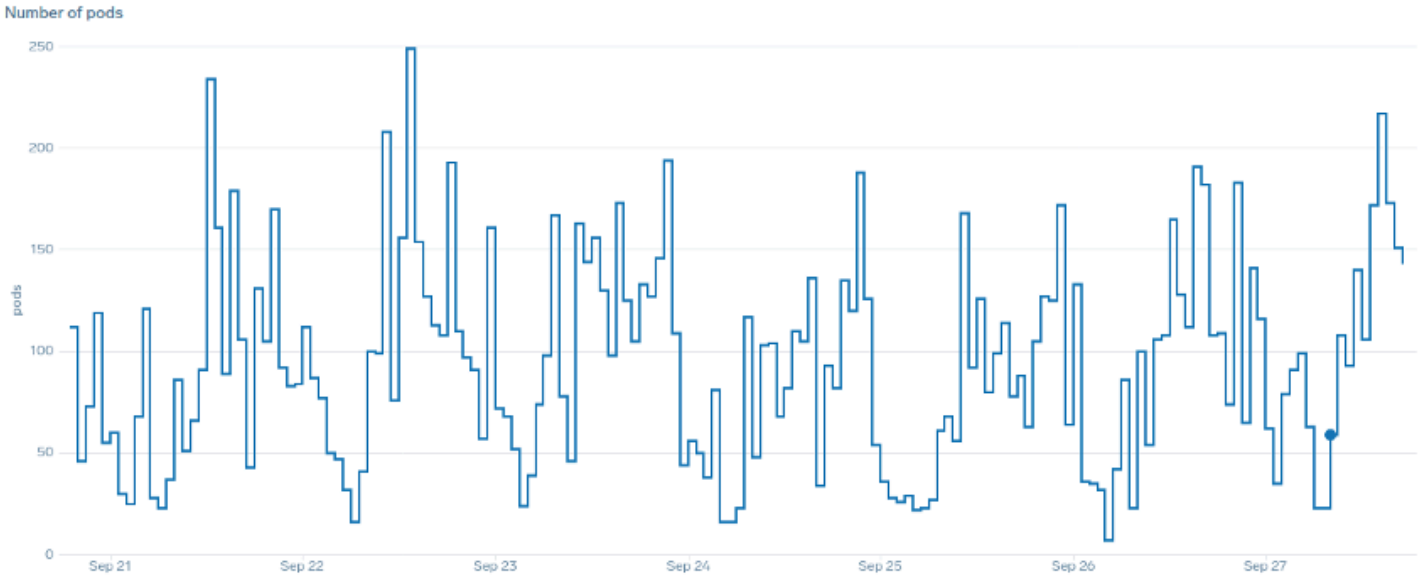
Sin embargo, nos dimos cuenta del peligro que supone utilizar este tipo de métricas para aplicaciones web complejas. En este caso, sería mejor utilizar una métrica personalizada o externa que refleje con precisión la saturación de nuestra aplicación en comparación con la utilización. HPAv2 lo permite al tener la capacidad de escalar de acuerdo con cualquier métrica; sin embargo, aún necesitamos encontrar esa métrica y exportarla a Kubernetes para su uso.

Por ejemplo, podemos ver el recuento de colas de subprocesos activos en Apache. Esto suele crear un perfil de escalado «más fluido» (pronto hablaremos de ese término). Si un subproceso está activo, no importa si ese subproceso está esperando en una capa de base de datos o atendiendo una solicitud de forma local; si se están utilizando todos los subprocesos de la aplicación, es un buen indicio de que la aplicación está saturada.

Podemos utilizar el agotamiento de los subprocesos como señal para crear un nuevo pod con un conjunto de subprocesos totalmente disponible. Esto también nos permite controlar el tamaño del búfer que queremos que absorba la aplicación en momentos de tráfico intenso. Por ejemplo, si tuviéramos un total de 10 subprocesos, escalarlos a 4 subprocesos utilizados frente a los 8 subprocesos utilizados tendría un gran impacto en el búfer del que disponemos al escalar la aplicación. Una configuración de 4 tendría sentido para una aplicación que necesita escalarse rápidamente con una carga pesada, mientras que una configuración de 8 sería más eficiente con nuestros recursos si tuviéramos tiempo suficiente para escalar, ya que el número de solicitudes aumenta de forma lenta o pronunciada con el tiempo.

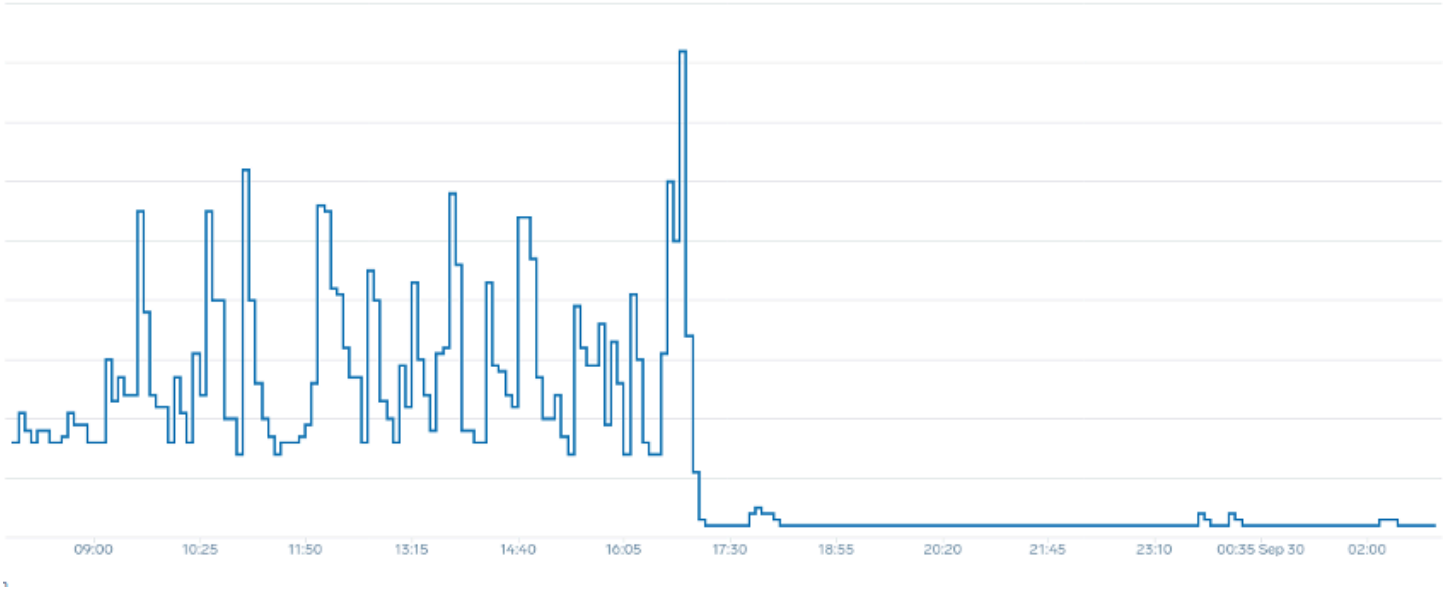


¿Qué queremos decir con el término «suave» cuando se trata de escalar? Observe el siguiente gráfico en el que utilizamos la CPU como métrica. Los módulos de esta implementación se incrementarán en poco tiempo, pasando de 50 módulos a 250 módulos, para luego volver a reducirse inmediatamente. Esto es muy ineficiente, el escalado es la principal causa de la pérdida de clientes en los clústeres.



Observe cómo, después de cambiar a una métrica que refleje el punto óptimo correcto de nuestra aplicación (parte media del gráfico), podemos escalar sin problemas. Nuestro escalado ahora es eficiente y nuestros módulos pueden ampliarse completamente con el margen de maniobra que les proporcionábamos al ajustar la configuración de las solicitudes. Ahora, un grupo más pequeño de

cápsulas está haciendo el trabajo que cientos de cápsulas hacían antes. Los datos del mundo real muestran que este es el factor número uno en la escalabilidad de los clústeres de Kubernetes.

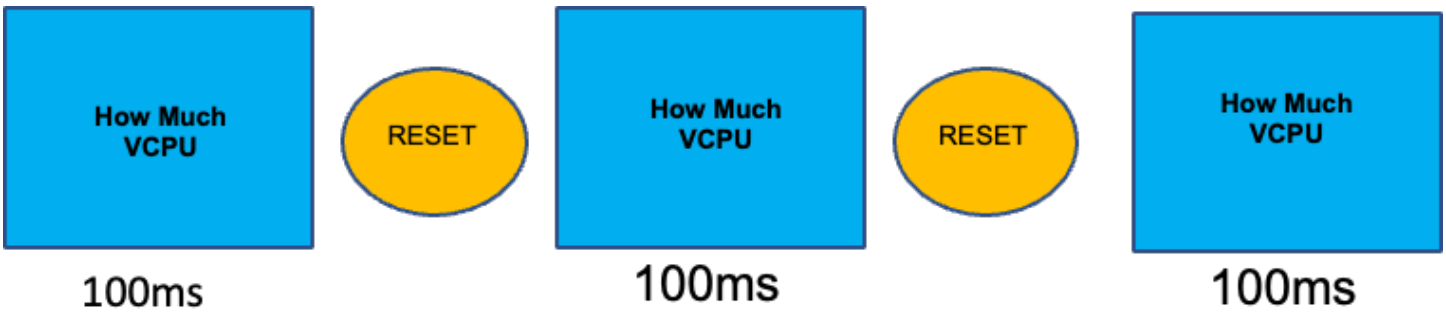


La conclusión clave es que la utilización de la CPU es solo una dimensión del rendimiento de las aplicaciones y los nodos. Utilizar el uso de la CPU como único indicador del estado de nuestros nodos y aplicaciones crea problemas de escalado, rendimiento y coste, conceptos todos ellos estrechamente relacionados. Cuanto más rendimiento tengan la aplicación y los nodos, menos tendrá que escalar, lo que, a su vez, reducirá sus costes.

Encontrar y usar las métricas de saturación correctas para escalar una aplicación en particular también le permite monitorear y alertar sobre los verdaderos cuellos de botella de esa aplicación. Si se omite este paso fundamental, los informes sobre problemas de rendimiento serán difíciles, si no imposibles, de entender.

Establecer límites de CPU

Para completar esta sección sobre temas mal entendidos, trataremos los límites de la CPU. En resumen, los límites son metadatos asociados al contenedor que tiene un contador que se restablece cada 100 ms. Esto ayuda a Linux a realizar un seguimiento de cuántos recursos de CPU utiliza un contenedor específico en todo el nodo en un período de 100 ms.



Un error común al establecer límites es suponer que la aplicación es de un solo subproceso y que solo se ejecuta en su vCPU «`asignada"`. En la sección anterior, aprendimos que el CFS no asigna núcleos y, en realidad, un contenedor que ejecute grandes grupos de subprocesos programará todas las CPU virtuales disponibles en la caja.

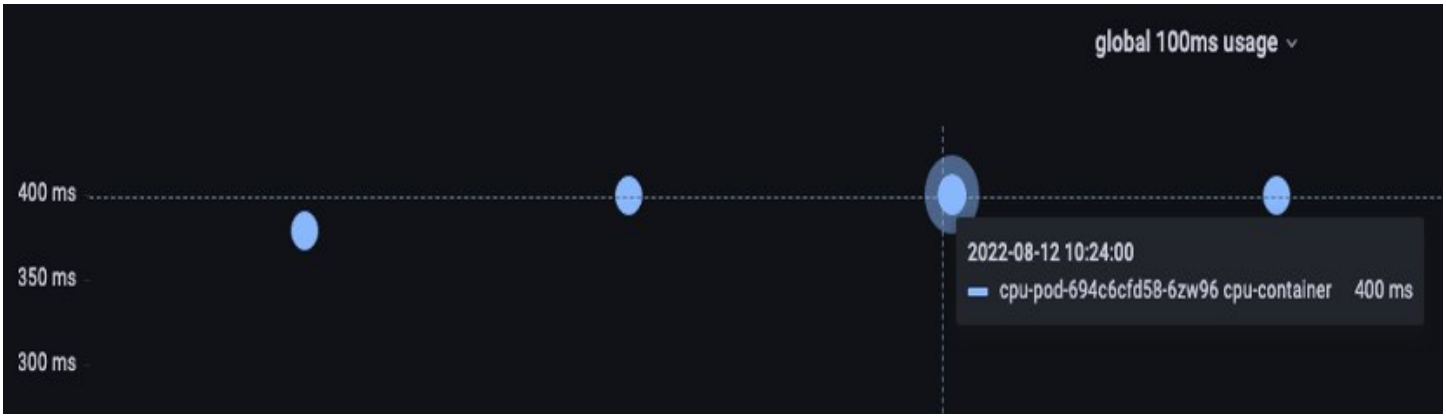
Si 64 subprocesos del sistema operativo se ejecutan en 64 núcleos disponibles (desde la perspectiva de los nodos de Linux), haremos que la factura total del tiempo de CPU utilizado en un período de 100 ms sea bastante alta después de sumar el tiempo de ejecución de todos esos 64 núcleos. Como es posible que esto solo ocurra durante un proceso de recolección de basura, es muy fácil pasar por alto algo como esto. Por eso es necesario utilizar métricas para asegurarnos de que tenemos el uso correcto a lo largo del tiempo antes de intentar establecer un límite.

Afortunadamente, tenemos una forma de ver exactamente la cantidad de vCPU que utilizan todos los subprocesos de una aplicación. Usaremos la métrica `container_cpu_usage_seconds_total` para este propósito.

Como la lógica de regulación se produce cada 100 ms y esta métrica es una métrica por segundo, utilizaremos PromQL para igualar este período de 100 ms. Si desea profundizar en esta declaración de ProMQL, consulte el siguiente blog.

Consulta de ProMQL:

```
topk(3, max by (pod, container)(rate(container_cpu_usage_seconds_total{image!="",
 instance="$instance"}[$__rate_interval]))) / 10
```



Una vez que consideremos que tenemos el valor correcto, podemos poner el límite a la producción. Luego, es necesario comprobar si nuestra solicitud se está retrasando debido a algo inesperado. Podemos hacerlo mirando `container_cpu_throttled_seconds_total`

```
topk(3, max by (pod, container)(rate(container_cpu_cfs_throttled_seconds_total{image!
=``""``, instance=``"$instance"``}[$__rate_interval]))) / 10
```



Memoria

La asignación de memoria es otro ejemplo en el que es fácil confundir el comportamiento de programación de Kubernetes con el comportamiento de Linux. CGroup Este es un tema más matizado, ya que se han producido cambios importantes en la forma en que la CGroup versión 2 gestiona la memoria en Linux y Kubernetes ha cambiado su sintaxis para reflejar estos cambios; lee este blog para obtener más información.

A diferencia de las solicitudes de CPU, las solicitudes de memoria no se utilizan una vez finalizado el proceso de programación. Esto se debe a que no podemos comprimir la memoria en la CGroup versión 1 de la misma manera que lo hacemos con la CPU. Esto nos deja solo con los límites de memoria, que están diseñados para actuar a prueba de fallos en caso de pérdidas de memoria al

cerrar el módulo por completo. Se trata de una propuesta de todo o nada, pero ahora se nos han dado nuevas formas de abordar este problema.

En primer lugar, es importante entender que configurar la cantidad correcta de memoria para los contenedores no es tan sencillo como parece. El sistema de archivos de Linux utilizará la memoria como caché para mejorar el rendimiento. Esta memoria caché aumentará con el paso del tiempo y puede resultar difícil saber cuánta memoria es buena tener para ella, pero se puede recuperar sin que ello repercuta de forma significativa en el rendimiento de las aplicaciones. Esto suele provocar una mala interpretación del uso de la memoria.

Tener la capacidad de «comprimir» la memoria fue uno de los principales impulsores de la CGroup versión 2. Para obtener más información sobre por qué era necesaria la CGroup V2, consulte la presentación de Chris Down, en la LISA21 que explica por qué no poder configurar correctamente la memoria mínima fue una de las razones que lo impulsaron a crear métricas de CGroup V2 y de estancarse por presión.

Afortunadamente, Kubernetes ahora tiene el concepto de «tener» y «menos». `memory.min` `memory.high` `requests.memory` Esto nos da la opción de liberar agresivamente esta memoria en caché para que la usen otros contenedores. Una vez que el contenedor alcanza el límite máximo de memoria, el núcleo puede recuperar la memoria de ese contenedor de forma agresiva hasta el valor establecido en. `memory.min` Esto nos da más flexibilidad cuando un nodo está bajo presión de memoria.

La pregunta clave es: ¿qué valor `memory.min` establecer? Aquí es donde entran en juego las métricas de bloqueo de la presión de la memoria. Podemos usar estas métricas para detectar la «pérdida de memoria» a nivel de contenedor. Luego, podemos usar controladores como fbtax para detectar los valores correctos buscando esta `memory.min` acumulación de memoria y establecer el `memory.min` valor de forma dinámica en esta configuración.

Resumen

Para resumir la sección, es fácil combinar los siguientes conceptos:

- Utilización y saturación
- Reglas de rendimiento de Linux con la lógica del programador de Kubernetes

Se debe tener mucho cuidado de mantener estos conceptos separados. El rendimiento y la escala están relacionados en un nivel profundo. El escalado innecesario crea problemas de rendimiento, lo que a su vez genera problemas de escalado.

Kubernetes Upstream SLOs

Amazon EKS ejecuta el mismo código que las versiones anteriores de Kubernetes y garantiza que los clústeres de EKS funcionen dentro de los parámetros SLOs definidos por la comunidad de Kubernetes. El Grupo de interés especial sobre escalabilidad (SIG) de Kubernetes define los objetivos de escalabilidad e investiga los obstáculos en el rendimiento a través de y. SLIs SLOs

SLIs son la forma en que medimos un sistema, como métricas o medidas que se pueden utilizar para determinar qué tan «bien» está funcionando el sistema, por ejemplo, la latencia o el recuento de solicitudes. SLOs definen los valores esperados cuando el sistema funciona «bien», por ejemplo, la latencia de las solicitudes permanece por debajo de 3 segundos. Kubernetes SLOs y Kubernetes SLIs se centran en el rendimiento de los componentes de Kubernetes y son completamente independientes del servicio Amazon EKS, SLAs que se centra en la disponibilidad del punto final del clúster de EKS.

Kubernetes tiene una serie de funciones que permiten a los usuarios ampliar el sistema con complementos o controladores personalizados, como controladores CSI, webhooks de admisión y escaladores automáticos. Estas extensiones pueden afectar drásticamente al rendimiento de un clúster de Kubernetes de diferentes maneras, por ejemplo, un webhook de admisión `failurePolicy=Ignore` podría añadir latencia a las solicitudes de la API de K8 si el destino del webhook no está disponible. El SIG de escalabilidad de Kubernetes define la escalabilidad utilizando el marco «tú lo prometes, nosotros lo prometemos»:

> Si prometes: - configurar correctamente tu clúster - utilizar las funciones de extensibilidad de forma «razonable» - mantener la carga en el clúster dentro de los límites recomendados
>
> entonces le prometemos que su clúster escalará, es decir: - todos quedarán SLOs satisfechos.

Kubernetes SLOs

Los Kubernetes SLOs no tienen en cuenta todos los complementos y las limitaciones externas que podrían afectar a un clúster, como el escalado de los nodos de trabajo o los webhooks de admisión. Estos SLOs se centran en los componentes de Kubernetes y garantizan que las acciones y los recursos de Kubernetes funcionen a la altura de las expectativas. SLOs Ayudan a los desarrolladores de Kubernetes a garantizar que los cambios en el código de Kubernetes no degraden el rendimiento de todo el sistema.

El SIG de escalabilidad de Kuberntes define el siguiente SLO/ oficial. SLIs El equipo de Amazon EKS realiza periódicamente pruebas de escalabilidad en los clústeres de EKS SLOs/SLIs para monitorizar la degradación del rendimiento a medida que se realizan cambios y se lanzan nuevas versiones.

Objetivo	Definición	SLO
Latencia de solicitud de API (mutante)	Latencia del procesamiento de las llamadas a la API mutantes para objetos individuales para cada par (recurso, verbo), medida como el percentil 99 durante los últimos 5 minutos	En la instalación predeterminada de Kubernetes, para cada par (recurso, verbo), excluidos los recursos virtuales y agregados y las definiciones de recursos personalizadas, el percentil 99 por día de clúster <= 1 s
Latencia de solicitud de API (solo lectura)	Latencia del procesamiento de las llamadas a la API de solo lectura que no se transmiten en streaming para cada par (recurso, ámbito), medida como el percentil 99 durante los últimos 5 minutos	En la instalación predeterminada de Kubernetes, para cada par (recurso, ámbito), excluidos los recursos virtuales y agregados y las definiciones de recursos personalizadas, el percentil 99 por día de clúster: (a) <= 1 s si (b) <= 30 en caso contrario (si o) `scope=resource scope=namespace scope=cluster`
Latencia de inicio del pod	La latencia de inicio de los pods programables y sin estado, excluido el tiempo necesario para extraer imágenes y ejecutar los contenedores de inicio, se mide desde la fecha de creación del pod hasta el momento en que se informa	En la instalación predeterminada de Kubernetes, el percentil 99 por día de clúster es inferior a 5 segundos

Objetivo	Definición	SLO
	que todos sus contenedores se han iniciado y se observan mediante un reloj, medida como el percentil 99 durante los últimos 5 minutos	

Latencia de solicitud de API

Se `kube-apiserver` ha `--request-timeout` definido como `1m0s` predeterminada, lo que significa que una solicitud puede ejecutarse hasta un minuto (60 segundos) antes de que se agote el tiempo de espera y se cancele. La latencia SLOs definida se desglosa según el tipo de solicitud que se está realizando, que puede ser mutante o de solo lectura:

Mutante

Las solicitudes mutantes en Kubernetes realizan cambios en un recurso, como creaciones, eliminaciones o actualizaciones. Estas solicitudes son caras porque esos cambios deben escribirse en el backend etcd antes de que se devuelva el objeto actualizado. Etcd es un almacén de valores clave distribuido que se utiliza para todos los datos del clúster de Kubernetes.

Esta latencia se mide como el percentil 99 a lo largo de 5 minutos para pares (recurso, verbo) de recursos de Kubernetes. Por ejemplo, esto mediría la latencia de las solicitudes de creación de pods y de actualización de nodos. La latencia de la solicitud debe ser inferior a 1 segundo para cumplir con el SLO.

Solo lectura

Las solicitudes de solo lectura recuperan un único recurso (como Get Pod X) o una colección (como «Obtener todos los pods del espacio de nombres X»). `kube-apiserver`Mantiene una caché de objetos, por lo que es posible que los recursos solicitados se devuelvan de la caché o que sea necesario recuperarlos primero de etcd. Estas latencias también se miden con el percentil 99 a lo largo de 5 minutos; sin embargo, las solicitudes de solo lectura pueden tener ámbitos distintos. El SLO define dos objetivos diferentes:

- Para las solicitudes realizadas para un solo recurso (es decir,`kubectl get pod -n mynamespace my-controller-xxx`), la latencia de la solicitud debe permanecer <= 1 segundo.

- Para las solicitudes que se realizan para varios recursos en un espacio de nombres o un clúster (por ejemplo`kubectl get pods -A`), la latencia debe permanecer <= 30 segundos

El SLO tiene valores objetivo diferentes para los distintos ámbitos de solicitud, ya que las solicitudes realizadas para una lista de recursos de Kubernetes esperan que los detalles de todos los objetos de la solicitud se devuelvan en el SLO. En clústeres grandes o colecciones de recursos de gran tamaño, esto puede generar respuestas de gran tamaño, que pueden tardar algún tiempo en recuperarse. Por ejemplo, en un clúster que ejecute decenas de miles de pods y cada pod tenga aproximadamente 1 KiB cuando esté codificado en JSON, la devolución de todos los pods del clúster consistiría en 10 MB o más. Los clientes de Kubernetes pueden ayudar a reducir este tamaño de respuesta mediante el uso de APIList Chunking para recuperar grandes colecciones de recursos.

Latencia de inicio del pod

Este SLO se refiere principalmente al tiempo que pasa desde la creación del pod hasta que los contenedores de ese pod comienzan a ejecutarse. Para medirlo, se calcula la diferencia entre la marca temporal de creación registrada en el pod y el momento en que un WATCH informa que los contenedores han comenzado a funcionar (sin incluir el tiempo que se tarda en extraer la imagen del contenedor y en el que se inicia la ejecución del contenedor). Para cumplir con el SLO, el percentil 99 por día de clúster de la latencia de inicio de este pod debe ser inferior a 5 segundos.

Ten en cuenta que este SLO supone que los nodos de trabajo ya existen en este clúster y están preparados para que se programe el Pod. Este SLO no tiene en cuenta las extracciones de imágenes ni las ejecuciones de contenedores de inicio, y también limita la prueba a los «pods sin estado» que no utilizan complementos de almacenamiento persistente.

Métricas SLI de Kubernetes

Kubernetes también está mejorando la observabilidad en torno al SLIs añadir métricas de Prometheus a los componentes de Kubernetes que las rastrean a lo largo del tiempo. SLIs Con el lenguaje de consultas Prometheus (ProMQL), podemos crear consultas que muestren el rendimiento del SLI a lo largo del tiempo en herramientas como los paneles de Prometheus o Grafana. A continuación, se muestran algunos ejemplos de lo anterior. SLOs

Latencia de solicitudes del servidor API

Métrica	Definición
apiserver_request_sli_duration_seconds	Distribución de la latencia de respuesta (sin contar la duración de los webhooks ni los tiempos de espera de las colas de prioridad y equidad) en segundos para cada verbo, grupo, versión, recurso, subrecurso, ámbito y componente.
apiserver_request_duration_seconds	Distribución de la latencia de respuesta en segundos para cada verbo, valor de ensayo, grupo, versión, recurso, subrecurso, ámbito y componente.

> **Note**
>
> La métrica está disponible a partir de la versión 1.27 de Kubernetes. `apiserver_request_sli_duration_seconds`

Puedes usar estas métricas para investigar los tiempos de respuesta del servidor API y si hay cuellos de botella en los componentes de Kubernetes o en otros complementos o componentes. Las consultas que aparecen a continuación se basan en el panel de control de SLO de la comunidad.

Latencia de solicitud de API SLI (mutante): este tiempo no incluye la ejecución del webhook ni el tiempo de espera en la cola. `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`

Latencia total de solicitudes de API (mutante): es el tiempo total que tardó la solicitud en el servidor de API. Este tiempo puede ser superior al tiempo de SLI, ya que incluye la ejecución de webhooks y los tiempos de espera de prioridad y equidad de la API. `histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_bucket{verb=~"CREATE|DELETE|`

```
PATCH|POST|PUT", subresource!~"proxy|attach|log|exec|portforward"}[5m])) by
(resource, subresource, verb, scope, le)) > 0
```

En estas consultas, excluimos las solicitudes de la API de streaming que no se devuelven inmediatamente, como `kubectl port-forward` o `kubectl exec` requests (`subresource!~"proxy|attach|log|exec|portforward"`), y filtramos solo los verbos de Kubernetes que modifican objetos (). `verb=~"CREATE|DELETE|PATCH|POST|PUT"` A continuación, calculamos el percentil 99 de esa latencia durante los últimos 5 minutos.

Podemos usar una consulta similar para las solicitudes de API de solo lectura, simplemente modificamos los verbos por los que estamos filtrando para incluir las acciones de solo lectura y. `LIST` `GET` También hay diferentes umbrales de SLO en función del alcance de la solicitud, es decir, obtener un único recurso o enumerar varios recursos.

Latencia de solicitud de API SLI (solo lectura): este tiempo no incluye la ejecución del webhook ni el tiempo de espera en la cola. Para un solo recurso (alcance = recurso, umbral = 1 s) `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"GET", scope=~"resource"}[5m])) by (resource, subresource, verb, scope, le))`

Para un conjunto de recursos en el mismo espacio de nombres (scope=namespace, umbral=5s) `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"LIST", scope=~"namespace"}[5m])) by (resource, subresource, verb, scope, le))`

Para un conjunto de recursos de todo el clúster (alcance=clúster, umbral=30s) `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"LIST", scope=~"cluster"}[5m])) by (resource, subresource, verb, scope, le))`

Latencia total de la solicitud de la API (solo lectura): es el tiempo total que tardó la solicitud en el servidor de la API. Este tiempo puede ser superior al de la SLI, ya que incluye los tiempos de espera y de ejecución de los webhooks. Para un único recurso (scope=resource, umbral=1s) `histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_bucket{verb=~"GET", scope=~"resource"}[5m])) by (resource, subresource, verb, scope, le))`

Para un conjunto de recursos en el mismo espacio de nombres (scope=namespace, umbral=5s) `histogram_quantile(0.99,`

```
sum(rate(apiserver_request_duration_seconds_bucket{verb=~"LIST",
scope=~"namespace"}[5m])) by (resource, subresource, verb, scope, le))
```

Para un conjunto de recursos de todo el clúster (alcance=clúster, umbral=30s)

```
histogram_quantile(0.99,
sum(rate(apiserver_request_duration_seconds_bucket{verb=~"LIST",
scope=~"cluster"}[5m])) by (resource, subresource, verb, scope, le))
```

Las métricas del SLI proporcionan información sobre el rendimiento de los componentes de Kubernetes, ya que excluyen el tiempo que las solicitudes pasan esperando en las colas de prioridad y equidad de las API, pasando por webhooks de admisión u otras extensiones de Kubernetes. Las métricas totales proporcionan una visión más holística, ya que reflejan el tiempo que sus aplicaciones estarían esperando una respuesta del servidor de API. La comparación de estas métricas puede proporcionar información sobre dónde se están produciendo los retrasos en el procesamiento de las solicitudes.

Latencia de inicio del pod

Métrica	Definición
kubelet_pod_start_sli_duration_seconds	Duración en segundos que se tarda en iniciar un pod, excluido el tiempo necesario para extraer imágenes y ejecutar los contenedo res de inicio, que se mide desde la fecha de creación del pod hasta el momento en que se indica que todos sus contenedores están iniciados y se observan mediante un reloj
kubelet_pod_start_duration_seconds	Duración en segundos desde que Kubelet ve un pod por primera vez hasta que el pod empieza a funcionar. Esto no incluye el tiempo necesario para programar el pod ni ampliar la capacidad del nodo de trabajo.

> **Note**
>
> `kubelet_pod_start_sli_duration_seconds`está disponible a partir de la versión 1.27 de Kubernetes.

Al igual que en las consultas anteriores, puedes usar estas métricas para saber cuánto tiempo demoran el lanzamiento del pod el escalado de nodos, la extracción de imágenes y los contenedores de inicio, en comparación con las acciones de Kubelet.

Latencia de inicio del pod (SLI): es el tiempo transcurrido desde la creación del pod hasta el momento en que los contenedores de aplicaciones indican que están en ejecución. Esto incluye el tiempo que tarda la capacidad del nodo de trabajo en estar disponible y el pod en programarse, pero no incluye el tiempo que se tarda en extraer imágenes o en ejecutar los contenedores de inicio. `histogram_quantile(0.99, sum(rate(kubelet_pod_start_sli_duration_seconds_bucket[5m])) by (le))`

Latencia total de inicio del pod: es el tiempo que tarda el kubelet en iniciar el pod por primera vez. Se mide a partir del momento en que el kubelet recibe el pod mediante WATCH, sin incluir el tiempo necesario para escalar o programar el nodo de trabajo. Esto incluye el tiempo necesario para extraer imágenes e iniciar los contenedores para que se ejecuten. `histogram_quantile(0.99, sum(rate(kubelet_pod_start_duration_seconds_bucket[5m])) by (le))`

SLOs en tu clúster

Si recopila las métricas de Prometheus de los recursos de Kubernetes de su clúster de EKS, puede obtener información más profunda sobre el rendimiento de los componentes del plano de control de Kubernetes.

El repositorio de pruebas de rendimiento incluye paneles de Grafana que muestran las latencias y las métricas de rendimiento críticas del clúster durante las pruebas. La configuración de pruebas de rendimiento aprovecha el kube-prometheus-stack, un proyecto de código abierto que viene configurado para recopilar métricas de Kubernetes, pero también puedes usar Prometheus gestionado por Amazon y Grafana gestionado por Amazon.

Si utiliza la solución Prometheus `kube-prometheus-stack` o una similar, puede instalar el mismo panel de control para SLOs observarlo en su clúster en tiempo real.

1. Primero tendrá que instalar las reglas de Prometheus que se utilizan en los paneles de control con. `kubectl apply -f prometheus-rules.yaml` Puede descargar una copia de las reglas aquí: https://github.com/kubernetes/ perf- -rules.yaml tests/blob/master/clusterloader2/pkg/ prometheus/manifests/prometheus
   a. Asegúrese de comprobar que el espacio de nombres del archivo coincida con su entorno
   b. Compruebe que las etiquetas coincidan con el valor de `prometheus.prometheusSpec.ruleSelector` helm si está utilizando `kube-prometheus-stack`
2. A continuación, puede instalar los cuadros de mando en Grafana. Los cuadros de mando de json y los scripts de Python para generarlos están disponibles aquí: perf- https://github.com/kubernetes/ tests/tree/master/clusterloader2/pkg/prometheus/manifests/dashboards
   a. el `slo.json` panel muestra el rendimiento del clúster en relación con Kubernetes SLOs

Tenga en cuenta que SLOs se centran en el rendimiento de los componentes de Kubernetes en sus clústeres, pero hay métricas adicionales que puede revisar y que proporcionan diferentes perspectivas o conocimientos sobre su clúster. Los proyectos comunitarios de Kubernetes, como K, ube-state-metrics pueden ayudarte a analizar rápidamente las tendencias de tu clúster. Los complementos y controladores más comunes de la comunidad de Kubernetes también emiten métricas de Prometheus, lo que te permite investigar cosas como escaladores automáticos o programadores personalizados.

La guía de prácticas recomendadas de observabilidad incluye ejemplos de otras métricas de Kubernetes que puede utilizar para obtener más información.

Known Limits and Service Quotas

Amazon EKS se puede usar para una variedad de cargas de trabajo y puede interactuar con una amplia gama de servicios de AWS, y hemos visto que las cargas de trabajo de los clientes se enfrentan a un rango similar de cuotas de servicio de AWS y otros problemas que dificultan la escalabilidad.

Su cuenta de AWS tiene cuotas predeterminadas (un límite máximo en la cantidad de recursos de AWS que su equipo puede solicitar). Cada servicio de AWS define su propia cuota y, por lo general, las cuotas son específicas de cada región. Puede solicitar aumentos para algunas cuotas (límites flexibles) y otras cuotas no se pueden aumentar (límites estrictos). Debe tener en cuenta estos

valores al diseñar la arquitectura de sus aplicaciones. Considere revisar estos límites de servicio periódicamente e incorpórelos durante el diseño de la aplicación.

Puede revisar el uso en su cuenta y abrir una solicitud de aumento de cuota en la consola de AWS Service Quotas o mediante la CLI de AWS. Consulte la documentación de AWS del servicio de AWS correspondiente para obtener más información sobre las cuotas de servicio y cualquier otra restricción o aviso sobre su aumento.

> **Note**
>
> Amazon EKS Service Quotas incluye una lista de las cuotas de servicio y tiene enlaces para solicitar aumentos cuando estén disponibles.

Otras cuotas de AWS Service

Hemos visto cómo los clientes de EKS se ven afectados por las cuotas que se indican a continuación para otros servicios de AWS. Es posible que algunas de estas solo se apliquen a configuraciones o casos de uso específicos, pero puede considerar si su solución se adaptará a alguna de estas condiciones a medida que vaya escalando. Las cuotas están organizadas por servicio y cada cuota tiene un identificador con el formato L-XXXXXXXX que puede utilizar para buscarlo en la consola de AWS Service Quotas.

Servicio	Cuota (L-xxxxx)	Impact	CARNÉ DE IDENTIDAD (L-xxxxx)	predeterminado
IAM	Roles por cuenta	Puede limitar la cantidad de clústeres o funciones de IRSA en una cuenta.	L- D64 FE177	1 000
IAM	OpenId conecta proveedores por cuenta	Puede limitar la cantidad de clústeres por cuenta, IRSA	L-858F3967	100

Servicio	Cuota (L-xxxxx)	Impact	CARNÉ DE IDENTIDAD (L-xxxxx)	predeterminado
		utiliza OpenID Connect		
IAM	Longitud de la política de confianza de rol	Puede limitar la cantidad de clústeres a los que está asociada una función de IAM en el IRSA	L-C07B4B0D	2048
VPC	Grupos de seguridad por interfaz de red	Puede limitar el control o la conectividad de la red del clúster	L-2 AFB9258	5
VPC	IPv4 Bloques CIDR por VPC	Puede limitar la cantidad de nodos de trabajo de EKS	L-83CA0A9D	5
VPC	Rutas por tabla de enrutamiento	Puede limitar el control o la conectividad de la red del clúster	L-93826ACB	50
VPC	Interconexiones de VPC activas por VPC	Puede limitar el control o la conectividad de la red del clúster	L-7E9ECCDB	50

Servicio	Cuota (L-xxxxx)	Impact	CARNÉ DE IDENTIDAD (L-xxxxx)	predeterminado
VPC	Reglas de entrada o salida por grupo de seguridad.	Algunos controladores de EKS crean reglas nuevas, ya que pueden limitar el control o la conectividad de la red del clúster	L-0 095F EA8	50
VPC	VPCs por región	Puede limitar la cantidad de clústeres por cuenta o el control o la conectividad de la red de su clúster	L-F678F1CE	5
VPC	Puertas de enlace de Internet por región	Puede limitar la cantidad de clústeres por cuenta o el control o la conectividad de la red de su clúster	L-A4707A72	5

Servicio	Cuota (L-xxxxx)	Impact	CARNÉ DE IDENTIDAD (L-xxxxx)	predeterminado
VPC	Interfaces de red por región	Puede limitar el número de nodos Worker de EKS o afectar las actividad es del plano de control de EKS. scaling/update	L- DF5 E4 CA3	5 000
VPC	Uso de direccion es de red	Puede limitar la cantidad de clústeres por cuenta o el control o la conectividad de la red de su clúster	L- BB24 F6E5	64.000
VPC	Uso de direccion es de red interconectadas	Puede limitar la cantidad de clústeres por cuenta o el control o la conectividad de la red de su clúster	L- CD17 FD4 B	128.000
ELB	Oyentes por equilibrador de carga de red	Puede limitar el control de la entrada de tráfico al clúster.	L-57A373D6	50

Servicio	Cuota (L-xxxxx)	Impact	CARNÉ DE IDENTIDAD (L-xxxxx)	predeterminado
ELB	Grupos de destino por región	Puede limitar el control de la entrada de tráfico al clúster.	L-B22855CB	3000
ELB	Destinos por Equilibrador de carga de aplicación	Puede limitar el control de la entrada de tráfico al clúster.	L-7E6692B2	1 000
ELB	Destinos por equilibrador de carga de red	Puede limitar el control de la entrada de tráfico al clúster.	L- AD04 EEF1	3000
ELB	Destinos por zona de disponibilidad por equilibrador de carga de red	Puede limitar el control de la entrada de tráfico al clúster.	L-B211E961	500
ELB	Destinos por grupo de destino y región	Puede limitar el control de la entrada de tráfico al clúster.	L-A0D0B863	1 000
ELB	Equilibradores de carga de aplicaciones por región	Puede limitar el control de la entrada de tráfico al clúster.	L-53 B97 DA6	50

Servicio	Cuota (L-xxxxx)	Impact	CARNÉ DE IDENTIDAD (L-xxxxx)	predeterminado
ELB	Equilibrador de carga clásico por región	Puede limitar el control de la entrada de tráfico al clúster.	L-E9E9831D	20
ELB	Balanceadores de carga de red por región	Puede limitar el control de la entrada de tráfico al clúster.	L-69A177A2	50
EC2	Ejecución de instancias estándar bajo demanda (A, C, D, H, I, M, R, T, Z) (como número máximo de vCPU)	Puede limitar la cantidad de nodos de trabajo de EKS	L-1216C47A	5
EC2	Todas las solicitudes de instancia s puntuales estándar (A, C, D, H, I, M, R, T, Z) (como número máximo de vCPU)	Puede limitar la cantidad de nodos de trabajo de EKS	L-34B43A08	5

Servicio	Cuota (L-xxxxx)	Impact	CARNÉ DE IDENTIDAD (L-xxxxx)	predeterminado
EC2	EC2-VPC elástico IPs	Puede limitar la cantidad de NAT GWs (y, por lo tanto, VPCs), lo que puede limitar la cantidad de clústeres en una región	L-0263D0A3	5
EBS	Instantáneas por región	Puede limitar la estrategia de respaldo para cargas de trabajo con estado	L-309 BACF6	100 000
EBS	Almacenamiento de volúmenes de SSD de uso general (gp3), en TiB	Puede limitar la cantidad de nodos de trabajo o Persisten tVolume almacenamiento de EKS	L-7A658B76	50
EBS	Almacenamiento de volúmenes de SSD de uso general (gp2), en TiB	Puede limitar la cantidad de nodos de trabajo o almacenam iento de EKS Persisten tVolume	L-D18 D FCD1	50

Servicio	Cuota (L-xxxxx)	Impact	CARNÉ DE IDENTIDAD (L-xxxxx)	predeterminado
ECR	Repositorios registrados	Puede limitar la cantidad de cargas de trabajo en sus clústeres	L- E8D CFEB8	100 000
ECR	Imágenes por repositorio	Puede limitar la cantidad de cargas de trabajo en sus clústeres	L-03A36 CE1	20 000
SecretsManager	Secretos por región	Puede limitar la cantidad de cargas de trabajo en sus clústeres	L-2F66C23C	500.000

Limitación de solicitudes de AWS

Los servicios de AWS también implementan la limitación de solicitudes para garantizar que sigan funcionando y estén disponibles para todos los clientes. Al igual que en Service Quotas, cada servicio de AWS mantiene sus propios umbrales de limitación de solicitudes. Considere la posibilidad de revisar la documentación del servicio de AWS correspondiente si sus cargas de trabajo van a necesitar emitir rápidamente un gran número de llamadas a la API o si observa errores de limitación de solicitudes en su aplicación.

EC2 Las solicitudes de API relacionadas con el aprovisionamiento EC2 de interfaces de red o direcciones IP pueden verse limitadas en clústeres grandes o cuando los clústeres se amplían drásticamente. En la siguiente tabla, se muestran algunas de las acciones de la API que, según hemos observado, se limitan las solicitudes de los clientes. Puedes consultar los valores predeterminados del límite EC2 de velocidad y los pasos para solicitar un aumento del límite de velocidad en la EC2 documentación de Rate Throttling.

Acciones mutantes	Acciones de solo lectura
AssignPrivateIpAddresses	DescribeDhcpOptions
AttachNetworkInterface	DescribeInstances
CreateNetworkInterface	DescribeNetworkInterfaces
DeleteNetworkInterface	DescribeSecurityGroups
DeleteTags	DescribeTags
DetachNetworkInterface	DescribeVpcs
ModifyNetworkInterfaceAttribute	DescribeVolumes
UnassignPrivateIpAddresses	

Otros límites conocidos

- Route 53 también tiene un límite de velocidad bastante bajo de 5 solicitudes por segundo a la API de Route 53. Si tiene que actualizar una gran cantidad de dominios con un proyecto como el DNS externo, es posible que vea una reducción de la velocidad y retrasos en la actualización de los dominios.
  - Algunos tipos de instancias de Nitro tienen un límite de 28 volúmenes de adjuntos que se comparten entre los volúmenes de Amazon EBS, las interfaces de red y los volúmenes de almacenes de NVMe instancias. Si tus cargas de trabajo están acumulando numerosos volúmenes de EBS, es posible que la densidad de pods que puedas alcanzar con estos tipos de instancias esté limitada
  - Hay un número máximo de conexiones que se pueden rastrear por instancia de Ec2. Si sus cargas de trabajo gestionan un gran número de conexiones, es posible que se produzcan fallos o errores de comunicación debido a que se ha alcanzado este máximo. Puede usar las métricas de rendimiento `conntrack_allowance_available` y de la `conntrack_allowance_exceeded` red para monitorear la cantidad de conexiones rastreadas en sus nodos de trabajo de EKS.
  - En el entorno EKS, el límite de almacenamiento de etcd es de 8 GiB según las instrucciones iniciales. Supervise las métricas `apiserver_storage_size_bytes` para rastrear el

tamaño de etcd db. Puede consultar las reglas de alerta `etcdBackendQuotaLowSpace` y `etcdExcessiveDatabaseGrowth` configurar este monitoreo.

Prácticas recomendadas para las actualizaciones de clústeres

Esta guía muestra a los administradores de clústeres cómo planificar y ejecutar su estrategia de actualización de Amazon EKS. También describe cómo actualizar los nodos autogestionados, los grupos de nodos gestionados, los nodos Karpenter y los nodos Fargate. No incluye orientación sobre EKS Anywhere, Kubernetes autogestionado, AWS Outposts o AWS Local Zones.

Descripción general de

Una versión de Kubernetes abarca tanto el plano de control como el plano de datos. Para garantizar un funcionamiento sin problemas, tanto el plano de control como el plano de datos deben ejecutar la misma versión secundaria de Kubernetes, como la 1.24. Si bien AWS administra y actualiza el plano de control, es su responsabilidad actualizar los nodos de trabajo en el plano de datos.

- Plano de control: el servidor API de Kubernetes determina la versión del plano de control. En los clústeres de Amazon EKS, AWS se encarga de administrar este componente. Las actualizaciones del plano de control se pueden iniciar mediante la API de AWS.
- Plano de datos: la versión del plano de datos está asociada a las versiones de Kubelet que se ejecutan en sus nodos individuales. Es posible tener nodos del mismo clúster que ejecuten versiones diferentes. Puede comprobar las versiones de todos los nodos ejecutando`kubectl get nodes`.

Antes de actualizar

Si planea actualizar su versión de Kubernetes en Amazon EKS, hay algunas políticas, herramientas y procedimientos importantes que debe implementar antes de iniciar la actualización.

- Comprenda las políticas de obsolescencia: obtenga una comprensión profunda de cómo funciona la política de obsolescencia de Kubernetes. Esté al tanto de cualquier cambio futuro que pueda afectar a sus aplicaciones actuales. Las versiones más recientes de Kubernetes suelen eliminar gradualmente algunas APIs funciones, lo que podría provocar problemas en la ejecución de las aplicaciones.
- Revise el registro de cambios de Kubernetes: revise minuciosamente el registro de cambios de Kubernetes junto con las versiones de Amazon EKS Kubernetes para comprender cualquier

posible impacto en su clúster, como los cambios importantes que puedan afectar a sus cargas de trabajo.

- Evalúe la compatibilidad de los complementos del clúster: Amazon EKS no actualiza automáticamente un complemento cuando se publican nuevas versiones o después de actualizar el clúster a una nueva versión secundaria de Kubernetes. Consulte Actualización de un complemento para conocer la compatibilidad de cualquier complemento de clúster existente con la versión de clúster a la que pretende actualizarse.
- Habilite el registro en el plano de control: habilite el registro en el plano de control para capturar los registros, los errores o los problemas que puedan surgir durante el proceso de actualización. Considere la posibilidad de revisar estos registros para detectar cualquier anomalía. Pruebe las actualizaciones de los clústeres en un entorno que no sea de producción o integre pruebas automatizadas en su flujo de trabajo de integración continua para evaluar la compatibilidad de las versiones con sus aplicaciones, controladores e integraciones personalizadas.
- Explore eksctl para la administración de clústeres: considere usar eksctl para administrar su clúster de EKS. Le permite actualizar el plano de control, administrar los complementos y gestionar las actualizaciones de los nodos de trabajo. out-of-the-box
- Opte por el modo automático o los grupos de nodos gestionados: agilice y automatice las actualizaciones de los nodos de trabajo mediante el modo automático o los grupos de nodos gestionados por EKS. Estas opciones simplifican el proceso y reducen la intervención manual.
- Utilice el complemento kubectl Convert: aproveche el complemento kubectl convert para facilitar la conversión de los archivos de manifiesto de Kubernetes entre diferentes versiones de la API. Esto puede ayudar a garantizar que tus configuraciones sigan siendo compatibles con la nueva versión de Kubernetes.

Conserve su clúster up-to-date

Mantenerse al día con las actualizaciones de Kubernetes es fundamental para un entorno de EKS seguro y eficiente, que refleje el modelo de responsabilidad compartida de Amazon EKS. Al integrar estas estrategias en su flujo de trabajo operativo, se posiciona para mantener up-to-date clústeres seguros que aprovechan al máximo las funciones y mejoras más recientes. Tácticas:

- Política de versiones compatibles: en consonancia con la comunidad de Kubernetes, Amazon EKS suele ofrecer tres versiones activas de Kubernetes. Amazon EKS ofrece soporte estándar a una versión secundaria de Kubernetes durante los primeros 14 meses después de su lanzamiento. Cuando una versión supera la fecha de finalización del soporte estándar, pasa al periodo de

soporte extendido durante los 12 meses siguientes. Los avisos de obsolescencia se publican al menos 60 días antes de que una versión alcance la fecha de finalización del soporte estándar. Para obtener más información, consulte los documentos del ciclo de vida de la versión EKS.

- Política de actualización automática: le recomendamos encarecidamente que esté sincronizado con las actualizaciones de Kubernetes en su clúster de EKS. Los clústeres que se ejecuten en una versión de Kubernetes que haya completado su ciclo de vida de 26 meses (14 meses de soporte estándar más 12 meses de soporte extendido) se actualizarán automáticamente a la siguiente versión. Tenga en cuenta que puede deshabilitar el soporte extendido. Si no se actualiza de forma proactiva antes de la versión, se end-of-life desencadena una actualización automática, lo que podría interrumpir las cargas de trabajo y los sistemas. Para obtener información adicional, consulte la versión EKS. FAQs
- Cree manuales de actualización: establezca un proceso bien documentado para gestionar las actualizaciones. Como parte de su enfoque proactivo, desarrolle manuales y herramientas especializadas que se adapten a su proceso de actualización. Esto no solo mejora su preparación, sino que también simplifica las transiciones complejas. Convierta en una práctica estándar actualizar sus clústeres al menos una vez al año. Esta práctica lo alinea con los avances tecnológicos continuos y, por lo tanto, aumenta la eficiencia y la seguridad de su entorno.

Consulte el calendario de lanzamientos de EKS

Consulta el calendario de lanzamientos de EKS Kubernetes para saber cuándo llegarán nuevas versiones y cuándo finalizará el soporte para versiones específicas. Por lo general, EKS publica tres versiones secundarias de Kubernetes al año, y cada versión secundaria tiene soporte durante unos 14 meses.

Además, consulte la información original sobre las versiones de Kubernetes.

Comprenda cómo se aplica el modelo de responsabilidad compartida a las actualizaciones de clústeres

Usted es responsable de iniciar la actualización tanto del plano de control del clúster como del plano de datos. Obtenga información sobre cómo iniciar una actualización. Al iniciar una actualización de un clúster, AWS gestiona la actualización del plano de control del clúster. Usted es responsable de actualizar el plano de datos, incluidos los pods y complementos de Fargate. Debe validar y planificar las actualizaciones de las cargas de trabajo que se ejecutan en su clúster para garantizar que su disponibilidad y sus operaciones no se vean afectadas tras la actualización del clúster

Actualice los clústeres in situ

EKS admite una estrategia de actualización de clústeres in situ. Esto mantiene los recursos del clúster y mantiene la coherencia de la configuración del clúster (por ejemplo, el punto final de la API, el OIDC o los balanceadores ENIs de carga). Esto es menos molesto para los usuarios del clúster y utilizará las cargas de trabajo y los recursos existentes en el clúster sin necesidad de volver a implementar las cargas de trabajo o migrar recursos externos (por ejemplo, DNS o almacenamiento).

Al realizar una actualización del clúster in situ, es importante tener en cuenta que solo se puede ejecutar una actualización de versión secundaria a la vez (por ejemplo, de la versión 1.24 a la 1.25).

Esto significa que si necesitas actualizar varias versiones, necesitarás una serie de actualizaciones secuenciales. Planificar las actualizaciones secuenciales es más complicado y conlleva un mayor riesgo de tiempo de inactividad. En esta situación, consultethe section called “Evalúe Blue/Green los clústeres como una alternativa a las actualizaciones de clústeres in situ”.

Actualice el plano de control y el plano de datos en secuencia

Para actualizar un clúster, deberá realizar las siguientes acciones:

1. Revisa las notas de la versión de Kubernetes y EKS.
2. Realice una copia de seguridad del clúster. (opcional)
3. Identifique y corrija el uso de API obsoleto o eliminado en sus cargas de trabajo.
4. Asegúrese de que los grupos de nodos gestionados, si se utilizan, estén en la misma versión de Kubernetes que el plano de control. Los grupos de nodos gestionados por EKS y los nodos creados por los perfiles Fargate de EKS admiten 2 versiones menores de sesgo entre el plano de control y el plano de datos para la versión 1.27 y anteriores de Kubernetes. A partir de la versión 1.28, los grupos de nodos gestionados por EKS y los nodos creados por los perfiles Fargate de EKS admiten 3 versiones secundarias sesgadas entre el plano de control y el plano de datos. Por ejemplo, si la versión de su plano de control de EKS es la 1.28, puede utilizar de forma segura versiones de kubelet tan antiguas como la 1.25. Si tu versión de EKS es la 1.27, la versión más antigua de kubelet que puedes usar es la 1.25.
5. Actualice el plano de control del clúster mediante la consola o la CLI de AWS.
6. Revise la compatibilidad de los complementos. Actualice sus complementos y controladores personalizados de Kubernetes, según sea necesario.
7. Actualiza kubectl.

8. Actualice el plano de datos del clúster. Actualice sus nodos a la misma versión secundaria de Kubernetes que el clúster actualizado.

> **Tip**
>
> Si el clúster se creó con el modo automático de EKS, no es necesario que actualice el plano de datos del clúster. Tras actualizar el plano de control, el modo automático de EKS comenzará a actualizar de forma gradual los nodos gestionados, respetando todos los presupuestos de interrupción de los módulos. Asegúrese de supervisar estas actualizaciones para verificar el cumplimiento de sus requisitos operativos.

Utilice la documentación de EKS para crear una lista de verificación de actualizaciones

La documentación de la versión de EKS Kubernetes incluye una lista detallada de los cambios para cada versión. Cree una lista de comprobación para cada actualización.

Para obtener una guía específica sobre la actualización de la versión de EKS, consulte la documentación para ver los cambios y consideraciones más importantes para cada versión.

- Consulte las notas de publicación de las versiones de Kubernetes sobre el soporte estándar
- Revisa las notas de publicación de las versiones de Kubernetes sobre soporte extendido

Actualice los complementos y componentes mediante la API de Kubernetes

Antes de actualizar un clúster, debes saber qué versiones de los componentes de Kubernetes utilizas. Haga un inventario de los componentes del clúster e identifique los componentes que utilizan directamente la API de Kubernetes. Esto incluye componentes críticos del clúster, como agentes de monitoreo y registro, escaladores automáticos de clústeres, controladores de almacenamiento en contenedores (por ejemplo, EBS CSI, EFS CSI), controladores de ingreso y cualquier otra carga de trabajo o complemento que dependa directamente de la API de Kubernetes.

> **Tip**
>
> Los componentes críticos del clúster suelen instalarse en un espacio de nombres `*-system`

```
kubectl get ns | grep '-system'
```

Una vez que hayas identificado los componentes que dependen de la API de Kubernetes, consulta su documentación para ver los requisitos de compatibilidad de versiones y actualización. Por ejemplo, consulte la documentación del controlador Load Balancer de AWS para obtener información sobre la compatibilidad de las versiones. Es posible que sea necesario actualizar algunos componentes o cambiar la configuración antes de proceder a la actualización del clúster. Algunos componentes críticos que hay que comprobar son CoredNS, kube-proxy, VPC CNI y controladores de almacenamiento.

Los clústeres suelen contener muchas cargas de trabajo que utilizan la API de Kubernetes y son necesarias para la funcionalidad de las cargas de trabajo, como los controladores de entrada, los sistemas de entrega continua y las herramientas de supervisión. Al actualizar un clúster de EKS, también debe actualizar sus complementos y herramientas de terceros para asegurarse de que son compatibles.

Consulte los siguientes ejemplos de complementos comunes y la documentación de actualización correspondiente:

- Amazon VPC CNI: para ver la versión recomendada del complemento Amazon VPC CNI para cada versión del clúster, consulte Actualización del complemento CNI de Amazon VPC para el complemento autogestionado de Kubernetes. Cuando se instala como un complemento de Amazon EKS, solo se puede actualizar una versión secundaria a la vez.
- kube-proxy: consulte Actualización del complemento autogestionable kube-proxy de Kubernetes.
- CoreDNS: consulte Actualización del complemento autogestionado de CoreDNS.
- Controlador de equilibrio de carga de AWS: el controlador de equilibrio de carga de AWS debe ser compatible con la versión de EKS que haya implementado. Consulte la guía de instalación para obtener más información.
- Controlador de la interfaz de almacenamiento de contenedores (CSI) de Amazon Elastic Block Store (Amazon EBS): para obtener información sobre la instalación y la actualización, consulte Administración del controlador CSI de Amazon EBS como complemento de Amazon EKS.

- Controlador de la interfaz de almacenamiento de contenedores (CSI) de Amazon Elastic File System (Amazon EFS): para obtener información sobre la instalación y la actualización, consulte el controlador CSI de Amazon EFS.
- Servidor de métricas de Kubernetes: para obtener más información, consulte metrics-server en. GitHub
- Kubernetes Cluster Autoscaler: para actualizar la versión de Kubernetes Cluster Autoscaler, cambia la versión de la imagen en la implementación. El escalador automático de clústeres está estrechamente vinculado al programador de Kubernetes. Siempre tendrás que actualizarlo cuando actualices el clúster. Revisa las GitHub versiones para encontrar la dirección de la última versión correspondiente a tu versión secundaria de Kubernetes.
- Karpenter: Para obtener información sobre la instalación y la actualización, consulta la documentación de Karpenter.

> **Tip**
>
> No tiene que actualizar manualmente ninguna de las capacidades de Amazon EKS Auto Mode, incluidas las capacidades de escalado automático de cómputo, almacenamiento en bloques y equilibrio de carga.

Compruebe los requisitos básicos de EKS antes de realizar la actualización

AWS requiere determinados recursos en su cuenta para completar el proceso de actualización. Si estos recursos no están presentes, el clúster no se puede actualizar. Una actualización del plano de control requiere los siguientes recursos:

1. Direcciones IP disponibles: Amazon EKS necesita hasta cinco direcciones IP disponibles de las subredes que especificó al crear el clúster para poder actualizarlo. Si no es así, actualice la configuración del clúster para incluir nuevas subredes del clúster antes de actualizar la versión.
2. Función de IAM de EKS: la función de IAM del plano de control sigue presente en la cuenta con los permisos necesarios.
3. Si su clúster tiene habilitado el cifrado secreto, asegúrese de que el rol de IAM del clúster tenga permiso para usar la clave de AWS Key Management Service (AWS KMS).

Compruebe las direcciones IP disponibles

Para actualizar el clúster, Amazon EKS requiere hasta cinco direcciones IP disponibles de las subredes que se especificaron cuando creó el clúster.

Para comprobar que las subredes tienen direcciones IP suficientes para actualizar el clúster, puede ejecutar el siguiente comando:

```
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 |
+--------------------------+--------------+-------+
```

El asistente de métricas del CNI de la VPC se puede utilizar para crear un panel CloudWatch de control para las métricas de la VPC. Amazon EKS recomienda actualizar las subredes del clúster mediante la API «UpdateClusterConfiguration» antes de iniciar una actualización de la versión de Kubernetes si se está quedando sin direcciones IP en las subredes especificadas inicialmente durante la creación del clúster. Compruebe que las nuevas subredes que se le proporcionarán son las siguientes:

- pertenecen al mismo conjunto de los AZs que se seleccionaron durante la creación del clúster.
- pertenecen a la misma VPC proporcionada durante la creación del clúster

Considere la posibilidad de asociar bloques CIDR adicionales si se agotan las direcciones IP del bloque CIDR de VPC existente. AWS permite asociar bloques de CIDR adicionales con su VPC de clúster existente, lo que amplía de forma efectiva su conjunto de direcciones IP. Esta expansión se puede lograr mediante la introducción de rangos de IP privadas adicionales (RFC 1918) o, si es necesario, rangos de IP públicas (distintos de la RFC 1918). Debe añadir nuevos bloques de CIDR

de VPC y permitir que se complete la actualización de la VPC antes de que Amazon EKS pueda utilizar el nuevo CIDR. Después de eso, puede actualizar las subredes en función de los bloques CIDR recién configurados en la VPC.

Compruebe el rol de IAM de EKS

Para comprobar que el rol de IAM está disponible y tiene la política de asunción de roles correcta en tu cuenta, puedes ejecutar los siguientes comandos:

```
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"
        }
    ]
}
```

Migre a los complementos de EKS

Amazon EKS instala automáticamente complementos como el complemento CNI de Amazon VPC para Kubernetes y CoreDNS para cada `kube-proxy` clúster. Los complementos pueden autogestionarse o instalarse como complementos de Amazon EKS. Amazon EKS Add-ons es una forma alternativa de administrar los complementos mediante la API EKS.

Puede usar los complementos de Amazon EKS para actualizar las versiones con un solo comando. Por ejemplo:

```
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
```

Compruebe si tiene algún complemento de EKS con:

```
aws eks list-addons --cluster-name <cluster name>
```

> **Warning**
>
> Los complementos de EKS no se actualizan automáticamente durante una actualización del plano de control. Debe iniciar las actualizaciones de los complementos de EKS y seleccionar la versión deseada.

- Usted es responsable de seleccionar una versión compatible entre todas las versiones disponibles. Consulta la guía sobre la compatibilidad de las versiones complementarias.
- Los complementos de Amazon EKS solo se pueden actualizar una versión secundaria a la vez.

Obtenga más información sobre los componentes que están disponibles como complementos de EKS y cómo empezar.

Aprenda a proporcionar una configuración personalizada a un complemento de EKS.

Identifique y corrija el uso de API eliminado antes de actualizar el plano de control

Debe identificar el uso de la API si se ha eliminado APIs antes de actualizar su plano de control de EKS. Para ello, recomendamos utilizar herramientas que puedan comprobar un clúster en ejecución o archivos de manifiesto de Kubernetes renderizados y estáticos.

La comprobación de los archivos de manifiesto estáticos suele ser más precisa. Si se ejecutan en clústeres activos, estas herramientas pueden arrojar falsos positivos.

Una API de Kubernetes obsoleta no significa que se haya eliminado. Deberías consultar la Política de obsolescencia de Kubernetes para entender cómo afecta la eliminación de la API a tus cargas de trabajo.

Información sobre clústeres

Cluster Insights es una función que proporciona información sobre problemas que pueden afectar a la capacidad de actualizar un clúster de EKS a versiones más recientes de Kubernetes. Amazon EKS selecciona y gestiona estos hallazgos y ofrece recomendaciones sobre cómo solucionarlos. Al aprovechar Cluster Insights, puede minimizar el esfuerzo dedicado a actualizar a versiones más recientes de Kubernetes.

Para ver información sobre un clúster de EKS, puede ejecutar el comando:

```
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."
        }
    ]
}
```

Para obtener un resultado más descriptivo sobre la información recibida, puede ejecutar el comando:

```
aws eks describe-insight --region <region-code> --id <insight-id> --cluster-name <my-
cluster>
```

También tiene la opción de ver información en la consola Amazon EKS. Tras seleccionar el clúster de la lista de clústeres, los resultados de la información se encuentran en la `Upgrade Insights` pestaña.

Si encuentra información sobre el clúster`"status": ERROR`, debe abordar el problema antes de realizar la actualización del clúster. Ejecute el `aws eks describe-insight` comando que incluirá los siguientes consejos de corrección:

Recursos afectados:

```
"resources": [
      {
        "insightStatus": {
          "status": "ERROR"
        },
        "kubernetesResourceUri": "/apis/policy/v1beta1/podsecuritypolicies/null"
      }
]
```

APIs en desuso:

```
"deprecationDetails": [
      {
        "usage": "/apis/flowcontrol.apiserver.k8s.io/v1beta2/flowschemas",
        "replacedWith": "/apis/flowcontrol.apiserver.k8s.io/v1beta3/flowschemas",
        "stopServingVersion": "1.29",
        "clientStats": [],
        "startServingReplacementVersion": "1.26"
      }
]
```

Acción recomendada a tomar:

```
"recommendation": "Update manifests and API clients to use newer Kubernetes APIs if
 applicable before upgrading to Kubernetes v1.26."
```

El uso de información sobre los clústeres a través de la consola de EKS o la CLI ayuda a acelerar el proceso de actualización correcta de las versiones de los clústeres de EKS. Obtenga más información con los siguientes recursos: * Blog de lanzamiento oficial de EKS Docs * Cluster Insights.

K. ube-no-trouble

K ube-no-trouble es una utilidad de línea de comandos de código abierto con el comando`kubent`. Si lo ejecutas `kubent` sin ningún argumento, utilizará tu KubeConfig contexto actual, escaneará el clúster e imprimirá un informe con lo que APIs quede obsoleto y se eliminará.

```
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)
```

También se puede usar para escanear archivos de manifiestos estáticos y paquetes Helm. Se recomienda ejecutarlo `kubent` como parte de un proceso de integración continua (CI) para identificar los problemas antes de implementar los manifiestos. El análisis de los manifiestos también es más preciso que el análisis de los clústeres activos.

Kube-no-trouble proporciona un ejemplo de cuenta de servicio y rol con los permisos adecuados para escanear el clúster.

Plutón

Otra opción es pluto, que es similar a ella `kubent` porque permite escanear un clúster activo, archivos de manifiesto, gráficos de control y tiene una GitHub acción que puedes incluir en tu proceso de CI.

```
pluto detect-all-in-cluster

NAME             KIND                VERSION          REPLACEMENT   REMOVED
 DEPRECATED   REPL AVAIL
eks.privileged   PodSecurityPolicy   policy/v1beta1                 false     true
    true
```

Recursos

Para comprobar que tu clúster no esté en desuso APIs antes de la actualización, debes supervisar lo siguiente:

- métrica `apiserver_requested_deprecated_apis` desde 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
```

- eventos en los registros de auditoría configurados en: `k8s.io/deprecated 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
```

Lo que generará líneas si APIs están en uso si están en desuso:

```
{
    "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\"}}"
            },
[...]
```

Actualice las cargas de trabajo de Kubernetes. Usa kubectl-convert para actualizar los manifiestos

Una vez que hayas identificado qué cargas de trabajo y manifiestos deben actualizarse, es posible que tengas que cambiar el tipo de recurso de tus archivos de manifiesto (por ejemplo, a). PodSecurityPolicies PodSecurityStandards Para ello, será necesario actualizar la especificación del recurso y realizar más investigaciones en función del recurso que se sustituya.

Si el tipo de recurso sigue siendo el mismo pero es necesario actualizar la versión de la API, puedes usar el `kubectl-convert` comando para convertir automáticamente los archivos de manifiesto. Por ejemplo, para convertir una implementación anterior en`apps/v1`. Para obtener más información, consulta Instalar el complemento kubectl convert en el sitio web de Kubernetes.

```
kubectl-convert -f <file> --output-version <group>/<version>
```

Configure PodDisruptionBudgets y garantice topologySpreadConstraints la disponibilidad de sus cargas de trabajo mientras se actualiza el plano de datos

Asegúrese de que sus cargas de trabajo sean las adecuadas PodDisruptionBudgetsy topologySpreadConstraintsde que estén disponibles mientras se actualiza el plano de datos. No todas las cargas de trabajo requieren el mismo nivel de disponibilidad, por lo que debe validar la escala y los requisitos de su carga de trabajo.

Asegúrese de que las cargas de trabajo estén distribuidas en varias zonas de disponibilidad y en varios hosts con distribuciones de topología para aumentar el nivel de confianza de que las cargas de trabajo migrarán al nuevo plano de datos de forma automática y sin incidentes.

Este es un ejemplo de una carga de trabajo que siempre tendrá el 80% de las réplicas disponibles y distribuirá las réplicas entre zonas y hosts

```
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 ha incorporado Amazon Elastic Kubernetes Service (Amazon EKS) como recurso compatible. Resilience Hub proporciona un lugar único para definir, validar y realizar un seguimiento de la resiliencia de sus aplicaciones, de modo que pueda evitar tiempos de inactividad innecesarios causados por interrupciones del software, la infraestructura o las operaciones.

Utilice Managed Node Groups o Karpenter para simplificar las actualizaciones de los planos de datos

Tanto Managed Node Groups como Karpenter simplifican las actualizaciones de los nodos, pero adoptan enfoques diferentes.

Los grupos de nodos gestionados automatizan el aprovisionamiento y la gestión del ciclo de vida de los nodos. Esto significa que puede crear nodos, actualizarlos automáticamente o terminarlos con una sola operación.

En la configuración predeterminada, Karpenter crea automáticamente nuevos nodos mediante la última AMI optimizada para EKS compatible. A medida que EKS publique la versión actualizada de EKS Optimized AMIs o se actualice el clúster, Karpenter empezará a utilizar estas imágenes automáticamente. Karpenter también implementa Node Expiry para actualizar los nodos.

Karpenter se puede configurar para que utilice el modo personalizado. AMIs Si utilizas la opción personalizada AMIs con Karpenter, eres responsable de la versión de kubelet.

Confirma la compatibilidad de la versión con los nodos existentes y el plano de control

Antes de proceder a la actualización de Kubernetes en Amazon EKS, es fundamental garantizar la compatibilidad entre los grupos de nodos gestionados, los nodos autogestionados y el plano de control. La compatibilidad viene determinada por la versión de Kubernetes que utilice y varía en función de los distintos escenarios. Tácticas:

- Kubernetes v1.28+ — * * A partir de la versión 1.28 de Kubernetes y posteriores, existe una política de versiones más flexible para los componentes principales. En concreto, el sesgo

admitido entre el servidor API de Kubernetes y el kubelet se ha ampliado con una versión secundaria, pasando de la n-2 a la n-3. Por ejemplo, si la versión del plano de control de EKS es la 1.28, puedes usar de forma segura versiones de kubelet tan antiguas como la 1.25. Este sesgo de versiones es compatible con AWS Fargate, los grupos de nodos administrados y los nodos autoadministrados. Recomendamos encarecidamente conservar las versiones de Amazon Machine Image (AMI) up-to-date por motivos de seguridad. Las versiones antiguas de kubelet pueden suponer riesgos de seguridad debido a las posibles vulnerabilidades y exposiciones comunes (CVEs), que podrían superar las ventajas de utilizar versiones antiguas de kubelet.

- Kubernetes < v1.28: si utilizas una versión anterior a la v1.28, el sesgo admitido entre el servidor API y el kubelet es n-2. Por ejemplo, si tu versión de EKS es la 1.27, la versión más antigua de kubelet que puedes usar es la 1.25. Este sesgo de versiones se aplica a AWS Fargate, a los grupos de nodos administrados y a los nodos autoadministrados.

Habilite la caducidad de los nodos administrados por Karpenter

Una forma en que Karpenter implementa las actualizaciones de los nodos es utilizando el concepto de caducidad de los nodos. Esto reduce la planificación necesaria para las actualizaciones de los nodos. Cuando estableces un valor para ttlSecondsUntilExpirado en tu aprovisionador, se activa la caducidad del nodo. Cuando los nodos alcanzan la antigüedad definida en segundos, se vacían y eliminan de forma segura. Esto es cierto incluso si están en uso, lo que te permite reemplazar los nodos por instancias actualizadas recién aprovisionadas. Cuando se reemplaza un nodo, Karpenter utiliza la última versión optimizada para EKS. AMIs Para obtener más información, consulte Desaprovisionamiento en el sitio web de Karpenter.

Karpenter no añade automáticamente la fluctuación a este valor. Para evitar una interrupción excesiva de la carga de trabajo, defina un presupuesto para las interrupciones de los módulos, tal y como se muestra en la documentación de Kubernetes.

Si configuras ttlSecondsUntilExpired en un aprovisionador, esto se aplica a los nodos existentes asociados al aprovisionador.

Utilice la función Drift para los nodos gestionados por Karpenter

La función Drift de Karpenter puede actualizar automáticamente los nodos aprovisionados por Karpenter para que permanezcan sincronizados con el plano de control del EKS. Actualmente, Karpenter Drift debe habilitarse mediante una puerta de funciones. La configuración predeterminada

de Karpenter utiliza la última AMI optimizada para EKS para la misma versión principal y secundaria que el plano de control del clúster EKS.

Una vez completada la actualización del clúster de EKS, la función Drift de Karpenter detectará que los nodos aprovisionados por Karpenter utilizan la versión de clúster anterior optimizada AMIs para EKS y acordonará, drenará y sustituirá automáticamente esos nodos. Para facilitar el traslado de los pods a nuevos nodos, siga las mejores prácticas de Kubernetes: establezca las cuotas de recursos de los módulos adecuadas y utilice los presupuestos de interrupción de los módulos (PDB). El desaprovisionamiento de Karpenter activará previamente los nodos de reemplazo en función de las solicitudes de recursos de los módulos y respetará las condiciones al desaprovisionar los nodos. PDBs

Usa eksctl para automatizar las actualizaciones de los grupos de nodos autogestionados

Los grupos de nodos autogestionados son EC2 instancias que se implementaron en su cuenta y se adjuntaron al clúster fuera del servicio EKS. Por lo general, se implementan y administran mediante algún tipo de herramienta de automatización. Para actualizar los grupos de nodos autogestionados, consulte la documentación de sus herramientas.

Por ejemplo, eksctl permite eliminar y drenar nodos autogestionados.

Algunas herramientas comunes incluyen:

- eksctl
- KOPs
- Planos de EKS

Backup del clúster antes de la actualización

Las nuevas versiones de Kubernetes introducen cambios importantes en su clúster de Amazon EKS. Después de actualizar un clúster, no puede degradarlo.

Velero es una herramienta de código abierto respaldada por la comunidad que se puede utilizar para realizar copias de seguridad de los clústeres existentes y aplicarlas a un clúster nuevo.

Tenga en cuenta que solo puede crear nuevos clústeres para las versiones de Kubernetes actualmente compatibles con EKS. Si la versión que su clúster está ejecutando actualmente sigue

siendo compatible y se produce un error en la actualización, puede crear un clúster nuevo con la versión original y restaurar el plano de datos. Tenga en cuenta que los recursos de AWS, incluido IAM, no están incluidos en la copia de seguridad de Velero. Sería necesario volver a crear estos recursos.

Reinicie los despliegues de Fargate después de actualizar el plano de control

Para actualizar los nodos del plano de datos de Fargate, debe volver a implementar las cargas de trabajo. Puede identificar qué cargas de trabajo se ejecutan en los nodos de Fargate enumerando todos los pods con la opción. `-o wide` `fargate-`Será necesario volver a implementar en el clúster cualquier nombre de nodo que comience por.

Evalúe Blue/Green los clústeres como una alternativa a las actualizaciones de clústeres in situ

Algunos clientes prefieren utilizar una estrategia de blue/green actualización. Esto puede tener ventajas, pero también incluye desventajas que deben tenerse en cuenta.

Los beneficios incluyen:

- Es posible cambiar varias versiones de EKS a la vez (por ejemplo, de la 1.23 a la 1.25)
- Capaz de volver al clúster anterior
- Crea un nuevo clúster que se puede administrar con sistemas más nuevos (por ejemplo, terraform)
- Las cargas de trabajo se pueden migrar de forma individual

Algunas desventajas incluyen:

- El punto final de la API y el OIDC cambian, lo que requiere actualizar a los consumidores (por ejemplo, kubectl y CI/CD)
- Requiere que se ejecuten 2 clústeres en paralelo durante la migración, lo que puede resultar caro y limitar la capacidad de la región
- Se necesita una mayor coordinación si las cargas de trabajo dependen unas de otras para migrar juntas
- Los balanceadores de carga y el DNS externo no pueden abarcar fácilmente varios clústeres

Si bien esta estrategia es posible, es más costosa que una actualización local y requiere más tiempo para la coordinación y las migraciones de la carga de trabajo. Puede ser necesario en algunas situaciones y debe planificarse cuidadosamente.

Con altos grados de automatización y sistemas declarativos como estos GitOps, esto puede ser más fácil de hacer. Deberá tomar precauciones adicionales para las cargas de trabajo con estado, de modo que se hagan copias de seguridad de los datos y se migren a nuevos clústeres.

Consulte estas publicaciones de blog para obtener más información:

- Actualización del clúster de Kubernetes: la estrategia de despliegue azul-verde
- Migración de clústeres EKS de Amazon EKS a Azul/Green o Canary para cargas de trabajo ArgoCD sin estado

Realice un seguimiento de los principales cambios planificados en el proyecto Kubernetes: piense en el futuro

No se limite a mirar la próxima versión. Revisa las nuevas versiones de Kubernetes a medida que se publiquen e identifica los cambios más importantes. Por ejemplo, algunas aplicaciones utilizaban directamente la API de Docker y en Kubernetes se eliminó la compatibilidad con la Container Runtime Interface (CRI) para Docker (también conocida como Dockershim). `1.24` Prepararse para este tipo de cambio requiere más tiempo.

Revisa todos los cambios documentados de la versión a la que vas a actualizar y toma nota de los pasos de actualización necesarios. Además, tenga en cuenta los requisitos o procedimientos específicos de los clústeres gestionados por Amazon EKS.

- Registro de cambios de Kubernetes

Guía específica sobre la eliminación de funciones

Eliminación de Dockershim en la versión 1.25: utilice un detector para Docker Socket (DDS)

La AMI optimizada de EKS para 1.25 ya no incluye soporte para Dockershim. Si depende de Dockershim, por ejemplo, si va a montar el socket Docker, tendrá que eliminar esas dependencias antes de actualizar sus nodos de trabajo a 1.25.

Busca instancias en las que dependas del socket de Docker antes de actualizar a la versión 1.25. Te recomendamos usar Detector for Docker Socket (DDS), un complemento de kubectl. .

Eliminación de la PodSecurityPolicy versión 1.25: migración a los estándares de seguridad de Pod o a una policy-as-code solución

`PodSecurityPolicy`quedó obsoleto en Kubernetes 1.21 y se eliminó en Kubernetes 1.25. Si lo usa PodSecurityPolicy en su clúster, debe migrar a los estándares de seguridad (PSS) integrados de Kubernetes Pod o a una policy-as-code solución antes de actualizar el clúster a la versión 1.25 para evitar interrupciones en sus cargas de trabajo.

AWS publicó una sección de preguntas frecuentes detallada en la documentación de EKS.

Revise los estándares de seguridad de los Pod (PSS) y las prácticas recomendadas para la admisión de los Pod Security Admission (PSA).

Consulta la entrada del blog PodSecurityPolicy Deprecation en el sitio web de Kubernetes.

El controlador de almacenamiento integrado en árbol quedó obsoleto en la versión 1.23: migración a los controladores de la interfaz de almacenamiento en contenedores (CSI)

La interfaz de almacenamiento en contenedores (CSI) se diseñó para ayudar a Kubernetes a reemplazar sus mecanismos actuales de controladores de almacenamiento en árbol. La característica de migración de la interfaz de almacenamiento de contenedores (CSI) de Amazon EBS está habilitada de forma predeterminada en Amazon EKS `1.23` y los clústeres posteriores. Si tienes pods que se ejecutan en una versión `1.22` o anterior de un clúster, debes instalar el controlador CSI de Amazon EBS antes de actualizar el clúster a la versión `1.23` para evitar la interrupción del servicio.

Consulte las preguntas frecuentes sobre la migración de Amazon EBS CSI.

Recursos adicionales

ClowdHaus Guía de actualización de EKS

ClowdHaus EKS Upgrade Guidance es una CLI que ayuda a actualizar los clústeres de Amazon EKS. Puede analizar un clúster en busca de posibles problemas que puedan solucionarse antes de la actualización.

GoNoGo

GoNoGoes una herramienta de fase intermedia para determinar la fiabilidad de las actualizaciones de los complementos del clúster.

Mejores prácticas para la optimización de costos

La optimización de costos consiste en lograr los resultados de su negocio al precio más bajo. Si sigue la documentación de esta guía, optimizará sus cargas de trabajo de Amazon EKS.

Directrices generales

En la nube, hay una serie de pautas generales que pueden ayudarlo a optimizar los costos de sus microservicios:

- Asegúrese de que las cargas de trabajo que se ejecutan en Amazon EKS sean independientes de los tipos de infraestructura específicos para ejecutar sus contenedores, ya que esto le proporcionará una mayor flexibilidad a la hora de ejecutarlas en los tipos de infraestructura menos costosos. Al usar Amazon EKS con EC2, puede haber excepciones cuando tenemos cargas de trabajo que requieren tipos específicos de EC2 instancias, como una GPU u otros tipos de instancias, debido a la naturaleza de la carga de trabajo.
- Seleccione instancias de contenedores con un perfil óptimo: cree un perfil de sus entornos de producción o preproducción y supervise las métricas críticas, como la CPU y la memoria, mediante servicios como Amazon CloudWatch Container Insights para Amazon EKS o herramientas de terceros que están disponibles en el ecosistema de Kubernetes. Esto garantizará que podamos asignar la cantidad correcta de recursos y evitar el desperdicio de recursos.
- Aproveche las diferentes opciones de compra disponibles en AWS para ejecutar EKS con EC2, por ejemplo, On-Demand, Spot y Savings Plan.

Mejores prácticas de optimización de costos de EKS

Existen tres áreas de mejores prácticas generales para la optimización de costos en la nube:

- Recursos rentables (Auto Scaling, Down Scaling, políticas y opciones de compra)
- Conocimiento de los gastos (mediante AWS y herramientas de terceros)
- Optimización a lo largo del tiempo (tamaño adecuado)

Como ocurre con cualquier guía, hay ventajas y desventajas. Asegúrese de trabajar con su organización para comprender las prioridades de esta carga de trabajo y cuáles son las mejores prácticas más importantes.

Cómo usar esta guía

Esta guía está destinada a los equipos de DevOps responsables de implementar y administrar los clústeres de EKS y las cargas de trabajo que soportan. La guía está organizada en diferentes áreas de mejores prácticas para facilitar su uso. Cada tema incluye una lista de recomendaciones, herramientas que se pueden utilizar y prácticas recomendadas para la optimización de los costes de los clústeres de EKS. No es necesario leer los temas en un orden determinado.

Características clave de los servicios de AWS y Kubernetes

La optimización de costes está respaldada por los siguientes servicios y características de AWS:

- EC2 Tipos de instancias, Savings Plan (e instancias reservadas) e instancias puntuales, a precios diferentes.
- Auto Scaling junto con las políticas de Auto Scaling nativas de Kubernetes. Considere Savings Plan (instancias anteriormente reservadas) para cargas de trabajo predecibles. Utilice almacenes de datos gestionados, como EBS y EFS, para aumentar la elasticidad y durabilidad de los datos de la aplicación.
- El panel de control de la consola Billing and Cost Management, junto con AWS Cost Explorer, proporcionan información general sobre su uso de AWS. Utilice AWS Organizations para obtener detalles de facturación detallados. También se han compartido los detalles de varias herramientas de terceros.
- Amazon CloudWatch Container Metrics proporciona métricas sobre el uso de los recursos por parte del clúster de EKS. Además del panel de control de Kubernetes, hay varias herramientas en el ecosistema de Kubernetes que se pueden utilizar para reducir el desperdicio.

Esta guía incluye un conjunto de recomendaciones que puede utilizar para mejorar la optimización de costes de su clúster de Amazon EKS.

Comentarios

Esta guía se publica el GitHub día de hoy para recopilar comentarios y sugerencias directos de la comunidad más amplia de EKS/Kubernetes. Si tienes alguna práctica recomendada que consideres que deberíamos incluir en la guía, presenta un problema o envía un PR al repositorio. GitHub Nuestra intención es actualizar la guía periódicamente a medida que se añadan nuevas funciones al servicio o cuando surja una nueva práctica recomendada.

Marco de optimización de costos

La economía de la nube de AWS es una disciplina que ayuda a los clientes a aumentar la eficiencia y reducir sus costos mediante la adopción de tecnologías informáticas modernas como Amazon EKS. La disciplina recomienda seguir una metodología denominada «marco de gestión financiera en la nube (CFM)», que consta de 4 pilares:



El pilar See: medición y rendición de cuentas

El pilar See es un conjunto fundamental de actividades y tecnologías que definen cómo medir, monitorear y generar responsabilidad por el gasto en la nube. Suele denominarse «observabilidad», «instrumentación» o «telemetría». Las capacidades y limitaciones de la infraestructura de «observabilidad» determinan qué se puede optimizar. Obtener una imagen clara de sus costos es un primer paso fundamental en la optimización de costos, ya que necesita saber desde dónde empezar. Este tipo de visibilidad también servirá de guía para los tipos de actividades que necesitará realizar para optimizar aún más su entorno.

He aquí un breve resumen de nuestras mejores prácticas para el pilar See:

- Defina y mantenga una estrategia de etiquetado para sus cargas de trabajo.
  - Utilice el etiquetado de instancias, ya que etiquetar los clústeres de EKS le permite ver los costos individuales de los clústeres y asignarlos en sus informes de costos y uso.
- Establezca informes y supervise el uso de EKS mediante tecnologías como Kubecost.

  - Habilite los paneles de inteligencia en la nube. Al etiquetar los recursos correctamente y utilizar visualizaciones, podrá medir y estimar los costos.
- Asigne los costos de la nube a las aplicaciones, las líneas de negocio (LoBs) y las fuentes de ingresos.
- Defina, mida y comuníquese efficiency/value KPIs con las partes interesadas de la empresa. Por ejemplo, crea un KPI «métrico unitario» que mida el coste por transacción; por ejemplo, un servicio de transporte compartido podría tener un KPI que indique el «coste por viaje».

Para obtener más información sobre las tecnologías y actividades recomendadas relacionadas con este pilar, consulta la sección Optimización de costes: observabilidad de esta guía.

El pilar del ahorro: optimización de costes

Este pilar se basa en las tecnologías y capacidades desarrolladas en el pilar «Ver». Las siguientes actividades suelen encuadrarse en este pilar:

- Identifique y elimine los residuos en su entorno.
- Diseñe y diseñe para lograr una rentabilidad rentable.
- Elija la mejor opción de compra, por ejemplo, instancias bajo demanda o instancias puntuales.
- Adáptese a la evolución de los servicios: a medida que evolucionen los servicios de AWS, la forma de utilizarlos de manera eficiente puede cambiar. Esté dispuesto a adaptarse para tener en cuenta estos cambios.

Dado que estas actividades son operativas, dependen en gran medida de las características de su entorno. Pregúntese cuáles son los principales impulsores de los costos. ¿Qué valor empresarial ofrecen sus diferentes entornos? ¿Qué opciones de compra e infraestructura, por ejemplo, los tipos de familias de instancias, son las más adecuadas para cada entorno?

A continuación, se muestra una lista priorizada de los factores de costo más comunes de los clústeres de EKS:

1. Costes informáticos: es necesario analizar detenidamente la combinación de varios tipos de familias de instancias, las opciones de compra y el equilibrio entre la escalabilidad y la disponibilidad. Para obtener más información, consulta las recomendaciones de la sección Optimización de costos: procesamiento de esta guía.

2. Costos de red: el uso de 3 AZs para los clústeres EKS puede aumentar potencialmente los costos de tráfico entre zonas de disponibilidad. Para obtener nuestras recomendaciones sobre cómo equilibrar los requisitos de alta disponibilidad y mantener bajos los costos de tráfico de red, consulte la sección Optimización de costos: redes de esta guía.
3. Costes de almacenamiento: en función de la stateful/stateless naturaleza de las cargas de trabajo en los clústeres de EKS y de cómo se utilicen los distintos tipos de almacenamiento, el almacenamiento puede considerarse parte de la carga de trabajo. Para obtener información sobre los costes de almacenamiento de EKS, consulte la sección Optimización de costes: almacenamiento de esta guía.

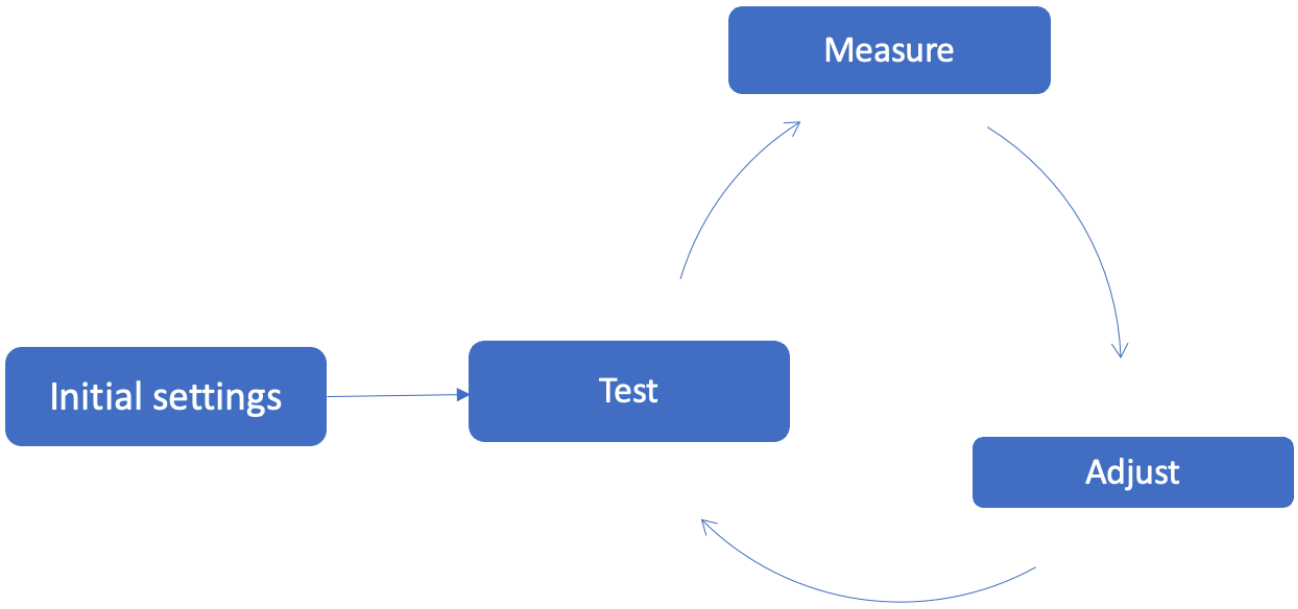
El pilar del plan: planificación y previsión

Una vez implementadas las recomendaciones del pilar See, los clústeres se optimizan de forma continua. A medida que se vaya adquiriendo experiencia en la explotación eficiente de los clústeres, las actividades de planificación y previsión se pueden centrar en:

- Presupuestar y pronosticar los costos de la nube de forma dinámica.
- Cuantificación del valor empresarial que ofrecen los servicios de contenedores de EKS.
- Integración de la gestión de costes del clúster de EKS con la planificación de la gestión financiera de TI.

El pilar de la ejecución

La optimización de costes es un proceso continuo e implica una serie de mejoras incrementales:



Conseguir el patrocinio ejecutivo para este tipo de actividades es crucial para integrar la optimización de los clústeres de EKS en las «FinOps» iniciativas de la organización. Permite alinear a las partes interesadas mediante una comprensión compartida de los costos de los clústeres de EKS, la implementación de barreras de costos de los clústeres de EKS y la garantía de que las herramientas, la automatización y las actividades evolucionen en función de las necesidades de la organización.

Referencias

- Economía de la nube de AWS, gestión financiera de la nube

Conocimiento del gasto

El conocimiento de los gastos consiste en entender quién, dónde y qué es lo que provoca los gastos en su clúster de EKS. Obtener una imagen precisa de estos datos le ayudará a dar a conocer sus gastos y a destacar las áreas que debe corregir.

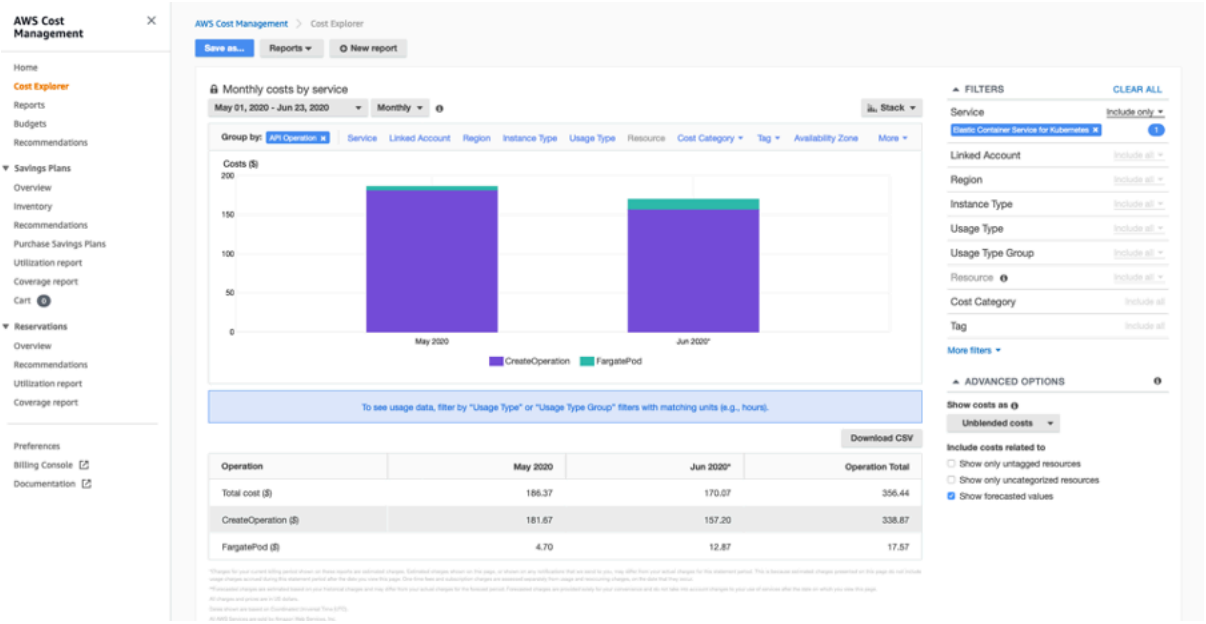
Recomendaciones

Utilice Cost Explorer

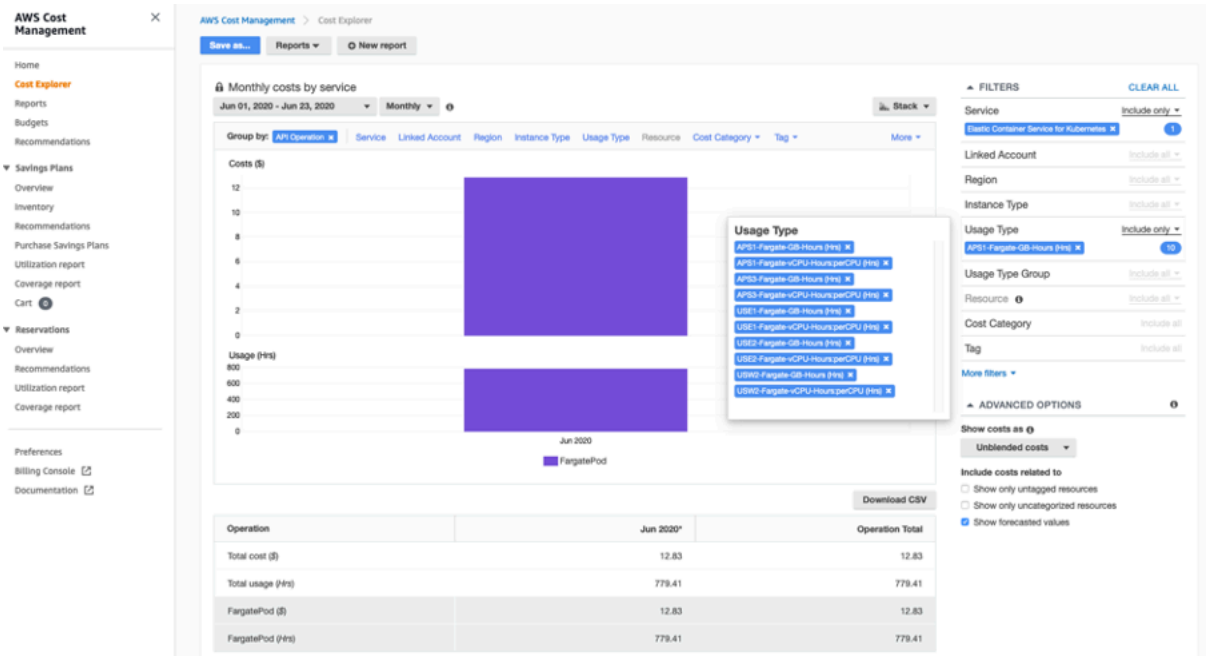
AWS Cost Explorer tiene una easy-to-use interfaz que le permite visualizar, comprender y administrar los costos y el uso de AWS a lo largo del tiempo. Puede analizar los datos de costo y uso en varios niveles mediante los filtros disponibles en Cost Explorer.

Costos del EKS Control Plane y del EKS Fargate

Con los filtros, podemos consultar los costos incurridos por los costos de EKS en el Plano de Control y el Fargate Pod, como se muestra en el siguiente diagrama:



Con los filtros, podemos consultar los costes totales incurridos por los Fargate Pods en todas las regiones de EKS, que incluyen tanto las horas de CPU virtual por CPU como los GB por hora, como se muestra en el siguiente diagrama:



Etiquetado de recursos

Amazon EKS admite la adición de etiquetas de AWS a los clústeres de Amazon EKS. Esto facilita el control del acceso a la API de EKS para administrar sus clústeres. Las etiquetas añadidas a un clúster de EKS son específicas del recurso del clúster de AWS EKS y no se propagan a otros recursos de AWS utilizados por el clúster, como EC2 instancias o balanceadores de carga. En la actualidad, el etiquetado de clústeres es compatible con todos los clústeres de EKS nuevos y existentes a través de la API, la consola y SDKs la consola de AWS.

AWS Fargate es una tecnología que proporciona capacidad informática a pedido y del tamaño adecuado para contenedores. Para poder programar pods en Fargate en el clúster, debe definir al menos un perfil de Fargate que especifique qué pods debe utilizar Fargate cuando se lancen.

Añadir y enumerar etiquetas a un clúster de EKS:

```
$ aws eks tag-resource --resource-arn arn:aws:eks:us-west-2:xxx:cluster/ekscluster1 --
tags team=devops,env=staging,bu=cio,costcenter=1234
$ aws eks list-tags-for-resource --resource-arn arn:aws:eks:us-west-2:xxx:cluster/
ekscluster1
{
    "tags": {
        "bu": "cio",
        "env": "staging",
        "costcenter": "1234",
        "team": "devops"
```

```
    }
}
```

Tras activar las etiquetas de asignación de costes en el explorador de costes de AWS, AWS las utiliza para organizar los costes de los recursos en el informe de asignación de costes, a fin de que le resulte más fácil categorizar y realizar un seguimiento de los costes de AWS.

Las etiquetas no tienen ningún significado semántico para Amazon EKS, por lo que se interpretan estrictamente como cadenas de caracteres. Por ejemplo, puede definir un conjunto de etiquetas para los clústeres de Amazon EKS a fin de ayudar a realizar un seguimiento del propietario y del nivel de pila de cada clúster.

Utilice AWS Trusted Advisor

AWS Trusted Advisor ofrece un amplio conjunto de recomendaciones y comprobaciones de prácticas recomendadas en cinco categorías: optimización de costes, seguridad, tolerancia a errores, rendimiento y límites de servicio.

Para optimizar los costos, Trusted Advisor ayuda a eliminar los recursos inactivos y no utilizados y recomienda comprometerse con la capacidad reservada. Las acciones clave que ayudarán a Amazon EKS se centrarán en las EC2 instancias poco utilizadas, las direcciones IP elásticas no asociadas, los balanceadores de carga inactivos y los volúmenes de EBS infrautilizados, entre otras cosas. La lista completa de comprobaciones se encuentra en -practice-checklist/. https:// aws.amazon.com/premiumsupport/ technology/trusted-advisor/best

The Trusted Advisor también ofrece recomendaciones sobre Savings Plans e Reserved EC2 Instances para instancias y Fargate, lo que le permite comprometerse a un monto de uso constante a cambio de tarifas con descuento.

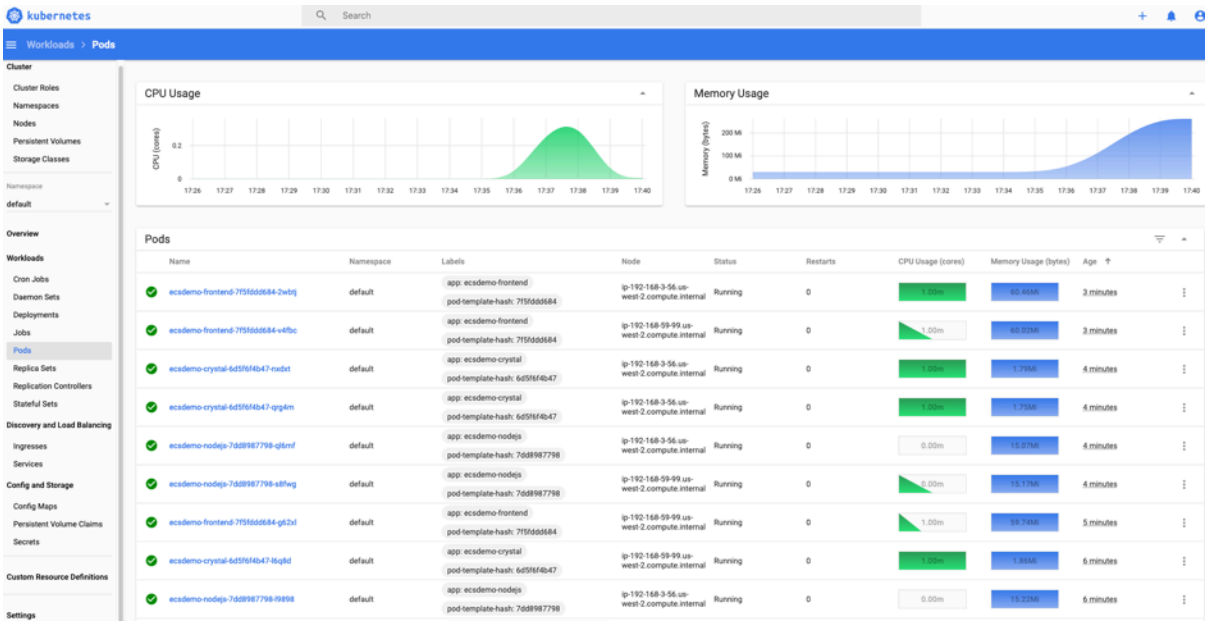
> **Note**
>
> Las recomendaciones de Trusted Advisor son recomendaciones genéricas y no específicas de EKS.

Utilice el panel de control de Kubernetes

Panel de control de Kubernetes

El panel de control de Kubernetes es una interfaz de usuario web de uso general para los clústeres de Kubernetes, que proporciona información sobre el clúster de Kubernetes, incluido el uso de los recursos a nivel de clúster, nodo y pod. La implementación del panel de control de Kubernetes en un clúster de Amazon EKS se describe en la documentación de Amazon EKS.

El panel proporciona desgloses del uso de los recursos para cada nodo y pod, así como metadatos detallados sobre los pods, los servicios, las implementaciones y otros objetos de Kubernetes. Esta información consolidada proporciona visibilidad de su entorno de Kubernetes.



kubectl: encabeza y describe los comandos

Ver las métricas de uso de recursos con los comandos kubectl top y kubectl describe. kubectl top mostrará el uso actual de CPU y memoria de los pods o nodos del clúster, o de un pod o nodo específico. El comando kubectl describe proporcionará información más detallada sobre un nodo o un pod específicos.

```
$ kubectl top pods
$ kubectl top nodes
$ kubectl top pod pod-name --namespace mynamespace --containers
```

Con el comando top, el resultado mostrará la cantidad total de CPU (en núcleos) y memoria (en MiB) que utiliza el nodo y los porcentajes de la capacidad asignable del nodo que representan esos números. A continuación, puedes ir al siguiente nivel, el nivel de contenedor dentro de los pods, añadiendo una marca --containers.

```
$ kubectl describe node <node>
$ kubectl describe pod <pod>
```

kubectl describe devuelve el porcentaje de la capacidad total disponible que representa cada solicitud o límite de recursos.

kubectl top y describe, realiza un seguimiento de la utilización y la disponibilidad de los recursos críticos, como la CPU, la memoria y el almacenamiento, en los pods, nodos y contenedores de Kubernetes. Esta toma de conciencia ayudará a comprender el uso de los recursos y a controlar los costos.

Utilice CloudWatch Container Insights

Utilice CloudWatch Container Insights para recopilar, agregar y resumir las métricas y los registros de sus aplicaciones y microservicios en contenedores. Container Insights está disponible para Amazon Elastic Kubernetes Service y para las plataformas Kubernetes EC2 en Amazon. EC2 Las métricas incluyen la utilización de recursos como CPU, memoria, disco y red.

La instalación de los conocimientos se detalla en la documentación.

CloudWatch crea métricas agregadas a nivel de clúster, nodo, módulo, tarea y servicio como CloudWatch métricas.

La siguiente consulta muestra una lista de nodos, ordenados por el uso medio de la CPU de los nodos

```
STATS avg(node_cpu_utilization) as avg_node_cpu_utilization by NodeName
| SORT avg_node_cpu_utilization DESC
```

Uso de CPU por nombre de contenedor

```
stats pct(container_cpu_usage_total, 50) as CPUPercMedian by kubernetes.container_name
| filter Type="Container"
```

Uso de disco por nombre de contenedor

```
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
```

Encontrará más ejemplos de consultas en la documentación de Container Insights

Esta toma de conciencia ayudará a comprender el uso de los recursos y a controlar los costos.

Uso de Kubecost para informar y orientar sobre los gastos

También se pueden implementar herramientas de terceros, como kubecost, en Amazon EKS para obtener visibilidad del costo de ejecutar su clúster de Kubernetes. Consulte este blog de AWS para hacer un seguimiento de los costes mediante Kubecost

Implementación de kubecost con 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:

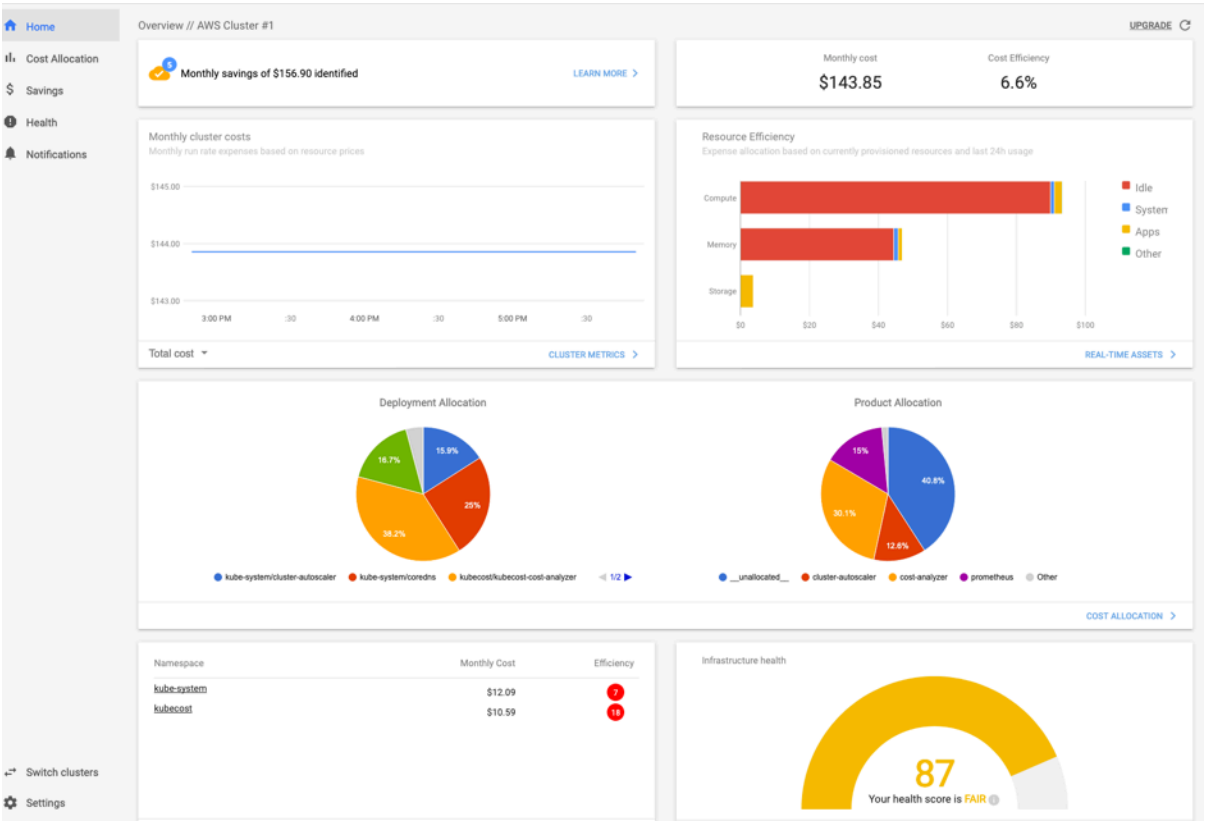
    kubectl port-forward --namespace kubecost deployment/kubecost-cost-analyzer 9090

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
```

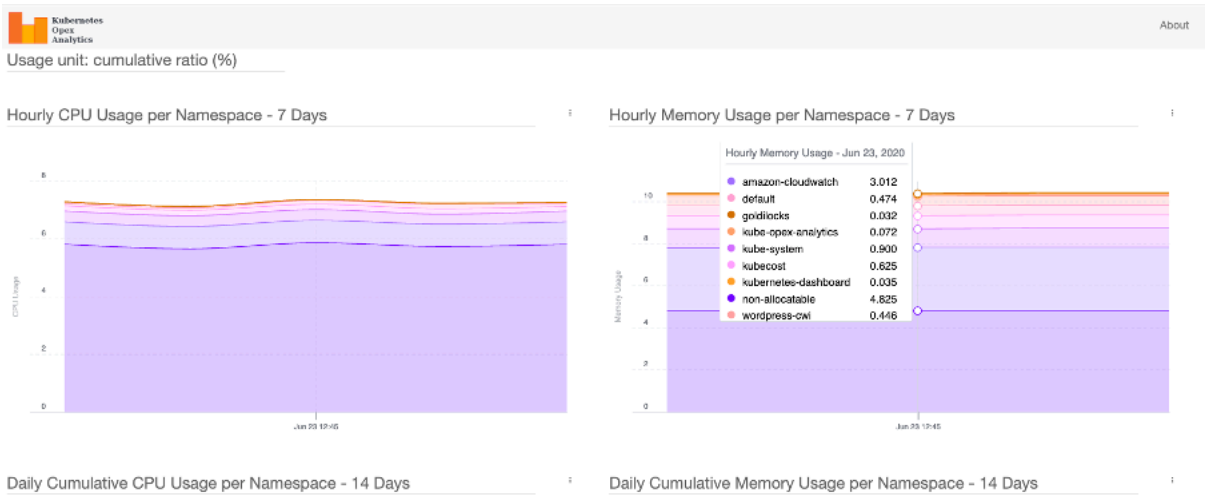
```
$ kubectl port-forward --namespace kubecost deployment/kubecost-cost-analyzer 8080:9090
```

Panel de control de Kubecost -



Utilice la herramienta de análisis de asignación de costes y planificación de la capacidad de Kubernetes

Kubernetes Opex Analytics es una herramienta que ayuda a las organizaciones a realizar un seguimiento de los recursos que consumen sus clústeres de Kubernetes para evitar pagar de más. Para ello, genera informes de uso a corto (7 días), medio (14 días) y largo (12 meses) que muestran información relevante sobre la cantidad de recursos que gasta cada proyecto a lo largo del tiempo.



Yotascale

Yotascale ayuda a asignar con precisión los costos de Kubernetes. La función de asignación de costes de Kubernetes de Yotascale utiliza datos de costes reales, que incluyen los descuentos en las instancias reservadas y los precios de las instancias puntuales, en lugar de estimaciones genéricas de los precios de mercado, para informar sobre la huella de costes total de Kubernetes

Puede encontrar más información en su sitio web.

Asesor Alcide

Alcide es un socio de tecnología avanzada de AWS Partner Network (APN). Alcide Advisor ayuda a garantizar que la configuración de sus clústeres, nodos y pods de Amazon EKS esté ajustada para que se ejecute de acuerdo con las mejores prácticas de seguridad y las directrices internas. Alcide Advisor es un servicio sin agentes para la auditoría y el cumplimiento de Kubernetes, diseñado para garantizar un DevSecOps flujo seguro y sin fricciones al reforzar la fase de desarrollo antes de pasar a la fase de producción.

Puedes encontrar más detalles en esta entrada de blog.

Otras herramientas

Recolección de basura de Kubernetes

La función del recolector de basura de Kubernetes es eliminar ciertos objetos que alguna vez tuvieron un propietario, pero que ya no lo tienen.

Condado de Fargate

Fargatecount es una herramienta útil que permite a los clientes de AWS realizar un seguimiento, con una CloudWatch métrica personalizada, del número total de pods de EKS que se han implementado en Fargate en una región específica de una cuenta específica. Esto ayuda a realizar un seguimiento de todos los módulos Fargate que se encuentran en un clúster EKS.

Popeye: un desinfectante de clústeres de Kubernetes

Popeye: un desinfectante de clústeres de Kubernetes es una utilidad que escanea un clúster de Kubernetes en tiempo real e informa sobre posibles problemas con los recursos y las configuraciones desplegados. Desinfecta el clúster en función de lo que está desplegado y no de lo que está en el disco. Al escanear el clúster, detecta los errores de configuración y ayuda a garantizar que se apliquen las mejores prácticas

Recursos

Consulte los siguientes recursos para obtener más información sobre las mejores prácticas para la optimización de costes.

Documentación y blogs

- Amazon EKS admite el etiquetado

Tools (Herramientas)

- ¿Qué es AWS Billing and Cost Management?
- Información sobre CloudWatch contenedores de Amazon
- Cómo realizar un seguimiento de los costes en clústeres de Amazon EKS multiusuario mediante Kubecost
- Kubecost
- Kube Opsview
- Análisis Opex de Kubernetes

Computación y escalado automático

Como desarrollador, harás estimaciones sobre las necesidades de recursos de tu aplicación, por ejemplo, de CPU y memoria, pero si no las ajustas continuamente, es posible que queden

anticuadas, lo que podría aumentar tus costes y empeorar el rendimiento y la fiabilidad. Es más importante ajustar continuamente los requisitos de recursos de una aplicación que hacerlos bien la primera vez.

Las mejores prácticas que se mencionan a continuación le ayudarán a crear y operar cargas de trabajo rentables que logren resultados empresariales y, al mismo tiempo, minimicen los costos y permitan a su organización maximizar el retorno de la inversión. Un orden de importancia de alto nivel para optimizar los costos de cómputo de los clústeres es:

1. Cargas de trabajo del tamaño correcto
2. Reduzca la capacidad no utilizada
3. Optimice los tipos de capacidad de cómputo (por ejemplo, Spot) y los aceleradores (por ejemplo GPUs)

Dimensione sus cargas de trabajo de forma adecuada

En la mayoría de los clústeres de EKS, la mayor parte del costo proviene de las EC2 instancias que se utilizan para ejecutar las cargas de trabajo en contenedores. No podrá dimensionar correctamente sus recursos informáticos sin comprender los requisitos de sus cargas de trabajo. Por este motivo, es esencial que utilice las solicitudes y los límites adecuados y que realice los ajustes necesarios en esos ajustes. Además, las dependencias, como el tamaño de la instancia y la selección del almacenamiento, pueden afectar al rendimiento de la carga de trabajo, lo que puede tener diversas consecuencias imprevistas en los costes y la fiabilidad.

Las solicitudes deben ajustarse a la utilización real. Si las solicitudes de un contenedor son demasiado altas, habrá capacidad no utilizada, lo que constituye un factor importante en los costos totales del clúster. Cada contenedor de un módulo, por ejemplo, una aplicación o un sidecar, debe tener sus propias solicitudes y límites establecidos para garantizar que los límites totales de los módulos sean lo más precisos posible.

Utilice herramientas como Goldilocks, KRR y Kubecost, que calculan las solicitudes de recursos y los límites de sus contenedores. Según la naturaleza de las aplicaciones, performance/cost los requisitos y la complejidad, debe evaluar qué métricas son las mejores para escalar, en qué punto se degrada el rendimiento de la aplicación (punto de saturación) y cómo ajustar las solicitudes y los límites en consecuencia. Consulte el artículo sobre el tamaño correcto de las aplicaciones para obtener más información sobre este tema.

Recomendamos usar el escalador automático de pods horizontales (HPA) para controlar cuántas réplicas de la aplicación deben ejecutarse, el escalador automático de pods verticales (VPA) para ajustar el número de solicitudes y los límites que la aplicación necesita por réplica, y un escalador automático de nodos como Karpenter o Cluster Autoscaler para ajustar continuamente el número total de nodos del clúster. Las técnicas de optimización de costos que utilizan Karpenter y Cluster Autoscaler se documentan en una sección posterior de este documento.

El escalador automático Vertical Pod puede ajustar las solicitudes y los límites asignados a los contenedores para que las cargas de trabajo se ejecuten de manera óptima. Debe ejecutar el VPA en modo de auditoría para que no realice cambios automáticamente ni reinicie los pods. Sugerirá cambios en función de las métricas observadas. Si se producen cambios que afecten a las cargas de trabajo de producción, primero debe revisarlos y probarlos en un entorno que no sea de producción, ya que pueden afectar a la fiabilidad y el rendimiento de la aplicación.

Reduzca el consumo

La mejor forma de ahorrar dinero es aprovisionar menos recursos. Una forma de hacerlo es ajustar las cargas de trabajo en función de sus requisitos actuales. Debe comenzar cualquier esfuerzo de optimización de costos asegurándose de que sus cargas de trabajo definan sus requisitos y escalen de forma dinámica. Para ello, será necesario obtener las métricas de las aplicaciones y configurar configuraciones (por ejemplo, Pod Readiness Gates) para garantizar que la aplicación pueda ampliarse y reducirse de forma dinámica y segura. `PodDisruptionBudgets` Es importante tener en cuenta que las restricciones PodDisruptionBudgets pueden impedir que Cluster Autoscaler y Karpenter reduzcan la escala de los nodos, ya que tanto Cluster Autoscaler como Karpenter respetan. PodDisruptionBudgets El valor «minAvailable» en el campo «minAvailable» siempre PodDisruptionBudget debe ser inferior al número de pods de la implementación y se debe mantener una buena separación entre ambos, por ejemplo, en un despliegue de 6 pods en el que desee tener un mínimo de 4 pods en funcionamiento en todo momento, establezca el valor «minAvailable» en 4. PodDisruptionBidget Esto permitirá que Cluster Autoscaler y Karpenter drenen y desalojen de forma segura los pods de los nodos infrautilizados durante un evento de reducción de escala de nodos. Consulte el documento de preguntas frecuentes sobre Cluster Autoscaler.

El escalador automático horizontal Pod es un escalador automático de cargas de trabajo flexible que puede ajustar el número de réplicas que se necesitan para cumplir con los requisitos de rendimiento y confiabilidad de su aplicación. Cuenta con un modelo flexible para definir cuándo escalar hacia arriba y hacia abajo en función de varias métricas, como la CPU o la memoria, o métricas personalizadas, por ejemplo, la profundidad de la cola, el número de conexiones a un pod, etc.

El Kubernetes Metrics Server permite el escalado en respuesta a métricas integradas, como el uso de la CPU y la memoria, pero si quieres escalar en función de otras métricas, como la profundidad de colas de Amazon CloudWatch o SQS, deberías considerar proyectos de escalado automático basados en eventos, como KEDA. Consulte esta entrada de blog sobre cómo utilizar KEDA con las métricas. CloudWatch Si no está seguro de qué métricas debe monitorizar y en qué base escalar, consulte las mejores prácticas de monitorización de las métricas más importantes.

Reducir el consumo de carga de trabajo crea un exceso de capacidad en un clúster y, con una configuración de escalado automático adecuada, puede reducir los nodos automáticamente y reducir el gasto total. Le recomendamos que no intente optimizar la capacidad de cómputo manualmente. El programador de Kubernetes y los escaladores automáticos de nodos se diseñaron para gestionar este proceso por ti.

Reduzca la capacidad no utilizada

Una vez que haya determinado el tamaño correcto de las aplicaciones y reduzca el exceso de solicitudes, puede empezar a reducir la capacidad informática aprovisionada. Debería poder hacerlo de forma dinámica si se ha tomado el tiempo de dimensionar correctamente sus cargas de trabajo según las secciones anteriores. Hay dos escaladores automáticos de nodos principales que se utilizan con Kubernetes en AWS.

Karpenter y Cluster Autoscaler

Tanto Karpenter como el escalador automático de clústeres de Kubernetes escalarán la cantidad de nodos del clúster a medida que se creen o eliminen los pods y cambien los requisitos de procesamiento. El objetivo principal de ambos es el mismo, pero Karpenter adopta un enfoque diferente para la administración de nodos, el aprovisionamiento y el desaprovisionamiento, lo que puede ayudar a reducir los costos y optimizar el uso en todo el clúster.

A medida que los clústeres aumentan de tamaño y la variedad de cargas de trabajo, se hace más difícil configurar previamente los grupos de nodos y las instancias. Al igual que ocurre con las solicitudes de carga de trabajo, es importante establecer un punto de referencia inicial y ajustarlo continuamente según sea necesario.

Si utiliza Cluster AutoScaler, respetará los valores «mínimo» y «máximo» de cada grupo de Auto Scaling (ASG) y solo ajustará el valor «deseado». Es importante prestar atención al establecer estos valores para el ASG subyacente, ya que Cluster Autoscaler no podrá reducir un ASG más allá de su recuento «mínimo». Defina el recuento «deseado» como el número de nodos que necesita durante el

horario laboral normal y «mínimo» como el número de nodos que necesita fuera del horario laboral. Consulte el documento de preguntas frecuentes sobre Cluster Autoscaler.

Expansor prioritario de escalador automático de clústeres

El escalador automático de clústeres de Kubernetes funciona escalando grupos de nodos (denominados grupo de nodos) hacia arriba y hacia abajo a medida que las aplicaciones se escalan hacia arriba y hacia abajo. Si no escala las cargas de trabajo de forma dinámica, el escalador automático de clústeres no le ayudará a ahorrar dinero. El escalador automático de clústeres requiere que un administrador de clústeres cree grupos de nodos con antelación para que las cargas de trabajo se consuman. Los grupos de nodos deben configurarse para usar instancias que tengan el mismo «perfil», es decir, aproximadamente la misma cantidad de CPU y memoria.

Puede tener varios grupos de nodos y el escalador automático de clústeres se puede configurar para establecer niveles de escalado prioritarios, y cada grupo de nodos puede contener nodos de diferentes tamaños. Los grupos de nodos pueden tener diferentes tipos de capacidad y el expansor de prioridad se puede usar para escalar primero los grupos menos costosos.

A continuación, se muestra un ejemplo de un fragmento de configuración de clúster que utiliza `ConfigMap` a para priorizar la capacidad reservada antes de usar instancias bajo demanda. Puede utilizar la misma técnica para dar prioridad a las instancias Graviton o puntual frente a otros tipos.

```
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.*
```

El uso de grupos de nodos puede ayudar a los recursos informáticos subyacentes a cumplir lo esperado de forma predeterminada, es decir, distribuir los nodos entre sí AZs, pero no todas las cargas de trabajo tienen los mismos requisitos o expectativas, por lo que es mejor dejar que las aplicaciones declaren sus requisitos de forma explícita. Para obtener más información sobre el escalador automático de clústeres, consulta la sección de prácticas recomendadas.

Desprogramador

El escalador automático de clústeres puede añadir y eliminar la capacidad de los nodos de un clúster en función de la necesidad de programar nuevos pods o de la infrautilización de los nodos. No adopta una visión holística de la ubicación de los pods una vez que se han programado en un nodo. Si utiliza el escalador automático de clústeres, también debería utilizar el desprogramador de Kubernetes para evitar desperdiciar capacidad en su clúster.

Si tiene 10 nodos en un clúster y cada nodo se utiliza en un 60%, no está utilizando el 40% de la capacidad aprovisionada en el clúster. Con el escalador automático de clústeres, puede establecer el umbral de utilización por nodo en un 60%, pero solo intentará reducir la escala de un nodo si la utilización cayera por debajo del 60%.

Con el desprogramador, puede analizar la capacidad y la utilización del clúster una vez que se hayan programado los pods o se hayan agregado nodos al clúster. Intenta mantener la capacidad total del clúster por encima de un umbral específico. También puede eliminar los pods en función de la contaminación de los nodos o de los nuevos nodos que se unan al clúster para garantizar que los pods se ejecuten en su entorno informático óptimo. Ten en cuenta que el desprogramador no programa el reemplazo de los pods desalojados, sino que se basa en el programador predeterminado para ello.

Consolidación de Karpenter

Karpenter adopta un enfoque «sin grupos» para la gestión de nodos. Este enfoque es más flexible para los diferentes tipos de carga de trabajo y requiere una configuración inicial menor para los administradores de clústeres. En lugar de predefinir los grupos y escalar cada grupo según las

necesidades de las cargas de trabajo, Karpenter utiliza aprovisionadores y plantillas de nodos para definir en términos generales qué tipo de EC2 instancias se pueden crear y la configuración de las instancias a medida que se crean.

El empaquetado en contenedores es la práctica de utilizar más recursos de la instancia, agrupando más cargas de trabajo en un menor número de instancias con un tamaño óptimo. Si bien esto ayuda a reducir los costos de procesamiento al aprovisionar únicamente los recursos que utilizan sus cargas de trabajo, tiene su contrapartida. El inicio de nuevas cargas de trabajo puede tardar más tiempo, ya que hay que añadir capacidad al clúster, especialmente durante eventos de gran escalado. Tenga en cuenta el equilibrio entre la optimización de costos, el rendimiento y la disponibilidad al configurar el embalaje en contenedores.

Karpenter puede monitorear y empaquetar archivos de forma continua para mejorar la utilización de los recursos de las instancias y reducir los costos de procesamiento. Karpenter también puede seleccionar un nodo de trabajadores más rentable para su carga de trabajo. Esto se puede lograr activando el indicador de «consolidación» en true en el aprovisionador (véase un fragmento de código de ejemplo a continuación). En el siguiente ejemplo, se muestra un ejemplo de aprovisionador que permite la consolidación. En el momento de escribir esta guía, Karpenter no sustituirá una instancia de Spot en ejecución por una instancia de Spot más económica. Para obtener más información sobre la consolidación de Karpenter, consulta este blog.

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
  name: enable-binpacking
spec:
  consolidation:
    enabled: true
```

En el caso de cargas de trabajo que podrían no ser interrumpibles, por ejemplo, trabajos por lotes de larga duración sin puntos de control, considere la posibilidad de anotar los módulos con esa anotación. `do-not-evict` Al optar por no desalojar los módulos, le estás diciendo a Karpenter que no debe eliminar voluntariamente los nodos que contengan este módulo. Sin embargo, si se `do-not-evict` añade un módulo a un nodo mientras éste se está vaciando, los módulos restantes se seguirán desalojando, pero ese módulo bloqueará la terminación hasta que se elimine. En cualquier caso, se acordonará el nodo para evitar que se programen trabajos adicionales en el nodo. A continuación se muestra un ejemplo que muestra cómo configurar la anotación:

```
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
```

Elimine los nodos infrautilizados ajustando los parámetros del escalador automático del clúster

La utilización de los nodos se define como la suma de los recursos solicitados dividida por la capacidad. De forma predeterminada, `scale-down-utilization-threshold` se establece en el 50%. Este parámetro se puede utilizar junto con y`scale-down-unneeded-time`, que determina cuánto tiempo no se necesitará un nodo antes de que pueda ser reducido (el valor predeterminado es de 10 minutos). Kube-scheduler programará los pods que sigan ejecutándose en un nodo reducido en otros nodos. Ajustar esta configuración puede ayudar a eliminar los nodos que están infrautilizados, pero es importante que primero pruebes estos valores para no forzar al clúster a reducir su escala prematuramente.

Para evitar que se produzca una reducción de tamaño, asegúrese de que los módulos cuyo desalojo resulta caro estén protegidos por una etiqueta reconocida por el escalador automático de clústeres. Para ello, asegúrese de que los módulos cuyo desalojo resulta caro tengan la anotación correspondiente. `cluster-autoscaler.kubernetes.io/safe-to-evict=false` A continuación se muestra un ejemplo de yaml para configurar la anotación:

```
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
```

Etiquete los nodos con Cluster Autoscaler y Karpenter

Las etiquetas de recursos de AWS se utilizan para organizar los recursos y realizar un seguimiento detallado de los costes de AWS. No se correlacionan directamente con las etiquetas de Kubernetes para el seguimiento de los costos. Se recomienda empezar con el etiquetado de los recursos de Kubernetes y utilizar herramientas como Kubecost para obtener informes sobre los costes de infraestructura basados en las etiquetas de Kubernetes en los pods, los espacios de nombres, etc.

Los nodos de trabajo deben tener etiquetas para mostrar la información de facturación en AWS Cost Explorer. Con Cluster AutoScaler, etiquete sus nodos de trabajo dentro de un grupo de nodos administrado mediante una plantilla de lanzamiento. En el caso de los grupos de nodos autogestionados, etiquete las instancias mediante un grupo de escalado EC2 automático. En el caso de las instancias aprovisionadas por Karpenter, etiquétalas con spec.tags en la plantilla de nodos.

Clústeres con varios inquilinos

Cuando trabajas en clústeres que comparten diferentes equipos, es posible que no puedas ver otras cargas de trabajo que se ejecutan en el mismo nodo. Si bien las solicitudes de recursos pueden ayudar a aislar algunos problemas de «vecinos ruidosos», como el uso compartido de la CPU, es posible que no aíslen todos los límites de los recursos, como el I/O agotamiento del disco. No todos los recursos consumibles de una carga de trabajo pueden aislarse o limitarse. Las cargas de trabajo que consumen recursos compartidos a un ritmo mayor que otras cargas de trabajo deben aislarse teniendo en cuenta las limitaciones y tolerancias de los nodos. Otra técnica avanzada para este tipo de carga de trabajo es la fijación de la CPU, que garantiza la utilización exclusiva de la CPU en lugar de una CPU compartida para el contenedor.

Aislar las cargas de trabajo a nivel de nodo puede resultar más caro, pero es posible programar los BestEfforttrabajos o aprovechar los ahorros adicionales mediante el uso de instancias reservadas, procesadores Graviton o Spot.

Los clústeres compartidos también pueden tener limitaciones de recursos a nivel de clúster, como el agotamiento de la IP, los límites de servicio de Kubernetes o las solicitudes de escalado de las API.

Deberías revisar la guía de prácticas recomendadas de escalabilidad para asegurarte de que tus clústeres eviten estos límites.

Puedes aislar los recursos a nivel de un espacio de nombres o de un aprovisionador de Karpenter. Las cuotas de recursos proporcionan una forma de establecer límites sobre la cantidad de recursos que pueden consumir las cargas de trabajo en un espacio de nombres. Esta puede ser una buena barrera inicial, pero debe evaluarse continuamente para garantizar que no restrinja artificialmente el escalamiento de las cargas de trabajo.

Los aprovisionadores de Karpenter pueden establecer límites para algunos de los recursos consumibles de un clúster (por ejemplo, la CPU o la GPU), pero tendrá que configurar las aplicaciones arrendatarias para que utilicen el aprovisionador adecuado. Esto puede evitar que un solo aprovisionador cree demasiados nodos en un clúster, pero debe evaluarse continuamente para garantizar que el límite no sea demasiado bajo y, a su vez, evitar que las cargas de trabajo se escalen.

Escalado automático programado

Es posible que tenga que reducir la escala de sus clústeres los fines de semana y fuera del horario laboral. Esto es especialmente importante para los clústeres de prueba y los que no son de producción, en los que se desea reducir la escala a cero cuando no están en uso. Soluciones como la reducción del tamaño de los clústeres permiten reducir las réplicas a cero en función de una programación cron. También puede lograrlo con Karpenter, tal y como se describe en el siguiente blog de AWS.

Optimice los tipos de capacidad informática

Tras optimizar la capacidad total de procesamiento de su clúster y utilizar el empaquetado en contenedores, debe analizar qué tipo de procesamiento ha aprovisionado en sus clústeres y cómo paga esos recursos. AWS cuenta con planes de ahorro informático que pueden reducir el coste de su procesamiento, que clasificaremos en los siguientes tipos de capacidad:

- Spot
- Savings Plans
- Bajo demanda
- Fargate

Cada tipo de capacidad tiene diferentes desventajas en cuanto a los gastos generales de administración, la disponibilidad y los compromisos a largo plazo, y tendrá que decidir cuál es la adecuada para su entorno. Ningún entorno debe depender de un solo tipo de capacidad y puede combinar varios tipos de ejecución en un solo clúster para optimizar los costos y los requisitos de carga de trabajo específicos.

Spot

El tipo de capacidad puntual aprovisiona EC2 instancias a partir de la capacidad sobrante de una zona de disponibilidad. Spot ofrece los mayores descuentos (hasta un 90%), pero esas instancias se pueden interrumpir cuando se necesiten en otros lugares. Además, es posible que no siempre haya capacidad para aprovisionar nuevas instancias de Spot y las instancias de Spot existentes se pueden recuperar con un aviso de interrupción de 2 minutos. Si su aplicación tiene un proceso de inicio o cierre prolongado, es posible que las instancias puntuales no sean la mejor opción.

La computación puntual debe utilizar una amplia variedad de tipos de instancias para reducir la probabilidad de no disponer de capacidad puntual. Las interrupciones de las instancias deben gestionarse para cerrar los nodos de forma segura. Los nodos aprovisionados con Karpenter o que forman parte de un grupo de nodos gestionado admiten automáticamente las notificaciones de interrupción de instancias. Si utilizas nodos autogestionados, tendrás que ejecutar el controlador de terminación de nodos por separado para cerrar correctamente las instancias puntuales.

Es posible equilibrar las instancias puntuales y bajo demanda en un solo clúster. Con Karpenter, puede crear aprovisionadores ponderados para lograr un equilibrio entre los diferentes tipos de capacidad. Con Cluster Autoscaler, puede crear grupos de nodos mixtos con instancias puntuales y bajo demanda o reservadas.

A continuación, se muestra un ejemplo del uso de Karpenter para priorizar las instancias puntuales antes que las instancias bajo demanda. Al crear un aprovisionador, puede especificar las opciones puntuales, bajo demanda o ambas (como se muestra a continuación). Si especificas ambas opciones, y si el pod no especifica explícitamente si necesita usar Spot o On-Demand, Karpenter prioriza Spot al aprovisionar un nodo con una estrategia de asignación. 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 y AWS EDP

Puede reducir su gasto informático mediante un plan de ahorro informático. Los planes de ahorro ofrecen precios reducidos con un compromiso de uso informático de 1 o 3 años. El uso se puede aplicar a EC2 las instancias de un clúster de EKS, pero también se aplica a cualquier uso informático, como Lambda y Fargate. Con los planes de ahorro, puede reducir los costos y seguir eligiendo cualquier tipo de EC2 instancia durante su período de compromiso.

El plan de ahorro informático puede reducir sus EC2 costes hasta en un 66% sin tener que comprometerse con los tipos de instancias, las familias o las regiones que desee utilizar. Los ahorros se aplican automáticamente a las instancias a medida que las usa.

EC2 Instance Savings Plans ofrece hasta un 72% de ahorro en cómputo con un compromiso de uso en una región y EC2 familia específicas, por ejemplo, instancias de la familia C. Puede cambiar el uso a cualquier zona de disponibilidad de la región, utilizar cualquier generación de la familia de instancias, por ejemplo, c5 o c6, y utilizar instancias de cualquier tamaño dentro de la familia. El descuento se aplicará automáticamente a cualquier instancia de tu cuenta que cumpla con los criterios del plan de ahorro.

Las instancias reservadas son similares a EC2 Instance Savings Plan, pero también garantizan la capacidad en una zona o región de disponibilidad y reducen los costos (hasta un 72%) en comparación con las instancias bajo demanda. Una vez que calcule la cantidad de capacidad reservada que necesitará, puede seleccionar durante cuánto tiempo desea reservarla (1 o 3 años). Los descuentos se aplicarán automáticamente a medida que ejecute esas EC2 instancias en su cuenta.

Los clientes también tienen la opción de suscribirse a un acuerdo empresarial con AWS. Los contratos Enterprise ofrecen a los clientes la opción de personalizar los acuerdos que mejor se adapten a sus necesidades. Los clientes pueden disfrutar de descuentos en los precios basados en AWS EDP (Enterprise Discount Program). Para obtener información adicional sobre los acuerdos empresariales, póngase en contacto con su representante de ventas de AWS.

Bajo demanda

EC2 Las instancias bajo demanda ofrecen la ventaja de estar disponibles sin interrupciones (en comparación con las instancias puntuales) y de no tener compromisos a largo plazo (en comparación

con los planes de ahorro). Si desea reducir los costos en un clúster, debe reducir el uso de EC2 instancias bajo demanda.

Tras optimizar los requisitos de carga de trabajo, debería poder calcular la capacidad mínima y máxima de los clústeres. Este número puede cambiar con el tiempo, pero rara vez disminuye. Considere la posibilidad de utilizar un Savings Plan para todo lo que esté por debajo del mínimo y busque una capacidad que no afecte a la disponibilidad de su solicitud. Cualquier otra cosa que no se utilice de forma continua o que sea necesaria para garantizar su disponibilidad se puede utilizar bajo demanda.

Como se menciona en esta sección, la mejor manera de reducir el uso es consumir menos recursos y utilizar los recursos que aprovisiona en la mayor medida posible. Con el escalador automático de clústeres, puede eliminar los nodos infrautilizados con esta configuración. `scale-down-utilization-threshold` Con Karpenter, se recomienda habilitar la consolidación.

Para identificar manualmente los tipos de EC2 instancias que se pueden usar con sus cargas de trabajo, debe usar ec2-instance-selector, que puede mostrar las instancias que están disponibles en cada región, así como las instancias compatibles con EKS. Ejemplo de uso para instancias con arquitectura de procesos x86, 4 Gb de memoria, 2 v CPUs y disponibles en la región 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
```

Para los entornos que no son de producción, puede hacer que los clústeres se reduzcan automáticamente durante las horas no utilizadas, como la noche y los fines de semana. El proyecto kubecost cluster-turndown es un ejemplo de un controlador que puede reducir automáticamente la escala del clúster en función de un cronograma establecido.

Computación Fargate

Fargate Compute es una opción de procesamiento totalmente gestionada para los clústeres EKS. Proporciona aislamiento de pods mediante la programación de un pod por nodo en un clúster de

Kubernetes. Le permite ajustar el tamaño de sus nodos de procesamiento a los requisitos de CPU y RAM de su carga de trabajo para controlar estrictamente el uso de la carga de trabajo en un clúster.

Fargate puede escalar cargas de trabajo tan pequeñas como 0,25 vCPU con 0,5 GB de memoria y tan grandes como 16 vCPU con 120 GB de memoria. Hay límites en cuanto a la cantidad de variaciones de tamaño de cápsula disponibles y tendrá que entender cuál es la mejor manera de adaptar su carga de trabajo a una configuración de Fargate. Por ejemplo, si su carga de trabajo requiere 1 vCPU con 0,5 GB de memoria, el pod Fargate más pequeño será 1 vCPU con 2 GB de memoria.

Si bien Fargate tiene muchas ventajas, como la ausencia de administración de EC2 instancias o sistemas operativos, es posible que requiera más capacidad de procesamiento que EC2 las instancias tradicionales debido a que cada pod implementado está aislado como un nodo independiente en el clúster. Esto requiere una mayor duplicación en el caso de elementos como el Kubelet, los agentes de registro y todos los DaemonSets que se suelen implementar en un nodo. DaemonSets no son compatibles con Fargate y deberán convertirse en un pod «`sidecars"` y ejecutarse junto con la aplicación.

Fargate no puede beneficiarse del empaquetado en contenedores ni del sobreaprovisionamiento de la CPU porque el límite de la carga de trabajo es un nodo que no se puede dividir ni compartir entre cargas de trabajo. Fargate le ahorrará tiempo de administración de EC2 instancias, lo que en sí mismo tiene un costo, pero los costos de CPU y memoria pueden ser más caros que otros tipos de EC2 capacidad. Los pods Fargate pueden aprovechar el plan de ahorro de cómputo para reducir el costo bajo demanda.

Optimice el uso de la computación

Otra forma de ahorrar dinero en su infraestructura de cómputo es utilizar un cómputo más eficiente para la carga de trabajo. Esto puede provenir de un cómputo de uso general de mayor rendimiento, como los procesadores Graviton, que son hasta un 20% más baratos y un 60% más eficientes desde el punto de vista energético que los x86, o de aceleradores específicos para cargas de trabajo, como y. GPUs FPGAs Necesitará crear contenedores que puedan ejecutarse en una arquitectura ARM y configurar los nodos con los aceleradores adecuados para sus cargas de trabajo.

EKS tiene la capacidad de ejecutar clústeres con arquitecturas mixtas (por ejemplo, amd64 y arm64) y, si sus contenedores están compilados para varias arquitecturas, puede aprovechar los procesadores Graviton con Karpenter y permitir que ambas arquitecturas estén integradas en su aprovisionador. Sin embargo, para mantener un rendimiento uniforme, se recomienda mantener cada

carga de trabajo en una única arquitectura de cómputo y utilizar solo una arquitectura diferente si no hay capacidad adicional disponible.

Los aprovisionadores se pueden configurar con varias arquitecturas y las cargas de trabajo también pueden solicitar arquitecturas específicas en sus especificaciones de carga de trabajo.

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
  name: default
spec:
  requirements:
  - key: "kubernetes.io/arch"
    operator: In
    values: ["arm64", "amd64"]
```

Con Cluster Autoscaler, necesitará crear un grupo de nodos para las instancias de Graviton y establecer tolerancias de nodos en su carga de trabajo para utilizar la nueva capacidad.

GPUs y FPGAs puede aumentar considerablemente el rendimiento de su carga de trabajo, pero será necesario optimizar la carga de trabajo para poder utilizar el acelerador. Se pueden usar muchos tipos de cargas de trabajo para el aprendizaje automático y la inteligencia artificial GPUs para la computación, y las instancias se pueden agregar a un clúster y montar en una carga de trabajo mediante solicitudes de recursos.

```
spec:
  template:
    spec:
    - containers:
      ...
      resources:
          limits:
            nvidia.com/gpu: "1"
```

Parte del hardware de la GPU se puede compartir entre varias cargas de trabajo, por lo que se puede aprovisionar y utilizar una sola GPU. Para ver cómo configurar el uso compartido de la GPU de la carga de trabajo, consulte el complemento del dispositivo de GPU virtual para obtener más información. También puedes consultar los siguientes blogs:

- Uso compartido de GPU en Amazon EKS con instancias aceleradas y segmentadas de NVIDIA EC2

- Maximización de la utilización de la GPU con la GPU de múltiples instancias (MIG) de NVIDIA en Amazon EKS: cómo ejecutar más pods por GPU para mejorar el rendimiento

Optimización de costos: redes

Diseñar sistemas para una alta disponibilidad (HA) es una práctica recomendada para lograr la resiliencia y la tolerancia a los fallos. En la práctica, esto significa distribuir las cargas de trabajo y la infraestructura subyacente en varias zonas de disponibilidad (AZs) de una región de AWS determinada. Asegurarse de que estas características estén implementadas en su entorno Amazon EKS mejorará la confiabilidad general de su sistema. Además, es probable que sus entornos EKS también estén compuestos por una variedad de estructuras (es decir VPCs), componentes (es decir) e integraciones (es decir, el ECR y otros registros de contenedores ELBs).

La combinación de sistemas de alta disponibilidad y otros componentes específicos para cada caso de uso puede desempeñar un papel importante en la forma en que se transfieren y procesan los datos. Esto, a su vez, repercutirá en los costes incurridos por la transferencia y el procesamiento de los datos.

Las prácticas que se detallan a continuación le ayudarán a diseñar y optimizar sus entornos de EKS a fin de lograr la rentabilidad en diferentes dominios y casos de uso.

Comunicación de pod a pod

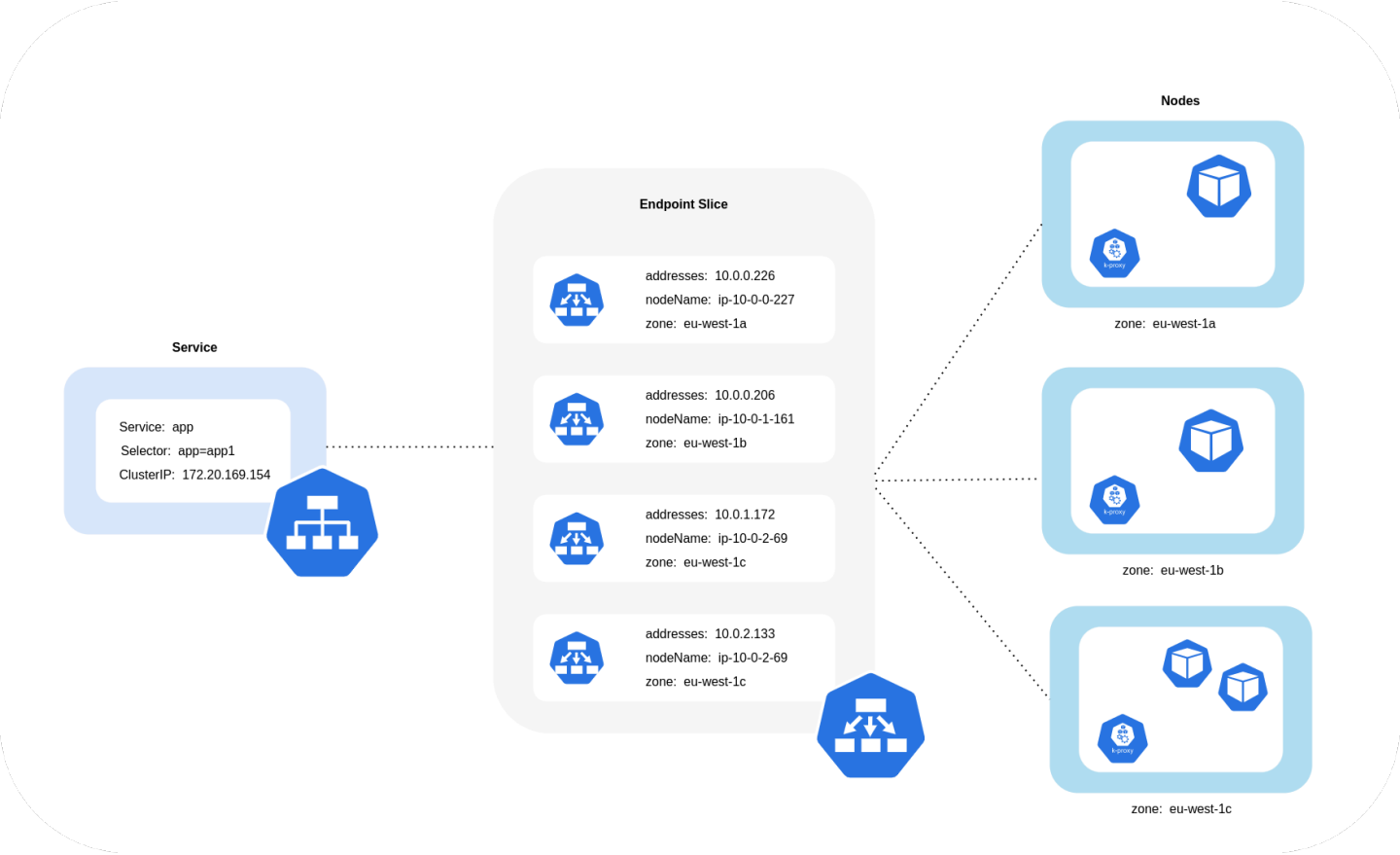
Según la configuración, la comunicación de red y la transferencia de datos entre los pods pueden tener un impacto significativo en el costo total de ejecutar las cargas de trabajo de Amazon EKS. En esta sección se tratarán diferentes conceptos y enfoques para mitigar los costos relacionados con la comunicación entre los módulos, teniendo en cuenta las arquitecturas de alta disponibilidad (HA), el rendimiento de las aplicaciones y la resiliencia.

Restringir el tráfico a una zona de disponibilidad

Desde el principio, el proyecto Kubernetes comenzó a desarrollar estructuras sensibles a la topología, incluidas etiquetas como kubernetes. io/hostname, topology.kubernetes.io/region, and topology.kubernetes.io/zonese asignan a los nodos para habilitar funciones como la distribución de la carga de trabajo entre los dominios de fallas y los aprovisionadores de volúmenes sensibles a la topología. Tras licenciarse en Kubernetes 1.17, las etiquetas también se utilizaron para habilitar capacidades de enrutamiento con reconocimiento de topología para la comunicación de un pod a otro.

A continuación, se muestran algunas estrategias sobre cómo controlar la cantidad de tráfico entre zonas de disponibilidad y disponibilidad entre los pods de su clúster de EKS para reducir los costes y minimizar la latencia.

Si quieres ver de forma pormenorizada la cantidad de tráfico entre zonas de disponibilidad (Cross-AZ) entre los pods de tu clúster (por ejemplo, la cantidad de datos transferidos en bytes), consulta esta publicación.



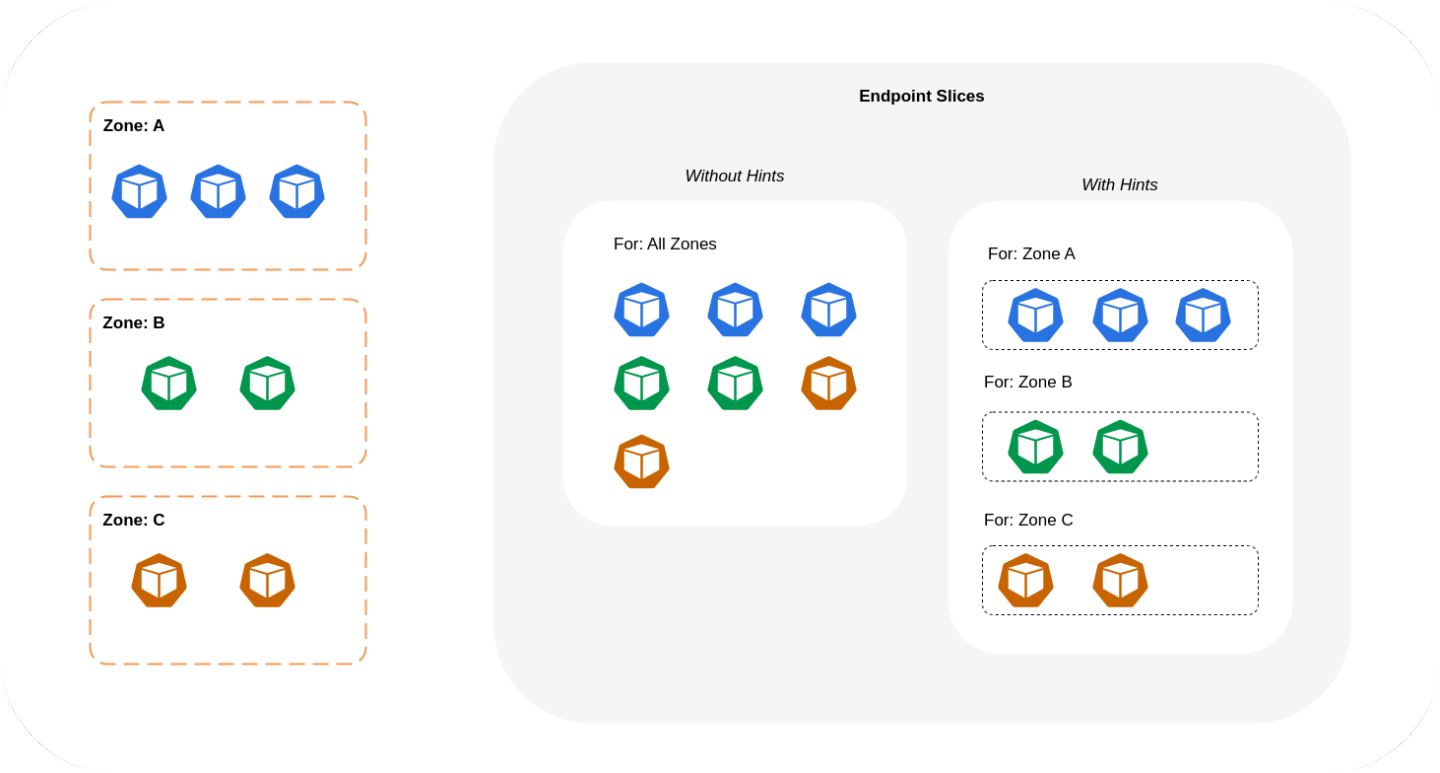
Como se muestra en el diagrama anterior, los servicios son la capa de abstracción de red estable que recibe el tráfico destinado a tus pods. Cuando se crea un servicio, EndpointSlices se crean varios. Cada uno EndpointSlice tiene una lista de puntos finales que contiene un subconjunto de direcciones de pod junto con los nodos en los que se ejecutan y cualquier información adicional sobre la topología. Cuando se utiliza la CNI de Amazon VPC, kube-proxy, un daemonset que se ejecuta en todos los nodos, mantiene las reglas de red para permitir la comunicación entre los pods y la detección de servicios (las alternativas basadas en EBPF pueden no utilizar kube-proxy pero ofrecer un comportamiento equivalente CNIs). Cumple la función de enrutamiento interno, pero lo hace en función de lo que consume de lo creado. EndpointSlices

En EKS, kube-proxy utiliza principalmente las reglas NAT de iptables (o IPVS, NFTablescomo alternativa) para distribuir el tráfico entre todos los pods del clúster, independientemente de su ubicación en el nodo o en la zona de disponibilidad. Esta distribución predeterminada puede provocar el enrutamiento del tráfico entre zonas de disponibilidad, lo que podría provocar un aumento de la latencia para las aplicaciones confidenciales y de los costes de transferencia de datos entre zonas de disponibilidad en grandes despliegues.

Uso del enrutamiento compatible con la topología (anteriormente denominado Topology Aware Hints)

Cuando el enrutamiento con reconocimiento de topología está habilitado e implementado en un servicio de Kubernetes, el EndpointSlice controlador asignará proporcionalmente los puntos finales a las diferentes zonas en las que se encuentra distribuido el clúster. Para cada uno de esos puntos finales, el EndpointSlice controlador también establecerá una pista para la zona. Las sugerencias describen en qué zona debe funcionar el tráfico un punto final. `kube-proxy`a continuación, enrutará el tráfico de una zona a un punto final en función de las sugerencias que se apliquen.

En el siguiente diagrama se muestra cómo EndpointSlices las sugerencias están organizadas de tal forma que `kube-proxy` permiten saber a qué destino deben dirigirse en función de su punto de origen zonal. Sin pistas, no existe tal asignación u organización y el tráfico se redirigirá a diferentes destinos zonales, independientemente de su procedencia.



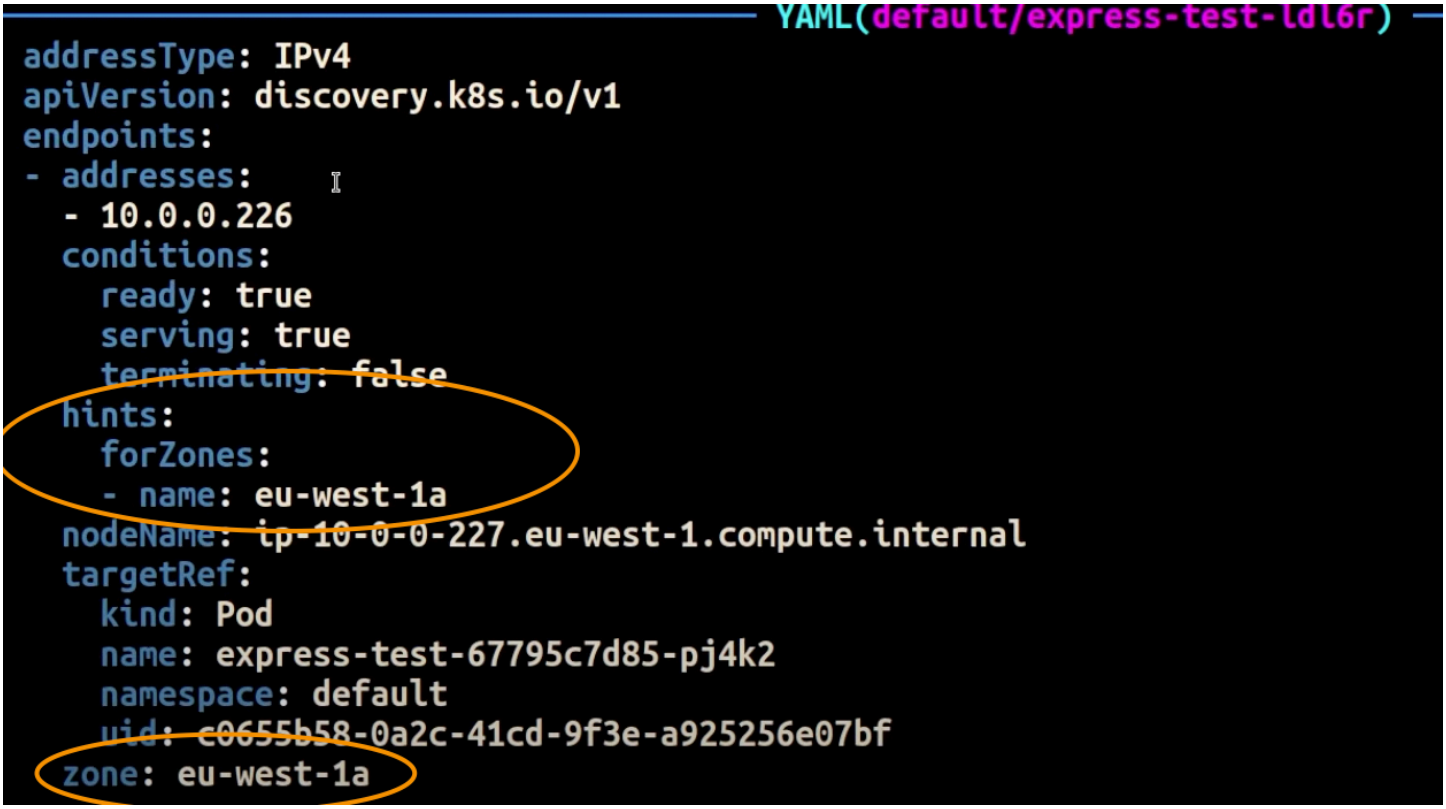
En algunos casos, el EndpointSlice controlador puede aplicar una sugerencia para una zona diferente, lo que significa que el punto final podría terminar sirviendo tráfico que se origina en una zona diferente. El motivo es intentar mantener una distribución uniforme del tráfico entre los puntos finales de distintas zonas.

A continuación, se muestra un fragmento de código sobre cómo habilitar el enrutamiento con reconocimiento de topología para un servicio.

```
apiVersion: v1
kind: Service
metadata:
  name: orders-service
  namespace: ecommerce
  annotations:
    service.kubernetes.io/topology-mode: Auto
spec:
  selector:
    app: orders
  type: ClusterIP
  ports:

* protocol: TCP
port: 3003
targetPort: 3003
```

La siguiente captura de pantalla muestra el resultado de que el EndpointSlice controlador haya aplicado correctamente una sugerencia a un punto final para una réplica de un pod que se ejecuta en la zona de disponibilidad. `eu-west-1a`



> **Note**
>
> Es importante tener en cuenta que el enrutamiento compatible con la topología aún se encuentra en versión beta. Esta función funciona de forma más predecible con cargas de trabajo distribuidas uniformemente en toda la topología del clúster, ya que el controlador asigna los puntos finales de forma proporcional entre las zonas, pero puede omitir las asignaciones de sugerencias cuando los recursos de los nodos de una zona están demasiado desequilibrados como para evitar una sobrecarga excesiva. Por lo tanto, se recomienda encarecidamente utilizarla junto con las restricciones de programación que aumenten la disponibilidad de una aplicación, como las restricciones de dispersión de la topología de los módulos. Tenga en cuenta que es posible que tampoco se asignen sugerencias cuando la capacidad fluctúa entre zonas, como cuando se utilizan Amazon EC2 Spot Instances, ya que las interrupciones o los reemplazos no se detectan en tiempo real al calcular la distribución proporcional.

Uso de la distribución del tráfico

Introducido en Kubernetes 1.30 y puesto a disposición del público en general en 1.33, Traffic Distribution ofrece una alternativa más sencilla al enrutamiento compatible con la topología para preferir el tráfico en la misma zona. Si bien el enrutamiento sensible a la topología intenta utilizar un enfoque inteligente del enrutamiento del tráfico para evitar sobrecargar los puntos finales, se ha traducido en un comportamiento impredecible. En cambio, Traffic Distribution prioriza la previsibilidad. La PreferClose opción indica a kube-proxy que cree reglas que dirijan primero el tráfico a los puntos finales de la misma zona en función de la sugerencia zonal establecida por el controlador. EndpointSlice Cuando no hay puntos de enlace disponibles en la misma zona, se recurre a la distribución del tráfico entre cualquier punto de enlace del clúster para el Servicio. Esta función está diseñada para cargas de trabajo que aceptan la compensación de optimizar en función de la proximidad en lugar de intentar una distribución uniforme de la carga que proporciona el enrutamiento con reconocimiento de topología.

A continuación se muestra un fragmento de código sobre cómo habilitar la distribución del tráfico para un servicio.

```
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
```

Al habilitar la distribución del tráfico, surge un desafío común: los puntos finales dentro de una única zona de disponibilidad pueden sobrecargarse si la mayor parte del tráfico se origina en esa misma zona. Esta sobrecarga puede crear problemas importantes:

- Un escalador automático de pods horizontal (HPA) único que gestione una implementación en zonas de disponibilidad múltiples puede responder escalando los pods en diferentes áreas de disponibilidad. AZs Sin embargo, esta acción no aborda de manera eficaz el aumento de la carga en la zona afectada.

- Esta situación, a su vez, puede provocar una ineficiencia de los recursos. Cuando los escaladores automáticos de clústeres, como Karpenter, detectan que el módulo se amplía de forma horizontal, pueden aprovisionar nodos adicionales en los que no estén afectados AZs, lo que se traduce en una asignación de recursos AZs innecesaria.

Para superar este desafío:

- Cree despliegues separados por zona que tengan los suyos propios HPAs para escalar de forma independiente unos de otros.
- Aproveche las restricciones de distribución de la topología para garantizar la distribución de la carga de trabajo en todo el clúster, lo que ayuda a evitar la sobrecarga de los puntos finales en las zonas de alto tráfico.

Uso de escaladores automáticos: aprovisione nodos a una zona de disponibilidad específica

Recomendamos encarecidamente ejecutar las cargas de trabajo en varios entornos de alta disponibilidad. AZs Esto mejora la confiabilidad de sus aplicaciones, especialmente cuando se produce un incidente o un problema con una zona de disponibilidad. En caso de que esté dispuesto a sacrificar la confiabilidad para reducir los costos relacionados con la red, puede restringir sus nodos a una única zona de disponibilidad.

Para ejecutar todos los pods en la misma zona de disponibilidad, aprovisione los nodos de trabajo en la misma zona de disponibilidad o programe los pods de los nodos de trabajo para que se ejecuten en la misma zona de disponibilidad. Para aprovisionar nodos dentro de una única AZ, defina un grupo de nodos con subredes que pertenezcan a la misma AZ con Cluster Autoscaler (CA). En el caso de Karpenter, utilice `topology.kubernetes.io/zone` y especifique la AZ en la que desea crear los nodos de trabajo. Por ejemplo, el siguiente fragmento de código de aprovisionamiento de Karpenter aprovisiona los nodos de la zona de distribución us-west-2a.

Karpenter

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
name: single-az
spec:
  requirements:

* key: "topology.kubernetes.io/zone"`
```

```
operator: In
values: ["us-west-2a"]
```

Escalador automático de clústeres (CA)

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-ca-cluster
  region: us-east-1
  version: "1.21"
availabilityZones:

* us-east-1a
managedNodeGroups:
* name: managed-nodes
labels:
  role: managed-nodes
instanceType: t3.medium
minSize: 1
maxSize: 10
desiredCapacity: 1
...
```

Uso de la asignación de pods y la afinidad de nodos

Como alternativa, si tienes nodos de trabajo que se ejecutan en varios nodos AZs, cada nodo tendría la etiqueta topology.kubernetes.io/zone con el valor de su zona de disponibilidad (como us-west-2a o us-west-2b). Puedes utilizar o programar los pods para los nodos de una única zona de disponibilidad. `nodeSelector` `nodeAffinity` Por ejemplo, el siguiente archivo de manifiesto programará el pod dentro de un nodo que se ejecute en la AZ us-west-2a.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  nodeSelector:
    topology.kubernetes.io/zone: us-west-2a
  containers:
```

```
* name: nginx
image: nginx
imagePullPolicy: IfNotPresent
```

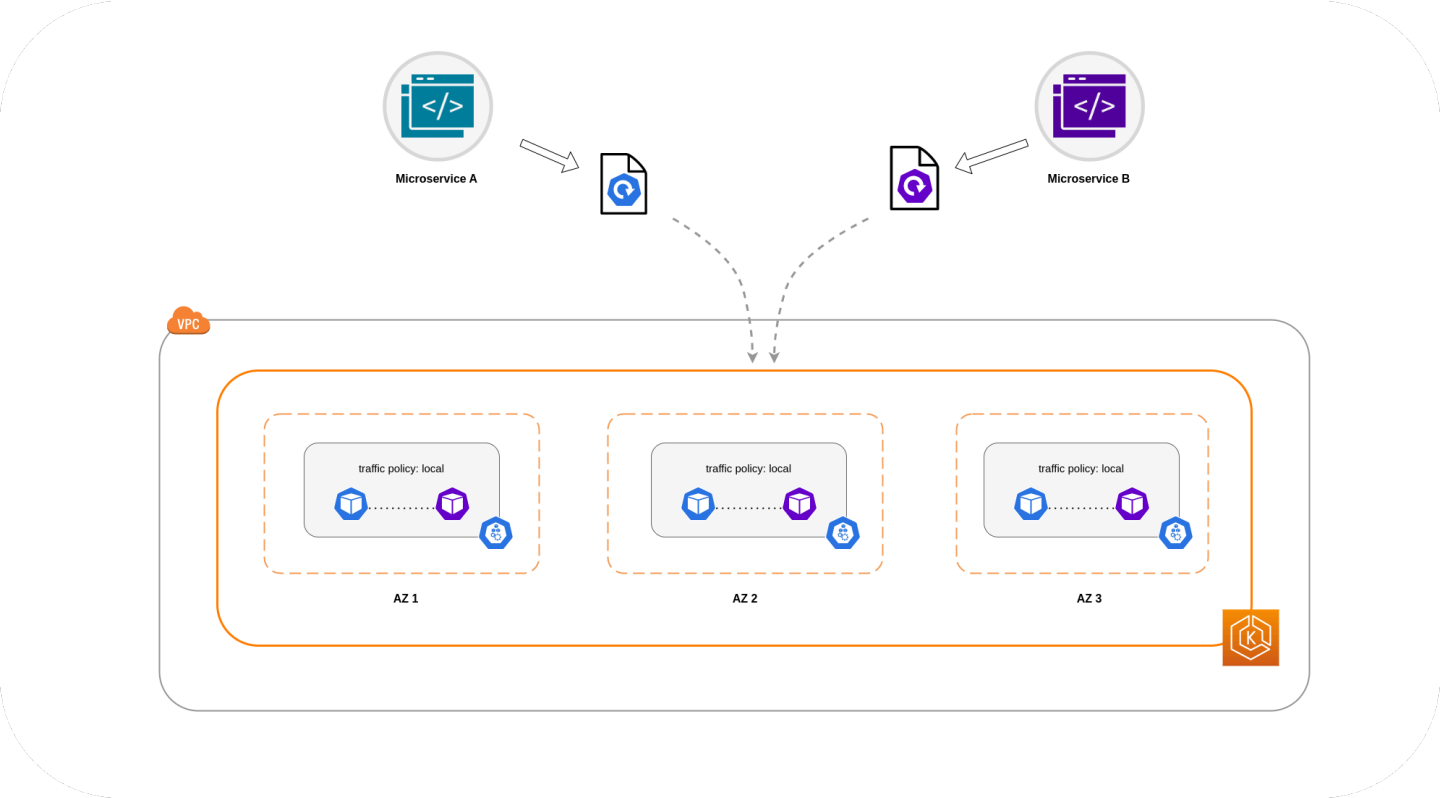
Restringir el tráfico a un nodo

Hay casos en los que restringir el tráfico a nivel zonal no es suficiente. Además de reducir los costes, es posible que tengas el requisito adicional de reducir la latencia de la red entre determinadas aplicaciones que tienen una intercomunicación frecuente. Para lograr un rendimiento de red óptimo y reducir los costos, necesita una forma de restringir el tráfico a un nodo específico. Por ejemplo, el microservicio A siempre debe comunicarse con el microservicio B en el nodo 1, incluso en configuraciones de alta disponibilidad (HA). Hacer que el microservicio A en el nodo 1 se comunique con el microservicio B en el nodo 2 puede tener un impacto negativo en el rendimiento deseado para aplicaciones de esta naturaleza, especialmente si el nodo 2 está completamente en una zona de disponibilidad separada.

Uso de la política de tráfico interna del servicio

Para restringir el tráfico de la red del Pod a un nodo, puedes utilizar la política de tráfico interna del Servicio. De forma predeterminada, el tráfico enviado al servicio de una carga de trabajo se distribuirá aleatoriamente entre los diferentes puntos finales generados. Por lo tanto, en una arquitectura de alta disponibilidad, eso significa que el tráfico del microservicio A podría ir a cualquier réplica del microservicio B en cualquier nodo dado a través de los diferentes nodos. AZs Sin embargo, con la política de tráfico interna del Servicio establecida en`Local`, el tráfico se restringirá a los puntos finales del nodo desde el que se originó el tráfico. Esta política dicta el uso exclusivo de los puntos finales locales de los nodos. Por lo tanto, los costos relacionados con el tráfico de red para esa carga de trabajo serán más bajos que si la distribución se realizara en todo el clúster. Además, la latencia será menor, lo que aumentará el rendimiento de la aplicación.

> **Note**
>
> Es importante tener en cuenta que esta función no se puede combinar con el enrutamiento compatible con la topología en Kubernetes.



A continuación se muestra un fragmento de código sobre cómo configurar la política de tráfico interno de un servicio.

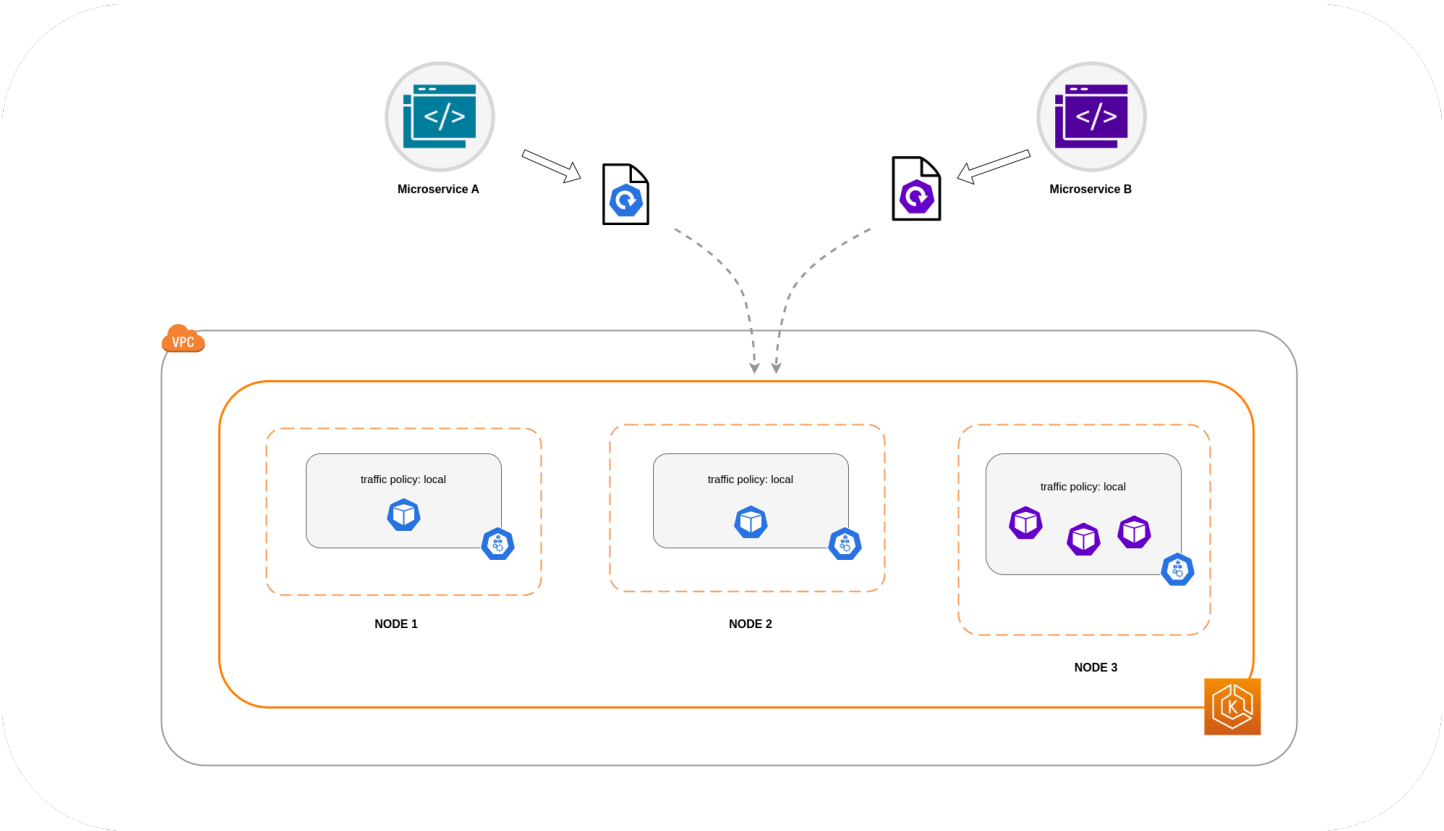
```
apiVersion: v1
kind: Service
metadata:
  name: orders-service
  namespace: ecommerce
spec:
  selector:
    app: orders
  type: ClusterIP
  ports:

* protocol: TCP
port: 3003
targetPort: 3003
  internalTrafficPolicy: Local
```

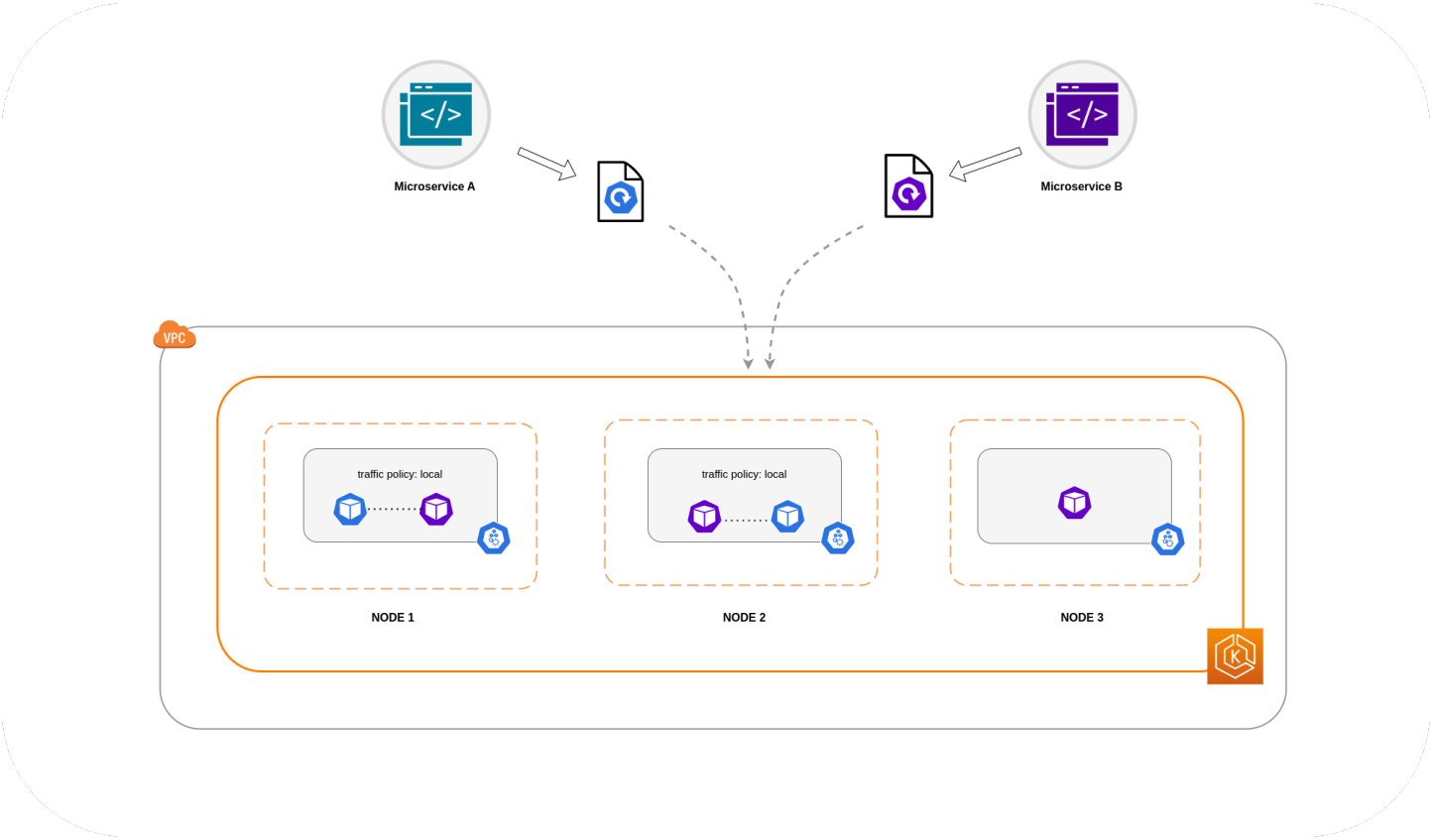
Para evitar que tu aplicación se comporte de forma inesperada debido a las caídas de tráfico, debes considerar los siguientes enfoques:

- Ejecuta suficientes réplicas para cada uno de los pods que se comunican
- Tenga una distribución relativamente uniforme de los pods utilizando restricciones de dispersión topológica
- Utiliza las reglas de afinidad entre los pods para la ubicación conjunta de los pods que se comunican

En este ejemplo, tiene 2 réplicas del microservicio A y 3 réplicas del microservicio B. Si el microservicio A tiene sus réplicas distribuidas entre los nodos 1 y 2, y el microservicio B tiene las 3 réplicas en el nodo 3, no podrán comunicarse debido a la política de tráfico interna. `Local` Cuando no hay puntos de enlace locales disponibles en los nodos, se interrumpe el tráfico.



Si el microservicio B tiene 2 de sus 3 réplicas en los nodos 1 y 2, habrá comunicación entre las aplicaciones homólogas. Sin embargo, seguiría teniendo una réplica aislada del microservicio B sin ninguna réplica homóloga con la que comunicarse.



> **Note**
>
> En algunos escenarios, una réplica aislada como la que se muestra en el diagrama anterior puede no ser motivo de preocupación si sigue cumpliendo un propósito (por ejemplo, atender solicitudes del tráfico entrante externo).

Uso de la política de tráfico interno del servicio con restricciones de dispersión topológica

El uso de la política de tráfico interna junto con las restricciones de dispersión de la topología puede resultar útil para garantizar que se dispone del número correcto de réplicas para comunicar los microservicios en los distintos nodos.

```
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
```

Uso de la política de tráfico interno del servicio con las reglas de afinidad de los pods

Otro enfoque consiste en utilizar las reglas de afinidad de los pods al utilizar la política de tráfico interna del Servicio. Gracias a la afinidad por los pods, puedes influir en el programador para que ubique determinados pods en un mismo lugar, ya que se comunican con frecuencia. Si aplicas restricciones de programación estrictas (`requiredDuringSchedulingIgnoredDuringExecution`) a determinados pods, obtendrás mejores resultados en cuanto a la ubicación conjunta de los pods cuando el programador coloque los pods en nodos.

```
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"
```

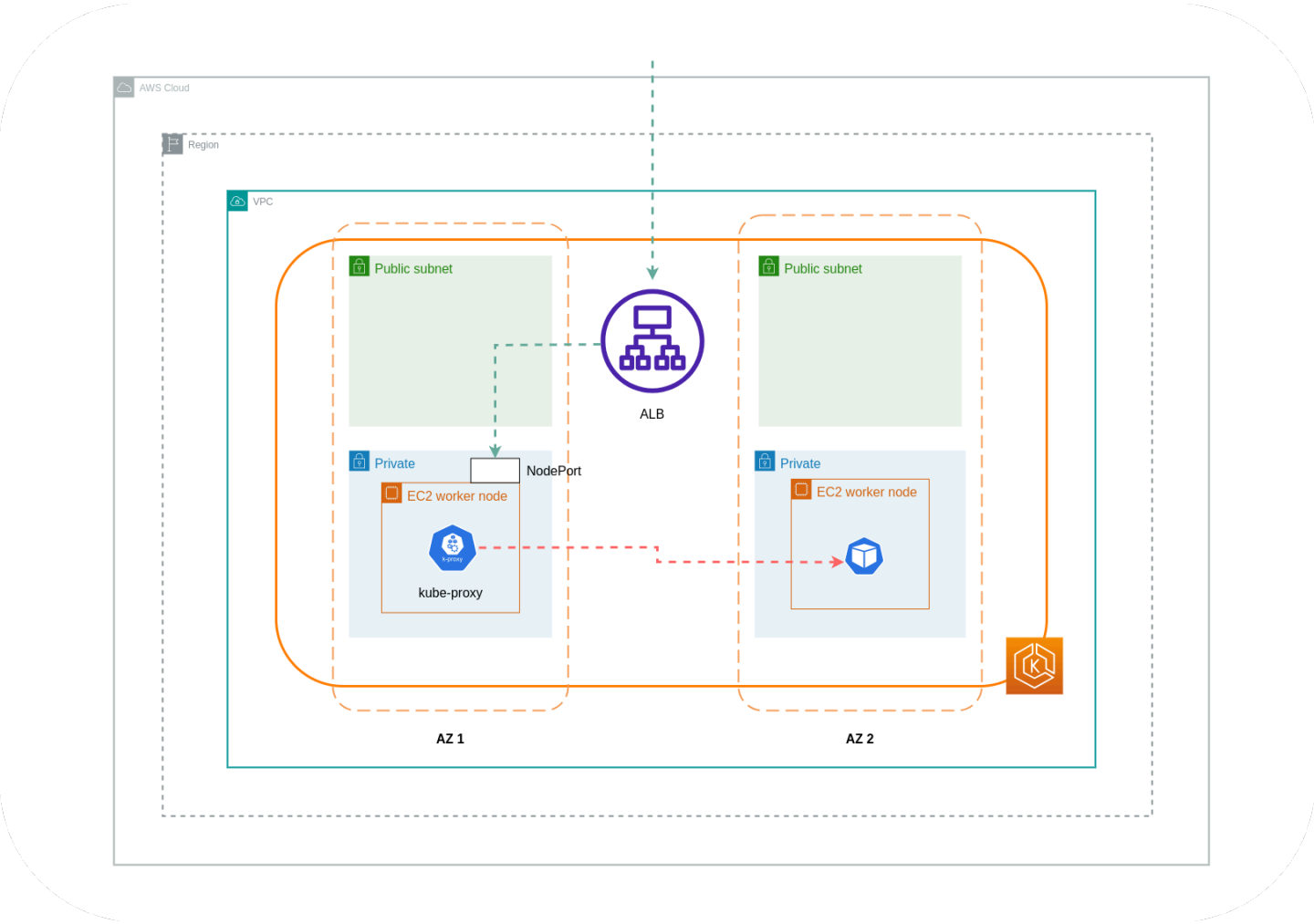
Comunicación entre Load Balancer y Pod

Las cargas de trabajo de EKS suelen estar respaldadas por un balanceador de cargas que distribuye el tráfico a los pods correspondientes del clúster de EKS. Su arquitectura puede incluir balanceadores de carga internos y and/or externos. Según la arquitectura y las configuraciones del tráfico de la red, la comunicación entre los balanceadores de carga y los pods puede contribuir en gran medida a los gastos de transferencia de datos.

Puede utilizar el controlador Load Balancer de AWS para gestionar automáticamente la creación de recursos de ELB (ALB y NLB). Los gastos de transferencia de datos en los que incurra en dichas configuraciones dependerán de la ruta que siga el tráfico de la red. El controlador Load Balancer de AWS admite dos modos de tráfico de red, el modo instancia y el modo ip.

Cuando utilice el modo instancia, se NodePort abrirá una en cada nodo del clúster de EKS. A continuación, el balanceador de cargas distribuirá el tráfico de manera uniforme entre los nodos. Si un nodo tiene el pod de destino en ejecución, no se incurrirá en costes de transferencia de datos. Sin embargo, si el pod de destino está en un nodo independiente y en una zona de disponibilidad diferente a la que NodePort recibe el tráfico, habrá un salto de red adicional desde el kube-proxy al pod de destino. En tal escenario, se cobrarán cargos por la transferencia de datos entre zonas de disponibilidad. Debido a la distribución uniforme del tráfico entre los nodos, es muy probable que los saltos de tráfico de red entre zonas desde los proxies de Kube a los pods de destino correspondientes conlleven gastos adicionales por la transferencia de datos.

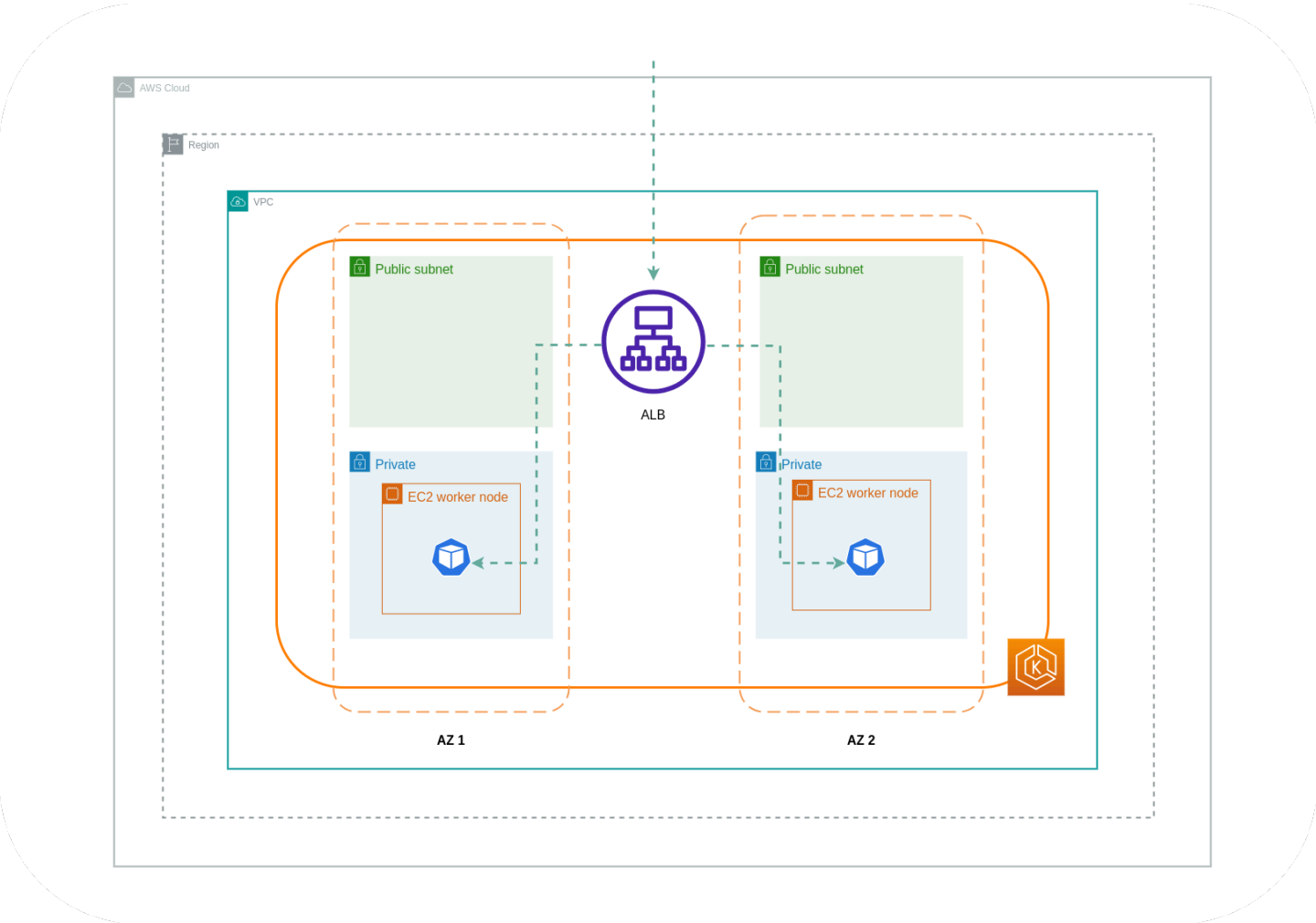
En el siguiente diagrama, se muestra una ruta de red para el tráfico que va del balanceador de cargas al pod de destino y NodePort, posteriormente, al pod de destino en un `kube-proxy` nodo independiente de una zona de disponibilidad diferente. Este es un ejemplo de la configuración del modo de instancia.



Cuando se utiliza el modo ip, el tráfico de red se envía por proxy desde el balanceador de cargas directamente al pod de destino. Como resultado, este enfoque no implica gastos de transferencia de datos.

> **Note**
>
> Se recomienda configurar el balanceador de carga en modo de tráfico IP para reducir los cargos por transferencia de datos. Para esta configuración, también es importante asegurarse de que el balanceador de carga esté implementado en todas las subredes de la VPC.

En el siguiente diagrama, se muestran las rutas de red del tráfico que fluye desde el balanceador de cargas a los pods en el modo IP de la red.



Transferencia de datos desde el registro de contenedores

Amazon ECR

La transferencia de datos al registro privado de Amazon ECR es gratuita. La transferencia de datos dentro de una región es gratuita, pero la transferencia de datos a Internet y entre regiones se cobrará según las tarifas de transferencia de datos por Internet en ambos lados de la transferencia.

Debe utilizar la función de replicación de imágenes ECRs integrada para replicar las imágenes del contenedor pertinentes en la misma región que sus cargas de trabajo. De esta forma, la replicación se cobraría una sola vez y todas las extracciones de imágenes de una misma región (intrarregión) serían gratuitas.

Puede reducir aún más los costos de transferencia de datos asociados con la extracción de imágenes de la ECR (transferencia de datos saliente) mediante el uso de puntos finales de VPC de interfaz para conectarse a los repositorios de ECR de la región. El enfoque alternativo de conectarse

al punto final público de AWS del ECR (a través de una puerta de enlace NAT y una puerta de enlace de Internet) generará mayores costos de procesamiento y transferencia de datos. En la siguiente sección, se analizará con más detalle la reducción de los costes de transferencia de datos entre sus cargas de trabajo y los servicios de AWS.

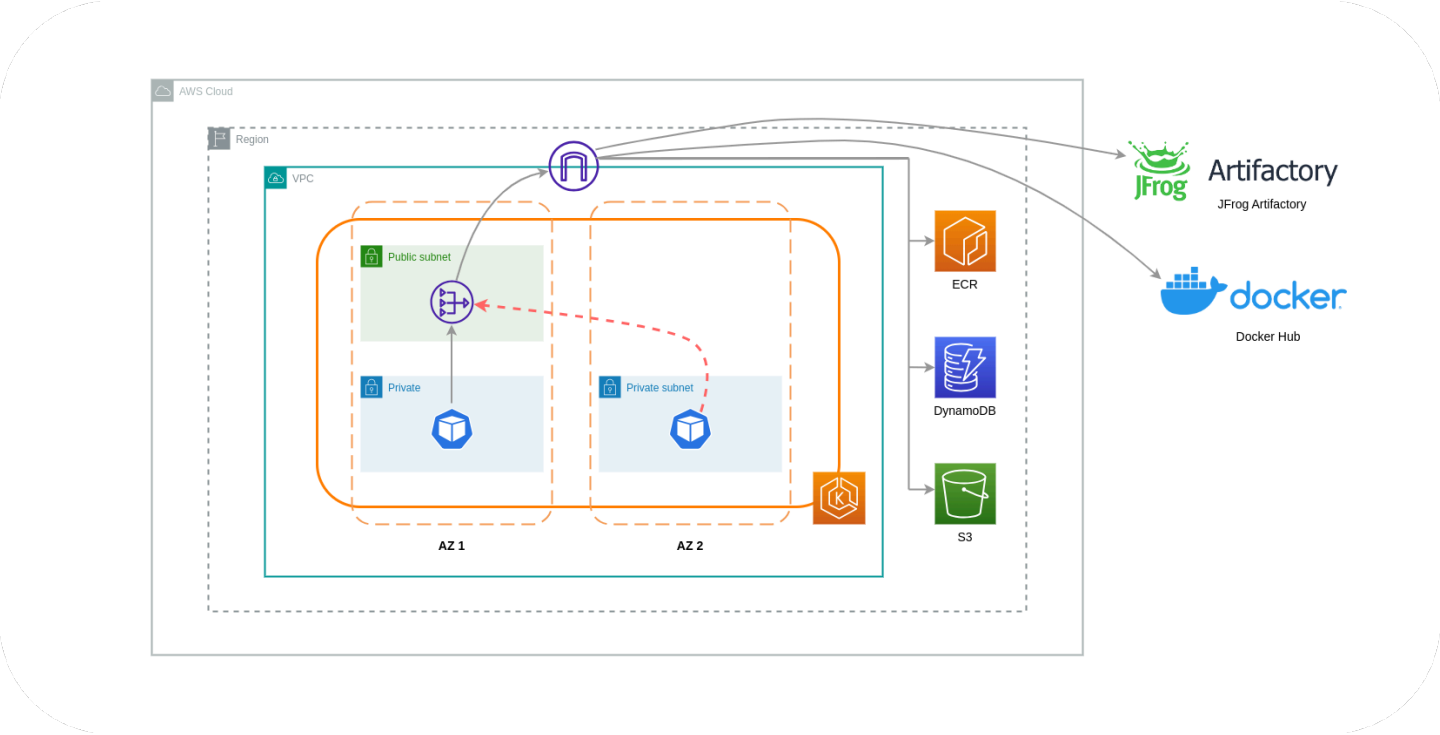
Si ejecuta cargas de trabajo con imágenes especialmente grandes, puede crear sus propias Amazon Machine Images (AMIs) personalizadas con imágenes de contenedores prealmacenadas en caché. Esto puede reducir el tiempo inicial de extracción de la imagen y los posibles costes de transferencia de datos desde un registro de contenedores a los nodos de trabajo de EKS.

Transferencia de datos a Internet y a los servicios de AWS

Es una práctica habitual integrar las cargas de trabajo de Kubernetes con otros servicios de AWS o herramientas y plataformas de terceros a través de Internet. La infraestructura de red subyacente utilizada para enrutar el tráfico hacia y desde el destino correspondiente puede afectar a los costos incurridos en el proceso de transferencia de datos.

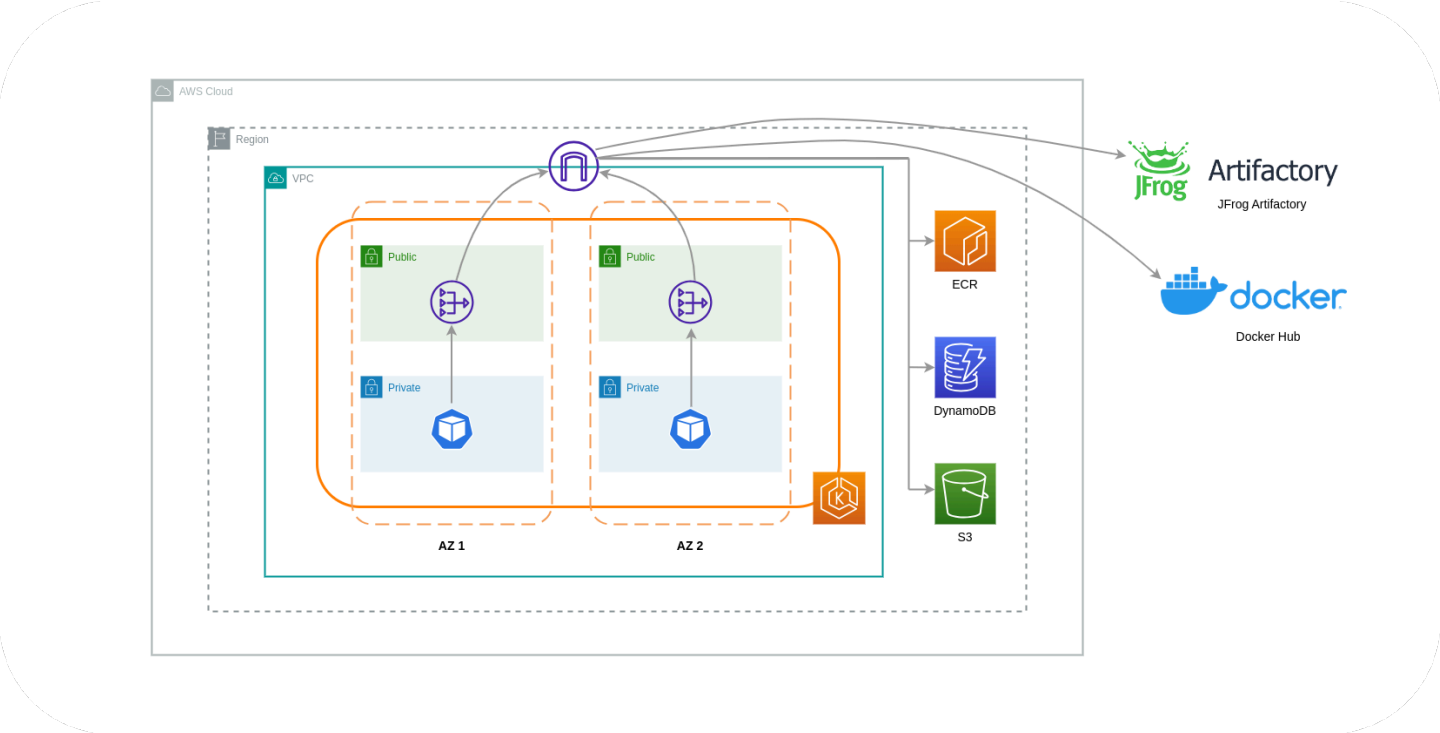
Uso de pasarelas NAT

Las puertas de enlace NAT son componentes de red que realizan la traducción de direcciones de red (NAT). El siguiente diagrama muestra los pods de un clúster de EKS que se comunican con otros servicios de AWS (Amazon ECR, DynamoDB y S3) y plataformas de terceros. En este ejemplo, los pods se ejecutan en subredes privadas separadas. AZs Para enviar y recibir tráfico de Internet, se implementa una puerta de enlace NAT en la subred pública de una zona de disponibilidad, lo que permite que cualquier recurso con direcciones IP privadas comparta una única dirección IP pública para acceder a Internet. Esta puerta de enlace NAT, a su vez, se comunica con el componente de puerta de enlace de Internet, lo que permite enviar los paquetes a su destino final.



Al utilizar pasarelas NAT para estos casos de uso, puede minimizar los costes de transferencia de datos mediante la implementación de una pasarela NAT en cada zona de disponibilidad. De esta forma, el tráfico dirigido a Internet pasará por la puerta de enlace NAT en la misma zona de disponibilidad, lo que evitará la transferencia de datos entre zonas de disponibilidad. Sin embargo, aunque ahorrará en el costo de la transferencia de datos entre zonas de disponibilidad, esta configuración implica que incurrirá en el costo de una puerta de enlace NAT adicional en su arquitectura.

Este enfoque recomendado se muestra en el siguiente diagrama.



Usar un punto de enlace de la VPC

Para reducir aún más los costos en dichas arquitecturas, debe usar los puntos de enlace de VPC para establecer la conectividad entre sus cargas de trabajo y los servicios de AWS. Los puntos de enlace de la VPC le permiten acceder a los servicios de AWS desde una VPC sin data/network que los paquetes atraviesen Internet. Todo el tráfico es interno y permanece dentro de la red de AWS. Hay dos tipos de puntos de enlace de VPC: puntos de enlace de VPC de interfaz (compatibles con muchos servicios de AWS) y puntos de enlace de VPC de puerta de enlace (solo compatibles con S3 y DynamoDB).

Puntos finales de VPC de Gateway

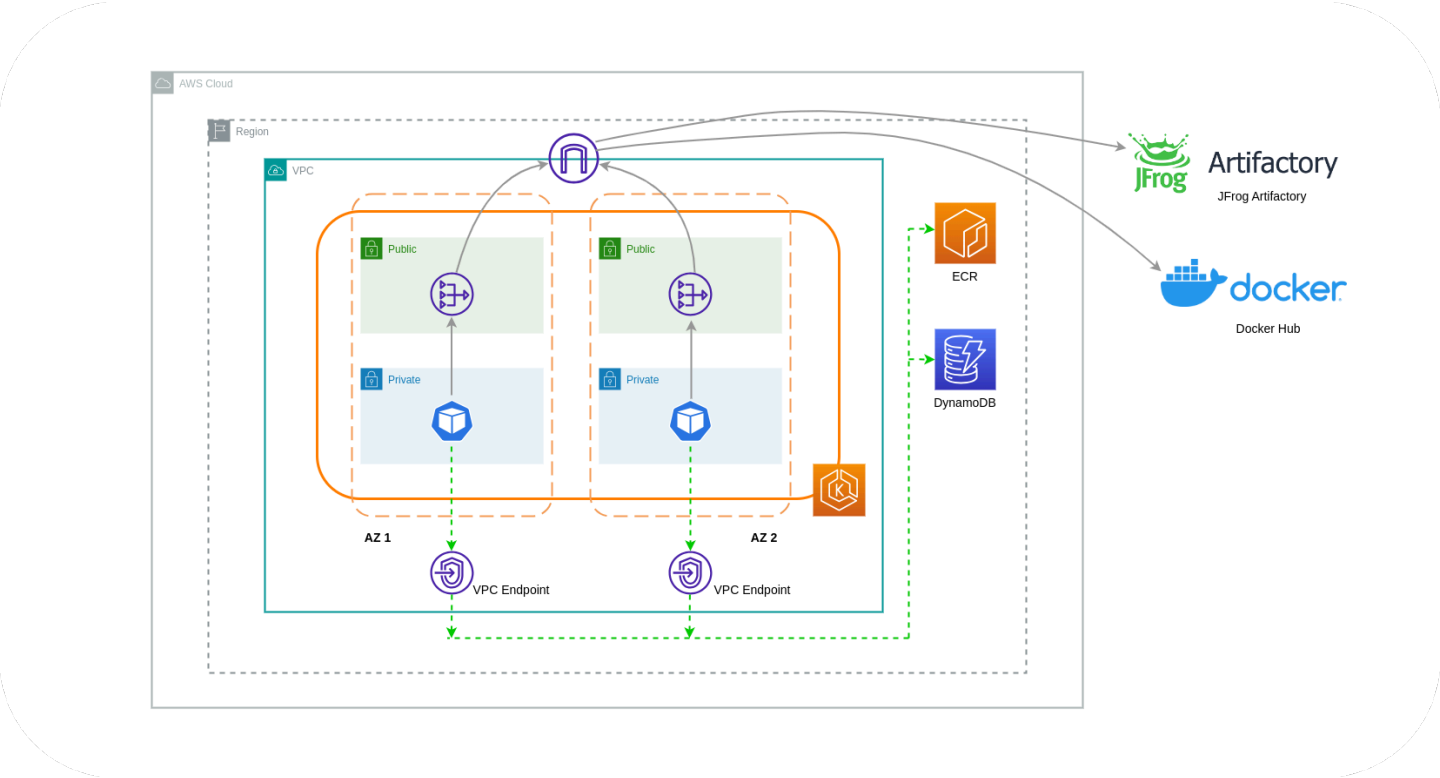
No hay costes por hora ni de transferencia de datos asociados a los puntos finales de VPC de Gateway. Al utilizar los puntos de enlace de VPC de Gateway, es importante tener en cuenta que no se pueden extender más allá de los límites de la VPC. No se pueden usar en la interconexión de VPC, en redes VPN ni a través de Direct Connect.

Puntos finales de VPC de interfaz

Los puntos finales de VPC tienen un cargo por hora y un cargo adicional asociado al procesamiento de datos a través del ENI subyacente. Tenga en cuenta que la transferencia de datos entre zonas

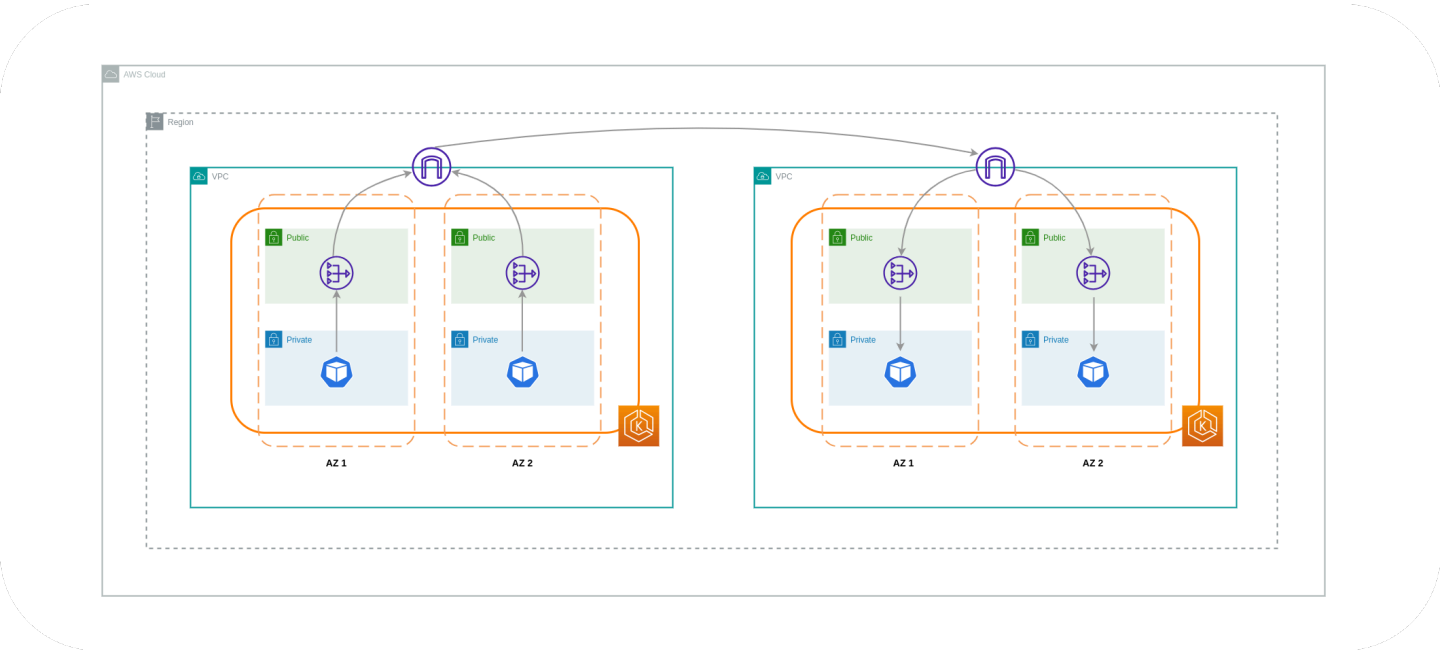
de disponibilidad [no se cobra] (https://aws.amazon.com/about-aws/aws-data-transfer-pricewhats-new/2022/04/ - -/). reduction-privatelink-transit-gateway client-vpn-services

En el siguiente diagrama, se muestran los pods que se comunican con los servicios de AWS a través de puntos de enlace de VPC.



Transferencia de datos entre VPCs

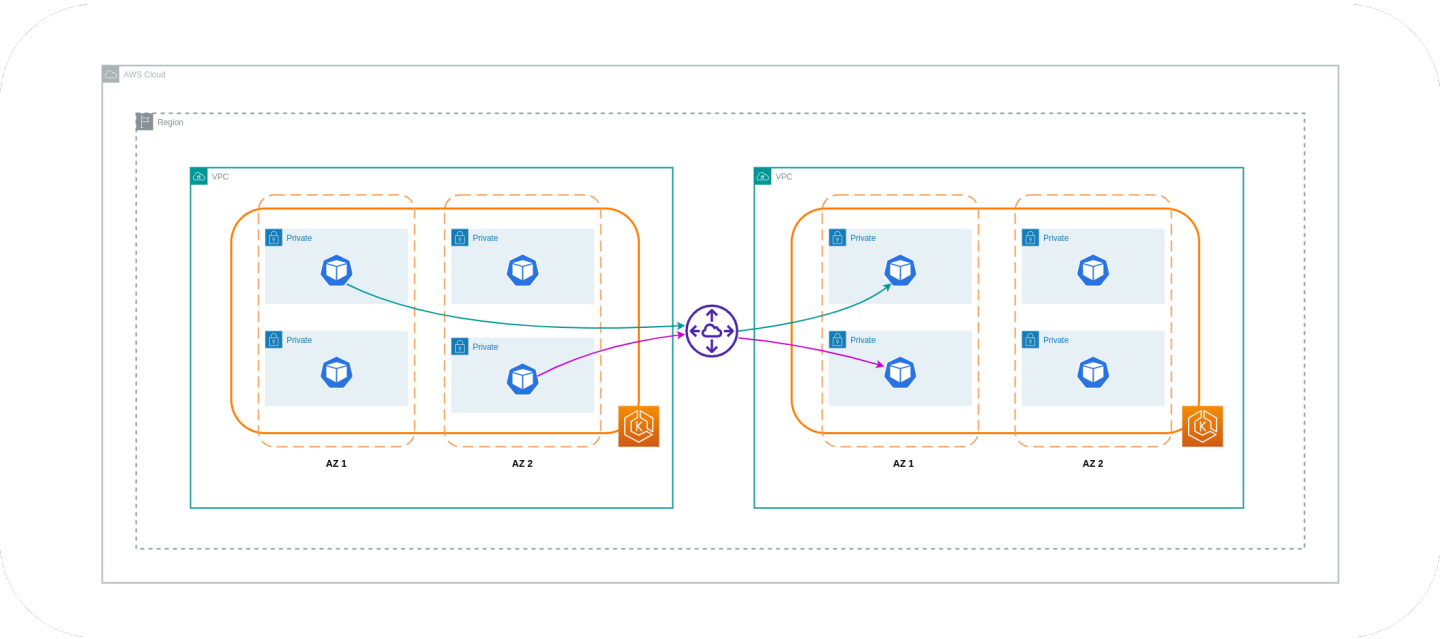
En algunos casos, es posible que tenga cargas de trabajo distintas VPCs (dentro de la misma región de AWS) que necesiten comunicarse entre sí. Esto se puede lograr permitiendo que el tráfico atraviese la Internet pública a través de las pasarelas de Internet conectadas a las respectivas. VPCs Esta comunicación se puede habilitar mediante la implementación de componentes de infraestructura como EC2 instancias, pasarelas NAT o instancias NAT en subredes públicas. Sin embargo, una configuración que incluya estos componentes generará cargos por la entrada y salida de processing/transferring los datos. VPCs Si el tráfico hacia y desde la unidad VPCs es transversal AZs, se cobrará un cargo adicional por la transferencia de datos. El siguiente diagrama muestra una configuración que utiliza puertas de enlace NAT y puertas de enlace de Internet para establecer la comunicación entre cargas de trabajo de diferentes tipos. VPCs



Interconexiones de VPC

Para reducir los costes de estos casos de uso, puede utilizar el emparejamiento de VPC. Con una conexión de interconexión de VPC, no se cobran cargos por transferencia de datos para el tráfico de red que permanece dentro de la misma zona de disponibilidad. Si el tráfico se cruza AZs, se incurrirá en un coste. No obstante, se recomienda el enfoque de emparejamiento de VPC para una comunicación rentable entre cargas de trabajo independientes dentro de VPCs la misma región de AWS. Sin embargo, es importante tener en cuenta que la interconexión de VPC es principalmente eficaz para la conectividad de VPC 1:1 porque no permite la creación de redes transitivas.

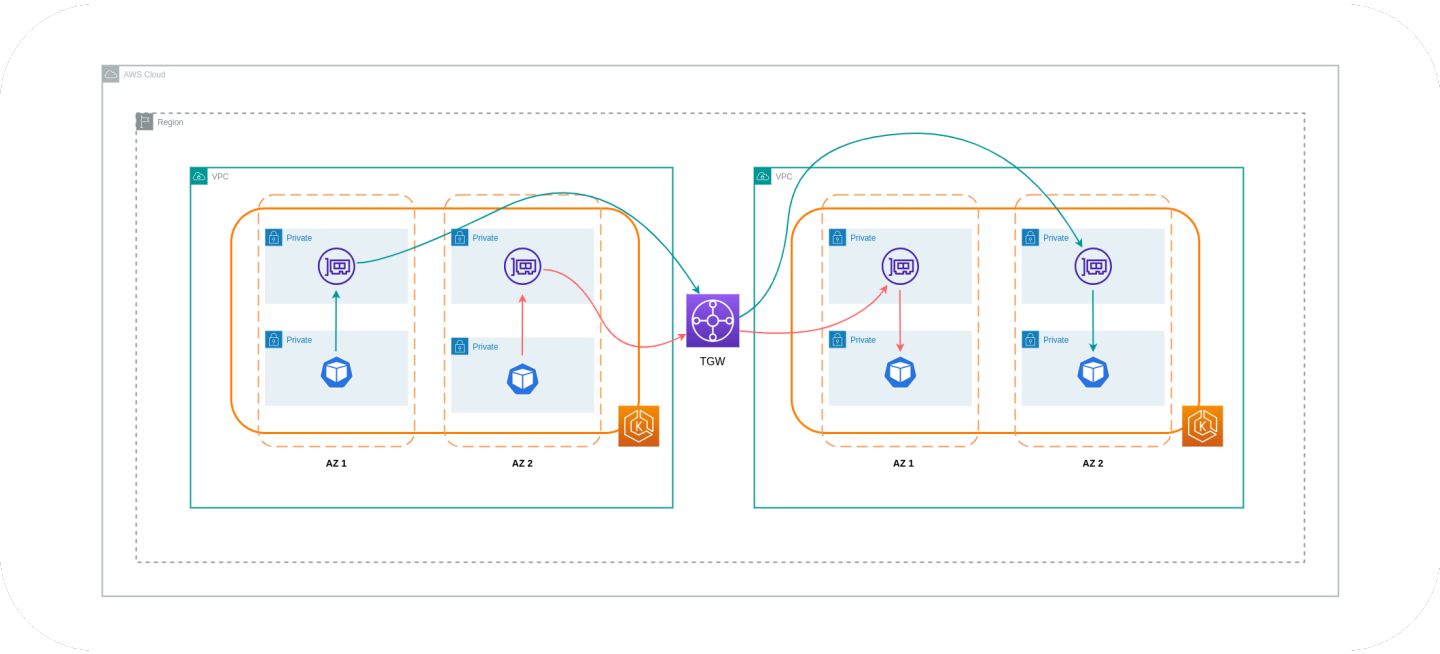
El siguiente diagrama es una representación de alto nivel de la comunicación de las cargas de trabajo a través de una conexión de emparejamiento de VPC.



Conexiones de red transitivas

Como se señaló en la sección anterior, las conexiones de interconexión de VPC no permiten la conectividad de red transitiva. Si desea conectar 3 o más VPCs con requisitos de red transitiva, debe utilizar una Transit Gateway (TGW). Esto le permitirá superar los límites del emparejamiento de VPC o cualquier sobrecarga operativa asociada a tener varias conexiones de emparejamiento de VPC entre varias. VPCs Se le facturará por hora y por los datos que se envíen al TGW. No hay ningún coste de destino asociado al tráfico entre zonas de disponibilidad que circula por el TGW.

El siguiente diagrama muestra el tráfico entre zonas de disponibilidad que fluye a través de un TGW entre cargas de trabajo de distintas regiones de AWS VPCs , pero dentro de la misma.



Uso de una malla de servicios

Las mallas de servicio ofrecen potentes capacidades de red que se pueden utilizar para reducir los costes relacionados con la red en los entornos de clústeres de EKS. Sin embargo, debe considerar detenidamente las tareas operativas y la complejidad que una malla de servicios introducirá en su entorno si la adopta.

Restringir el tráfico a las zonas de disponibilidad

Uso de la distribución ponderada por localidad de Istio

Istio le permite aplicar políticas de red al tráfico después de que se produzca el enrutamiento. Esto se hace mediante reglas de destino, como la distribución ponderada por localidad. Con esta función, puede controlar el peso (expresado como porcentaje) del tráfico que puede dirigirse a un destino determinado en función de su origen. El origen de este tráfico puede provenir de un balanceador de cargas externo (o público) o de un pod dentro del propio clúster. Cuando todos los puntos finales del pod estén disponibles, la localidad se seleccionará en función de un algoritmo de equilibrio de carga ponderado por turnos. En el caso de que algunos puntos finales no estén en buen estado o no estén disponibles, el peso de la localidad se ajustará automáticamente para reflejar este cambio en los puntos finales disponibles.

> **Note**
>
> Antes de implementar la distribución ponderada por localidad, debe empezar por comprender los patrones de tráfico de su red y las implicaciones que la política de reglas de destino puede tener en el comportamiento de su aplicación. Por ello, es importante contar con mecanismos de rastreo distribuidos con herramientas como AWS X-Ray o Jaeger.

Las reglas de destino de Istio detalladas anteriormente también se pueden aplicar para administrar el tráfico desde un balanceador de cargas a los pods de tu clúster de EKS. Las reglas de distribución ponderadas por localidad se pueden aplicar a un servicio que recibe tráfico de un balanceador de cargas de alta disponibilidad (específicamente, la puerta de enlace de entrada). Estas reglas te permiten controlar la cantidad de tráfico que va a cada lugar en función de su origen zonal (en este caso, el balanceador de carga). Si se configura correctamente, se generará menos tráfico de salida entre zonas en comparación con un balanceador de cargas que distribuya el tráfico de manera uniforme o aleatoria entre las réplicas de Pod de distintas zonas. AZs

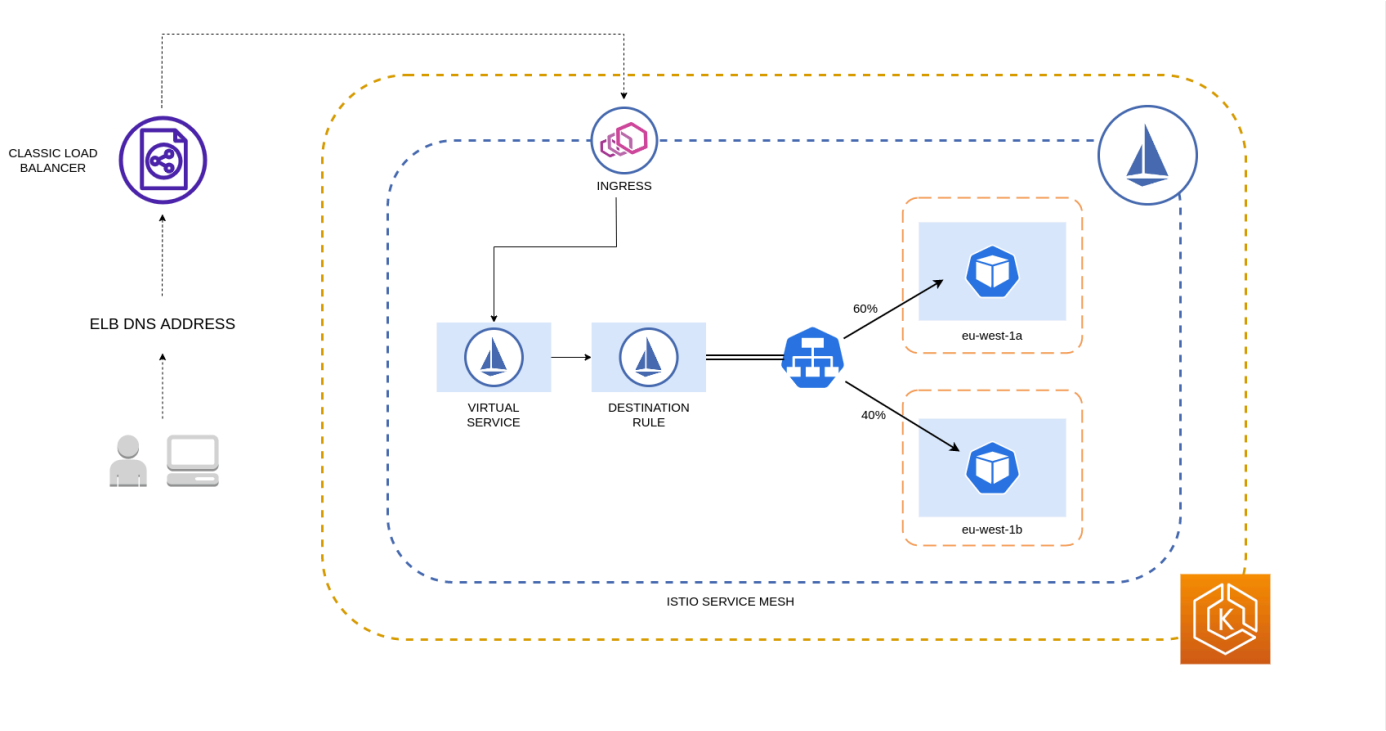
A continuación, se muestra un ejemplo de bloque de código de un recurso de regla de destino en Istio. Como se puede ver a continuación, este recurso especifica las configuraciones ponderadas para el tráfico entrante de 3 países diferentes AZs de la `eu-west-1` región. Estas configuraciones declaran que la mayoría del tráfico entrante (el 70% en este caso) de una zona de disponibilidad determinada debe enviarse mediante proxy a un destino de la misma zona de origen.

```
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**
>
> El peso mínimo que se puede distribuir a un destino es del 1%. El motivo es mantener las regiones y zonas de conmutación por error en caso de que los puntos finales del destino principal dejen de funcionar o no estén disponibles.

El siguiente diagrama muestra un escenario en el que hay un balanceador de carga de alta disponibilidad en la región eu-west-1 y se aplica una distribución ponderada por localidad. La política de reglas de destino de este diagrama está configurada para enviar el 60% del tráfico procedente de eu-west-1a a los pods de la misma zona de disponibilidad, mientras que el 40% del tráfico de eu-west-1a debe dirigirse a los pods de eu-west-1b.



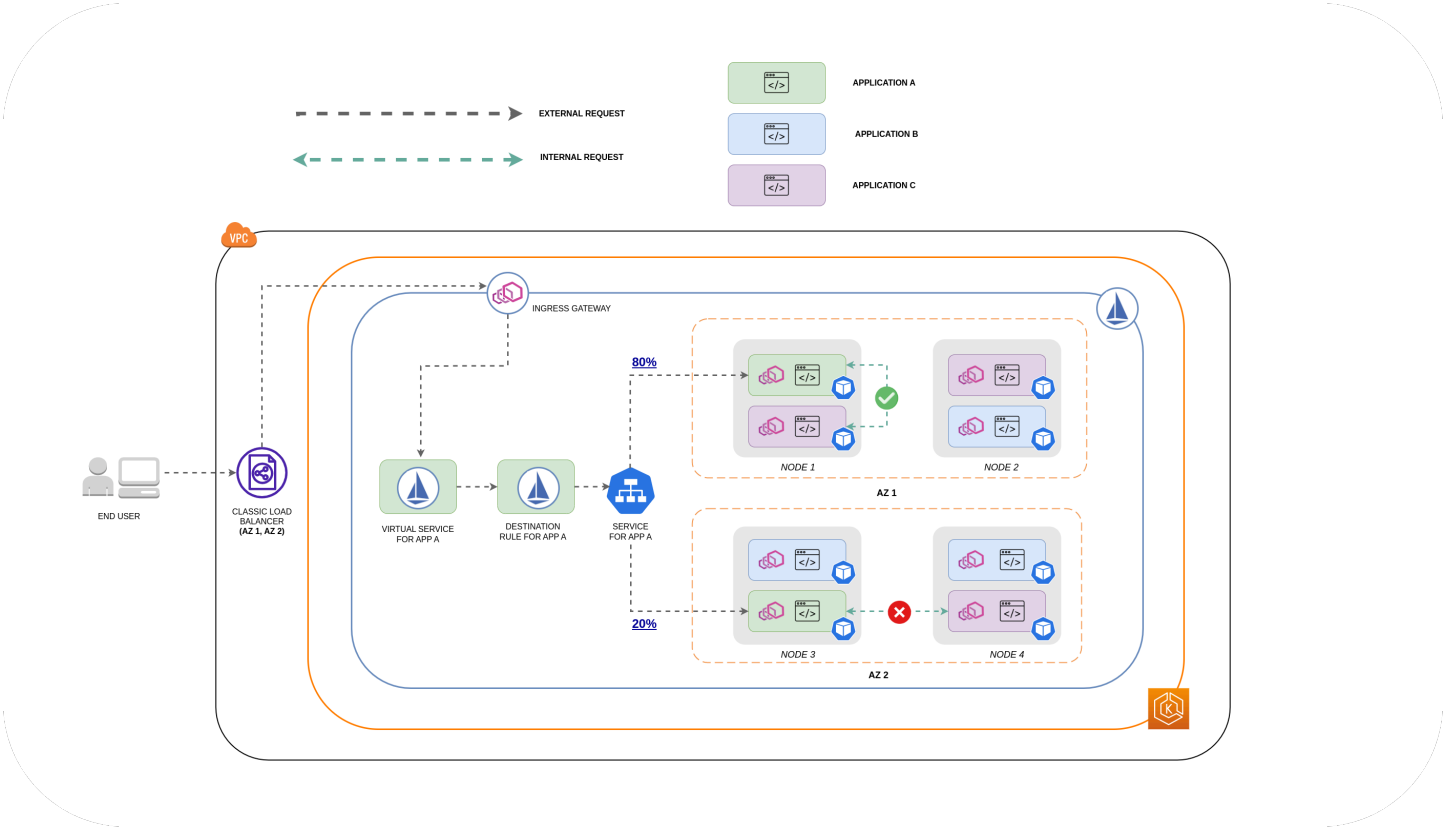
Restringir el tráfico a las zonas y nodos de disponibilidad

Uso de la política de tráfico interno del servicio con Istio

Para reducir los costes de red asociados al tráfico entrante externo y al tráfico interno entre los pods, puedes combinar las reglas de destino de Istio y la política de tráfico interna del Servicio Kubernetes. La forma de combinar las reglas de destino de Istio con la política de tráfico interna del servicio dependerá en gran medida de tres factores:

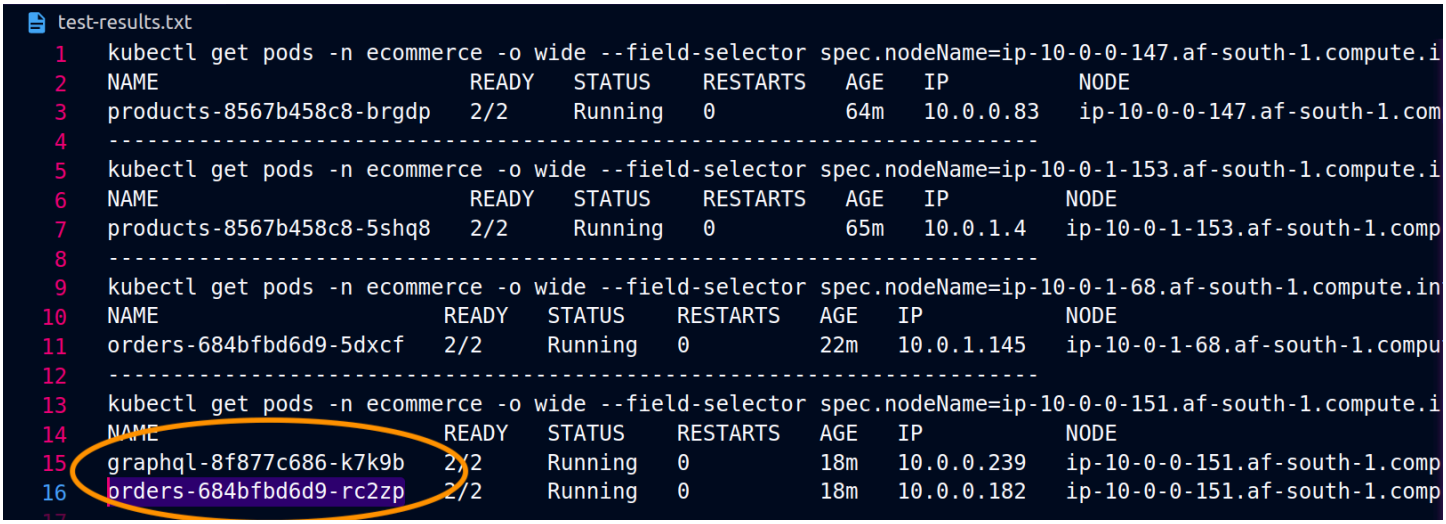
- El papel de los microservicios
- Patrones de tráfico de red en los microservicios
- Cómo deben implementarse los microservicios en la topología de clústeres de Kubernetes

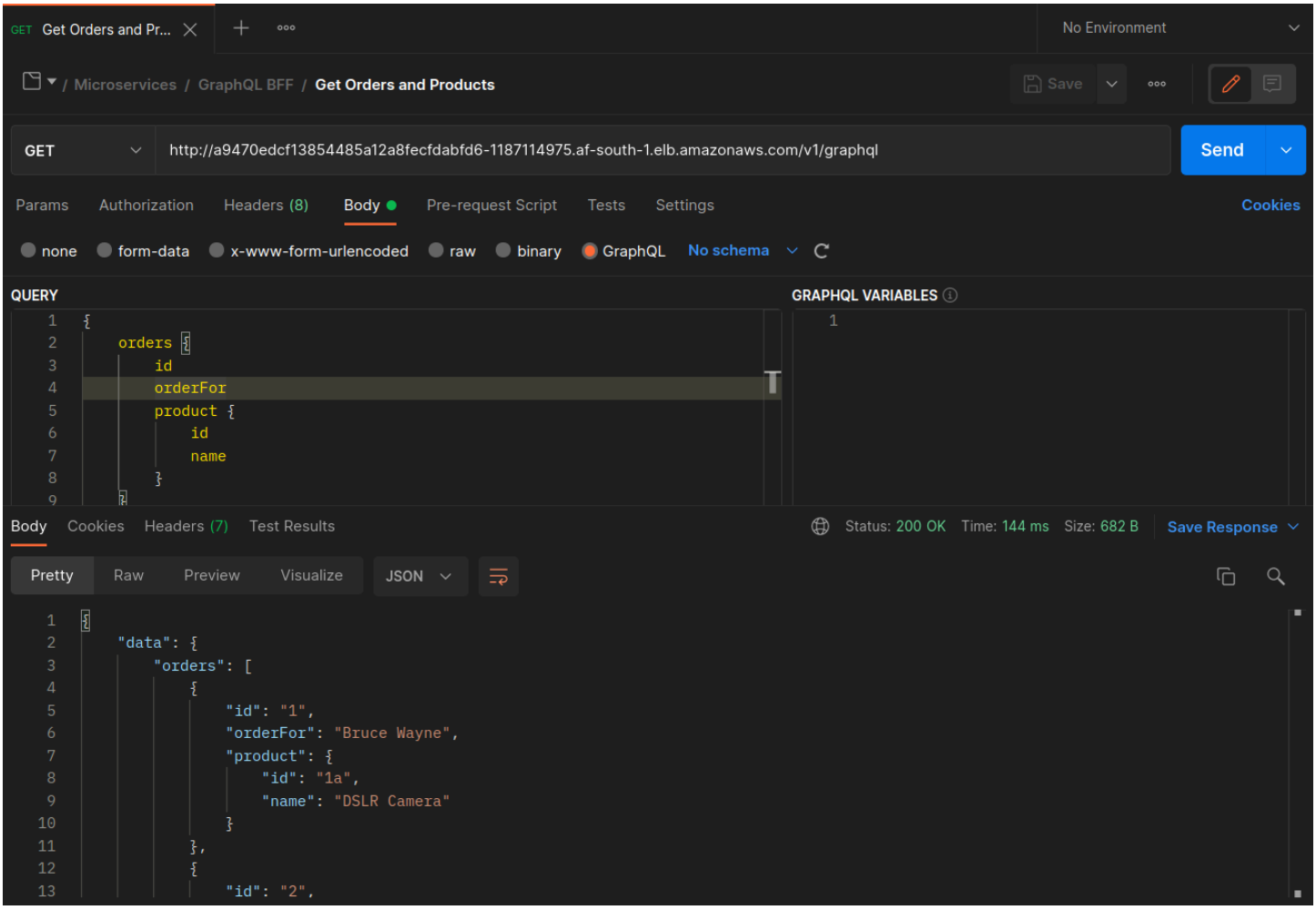
El siguiente diagrama muestra cómo sería el flujo de red en el caso de una solicitud anidada y cómo las políticas antes mencionadas controlarían el tráfico.



1. El usuario final realiza una solicitud a la APLICACIÓN A, que a su vez realiza una solicitud anidada a la APLICACIÓN C. Esta solicitud se envía primero a un balanceador de cargas de alta disponibilidad, que tiene instancias en la AZ 1 y la AZ 2, como se muestra en el diagrama anterior.
2. A continuación, el servicio virtual de Istio enruta la solicitud entrante externa al destino correcto.
3. Una vez enrutada la solicitud, la regla de destino de Istio controla la cantidad de tráfico que se dirige al respectivo país en AZs función del lugar de origen (AZ 1 o AZ 2).
4. A continuación, el tráfico se dirige al servicio de la APLICACIÓN A y, a continuación, se envía mediante proxy a los puntos finales del pod correspondientes. Como se muestra en el diagrama, el 80% del tráfico entrante se envía a los puntos finales del pod en la AZ 1 y el 20% del tráfico entrante se envía a la AZ 2.
5. A continuación, la APLICACIÓN A hace una solicitud interna a la APLICACIÓN C. El servicio de APP C tiene habilitada una política de tráfico interna (`internalTrafficPolicy`: Local`).
6. La solicitud interna de la APLICACIÓN A (en el NODO 1) a la APLICACIÓN C se ha realizado correctamente debido al punto final local del nodo disponible para la APLICACIÓN C.
7. La solicitud interna de la APLICACIÓN A (en el NODO 3) a la APLICACIÓN C falla porque no hay puntos de enlace locales de nodo disponibles para la APLICACIÓN C. Como se muestra en el diagrama, la APLICACIÓN C no tiene réplicas en el NODO 3. *

Las siguientes capturas de pantalla están capturadas a partir de un ejemplo en vivo de este enfoque. El primer conjunto de capturas de pantalla muestra una solicitud externa correcta a una `graphql` y una solicitud anidada correcta desde una `orders` réplica ubicada en el `graphql` mismo nodo. `ip-10-0-0-151.af-south-1.compute.internal`





Con Istio, puede verificar y exportar las estadísticas de cualquier [clúster upstream] (https://www.envoyproxy). io/docs/envoy/latest/intro/arch_overview/intro/terminology) y puntos finales que sus proxies conocen. Esto puede ayudar a proporcionar una imagen del flujo de la red, así como de la distribución entre los servicios de una carga de trabajo. Siguiendo con el mismo ejemplo, los `orders` puntos finales que conoce el `graphql` proxy se pueden obtener mediante el siguiente comando:

```
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**
...
```

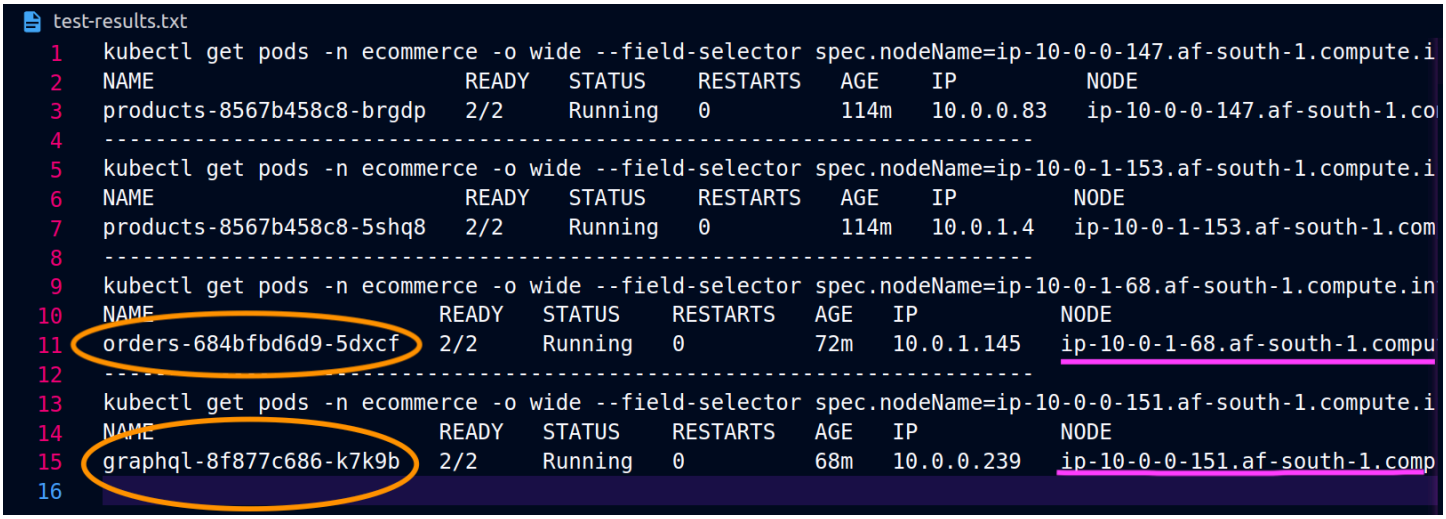
En este caso, el `graphql` proxy solo conoce el `orders` punto final de la réplica con el que comparte un nodo. Si eliminas la `internalTrafficPolicy: Local` configuración del servicio de pedidos y vuelves a ejecutar un comando como el anterior, los resultados devolverán todos los puntos finales de las réplicas repartidos por los distintos nodos. Además, al examinar los `rq_total` puntos finales respectivos, observará una participación relativamente uniforme en la distribución de la red. En consecuencia, si los puntos finales están asociados a servicios ascendentes que se ejecutan en diferentes zonas AZs, esta distribución de la red entre zonas generará costes más altos.

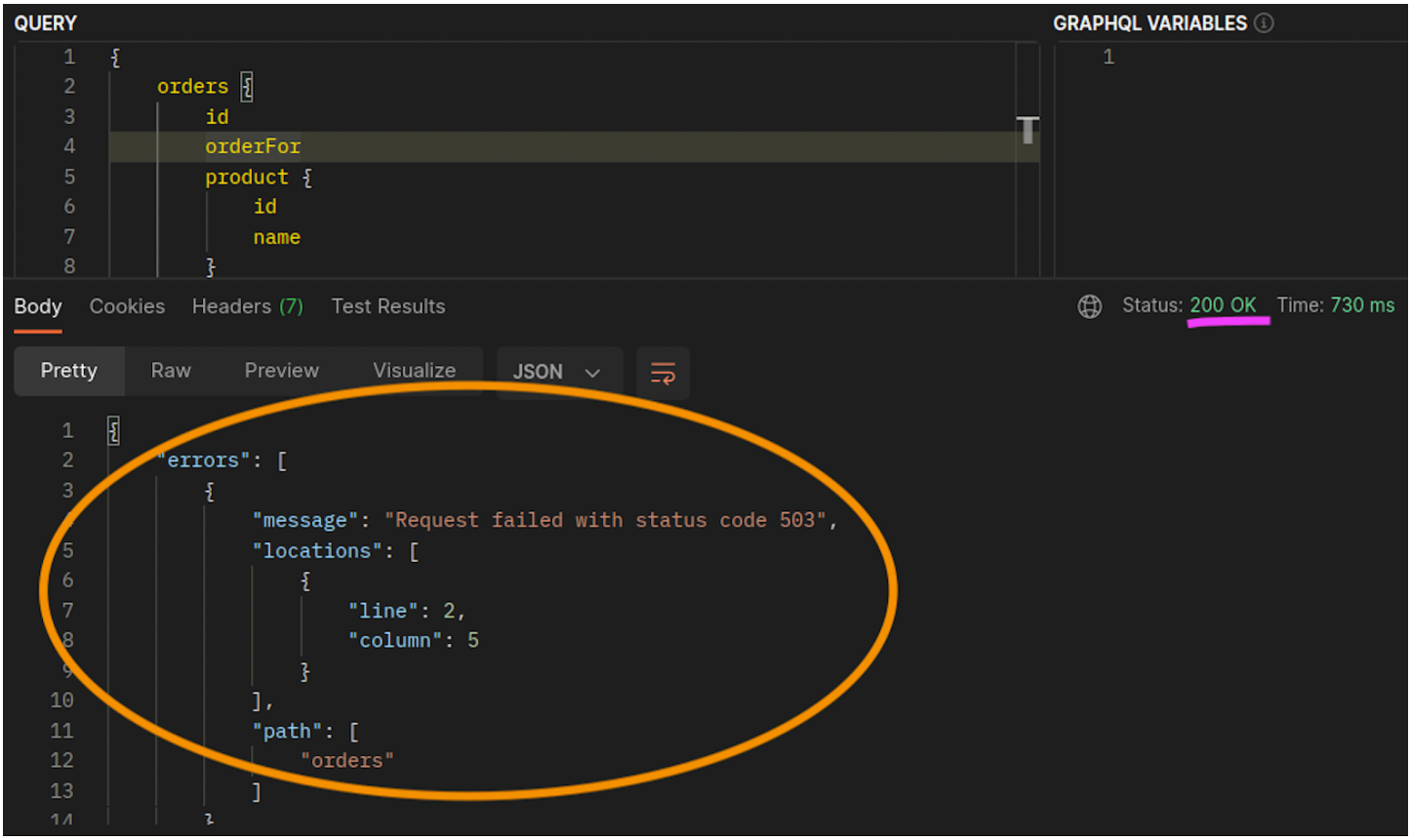
Como se mencionó anteriormente en la sección anterior, puedes ubicar en un mismo lugar los pods que se comunican con frecuencia si utilizas la afinidad con los pods.

```
...
spec:
...
  template:
    metadata:
      labels:
        app: graphql
        role: api
        workload: ecommerce
```

```
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - orders
            topologyKey: "kubernetes.io/hostname"
      nodeSelector:
        managedBy: karpenter
        billing-team: ecommerce
...
```

Cuando las `orders` réplicas `graphql` y las réplicas no coexisten en el mismo nodo (`ip-10-0-0-151.af-south-1.compute.internal`), la primera solicitud a `graphql` se realiza correctamente, como se muestra `200 response code` en la siguiente captura de pantalla de Postman, mientras que la segunda solicitud anidada de to falla con a. `graphql` `orders` `503 response code`





Recursos adicionales

- Abordar los costos de latencia y transferencia de datos en EKS con Istio
- Análisis del efecto de las sugerencias basadas en la topología en el tráfico de red en Amazon Elastic Kubernetes Service
- Obtener visibilidad de los bytes de red de pod a pod de Amazon EKS Cross-AZ

- Optimice el tráfico AZ con Istio
- Optimice el tráfico AZ con un enrutamiento compatible con la topología
- Optimice el costo y el rendimiento de Kubernetes con la política de tráfico interno del servicio
- Optimice el costo y el rendimiento de Kubernetes con Istio y la política de tráfico interno del servicio
- Overview of Data Transfer Costs for Common Architectures
- Descripción de los costos de transferencia de datos de los servicios de contenedores de AWS

Almacenamiento

Descripción general

Hay situaciones en las que es posible que desee ejecutar aplicaciones que necesiten conservar los datos a corto o largo plazo. Para estos casos de uso, los pods pueden definir y montar los volúmenes para que sus contenedores puedan acceder a diferentes mecanismos de almacenamiento. Kubernetes admite diferentes tipos de volúmenes para el almacenamiento efímero y persistente. La elección del almacenamiento depende en gran medida de los requisitos de la aplicación. Cada enfoque tiene implicaciones en términos de costos y las prácticas que se detallan a continuación le ayudarán a lograr la rentabilidad de las cargas de trabajo que necesitan algún tipo de almacenamiento en sus entornos de EKS.

Volúmenes efímeros

Los volúmenes efímeros son para aplicaciones que requieren volúmenes locales transitorios, pero que no requieren que los datos se conserven después de reiniciarse. Algunos ejemplos de ello son los requisitos de espacio libre, el almacenamiento en caché y los datos de entrada de solo lectura, como los datos de configuración y los secretos. Puedes encontrar más información sobre los volúmenes efímeros de Kubernetes aquí. La mayoría de los volúmenes efímeros (por ejemplo, emptyDir, ConfigMap, DownwardAPI, secret, hostpath) están respaldados por dispositivos grabables conectados localmente (normalmente el disco raíz) o RAM, por lo que es importante elegir el volumen de host más rentable y con mejor rendimiento.

Uso de volúmenes de EBS

Recomendamos empezar con gp3 como volumen raíz del host. Es el último volumen SSD de uso general ofrecido por Amazon EBS y también ofrece un precio más bajo (hasta un 20%) por GB en comparación con los volúmenes gp2.

Uso de Amazon EC2 Instance Stores

Los almacenes de EC2 instancias de Amazon proporcionan almacenamiento temporal a nivel de bloque para tus EC2 instancias. Se puede acceder al almacenamiento que proporcionan los almacenes de EC2 instancias a través de discos que están conectados físicamente a los hosts. A diferencia de Amazon EBS, solo puede adjuntar volúmenes del almacén de instancias cuando se lanza la instancia, y estos volúmenes solo existen durante la vida útil de la instancia. No se pueden separar ni volver a conectar a otras instancias. Puedes obtener más información sobre los almacenes de EC2 instancias de Amazon aquí. No hay cargos adicionales asociados al volumen de un almacén de instancias. Esto las hace (volúmenes de almacenamiento de instancias) más rentables que las EC2 instancias generales con grandes volúmenes de EBS.

Para usar volúmenes de almacenamiento local en Kubernetes, debes particionar, configurar y formatear los discos con los EC2 datos de usuario de Amazon para que los volúmenes se puedan montar según las especificaciones del pod HostPath. Como alternativa, puedes aprovechar el aprovisionador estático de volúmenes persistentes locales para simplificar la administración del almacenamiento local. El aprovisionador estático de volúmenes persistentes locales le permite acceder a los volúmenes de los almacenes de instancias locales a través de la interfaz estándar de Kubernetes PersistentVolumeClaim (PVC). Además, proporcionará PersistentVolumes (PVs), que contiene información sobre la afinidad de los nodos, para programar los pods en los nodos correctos. Aunque usa Kubernetes PersistentVolumes, los volúmenes de los almacenes de EC2 instancias son de naturaleza efímera. Los datos escritos en discos efímeros solo están disponibles durante la vida útil de la instancia. Cuando la instancia finaliza, también lo hacen los datos. Consulte este blog para obtener más información.

Ten en cuenta que cuando usas volúmenes del almacén de EC2 instancias de Amazon, el límite total de IOPS se comparte con el host y vincula los pods a un host específico. Debe revisar minuciosamente sus requisitos de carga de trabajo antes de adoptar los volúmenes de almacenes de EC2 instancias de Amazon.

Volúmenes persistentes

Kubernetes suele estar asociado a la ejecución de aplicaciones sin estado. Sin embargo, hay situaciones en las que es posible que desee ejecutar microservicios que necesiten conservar los datos o la información persistentes de una solicitud a otra. Las bases de datos son un ejemplo habitual de estos casos de uso. Sin embargo, los pods y los contenedores o procesos que contienen son de naturaleza efímera. Para conservar los datos más allá de la vida útil de un pod, puedes definir el acceso al almacenamiento en una ubicación específica que sea independiente del pod. PVs Los

costes asociados PVs dependen en gran medida del tipo de almacenamiento que se utilice y de cómo lo consuman las aplicaciones.

Aquí se enumeran diferentes tipos de opciones de almacenamiento compatibles con Kubernetes en PVs Amazon EKS. Las opciones de almacenamiento que se describen a continuación son Amazon EBS, Amazon EFS, Amazon FSx for Lustre y Amazon FSx for NetApp ONTAP.

Volúmenes de Amazon Elastic Block Store (EBS)

Los volúmenes de Amazon EBS se pueden consumir como Kubernetes PVs para proporcionar volúmenes de almacenamiento a nivel de bloque. Son ideales para bases de datos que se basan en lecturas y escrituras aleatorias y para aplicaciones de alto rendimiento que realizan lecturas y escrituras prolongadas y continuas. El controlador de la interfaz de almacenamiento de contenedores (CSI) de Amazon Elastic Block Store permite a los clústeres de Amazon EKS gestionar el ciclo de vida de los volúmenes de Amazon EBS para los volúmenes persistentes. La interfaz de almacenamiento en contenedores permite y facilita la interacción entre Kubernetes y un sistema de almacenamiento. Cuando se implementa un controlador CSI en su clúster de EKS, puede acceder a sus capacidades a través de los recursos de almacenamiento nativos de Kubernetes, como Persistent Volumes (PVs), Persistent Volume Claims () y Storage Classes (PVCs). SCs Este enlace proporciona ejemplos prácticos de cómo interactuar con los volúmenes de Amazon EBS con el controlador CSI de Amazon EBS.

Elegir el volumen correcto

Recomendamos utilizar la última generación de almacenamiento en bloque (gp3), ya que proporciona el equilibrio adecuado entre precio y rendimiento. También le permite escalar el volumen de IOPS y el rendimiento independientemente del tamaño del volumen sin necesidad de aprovisionar capacidad adicional de almacenamiento en bloques. Si actualmente utiliza volúmenes gp2, le recomendamos encarecidamente que migre a los volúmenes gp3. La entrada del blog Migración de clústeres de Amazon EKS de volúmenes de EBS de gp2 a gp3 explica cómo migrar de gp2 a gp3 en clústeres de Amazon EKS con copia de seguridad y restauración mediante la función CSI Volume Snapshots, que requiere el tiempo de inactividad de las aplicaciones.

Amazon EBS permite cambiar las características del volumen, como el tamaño del volumen, las IOPS y el rendimiento en línea. Al utilizar esta función, se puede migrar de gp2 a gp3 sin tiempo de inactividad de la aplicación mediante anotaciones de PVC, tal como se describe en este blog, que requiere el controlador CSI de EBS v1.19.0+, o empezando por Amazon EKS v1.31 y el controlador CSI de EBS 1.35 mediante la API, tal como se describe aquí. VolumeAttributesClass

Si tiene aplicaciones que requieren un mayor rendimiento y necesitan volúmenes superiores a los que puede admitir un único volumen de gp3, debería considerar la posibilidad de utilizar io2 block express. Este tipo de almacenamiento es ideal para su implementación más grande, I/O intensiva y de misión crítica, como SAP HANA u otras bases de datos grandes con requisitos de baja latencia. Tenga en cuenta que el rendimiento de EBS de una instancia está limitado por los límites de rendimiento de la instancia, por lo que no todas las instancias admiten los volúmenes express de bloques de io2. Puedes comprobar los tipos de instancias compatibles y otras consideraciones en este documento.

Un solo volumen gp3 puede admitir hasta 16 000 IOPS como máximo, 1 000 con un rendimiento máximo y 16 TiB como MiB/s máximo. Volumen SSD de IOPS aprovisionadas de última generación que proporciona hasta 256 000 IOPS, 4000 MiB/s, rendimiento y 64 TiB.

Entre estas opciones, es la que mejor adapta el rendimiento y el coste del almacenamiento a las necesidades de sus aplicaciones.

Supervise y optimice a lo largo del tiempo

Es importante entender el rendimiento básico de la aplicación y monitorizarla para ver si se trata de volúmenes seleccionados para comprobar si cumple con las expectativas requirements/expectations o si está sobreaprovisionada (por ejemplo, en un escenario en el que las IOPS aprovisionadas no se utilizan al máximo).

En lugar de asignar un volumen grande desde el principio, puede aumentar gradualmente el tamaño del volumen a medida que acumula datos. Puede cambiar el tamaño de los volúmenes de forma dinámica mediante la función de redimensionamiento de volúmenes del controlador CSI de Amazon Elastic Block Store ()aws-ebs-csi-driver. Tenga en cuenta que solo puede aumentar el tamaño del volumen de EBS.

Para identificar y eliminar cualquier volumen de EBS pendiente, puede utilizar la categoría de optimización de costes de AWS Trusted Advisor. Esta función le ayuda a identificar los volúmenes independientes o los volúmenes con una actividad de escritura muy baja durante un período de tiempo. Existe una herramienta de solo lectura y código abierto nativa de la nube llamada Popeye que escanea los clústeres de Kubernetes activos e informa sobre posibles problemas con los recursos y las configuraciones desplegados. Por ejemplo, puede buscar los elementos no utilizados PVs PVCs y comprobar si están enlazados o si hay algún error al montar el volumen.

Para obtener información detallada sobre la monitorización, consulte la guía de observabilidad de la optimización de costes de EKS.

Otra opción que puede considerar son las recomendaciones de volumen de AWS Compute Optimizer para Amazon EBS. Esta herramienta identifica automáticamente la configuración de volumen óptima y el nivel correcto de rendimiento necesario. Por ejemplo, se puede utilizar para configurar de forma óptima las IOPS aprovisionadas, los tamaños de los volúmenes y los tipos de volúmenes de EBS, en función de la utilización máxima durante los últimos 14 días. También cuantifica los posibles ahorros de costos mensuales derivados de sus recomendaciones. Puede revisar este blog para obtener más detalles.

Política de retención de copias de seguridad

Puede realizar copias de seguridad de los datos de sus volúmenes de Amazon EBS realizando point-in-time instantáneas. El controlador CSI de Amazon EBS admite instantáneas de volumen. Puede obtener información sobre cómo crear una instantánea y restaurar un EBS PV siguiendo los pasos que se describen aquí.

Las instantáneas posteriores son copias de seguridad incrementales, lo que significa que solo se guardan los bloques del dispositivo que han cambiado después de la instantánea más reciente. Esto disminuye el tiempo necesario para crearlo y ahorra costos de almacenamiento, ya que no se duplican los datos. Sin embargo, aumentar el número de instantáneas antiguas de EBS sin una política de retención adecuada puede generar costes inesperados al operar a gran escala. Si va a realizar copias de seguridad de los volúmenes de Amazon EBS directamente a través de la API de AWS, puede utilizar Amazon Data Lifecycle Manager (DLM), que proporciona una solución de gestión del ciclo de vida automatizada y basada en políticas para las instantáneas de Amazon Elastic Block Store (EBS) y las Amazon Machine Images () respaldadas por EBS (). AMIs La consola facilita la automatización de la creación, la retención y la eliminación de las instantáneas de EBS y. AMIs

> **Note**
>
> Actualmente no hay forma de utilizar Amazon DLM a través del controlador CSI de Amazon EBS.

En un entorno de Kubernetes, puede utilizar una herramienta de código abierto llamada Velero para hacer copias de seguridad de sus volúmenes persistentes de EBS. Puede establecer un indicador TTL al programar el trabajo para que caduque las copias de seguridad. He aquí una guía de Velero como ejemplo.

Amazon Elastic File System (EFS)

Amazon Elastic File System (EFS) es un sistema de archivos totalmente elástico y sin servidor que le permite compartir datos de archivos mediante una interfaz de sistema de archivos estándar y una semántica del sistema de archivos para una amplia gama de cargas de trabajo y aplicaciones. Algunos ejemplos de cargas de trabajo y aplicaciones incluyen Wordpress y Drupal, herramientas para desarrolladores como JIRA y Git, y sistemas de libretas compartidas como Jupyter, así como directorios principales.

Una de las principales ventajas de Amazon EFS es que se puede montar en varios contenedores repartidos en varios nodos y varias zonas de disponibilidad. Otra ventaja es que solo paga por el almacenamiento que utilice. Los sistemas de archivos EFS crecerán y se reducirán automáticamente a medida que añada y elimine archivos, lo que elimina la necesidad de planificar la capacidad.

Para utilizar Amazon EFS en Kubernetes, debe utilizar el controlador Amazon Elastic File System Container Storage Interface (CSI),. aws-efs-csi-driver Actualmente, el controlador puede crear puntos de acceso de forma dinámica. Sin embargo, primero se debe aprovisionar el sistema de archivos Amazon EFS y proporcionarlo como entrada al parámetro de clase de almacenamiento de Kubernetes.

Cómo elegir la clase de almacenamiento EFS adecuada

Amazon EFS ofrece cuatro clases de almacenamiento.

Dos clases de almacenamiento estándar:

- Estándar Amazon EFS
- Amazon EFS Standard-Infrequent Access (EFS Standard-IA)

Dos clases de almacenamiento de una zona:

- Amazon EFS One Zone
- Amazon EFS One Zone-Infrequent Access (EFS One Zone-IA)

Las clases de almacenamiento de acceso infrecuente (IA) optimizan los costos para los archivos a los que no se accede todos los días. Con la administración del ciclo de vida de Amazon EFS, puede mover los archivos a los que no se haya accedido durante la vigencia de la política de ciclo de vida (7, 14, 30, 60 o 90 días) a las clases de almacenamiento de IA, lo que puede reducir el costo de

almacenamiento hasta un 92 por ciento en comparación con las clases de almacenamiento EFS Standard y EFS One Zone, respectivamente.

Con EFS Intelligent-Tiering, la administración del ciclo de vida supervisa los patrones de acceso de su sistema de archivos y mueve automáticamente los archivos a la clase de almacenamiento más óptima.

> **Note**
>
> aws-efs-csi-driver actualmente no controla los cambios en las clases de almacenamiento, la administración del ciclo de vida ni la organización inteligente por niveles. Deben configurarse manualmente en la consola de AWS o a través de EFS APIs.

> **Note**
>
> aws-efs-csi-driver no es compatible con las imágenes de contenedores basadas en Windows.

> **Note**
>
> Se sabe que hay un problema de memoria cuando vol-metrics-opt-in(emitir métricas de volumen) está habilitada debido a que la DiskUsagefunción consume una cantidad de memoria proporcional al tamaño del sistema de archivos. Actualmente, recomendamos desactivar la opción `-- vol-metrics-opt-in `en sistemas de archivos de gran tamaño para evitar consumir demasiada memoria. Aquí tienes un enlace a un problema de GitHub para obtener más información.

Amazon FSx para Lustre

Lustre es un sistema de archivos paralelos de alto rendimiento que se utiliza habitualmente en cargas de trabajo que requieren un rendimiento de hasta cientos GB/s y latencias de menos de un milisegundo por operación. Se utiliza para escenarios como la formación en aprendizaje automático, la elaboración de modelos financieros, la HPC y el procesamiento de vídeo. Amazon FSx for Lustre proporciona un almacenamiento compartido totalmente gestionado con escalabilidad y rendimiento integrados a la perfección con Amazon S3.

Puede usar los volúmenes de almacenamiento persistente de Kubernetes respaldados por FSx for Lustre mediante el FSx controlador CSI for Lustre de Amazon EKS o su clúster de Kubernetes autogestionado en AWS. Consulte la documentación de Amazon EKS para obtener más detalles y ejemplos.

Enlace a Amazon S3

Se recomienda vincular un repositorio de datos de larga duración y larga duración que resida en Amazon S3 con su sistema de archivos FSx for Lustre. Una vez enlazados, los conjuntos de datos de gran tamaño se cargan de forma diferida según sea necesario desde Amazon S3 hasta los sistemas de archivos FSx Lustre. También puede volver a ejecutar los análisis y los resultados en S3 y, a continuación, eliminar el sistema de archivos de Lustre.

Elegir las opciones de despliegue y almacenamiento adecuadas

FSx for Lustre ofrece diferentes opciones de despliegue. La primera opción se llama scratch y no replica los datos, mientras que la segunda opción se llama persistente, que, como su nombre lo indica, conserva los datos.

La primera opción (scratch) se puede utilizar para reducir el coste del procesamiento temporal de datos a corto plazo. La opción de implementación persistente está diseñada para un almacenamiento a largo plazo que replica automáticamente los datos dentro de una zona de disponibilidad de AWS. También es compatible con el almacenamiento en SSD y en disco duro.

Puede configurar el tipo de despliegue deseado en los parámetros de Kubernetes del sistema de archivos FSx for lustre. StorageClass Aquí hay un enlace que proporciona plantillas de muestra.

> **Note**
>
> Para las cargas de trabajo sensibles a la latencia o las cargas de trabajo que requieren los niveles más altos de IOPs/rendimiento, debe elegir el almacenamiento SSD. Para las cargas de trabajo centradas en el rendimiento que no son sensibles a la latencia, debe elegir el almacenamiento en disco duro.

Habilite la compresión de datos

También puede habilitar la compresión de datos en su sistema de archivos especificando LZ4 "» como tipo de compresión de datos. Una vez activado, todos los archivos recién escritos se

comprimirán automáticamente FSx para Lustre antes de escribirse en el disco y se descomprimirán cuando se lean. LZ4 El algoritmo de compresión de datos no tiene pérdidas, por lo que los datos originales se pueden reconstruir completamente a partir de los datos comprimidos.

Puede configurar el tipo de compresión de datos como se indica en los parámetros LZ4 del Kubernetes del sistema de FSx archivos for lustre. StorageClass La compresión se desactiva cuando el valor se establece en NONE, que es el valor predeterminado. Este enlace proporciona plantillas de ejemplo.

> **Note**
>
> Amazon FSx for Lustre no es compatible con las imágenes de contenedores basadas en Windows.

Amazon FSx para NetApp ONTAP

Amazon FSx for NetApp ONTAP es un almacenamiento compartido totalmente gestionado que se basa en el sistema NetApp de archivos ONTAP. FSx for ONTAP proporciona un almacenamiento de archivos compartido rápido, flexible y rico en funciones al que se puede acceder fácilmente desde instancias informáticas de Linux, Windows y macOS que se ejecutan en AWS o de forma local.

Amazon FSx for NetApp ONTAP admite dos niveles de almacenamiento: 1 nivel principal y 2 niveles de grupo de capacidad.

El nivel principal es un nivel aprovisionado y basado en SSD de alto rendimiento para datos activos y sensibles a la latencia. El nivel del pool de capacidad, totalmente elástico, optimiza los costes para los datos a los que se accede con poca frecuencia, se escala automáticamente a medida que los datos se agrupan en niveles y ofrece petabytes de capacidad prácticamente ilimitados. Puede habilitar la compresión y la deduplicación de datos en el almacenamiento del pool de capacidad y reducir aún más la cantidad de capacidad de almacenamiento que consumen sus datos. NetAppSu FabricPool función nativa basada en políticas monitorea continuamente los patrones de acceso a los datos y transfiere automáticamente los datos de forma bidireccional entre los niveles de almacenamiento para optimizar el rendimiento y los costes.

NetAppde Astra Trident ofrece una organización dinámica del almacenamiento mediante un controlador CSI que permite a los clústeres EKS de Amazon gestionar el ciclo de vida de los volúmenes persistentes respaldados PVs por Amazon FSx para los sistemas de archivos NetApp

ONTAP. Para empezar, consulta Cómo usar el Astra Trident con Amazon FSx para NetApp ONTAP en la documentación del Astra Trident.

Otras consideraciones

Minimice el tamaño de la imagen del contenedor

Una vez desplegados los contenedores, las imágenes de los contenedores se almacenan en caché en el host en varias capas. Al reducir el tamaño de las imágenes, se puede reducir la cantidad de almacenamiento necesaria en el host.

Al utilizar desde el principio imágenes básicas reducidas, como imágenes de borrador o imágenes de contenedores sin distribución (que solo contienen la aplicación y sus dependencias de tiempo de ejecución), puede reducir los costes de almacenamiento, además de otras ventajas adicionales, como la reducción del área de superficie de ataque y la reducción de los tiempos de extracción de las imágenes.

También debería considerar la posibilidad de utilizar herramientas de código abierto, como Slim.ai, que proporciona una forma fácil y segura de crear imágenes mínimas.

Varias capas de paquetes, herramientas, dependencias de aplicaciones y bibliotecas pueden aumentar fácilmente el tamaño de la imagen del contenedor. Al usar compilaciones de varias etapas, puedes copiar artefactos de forma selectiva de una etapa a otra, excluyendo todo lo que no sea necesario de la imagen final. Puedes consultar más prácticas recomendadas para la creación de imágenes aquí.

Otra cosa a tener en cuenta es cuánto tiempo se deben conservar las imágenes en caché. Es posible que desee limpiar las imágenes obsoletas de la memoria caché de imágenes cuando se utilice cierta cantidad de disco. Si lo hace, le ayudará a asegurarse de que dispone de suficiente espacio para el funcionamiento del host. De forma predeterminada, el kubelet recolecta basura en las imágenes no utilizadas cada cinco minutos y en los contenedores no utilizados cada minuto.

Para configurar las opciones de recolección de basura de contenedores e imágenes no utilizadas, ajuste el kubelet mediante un archivo de configuración y cambie los parámetros relacionados con la recolección de basura utilizando el tipo de recurso. *`KubeletConfiguration`*

Puedes obtener más información al respecto en la documentación de Kubernetes.

Observabilidad

Introducción

Las herramientas de observabilidad le ayudan a detectar, corregir e investigar sus cargas de trabajo de manera eficiente. El costo de los datos de telemetría aumenta naturalmente a medida que aumenta el uso de EKS. A veces, puede resultar difícil equilibrar sus necesidades operativas y medir lo que es importante para su empresa y mantener los costes de observabilidad bajo control. Esta guía se centra en las estrategias de optimización de costes para los tres pilares de la observabilidad: los registros, las métricas y los rastreos. Cada una de estas mejores prácticas se puede aplicar de forma independiente para adaptarse a los objetivos de optimización de su organización.

Registro

El registro desempeña un papel fundamental en la supervisión y la solución de problemas de las aplicaciones de su clúster. Existen varias estrategias que se pueden emplear para optimizar los costos de registro. Las estrategias recomendadas que se enumeran a continuación incluyen examinar sus políticas de retención de registros para implementar controles detallados sobre el tiempo que se conservan los datos de registro, enviar los datos de registro a diferentes opciones de almacenamiento según su importancia y utilizar el filtrado de registros para reducir los tipos de mensajes de registro que se almacenan. La administración eficiente de la telemetría de registros puede suponer un ahorro de costes para sus entornos.

Plano de control EKS

Optimice los registros de su plano de control

El plano de control de Kubernetes es un conjunto de componentes que administran los clústeres y estos componentes envían distintos tipos de información en forma de flujos de registro a un grupo de registros en Amazon. CloudWatch Si bien habilitar todos los tipos de registros del plano de control tiene sus ventajas, debe tener en cuenta la información de cada registro y los costos asociados al almacenamiento de toda la telemetría del registro. Se le cobrarán los costes estándar CloudWatch de ingesta y almacenamiento de datos de Logs para los registros enviados a Amazon CloudWatch Logs desde sus clústeres. Antes de habilitarlos, evalúe si cada flujo de registros es necesario.

Por ejemplo, en los clústeres que no son de producción, habilite de forma selectiva tipos de registro específicos, como los registros del servidor api, solo para su análisis y desactívelos después. Sin embargo, en el caso de los clústeres de producción, en los que es posible que no puedas reproducir

eventos y para resolver problemas se requiera más información de registro, puedes habilitar todos los tipos de registro. En esta entrada de blog encontrará más detalles sobre la implementación de la optimización de costos del plano de control.

Transmita los registros a S3

Otra práctica recomendada para la optimización de costes es transmitir los registros del plano de control a S3 mediante suscripciones a CloudWatch Logs. Aprovechar las suscripciones a CloudWatch Logs le permite reenviar los registros de forma selectiva a S3, lo que proporciona un almacenamiento a largo plazo más rentable en comparación con conservar los registros indefinidamente. CloudWatch Por ejemplo, en el caso de los clústeres de producción, puede crear un grupo de registros importante y aprovechar las suscripciones para transmitir estos registros a S3 después de 15 días. Esto garantizará que tenga acceso rápido a los registros para su análisis, pero también ahorrará costos al mover los registros a un almacenamiento más rentable.

> **Important**
>
> A partir del 5 de septiembre de 2023, los registros de EKS se clasifican como registros vendidos en Amazon Logs. CloudWatch Los registros vendidos son registros de servicios de AWS específicos que los servicios de AWS publican de forma nativa en nombre del cliente y están disponibles a precios de descuento por volumen. Visita la página de CloudWatch precios de Amazon para obtener más información sobre los precios de Vended Logs.

Plano de datos EKS

Retención de registros

La política CloudWatch de retención predeterminada de Amazon consiste en conservar los registros indefinidamente y no caducar nunca, lo que implica costes de almacenamiento aplicables a su región de AWS. Para reducir los costes de almacenamiento, puede personalizar la política de retención para cada grupo de registros en función de sus requisitos de carga de trabajo.

En un entorno de desarrollo, puede que no sea necesario un período de retención prolongado. Sin embargo, en un entorno de producción, puede establecer una política de retención más prolongada para cumplir con los requisitos de resolución de problemas, conformidad y planificación de la capacidad. Por ejemplo, si ejecuta una aplicación de comercio electrónico durante la temporada alta de fiestas, el sistema está sobrecargado y pueden surgir problemas que no se noten de inmediato,

querrá configurar una retención de registros más prolongada para solucionar problemas detallados y realizar un análisis posterior al evento.

Puede configurar sus períodos de retención en la CloudWatch consola de AWS o en la API de AWS con una duración de 1 día a 10 años en función de cada grupo de registros. Tener un período de retención flexible puede ahorrar costos de almacenamiento de registros y, al mismo tiempo, mantener los registros críticos.

Opciones de almacenamiento de registros

El almacenamiento es un factor importante de los costos de observabilidad, por lo que es crucial optimizar la estrategia de almacenamiento de registros. Sus estrategias deben ajustarse a los requisitos de sus cargas de trabajo y, al mismo tiempo, mantener el rendimiento y la escalabilidad. Una estrategia para reducir los costes de almacenamiento de registros consiste en aprovechar los buckets S3 de AWS y sus diferentes niveles de almacenamiento.

Reenvíe los registros directamente a S3

Considere la posibilidad de reenviar los registros menos críticos, como los entornos de desarrollo, directamente a S3 en lugar de a Cloudwatch. Esto puede tener un impacto inmediato en los costes de almacenamiento de registros. Una opción es reenviar los registros directamente a S3 con Fluentbit. Defina esto en la `[OUTPUT]` sección, el destino al que se FluentBit transmiten los registros del contenedor para su retención. Revise los parámetros de configuración adicionales aquí.

```
[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
```

Reenvíe los registros a CloudWatch solo para su análisis a corto plazo

Para los registros más críticos, como los entornos de producción en los que podría necesitar realizar un análisis inmediato de los datos, considere la posibilidad de reenviar los registros a CloudWatch. Defina esto en la `[OUTPUT]` sección, el destino al que se FluentBit transmiten los registros del contenedor para su conservación. Revise los parámetros de configuración adicionales aquí.

```
[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
```

Sin embargo, esto no tendrá un efecto inmediato en sus ahorros de costos. Para ahorrar más, tendrá que exportar estos registros a Amazon S3.

Exportación a Amazon S3 desde CloudWatch

Para almacenar CloudWatch los registros de Amazon a largo plazo, recomendamos exportar CloudWatch los registros de Amazon EKS a Amazon Simple Storage Service (Amazon S3). Puede reenviar los registros al bucket de Amazon S3 creando una tarea de exportación a través de la consola o la API. Una vez hecho esto, Amazon S3 presenta muchas opciones para reducir aún más los costos. Puede definir sus propias reglas del ciclo de vida de Amazon S3 para mover sus registros a una clase de almacenamiento que se adapte a sus necesidades, o aprovechar la clase de almacenamiento por niveles inteligentes de Amazon S3 para que AWS mueva automáticamente los datos al almacenamiento a largo plazo en función de su patrón de uso. Consulte este blog para obtener más información. Por ejemplo, en el caso de su entorno de producción, los registros CloudWatch permanecen más de 30 días y luego se exportan al bucket de Amazon S3. A continuación, puede utilizar Amazon Athena para consultar los datos del bucket de Amazon S3 si necesita volver a consultar los registros más adelante.

Reduzca los niveles de registro

Practique el registro selectivo para su aplicación. Tanto las aplicaciones como los nodos generan registros de forma predeterminada. Para los registros de sus aplicaciones, ajuste los niveles de registro para adaptarlos a la importancia de la carga de trabajo y el entorno. Por ejemplo, la siguiente aplicación java genera `INFO` registros, que es la configuración de aplicación predeterminada típica y, según el código, puede generar un gran volumen de datos de registro.

```
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!");    } }
```

En un entorno de desarrollo, cambia el nivel de registro a`DEBUG`, ya que esto puede ayudarte a depurar problemas o a atrapar los posibles antes de que entren en producción.

```
      log.setLevel(Level.DEBUG);
```

En un entorno de producción, considere la posibilidad de modificar el nivel de registro a `ERROR` o`FATAL`. Esto generará un registro solo cuando la aplicación tenga errores, lo que reducirá la salida del registro y le ayudará a centrarse en los datos importantes sobre el estado de la solicitud.

```
      log.setLevel(Level.ERROR);
```

Puedes ajustar con precisión los niveles de registro de varios componentes de Kubernetes. Por ejemplo, si utilizas Bottlerocket como sistema operativo de EKS Node, hay ajustes de configuración que te permiten ajustar el nivel de registro de procesos de Kubelet. A continuación se muestra un fragmento de este ajuste de configuración. Tenga en cuenta que el nivel de registro predeterminado es 2, que ajusta la verbosidad del registro del proceso. `kubelet`

```
[settings.kubernetes]
log-level = "2"
image-gc-high-threshold-percent = "85"
image-gc-low-threshold-percent = "80"
```

En un entorno de desarrollo, puede establecer un nivel de registro superior a 2 para ver eventos adicionales, lo que resulta útil para la depuración. En un entorno de producción, puede establecer el nivel en 0 para ver solo los eventos críticos.

Aproveche los filtros

Si utiliza una configuración predeterminada de EKS Fluentbit para enviar los registros de contenedores a Cloudwatch, FluentBit captura y envía TODOS los registros de contenedores de aplicaciones enriquecidos con metadatos de Kubernetes a Cloudwatch, tal y como se muestra en el bloque de configuración siguiente. `[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}
```

En la `[INPUT]` sección anterior se recopilan todos los registros del contenedor. Esto puede generar una gran cantidad de datos que podrían no ser necesarios. Filtrar estos datos puede reducir la cantidad de datos de registro enviados y, CloudWatch por lo tanto, reducir los costos. Puede aplicar un filtro a sus registros antes de que se envíen a CloudWatch. Fluentbit define esto en la `[FILTER]` sección. Por ejemplo, filtrar los metadatos de Kubernetes para que no se adjunten a los eventos de registro puede reducir el volumen de registro.

```
    [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.
```

Métricas

Las métricas proporcionan información valiosa sobre el rendimiento del sistema. Al consolidar todas las métricas de recursos disponibles o relacionadas con el sistema en una ubicación centralizada, podrá comparar y analizar los datos de rendimiento. Este enfoque centralizado le permite tomar decisiones estratégicas más informadas, como ampliar o reducir los recursos. Además, las métricas desempeñan un papel crucial a la hora de evaluar el estado de los recursos, ya que permiten tomar medidas proactivas cuando es necesario. Por lo general, los costos de observabilidad aumentan con la recopilación y retención de datos de telemetría. A continuación, se muestran algunas estrategias que puede implementar para reducir el costo de la telemetría métrica: recopilar solo las métricas que importan, reducir la cardinalidad de los datos de telemetría y ajustar la granularidad de la recopilación de datos de telemetría.

Supervise lo que importa y recopile solo lo que necesita

La primera estrategia de reducción de costos consiste en reducir la cantidad de métricas que se recopilan y, a su vez, reducir los costos de retención.

1. Comience por analizar sus requisitos o los de las partes interesadas para determinar las métricas más importantes. ¡Las métricas de éxito son diferentes para cada persona! Sepa qué se ve bien y mídalo.
2. Considere analizar en profundidad las cargas de trabajo a las que está dando soporte e identificar sus indicadores clave de rendimiento (KPIs), también conocidos como «señales de oro». Estos deben ajustarse a los requisitos empresariales y de las partes interesadas. Calcular SLIs y SLAs utilizar Amazon CloudWatch y Metric Math es crucial para gestionar la fiabilidad del servicio. SLOs Siga las mejores prácticas descritas en esta guía para supervisar y mantener de forma eficaz el rendimiento de su entorno EKS.
3. Luego, continúe recorriendo los diferentes niveles de infraestructura para conectar y correlacionar las métricas de clúster, nodo e infraestructura adicional de EKS con su carga KPIs de trabajo. Guarde sus métricas empresariales y operativas en un sistema donde pueda correlacionarlas y sacar conclusiones basadas en los impactos observados en ambas.
4. EKS expone las métricas del plano de control, el clúster kube-state-metrics, los módulos y los nodos. La relevancia de todas estas métricas depende de sus necesidades, sin embargo, es

probable que no necesite todas las métricas de los distintos niveles. Puede utilizar esta guía de métricas esenciales de EKS como referencia para supervisar el estado general de un clúster de EKS y sus cargas de trabajo.

A continuación se muestra un ejemplo de la configuración Scrape de Prometheus, en la que utilizamos únicamente las `relabel_config` métricas de Kubelet y `metric_relabel_config` descartamos todas las métricas de contenedores.

```
  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
```

Reduzca la cardinalidad cuando proceda

La cardinalidad se refiere a la unicidad de los valores de los datos en combinación con sus dimensiones (por ejemplo, las etiquetas de Prometheus) para un conjunto de métricas específico. Las métricas de alta cardinalidad tienen muchas dimensiones y cada combinación de métricas dimensionales tiene una singularidad mayor. Una mayor cardinalidad se traduce en un mayor tamaño de los datos de telemetría métrica y en mayores necesidades de almacenamiento, lo que aumenta el coste.

En el ejemplo de cardinalidad alta que aparece a continuación, vemos que la métrica Latency tiene Dimensiones, RequestID, CustomeriD y Service, y cada dimensión tiene muchos valores únicos. La cardinalidad es la medida de la combinación del número de valores posibles por dimensión. En

Prometheus, cada conjunto de dimensiones/etiquetas únicas se considera una métrica nueva, por lo que una cardinalidad alta significa más métricas.

En los entornos de EKS con muchas métricas y dimensiones o etiquetas por métrica (clúster, espacio de nombres, servicio, pod, contenedor, etc.), la cardinalidad tiende a aumentar. Para optimizar los costos, considere cuidadosamente la cardinalidad de las métricas que está recopilando. Por ejemplo, si va a agregar una métrica específica para visualizarla a nivel de clúster, puede eliminar las etiquetas adicionales que se encuentran en una capa inferior, como la etiqueta del espacio de nombres.

Para identificar las métricas de alta cardinalidad en Prometheus, puede ejecutar la siguiente consulta de PROMQL para determinar qué objetivos de raspado tienen el mayor número de métricas (cardinalidad):

```
topk_max(5, max_over_time(scrape_samples_scraped[1h]))
```

y la siguiente consulta de PROMQL puede ayudarle a determinar qué objetivos de raspado tienen las tasas más altas de abandono de métricas (cuántas series de métricas nuevas se crearon en un raspado determinado):

```
topk_max(5, max_over_time(scrape_series_added[1h]))
```

Si está utilizando grafana, puede usar la herramienta Mimirtool de Grafana Lab para analizar sus paneles de grafana y las reglas de Prometheus para identificar las métricas de alta cardinalidad no utilizadas. Sigue esta guía sobre cómo usar los `mimirtool analyze prometheus` comandos `mimirtool analyze` y para identificar las métricas activas a las que no se hace referencia en tus paneles.

Tenga en cuenta la granularidad de las métricas

La recopilación de métricas con una granularidad mayor, como cada segundo en lugar de cada minuto, puede tener un gran impacto en la cantidad de telemetría que se recopila y almacena, lo que aumenta el costo. Determine intervalos razonables de recopilación de métricas o de recopilación de métricas que equilibren entre una granularidad suficiente para detectar problemas transitorios y lo suficientemente bajos como para ser rentables. Reduzca la granularidad de las métricas que se utilizan para planificar la capacidad y analizar intervalos de tiempo más amplios.

A continuación, se muestra un fragmento de la configuración predeterminada del recopilador de complementos EKS Distro for Opentelemetry (ADOT) de AWS.

> **Important**
>
> el intervalo global de raspado de Prometheus está establecido en 15 segundos. Este intervalo de raspado se puede aumentar, lo que resulta en una disminución de la cantidad de datos métricos recopilados en Prometheus.

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: my-collector-amp

...

config: |
    extensions:
      sigv4auth:
        region: "+++<YOUR_AWS_REGION>+++" service: "aps"+++</YOUR_AWS_REGION>+++

 receivers:
   #
   # Scrape configuration for the Prometheus Receiver
   # This is the same configuration used when Prometheus is installed using the
 community Helm chart
   #
   prometheus:
     config:
       global:   scrape_interval: 15s
         scrape_timeout: 10s
```

Rastreo

El coste principal asociado al rastreo proviene de la generación de almacenamiento de trazas. Con el rastreo, el objetivo es recopilar datos suficientes para diagnosticar y comprender los aspectos del rendimiento. Sin embargo, dado que los costos de las trazas de rayos X se basan en los datos enviados a X-Ray, borrar las trazas una vez reenviadas no reducirá sus costos. Repasemos las formas de reducir los costes de rastreo y, al mismo tiempo, conservar los datos para que pueda realizar un análisis adecuado.

Aplique las reglas de muestreo

La frecuencia de muestreo de rayos X es conservadora por defecto. Defina reglas de muestreo que le permitan controlar la cantidad de datos que recopila. Esto mejorará la eficiencia del rendimiento y, al mismo tiempo, reducirá los costes. Al reducir la frecuencia de muestreo, puede recopilar las trazas de la solicitud únicamente en función de las necesidades de sus cargas de trabajo y, al mismo tiempo, mantener una estructura de costes más baja.

Por ejemplo, tiene una aplicación Java que desea depurar los rastros de todas las solicitudes de una ruta problemática.

Configure mediante el SDK para cargar las reglas de muestreo desde un documento JSON

```
{
"version": 2,
  "rules": [
    {
"description": "debug-eks",
      "host": "*",
      "http_method": "PUT",
      "url_path": "/history/*",
      "fixed_target": 0,
      "rate": 1,
      "service_type": "debug-eks"
    }
  ],
  "default": {
"fixed_target": 1,
    "rate": 0.1
  }
}
```

A través de la consola

Aplique Tail Sampling con AWS Distro for OpenTelemetry (ADOT)

ADOT Tail Sampling le permite controlar el volumen de trazas ingeridas en el servicio. Sin embargo, Tail Sampling le permite definir las políticas de muestreo una vez que se hayan completado todos los períodos de la solicitud, en lugar de hacerlo al principio. Esto limita aún más la cantidad de datos sin procesar a los que se transfieren y CloudWatch, por lo tanto, reduce los costos.

Por ejemplo, si muestreas el 1% del tráfico a una página de destino y el 10% de las solicitudes a una página de pago, es posible que obtengas 300 seguimientos durante un período de 30 minutos. Con una regla de muestreo gradual de ADOT que filtra errores específicos, podrías quedarte con 200 trazas, lo que reduce el número de trazas almacenadas.

```
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]
```

Aproveche las opciones de almacenamiento de Amazon S3

Debe aprovechar el bucket de AWS S3 y sus diferentes clases de almacenamiento para almacenar las trazas. Exporte las trazas a S3 antes de que venza el período de retención. Utilice las reglas del ciclo de vida de Amazon S3 para mover los datos de rastreo a la clase de almacenamiento que cumpla con sus requisitos.

Por ejemplo, si tiene trazas que tienen 90 días de antigüedad, Amazon S3 Intelligent-Tiering puede mover automáticamente los datos a un almacenamiento prolongado en función de su patrón de uso. Puede utilizar Amazon Athena para consultar los datos en Amazon S3 si necesita volver a consultar las trazas más adelante. Esto puede reducir aún más el costo del rastreo distribuido.

Recursos adicionales:

- Guía de mejores prácticas de observabilidad
- Recopilación de métricas de mejores prácticas

- AWS re:Invent 2022: prácticas recomendadas de observabilidad en Amazon () COP343
- AWS re:Invent 2022 - Observabilidad: prácticas recomendadas para aplicaciones modernas () COP344

Prácticas recomendadas para Windows

Esta guía proporciona consejos sobre la ejecución de contenedores y nodos de Windows.

Temas

- Amazon EKS optimizó la administración de AMI de Windows
- Configurar gMSA para pods y contenedores de Windows
- Los nodos de trabajo de Windows se están endureciendo
- Escaneo de imágenes de contenedores
- Versión y licencia de Windows Server
- Registro
- Supervisión
- Redes de Windows
- Evitar los errores de OOM
- Aplicación de parches a servidores y contenedores de Windows
- Ejecución de cargas de trabajo heterogéneas
- Contextos de seguridad de los pods
- Opciones de almacenamiento persistente
- Fortalecimiento de las imágenes de los contenedores de Windows

Amazon EKS optimizó la administración de AMI de Windows

AMIs Los optimizados para Windows Amazon EKS se basan en Windows Server 2019 y Windows Server 2022. Están configuradas de modo que sirvan de imagen base para los nodos de Amazon EKS. De forma predeterminada, AMIs incluyen los siguientes componentes:

- kubelet
- kube-proxy
- AWS IAM Authenticator para Kubernetes
- csi-proxy
- containerd

Puede recuperar mediante programación el ID de Amazon Machine Image (AMI) para Amazon EKS optimizado AMIs consultando la API de AWS Systems Manager Parameter Store. Este parámetro elimina la necesidad de buscar manualmente la AMI optimizada para Amazon EKS IDs. Para obtener más información sobre la API del almacén de parámetros de Systems Manager, consulte GetParameter. Su cuenta de usuario debe tener el permiso ssm: GetParameter IAM para recuperar los metadatos de la AMI optimizados para Amazon EKS.

El siguiente ejemplo recupera el ID de AMI de la última AMI optimizada para Amazon EKS para Windows Server 2019 LTSC Core. El número de versión que aparece en el nombre de la AMI se refiere a la compilación de Kubernetes correspondiente para la que está preparada.

```
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
```

Ejemplo de código de salida:

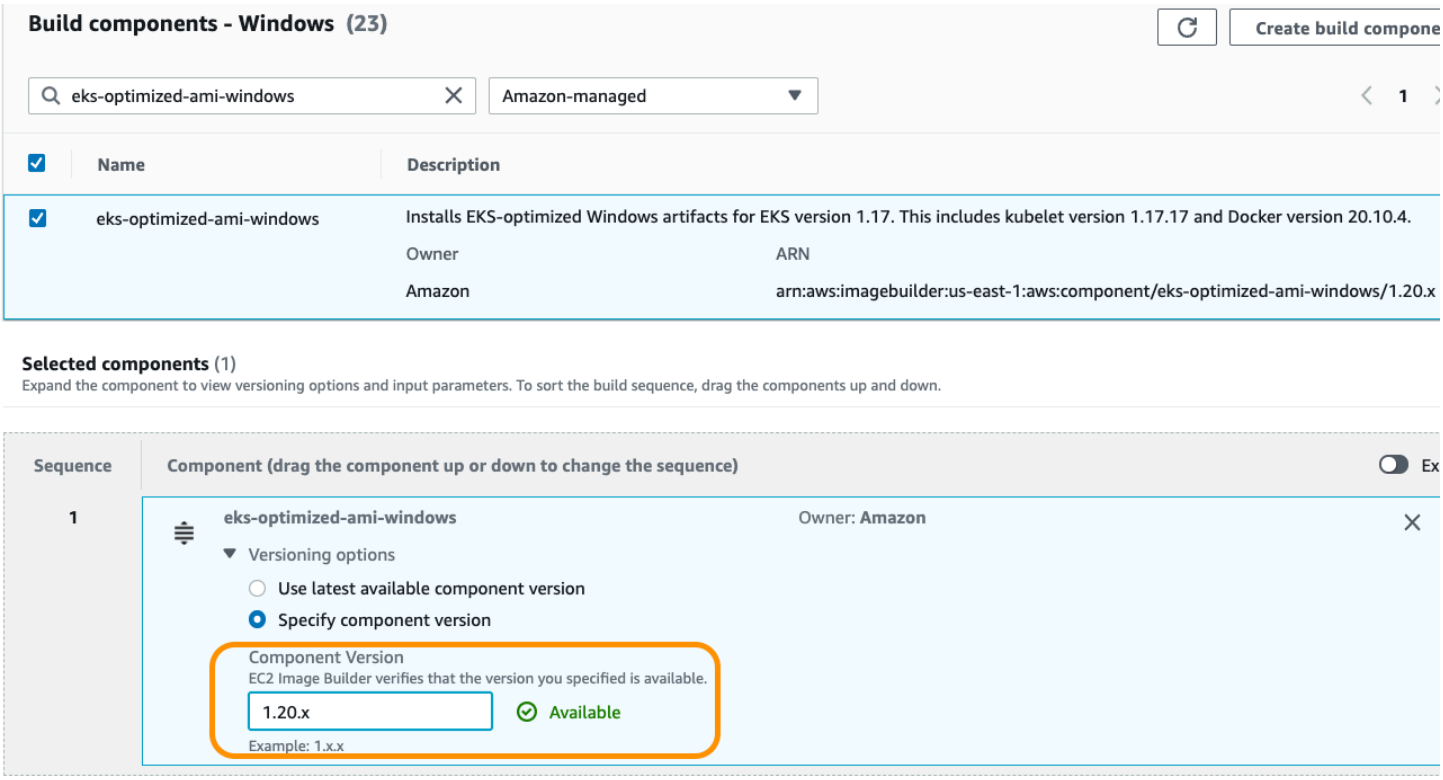
```
ami-09770b3eec4552d4e
```

Administrar su propia AMI de Windows optimizada para Amazon EKS

Un paso esencial hacia los entornos de producción es mantener la misma versión de kubelet y AMI de Windows optimizadas para Amazon EKS en todo el clúster de Amazon EKS.

El uso de la misma versión en todo el clúster de Amazon EKS reduce el tiempo de resolución de problemas y aumenta la coherencia del clúster. Amazon EC2 Image Builder ayuda a crear y mantener Windows optimizados para Amazon EKS personalizados AMIs para su uso en un clúster de Amazon EKS.

Utilice Amazon EC2 Image Builder para seleccionar entre las versiones de Windows Server, las fechas de lanzamiento de la AMI de AWS Windows Server y la versión de compilación del and/ or sistema operativo. El paso de construcción de los componentes le permite seleccionar entre los artefactos de Windows optimizados para EKS existentes y las versiones de kubelet. Para más información: - .html https://docs.aws.amazon.com/eks/ latest/userguide/eks custom-ami-windows



NOTA: Antes de seleccionar una imagen base, consulte la sección Versión y licencia de Windows Server para obtener detalles importantes relacionados con las actualizaciones del canal de lanzamiento.

Configuración de un inicio más rápido para un EKS personalizado y optimizado AMIs

Cuando se utiliza una AMI personalizada optimizada para Amazon EKS para Windows, los nodos de trabajo de Windows se pueden lanzar hasta un 65% más rápido si se habilita la función Fast Launch. Esta función mantiene un conjunto de instantáneas previamente aprovisionadas que ya han completado los pasos de Windows Out of Box Experience (OOBE) especializados en Sysprep y los reinicios necesarios. Estas instantáneas se utilizan luego en lanzamientos posteriores, lo que reduce el tiempo necesario para escalar o reemplazar los nodos. Fast Launch solo se puede habilitar por AMIs su cuenta a través de la EC2 consola o en la AWS CLI, y la cantidad de instantáneas mantenidas es configurable.

NOTA: Fast Launch no es compatible con la AMI optimizada para EKS predeterminada proporcionada por Amazon. Cree una AMI personalizada como se indica arriba antes de intentar habilitarla.

Para obtener más información: AWS Windows AMIs : configure su AMI para un lanzamiento más rápido

Almacenamiento en caché de las capas base de Windows de forma personalizada AMIs

Las imágenes de los contenedores de Windows son más grandes que las de Linux. Si ejecuta una aplicación contenerizada basada en .NET Framework, el tamaño medio de la imagen es de unos 8,24 GB. Durante la programación del pod, se debe extraer completamente la imagen del contenedor y extraerla en el disco antes de que el pod alcance el estado En ejecución.

Durante este proceso, el motor de ejecución del contenedor (containerd) extrae y extrae toda la imagen del contenedor en el disco. La operación de extracción es un proceso paralelo, lo que significa que el tiempo de ejecución del contenedor extrae las capas de imagen del contenedor en paralelo. Por el contrario, la operación de extracción se produce en un proceso secuencial y es I/O intensivo. Por eso, la imagen del contenedor puede tardar más de 8 minutos en extraerse por completo y estar lista para ser utilizada por el motor de ejecución del contenedor (containerd) y, como resultado, el tiempo de inicio del pod puede tardar varios minutos.

Como se menciona en el tema Cómo aplicar parches a servidores y contenedores de Windows, existe la opción de crear una AMI personalizada con EKS. Durante la preparación de la AMI, puede añadir un componente generador de EC2 imágenes adicional para extraer localmente todas las imágenes de contenedores de Windows necesarias y, a continuación, generar la AMI. Esta estrategia reducirá drásticamente el tiempo en que un pod alcanza el estado En ejecución.

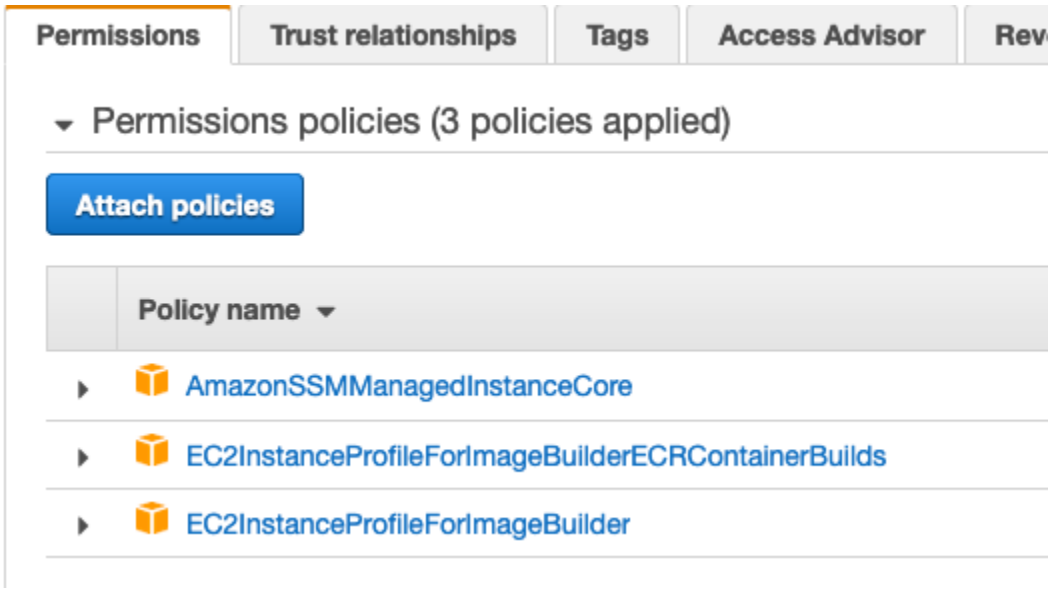
En Amazon EC2 Image Builder, cree un componente para descargar las imágenes necesarias y adjúntelo a la receta de imágenes. El siguiente ejemplo extrae una imagen específica de un repositorio de ECR.

```
name: ContainerdPull
description: This component pulls the necessary containers images for a cache strategy.
schemaVersion: 1.0

phases:
  - name: build
    steps:
      - name: containerdpull
        action: ExecutePowerShell
        inputs:
          commands:
```

```
            - Set-ExecutionPolicy Unrestricted -Force
            - (Get-ECRLoginCommand).Password | docker login --username AWS --password-
stdin 111000111000.dkr.ecr.us-east-1.amazonaws.com
            - ctr image pull mcr.microsoft.com/dotnet/framework/aspnet:latest
            - ctr image pull 111000111000.dkr.ecr.us-east-1.amazonaws.com/
myappcontainerimage:latest
```

Para asegurarse de que el siguiente componente funciona según lo esperado, compruebe si la función de IAM utilizada por EC2 Image builder (EC2InstanceProfileForImageBuilder) tiene las políticas adjuntas:



Publicación de blog.

En la siguiente entrada del blog, encontrará información paso a paso sobre cómo implementar una estrategia de almacenamiento en caché para Amazon EKS Windows AMIs personalizado:

Agilice los tiempos de lanzamiento de los contenedores de Windows con el creador de EC2 imágenes y la estrategia de caché de imágenes

Configurar gMSA para pods y contenedores de Windows

Qué es una cuenta gMSA

Las aplicaciones basadas en Windows, como las aplicaciones.NET, suelen utilizar Active Directory como proveedor de identidad y authorization/authentication utilizan el protocolo NTLM o Kerberos.

Un servidor de aplicaciones para intercambiar tickets de Kerberos con Active Directory debe estar unido a un dominio. Los contenedores de Windows no admiten la unión de dominios y no tendrían mucho sentido, ya que los contenedores son recursos efímeros, lo que supone una carga para el conjunto de RID de Active Directory.

Sin embargo, los administradores pueden aprovechar las cuentas de Active Directory de gMSA para negociar una autenticación de Windows para recursos como contenedores de Windows, NLB y granjas de servidores.

Caso de uso de contenedor de Windows y gMSA

Las aplicaciones que utilizan la autenticación de Windows y se ejecutan como contenedores de Windows se benefician de la gMSA, ya que el nodo de Windows se utiliza para intercambiar el vale de Kerberos en nombre del contenedor. Hay dos opciones disponibles para configurar el nodo de trabajo de Windows para que sea compatible con la integración de gMSA:

1: Nodos de trabajo de Windows unidos a un dominio

En esta configuración, el nodo de trabajo de Windows está unido a un dominio en el dominio de Active Directory y la cuenta de equipo AD de los nodos de trabajo de Windows se utiliza para autenticarse en Active Directory y recuperar la identidad gMSA que se utilizará con el pod.

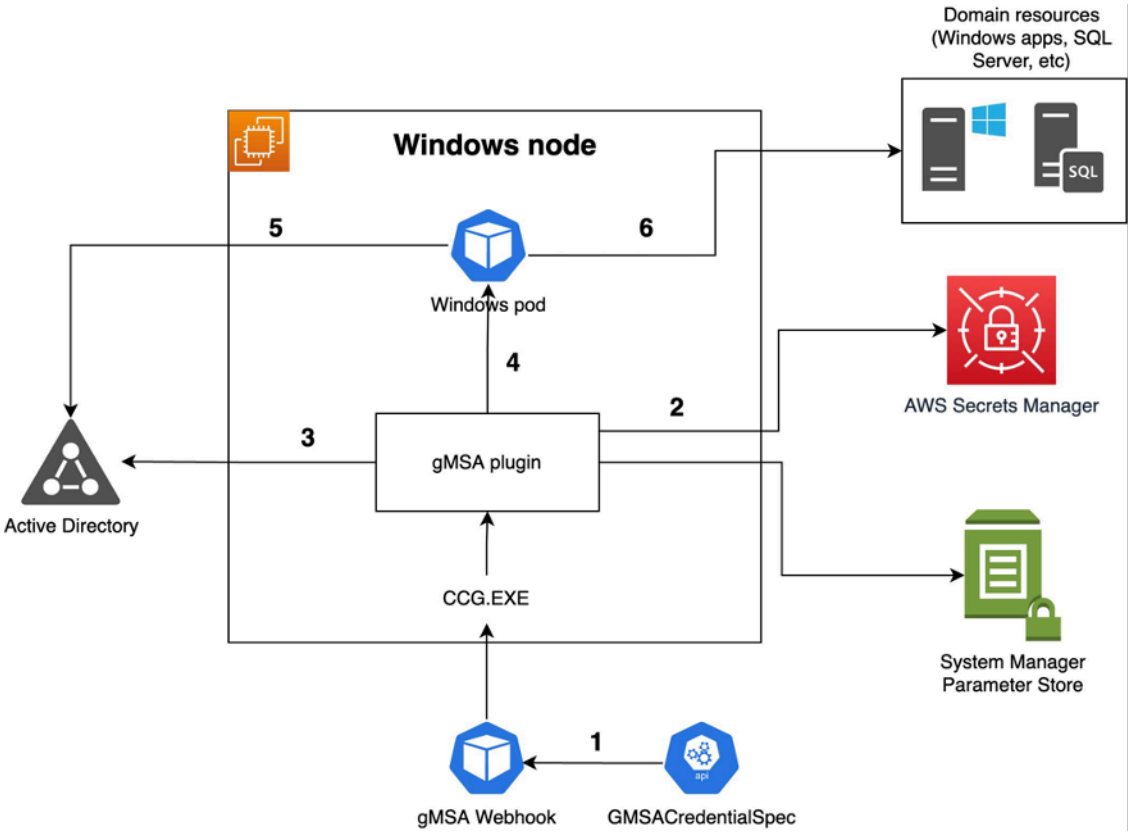
En el enfoque de unión a un dominio, puede administrar y reforzar fácilmente los nodos de trabajo de Windows mediante el Active Directory existente GPOs; sin embargo, esto genera una sobrecarga operativa adicional y demoras cuando los nodos de trabajo de Windows se unen al clúster de Kubernetes, ya que requiere reinicios adicionales durante el inicio del nodo y la limpieza del garaje de Active Directory una vez que el clúster de Kubernetes termina los nodos.

En la siguiente entrada del blog, encontrará información detallada sobre cómo implementar el enfoque de nodos de trabajo de Windows unidos a un dominio: step-by-step

Autenticación de Windows en los pods de Windows de Amazon EKS

2: Nodos de trabajo de Windows sin dominio

En esta configuración, el nodo de trabajo de Windows no está unido al dominio de Active Directory y se utiliza una identidad «portátil» (usuario/contraseña) para autenticarse en Active Directory y recuperar la identidad gMSA que se utilizará con el pod.



La identidad portátil es un usuario de Active Directory; la identidad (usuario/contraseña) se almacena en el almacén de parámetros de AWS Secrets Manager o AWS System Manager, y se utilizará un complemento desarrollado por AWS llamado ccg_plugin para recuperar esta identidad del almacén de parámetros de AWS Secrets Manager o AWS System Manager y pasarla a containerd para recuperar la identidad de gMSA y ponerla a disposición del pod.

Con este enfoque sin dominio, puede beneficiarse de no tener ninguna interacción con Active Directory durante el inicio del nodo de trabajo de Windows al utilizar gMSA y de reducir la sobrecarga operativa para los administradores de Active Directory.

En la siguiente entrada del blog, encontrará información detallada step-by-step sobre cómo implementar el enfoque de nodos de trabajo de Windows sin dominio:

Autenticación de Windows sin dominio para los pods de Windows de Amazon EKS

Notas importantes

A pesar de que el pod puede utilizar una cuenta gMSA, también es necesario configurar la aplicación o el servicio en consecuencia para admitir la autenticación de Windows. Por ejemplo, para configurar Microsoft IIS para que admita la autenticación de Windows, debe prepararlo mediante dockerfile:

```
RUN Install-WindowsFeature -Name Web-Windows-Auth -IncludeAllSubFeature
RUN Import-Module WebAdministration; Set-ItemProperty 'IIS:\AppPools\SiteName' -name
 processModel.identityType -value 2
RUN Import-Module WebAdministration; Set-WebConfigurationProperty -Filter '/
system.webServer/security/authentication/anonymousAuthentication' -Name Enabled -Value
 False -PSPath 'IIS:\' -Location 'SiteName'
RUN Import-Module WebAdministration; Set-WebConfigurationProperty -Filter '/
system.webServer/security/authentication/windowsAuthentication' -Name Enabled -Value
 True -PSPath 'IIS:\' -Location 'SiteName'
```

Los nodos de trabajo de Windows se están endureciendo

El endurecimiento del sistema operativo es una combinación de la configuración del sistema operativo, la aplicación de parches y la eliminación de paquetes de software innecesarios, con el objetivo de bloquear un sistema y reducir la superficie de ataque. Se recomienda preparar su propia AMI de Windows optimizada para EKS con las configuraciones de refuerzo requeridas por su empresa.

Cada mes, AWS proporciona una nueva AMI de Windows optimizada para EKS que contiene los últimos parches de seguridad de Windows Server. Sin embargo, sigue siendo responsabilidad del usuario reforzar su AMI mediante la aplicación de las configuraciones de sistema operativo necesarias, independientemente de si utilizan grupos de nodos autogestionados o gestionados.

Microsoft ofrece una gama de herramientas, como Microsoft Security Compliance Toolkit y Security Baselines, que le ayudan a lograr el refuerzo en función de las necesidades de sus políticas de seguridad. Los puntos de referencia de CIS también están disponibles y deben implementarse sobre una AMI de Windows optimizada para Amazon EKS para entornos de producción.

Reducir la superficie de ataque con Windows Server Core

Windows Server Core es una opción de instalación mínima que está disponible como parte de la AMI de Windows optimizada para EKS. La implementación de Windows Server Core tiene un par de ventajas. En primer lugar, ocupa un espacio de disco relativamente pequeño: 6 GB en Server Core frente a 10 GB en Windows Server con experiencia de escritorio. En segundo lugar, tiene una superficie de ataque más pequeña debido a que su base de código es más pequeña y está disponible. APIs

AWS proporciona a los clientes un nuevo Windows optimizado para Amazon EKS AMIs todos los meses, que contiene los últimos parches de seguridad de Microsoft, independientemente de

la versión compatible con Amazon EKS. Como práctica recomendada, los nodos de trabajo de Windows deben sustituirse por otros nuevos basados en la última AMI optimizada para Amazon EKS. Los nodos que se ejecuten durante más de 45 días sin una actualización o un reemplazo de nodos carecen de las mejores prácticas de seguridad.

Evitar las conexiones RDP

El Protocolo de escritorio remoto (RDP) es un protocolo de conexión desarrollado por Microsoft para proporcionar a los usuarios una interfaz gráfica para conectarse a otro equipo Windows a través de una red.

Como práctica recomendada, debe tratar los nodos de trabajo de Windows como si fueran hosts efímeros. Eso significa que no hay conexiones de administración, actualizaciones ni solución de problemas. Cualquier modificación y actualización debe implementarse como una nueva AMI personalizada y sustituirse por la actualización de un grupo de Auto Scaling. Consulte Aplicación de parches a servidores y contenedores de Windows y Administración optimizada de AMI de Windows por Amazon EKS.

Inhabilite las conexiones RDP en los nodos de Windows durante la implementación pasando el valor false a la propiedad ssh, como se muestra en el siguiente ejemplo:

```
nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
```

Si necesita acceder al nodo de Windows, utilice el administrador de sesiones de AWS System Manager para establecer una PowerShell sesión segura a través de la consola de AWS y el agente SSM. Para ver cómo implementar la solución, consulte Acceda de forma segura a las instancias de Windows mediante el administrador de sesiones de AWS Systems Manager.

Para utilizar el administrador de sesiones de System Manager, se debe aplicar una política de IAM adicional a la función de IAM utilizada para lanzar el nodo de trabajo de Windows. A continuación, se muestra un ejemplo en el SSMManaged InstanceCore que se especifica Amazon en el manifiesto del `eksctl` clúster:

```
 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
```

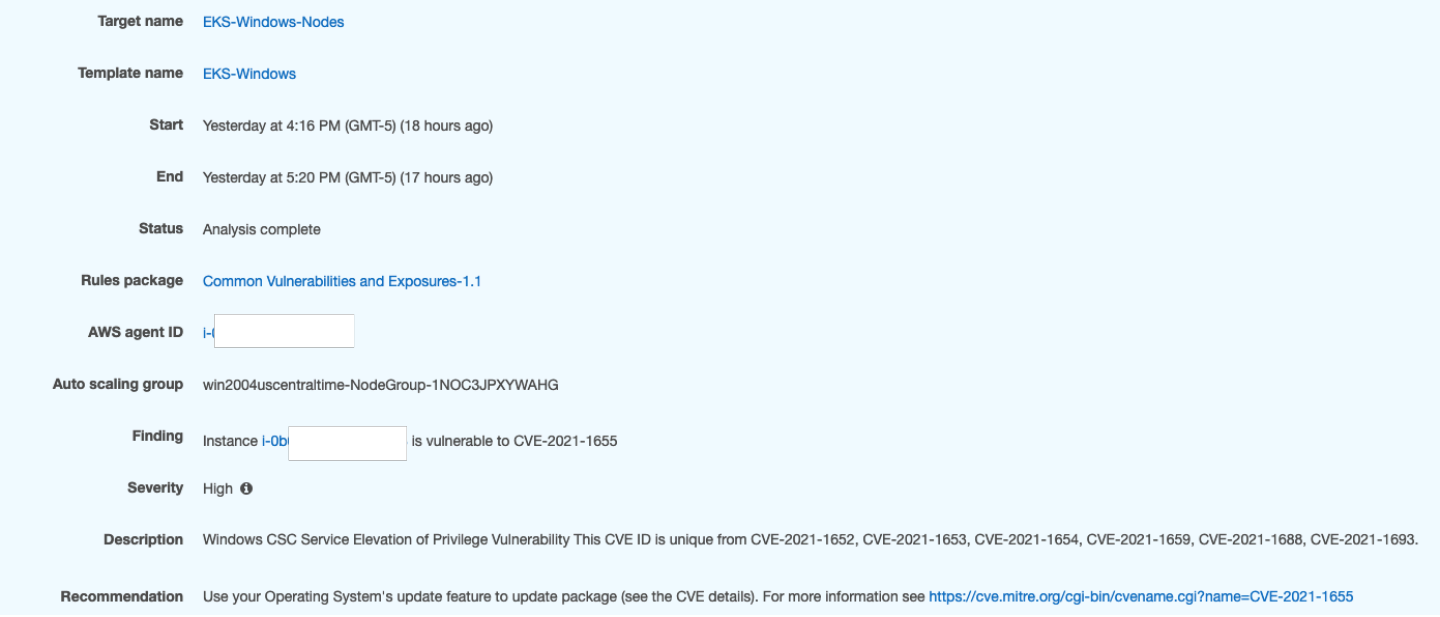
Amazon Inspector

> Amazon Inspector es un servicio de evaluación de seguridad automatizado que ayuda a mejorar la seguridad y la conformidad de las aplicaciones implementadas en AWS. Amazon Inspector evalúa automáticamente las aplicaciones para detectar la exposición, las vulnerabilidades y las desviaciones de las mejores prácticas. Después de la evaluación, Amazon Inspector genera una lista detallada de resultados relacionados con la seguridad priorizados por nivel de gravedad. Estos resultados se pueden revisar directamente o como parte de informes de evaluación detallados que están disponibles a través de la consola o la API de Amazon Inspector.

Amazon Inspector se puede utilizar para ejecutar la evaluación de CIS Benchmark en el nodo de trabajo de Windows y se puede instalar en un Windows Server Core realizando las siguientes tareas:

1. Descargue el siguiente archivo.exe: https://inspector-agent.amazonaws.com/windows/ installer/latest/AWSAgentInstall .exe
2. Transfiera el agente al nodo de trabajo de Windows.
3. Ejecute el siguiente comando PowerShell para instalar el agente de Amazon Inspector: `. \AWSAgentInstall.exe /install`

A continuación se muestra el resultado después de la primera ejecución. Como puede ver, generó resultados basados en la base de datos del CVE. Puede usarlo para reforzar sus nodos de trabajo o crear una AMI basada en las configuraciones reforzadas.



Para obtener más información sobre Amazon Inspector, incluido cómo instalar los agentes de Amazon Inspector, configurar la evaluación comparativa del CIS y generar informes, vea el vídeo Cómo mejorar la seguridad y la conformidad de las cargas de trabajo de Windows con Amazon Inspector.

Amazon GuardDuty

> Amazon GuardDuty es un servicio de detección de amenazas que monitorea continuamente la actividad maliciosa y el comportamiento no autorizado para proteger sus cuentas, cargas de trabajo y datos de AWS almacenados en Amazon S3. Con la nube, se simplifica la recopilación y agregación de las actividades de cuentas y redes, pero los equipos de seguridad pueden tardar mucho tiempo en analizar continuamente los datos del registro de eventos para detectar posibles amenazas.

Al utilizar Amazon, GuardDuty tiene visibilidad de las actividades maliciosas contra los nodos de trabajo de Windows, como los ataques de fuerza bruta RDP y Port Probe.

Vea el GuardDuty vídeo Detección de amenazas para cargas de trabajo de Windows mediante Amazon para aprender a implementar y ejecutar los puntos de referencia de CIS en una AMI de Windows optimizada para EKS

Seguridad en Amazon EC2 para Windows

Lea las prácticas recomendadas de seguridad para las instancias de Amazon EC2 Windows para implementar controles de seguridad en todos los niveles.

Escaneo de imágenes de contenedores

El escaneo de imágenes es una función de evaluación automática de vulnerabilidades que ayuda a mejorar la seguridad de las imágenes del contenedor de su aplicación al escanearlas para detectar una amplia gama de vulnerabilidades del sistema operativo.

Actualmente, Amazon Elastic Container Registry (ECR) solo puede escanear la imagen del contenedor de Linux en busca de vulnerabilidades. Sin embargo, hay herramientas de terceros que se pueden integrar en una CI/CD canalización existente para el escaneo de imágenes de contenedores de Windows.

- Anchore
- PaloAlto Nube de prisma
- Trend Micro: verificación inteligente de seguridad profunda

Para obtener más información sobre cómo integrar estas soluciones con Amazon Elastic Container Repository (ECR), consulte:

- Anchore, digitalización de imágenes en Amazon Elastic Container Registry (ECR)
- PaloAlto, escaneando imágenes en Amazon Elastic Container Registry (ECR)
- TrendMicro, escaneando imágenes en Amazon Elastic Container Registry (ECR)

Versión y licencia de Windows Server

Versión de Windows Server

Una AMI de Windows optimizada para Amazon EKS se basa en las ediciones Datacenter de Windows Server 2019 y 2022 del Long-Term Servicing Channel (LTSC). La versión Datacenter no limita la cantidad de contenedores que se ejecutan en un nodo de trabajo. Para obtener más información: https://docs.microsoft.com/en-us/ virtualization/windowscontainers/about/faq

Canal de servicio a largo plazo (LTSC)

Anteriormente denominado «rama de servicios a largo plazo», este es el modelo de lanzamiento con el que ya está familiarizado, en el que se publica una nueva versión principal de Windows Server cada 2 o 3 años. Los usuarios tienen derecho a 5 años de soporte estándar y 5 años de soporte extendido.

Licencias

Al lanzar una EC2 instancia de Amazon con una AMI basada en Windows Server, Amazon se hace cargo de los costes de licencia y del cumplimiento de la licencia.

Registro

Las aplicaciones en contenedores suelen dirigir los registros de las aplicaciones a STDOUT. El motor de ejecución del contenedor captura estos registros y hace algo con ellos; por lo general, graba en un archivo. El lugar donde se almacenan estos archivos depende del tiempo de ejecución y la configuración del contenedor.

Una diferencia fundamental con los pods de Windows es que no generan STDOUT. Puede ejecutar LogMonitorpara recuperar el ETW (seguimiento de eventos para Windows), los registros de eventos de Windows y otros registros específicos de aplicaciones desde contenedores de Windows en ejecución y canalizar la salida de registro formateada a STDOUT. Luego, estos registros se pueden transmitir mediante fluent-bit o fluentd al destino que desee, como Amazon. CloudWatch

El mecanismo de recopilación de registros recupera STDOUT/STDERR los registros de los pods de Kubernetes. A DaemonSetes una forma común de recopilar registros de los contenedores. Permite gestionar los registros de routing/filtering/enrichment forma independiente de la aplicación. Se DaemonSet puede usar un fluentd para transmitir estos registros y cualquier otro registro generado por la aplicación al agregador de registros deseado.

Aquí se explica información más detallada sobre la transmisión de registros desde las cargas de trabajo de Windows CloudWatch

Recomendaciones de registro

Las prácticas recomendadas generales de registro no son diferentes cuando se utilizan cargas de trabajo de Windows en Kubernetes.

- Registre siempre las entradas de registro estructuradas (JSON/SYSLOG), lo que facilita la gestión de las entradas de registro, ya que hay muchos analizadores preescritos para dichos formatos estructurados.
- Centralice los registros: los contenedores de registro dedicados se pueden utilizar específicamente para recopilar y reenviar los mensajes de registro de todos los contenedores a un destino

- Mantenga baja la verbosidad de los registros, excepto cuando esté depurando. La verbosidad ejerce mucha presión sobre la infraestructura maderera y los eventos importantes pueden perderse en el ruido.
- Registre siempre la información de la solicitud junto con el identificador de la transacción/solicitud para garantizar la trazabilidad. Los objetos de Kubernetes no llevan el nombre de la aplicación, por lo que, por ejemplo, el nombre de un pod `windows-twryrqyw` puede no tener ningún significado al depurar los registros. Esto contribuye a la trazabilidad y a la solución de problemas de las aplicaciones con los registros agregados.

  La forma de generar estos transaction/correlation identificadores depende de la estructura de programación. Sin embargo, un patrón muy común es usar un Aspect/Interceptor de registro, que puede usar el MDC (contexto de diagnóstico mapeado) para inyectar un transaction/correlation identificador único a cada solicitud entrante, de la siguiente manera:

```
import org.slf4j.MDC;
import java.util.UUID;
Class LoggingAspect { //interceptor

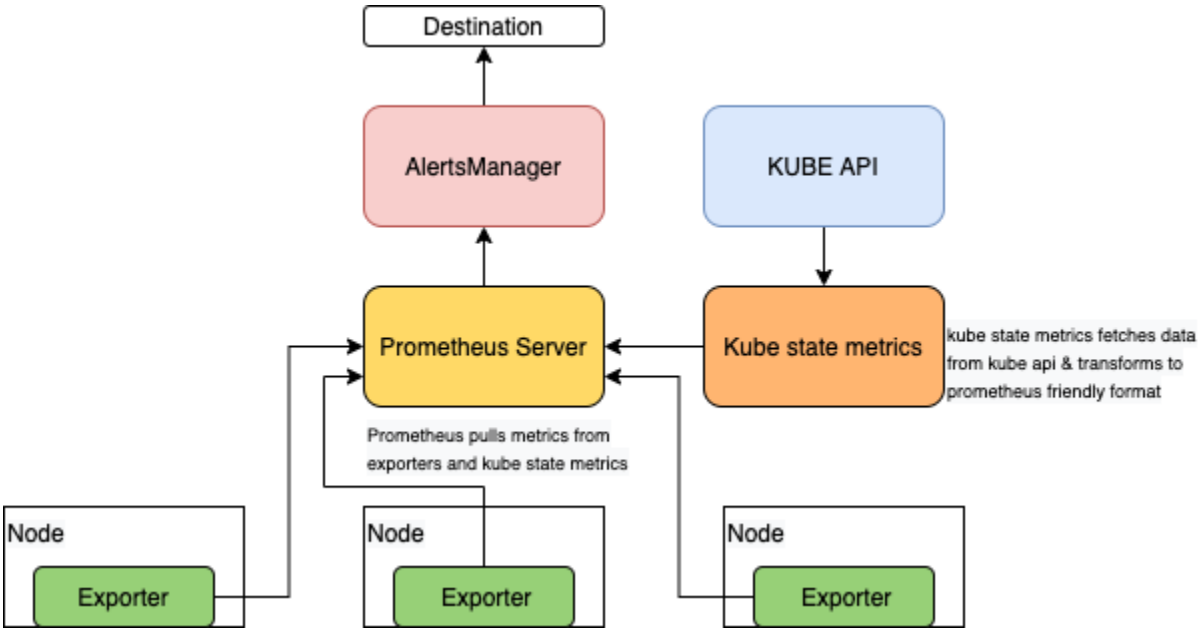
    @Before(value = "execution(* *.*(..))")
    func before(...) {
        transactionId = generateTransactionId();
        MDC.put(CORRELATION_ID, transactionId);
    }

    func generateTransactionId() {
        return UUID.randomUUID().toString();
    }
}
```

Supervisión

Prometheus, un proyecto de CNCF graduado, es, con diferencia, el sistema de monitoreo más popular con integración nativa en Kubernetes. Prometheus recopila métricas en torno a contenedores, pods, nodos y clústeres. Además, Prometheus lo AlertsManager aprovecha, lo que le permite programar alertas para advertirle si algo en su clúster va mal. Prometheus almacena los datos métricos como datos de series temporales identificados por el nombre y los pares de la métrica. key/value Prometheus incluye una forma de realizar consultas mediante un lenguaje llamado ProMQL, que es la abreviatura de Prometheus Query Language.

La arquitectura de alto nivel de la colección de métricas de Prometheus se muestra a continuación:



Prometheus utiliza un mecanismo de extracción y extrae las métricas de los objetivos mediante los exportadores y de la API de Kubernetes mediante las métricas del estado de kube. Esto significa que las aplicaciones y los servicios deben exponer un punto final HTTP (S) que contenga métricas con formato Prometheus. Prometheus, según su configuración, extraerá periódicamente métricas de estos puntos finales HTTP (S).

Un exportador te permite consumir métricas de terceros como métricas con formato de Prometheus. Por lo general, se despliega un exportador de Prometheus en cada nodo. Para obtener una lista completa de los exportadores, consulte los exportadores de Prometheus. Si bien el exportador de nodos es adecuado para exportar el hardware del host y las métricas del sistema operativo para los nodos de Linux, no funcionará para los nodos de Windows.

En un clúster EKS de nodos mixtos con nodos de Windows, cuando utilice el gráfico de mando estable de Prometheus, verá pods defectuosos en los nodos de Windows, ya que este exportador no está diseñado para Windows. Deberá tratar el grupo de trabajadores de Windows por separado y, en su lugar, instalar el exportador de Windows en el grupo de nodos de trabajo de Windows.

Para configurar la supervisión de Prometheus para los nodos de Windows, debe descargar e instalar el exportador WMI en el propio servidor de Windows y, a continuación, configurar los objetivos dentro de la configuración básica del archivo de configuración de Prometheus. La página de versiones incluye todos los instaladores de .msi disponibles, con los respectivos conjuntos de funciones y correcciones de errores. El instalador configurará el windows_exporter como un servicio de Windows y creará una excepción en el firewall de Windows. Si el instalador se ejecuta sin ningún parámetro,

el exportador se ejecutará con la configuración predeterminada para los colectores, puertos, etc. habilitados.

Puede consultar la sección de prácticas recomendadas de programación de esta guía, en la que se sugiere utilizar taints/tolerations o RuntimeClass implementar selectivamente el exportador de nodos únicamente en los nodos de Linux, mientras que el exportador de Windows se instala en los nodos de Windows al arrancar el nodo o mediante la herramienta de administración de la configuración que prefiera (por ejemplo, chef, Ansible, SSM, etc.).

Tenga en cuenta que, a diferencia de los nodos de Linux en los que el exportador de nodos está instalado como un daemonset, en los nodos de Windows el exportador de WMI se instala en el propio host. El exportador exportará métricas como el uso de la CPU, la memoria y el uso del disco I/O , y también se puede utilizar para supervisar los sitios y las aplicaciones de IIS, las interfaces de red y los servicios.

De forma predeterminada, el windows_exporter mostrará todas las métricas de los recopiladores habilitados. Esta es la forma recomendada de recopilar métricas para evitar errores. Sin embargo, para un uso avanzado, se puede pasar al windows_exporter una lista opcional de recopiladores para filtrar las métricas. El parámetro collect [] de la configuración de Prometheus le permite hacerlo.

Los pasos de instalación predeterminados para Windows incluyen descargar e iniciar el exportador como un servicio durante el proceso de arranque con argumentos, como los recopiladores que desea filtrar.

```
> 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"
```

De forma predeterminada, las métricas se pueden almacenar en el punto final /metrics del puerto 9182. En este punto, Prometheus puede consumir las métricas añadiendo el siguiente scrape_config a la configuración de Prometheus

```
scrape_configs:
    - job_name: "prometheus"
      static_configs:
        - targets: ['localhost:9090']
    ...
    - job_name: "wmi_exporter"
```

```
      scrape_interval: 10s
      static_configs:
        - targets: ['<windows-node1-ip>:9182', '<windows-node2-ip>:9182', ...]
```

La configuración de Prometheus se recarga mediante

```
> ps aux | grep prometheus
> kill HUP <PID>
```

Una forma mejor y recomendada de añadir objetivos es utilizar una definición de recurso personalizada llamada ServiceMonitor [que viene como parte del operador Prometheus] que proporciona la definición de ServiceMonitor un objeto y un controlador que activará la configuración de Prometheus que definimos y construirá automáticamente ServiceMonitors la configuración de Prometheus requerida.

La ServiceMonitor, que especifica de forma declarativa cómo deben supervisarse los grupos de servicios de Kubernetes, se utiliza para definir una aplicación de la que se quieren extraer métricas de Kubernetes. En ellas ServiceMonitor especificamos las etiquetas de Kubernetes que el operador puede usar para identificar el servicio de Kubernetes, lo que a su vez identifica los pods que queremos monitorear.

Para aprovecharlos ServiceMonitor, cree un objeto Endpoint que apunte a objetivos específicos de Windows, un servicio headless y un objeto para los nodos de Windows. ServiceMontor

```
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:
```

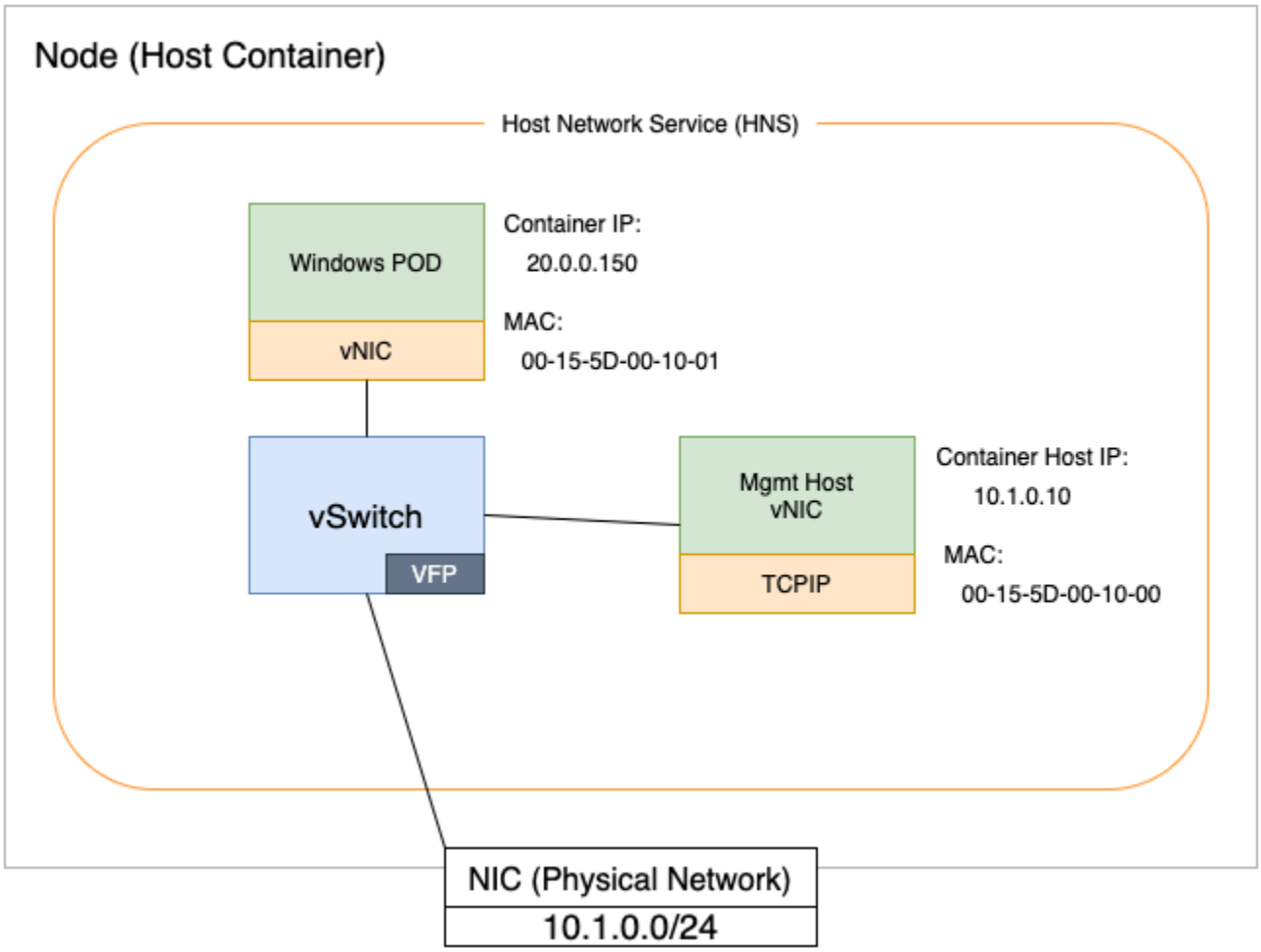
```
    matchLabels:
      k8s-app: wmiexporter
```

Para obtener más información sobre el operador y su uso ServiceMonitor, consulte la documentación oficial del operador. Tenga en cuenta que Prometheus admite la detección dinámica de objetivos mediante muchas opciones de descubrimiento de servicios.

Redes de Windows

Descripción general de las redes de contenedores de Windows

Los contenedores de Windows son fundamentalmente diferentes a los contenedores de Linux. Los contenedores de Linux utilizan construcciones de Linux como los espacios de nombres, el sistema de archivos de unión y los cgroups. En Windows, esas construcciones se abstraen de las que contiene el Host Compute Service (HCS). El HCS actúa como una capa de API que se encuentra por encima de la implementación del contenedor en Windows. Los contenedores de Windows también utilizan el Host Network Service (HNS), que define la topología de red de un nodo.



Desde el punto de vista de las redes, HCS y HNS hacen que los contenedores de Windows funcionen como máquinas virtuales. Por ejemplo, cada contenedor tiene un adaptador de red virtual (vNIC) que está conectado a un conmutador virtual de Hyper-V (vSwitch), como se muestra en el diagrama anterior.

Administración de direcciones IP

Un nodo de Amazon EKS utiliza su interfaz de red elástica (ENI) para conectarse a una red de VPC de AWS. En la actualidad, solo se admite un ENI por nodo de trabajo de Windows. La administración de direcciones IP para los nodos de Windows la realiza el controlador de recursos de VPC, que se ejecuta en el plano de control. Puede encontrar más detalles sobre el flujo de trabajo para la administración de direcciones IP de los nodos de Windows aquí.

El número de pods que puede admitir un nodo de trabajo de Windows depende del tamaño del nodo y del número de IPv4 direcciones disponibles. Puede calcular la IPv4 dirección disponible en el nodo de la siguiente manera:

- De forma predeterminada, solo se asignan IPv4 direcciones secundarias al ENI. En tal caso:

```
Total IPv4 addresses available for Pods = Number of supported IPv4 addresses in the
 primary interface - 1
```

  Reducimos una del recuento total, ya que una IPv4 dirección se utilizará como dirección principal del ENI y, por lo tanto, no se podrá asignar a los Pods.
- Si el clúster se ha configurado para una alta densidad de pods habilitando la función de delegación de prefijos, entonces-

```
Total IPv4 addresses available for Pods = (Number of supported IPv4 addresses in the
 primary interface - 1) * 16
```

  Aquí, en lugar de asignar IPv4 direcciones secundarias, el controlador de recursos de VPC las `/28` `prefixes` asignará y, por lo tanto, el número total de direcciones IPv4 disponibles aumentará 16 veces.

Con la fórmula anterior, podemos calcular el número máximo de pods para un nodo de trabajo de Windows basado en una instancia m5.large de la siguiente manera:

- De forma predeterminada, cuando se ejecuta en modo IP secundario:

```
10 secondary IPv4 addresses per ENI - 1 = 9 available IPv4 addresses
```

- Cuando se usa `prefix delegation` -

```
(10 secondary IPv4 addresses per ENI - 1) * 16 = 144 available IPv4 addresses
```

Para obtener más información sobre cuántas direcciones IP puede admitir un tipo de instancia, consulta Direcciones IP por interfaz de red por tipo de instancia.

Otra consideración clave es el flujo del tráfico de red. Con Windows, existe el riesgo de que se agoten los puertos en los nodos con más de 100 servicios. Cuando se presente esta condición, los nodos comenzarán a generar errores con el siguiente mensaje:

«Falló la creación de la política: el hcnCreateLoad balanceador falló en Win32: el puerto especificado ya existe».

Para solucionar este problema, utilizamos Direct Server Return (DSR). El DSR es una implementación de la distribución asimétrica de la carga de la red. En otras palabras, el tráfico de solicitud y respuesta utiliza diferentes rutas de red. Esta función acelera la comunicación entre los módulos y reduce el riesgo de agotamiento de los puertos. Por lo tanto, recomendamos habilitar el DSR en los nodos de Windows.

El DSR está habilitado de forma predeterminada en Windows Server SAC EKS Optimized. AMIs En el caso de Windows Server 2019 LTSC EKS Optimized AMIs, tendrá que habilitarlo durante el aprovisionamiento de instancias mediante el siguiente script y utilizando Windows Server 2019 Full o Core como AMIFamily en el NodeGroup. `eksctl` Consulte la AMI personalizada de eksctl para obtener información adicional.

```
nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
```

Para utilizar DSR en Windows Server 2019 y versiones posteriores, tendrás que especificar los siguientes indicadores de kube-proxy durante el inicio de la instancia. Para ello, puede ajustar el script de datos de usuario asociado a la plantilla de lanzamiento del grupo de nodos autogestionado.

```
<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>
```

La habilitación del DSR se puede verificar siguiendo las instrucciones del blog de Microsoft Networking y del laboratorio de Windows Containers on AWS.

```
PS C:\temp> (Get-Content C:\temp\pmsx1eau.0ia\policy.txt | ConvertFrom-JSON).Policies


ExternalPort : 53
InternalPort : 53
IsDSR        : True
Protocol     : 17
Type         : ELB
VIPs         : {172.20.0.10}

ExternalPort : 53
InternalPort : 53
IsDSR        : True
Protocol     : 6
Type         : ELB
VIPs         : {172.20.0.10}

ExternalPort : 443
InternalPort : 443
IsDSR        : True
Protocol     : 6
Type         : ELB
VIPs         : {172.20.0.1}
```

Si conservar IPv4 las direcciones disponibles y minimizar el desperdicio es crucial para su subred, generalmente se recomienda evitar el uso del modo de delegación de prefijos, tal como se menciona en Modo de prefijo para Windows: Cuándo evitar. Si aún desea utilizar la delegación de prefijos, puede tomar medidas para optimizar la utilización de IPv4 direcciones en la subred. Consulte Configuración de los parámetros para la delegación de prefijos para obtener instrucciones detalladas sobre cómo ajustar el proceso de solicitud y asignación de IPv4 direcciones. Ajustar estas configuraciones puede ayudarle a lograr un equilibrio entre la conservación de las IPv4 direcciones y las ventajas de la delegación de prefijos en cuanto a densidad de pods.

Cuando se utiliza la configuración predeterminada de asignar IPv4 direcciones secundarias, actualmente no hay configuraciones compatibles para manipular la forma en que el controlador de recursos de VPC solicita y IPv4 asigna las direcciones. Más específicamente, solo `warm-ip-target` se admiten en `minimum-ip-target` el modo de delegación de prefijos. Tenga en cuenta también que, en el modo IP secundario, según las direcciones IP disponibles en la interfaz, el controlador de recursos de la VPC normalmente asignará 3 IPv4 direcciones no utilizadas en el nodo en su nombre IPs para mantenerlas calientes y acelerar los tiempos de inicio del pod. Si quiere minimizar el despilfarro de IP generado por las direcciones IP cálidas no utilizadas, puede programar más pods en un nodo de Windows determinado, de forma que utilice la mayor capacidad de direcciones IP del ENI como sea posible. IPs De forma más explícita, se podría evitar que el nodo y los módulos en ejecución utilizaran ya todas las direcciones IP del ENI. Otra solución alternativa

que le ayude a resolver las limitaciones de disponibilidad de las direcciones IP en sus subredes podría consistir en aumentar el tamaño de la subred o separar los nodos de Windows en sus propias subredes dedicadas.

Además, es importante tener en cuenta que, por el momento, no IPv6 es compatible con los nodos de Windows.

Opciones de interfaz de red de contenedores (CNI)

El AWSVPC CNI es el complemento CNI de facto para los nodos de trabajo de Windows y Linux. Si bien el AWSVPC CNI satisface las necesidades de muchos clientes, puede que haya ocasiones en las que necesite considerar alternativas como una red superpuesta para evitar el agotamiento de la IP. En estos casos, se puede utilizar el CNI de Calico en lugar del CNI. AWSVPC Project Calico es un software de código abierto desarrollado por Tigera. Ese software incluye un CNI que funciona con EKS. Las instrucciones para instalar Calico CNI en EKS se encuentran en la página de instalación del Proyecto Calico EKS.

Políticas de red

Se considera una buena práctica cambiar del modo predeterminado de comunicación abierta entre los pods del clúster de Kubernetes a limitar el acceso en función de las políticas de red. El proyecto Calico, de código abierto, apoya decididamente las políticas de red que funcionan con nodos de Linux y Windows. Esta función es independiente y no depende del uso del CNI de Calico. Por lo tanto, recomendamos instalar Calico y usarlo para la administración de políticas de red.

Las instrucciones para instalar Calico en EKS se encuentran en la página Instalación de Calico en Amazon EKS.

Además, los consejos proporcionados en la sección Red de la Guía de mejores prácticas de seguridad de Amazon EKS se aplican por igual a los clústeres de EKS con nodos de trabajo de Windows; sin embargo, Windows no admite algunas funciones, como los «Grupos de seguridad para pods», en este momento.

Evitar los errores de OOM

Windows no tiene un asesino de out-of-memory procesos como Linux. Windows siempre trata todas las asignaciones de memoria en modo de usuario como virtuales y los archivos de paginación son obligatorios. El efecto neto es que Windows no se queda sin memoria de la misma manera que Linux. Los procesos se archivarán en el disco en lugar de estar sujetos a una interrupción por falta

de memoria (OOM). Si la memoria está sobreaprovisionada y toda la memoria física está agotada, la paginación puede ralentizar el rendimiento.

Reservar la memoria del sistema y del kubelet

A diferencia de Linux, donde se `--kubelet-reserve` captura la reserva de recursos para los demonios del sistema Kubernetes, como kubelet, el tiempo de ejecución de contenedores, etc., y se `--system-reserve` captura la reserva de recursos para los demonios del sistema operativo, como sshd, udev, etc. En Windows, estos indicadores no capturan ni establecen límites de memoria en el kubelet ni en los procesos que se ejecutan en el nodo.

Sin embargo, puedes combinar estos indicadores NodeAllocatablepara reducir la capacidad del nodo con el límite de recursos de memoria del manifiesto del pod para controlar la asignación de memoria por pod. Con esta estrategia, tendrá un mejor control de la asignación de memoria, así como un mecanismo de minimización out-of-memory (OOM) en los nodos de Windows.

En los nodos de Windows, se recomienda reservar al menos 2 GB de memoria para el sistema operativo y el proceso. Úselo `--kubelet-reserve` and/or `--system-reserve` para reducir NodeAllocatable.

Siguiendo la documentación de los nodos autogestionados de Windows de Amazon EKS, utilice la CloudFormation plantilla para lanzar un nuevo grupo de nodos de Windows con personalizaciones de la configuración de kubelet. CloudFormation Tiene un elemento llamado `BootstrapArguments` que es igual a. `KubeletExtraArgs` Utilícelo con los siguientes indicadores y valores:

```
--kube-reserved memory=0.5Gi,ephemeral-storage=1Gi --system-
reserved memory=1.5Gi,ephemeral-storage=1Gi --eviction-hard
 memory.available<200Mi,nodefs.available<10%"
```

Si eksctl es la herramienta de despliegue, consulta la siguiente documentación para personalizar la configuración de kubelet en https://eksctl. io/usage/customizing-the-kubelet/

Requisitos de memoria para contenedores de Windows

Según la documentación de Microsoft, una imagen base de Windows Server para NANO requiere al menos 30 MB, mientras que Server Core requiere 45 MB. Estas cifras aumentan a medida que se agregan componentes de Windows, como .NET Framework, servicios web como IIS y aplicaciones.

Es fundamental que conozca la cantidad mínima de memoria que necesita la imagen de contenedor de Windows, es decir, la imagen base más sus capas de aplicación, y que la defina como la del

contenedor resources/requests en la especificación del pod. También deberías establecer un límite para evitar que los pods consuman toda la memoria de nodo disponible en caso de que se produzca un problema con la aplicación.

En el siguiente ejemplo, cuando el programador de Kubernetes intenta colocar un pod en un nodo, las solicitudes del pod se utilizan para determinar qué nodo tiene suficientes recursos disponibles para la programación.

```
spec:
 - name: iis
   image: mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019
   resources:
     limits:
       cpu: 1
       memory: 800Mi
     requests:
       cpu: .1
       memory: 128Mi
```

Conclusión

El uso de este enfoque minimiza los riesgos de agotamiento de la memoria, pero no evita que se produzca. Con Amazon CloudWatch Metrics, puede configurar alertas y soluciones en caso de que se agote la memoria.

Aplicación de parches a servidores y contenedores de Windows

La aplicación de parches a Windows Server es una tarea de administración estándar para los administradores de Windows. Esto se puede lograr con diferentes herramientas, como Amazon System Manager: Patch Manager, WSUS, System Center Configuration Manager y muchas otras. Sin embargo, los nodos de Windows de un clúster de Amazon EKS no deben tratarse como servidores Windows normales. Deben tratarse como un servidor inmutable. En pocas palabras, evite actualizar un nodo existente, simplemente lance uno nuevo basado en una AMI nueva y actualizada.

Con EC2 Image Builder puedes automatizar la AMIs construcción, creando recetas y añadiendo componentes.

El siguiente ejemplo muestra los componentes, que pueden ser preexistentes creados por AWS (administrados por Amazon), así como los componentes que usted crea (de mi propiedad). Preste

mucha atención al componente gestionado por Amazon denominado update-windows, que actualiza Windows Server antes de generar la AMI mediante la canalización de Image Builder EC2 .

Associated components

Search components

Component name	Version	Description	Date created	Platform	Owner	ARN
eks-optimized-ami-windows	1.20.0	Installs EKS-optimized Windows artifacts for EKS version 1.20. This includes kubelet version 1.20.4 and Docker version 20.10.6.	Aug 18, 2021 6:33 PM	Windows	Amazon	arn:aws:imagebuilder:us-east-1:aws:component/eks-optimized-ami-windows/1.20.0/1
update-windows	1.0.1	Updates Windows with the latest security updates.	Aug 18, 2021 6:33 PM	Windows	Amazon	arn:aws:imagebuilder:us-east-1:aws:component/update-windows/1.0.1/1
Docker Pull	1.0.1	-	Dec 22, 2021 12:08 PM	Windows	[redacted]	arn:aws:imagebuilder:us-east-[redacted]component/docker-pull/1.0.1/1
simple-boot-test-windows	1.0.0	Executes a simple boot test.	Nov 30, 2019 11:39 PM	Windows	Amazon	arn:aws:imagebuilder:us-east-1:aws:component/simple-boot-test-windows/1.0.0/1

EC2 Image Builder le permite crear AMI basadas en Amazon Managed Public AMIs y personalizarlas para que se ajusten a los requisitos de su empresa. A continuación, puede AMIs asociarlas a plantillas de lanzamiento, lo que le permite vincular una nueva AMI al grupo de Auto Scaling creado por el grupo de nodos EKS. Una vez finalizado, puede empezar a cerrar los nodos de Windows existentes y se lanzarán otros nuevos en función de la nueva AMI actualizada.

Empujar y extraer imágenes de Windows

Amazon publica una versión optimizada para EKS AMIs que incluye dos imágenes de contenedores de Windows en caché.

```
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
```

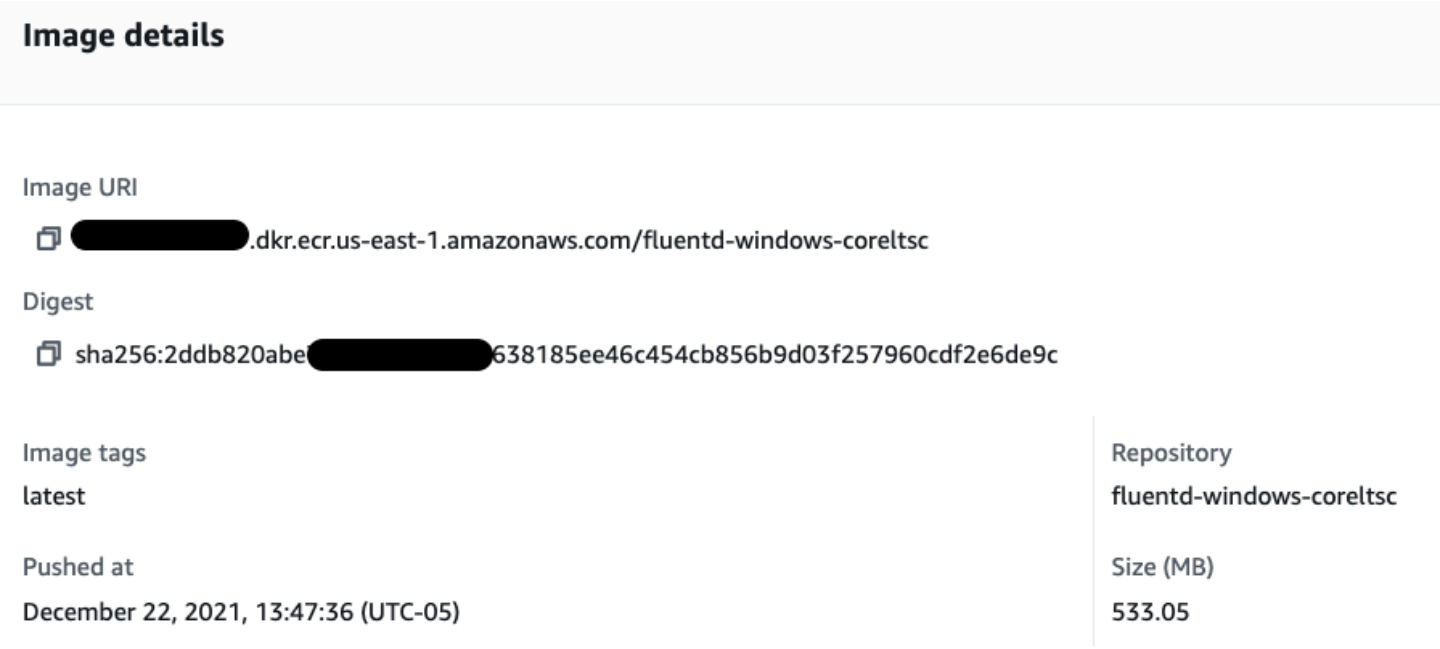
Las imágenes almacenadas en caché se actualizan después de las actualizaciones del sistema operativo principal. Cuando Microsoft publique una nueva actualización de Windows que afecte directamente a la imagen base del contenedor de Windows, la actualización se lanzará como una actualización de Windows normal en el sistema operativo principal. Mantener el entorno up-to-date ofrece un entorno más seguro a nivel de nodo y contenedor.

El tamaño de la imagen de un contenedor de Windows influye en push/pull las operaciones, lo que puede provocar que los tiempos de inicio del contenedor sean lentos. El almacenamiento en caché de las imágenes de contenedores de Windows permite que las costosas I/O operaciones (extracción de archivos) se realicen en la creación de la compilación de la AMI en lugar de en el lanzamiento del contenedor. Como resultado, todas las capas de imágenes necesarias se extraerán

en la AMI y estarán listas para usarse, lo que acelerará el tiempo de lanzamiento de un contenedor de Windows y podrá empezar a aceptar tráfico. Durante una operación de inserción, solo se cargan en el repositorio las capas que componen la imagen.

El siguiente ejemplo muestra que en el Amazon ECR las imágenes de fluentd-windows-sac2004 tienen solo 390,18 MB. Esta es la cantidad de carga que se produjo durante la operación de inserción.

El siguiente ejemplo muestra una imagen ltsc de Windows fluida insertada en un repositorio de Amazon ECR. El tamaño de la capa almacenada en ECR es 533,05 MB.



En el siguiente resultado`docker image ls`, el tamaño del fluentd v1.14-windows-ltsc2019-1 es de 6,96 GB en disco, pero eso no significa que haya descargado y extraído esa cantidad de datos.

En la práctica, durante la operación de extracción solo se descargarán y extraerán los 533,05 MB comprimidos.

```
REPOSITORY                                                                TAG
            IMAGE ID       CREATED         SIZE
111122223333.dkr.ecr.us-east-1.amazonaws.com/fluentd-windows-coreltsc   latest
            721afca2c725   7 weeks ago     6.96GB
fluent/fluentd                                                            v1.14-windows-
ltsc2019-1   721afca2c725   7 weeks ago     6.96GB
amazonaws.com/eks/pause-windows                                           latest
            6392f69ae6e7   10 months ago   255MB
```

La columna de tamaño muestra el tamaño total de la imagen, 6,96 GB. Desglosándolo:

- Imagen básica de Windows Server Core 2019 LTSC = 5,74 GB
- Imagen base sin comprimir de Fluentd = 6,96 GB
- Diferencia en el disco = 1,2 GB
- Imagen final comprimida con Fluentd (ECR) = 533,05 MB

La imagen base ya existe en el disco local, por lo que la cantidad total en el disco es de 1,2 GB adicionales. La próxima vez que vea la cantidad de GBs en la columna de tamaño, no se preocupe demasiado, es probable que más del 70% ya esté en el disco como imagen de contenedor almacenada en caché.

Referencia

Acelera los tiempos de lanzamiento de los contenedores de Windows con el creador de EC2 imágenes y la estrategia de caché de imágenes

Ejecución de cargas de trabajo heterogéneas

Kubernetes admite clústeres heterogéneos, en los que puede tener una combinación de nodos de Linux y Windows en el mismo clúster. Dentro de ese clúster, puedes tener una combinación de pods que se ejecutan en Linux y pods que se ejecutan en Windows. Incluso puedes ejecutar varias versiones de Windows en el mismo clúster. Sin embargo, hay varios factores (como se menciona a continuación) que deberán tenerse en cuenta al tomar esta decisión.

Asignación PODs a nodos: mejores prácticas

Para mantener las cargas de trabajo de Linux y Windows en sus respectivos nodos específicos del sistema operativo, es necesario utilizar alguna combinación de selectores de nodos y de restricciones o tolerancias. El objetivo principal de programar las cargas de trabajo en un entorno heterogéneo es evitar que se rompa la compatibilidad con las cargas de trabajo de Linux existentes.

Garantizar que las cargas de trabajo específicas del sistema operativo lleguen al host contenedor adecuado

Los usuarios pueden asegurarse de que los contenedores de Windows se puedan programar en el host adecuado mediante NodeSelectors. En la actualidad, todos los nodos de Kubernetes tienen las siguientes etiquetas predeterminadas:

```
kubernetes.io/os = [windows|linux]
kubernetes.io/arch = [amd64|arm64|...]
```

Si la especificación de un pod no incluye un NodeSelector`"kubernetes.io/os": windows`, el pod puede programarse en cualquier host, ya sea Windows o Linux. Esto puede ser problemático, ya que un contenedor de Windows solo puede ejecutarse en Windows y un contenedor de Linux solo puede ejecutarse en Linux.

En los entornos empresariales, no es raro tener una gran cantidad de implementaciones preexistentes de contenedores de Linux, así como un ecosistema de off-the-shelf configuraciones, como los gráficos de Helm. En estas situaciones, es posible que dude en realizar cambios en los NodeSelectors de una implementación. La alternativa es usar Taints.

Por ejemplo: `--register-with-taints='os=windows:NoSchedule'`

Si está utilizando EKS, eksctl ofrece formas de aplicar contaminaciones a través de ClusterConfig:

```
NodeGroups:
  - name: windows-ng
    amiFamily: WindowsServer2022FullContainer
    ...
    labels:
      nodeclass: windows2022
    taints:
      os: "windows:NoSchedule"
```

Como consecuencia de lo cual todos los nodos de Windows se ven afectados, el programador no programará los pods en esos nodos a menos que toleren dicha contaminación. Ejemplo de manifiesto de pod:

```
nodeSelector:
    kubernetes.io/os: windows
tolerations:
    - key: "os"
      operator: "Equal"
      value: "windows"
      effect: "NoSchedule"
```

Manejo de varias compilaciones de Windows en el mismo clúster

La imagen base del contenedor de Windows utilizada por cada pod debe coincidir con la misma versión de compilación del núcleo que el nodo. Si desea utilizar varias compilaciones de Windows Server en el mismo clúster, debe establecer etiquetas de nodo adicionales, como nodeSelectors, o utilizar una etiqueta llamada windows-build.

Kubernetes 1.17 añade automáticamente una nueva etiqueta node.kubernetes.io/windows-build para simplificar la administración de varias compilaciones de Windows en el mismo clúster. Si utilizas una versión anterior, se recomienda añadir esta etiqueta manualmente a los nodos de Windows.

Esta etiqueta refleja el número principal, secundario y de compilación de Windows que deben coincidir para garantizar la compatibilidad. A continuación se muestran los valores que se utilizan en la actualidad para cada versión de Windows Server.

Es importante tener en cuenta que Windows Server pasará al canal de servicio a largo plazo (LTSC) como canal de versiones principal. El canal semestral (SAC) de Windows Server se retiró el 9 de agosto de 2022. No habrá futuras versiones SAC de Windows Server.

Product Name (Nombre del producto)	Número (s) de compilación
Servidor completo LTSC 2022	10.0.20348
Núcleo de servidor LTSC 2019	10.0.17763

Es posible comprobar la versión de compilación del sistema operativo mediante el siguiente comando:

```
kubectl get nodes -o wide
```

El resultado de KERNEL-VERSION coincide con la versión de compilación del sistema operativo Windows.

```
NAME                          STATUS   ROLES    AGE   VERSION               INTERNAL-
IP   EXTERNAL-IP     OS-IMAGE                         KERNEL-VERSION
 CONTAINER-RUNTIME
ip-10-10-2-235.ec2.internal   Ready    <none>   23m   v1.24.7-eks-fb459a0
 10.10.2.235   3.236.30.157    Windows Server 2022 Datacenter   10.0.20348.1607
         containerd://1.6.6
```

```
ip-10-10-31-27.ec2.internal   Ready    <none>   23m   v1.24.7-eks-fb459a0
 10.10.31.27   44.204.218.24   Windows Server 2019 Datacenter   10.0.17763.4131
        containerd://1.6.6
ip-10-10-7-54.ec2.internal    Ready    <none>   31m   v1.24.11-eks-a59e1f0   10.10.7.54
    3.227.8.172     Amazon Linux 2                  5.10.173-154.642.amzn2.x86_64
 containerd://1.6.19
```

En el siguiente ejemplo, se aplica un NodeSelector adicional al manifiesto del pod para que coincida con la versión de compilación de Windows correcta al ejecutar diferentes versiones de sistemas operativos de grupos de nodos de Windows.

```
nodeSelector:
    kubernetes.io/os: windows
    node.kubernetes.io/windows-build: '10.0.20348'
tolerations:
    - key: "os"
    operator: "Equal"
    value: "windows"
    effect: "NoSchedule"
```

La simplificación y la tolerancia NodeSelector en el pod se manifiestan mediante RuntimeClass

También puede utilizarlos RuntimeClass para simplificar el proceso de uso de contaminantes y toleraciones. Esto se puede lograr creando un RuntimeClass objeto que se utilice para encapsular estas manchas y tolerancias.

Cree un RuntimeClass ejecutando el siguiente manifiesto:

```
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"
```

Una vez creada la clase Runtimeclass, asígnala como especificación en el manifiesto del Pod:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: iis-2022
  labels:
    app: iis-2022
spec:
  replicas: 1
  template:
    metadata:
      name: iis-2022
      labels:
        app: iis-2022
    spec:
      runtimeClassName: windows-2022
      containers:
      - name: iis
```

Soporte para grupos de nodos gestionados

Para ayudar a los clientes a ejecutar sus aplicaciones de Windows de una manera más ágil, AWS lanzó la compatibilidad con Amazon EKS Managed Node Group (MNG) para contenedores de Windows el 15 de diciembre de 2022. Para ayudar a alinear los equipos de operaciones, Windows utiliza MNGs los mismos flujos de trabajo y herramientas que Linux. MNGs Se admiten las versiones completas y principales de la familia AMI (Amazon Machine Image) de Windows Server 2019 y 2022.

Los grupos de nodos gestionados (MNG) admiten las siguientes familias de AMI.

Familia AMI
Windows_Core_2019_x86_64
Windows_Full_2019_x86_64
Windows_Core_2022_x86_64

Familia AMI
Windows_Full_2022_x86_64

Documentación adicional

Documentación oficial de AWS: https://docs.aws.amazon.com/eks/ latest/userguide/windows - support.html

Para comprender mejor cómo funciona Pod Networking (CNI), consulte el siguiente enlace: - networking.html https://docs.aws.amazon.com/eks/ latest/userguide/pod

Blog de AWS sobre la implementación de un grupo de nodos gestionado para Windows en EKS: https://aws.amazon.com/blogs/ containers/ -/deploying-amazon-eks-windowsmanaged-node-groups

Contextos de seguridad de los pods

Las políticas de seguridad de Pod (PSP) y los estándares de seguridad de Pod (PSS) son dos formas principales de reforzar la seguridad en Kubernetes. Tenga en cuenta que PodSecurityPolicy está en desuso a partir de la versión 1.21 de Kubernetes y se eliminará en la versión 1.25. El estándar de seguridad de Pod (PSS) es el enfoque recomendado por Kubernetes para reforzar la seguridad en el futuro.

La política de seguridad de Pod (PSP) es una solución nativa de Kubernetes para implementar políticas de seguridad. La PSP es un recurso a nivel de clúster que controla los aspectos sensibles a la seguridad de la especificación del Pod. Con la política de seguridad de los pods, puedes definir un conjunto de condiciones que los pods deben cumplir para que el clúster los acepte. La función PSP está disponible desde los inicios de Kubernetes y está diseñada para impedir que se creen pods mal configurados en un clúster determinado.

Para obtener más información sobre las políticas de seguridad de los pods, consulta la documentación de Kubernetes. Según la política de obsolescencia de Kubernetes, las versiones anteriores dejarán de recibir soporte nueve meses después de que la función quede obsoleta.

Por otro lado, los estándares de seguridad de los pods (PSS), que son el enfoque de seguridad recomendado y que normalmente se implementan mediante contextos de seguridad, se definen como parte de las especificaciones del pod y del contenedor en el manifiesto del pod. El PSS es el estándar oficial que el equipo del proyecto de Kubernetes ha definido para abordar las mejores

prácticas relacionadas con la seguridad de los pods. Define políticas como las de referencia (mínimamente restrictivas, predeterminadas), las privilegiadas (no restrictivas) y las restringidas (las más restrictivas).

Recomendamos comenzar con el perfil de referencia. El perfil de referencia del PSS proporciona un equilibrio sólido entre la seguridad y las posibles fricciones, ya que requiere una lista mínima de excepciones y sirve como un buen punto de partida para la seguridad de la carga de trabajo. Si está utilizando actualmente, le PSPs recomendamos que cambie a PSS. Puedes encontrar más detalles sobre las políticas de PSS en la documentación de Kubernetes. Estas políticas se pueden aplicar con varias herramientas, incluidas las de OPA y Kyverno. Por ejemplo, Kyverno proporciona la colección completa de políticas de PSS aquí.

La configuración del contexto de seguridad permite otorgar privilegios para seleccionar procesos, usar perfiles de programas para restringir las capacidades a programas individuales, permitir la escalada de privilegios y filtrar las llamadas al sistema, entre otras cosas.

En lo que respecta a los contextos de seguridad, los pods de Windows de Kubernetes presentan algunas limitaciones y elementos que los diferencian de las cargas de trabajo estándar basadas en Linux.

Windows usa un objeto Job por contenedor con un filtro de espacio de nombres del sistema para incluir todos los procesos en un contenedor y proporcionar un aislamiento lógico del host. No hay forma de ejecutar un contenedor de Windows sin el filtrado de espacios de nombres establecido. Esto significa que los privilegios del sistema no se pueden hacer valer en el contexto del host y, por lo tanto, los contenedores privilegiados no están disponibles en Windows.

Las siguientes `windowsOptions` son las únicas opciones del contexto de seguridad de Windows documentadas, mientras que el resto son opciones generales del contexto de seguridad

Para obtener una lista de los atributos del contexto de seguridad compatibles con Windows y Linux, consulte la documentación oficial aquí.

La configuración específica del pod se aplica a todos los contenedores. Si no se especifica, se PodSecurityContext utilizarán las opciones del. Si se establece en ambos SecurityContext valores PodSecurityContext, SecurityContext prevalecerá el valor especificado en.

Por ejemplo, la configuración de runAsUser nombres para los pods y los contenedores, que es una opción de Windows, equivale aproximadamente a la runAsUser configuración específica de Linux y, en el siguiente manifiesto, el contexto de seguridad específico del pod se aplica a todos los contenedores

```
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
```

Mientras que, a continuación, el contexto de seguridad a nivel de contenedor anula el contexto de seguridad a nivel de pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: run-as-username-container-demo
spec:
  securityContext:
    windowsOptions:
      runAsUserName: "ContainerUser"
  containers:
  - name: run-as-username-demo
    ..
    securityContext:
        windowsOptions:
            runAsUserName: "ContainerAdministrator"
  nodeSelector:
    kubernetes.io/os: windows
```

Ejemplos de valores aceptables para el campo runAsUser Nombre: ContainerAdministrator, ContainerUser, NT AUTHORITY\ NETWORK SERVICE, NT AUTHORITY\ LOCAL SERVICE

Por lo general, es una buena idea ejecutar los contenedores con ContainerUser los pods de Windows. Los usuarios no se comparten entre el contenedor y el ContainerAdministrator host, pero tienen privilegios adicionales dentro del contenedor. Tenga en cuenta que hay limitaciones de nombre de usuario que debe tener en cuenta.

Un buen ejemplo de cuándo usarlo ContainerAdministrator es configurar PATH. Puedes usar la directiva USER para hacerlo, de la siguiente manera:

```
USER ContainerAdministrator
RUN setx /M PATH "%PATH%;C:/your/path"
USER ContainerUser
```

Tenga en cuenta también que los secretos están escritos en texto claro en el volumen del nodo (a diferencia de tmpfs/in-memory en Linux). Esto significa que tienes que hacer dos cosas

- Utilice el archivo ACLs para proteger la ubicación del archivo secreto
- Utilice el cifrado a nivel de volumen mediante BitLocker

Opciones de almacenamiento persistente

¿Qué es un plugin integrado en el árbol o un plugin out-of-tree de volumen?

Antes de la introducción de la interfaz de almacenamiento de contenedores (CSI), todos los complementos de volumen estaban integrados en un árbol, lo que significaba que se creaban, vinculaban, compilaban y distribuían con los archivos binarios principales de Kubernetes y ampliaban la API principal de Kubernetes. Esto significaba que para añadir un nuevo sistema de almacenamiento a Kubernetes (un complemento de volumen) era necesario introducir el código en el repositorio principal de códigos de Kubernetes.

Out-of-tree Los complementos de volumen se desarrollan de forma independiente del código base de Kubernetes y se despliegan (instalan) en los clústeres de Kubernetes como extensiones. Esto permite a los proveedores actualizar los controladores, es decir out-of-band, de forma independiente del ciclo de lanzamiento de Kubernetes. Esto es posible en gran medida porque Kubernetes ha creado una interfaz de almacenamiento (CSI) que proporciona a los proveedores una forma estándar de interactuar con los k8s.

Puede obtener más información sobre las clases de almacenamiento de Amazon Elastic Kubernetes Services (EKS) y los controladores CSI en .html https://docs.aws.amazon.com/eks/ latest/userguide/ storage

Complemento In-Tree Volume para Windows

Los volúmenes de Kubernetes permiten implementar aplicaciones en Kubernetes con requisitos de persistencia de datos. La administración de los volúmenes persistentes consiste en contenedores

individuales en un podprovisioning/de-provisioning/resizing of volumes, attaching/detaching a volume to/from a Kubernetes node, and mounting/dismounting a volume to/from. El código para implementar estas acciones de administración de volúmenes para un protocolo o servidor de almacenamiento específico se envía en forma de un complemento de volumen de Kubernetes (In-Tree Volume Plugins). En Amazon Elastic Kubernetes Services (EKS), Windows admite la siguiente clase de complementos de volumen de Kubernetes:

Complemento awsElasticBlockde volumen integrado en el árbol: Store

Para utilizar el complemento de volumen integrado en el árbol en los nodos de Windows, es necesario crear un complemento adicional StorageClass para utilizar NTFS como FSType. En EKS, el valor predeterminado StorageClass usa ext4 como FSType predeterminado.

A StorageClass proporciona a los administradores una forma de describir las «clases» de almacenamiento que ofrecen. Las diferentes clases pueden asignarse a quality-of-service niveles, políticas de respaldo o políticas arbitrarias determinadas por los administradores del clúster. Kubernetes no tiene opiniones sobre lo que representan las clases. Este concepto a veces se denomina «perfiles» en otros sistemas de almacenamiento.

Puede comprobarlo ejecutando el siguiente comando:

```
kubectl describe storageclass gp2
```

Salida:

```
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>
```

Para crear el nuevo StorageClass que sea compatible con NTFS, usa el siguiente manifiesto:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gp2-windows
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  fsType: ntfs
volumeBindingMode: WaitForFirstConsumer
```

Cree el StorageClass ejecutando el siguiente comando:

```
kubectl apply -f NTFSStorageClass.yaml
```

El siguiente paso es crear una reclamación de volumen persistente (PVC).

Un PersistentVolume (PV) es una pieza de almacenamiento del clúster que ha sido aprovisionada por un administrador o aprovisionada dinámicamente mediante PVC. Es un recurso del clúster del mismo modo que un nodo es un recurso del clúster. Este objeto de API captura los detalles de la implementación del almacenamiento, ya sea NFS, iSCSI o cloud-provider-specific un sistema de almacenamiento.

A PersistentVolumeClaim (PVC) es una solicitud de almacenamiento realizada por un usuario. Las reclamaciones pueden solicitar tamaños y modos de acceso específicos (por ejemplo, se pueden montar ReadWriteOnce ReadOnlyMany o ReadWriteMany).

Los usuarios necesitan PersistentVolumes diferentes atributos, como el rendimiento, para diferentes casos de uso. Los administradores de clústeres deben poder ofrecer una variedad PersistentVolumes que se diferencie en algo más que en el tamaño y los modos de acceso, sin exponer a los usuarios a los detalles de cómo se implementan esos volúmenes. Para estas necesidades, existe el StorageClass recurso.

En el siguiente ejemplo, el PVC se creó dentro de las ventanas del espacio de nombres.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ebs-windows-pv-claim
  namespace: windows
```

```
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gp2-windows
  resources:
    requests:
      storage: 1Gi
```

Cree el PVC ejecutando el siguiente comando:

```
kubectl apply -f persistent-volume-claim.yaml
```

El siguiente manifiesto crea un pod de Windows, VolumeMount lo configura `C:\Data` y utiliza el PVC como almacenamiento adjunto`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'
```

Pruebe los resultados accediendo al pod de Windows a través de PowerShell:

```
kubectl exec -it podname powershell -n windows
```

Dentro del pod de Windows, ejecute: `ls`

Salida:

```
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
```

El directorio de datos lo proporciona el volumen de EBS.

Out-of-tree para Windows

El código asociado a los complementos de CSI se envía como out-of-tree scripts y binarios que normalmente se distribuyen como imágenes de contenedor y se implementan mediante construcciones estándar de Kubernetes como y. DaemonSets StatefulSets Los complementos de CSI gestionan una amplia gama de acciones de gestión de volúmenes en Kubernetes. Los

complementos de CSI suelen consistir en complementos de nodos (que se ejecutan en cada nodo como si fueran) y complementos de controlador. DaemonSet

Los complementos de nodos CSI (especialmente los asociados a volúmenes persistentes expuestos como dispositivos de bloques o a través de un sistema de archivos compartido) deben realizar diversas operaciones privilegiadas, como escanear dispositivos de disco, montar sistemas de archivos, etc. Estas operaciones son diferentes para cada sistema operativo anfitrión. En el caso de los nodos de trabajo de Linux, los complementos de nodos CSI en contenedores se implementan normalmente como contenedores privilegiados. En el caso de los nodos de trabajo de Windows, las operaciones privilegiadas de los complementos de nodos CSI en contenedores se admiten mediante csi-proxy, un binario independiente y administrado por la comunidad que debe estar preinstalado en cada nodo de Windows.

La AMI optimizada para Windows de Amazon EKS incluye CSI-Proxy a partir de abril de 2022. Los clientes pueden utilizar el controlador CSI SMB en los nodos de Windows para acceder a Amazon FSx for Windows File Server, Amazon FSx for NetApp ONTAP SMB Shares y and/or AWS Storage Gateway — File Gateway.

El siguiente blog contiene detalles de implementación sobre cómo configurar el controlador SMB CSI para usar Amazon FSx for Windows File Server como almacenamiento persistente para Windows Pods.

Servidor FSx de archivos Amazon para Windows

Una opción es utilizar Amazon FSx for Windows File Server a través de una función SMB denominada SMB Global Mapping, que permite montar un recurso compartido SMB en el host y, a continuación, pasar los directorios de ese recurso compartido a un contenedor. No es necesario configurar el contenedor con un servidor, recurso compartido, nombre de usuario o contraseña específicos, sino que todo se gestiona en el servidor. El contenedor funcionará igual que si tuviera almacenamiento local.

> El mapeo global de pequeñas y medianas empresas es transparente para el organizador y se monta a través de él, HostPath lo que puede implicar problemas de seguridad.

En el ejemplo siguiente, la ruta `G:\Directory\app-state` es un recurso compartido SMB en el nodo de Windows.

```
apiVersion: v1
```

```
kind: Pod
metadata:
  name: test-fsx
spec:
  containers:
  - name: test-fsx
    image: mcr.microsoft.com/windows/servercore:ltsc2019
    command:
      - powershell.exe
      - -command
      - "Add-WindowsFeature Web-Server; Invoke-WebRequest -UseBasicParsing
 -Uri 'https://dotnetbinaries.blob.core.windows.net/servicemonitor/2.0.1.6/
ServiceMonitor.exe' -OutFile 'C:\\ServiceMonitor.exe'; echo '<html><body><br/
><br/><marquee><H1>Hello EKS!!!<H1><marquee></body><html>' > C:\\inetpub\\wwwroot\
\default.html; C:\\ServiceMonitor.exe 'w3svc'; "
    volumeMounts:
      - mountPath: C:\dotnetapp\app-state
        name: test-mount
  volumes:
    - name: test-mount
      hostPath:
        path: G:\Directory\app-state
        type: Directory
  nodeSelector:
      beta.kubernetes.io/os: windows
      beta.kubernetes.io/arch: amd64
```

El siguiente blog contiene detalles de implementación sobre cómo configurar Amazon FSx for Windows File Server como almacenamiento persistente para Windows Pods.

Fortalecimiento de las imágenes de los contenedores de Windows

¿Está reforzando las imágenes de sus contenedores de Windows? A lo largo de los años, he trabajado con clientes de todo el mundo para ayudarlos a migrar las cargas de trabajo antiguas a contenedores, especialmente las cargas de trabajo de Windows. Con más de 20 años de experiencia, he visto cómo las organizaciones dedican esfuerzos y recursos considerables a reforzar sus servidores Windows, implementando desde los puntos de referencia del CIS hasta la protección antivirus en tiempo de ejecución para proteger los datos confidenciales.

Sin embargo, ha surgido una tendencia preocupante. A medida que estas máquinas virtuales de alta seguridad se modernizan para convertirlas en contenedores, se pasan por alto muchas prácticas

críticas de protección. Las mejores prácticas de seguridad de Windows, desde la imagen básica (OS) hasta los servicios web como el IIS, suelen pasarse por alto y la mayor parte del tiempo se centra únicamente en proteger el host del contenedor. Es fundamental reconocer que, si bien los contenedores funcionan en espacios de nombres aislados, comparten las primitivas del núcleo con el anfitrión. Los atacantes suelen estar más interesados en el movimiento lateral que en atacar directamente al anfitrión del contenedor, lo que les permite aprovechar la débil configuración de seguridad del contenedor y acceder a datos confidenciales.

El objetivo de la documentación es destacar algunos ajustes de seguridad esenciales que se deben implementar específicamente para los contenedores de Windows que alojan `ASP.NET` sitios web en IIS. Nos centraremos en cuatro áreas clave:

- Políticas de seguridad de las cuentas
- Políticas de auditoría
- Prácticas recomendadas de seguridad de IIS
- Principio de privilegios mínimos

Empezaremos analizando por qué cada una de estas configuraciones de seguridad es vital para proteger sus contenedores de Windows, analizando los riesgos específicos que mitigan y los beneficios de seguridad que ofrecen. A continuación, analizaremos un fragmento de código que muestra cómo implementar estas configuraciones correctamente en su Dockerfile, garantizando que su contenedor esté protegido contra posibles amenazas. Por último, analizaremos cada configuración en detalle y ofreceremos una explicación completa de su función, su impacto en la seguridad de los contenedores y cómo contribuye a proteger sus aplicaciones. Este enfoque no solo le mostrará cómo aplicar estas mejores prácticas, sino que también le permitirá comprender por qué son esenciales para mantener una postura de seguridad sólida en entornos contenerizados.

1. Configure las políticas de cuentas (contraseña o bloqueo) mediante las políticas de seguridad y el registro locales

Windows Server Core es una opción de instalación mínima que está disponible como parte de la [AMI de Windows optimizada para EKS] (https://docs.aws.amazon.com/eks/latest/userguide/eks-optimized-windows-ami .html). La configuración de las políticas de cuenta (contraseña o bloqueo) mediante políticas de seguridad locales y el registro refuerza la seguridad del sistema al aplicar reglas sólidas de contraseña y bloqueo. Estas políticas exigen que los usuarios creen contraseñas seguras con una longitud y complejidad mínimas definidas, a fin de protegerlas de los ataques habituales relacionados con las contraseñas.

Al establecer una antigüedad máxima para las contraseñas, se les pide a los usuarios que las actualicen periódicamente, lo que reduce la probabilidad de que sus credenciales estén comprometidas. Las políticas de bloqueo añaden un nivel adicional de protección al bloquear temporalmente las cuentas tras un número determinado de intentos fallidos de inicio de sesión, lo que ayuda a prevenir los ataques de fuerza bruta. La configuración de estos ajustes a través del Registro de Windows permite a los administradores aplicar estas medidas de seguridad a nivel del sistema, lo que garantiza la uniformidad y el cumplimiento en toda la organización. La aplicación de estas políticas de cuentas en un contenedor de Windows es esencial para mantener la coherencia de la seguridad, aunque los contenedores suelen ser efímeros y están destinados a cargas de trabajo aisladas:

Coherencia de seguridad

- Cumplimiento: aplicar políticas de contraseñas y reglas de bloqueo coherentes en los contenedores ayuda a mantener el cumplimiento de las normas de seguridad, especialmente en entornos que requieren controles de acceso estrictos (por ejemplo, el cumplimiento de normas como la HIPAA o el PCI-DSS).
- Contenedores reforzados: la aplicación de esta configuración garantiza que su contenedor de Windows esté protegido contra el acceso no autorizado o los ataques basados en contraseñas, lo que alinea la postura de seguridad de su contenedor con las políticas de seguridad más amplias del sistema.

Protección contra los ataques de fuerza bruta

- Bloqueo de cuentas: esta configuración ayuda a evitar los intentos de inicio de sesión por la fuerza bruta al bloquear las cuentas después de un número específico de intentos fallidos de inicio de sesión. Esto evita que los atacantes prueben con un número ilimitado de contraseñas.
- Complejidad de las contraseñas: requerir contraseñas complejas con una longitud suficiente reduce la probabilidad de que se exploten las contraseñas débiles, incluso en entornos aislados y en contenedores.

Escenarios multiusuario

- Si la aplicación contenerizada está diseñada para gestionar varios usuarios o requiere la autenticación de los usuarios, aplicar políticas de contraseñas garantiza que las cuentas de

usuario del contenedor cumplan estrictas normas de seguridad, lo que limita el acceso solo a los usuarios autorizados.

Contenedores de Windows persistentes

- Si bien los contenedores generalmente se consideran efímeros, algunos contenedores de Windows pueden ejecutar servicios a largo plazo o gestionar los usuarios, por lo que es importante aplicar las políticas de seguridad adecuadas de forma similar a la de un servidor Windows normal.

Coherencia en entornos híbridos

- Si ejecuta máquinas virtuales y contenedores en su infraestructura, aplicar las mismas políticas de seguridad (por ejemplo, password/lockout políticas) en todos los entornos garantiza estándares de seguridad uniformes, lo que simplifica la gobernanza y la administración.

En resumen, la aplicación de estas políticas de cuentas en los contenedores de Windows garantiza que sus contenedores no sean un punto débil en su estrategia de seguridad, ya que protegen contra los ataques con contraseñas y refuerzan la coherencia en todo el entorno.

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
```

Explicación:

En esta sección, se configuran las políticas de la cuenta para la configuración de contraseñas y bloqueos a través del Registro de Windows. Estas políticas ayudan a reforzar la seguridad al controlar los requisitos de contraseñas y los umbrales de bloqueo de las cuentas.

1. MinimumPasswordLength (MINPWLEN) = 14 Esta configuración define el número mínimo de caracteres de una contraseña. El intervalo es de 0 a 14 caracteres; el valor predeterminado es de seis caracteres.
2. MaximumPasswordAge (MAXPWAGE) = 60 Esta configuración define el número máximo de días de validez de una contraseña. No se especifica ningún límite mediante UNLIMITED. /MAXPWAGE

no puede ser inferior a /MINPWAGE. El intervalo es de 1 a 999; el valor predeterminado es de 90 días

3. Umbral de bloqueo (LOCKOUTTHRESHOLD) = 5 Esta configuración define el umbral para los intentos de inicio de sesión fallidos. Tras 5 intentos incorrectos, la cuenta se bloqueará.

Esta configuración ayuda a mejorar la seguridad de las contraseñas y a prevenir los ataques de fuerza bruta al aplicar políticas de contraseñas seguras y bloquear las cuentas tras un número determinado de intentos fallidos de inicio de sesión.

2. Políticas de auditoría

Las políticas de auditoría son importantes para los contenedores de Windows porque proporcionan una visibilidad fundamental de los eventos de seguridad, como los intentos de inicio de sesión y el uso de privilegios, y ayudan a detectar el acceso no autorizado, supervisar la actividad de los usuarios y garantizar el cumplimiento de las normas reglamentarias. Incluso en la naturaleza efímera de los contenedores, los registros de auditoría son esenciales para la investigación de incidentes, la detección proactiva de amenazas y el mantenimiento de una postura de seguridad uniforme en todos los entornos contenerizados.

Supervisión y cumplimiento de la seguridad:

- Realice un seguimiento de las actividades de los usuarios: las políticas de auditoría permiten a los administradores supervisar las actividades de los usuarios, como los intentos de inicio de sesión y el uso de privilegios, dentro del contenedor. Esto es fundamental para detectar el acceso no autorizado o el comportamiento sospechoso.
- Cumplimiento normativo: muchas organizaciones están obligadas a registrar los eventos de seguridad para cumplir con normativas como la HIPAA, el PCI-DSS y el GDPR. Habilitar las políticas de auditoría en los contenedores garantiza el cumplimiento de estos requisitos, incluso en entornos contenerizados.

Investigación de incidentes:

- Análisis y análisis forenses: si una aplicación o un servicio en contenedores se ve comprometido, los registros de auditoría pueden proporcionar información valiosa para el análisis posterior al incidente. Ayudan a los equipos de seguridad a rastrear las acciones emprendidas por los atacantes o a identificar cómo se produjo una infracción.

- Detección en tiempo real: los registros de auditoría permiten a los administradores configurar alertas en tiempo real para eventos críticos (por ejemplo, intentos fallidos de inicio de sesión o aumento de privilegios). Esta supervisión proactiva ayuda a detectar los ataques de forma temprana y permite tiempos de respuesta más rápidos.

Consistencia en todos los entornos:

- Postura de seguridad uniforme: al aplicar políticas de auditoría en los contenedores a través del registro, se garantizan prácticas de seguridad coherentes en los entornos contenerizados y no contenerizados. Esto evita que los contenedores se conviertan en un punto ciego para la supervisión de la seguridad.
- Visibilidad en entornos híbridos: para las organizaciones que utilizan servidores y contenedores Windows tradicionales, las políticas de auditoría proporcionan una visibilidad y un control similares en todas las plataformas, lo que facilita y hace que la administración sea más sencilla y eficaz.

Seguimiento de las operaciones privilegiadas:

- Auditoría de uso de privilegios: en entornos de contenedores donde las aplicaciones se ejecutan con privilegios elevados o donde se realizan tareas administrativas, la auditoría de las operaciones con privilegios garantiza la rendición de cuentas. Puede registrar quién accedió a recursos confidenciales o realizó tareas críticas dentro del contenedor.
- Evite el abuso de privilegios: al supervisar el uso de los privilegios, puede detectar si usuarios no autorizados intentan aumentar sus privilegios o acceder a áreas restringidas del contenedor, lo que ayuda a prevenir ataques internos o externos.

Detección de intentos de acceso no autorizados:

- Intentos de inicio de sesión fallidos: la activación de políticas de auditoría para los intentos de inicio de sesión fallidos ayuda a identificar los ataques de fuerza bruta o los intentos no autorizados de acceder a aplicaciones en contenedores. Esto proporciona visibilidad sobre quién intenta acceder al sistema y con qué frecuencia.
- Supervisión del bloqueo de cuentas: la auditoría de los eventos de bloqueo de cuentas permite a los administradores detectar e investigar posibles bloqueos causados por actividades sospechosas o maliciosas.

Seguridad persistente incluso en entornos efímeros:

- Efímeros pero seguros: si bien los contenedores son efímeros, lo que significa que se pueden eliminar y volver a crear con frecuencia, la auditoría sigue desempeñando un papel clave para garantizar que los eventos de seguridad se capturen mientras el contenedor está en funcionamiento. Esto garantiza que los eventos de seguridad críticos se registren durante todo el ciclo de vida del contenedor.

Registro centralizado:

- Reenvío de registros a sistemas centralizados: los contenedores se pueden integrar con sistemas de registro centralizados (por ejemplo, ELK stack, AWS CloudWatch) para capturar los registros de auditoría de varias instancias de contenedores. Esto permite analizar y correlacionar mejor los eventos de seguridad en toda su infraestructura.

Dockerfile:

```
# 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
```

Explicación:

En esta sección se configuran las políticas de auditoría mediante modificaciones del registro. Las políticas de auditoría controlan los eventos de seguridad que registra Windows, lo que ayuda a supervisar y detectar los intentos de acceso no autorizados.

1. SCENoApplyLegacyAuditPolicy = 0 Esto deshabilita el formato de política de auditoría anterior, lo que permite introducir políticas de auditoría más detalladas en las versiones posteriores de Windows. Esto es importante para las configuraciones de auditoría modernas.
2. Subcategoría de Auditpol: «Inicio de sesión» Esta configuración permite auditar los eventos de inicio de sesión correctos y erróneos. El valor 3 significa que Windows registrará los intentos de

inicio de sesión correctos y fallidos. Esto ayuda a controlar quién accede al sistema y a detectar los intentos fallidos de inicio de sesión.

Estas políticas de auditoría son fundamentales para la supervisión y el cumplimiento de la seguridad, ya que proporcionan registros detallados de los eventos de seguridad importantes, como los intentos de inicio de sesión y el uso de operaciones privilegiadas.

3. Prácticas recomendadas de seguridad de IIS para contenedores de Windows

La implementación de las prácticas recomendadas de IIS en los contenedores de Windows es importante por varios motivos, ya que garantiza que las aplicaciones sean seguras, de alto rendimiento y escalables. Si bien los contenedores proporcionan aislamiento y un entorno ligero, requieren una configuración adecuada para evitar vulnerabilidades y problemas operativos. Estos son los motivos por los que es fundamental seguir las prácticas recomendadas para IIS en los contenedores de Windows:

Seguridad

- Prevención de vulnerabilidades comunes: IIS suele ser objeto de ataques como el uso de scripts entre sitios (XSS), el robo de clics y la divulgación de información. La implementación de encabezados de seguridad (por ejemplo, X-Content-Type-Options X-Frame-Options, y Strict-Transport-Security) ayuda a proteger su aplicación de estas amenazas.
- El aislamiento no es suficiente: los contenedores están aislados, pero una instancia de IIS mal configurada puede exponer información confidencial, como los detalles de la versión del servidor, las listas de directorios o las comunicaciones no cifradas. Al deshabilitar funciones como la navegación por directorios y eliminar el encabezado de la versión de IIS, se minimiza la superficie de ataque.
- Cifrado y HTTPS: las prácticas recomendadas, como hacer cumplir las conexiones únicamente con HTTPS, garantizan que los datos en tránsito estén cifrados y protegen la información confidencial de la interceptación.

Rendimiento

- Uso eficiente de los recursos: las mejores prácticas de IIS, como permitir la compresión dinámica y estática, reducen el uso del ancho de banda y mejoran los tiempos de carga. Estas optimizaciones

son especialmente importantes en los entornos contenerizados, donde los recursos se comparten entre los contenedores y el sistema anfitrión.

- Registro optimizado: la configuración adecuada del registro (por ejemplo, incluyendo el X-Forwarded-For encabezado) garantiza que se pueda rastrear la actividad de los clientes y, al mismo tiempo, minimizar la sobrecarga de registro innecesaria. Esto le ayuda a recopilar datos relevantes para la resolución de problemas sin degradar el rendimiento.

Escalabilidad y mantenibilidad

- Coherencia en todos los entornos: siguiendo las prácticas recomendadas, se asegura de que la configuración de IIS sea coherente en varias instancias de contenedores. Esto simplifica el escalado y garantiza que, cuando se implementen nuevos contenedores, sigan las mismas pautas de seguridad y rendimiento.
- Configuraciones automatizadas: las mejores prácticas en Dockerfiles, como configurar los permisos de las carpetas y deshabilitar las funciones innecesarias, garantizan que cada contenedor nuevo se configure automáticamente de forma correcta. Esto reduce la intervención manual y reduce el riesgo de errores humanos.

Conformidad

- Cumplimiento de los requisitos reglamentarios: muchos sectores tienen requisitos reglamentarios estrictos (por ejemplo, el PCI-DSS o la HIPAA) que exigen medidas de seguridad específicas, como las comunicaciones cifradas (HTTPS) y el registro de las solicitudes de los clientes. Seguir las mejores prácticas de IIS en contenedores ayuda a garantizar el cumplimiento de estos estándares.
- Auditabilidad: la implementación de políticas de auditoría y un registro seguro permite la trazabilidad de los eventos, lo cual es fundamental en las auditorías. Por ejemplo, al registrar el X-Forwarded-For encabezado se garantiza que las direcciones IP de los clientes se registren correctamente en las arquitecturas basadas en proxies.

Minimizar el riesgo en entornos compartidos

- Evitar configuraciones incorrectas: los contenedores comparten el núcleo del host y, aunque estén aislados unos de otros, una instancia de IIS mal configurada podría exponer vulnerabilidades o crear cuellos de botella en el rendimiento. Las prácticas recomendadas garantizan que

cada instancia de IIS se ejecute de forma óptima, lo que reduce el riesgo de problemas entre contenedores.

- Acceso con privilegios mínimos: establecer los permisos adecuados para las carpetas y los archivos del contenedor (por ejemplo, mediante Set-Acl in PowerShell) garantiza que los usuarios y los procesos del contenedor solo tengan el acceso necesario, lo que reduce el riesgo de que se aumenten los privilegios o se alteren los datos.

Resiliencia en entornos efímeros

- Naturaleza efímera de los contenedores: los contenedores suelen durar poco y se reconstruyen con frecuencia. La aplicación de las mejores prácticas de IIS garantiza que cada contenedor se configure de forma segura y coherente, independientemente del número de veces que se vuelva a implementar. Esto evita que se introduzcan errores de configuración con el paso del tiempo.
- Mitigar los posibles errores de configuración: al aplicar automáticamente las mejores prácticas (por ejemplo, deshabilitar los protocolos o encabezados débiles), se minimiza el riesgo de que se produzcan errores de configuración durante el reinicio o las actualizaciones de los contenedores.

Dockerfile:

```
# 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"
```

Explicación:

Este comando configura IIS para que registre el X-Forwarded-For encabezado, que se suele utilizar para capturar la dirección IP original del cliente cuando una solicitud pasa por un proxy o un balanceador de cargas. De forma predeterminada, IIS solo registra la dirección IP del balanceador de carga o del proxy inverso, por lo que agregar este campo de registro personalizado ayuda a rastrear la IP real del cliente para realizar auditorías de seguridad, análisis y solución de problemas.

1. X-Forwarded-For encabezado que se suele utilizar para capturar la dirección IP original del cliente cuando una solicitud pasa por un proxy o un balanceador de cargas. De forma predeterminada, IIS solo registra la dirección IP del balanceador de carga o del proxy inverso, por lo que agregar este campo de registro personalizado ayuda a rastrear la verdadera IP del cliente para realizar auditorías de seguridad, análisis y solución de problemas.
2. Strict-Transport-Security (HSTS) Garantiza que los navegadores solo se comuniquen a través de HTTPS. El valor max-age=31536000 especifica que esta política se aplica durante 1 año y includeSubDomains la aplica a todos los subdominios.
3. X-Content-Type-Options impide que los navegadores «rastreen mediante MIME» una respuesta distinta del tipo de contenido declarado. Esto ayuda a prevenir algunos tipos de ataques.
4. La protección X-XSS permite la protección mediante secuencias de comandos entre sitios (XSS) en los navegadores.
5. X-Frame-Options evita que la página se incruste en iframes, lo que la protege contra los ataques de hackeo de clics.
6. Desactivar la divulgación de la versión de IIS Este comando desactiva el encabezado del servidor en las respuestas HTTP, que de forma predeterminada revela la versión de IIS que se está utilizando. Ocultar esta información ayuda a reducir el riesgo de que los atacantes identifiquen y ataquen vulnerabilidades específicas de la versión de IIS.
7. Habilitar conexiones únicamente HTTPS Esta sección (comentada) aplica las conexiones HTTPS y desactiva HTTP. Si no se comenta, el Dockerfile configurará IIS para que escuche solo en el

puerto 443 (HTTPS) y eliminará el enlace HTTP predeterminado en el puerto 80. Esto resulta útil cuando se termina el SSL dentro del contenedor y garantiza que todo el tráfico esté cifrado.

8. Desactivar la exploración de directorios Impide que IIS muestre una lista de directorios cuando no hay ningún documento predeterminado. Esto evita exponer la estructura interna de archivos a los usuarios.
9. Pasar por páginas de error personalizadas Garantiza que, si la aplicación tiene su propio sistema de gestión de errores, IIS permitirá que pasen las páginas de error de la aplicación en lugar de mostrar las páginas de error de IIS predeterminadas.
10. El modo de error detallado configura IIS para mostrar mensajes de error detallados únicamente para las solicitudes locales, lo que ayuda a los desarrolladores a diagnosticar problemas sin exponer información confidencial a usuarios externos.
11. Garantice los permisos de carpeta adecuados Este bloque configura los permisos de carpeta para la raíz web de IIS (C:\inetpub\wwwroot). Establece los permisos de lectura y ejecución para los grupos IIS_IUSRS y Usuarios, lo que garantiza que estos usuarios puedan acceder a la carpeta pero no modificar los archivos. Al establecer los permisos correctos, se minimiza el riesgo de acceso no autorizado o de manipulación de los archivos alojados en el servidor web.

Seguir las prácticas recomendadas de IIS en Windows Containers garantiza que las aplicaciones en contenedores sean seguras, tengan un alto rendimiento y sean escalables. Estas prácticas ayudan a prevenir vulnerabilidades, optimizar el uso de los recursos, garantizar el cumplimiento y mantener la coherencia en todas las instancias de contenedores. Aunque los contenedores están diseñados para estar aislados, es necesaria una configuración adecuada para minimizar los riesgos y garantizar la fiabilidad de la aplicación en los entornos de producción.

4. Principio de privilegios mínimos

El principio de privilegios mínimos (PoLP) es fundamental para los contenedores de Windows por varias razones importantes, especialmente para mejorar la seguridad y minimizar los riesgos en los entornos contenerizados. Este principio dicta que un sistema o aplicación debe funcionar con el nivel mínimo de permisos necesario para funcionar correctamente. Estos son los motivos por los que es importante en los contenedores de Windows:

Minimizar la superficie de ataque

- Los contenedores suelen ejecutar aplicaciones que interactúan con varios componentes del sistema y, cuantos más privilegios tenga una aplicación, más amplio será su acceso a esos componentes. Al limitar los permisos del contenedor solo a lo necesario, PoLP reduce

significativamente la superficie de ataque, lo que dificulta que un atacante explote el contenedor si se ve comprometido.

Limitar el impacto de los contenedores comprometidos

- Si un contenedor de Windows se ve comprometido, ejecutar aplicaciones con privilegios excesivos (por ejemplo, acceso de administrador o de raíz) podría permitir a un atacante hacerse con el control de los archivos críticos del sistema o aumentar los privilegios en todo el host del contenedor. Al aplicar la PoLP, incluso si se infringe un contenedor, el atacante se ve limitado en cuanto a lo que puede hacer, lo que impide una mayor escalada y el acceso a recursos confidenciales u otros contenedores.

Protección en entornos con varios usuarios

- En entornos empresariales o de nube, se pueden ejecutar varios contenedores en la misma infraestructura física o virtual. PoLP garantiza que un contenedor comprometido no tenga la capacidad de acceder a los recursos o datos que pertenecen a otros inquilinos. Este aislamiento es crucial para mantener la seguridad en entornos compartidos con varios inquilinos, ya que protege contra el movimiento lateral entre contenedores.

Mitigar el aumento de privilegios

- Los atacantes pueden utilizar los contenedores que se ejecutan con altos privilegios para aumentar los privilegios dentro del sistema. PoLP mitiga este riesgo al restringir el acceso del contenedor a los recursos del sistema, lo que evita acciones no autorizadas o el aumento de privilegios más allá del entorno del contenedor.

Cumplimiento y auditoría

- Muchos estándares regulatorios y marcos de seguridad (por ejemplo, PCI DSS, HIPAA, GDPR) exigen que los sistemas cumplan con la PoLP para limitar el acceso a los datos confidenciales. El uso de contenedores de Windows con privilegios restringidos ayuda a las organizaciones a cumplir con estas normas y garantiza que las aplicaciones solo tengan acceso a los recursos que necesitan específicamente.

Reducir el riesgo de errores de configuración

- Cuando los contenedores se ejecutan con privilegios innecesarios, incluso un pequeño error de configuración puede provocar graves vulnerabilidades de seguridad. Por ejemplo, si un contenedor que funciona como administrador se expone accidentalmente a Internet, un atacante podría hacerse con el control del sistema. PoLP ayuda a prevenir estos riesgos al utilizar de forma predeterminada privilegios limitados, lo que hace que las configuraciones incorrectas sean menos peligrosas.

Mejora de la postura de seguridad de los contenedores

- Al seguir la PoLP, los contenedores se aíslan mejor del sistema anfitrión subyacente y entre sí. Esto garantiza que sea menos probable que la aplicación contenerizada acceda a los archivos o procesos del sistema o los modifique fuera del ámbito definido, lo que preserva la integridad del sistema operativo anfitrión y otras cargas de trabajo.

Dockerfile:

```
# Strongly recommended that when deploying a Windows server container to any multi-
tenant environment that your application runs via the ContainerUser account
USER ContainerUser
```

Explicación:

En esta sección, el ContainerUser comando USER especifica que la aplicación incluida en el contenedor de Windows debe ejecutarse con la ContainerUser cuenta y no con la cuenta de administrador predeterminada.

Estos son los motivos por los que esto es importante, especialmente en un entorno multiusuario:

1. Principio de privilegio mínimo: la ContainerUser cuenta es un usuario no administrativo con privilegios limitados. Al ejecutar la aplicación con esta cuenta, se sigue el principio del privilegio mínimo, lo que ayuda a minimizar el riesgo de explotación. Si un atacante pusiera en peligro la aplicación, tendría un acceso limitado al sistema, lo que reduciría los posibles daños.
2. Seguridad mejorada: en entornos multiusuario, los contenedores pueden compartir la misma infraestructura subyacente. La ejecución de AS ContainerUser garantiza que, incluso si un contenedor se ve comprometido, no tendrá privilegios administrativos para acceder o modificar archivos críticos del sistema u otros contenedores. Esto reduce considerablemente la superficie de ataque.

3. Evitar el acceso root: de forma predeterminada, los contenedores pueden funcionar con permisos elevados (similar al acceso root en los contenedores de Linux), lo que puede resultar peligroso si se explota. ContainerUser Su uso garantiza que la aplicación no se ejecute con derechos administrativos innecesarios, lo que dificulta que los atacantes aumenten los privilegios.
4. Mejores prácticas para entornos con varios usuarios: en entornos en los que varios usuarios u organizaciones comparten la misma infraestructura (por ejemplo, en la nube), la seguridad es fundamental. La ejecución de aplicaciones con permisos restringidos evita que la aplicación de un inquilino afecte a otras, protegiendo los datos y recursos confidenciales en toda la plataforma.

El ContainerUser comando USER garantiza que la aplicación se ejecute con privilegios mínimos, lo que mejora la seguridad en entornos con varios usuarios al limitar los daños que podrían producirse si el contenedor se viera comprometido. Esta es la mejor práctica para evitar el acceso no autorizado o la escalada de privilegios en un entorno contenerizado.

El principio de privilegios mínimos es esencial para los contenedores de Windows porque limita el posible impacto de las infracciones de seguridad, reduce la superficie de ataque e impide el acceso no autorizado a los componentes críticos del sistema. Al ejecutar aplicaciones en contenedores solo con los permisos necesarios, las organizaciones pueden mejorar considerablemente la seguridad y la estabilidad de sus entornos de contenedores, especialmente en infraestructuras compartidas y con varios usuarios.

Reflexiones finales: Por qué proteger sus contenedores de Windows es imprescindible en el panorama actual de amenazas

En el mundo digital actual, en rápida evolución, en el que las amenazas son cada vez más sofisticadas y abundantes, proteger sus contenedores de Windows no es solo una recomendación, sino una necesidad absoluta. Los contenedores proporcionan una forma ligera y flexible de empaquetar e implementar aplicaciones, pero no son inmunes a las vulnerabilidades de seguridad. A medida que más empresas adoptan los contenedores para optimizar su infraestructura, también se convierten en un objetivo potencial de ciberataques si no se protegen adecuadamente.

Internet está plagado de diversas amenazas, que van desde actores malintencionados que atacan vulnerabilidades no parcheadas hasta bots automatizados que buscan configuraciones incorrectas. Si no se implementan las medidas de seguridad adecuadas, los contenedores pueden explotarse para exponer datos confidenciales, aumentar los privilegios o servir como puntos de entrada para ataques que pueden poner en peligro su infraestructura en general. Esto hace que la seguridad de los contenedores sea tan importante como la protección de cualquier otra parte del entorno.

Al utilizar contenedores de Windows, se siguen aplicando muchas de las mejores prácticas de seguridad tradicionales. La implementación de políticas de cuentas sólidas, la protección de las configuraciones de IIS, el cumplimiento de HTTPS, el uso de reglas de firewall estrictas y la aplicación del acceso con privilegios mínimos a los archivos críticos son medidas clave para garantizar que el contenedor siga siendo resistente a los ataques. Además, las auditorías y el registro periódicos proporcionan visibilidad de lo que sucede dentro del contenedor, lo que le permite detectar cualquier actividad sospechosa antes de que se convierta en un incidente en toda regla.

La protección de los contenedores de Windows también se ajusta a los requisitos normativos que obligan a proteger los datos confidenciales y a garantizar la integridad de las aplicaciones. A medida que se generalicen las arquitecturas nativas de la nube y en contenedores, garantizar la seguridad en todos los niveles, desde la imagen base hasta el contenedor en ejecución, ayudará a proteger sus operaciones y a mantener la confianza de los clientes.

En resumen, el auge de las aplicaciones en contenedores, junto con el creciente número de ciberamenazas, hacen que la seguridad de los contenedores sea un aspecto innegociable de la administración de la infraestructura moderna. Al seguir las mejores prácticas y monitorear continuamente las vulnerabilidades, las empresas pueden disfrutar de la agilidad y la eficiencia de los contenedores de Windows sin comprometer la seguridad. En este entorno repleto de amenazas, proteger los contenedores de Windows no es solo una opción, sino una necesidad.

Prácticas recomendadas para implementaciones híbridas

Esta guía proporciona orientación sobre cómo ejecutar despliegues en entornos locales o periféricos con EKS Hybrid Nodes o EKS Anywhere.

Actualmente, hemos publicado guías sobre los siguientes temas:

- Mejores prácticas para los nodos híbridos de EKS y las desconexiones de red

Nodos híbridos EKS y desconexiones de red

La arquitectura de nodos híbridos de EKS puede resultar nueva para los clientes que están acostumbrados a ejecutar clústeres de Kubernetes locales por completo en sus propios centros de datos o ubicaciones periféricas. Con los nodos híbridos de EKS, el plano de control de Kubernetes se ejecuta en una región de AWS y solo los nodos se ejecutan de forma local, lo que da como resultado una arquitectura de clúster de Kubernetes «extendida» o «extendida».

Esto lleva a una pregunta común: «¿Qué ocurre si mis nodos se desconectan del plano de control de Kubernetes?»

En esta guía, respondemos a esa pregunta mediante una revisión de los siguientes temas. Se recomienda validar la estabilidad y la fiabilidad de las aplicaciones mediante la desconexión de la red, ya que cada aplicación puede comportarse de forma diferente en función de sus dependencias, configuración y entorno. Consulte el aws-samples/ eks-hybrid-examples GitHub repo para ver la configuración, los procedimientos y los resultados de las pruebas que puede consultar para probar las desconexiones de red con los nodos híbridos de EKS y sus propias aplicaciones. El GitHub repositorio también contiene detalles adicionales de las pruebas utilizadas para validar el comportamiento explicado en esta guía.

- Mejores prácticas para garantizar la estabilidad mediante las desconexiones de la red
- Comportamiento de conmutación por error del pod de Kubernetes mediante desconexiones de red
- Tráfico de red de aplicaciones a través de desconexiones de red
- Aloje las credenciales mediante desconexiones de red

Mejores prácticas para garantizar la estabilidad mediante las desconexiones de la red

Redes de alta disponibilidad

El mejor enfoque para evitar las desconexiones de red entre los nodos híbridos y el plano de control de Kubernetes es utilizar conexiones redundantes y resilientes desde el entorno local hacia y desde AWS. Consulte el kit de herramientas de resiliencia de AWS Direct Connect y la documentación de AWS Site-to-Site VPN para obtener más información sobre cómo diseñar redes híbridas de alta disponibilidad con esas soluciones.

Aplicaciones de alta disponibilidad

Al diseñar la arquitectura de las aplicaciones, tenga en cuenta los dominios de los fallos y los efectos de los distintos tipos de interrupciones. Kubernetes proporciona mecanismos integrados para implementar y mantener réplicas de aplicaciones en dominios de nodos, zonas y regiones. El uso de estos mecanismos depende de la arquitectura de la aplicación, los entornos y los requisitos de disponibilidad. Por ejemplo, las aplicaciones sin estado a menudo se pueden implementar con múltiples réplicas y se pueden mover entre hosts y capacidades de infraestructura arbitrarias, y puede usar selectores de nodos y restricciones de distribución de topología para ejecutar instancias de la aplicación en diferentes dominios. Para obtener más información sobre las técnicas a nivel de aplicación para crear aplicaciones resilientes en Kubernetes, consulte la Guía de mejores prácticas de EKS.

Kubernetes evalúa la información zonal de los nodos que están desconectados del plano de control de Kubernetes al determinar si se deben mover los pods a otros nodos. Si no se puede acceder a todos los nodos de una zona, Kubernetes cancela los desalojos de los nodos de esa zona. Como práctica recomendada, si tiene una implementación con nodos que se ejecutan en varios centros de datos o ubicaciones físicas, asigne una zona a cada nodo en función de su centro de datos o ubicación física. Cuando ejecuta EKS con nodos en la nube, AWS aplica automáticamente esta etiqueta de zona cloud-controller-manager. Sin embargo, a no cloud-controller-manager se usa con nodos híbridos, por lo que puede pasar esta información a través de su configuración de kubelet. A continuación, se muestra un ejemplo de cómo configurar una zona en la configuración de nodos para nodos híbridos. La configuración se transfiere al conectar los nodos híbridos al clúster con la CLI (`nodeadm`) de los nodos híbridos. Para obtener más información sobre la `topology.kubernetes.io/zone` etiqueta, consulta la documentación de Kubernetes. Para obtener más información sobre la CLI de los nodos híbridos, consulte la referencia nodeadm de los nodos híbridos.

```
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:
    ...
```

Monitorización de la red

Si utiliza AWS Direct Connect o AWS Site-to-Site VPN para su conectividad híbrida, puede aprovechar CloudWatch las alarmas, los registros y las métricas para observar el estado de la conexión híbrida y diagnosticar problemas. Para obtener más información, consulte Supervisión de los recursos de AWS Direct Connect y Supervisión de una conexión Site-to-Site VPN de AWS.

Se recomienda crear alarmas para `NodeNotReady` los eventos notificados por la node-lifecycle-controller ejecución en el plano de control del EKS, lo que indica que un nodo híbrido podría estar sufriendo una desconexión de la red. Para crear esta alarma, habilite el registro del plano de control EKS para el Controller Manager y cree un filtro métrico CloudWatch para el mensaje «Grabando un mensaje de evento de cambio de estado para el nodo» con el status= «». NodeNotReady Tras crear un filtro métrico, puede crear una alarma para este filtro en función de los umbrales que desee. Para obtener más información, consulte Alarmar los registros en la CloudWatch documentación.

Puede usar las métricas integradas Transit Gateway (TGW) y Virtual Private Gateway (VGW) para observar el tráfico de red que entra y sale de su TGW o VGW. Puede crear alarmas para estas métricas a fin de detectar situaciones en las que el tráfico de la red cae por debajo de los niveles normales, lo que indica un posible problema de red entre los nodos híbridos y el plano de control del EKS. Las métricas de TGW y VGW se describen en la siguiente tabla.

Puerta de enlace	Métrica	Description (Descripción)
Transit Gateway	BytesIn	Los bytes recibidos por TGW desde el archivo adjunto (del plano de control EKS a los nodos híbridos)

Puerta de enlace	Métrica	Description (Descripción)
Transit Gateway	BytesOut	Los bytes enviados desde TGW al adjunto (nodos híbridos al plano de control EKS)
Puerta de enlace privada virtual	TunnelDataIn	Los bytes enviados desde el lado de AWS de la conexión a través del túnel VPN hasta la puerta de enlace del cliente (del plano de control EKS a los nodos híbridos)
Puerta de enlace privada virtual	TunnelDataOut	Los bytes recibidos en el lado de AWS de la conexión a través del túnel VPN desde la puerta de enlace del cliente (nodos híbridos al plano de control de EKS)

También puede usar CloudWatch Network Monitor para obtener una visión más profunda de sus conexiones híbridas a fin de reducir el tiempo medio de recuperación y determinar si los problemas de red se originan en AWS o en su entorno. CloudWatch El monitor de red se puede utilizar para visualizar la pérdida de paquetes y la latencia en las conexiones de la red híbrida, establecer alertas y umbrales y, a continuación, tomar medidas para mejorar el rendimiento de la red. Para obtener más información, consulte Uso de Amazon CloudWatch Network Monitor.

EKS ofrece varias opciones para monitorear el estado de sus clústeres y aplicaciones. Para conocer el estado de los clústeres, puede utilizar el panel de observabilidad de la consola de EKS para detectar, solucionar y solucionar problemas rápidamente. También puede usar Amazon Managed Service para Prometheus, AWS Distro for Open Telemetry (ADOT) y para la supervisión de clústeres CloudWatch , aplicaciones e infraestructuras. Para obtener más información sobre las opciones de observabilidad de EKS, consulte Supervisar el rendimiento del clúster y ver los registros.

Solución de problemas locales

Para prepararse para las desconexiones de la red entre los nodos híbridos y el plano de control de EKS, puede configurar backends secundarios de monitoreo y registro para mantener la observabilidad de las aplicaciones cuando no se pueda acceder a los servicios regionales de AWS. Por ejemplo, puede configurar el recopilador AWS Distro for Open Telemetry (ADOT) para enviar métricas y registros a varios backends. También puedes usar herramientas locales, como la `crictl` CLI, para interactuar localmente con los pods y contenedores como reemplazo de otros clientes compatibles con la API de Kubernetes que normalmente consultan el punto final del servidor de la API de Kubernetes. `kubectl` Para obtener más información, consulte la documentación de cri-tools`crictl`. `crictl` GitHub A continuación se enumeran algunos `crictl` comandos útiles.

Enumere los pods que se ejecutan en el host:

```
crictl pods
```

Enumere los contenedores que se ejecutan en el host:

```
crictl ps
```

Enumere las imágenes que se ejecutan en el host:

```
crictl images
```

Obtenga los registros de un contenedor que se ejecuta en el host:

```
crictl logs CONTAINER_NAME
```

Obtenga estadísticas de los pods que se ejecutan en el host:

```
crictl statsp
```

Tráfico de red de aplicaciones

Al utilizar nodos híbridos, es importante tener en cuenta y comprender los flujos de red del tráfico de las aplicaciones y las tecnologías que se utilizan para exponer las aplicaciones de forma externa al clúster. Las diferentes tecnologías para el equilibrio de carga y la entrada de aplicaciones se comportan de forma diferente durante las desconexiones de la red. Por ejemplo, si utiliza la función del plano de control BGP de Cilium para equilibrar la carga de las aplicaciones, es posible que la sesión de BGP de sus pods y servicios se interrumpa durante las desconexiones de la red. Esto se debe a que la funcionalidad del altavoz BGP está integrada en el agente Cilium, y este se reiniciará continuamente cuando se desconecte del plano de control de Kubernetes. El motivo del reinicio se debe a que la comprobación del estado de Cilium ha fallado, ya que su estado va acompañado del acceso al plano de control de Kubernetes (consulte CFP: #31702, que incluye una mejora opcional en la versión 1.17 de Cilium). Del mismo modo, si utiliza balanceadores de carga de aplicaciones (ALB) o balanceadores de carga de red (NLB) para el tráfico de aplicaciones originado en la región de AWS, es posible que ese tráfico se interrumpa temporalmente si su entorno local pierde la conectividad con la región de AWS. Se recomienda comprobar que las tecnologías que utiliza para el

equilibrio de carga y la entrada permanezcan estables durante las desconexiones de la red antes de implementarlas en producción. El ejemplo del aws-samples/ eks-hybrid-examples GitHub repo utiliza MetallB para equilibrar la carga en el modo L2, que permanece estable durante las desconexiones de red entre los nodos híbridos y el plano de control de EKS.

Revise las dependencias de los servicios remotos de AWS

Cuando utilice nodos híbridos, tenga en cuenta las dependencias que asume con los servicios regionales de AWS que son externos a su entorno local o periférico. Algunos ejemplos incluyen el acceso a Amazon S3 o Amazon RDS para los datos de la aplicación, el uso de Amazon Managed Service for Prometheus CloudWatch o para obtener métricas y registros, el uso de balanceadores de carga de redes y aplicaciones para el tráfico originado en la región y la extracción de contenedores de Amazon Elastic Container Registry. No se podrá acceder a estos servicios durante las desconexiones de red entre su entorno local y AWS. Si su entorno local es propenso a desconectarse de la red con AWS, revise el uso que hace de los servicios de AWS y asegúrese de que la pérdida de la conexión a esos servicios no comprometa la estabilidad estática de sus aplicaciones.

Ajuste el comportamiento de conmutación por error del pod de Kubernetes

Existen opciones para ajustar el comportamiento de la conmutación por error de los pods durante las desconexiones de la red para las aplicaciones que no son portátiles entre los hosts o para los entornos con recursos limitados que no tienen capacidad sobrante para la conmutación por error de los pods. Por lo general, es importante tener en cuenta los requisitos de recursos de las aplicaciones y disponer de la capacidad suficiente para que una o más instancias de la aplicación se conmuten por error a un host diferente en caso de que se produzca un error en un nodo.

- Opción 1: Uso DaemonSets: esta opción se aplica a las aplicaciones que pueden y deben ejecutarse en todos los nodos del clúster. DaemonSets se configuran automáticamente para tolerar la contaminación inalcanzable, que mantiene a los DaemonSet pods conectados a sus nodos debido a las desconexiones de la red.
- Opción 2: Ajustarlos `tolerationSeconds` para evitar que se vean contaminados: puedes ajustar el tiempo que los pods permanecen conectados a los nodos durante las desconexiones de la red. Para ello, configure los pods de aplicaciones para que toleren la contaminación inalcanzable con el `NoExecute` efecto que especifique (en las especificaciones de la aplicación). `tolerationSeconds` Con esta opción, cuando se producen desconexiones de red, los pods de aplicaciones permanecen enlazados a los nodos hasta que caduquen. `tolerationSeconds` Tenga esto en cuenta, ya que si se aumenta la contaminación `tolerationSeconds` por lo

inalcanzable, los pods que `NoExecute` se ejecutan en hosts inalcanzables pueden tardar más en trasladarse a otros hosts accesibles y en buen estado.

- Opción 3: Controlador personalizado: puedes crear y ejecutar un controlador personalizado (u otro software) que supervise Kubernetes en busca de la contaminación inalcanzable que provoca este efecto. `NoExecute` Cuando se detecta esta contaminación, el controlador personalizado puede comprobar las métricas específicas de la aplicación para evaluar su estado. Si la aplicación está en buen estado, el controlador personalizado puede eliminar la contaminación inalcanzable e impedir que se desalojen los pods de los nodos durante las desconexiones de la red.

A continuación, se muestra un ejemplo de cómo configurar una implementación con `tolerationSeconds` una contaminación inalcanzable. En el ejemplo, `tolerationSeconds` está configurado en `1800` (30 minutos), lo que significa que los pods que se ejecuten en nodos inalcanzables solo se desalojarán si la desconexión de la red dura más de 30 minutos.

```
apiVersion: apps/v1
kind: Deployment
metadata:
...
spec:
...
      tolerations:
      - key: "node.kubernetes.io/unreachable"
        operator: "Exists"
        effect: "NoExecute"
        tolerationSeconds: 1800
```

Conmutación por error de los pods de Kubernetes mediante desconexiones de red

Comenzamos con una revisión de los conceptos, componentes y configuraciones clave que influyen en el comportamiento de Kubernetes durante las desconexiones de red entre los nodos y el plano de control de Kubernetes. En principio, EKS se ajusta a Kubernetes, por lo que todos los conceptos, componentes y configuraciones de Kubernetes que se describen aquí se aplican a las implementaciones de EKS y EKS Hybrid Nodes.

Se han introducido mejoras en EKS específicamente para mejorar el comportamiento de conmutación por error de los pods durante las desconexiones de la red. Para obtener más información, consulte los números #131294 y #131481 del repositorio original de Kubernetes. GitHub

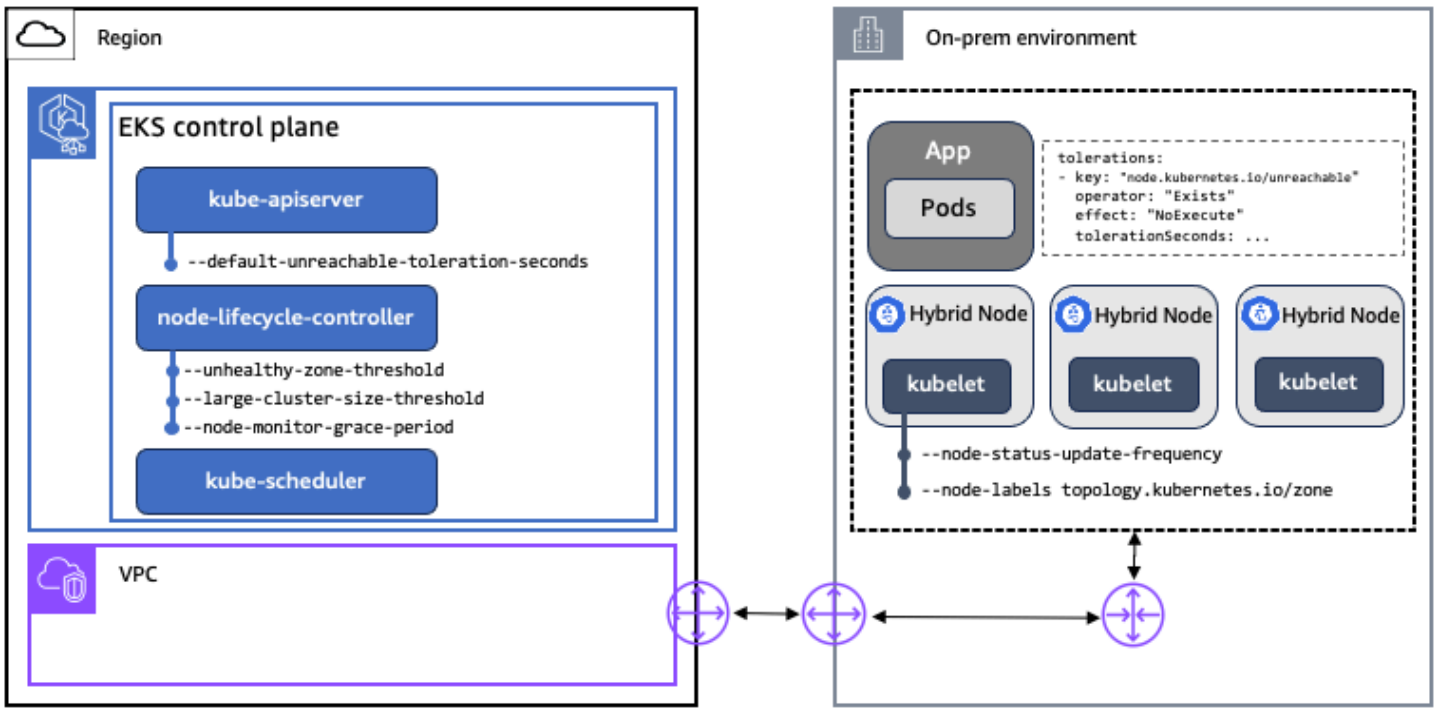
Conceptos

Contaminaciones y tolerancias: en Kubernetes se utilizan las restricciones y tolerancias para controlar la programación de los pods en los nodos. Los límites se establecen node-lifecycle-controller para indicar que los nodos no son aptos para la programación o que se deben desalojar los módulos de esos nodos. Cuando no se puede acceder a los nodos debido a una desconexión de la red, se aplica el comando 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 NodeCondition corresponde a que Ready being Unknown. Los usuarios pueden especificar las tolerancias de contaminación a nivel de aplicación en el. PodSpec

- NoSchedule: No hay nuevos pods programados en el nodo contaminado a menos que tengan una tolerancia coincidente. Los pods que ya se están ejecutando en el nodo no se desalojan.
- NoExecute: Las cápsulas que no toleran la contaminación se desalojan inmediatamente. Las cápsulas que toleran la contaminación (sin especificar los segundos de tolerancia) permanecen atadas para siempre. Los pods que toleran la contaminación con un TolerationSeconds específico permanecen enlazados durante el tiempo especificado. Una vez transcurrido ese tiempo, el controlador del ciclo de vida del nodo expulsa los pods del nodo.

Arrendamientos de nodos: Kubernetes usa la API de arrendamiento para comunicar los latidos de los nodos de Kubelet al servidor de la API de Kubernetes. Para cada nodo, hay un objeto de arrendamiento con un nombre coincidente. Internamente, cada latido del kubelet actualiza el campo spec.RenewTime del objeto Lease. El plano de control de Kubernetes usa la marca de tiempo de este campo para determinar la disponibilidad de los nodos. Si los nodos están desconectados del plano de control de Kubernetes, no pueden actualizar Spec.RenewTime para su arrendamiento, y el plano de control interpreta que está listo para ser desconocido. NodeCondition

Componentes



Componente	Subcomponente	Description (Descripción)
Plano de control de Kubernetes	kube-api-server	El servidor de API es un componente central del plano de control de Kubernetes que expone la API de Kubernetes.
Plano de control de Kubernetes	node-lifecycle-controller	Uno de los controladores que ejecuta. kube-controller-manager Es responsable de detectar y responder a los problemas de los nodos.
Plano de control de Kubernetes	programador de kube	Un componente del plano de control que observa los pods recién creados sin un nodo asignado y selecciona un nodo en el que ejecutarlos.
nodos de Kubernetes	kubelet	Un agente que se ejecuta en cada nodo del clúster. El kubelet vigila PodSpecs y se asegura de que los contenedores descritos en ellas PodSpecs estén funcionando y en buen estado.

Opciones de configuración

Compor e	Opción	Description (Descripción)	K8 es el predete inado	Predete inado por EKS	Configura ble en EKS
kube-api-server	default-unreachable-toleration-seconds	Indica `tolerationSeconds` si la tolerancia `unreachable:NoExecute` que se añade de forma predeterminada a todos los módulos que aún no tienen dicha tolerancia.	300	300	No
node-lifecycle-controller	node-monitor-grace-period	El tiempo que un nodo puede estar inactivo antes de que se marque como no está en buen estado. Debe ser N veces mayor que el de kubeletnodeStatusUpdateFrequency , donde N es el número de reintentos permitidos para que el kubelet publique el estado de nodo.	40	40	No
node-lifecycle-controller	large-cluster-size-threshold	El número de nodos en los que considera que el clúster es grande node-lifecycle-controller para la lógica de desalojo. `--secondary-node-eviction-rate` se sustituye por 0 para los clústeres de este tamaño o más pequeños.	50	100 000	No
node-lifecycle-controller	unhealthy-zone-threshold	El porcentaje de nodos de una zona que no deben estar preparados para que esa zona se considere en mal estado.	55%	55%	No
kubelet	node-status-update-frequency	Con qué frecuencia el kubelet publica el estado del nodo en el plano de control. Debe ser compatible con nodeMonit	10	10	Sí

Compor e	Opción	Description (Descripción)	K8 es el predete inado	Predete inado por EKS	Configura ble en EKS
		`orGracePeriod` in. node-lifecycle-con troller			
kubelet	etiquetas de nodo	Etiquetas para añadir al registrar el nodo en el clúster. La etiqueta se `topology.kubernetes.io/zone` puede especificar con nodos híbridos para agrupar los nodos en zonas.	Ningunc	Ningunc	Sí

Conmutación por error del pod de Kubernetes mediante desconexiones de red

El comportamiento que se describe aquí supone que los pods se ejecutan como despliegues de Kubernetes con la configuración predeterminada y que EKS se utiliza como proveedor de Kubernetes. El comportamiento real puede variar según el entorno, el tipo de desconexión de la red, las aplicaciones, las dependencias y la configuración del clúster. El contenido de esta guía se validó mediante una aplicación, una configuración de clúster y un subconjunto de complementos específicos. Se recomienda encarecidamente probar el comportamiento en su propio entorno y con sus propias aplicaciones antes de pasar a la fase de producción.

Cuando hay desconexiones de red entre los nodos y el plano de control de Kubernetes, el kubelet de cada nodo desconectado no puede comunicarse con el plano de control de Kubernetes. En consecuencia, el kubelet no puede desalojar los pods de esos nodos hasta que se restablezca la conexión. Esto significa que los pods que se ejecutaban en esos nodos antes de la desconexión de la red seguirán funcionando durante la desconexión, siempre que no haya otros fallos que provoquen su cierre. En resumen, se puede lograr una estabilidad estática durante las desconexiones de la red entre los nodos y el plano de control de Kubernetes, pero no se pueden realizar operaciones de mutación en los nodos o las cargas de trabajo hasta que se restablezca la conexión.

Existen cinco escenarios principales que producen diferentes comportamientos de conmutación por error entre los módulos en función de la naturaleza de la desconexión de la red. En todos los escenarios, el clúster vuelve a estar en buen estado sin la intervención del operador una vez que los nodos se vuelven a conectar al plano de control de Kubernetes. Los siguientes escenarios

describen los resultados esperados en función de nuestras observaciones, pero es posible que estos resultados no se apliquen a todas las configuraciones posibles de aplicaciones y clústeres.

Escenario 1: Interrupción total del clúster

Resultado esperado: los pods de los nodos inalcanzables no se expulsan y siguen ejecutándose en esos nodos.

Una interrupción total del clúster significa que todos los nodos del clúster se desconectan del plano de control de Kubernetes. En este escenario, el node-lifecycle-controller plano de control detecta que no se puede acceder a todos los nodos del clúster y cancela cualquier desalojo de un módulo.

Los administradores del clúster verán todos los nodos con su estado `Not Ready` durante la desconexión. El estado de los pods no cambia y no se programan nuevos pods en ningún nodo durante la desconexión y la posterior reconexión.

Escenario 2: Interrupción total de la zona

Resultado esperado: los pods de los nodos inalcanzables no se expulsan y siguen ejecutándose en esos nodos.

Una interrupción total de la zona significa que todos los nodos de la zona están desconectados del plano de control de Kubernetes. En este escenario, el node-lifecycle-controller plano de control detecta que no se puede acceder a todos los nodos de la zona y cancela cualquier desalojo de un módulo.

Los administradores del clúster verán todos los nodos con su estado `Not Ready` durante la desconexión. El estado de los pods no cambia y no se programan nuevos pods en ningún nodo durante la desconexión y la posterior reconexión.

Escenario 3: Interrupción de la zona mayoritaria

Resultado esperado: los pods de los nodos inalcanzables no se expulsan y siguen ejecutándose en esos nodos.

Una interrupción mayoritaria de una zona significa que la mayoría de los nodos de una zona determinada están desconectados del plano de control de Kubernetes. Las zonas de Kubernetes se definen mediante nodos con la misma etiqueta. `topology.kubernetes.io/zone` Si no hay ninguna zona definida en el clúster, una interrupción mayoritaria significa que la mayoría de los nodos de todo el clúster están desconectados. De forma predeterminada, la mayoría se define por

sunhealthy-zone-threshold, que se establece en un 55% tanto en Kubernetes como en EKS. node-lifecycle-controller Como large-cluster-size-threshold está establecido en 100 000 en EKS, si no se puede acceder al 55% o más de los nodos de una zona, se cancelan los desalojos de los pods (dado que la mayoría de los clústeres son mucho más pequeños que 100 000 nodos).

Los administradores de clústeres verán la mayoría de los nodos de la zona en estado Not Ready durante la desconexión, pero el estado de los pods no cambiará y no se reprogramarán en otros nodos.

Tenga en cuenta que el comportamiento anterior solo se aplica a los clústeres de más de tres nodos. En los clústeres de tres nodos o menos, se programa el desalojo de los módulos de los nodos inalcanzables y se programa el desalojo de los nuevos módulos de los nodos en buen estado.

Durante las pruebas, en ocasiones observamos que los pods eran expulsados exactamente de un nodo inalcanzable durante las desconexiones de la red, incluso cuando no se podía acceder a la mayoría de los nodos de la zona. Seguimos investigando una posible condición de carrera en Kubernetes como causa de este comportamiento node-lifecycle-controller.

Escenario 4: Interrupción en una zona minoritaria

Resultado esperado: se expulsan los pods de los nodos inalcanzables y se programan nuevos pods en los nodos disponibles y aptos.

Una interrupción minoritaria significa que un porcentaje menor de nodos de una zona están desconectados del plano de control de Kubernetes. Si no hay ninguna zona definida en el clúster, una interrupción minoritaria significa que la minoría de nodos de todo el clúster está desconectada. Como se ha indicado, la minoría se define mediante el unhealthy-zone-threshold ajuste de node-lifecycle-controller, que es del 55% de forma predeterminada. En este escenario, si la desconexión de la red dura más de default-unreachable-toleration-seconds (5 minutos) y node-monitor-grace-period (40 segundos) y no se puede acceder a menos del 55% de los nodos de una zona, se programan nuevos pods en los nodos en buen estado, mientras que los pods de los nodos inalcanzables se marcan para su desalojo.

Los administradores de clústeres verán los nuevos pods creados en los nodos en buen estado y los pods de los nodos desconectados se mostrarán como. Terminating Recuerde que, aunque los pods de los nodos desconectados tengan un Terminating estado, no se desalojarán por completo hasta que el nodo se vuelva a conectar al plano de control de Kubernetes.

Escenario 5: Reinicio del nodo durante una interrupción de la red

Resultado esperado: los pods de los nodos inalcanzables no se inician hasta que los nodos se vuelven a conectar al plano de control de Kubernetes. La conmutación por error de los pods sigue la lógica descrita en los escenarios 1 a 3, en función del número de nodos inalcanzables.

El reinicio de un nodo durante una interrupción de la red significa que se ha producido otro fallo (como un ciclo de alimentación, un out-of-memory evento u otro problema) en un nodo al mismo tiempo que se desconecta la red. Los pods que se estaban ejecutando en ese nodo cuando se inició la desconexión de la red no se reinician automáticamente durante la desconexión si el kubelet también se ha reiniciado. El kubelet consulta al servidor de la API de Kubernetes durante el inicio para saber qué pods debe ejecutar. Si el kubelet no puede acceder al servidor de la API debido a una desconexión de la red, no podrá recuperar la información necesaria para iniciar los pods.

En este escenario, las herramientas de solución de problemas locales, como la `crictl` CLI, no se pueden utilizar para iniciar los pods manualmente como medida de «romper cristales». Por lo general, Kubernetes elimina los pods que fallan y crea otros nuevos en lugar de reiniciar los pods existentes (consulta #10213 en el repositorio containerd para obtener más información). GitHub Los pods estáticos son el único objeto de carga de trabajo de Kubernetes que controla el kubelet y que se pueden reiniciar en estos escenarios. Sin embargo, por lo general no se recomienda utilizar pods estáticos para las implementaciones de aplicaciones. En su lugar, despliegue varias réplicas en distintos hosts para garantizar la disponibilidad de las aplicaciones en caso de que se produzcan varios fallos simultáneos, como un fallo en un nodo o una desconexión de la red entre los nodos y el plano de control de Kubernetes.

Tráfico de red de aplicaciones a través de desconexiones de red

Los temas de esta página están relacionados con las redes de clústeres de Kubernetes y el tráfico de aplicaciones durante las desconexiones de red entre los nodos y el plano de control de Kubernetes.

Cilium

Cilium tiene varios modos de administración de direcciones IP (IPAM), encapsulación, equilibrio de carga y enrutamiento de clústeres. Los modos validados en esta guía utilizaban el IPAM con alcance de clúster, la superposición de VXLAN, el equilibrio de carga de BGP y el kube-proxy. También se utilizó Cilium sin balanceo de carga BGP, sustituyéndolo por el balanceo de carga MetalLB L2.

La base de la instalación Cilium está formada por el operador Cilium y los agentes Cilium. El operador de Cilium se ejecuta como una implementación y registra las definiciones de recursos

personalizadas de Cilium (CRDs), administra el IPAM y sincroniza los objetos del clúster con el servidor API de Kubernetes, entre otras funciones. Los agentes de Cilium se ejecutan en cada nodo como un programa eBPF DaemonSet y los administran para controlar las reglas de red para las cargas de trabajo que se ejecutan en el clúster.

Por lo general, el enrutamiento dentro del clúster configurado por Cilium permanece disponible y activo durante las desconexiones de la red, lo que se puede confirmar observando los flujos de tráfico del clúster y las reglas de la tabla de direcciones IP (iptables) de la red de módulos.

```
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
...
```

Sin embargo, durante las desconexiones de la red, el operador de Cilium y los agentes de Cilium se reinician debido a la combinación de sus comprobaciones de estado con el estado de la conexión con el servidor API de Kubernetes. Se espera que aparezca lo siguiente en los registros del operador de Cilium y de los agentes de Cilium durante las desconexiones de la red. Durante las desconexiones de la red, puede utilizar herramientas como la `crictl` CLI para observar los reinicios de estos componentes, incluidos sus registros.

```
msg="Started gops server" address="127.0.0.1:9890" subsys=gops
msg="Establishing connection to apiserver" host="https://<k8s-cluster-ip>:443"
 subsys=k8s-client
msg="Establishing connection to apiserver" host="https://<k8s-cluster-ip>:443"
 subsys=k8s-client
msg="Unable to contact k8s api-server" error="Get \"https://<k8s-cluster-ip>:443/
api/v1/namespaces/kube-system\": dial tcp <k8s-cluster-ip>:443: i/o timeout"
 ipAddr="https://<k8s-cluster-ip>:443" subsys=k8s-client
msg="Start hook failed" function="client.(*compositeClientset).onStart
 (agent.infra.k8s-client)" error="Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/
kube-system\": dial tcp <k8s-cluster-ip>:443: i/o timeout"
msg="Start failed" error="Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/kube-
system\": dial tcp <k8s-cluster-ip>:443: i/o timeout" duration=1m5.003834026s
msg=Stopping
```

```
msg="Stopped gops server" address="127.0.0.1:9890" subsys=gops
msg="failed to start: Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/kube-system
\": dial tcp <k8s-cluster-ip>:443: i/o timeout" subsys=daemon
```

Si utiliza la función del plano de control BGP de Cilium para equilibrar la carga de las aplicaciones, es posible que la sesión de BGP de sus pods y servicios se interrumpa durante las desconexiones de la red, ya que la funcionalidad de los altavoces BGP está integrada en el agente de Cilium, y el agente de Cilium se reiniciará continuamente cuando se desconecte del plano de control de Kubernetes. Para obtener más información, consulte la Guía de funcionamiento del plano de control BGP de Cilium en la documentación de Cilium. Además, si se produce un fallo simultáneo durante una desconexión de la red, como un ciclo de alimentación o un reinicio de la máquina, las rutas de Cilium no se conservarán mediante estas acciones, aunque se volverán a crear cuando el nodo se vuelva a conectar al plano de control de Kubernetes y Cilium se vuelva a iniciar.

Calico

Próximamente

Metal LB

MetalLB tiene dos modos de equilibrio de carga: modo L2 y modo BGP. Consulte la documentación de MetalLB para obtener detalles sobre cómo funcionan estos modos de equilibrio de carga y sus limitaciones. En la validación de esta guía se utilizó MetallB en el modo L2, en el que una máquina del clúster asume la propiedad del servicio de Kubernetes y utiliza ARP para hacer que las direcciones IP del balanceador de carga estén IPv4 accesibles en la red local. Cuando se ejecuta MetallB, hay un controlador que se encarga de la asignación de IP y altavoces en cada nodo que se encargan de anunciar los servicios con direcciones IP asignadas. El controlador MetalLB funciona como un Deployment y los altavoces MetalLB funcionan como un. DaemonSet Durante las desconexiones de la red, el controlador MetalLB y los altavoces no vigilan el servidor API de Kubernetes en busca de recursos del clúster, pero siguen funcionando. Y lo que es más importante, los servicios que utilizan MetallB para la conectividad externa permanecen disponibles y accesibles durante las desconexiones de la red.

kube-proxy

En los clústeres de EKS, kube-proxy se ejecuta DaemonSet en cada nodo y es responsable de gestionar las reglas de red para permitir la comunicación entre los servicios y los pods mediante la traducción de las direcciones IP de los servicios a las direcciones IP de los pods subyacentes.

Las reglas de las tablas de IP (iptables) configuradas por kube-proxy se mantienen durante las desconexiones de la red, el enrutamiento dentro del clúster sigue funcionando y los pods de kube-proxy siguen funcionando.

Puedes observar las reglas de kube-proxy con los siguientes comandos de iptables. El primer comando muestra que los paquetes que atraviesan la `PREROUTING` cadena se dirigen a la cadena. `KUBE-SERVICES`

```
iptables -t nat -L PREROUTING
```

```
Chain PREROUTING (policy ACCEPT)
target         prot opt source      destination
KUBE-SERVICES  all  --  anywhere    anywhere      /* kubernetes service portals */
```

Al inspeccionar la `KUBE-SERVICES` cadena, podemos ver las reglas de los distintos servicios del clúster.

```
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 */
```

Al inspeccionar la cadena del servicio de interfaz de la aplicación, podemos ver las direcciones IP de los módulos que respaldan el servicio.

```
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 */
```

Se esperan los siguientes mensajes de registro de kube-proxy durante las desconexiones de la red, ya que intenta vigilar el servidor API de Kubernetes en busca de actualizaciones en los recursos de los nodos y terminales.

```
"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

De forma predeterminada, los pods de los clústeres de EKS utilizan la dirección IP del clúster de CoreDNS como servidor de nombres para las consultas de DNS dentro del clúster. En los clústeres de EKS, CoredNS se ejecuta como una implementación en los nodos. Con los nodos híbridos, los pods pueden seguir comunicándose con el CoreDNS durante las desconexiones de la red cuando hay réplicas de CoreDNS ejecutándose localmente en los nodos híbridos. Si tiene un clúster de EKS con nodos en la nube y nodos híbridos en su entorno local, se recomienda tener al menos una réplica de CoredNS en cada entorno. CoredNS sigue atendiendo consultas de DNS para los registros

que se crearon antes de la desconexión de la red y continúa ejecutándose durante la reconexión de la red para garantizar una estabilidad estática.

Se esperan los siguientes mensajes de registro de CoredNS durante las desconexiones de la red, ya que intenta enumerar los objetos del servidor API de Kubernetes.

```
Failed to watch *v1.Namespace: failed to list *v1.Namespace: Get "https://<k8s-cluster-
ip>:443/api/v1/namespaces?resourceVersion=2263964": dial tcp <k8s-cluster-ip>:443: i/o
 timeout
Failed to watch *v1.Service: failed to list *v1.Service: Get "https://<k8s-cluster-
ip>:443/api/v1/services?resourceVersion=2263966": dial tcp <k8s-cluster-ip>:443: i/o
 timeout
Failed to watch *v1.EndpointSlice: failed to list *v1.EndpointSlice: Get "https://<k8s-
cluster-ip>:443/apis/discovery.k8s.io/v1/endpointslices?resourceVersion=2263896": dial
 tcp <k8s-cluster-ip>: i/o timeout
```

Aloje las credenciales mediante desconexiones de red

EKS Hybrid Nodes se integra con las activaciones híbridas de AWS Systems Manager (SSM) y con AWS IAM Roles Anywhere para obtener credenciales de IAM temporales que se utilizan para autenticar el nodo con el plano de control de EKS. Tanto SSM como IAM Roles Anywhere actualizan automáticamente las credenciales temporales que administran en los hosts locales. Se recomienda utilizar un único proveedor de credenciales en todos los nodos híbridos del clúster: SSM Hybrid Activations o IAM Roles Anywhere, pero no ambos.

Activaciones híbridas de SSM

Las credenciales temporales proporcionadas por SSM son válidas durante una hora. No puede modificar la duración de la validez de las credenciales si utiliza SSM como proveedor de credenciales. SSM rota automáticamente las credenciales temporales antes de que caduquen y la rotación no afecta al estado de los nodos o las aplicaciones. Sin embargo, cuando hay desconexiones de red entre el agente de SSM y el punto de conexión regional de SSM, SSM no puede actualizar las credenciales y estas podrían caducar.

SSM utiliza un retardo exponencial para los reintentos de actualización de credenciales si no puede conectarse a los puntos finales regionales de SSM. En la versión del agente SSM `3.3.808.0` y versiones posteriores (publicadas en agosto de 2024), el retraso exponencial está limitado a 30 minutos. Según la duración de la desconexión de la red, SSM puede tardar hasta 30 minutos en actualizar las credenciales y los nodos híbridos no se volverán a conectar al plano de control de EKS

hasta que se actualicen las credenciales. En este escenario, puede reiniciar el agente SSM para forzar la actualización de las credenciales. Como efecto secundario del comportamiento actual de actualización de credenciales de SSM, es posible que los nodos se vuelvan a conectar en distintos momentos, según el momento en que el agente de SSM de cada nodo consiga actualizar sus credenciales. Por este motivo, es posible que se produzca una conmutación por error entre los nodos que aún no se han vuelto a conectar y los nodos que ya se han vuelto a conectar.

Obtenga la versión del agente SSM. También puedes consultar la sección Fleet Manager de la consola SSM:

```
# AL2023, RHEL
yum info amazon-ssm-agent
# Ubuntu
snap list amazon-ssm-agent
```

Reinicie el agente SSM:

```
# AL2023, RHEL
systemctl restart amazon-ssm-agent
# Ubuntu
systemctl restart snap.amazon-ssm-agent.amazon-ssm-agent
```

Vea los registros del agente SSM:

```
tail -f /var/log/amazon/ssm/amazon-ssm-agent.log
```

Mensajes de registro esperados durante las desconexiones de la red:

```
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

De forma predeterminada, las credenciales temporales proporcionadas por IAM Roles Anywhere son válidas durante una hora. Puede configurar la duración de la validez de las credenciales con IAM Roles Anywhere a través del `durationSeconds`campo de su perfil de IAM Roles Anywhere. La validez máxima de las credenciales es de 12 horas. La `MaxSessionDuration`configuración de su función de IAM de Hybrid Nodes debe ser mayor que la `durationSeconds` configuración de su perfil de IAM Roles Anywhere.

Al utilizar IAM Roles Anywhere como proveedor de credenciales para los nodos híbridos, la reconexión al plano de control del EKS tras la desconexión de la red suele producirse segundos después de la restauración de la red, ya que el kubelet llama para obtener las credenciales a pedido. `aws_signing_helper credential-process` Aunque no está directamente relacionado con los nodos híbridos o las desconexiones de la red, puede configurar notificaciones y alertas para el vencimiento de los certificados cuando utilice IAM Roles Anywhere. Para obtener más información, consulte Personalizar la configuración de notificaciones en IAM Roles Anywhere.

Mejores prácticas para ejecutar cargas AI/ML de trabajo

> **Tip**
>
> Visite Get Hands on with EKS para obtener información sobre los próximos AI/ML eventos y talleres de Amazon EKS.

La implementación de las mejores prácticas al ejecutar AI/ML cargas de trabajo en EKS puede garantizar que esas cargas de trabajo sean eficientes, rentables, resilientes y cuenten con los recursos adecuados. Las mejores prácticas se dividen en las siguientes secciones generales: computación, redes, almacenamiento, observabilidad y rendimiento.

Comentarios

Esta guía se publica el día de GitHub hoy para recopilar comentarios y sugerencias directos de la EKS/Kubernetes comunidad en general. Si tienes alguna buena práctica que consideres que deberíamos incluir en la guía, presenta un problema o envía un PR al GitHub repositorio. Nuestra intención es actualizar la guía periódicamente a medida que se agreguen nuevas funciones al servicio o cuando surja una nueva mejor práctica.

Cómputo y escalado automático

Optimización de recursos de GPU y gestión de costes

Programe las cargas de trabajo con los requisitos de la GPU mediante etiquetas conocidas

Para AI/ML las cargas de trabajo sensibles a diferentes características de la GPU (p. ej., GPU o memoria de la GPU), recomendamos especificar los requisitos de la GPU mediante etiquetas de programación conocidas compatibles con los tipos de nodos utilizados con Karpenter y los grupos de nodos gestionados. Si no se definen, es posible que los pods se programen en instancias con recursos de GPU inadecuados, lo que puede provocar fallos o reducir el rendimiento. Recomendamos usar NodeSelector o Node Affinity para especificar en qué nodo debe ejecutarse un pod y configurar los recursos de cómputo (CPU, memoria, GPUs etc.) en la sección de recursos del pod.

Ejemplo

Por ejemplo, usar el selector de nodos con el nombre de la GPU cuando se usa Karpenter:

```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod-example
spec:
  containers:
  - name: ml-workload
    image: <image>
    resources:
      limits:
        nvidia.com/gpu: 1  # Request one NVIDIA GPU
  nodeSelector:
    karpenter.k8s.aws/instance-gpu-name: "l40s"  # Run on nodes with NVIDIA L40S GPUs
```

Usa el complemento de dispositivo de Kubernetes para exponer GPUs

Para exponer GPUs en los nodos, el controlador de la GPU NVIDIA debe estar instalado en el sistema operativo del nodo y el tiempo de ejecución del contenedor debe estar configurado para permitir que el programador de Kubernetes asigne los pods a los nodos que estén disponibles. GPUs El proceso de configuración del complemento para dispositivos NVIDIA Kubernetes depende de la AMI acelerada de EKS que utilice:

- AMI acelerada de Bottlerocket: esta AMI incluye el controlador de GPU NVIDIA y el complemento para dispositivos NVIDIA Kubernetes viene preinstalado y listo para usarse, lo que permite la compatibilidad con GPU desde el primer momento. No se requiere ninguna configuración adicional para exponerla al programador de Kubernetes. GPUs
- AL2023 AMI acelerada: esta AMI incluye el controlador de GPU NVIDIA, pero el complemento para dispositivos NVIDIA Kubernetes no viene preinstalado. Debe instalar y configurar el complemento del dispositivo por separado, normalmente mediante un. DaemonSet Ten en cuenta que si utilizas eksctl para crear tu clúster y especificas un tipo de instancia de GPU (por ejemplo`g5.xlarge`) en el tuyo, `eksctl` seleccionarás automáticamente la AMI adecuada e instalarás el complemento de dispositivo NVIDIA Kubernetes. ClusterConfig Para obtener más información, consulta la compatibilidad con GPU en la documentación de eksctl.

Si prefieres utilizar EKS Accelerated AMIs y el operador de GPU NVIDIA para gestionar componentes como el complemento para dispositivos NVIDIA Kubernetes, toma nota de deshabilitar la administración del controlador de GPU de NVIDIA y del kit de herramientas NVIDIA Container, tal y como se indica en la documentación de NVIDIA sobre los controladores de GPU NVIDIA preinstalados y el kit de herramientas NVIDIA Container Toolkit.

Para comprobar que el complemento para dispositivos NVIDIA está activo y GPUs correctamente expuesto, ejecuta:

```
kubectl describe node | grep nvidia.com/gpu
```

Este comando comprueba si el `nvidia.com/gpu` recurso está dentro de la capacidad y los recursos asignables del nodo. Por ejemplo, debería mostrarse un nodo con una GPU. `nvidia.com/gpu: 1` Consulta la Guía de programación de GPU de Kubernetes para obtener más información.

Usa muchos tipos de instancias diferentes EC2

El uso de tantos tipos de EC2 instancias diferentes como sea posible es una práctica recomendada importante para la escalabilidad en Amazon EKS, como se describe en la the section called "Plano de datos" sección. Esta recomendación también se aplica a las instancias con hardware acelerado (por ejemplo, GPUs). Si creas un clúster que usa solo un tipo de instancia e intentas escalar el número de nodos más allá de la capacidad de la región, es posible que recibas un error de capacidad insuficiente (ICE), lo que indica que no hay instancias disponibles. Es importante comprender las características únicas de sus AI/ML cargas de trabajo antes de diversificarlas arbitrariamente. Revisa los tipos de instancias disponibles con la herramienta EC2 Instance Type Explorer para generar una lista de tipos de instancias que coincidan con tus requisitos informáticos específicos y evita limitar arbitrariamente el tipo de instancias que se pueden usar en tu clúster.

Las instancias de computación acelerada se ofrecen en diferentes modelos de compra para adaptarse a cargas de trabajo estables, a corto y medio plazo. Para cargas de trabajo a corto plazo, flexibles y tolerantes a errores, en las que no quiera tener que hacer reservas, busque las instancias puntuales. Los bloques de capacidad, las instancias bajo demanda y los planes de ahorro le permiten aprovisionar instancias de cómputo acelerado para una carga de trabajo a medio y largo plazo. Para aumentar las posibilidades de acceder correctamente a la capacidad requerida en la opción de compra que prefiera, se recomienda utilizar una lista variada de tipos de instancias y zonas de disponibilidad. Como alternativa, si encuentra un modelo ICEs de compra específico, vuelva a intentarlo con un modelo diferente.

Ejemplo El siguiente ejemplo muestra cómo permitir que un Karpenter aprovisione instancias G y P superiores NodePool a las generaciones 3 (por ejemplo, p3). Para obtener más información, consulte la sección Escalabilidad.

```
- 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
```

Para obtener más información sobre el uso de instancias puntuales para GPUs, consulte «Considere usar instancias EC2 puntuales de Amazon GPUs con Karpenter» a continuación.

Considere la posibilidad de utilizar Amazon EC2 Spot Instances GPUs con Karpenter

Las Amazon EC2 Spot Instances le permiten aprovechar la EC2 capacidad no utilizada en la nube de AWS y están disponibles con hasta un 90% de descuento en comparación con los precios bajo demanda. Las instancias EC2 puntuales de Amazon se pueden interrumpir con un aviso de dos minutos cuando se EC2 necesita recuperar la capacidad. Para obtener más información, consulte Spot Instances en la Guía del EC2 usuario de Amazon. Amazon EC2 Spot puede ser una excelente opción para cargas de trabajo tolerantes a errores, sin estado y flexibles (tiempo y tipo de instancia). Para obtener más información sobre cuándo usar las instancias puntuales, consulte las prácticas recomendadas sobre las instancias EC2 puntuales. También puede usar instancias puntuales para AI/ML cargas de trabajo si son compatibles con Spot.

Casos de uso

Las cargas de trabajo compatibles con la localización pueden ser los macrodatos, las cargas de trabajo en contenedores, la CI/CD, los servidores web sin estado, la informática de alto rendimiento (HPC) y las cargas de trabajo de renderizado. Las instancias puntuales no son adecuadas para cargas de trabajo inflexibles, activas, intolerantes a errores o estrechamente acopladas entre nodos de instancia (por ejemplo, cargas de trabajo con procesos paralelos que dependen en gran medida unos de otros para realizar cálculos y que requieren una comunicación constante entre nodos, como las aplicaciones informáticas de alto rendimiento basadas en MPI, como la dinámica de fluidos computacional o las bases de datos distribuidas con interdependencias complejas). Estos son los casos de uso específicos que recomendamos (sin ningún orden en particular):

- Inferencia online en tiempo real: utilice instancias puntuales para optimizar los costes de sus cargas de trabajo de inferencia en tiempo real, siempre que sus cargas de trabajo sean aptas

para aplicaciones puntuales. En otras palabras, el tiempo de inferencia es inferior a dos minutos, la aplicación es tolerante a las interrupciones a los fallos y puede ejecutarse en distintos tipos de instancias. Garantice una alta disponibilidad mediante la diversidad de instancias (por ejemplo, entre varios tipos de instancias y zonas de disponibilidad) o mediante reservas, a la vez que implementa la tolerancia a errores a nivel de la aplicación para gestionar posibles interrupciones puntuales.

- Ajuste de hiperparámetros: utilice instancias puntuales para ejecutar trabajos de ajuste exploratorios de forma oportunista, ya que las interrupciones se pueden tolerar sin pérdidas significativas, especialmente en el caso de experimentos de corta duración.
- Aumento de datos: utilice las instancias de Spot para realizar tareas de preprocesamiento y aumento de datos que pueden reiniciarse desde los puntos de control si se interrumpen, lo que las hace ideales para la disponibilidad variable de Spot.
- Modelos de ajuste preciso: utilice instancias puntuales para realizar ajustes precisos con sólidos mecanismos de puntos de control para reanudar desde el último estado guardado, lo que minimiza el impacto de las interrupciones de las instancias.
- Inferencia por lotes: utilice las instancias de Spot para procesar grandes lotes de solicitudes de inferencia fuera de línea de non-real-time manera que los trabajos se puedan pausar y reanudar, lo que ofrece la mejor alineación con los ahorros de costos de Spot y gestiona las posibles interrupciones mediante reintentos o diversificación.
- Subconjuntos de formación oportunistas: utilice instancias puntuales para cargas de trabajo de formación marginales o experimentales (p. ej., modelos más pequeños con menos de 10 millones de parámetros), en las que las interrupciones sean aceptables y se puedan aplicar optimizaciones de eficiencia, como la diversificación entre tipos de instancias o regiones, aunque no se recomienda para la formación a escala de producción debido a las posibles interrupciones.

Consideraciones

Para usar instancias puntuales para cargas de trabajo aceleradas en Amazon EKS, hay una serie de consideraciones clave (sin ningún orden en particular):

- Utilice Karpenter para administrar las instancias puntuales con la consolidación avanzada habilitada. Al especificar karpenter.sh/capacity type como «spot» en su Karpenter, Karpenter aprovisionará las instancias Spot de forma predeterminada sin necesidad de realizar ninguna NodePool configuración adicional. Sin embargo, para permitir una Spot-to-Spot consolidación avanzada, que sustituya los nodos Spot infrautilizados por alternativas Spot de menor precio, es necesario activar la SpotToSpotConsolidation función Feature Gate configurando --feature-

gates =true en los argumentos del controlador Karpenter o mediante la variable de entorno FEATURE_GATES. SpotToSpotConsolidation Karpenter EC2 utiliza la estrategia de asignación para aprovisionar las instancias. price-capacity-optimized En función de los NodePool requisitos y las restricciones de los pods, Karpenter agrupa los pods no programables y envía un conjunto diverso de tipos de instancias a la API de Amazon Fleet. EC2 Puede usar la herramienta EC2 Instance Type Explorer para generar una lista de tipos de instancias que coincidan con sus requisitos informáticos específicos.

- Asegúrese de que las cargas de trabajo no tengan estado, sean flexibles y toleren errores. Las cargas de trabajo deben ser apátridas, tolerantes a errores y flexibles en cuanto al tamaño. instance/GPU Esto permite una reanudación sin problemas tras las interrupciones de Spot, y la flexibilidad de las instancias le permite permanecer en Spot durante más tiempo. Habilite la gestión de interrupciones puntuales en Karpenter configurando el valor Helm de settings.InterruptionQueue con el nombre de la cola de AWS SQS para detectar los eventos de interrupción de Spot. Por ejemplo, cuando realice la instalación mediante Helm, utilice --set «settings.InterruptionQueue=${CLUSTER_NAME}». Para ver un ejemplo, consulta la guía de introducción a Karpenter. Cuando Karpenter detecta una interrupción de Spot, acordona, contamina, drena y cierra automáticamente los nodos antes de que se produzca la interrupción para maximizar el período de gracia de terminación de los módulos. Al mismo tiempo, Karpenter iniciará inmediatamente un nuevo nodo para que esté listo lo antes posible.
- Evite restringir demasiado la selección del tipo de instancia. Debe evitar restringir los tipos de instancia en la medida de lo posible. Al no restringir los tipos de instancias, hay más probabilidades de adquirir capacidad puntual a gran escala con una menor frecuencia de interrupciones de las instancias puntuales y a un coste menor. Por ejemplo, evite limitarse a tipos específicos (por ejemplo, g5.xlarge). Considera la posibilidad de especificar un conjunto diverso de categorías y generaciones de instancias mediante claves como karpenter.k8s. aws/instance-category and karpenter.k8s.aws/instance-generation. Karpenter enables easier diversification of on-demand and Spot instance capacity across multiple instance types and Availability Zones (AZs). Moreover, if your AI/MLLa carga de trabajo requiere un número específico o limitado de aceleradores, pero es flexible entre regiones. Puedes usar Spot Placement Score para identificar dinámicamente la región óptima para implementar tu carga de trabajo antes del lanzamiento.
- Amplíe NodePool los requisitos para incluir un mayor número de familias de EC2 instancias similares. Cada grupo de instancias puntuales consta de una capacidad de EC2 instancias no utilizada para un tipo de instancia específico en una zona de disponibilidad (AZ) específica. Cuando Karpenter intenta aprovisionar un nodo nuevo, selecciona un tipo de instancia que se ajusta a sus requisitos NodePool. Si ningún tipo de instancia compatible tiene capacidad puntual en ninguna zona de disponibilidad, se produce un error en el aprovisionamiento. Para evitar este

problema, permite utilizar instancias más amplias de la serie g (cuarta generación o superior) de NVIDIA en todos los tamaños y zonas de disponibilidad (AZs) y, al mismo tiempo, tener en cuenta las necesidades de hardware, como la memoria de la GPU o el trazado de rayos. Como las instancias pueden ser de distintos tipos, debes asegurarte de que tu carga de trabajo pueda ejecutarse en cada tipo y de que el rendimiento que obtengas esté a la altura de tus necesidades.

- Aproveche todas las zonas de disponibilidad de una región. La capacidad disponible varía según la zona de disponibilidad (AZ). Es posible que un tipo de instancia específico no esté disponible en una zona de disponibilidad, pero sea abundante en otra. Cada combinación única de un tipo de instancia y una zona de disponibilidad constituye un conjunto de capacidad puntual independiente. Al solicitar capacidad AZs en toda una región que cumpla con sus NodePool requisitos de Karpenter, está buscando más grupos a la vez. Esto maximiza la cantidad de grupos de capacidad puntual y, por lo tanto, aumenta la probabilidad de adquirir capacidad puntual. Para ello, en tu NodePool configuración, omite por completo la clave topology.kubernetes.io/zone para que Karpenter pueda seleccionar entre todas las disponibles AZs en la región, o bien haz una lista de forma explícita AZs mediante el operador In y proporciona los valores (p. ej., us-west-2a).
- Considere la posibilidad de utilizar la puntuación de ubicación puntual (SPS) para tener una idea clara de las probabilidades de acceder correctamente a la capacidad requerida mediante instancias puntuales. La puntuación de ubicación puntual (SPS) es una herramienta que proporciona una puntuación para ayudarle a evaluar las probabilidades de éxito de una solicitud de subasta. Cuando usa SPS, primero especifica los requisitos de procesamiento para sus instancias puntuales y, a continuación, Amazon EC2 devuelve las 10 regiones o zonas de disponibilidad principales en las que es probable que su solicitud de spot se apruebe. AZs Las regiones y las zonas de disponibilidad se califican en una escala del 1 al 10. Una puntuación de 10 indica que es muy probable que su solicitud de spot tenga éxito, pero no está garantizado. Una puntuación de 1 indica que no es probable que la solicitud de spot tenga éxito. Es posible que se devuelva la misma puntuación para distintas regiones o zonas de disponibilidad. Para obtener más información, consulte la Guía para crear un panel de seguimiento de puntuaciones de ubicación puntual en AWS. Como la capacidad puntual fluctúa constantemente, SPS le ayudará a identificar qué combinación de tipos de instancias y regiones se adapta mejor a sus limitaciones de carga de trabajo (es decir, flexibilidad, rendimiento, tamaño, etc.). AZs Si su AI/ML carga de trabajo requiere un número específico o limitado de aceleradores, pero es flexible entre regiones, puede usar la puntuación de ubicación de Spot para identificar dinámicamente la región óptima para implementar su carga de trabajo antes del lanzamiento. Para ayudarte a determinar automáticamente la probabilidad de adquirir la capacidad de Spot, te ofrecemos una guía para crear un panel de control de SPS Tracker. Esta solución monitorea las puntuaciones de SPS a lo largo del tiempo mediante una configuración YAML para configuraciones diversificadas (por

ejemplo, incluidos los requisitos de las instancias GPUs), almacena las métricas y proporciona paneles para comparar las configuraciones. CloudWatch Defina paneles por carga de trabajo para evaluar las necesidades de vCPU, memoria y GPU, garantizando una configuración óptima para los clústeres de EKS, incluida la consideración del uso de otras regiones de AWS. Para obtener más información, consulte Cómo funciona la puntuación de ubicación de Spot.

- Maneje con elegancia las interrupciones y las pruebas de Spot. En el caso de un módulo con un período de finalización superior a dos minutos, el nodo anterior se interrumpirá antes de reprogramarlos, lo que podría afectar a la disponibilidad de la carga de trabajo. Tenga en cuenta el aviso de interrupción puntual de dos minutos al diseñar sus aplicaciones, implemente puntos de control en las aplicaciones de larga ejecución (por ejemplo, guardar el progreso en un almacenamiento persistente como Amazon S3) para que se reanude tras las interrupciones, amplíe los terminationGracePeriod segundos (de forma predeterminada son 30 segundos) en las especificaciones del pod para permitir más tiempo de cierre sin problemas y gestione las interrupciones mediante los enganches and/or SIGTERM del ciclo de vida de PreStop dentro de su aplicación para actividades de cierre elegantes, como la limpieza, el ahorro de estado y el cierre de la conexión. Para las cargas de trabajo en tiempo real, en las que el tiempo de escalado es importante y la aplicación tarda más de dos minutos en estar lista para atender el tráfico, considere la posibilidad de optimizar los tiempos de inicio de los contenedores y de carga de los modelos de aprendizaje automático mediante la revisión the section called “Almacenamiento” y las prácticas the section called “Desempeño” recomendadas. Para probar un nodo de reemplazo, utilice AWS Fault Injection Service (FIS) para simular interrupciones puntuales.

Además de estas prácticas recomendadas básicas de Spot, tenga en cuenta estos factores a la hora de gestionar las cargas de trabajo de GPU en Amazon EKS. A diferencia de las cargas de trabajo basadas en la CPU, las cargas de trabajo de la GPU son particularmente sensibles a los detalles del hardware, como las capacidades de la GPU y la memoria de la GPU disponible. Las cargas de trabajo de GPU pueden estar limitadas por los tipos de instancias que pueden usar, ya que hay menos opciones disponibles en comparación con ellas. CPUs Como primer paso, evalúa si tu carga de trabajo es flexible en cuanto a instancias. Si no sabe cuántos tipos de instancias puede usar su carga de trabajo, pruébelos de forma individual para garantizar la compatibilidad y la funcionalidad. Identifique qué tan flexible puede ser para diversificar lo más posible y, al mismo tiempo, confirme que la diversificación hace que la carga de trabajo funcione y comprenda cualquier impacto en el rendimiento (por ejemplo, en el rendimiento o el tiempo de finalización). Como parte de la diversificación de sus cargas de trabajo, tenga en cuenta lo siguiente:

- Revise la compatibilidad entre CUDA y el marco. Es posible que tus cargas de trabajo de GPU estén optimizadas para hardware o tipos de GPU específicos (p. ej., V100 en p3 frente a A100 en p4) o estén escritas para versiones específicas de CUDA para bibliotecas como, por ejemplo TensorFlow, asegúrate de revisar la compatibilidad de tus cargas de trabajo. Esta compatibilidad es fundamental para evitar errores de tiempo de ejecución, bloqueos o fallos en la aceleración de la GPU (p. ej., versiones de CUDA que no coincidan con marcos similares PyTorch o que TensorFlow puedan impedir la ejecución) o para poder aprovechar funciones de hardware como la precisión/. FP16 INT8
- Memoria de GPU. Asegúrate de evaluar los requisitos de memoria de tus modelos y elaborar un perfil del uso de memoria de tu modelo durante el tiempo de ejecución con herramientas como el DCGM Exporter, y establece la memoria de GPU mínima requerida para el tipo de instancia en etiquetas conocidas como karpenter.k8s.aws/. instance-gpu-memory La VRAM de la GPU varía según el tipo de instancia (p. ej., NVIDIA T4 tiene 16 GB, A10G tiene 24 GB y V100 tiene 16-32 GB) y los modelos de aprendizaje automático (p. ej., modelos de lenguaje de gran tamaño) pueden superar la memoria disponible y provocar errores out-of-memory (OOM) o bloqueos. En el caso de las instancias puntuales de EKS, esto puede limitar la diversificación. Por ejemplo, no puede incluir tipos de VRAM más bajos si su modelo no es adecuado, lo que puede limitar el acceso a los grupos de capacidad y aumentar el riesgo de interrupciones. Ten en cuenta que en el caso de inferencias con una sola GPU y un solo nodo (por ejemplo, varios pods programados en el mismo nodo para utilizar los recursos de la GPU), esto podría limitar la diversificación, ya que solo puedes incluir tipos de instancias con suficiente VRAM en la configuración de Spot.
- Precisión y rendimiento de punto flotante. No todas las arquitecturas de GPU de Nvidia tienen la misma precisión de punto flotante (por ejemplo,/). FP16 INT8 Evalúe los tipos principales (CUDA/Tensor/RT), el rendimiento y la precisión de punto flotante necesarios para sus cargas de trabajo. Utilizar una GPU de menor precio y menor rendimiento no significa que sea mejor, así que considere evaluar el rendimiento en términos del trabajo realizado en un período de tiempo específico para comprender el impacto de la diversificación.

Escenario: Diversificación para cargas de trabajo de inferencia en tiempo real

Para una carga de trabajo de inferencias online en tiempo real en instancias puntuales, puede configurar un Karpenter NodePool para diversificar entre familias y generaciones de instancias de GPU compatibles. Este enfoque garantiza una alta disponibilidad al utilizar varios grupos de puntos puntuales y, al mismo tiempo, mantener el rendimiento debido a las limitaciones de las capacidades, la memoria y la arquitectura de la GPU. Admite el uso de alternativas cuando la capacidad de la instancia es limitada, lo que minimiza las interrupciones y optimiza la latencia de inferencia. En este

ejemplo, se NodePool indican las instancias de las series g y p superiores a 3, que tengan más de 20 GB de memoria de GPU.

Ejemplo

```
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
```

Implemente puntos de control para trabajos de formación de larga duración

Los puntos de control son una técnica de tolerancia a errores que consiste en guardar periódicamente el estado de un proceso, lo que permite reanudarlo desde el último punto guardado en caso de interrupciones. En el aprendizaje automático, suele asociarse a la formación, en la que las tareas de larga duración permiten ahorrar peso a los modelos y optimizar estados para reanudar la formación en caso de fallos, como problemas de hardware o interrupciones en instancias puntuales.

Los puntos de control se utilizan para guardar el estado de los modelos de aprendizaje automático (ML) durante el entrenamiento. Los puntos de control son instantáneas del modelo y se pueden configurar mediante las funciones de devolución de llamadas de los marcos de ML. Puede utilizar los puntos de control guardados para reiniciar un trabajo de entrenamiento desde el último punto de control guardado. Con los puntos de control, guardas las instantáneas de tus modelos durante el entrenamiento debido a una interrupción inesperada del trabajo o la instancia de entrenamiento. Esto le permite reanudar el entrenamiento del modelo en el futuro desde un punto de control. Además de implementar un sistema de resiliencia de nodos, recomendamos implementar puntos de control para mitigar el impacto de las interrupciones, incluidas las causadas por fallas de hardware o interrupciones de Amazon EC2 Spot Instance.

Sin puntos de control, las interrupciones pueden suponer una pérdida de tiempo de procesamiento y de progreso, lo que resulta costoso si se trata de tareas de formación de larga duración. Los puntos de control permiten que los trabajos guarden su estado periódicamente (p. ej., los pesos del modelo y los estados del optimizador) y se reanuden desde el último punto de control (último lote procesado) tras una interrupción. Para implementar los puntos de control, diseñe su aplicación para que procese los datos en lotes grandes y guarde los resultados intermedios en un almacenamiento persistente, como un bucket de Amazon S3, mediante el controlador CSI de Mountpoint para Amazon S3 a medida que avanza el trabajo de formación.

Casos de uso

Los puntos de control son especialmente beneficiosos en situaciones específicas para equilibrar la tolerancia a los fallos con la sobrecarga de rendimiento. Considere la posibilidad de utilizar puntos de control en los siguientes casos:

- La duración del trabajo supera unas pocas horas: para trabajos de formación de larga duración (por ejemplo, más de 1 a 2 horas para modelos pequeños o days/weeks para modelos de

bases grandes con miles de millones de parámetros), en los que la pérdida de progreso por las interrupciones es costosa. Es posible que los trabajos más cortos no justifiquen los gastos generales. I/O

- Para instancias puntuales o fallos de hardware: en entornos propensos a sufrir interrupciones, como los fallos EC2 puntuales (con 2 minutos de antelación) o fallos de hardware (p. ej., errores de memoria de la GPU), los puntos de control permiten una reanudación rápida, lo que hace que Spot sea viable para ahorrar costes en cargas de trabajo tolerantes a fallos.
- Formación distribuida a escala: para configuraciones con aceleradores (por ejemplo, >100 GPUs), en hundreds/thousands las que el tiempo medio entre fallos disminuye de forma lineal con la escala. Utilícelo como model/data paralelismo para gestionar el acceso simultáneo a los puntos de control y evitar reinicios completos.
- Modelos a gran escala con una alta demanda de recursos: en la formación de LLM a escala de petabytes, donde las fallas son inevitables debido al tamaño del clúster, los enfoques escalonados (local rápido cada 5 a 30 minutos para los transitorios, durables cada hora para los fallos importantes) optimizan el tiempo de recuperación en lugar de la eficiencia.

Utilice bloques de capacidad ML para garantizar la capacidad de las instancias P y Trainium

Los bloques de capacidad para ML le permiten reservar instancias de GPU muy solicitadas, específicamente instancias P (por ejemplo, p6-b200, p5, p5e, p5en, p4d, p4de) e instancias de Trainium (por ejemplo, trn1, trn2), para que comiencen casi de inmediato o en una fecha futura para respaldar sus cargas de trabajo de aprendizaje automático (ML) de corta duración. Estas reservas son ideales para garantizar la capacidad para realizar tareas con un uso intensivo de cómputo, como el entrenamiento de modelos y el ajuste fino. EC2 El precio de Capacity Blocks consiste en una cuota de reserva y una cuota de sistema operativo. Para obtener más información sobre los precios, consulta los precios EC2 de Capacity Blocks for ML.

Para reservar GPUs para AI/ML las cargas de trabajo en Amazon EKS y garantizar la capacidad de forma predecible, le recomendamos que utilice los bloques de capacidad de aprendizaje automático para reservas de capacidad a corto plazo o bajo demanda (ODCRs) para garantizar la capacidad de uso general.

- ODCRs le permiten reservar la capacidad de las EC2 instancias (p. ej., instancias de GPU como la g5 o la p5) en una zona de disponibilidad específica durante un tiempo, lo que garantiza la disponibilidad, incluso en situaciones de alta demanda. ODCRs no tiene un compromiso a

largo plazo, pero paga la tarifa bajo demanda por la capacidad reservada, ya sea utilizada o inactiva. En EKS, ODCRs son compatibles con tipos de nodos como Karpenter y grupos de nodos gestionados. Para priorizar ODCRs en Karpenter, configure el campo NodeClass para usar. `capacityReservationSelectorTerms` Consulte la documentación de Karpenter NodePools .

- Los bloques de capacidad son un mecanismo de reserva especializado para instancias de GPU (p. ej., p5, p4d) o Trainium (trn1, trn2), diseñado para cargas de trabajo de aprendizaje automático a corto plazo, como el entrenamiento de modelos, el ajuste o la experimentación. Usted reserva capacidad durante un período definido (normalmente de 24 horas a 182 días) a partir de una fecha futura y paga solo por el tiempo reservado. Se pagan por adelantado, requieren una planificación previa en función de las necesidades de capacidad y no admiten el escalado automático, pero se colocan en EC2 UltraClusters el mismo lugar para redes de baja latencia. Solo cobran por el período reservado. Para obtener más información, consulte Buscar y comprar bloques de capacidad o comience por configurar grupos de nodos gestionados con bloques de capacidad siguiendo las instrucciones de Cómo crear un grupo de nodos gestionado con bloques de capacidad para aprendizaje automático.

Reserve capacidad a través de la consola de administración de AWS y configure sus nodos para que usen bloques de capacidad de aprendizaje automático. Planifique las reservas en función de los cronogramas de las cargas de trabajo y realice pruebas en un clúster provisional. Consulte la documentación sobre los bloques de capacidad para obtener más información.

Considere la posibilidad de reservar capacidad bajo demanda, Amazon EC2 Spot o On-Demand (ODCRs) para EC2 las instancias G Amazon

Para las EC2 instancias G de Amazon, considere las diferentes opciones de compra entre On-Demand, Amazon EC2 Spot Instances y On-Demand Capacity Reservations. ODCRspermiten reservar la capacidad de las EC2 instancias en una zona de disponibilidad específica durante un período determinado, lo que garantiza la disponibilidad incluso en momentos de alta demanda. A diferencia de los bloques de capacidad de aprendizaje automático, que solo están disponibles para las instancias P y Trainium, se ODCRs pueden usar para una gama más amplia de tipos de instancias, incluidas las instancias G, lo que los hace adecuados para cargas de trabajo que requieren diferentes capacidades de GPU, como inferencias o gráficos. Al utilizar Amazon EC2 Spot Instances, es fundamental poder diversificar los distintos tipos, tamaños y zonas de disponibilidad de instancias para poder permanecer en Spot durante más tiempo.

ODCRs no tiene un compromiso a largo plazo, pero paga la tarifa bajo demanda por la capacidad reservada, ya sea utilizada o inactiva. ODCRs puede crearse para su uso inmediato o programarse

para una fecha futura, lo que proporciona flexibilidad en la planificación de la capacidad. En Amazon EKS, ODCRs son compatibles con tipos de nodos como Karpenter y grupos de nodos gestionados. Para priorizar ODCRs en Karpenter, configure el campo NodeClass para usar. `capacityReservationSelectorTerms` Consulte la documentación de Karpenter NodePools . Para obtener más información sobre la creación ODCRs, incluidos los comandos CLI, consulte Introducción a la reserva de capacidad bajo demanda.

Considere otros tipos y tamaños de instancias aceleradas

La selección de la instancia acelerada y el tamaño adecuados es esencial para optimizar el rendimiento y el costo de sus cargas de trabajo de aprendizaje automático en Amazon EKS. Por ejemplo, las distintas familias de instancias de GPU tienen un rendimiento y capacidades diferentes, como la memoria de GPU. Para ayudarte a elegir la opción con la mejor relación precio-rendimiento, consulta las instancias de GPU disponibles en la página de tipos de instancias, en EC2 Computación acelerada. Evalúe varios tipos y tamaños de instancias para encontrar la que mejor se adapte a sus requisitos de carga de trabajo específicos. Tenga en cuenta factores como la cantidad GPUs, la memoria y el rendimiento de la red. Si selecciona cuidadosamente el tipo y el tamaño de instancia de GPU correctos, puede lograr una mejor utilización de los recursos y una mejor rentabilidad en sus clústeres de EKS.

Si utilizas una instancia de GPU en un nodo EKS, el `nvidia-device-plugin-daemonset` pod estará en el espacio de `kube-system` nombres de forma predeterminada. Para tener una idea rápida de si estás utilizando al máximo las GPU de tu instancia, puedes usar nvidia-smi como se muestra a continuación:

```
kubectl exec nvidia-device-plugin-daemonset-xxxxx \
  -n kube-system -- nvidia-smi \
  --query-
gpu=index,power.draw,power.limit,temperature.gpu,utilization.gpu,utilization.memory,memory.free
 \
  --format=csv -l 5
```

- Si `utilization.memory` se acerca al 100%, es probable que los códigos estén limitados a la memoria. Esto significa que la GPU (memoria) se utiliza al máximo, pero podría sugerir que se debería investigar una mayor optimización del rendimiento.
- Si `utilization.gpu` se acerca al 100%, esto no significa necesariamente que la GPU esté totalmente utilizada. Una métrica mejor a tener en cuenta es la relación entre `power.draw`

ypower.limit. Si esta proporción es del 100% o más, significa que tus códigos están utilizando al máximo la capacidad de procesamiento de la GPU.

- El `-l 5` indicador indica que hay que generar las métricas cada 5 segundos. En el caso de un único tipo de instancia de GPU, no se necesita el indicador de consulta de índice.

Para obtener más información, consulte las instancias de GPU en la documentación de AWS.

Optimice la asignación de recursos de GPU con la asignación fraccionada de GPU, MIG y segmentada

Los límites estáticos de los recursos en Kubernetes (por ejemplo, el número de CPU, memoria o GPU) pueden provocar un aprovisionamiento excesivo o una subutilización, especialmente en el caso de cargas de trabajo dinámicas, como la inferencia. AI/ML Es importante seleccionar la GPU adecuada. Para cargas de trabajo de bajo volumen o con picos de actividad, la división del tiempo permite que varias cargas de trabajo compartan una sola GPU al compartir sus recursos de cómputo, lo que podría mejorar la eficiencia y reducir el desperdicio. El uso compartido de la GPU se puede lograr mediante diferentes opciones:

- Aproveche los selectores de nodos y la afinidad entre los nodos para influir en la programación: asegúrese de que los nodos aprovisionados y los pods estén programados de forma adecuada GPUs a la carga de trabajo (por ejemplo,) `karpenter.k8s.aws/instance-gpu-name: "a100"`
- Distribución del tiempo: programa las cargas de trabajo para compartir los recursos informáticos de la GPU a lo largo del tiempo, lo que permite la ejecución simultánea sin particiones físicas. Esto es ideal para cargas de trabajo con demandas informáticas variables, pero que pueden carecer de aislamiento de memoria.
- GPU de varias instancias (MIG): la tecnología MIG permite dividir una sola GPU NVIDIA en varias instancias aisladas y es compatible con NVIDIA Ampere (por ejemplo, la GPU A100), NVIDIA Hopper (por ejemplo, la GPU H100) y NVIDIA Blackwell (por ejemplo, Blackwell). GPUs GPUs Cada instancia MIG recibe recursos de cómputo y memoria dedicados, lo que permite compartir recursos en entornos con varios usuarios o cargas de trabajo que requieren garantías de recursos, lo que le permite optimizar el uso de los recursos de la GPU, incluidos escenarios como el servicio de varios modelos con diferentes tamaños de lote mediante la división del tiempo.
- Asignación fraccionada de la GPU: utiliza una programación basada en software para asignar partes del cómputo o la memoria de una GPU a las cargas de trabajo, lo que ofrece flexibilidad

para las cargas de trabajo dinámicas. El programador KAI de NVIDIA, que forma parte de la plataforma Run:AI, lo permite al permitir que los módulos soliciten recursos de GPU fraccionados.

Para habilitar estas funciones en EKS, puedes implementar el complemento para dispositivos NVIDIA, que presenta recursos programables y admite la segmentación de GPUs tiempos y el MIG. Para obtener más información, consulte Separación de tiempo GPUs en Kubernetes y uso compartido de GPU en Amazon EKS con instancias aceleradas y segmentadas de NVIDIA. EC2

Ejemplo

Por ejemplo, para habilitar la división del tiempo con el complemento de dispositivo NVIDIA:

```
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
```

Ejemplo

Por ejemplo, si quieres usar el programador KAI para la asignación fraccionada de la GPU, despliégalo junto con el operador de GPU de NVIDIA y especifica los recursos fraccionarios de la GPU en las especificaciones del módulo:

```
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
```

Resiliencia de nodos y gestión de trabajos de formación

Implemente las comprobaciones de estado de los nodos con recuperación automática

En el caso de los trabajos de formación distribuidos en Amazon EKS que requieren una comunicación frecuente entre nodos, como el entrenamiento de modelos con varias GPU en varios nodos, los problemas de hardware, como los fallos de la GPU o del EFA, pueden provocar interrupciones en los trabajos de formación. Estas interrupciones pueden provocar una pérdida del progreso en la formación y un aumento de los costes, especialmente en el caso de las cargas de AI/ML trabajo de larga duración que dependen de un hardware estable.

Para aumentar la resistencia frente a los fallos de hardware, como los fallos de GPU en los clústeres de EKS que ejecutan cargas de trabajo de GPU, recomendamos utilizar el agente de monitorización de nodos de EKS con Auto Repair o Amazon. SageMaker HyperPod Si bien el agente de monitorización de nodos EKS con reparación automática proporciona funciones como la supervisión del estado de los nodos y la reparación automática mediante mecanismos estándar de Kubernetes, SageMaker HyperPod ofrece una resiliencia específica y funciones adicionales diseñadas específicamente para la formación de aprendizaje automático a gran escala, como controles de estado exhaustivos y reanudación automática de tareas.

- El agente de monitorización de nodos EKS con Node Auto Repair supervisa continuamente el estado de los nodos mediante la lectura de los registros y su aplicación NodeConditions, incluidas las condiciones estándar, `Ready` y las condiciones específicas del hardware acelerado, para identificar problemas como fallos en la GPU o en la red. Cuando se considera que un nodo está en mal estado, Node Auto Repair lo acordona y lo reemplaza por un nodo nuevo. La reprogramación de los pods y el reinicio de las tareas se basan en los mecanismos estándar de Kubernetes y en la política de reinicio de las tareas.
- El agente de controles de estado SageMaker HyperPodexhaustivos y monitorización del estado monitorea continuamente el estado de las instancias basadas en GPU y Trainium. Está diseñado

para las AI/ML cargas de trabajo y utiliza etiquetas (por ejemplo node-health-status) para gestionar el estado de los nodos. Cuando se considera que un nodo está en mal estado, se HyperPod activa la sustitución automática del hardware defectuoso, por ejemplo. GPUs Detecta los fallos relacionados con la red para la EFA mediante sus comprobaciones de estado básicas de forma predeterminada y admite la reanudación automática en los trabajos de formación interrumpidos, lo que permite continuar los trabajos desde el último punto de control y minimizar las interrupciones de las tareas de aprendizaje automático a gran escala.

Tanto para el agente de monitoreo de nodos de EKS con reparación automática como para SageMaker HyperPod los clústeres que utilizan EFA, para monitorear métricas específicas de EFA, como los errores de acceso remoto directo a la memoria (RDMA) y las caídas de paquetes, asegúrese de que el controlador EFA de AWS esté instalado. Además, recomendamos implementar el complemento CloudWatch Observability o utilizar herramientas como DCGM Exporter con Prometheus y Grafana para monitorear la EFA, la GPU y, en busca de métricas específicas relacionadas con sus funciones. SageMaker HyperPod

Desactive Karpenter Consolidation para las cargas de trabajo sensibles a las interrupciones

Para las cargas de trabajo sensibles a las interrupciones, como el procesamiento, las tareas de AI/ML predicción a gran escala o la formación, recomendamos ajustar las políticas de consolidación de Karpenter para evitar interrupciones durante la ejecución del trabajo. La función de consolidación de Karpenter optimiza automáticamente los costes de los clústeres al cerrar los nodos infrautilizados o sustituirlos por alternativas de menor precio. Sin embargo, incluso cuando una carga de trabajo utilice al máximo una GPU, Karpenter puede consolidar los nodos si identifica un tipo de instancia de menor precio y del tamaño adecuado que cumpla con los requisitos del módulo, lo que provoca interrupciones en el trabajo.

La política de `WhenEmptyOrUnderutilized` consolidación puede cerrar los nodos de forma prematura, lo que prolongaría los tiempos de ejecución. Por ejemplo, las interrupciones pueden retrasar la reanudación del trabajo debido a la reprogramación de los módulos o a la recarga de datos, lo que puede resultar costoso en el caso de los trabajos de inferencia por lotes de larga duración. Para mitigar esta situación, puedes establecer `WhenEmpty` y configurar una `consolidateAfter` duración, por ejemplo, de 1 hora, `consolidationPolicy` para retener los nodos durante los picos de carga de trabajo. Por ejemplo:

```
disruption:
  consolidationPolicy: WhenEmpty
```

```
  consolidateAfter: 60m
```

Este enfoque mejora la latencia de arranque de los módulos para cargas de trabajo de inferencia por lotes con picos de actividad y otras tareas sensibles a las interrupciones, como el procesamiento de datos de inferencias en línea en tiempo real o la formación de modelos, en las que el coste de la interrupción supera el ahorro de costes informáticos. Los presupuestos de disrupción de Karpenter son otra función para gestionar las NodePool interrupciones de Karpenter. Con los presupuestos, puede asegurarse de que no más de un número determinado de nodos se interrumpan en los nodos elegidos en un NodePool momento dado. También puede utilizar los presupuestos de interrupciones para evitar que todos los nodos se interrumpan en un momento determinado (por ejemplo, en las horas punta). Para obtener más información, consulte la documentación de consolidación de Karpenter.

Utilice ttlSecondsAfter Finished para limpiar automáticamente los trabajos de Kubernetes

Recomendamos configurar `ttlSecondsAfterFinished` los trabajos de Kubernetes en Amazon EKS para eliminar automáticamente los objetos de trabajo completados. Los objetos de trabajo persistentes consumen recursos del clúster, como la memoria del servidor API, y complican la supervisión al abarrotar los paneles de control (por ejemplo, Grafana o Amazon). CloudWatch Por ejemplo, si se establece un TTL de 1 hora, se garantiza que los trabajos se eliminen poco después de su finalización, lo que mantiene el clúster tidy. Para obtener más información, consulte Limpieza automática de trabajos terminados.

Configure la prioridad de tareas de baja prioridad para tareas o cargas de trabajo de mayor prioridad

Para las cargas de AI/ML trabajo de prioridad mixta en Amazon EKS, puede configurar la prioridad de tareas de baja prioridad para garantizar que las tareas de mayor prioridad (por ejemplo, la inferencia en tiempo real) reciban los recursos con prontitud. Sin preferencia, las cargas de trabajo de baja prioridad, como los procesos por lotes (por ejemplo, la inferencia por lotes o el procesamiento de datos), los servicios que no son por lotes (por ejemplo, tareas en segundo plano, trabajos cron) o los trabajos que consumen mucha CPU o memoria (por ejemplo, servicios web) pueden retrasar los pods críticos al ocupar nodos. La preferencia permite a Kubernetes desalojar los módulos de baja prioridad cuando los de alta prioridad necesitan recursos, lo que garantiza una asignación eficiente de los recursos en los nodos con memoria o memoria. GPUs CPUs Recomendamos usar `PriorityClass` Kubernetes para asignar prioridades y controlar el comportamiento de desalojo. `PodDisruptionBudget`

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: low-priority
value: 100
---
spec:
  priorityClassName: low-priority
```

Consulte la documentación de prioridad y preferencia de Kubernetes para obtener más información.

Escalado y rendimiento de las aplicaciones

Personalice la capacidad de cómputo para las cargas de trabajo de aprendizaje automático con Karpenter o nodos estáticos

Para garantizar una capacidad informática rentable y con capacidad de respuesta para los flujos de trabajo de aprendizaje automático (ML) en Amazon EKS, le recomendamos que adapte su estrategia de aprovisionamiento de nodos a las características de su carga de trabajo y a sus compromisos de costes. A continuación, se presentan dos enfoques a tener en cuenta: just-in-time escalar con Karpenter y con grupos de nodos estáticos para obtener capacidad reservada.

- Just-in-time escaladores de planos de datos como Karpenter: para flujos de trabajo de aprendizaje automático dinámicos con demandas informáticas variables (por ejemplo, inferencias basadas en GPU seguidas de gráficos basados en CPU), recomendamos utilizar escaladores de planos de datos como Karpenter. just-in-time
- Utilice grupos de nodos estáticos para cargas de trabajo predecibles: para cargas de trabajo de aprendizaje automático predecibles y estables o cuando utilice instancias reservadas, los grupos de nodos gestionados por EKS pueden ayudar a garantizar que la capacidad reservada se aprovisione y utilice por completo, lo que maximiza los ahorros. Este enfoque es ideal para tipos de instancias específicos comprometidos mediante o. RIs ODCRs

Ejemplo

Este es un ejemplo de un Karpenter diverso NodePoolque permite lanzar instancias de g Amazon en las que la generación de EC2 instancias es superior a tres.

```
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
```

Ejemplo

Ejemplo de uso de grupos de nodos estáticos para una carga de trabajo de formación:

```
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
```

Utilice restricciones y tolerancias para evitar que las cargas de trabajo no aceleradas se programen en instancias aceleradas

Programar las cargas de trabajo no aceleradas en los recursos de la GPU no es eficiente desde el punto de vista informático. Recomendamos utilizar las restricciones y la tolerancia para garantizar que los módulos de cargas de trabajo no aceleradas no se programen en nodos inadecuados. Consulte la documentación de Kubernetes para obtener más información.

Escale en función del rendimiento del modelo

Para las cargas de trabajo de inferencia, recomendamos utilizar el escalado automático basado en eventos (KEDA) de Kubernetes para escalar en función de las métricas de rendimiento del modelo, como las solicitudes de inferencia o el rendimiento de los tokens, con los períodos de espera adecuados. Las políticas de escalado estático pueden aprovisionar recursos de forma excesiva o insuficiente, lo que repercute en los costes y la latencia. Obtenga más información en la documentación de KEDA.

Asignación dinámica de recursos para una gestión avanzada de la GPU

La asignación dinámica de recursos (DRA) representa un avance fundamental en la gestión de los recursos de GPU de Kubernetes. La DRA supera las limitaciones tradicionales de los complementos de dispositivos para permitir el uso compartido sofisticado de la GPU, el reconocimiento de la topología y la coordinación de los recursos entre nodos. Disponible en la versión 1.33 de Amazon EKS, DRA aborda los desafíos críticos de las AI/ML cargas de trabajo al proporcionar lo siguiente:

- Asignación de GPU detallada
- Mecanismos de uso compartido avanzados, como el servicio multiproceso (MPS) y la GPU multiinstancia (MIG)
- Support para arquitecturas de hardware de próxima generación, incluida NVIDIA 00 GB2 UltraServers

La asignación de GPU tradicional trata los recursos GPUs como enteros opacos, lo que genera una subutilización significativa (a menudo del 30 al 40% en los clústeres de producción). Esto se debe a que las cargas de trabajo reciben acceso exclusivo a todos los recursos, GPUs incluso cuando solo requieren una fracción de los recursos. DRA transforma este modelo al introducir una asignación declarativa y estructurada que proporciona al programador de Kubernetes una visibilidad completa de las características del hardware y los requisitos de carga de trabajo. Esto permite tomar decisiones de ubicación inteligentes y compartir recursos de manera eficiente.

Ventajas de usar DRA en lugar del complemento de dispositivo NVIDIA

El complemento de dispositivo NVIDIA (a partir de la versión`0.12.0`) admite mecanismos de uso compartido de GPU, como la división de tiempo, el MPS y el MIG. Sin embargo, existen limitaciones arquitectónicas que DRA aborda.

Limitaciones de los complementos de dispositivo NVIDIA

- Configuración estática: las configuraciones de uso compartido de la GPU (réplicas segmentadas y ajustes de MPS) requieren una configuración previa de todo el clúster. `ConfigMaps` Esto dificulta proporcionar diferentes estrategias de uso compartido para diferentes cargas de trabajo.
- Selección detallada limitada: si bien el complemento del dispositivo expone las características de la GPU mediante etiquetas de nodos, las cargas de trabajo no pueden solicitar dinámicamente configuraciones de GPU específicas (tamaño de memoria y capacidades de procesamiento) como parte de la decisión de programación.
- No hay coordinación de recursos entre nodos: no se pueden gestionar los recursos de la GPU distribuidos en varios nodos ni cumplir requisitos de topología complejos, como los NVLink dominios de sistemas como NVIDIA 00. GB2
- Restricciones del programador: el programador de Kubernetes trata los recursos de la GPU como números enteros opacos, lo que limita su capacidad para tomar decisiones basadas en la topología o gestionar dependencias complejas de recursos.
- Complejidad de la configuración: la configuración de diferentes estrategias de uso compartido requiere un etiquetado cuidadoso y múltiple de los nodos, lo que crea complejidad operativa. `ConfigMaps`

Soluciones con DRA

- Selección dinámica de recursos: la DRA permite a las cargas de trabajo especificar requisitos detallados (memoria de la GPU, versiones de los controladores y atributos específicos) en el

momento de la solicitud. `resourceclaims` Esto permite una correspondencia de recursos más flexible.

- Conocimiento de la topología: mediante parámetros estructurados y selectores de dispositivos, DRA gestiona requisitos complejos, como la comunicación entre nodos de la GPU y las interconexiones coherentes con la memoria.
- Gestión de recursos entre nodos: `computeDomains` permite la coordinación de los recursos de la GPU distribuidos en varios nodos, algo fundamental para sistemas como 00 con canales IMEX. GB2
- Configuración específica de la carga de trabajo: cada uno de ellos `ResourceClaim` especifica diferentes estrategias y configuraciones de uso compartido, lo que permite un control pormenorizado por carga de trabajo, en lugar de hacerlo a nivel de todo el clúster.
- Integración mejorada del programador: DRA proporciona al programador información detallada del dispositivo y permite tomar decisiones de ubicación más inteligentes en función de la topología del hardware y las características de los recursos.

Importante: DRA no sustituye por completo al complemento del dispositivo NVIDIA. El controlador DRA de NVIDIA funciona junto con el complemento del dispositivo para ofrecer funciones mejoradas. El complemento del dispositivo sigue encargándose de la detección y la administración básicas de la GPU, mientras que DRA añade funciones avanzadas de asignación y programación.

Instancias compatibles con DRA y sus funciones

La compatibilidad con DRA varía según la familia de EC2 instancias de Amazon y la arquitectura de la GPU, como se muestra en la siguiente tabla.

Familia de instancias	Tipo de GPU	Segmentación del tiempo	Soporte MIG	Soporte MPS	Soporte IMEX	Casos de uso
G5	NVIDIA A10G	Sí	No	Sí	No	Cargas de trabajo de inferencia y gráficos
G6	NVIDIA L4	Sí	No	Sí	No	Inferencia de IA y procesamiento de vídeo

Familia de instancias	Tipo de GPU	Segmentación del tiempo	Soporte MIG	Soporte MPS	Soporte IMEX	Casos de uso
G6e	NVIDIA L40S	Sí	No	Sí	No	Entrenamiento, inferencia y gráficos
P4d/ P4de	NVIDIA A100	Sí	Sí	Sí	No	Formación a gran escala y HPC
P5	NVIDIA H100	Sí	Sí	Sí	No	Formación sobre modelos básicos
P6	NVIDIA B200	Sí	Sí	Sí	No	Modelos de mil millones o billones de parámetros, entrenamiento distribuido e inferencia
P6e	NVIDIA 00 GB2	Sí	Sí	Sí	Sí	Modelos de mil millones o billones de parámetros, entrenamiento distribuido e inferencia

Las siguientes son descripciones de cada función de la tabla:

- Segmentación del tiempo: permite que varias cargas de trabajo compartan los recursos informáticos de la GPU a lo largo del tiempo.
- GPU de instancias múltiples (MIG): partición a nivel de hardware que crea instancias de GPU aisladas.
- Servicio multiproceso (MPS): permite la ejecución simultánea de varios procesos CUDA en una sola GPU.

- Intercambio de memoria entre nodos (IMEX): comunicación coherente con la memoria entre los nodos para 00. GB2 UltraServers

Recursos adicionales

Para obtener más información sobre los controladores DRA de Kubernetes y DRA de NVIDIA, consulta los siguientes recursos en: GitHub

- Kubernetes dynamic-resource-allocation
- Propuesta de mejora de Kubernetes para DRA
- Controlador NVIDIA DRA para GPUs
- Ejemplos e inicio rápido de NVIDIA DRA

Configure la asignación dinámica de recursos para una administración avanzada de la GPU

En el siguiente tema, se muestra cómo configurar la asignación dinámica de recursos (DRA) para la administración avanzada de la GPU.

Requisitos previos

Antes de implementar DRA en Amazon EKS, asegúrese de que su entorno cumpla los siguientes requisitos.

Configuración del clúster

- Clúster Amazon EKS en ejecución (versión `1.33` o posterior)
- Grupos de nodos administrados por Amazon EKS (actualmente, DRA solo es compatible con grupos de nodos administrados con AL2 023 y Bottlerocket optimizados para NVIDIA AMIs, no con Karpenter)
- Nodos de trabajo compatibles con la GPU de NVIDIA con los tipos de instancias adecuados

Componentes necesarios

- Versión del complemento para dispositivos NVIDIA `0.17.1` o posterior
- Versión del controlador NVIDIA DRA `25.3.0` o posterior

Paso 1: Cree un clúster con un grupo de nodos habilitado para DRA mediante eksctl

1. Cree un archivo de configuración de clúster llamado: `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. Cree el clúster:

```
eksctl create cluster -f dra-eks-cluster.yaml
```

Paso 2: Implemente el complemento de dispositivo NVIDIA

Implemente el complemento de dispositivo NVIDIA para permitir la detección básica de la GPU:

1. Añada el repositorio Helm del complemento para dispositivos NVIDIA:

```
helm repo add nvidia https://nvidia.github.io/k8s-device-plugin
```

```
helm repo update
```

2. Cree valores personalizados para el complemento del dispositivo:

```
cat <<EOF > nvidia-device-plugin-values.yaml
gfd:
  enabled: true
nfd:
  enabled: true
tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
EOF
```

3. Instale el complemento para dispositivos NVIDIA:

```
helm install nvidia-device-plugin nvidia/nvidia-device-plugin \
 --namespace nvidia-device-plugin \
 --create-namespace \
 --version v0.17.1 \
 --values nvidia-device-plugin-values.yaml
```

Paso 3: Implemente el controlador DRA de NVIDIA Helm chart

1. Cree un archivo de `dra-driver-values.yaml` valores para el controlador DRA:

```
---
nvidiaDriverRoot: /

gpuResourcesEnabledOverride: true

resources:
  gpus:
    enabled: true
  computeDomains:
    enabled: true  # Enable for GB200 IMEX support

controller:
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
```

```
    effect: NoSchedule

kubeletPlugin:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: "nvidia.com/gpu.present"
            operator: In
            values: ["true"]
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
```

2. Añada el repositorio NVIDIA NGC Helm:

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
```

3. Instale el controlador NVIDIA DRA:

```
helm install nvidia-dra-driver nvidia/nvidia-dra-driver-gpu \
 --version="25.3.0-rc.2" \
 --namespace nvidia-dra-driver \
 --create-namespace \
 --values dra-driver-values.yaml
```

Paso 4: Compruebe la instalación de la DRA

1. Compruebe que los recursos de la API DRA estén disponibles:

```
kubectl api-resources | grep resource.k8s.io/v1beta1
```

   El resultado esperado es el siguiente:

```
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. Compruebe las clases de dispositivos disponibles:

```
kubectl get deviceclasses
```

A continuación se muestra un ejemplo del resultado esperado:

```
NAME                                        AGE
compute-domain-daemon.nvidia.com            4h39m
compute-domain-default-channel.nvidia.com   4h39m
gpu.nvidia.com                              4h39m
mig.nvidia.com                              4h39m
```

Cuando una instancia de GPU G6 recién creada se une a su clúster de Amazon EKS con DRA habilitada, se llevan a cabo las siguientes acciones:

- El controlador DRA de NVIDIA descubre automáticamente la GPU A10G y crea dos `resourceslices` en ese nodo.
- El `gpu.nvidia.com` segmento registra el dispositivo GPU A10G físico con sus especificaciones (memoria, capacidad de cómputo, etc.).
- Como el A10G no admite la partición MIG, el `compute-domain.nvidia.com` segmento crea un único dominio de procesamiento que representa todo el contexto de procesamiento de la GPU.
- Luego, `resourceslices` se publican en el servidor API de Kubernetes, lo que permite que los recursos de la GPU estén disponibles para su programación. `resourceclaims`

  El programador DRA ahora puede asignar esta GPU de forma inteligente a los pods que solicitan recursos de la GPU`resourceclaimtemplates`, lo que proporciona una administración de recursos más flexible en comparación con los enfoques tradicionales de complementos de dispositivos. Esto ocurre automáticamente sin intervención manual. El nodo simplemente estará disponible para las cargas de trabajo de la GPU una vez que el controlador DRA complete el proceso de descubrimiento y registro de los recursos.

  Al ejecutar el siguiente comando:

  ```
  kubectl get resourceslices
  ```

  A continuación se muestra un ejemplo del resultado esperado:

```
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
```

Siga en the section called “Programa una carga de trabajo de GPU sencilla mediante la asignación dinámica de recursos”.

Programa una carga de trabajo de GPU sencilla mediante la asignación dinámica de recursos

Para programar una carga de trabajo de GPU sencilla mediante la asignación dinámica de recursos (DRA), sigue estos pasos. Antes de continuar, asegúrate de haberlos seguidothe section called “Configure la asignación dinámica de recursos para una administración avanzada de la GPU”.

1. Crea una asignación básica `ResourceClaimTemplate` de GPU con un archivo llamado`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. Aplica la plantilla:

```
kubectl apply -f basic-gpu-claim-template.yaml
```

3. Compruebe el estado:

```
kubectl get resourceclaimtemplates -n gpu-test1
```

   A continuación, se muestra un ejemplo de la salida:

```
NAME         AGE
single-gpu   9m16s
```

4. Cree un pod que lo utilice `ResourceClaimTemplate` con un archivo llamado`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. Aplica y monitorea el Pod:

```
kubectl apply -f basic-gpu-pod.yaml
```

6. Comprueba el estado del pod:

```
kubectl get pod -n gpu-test1
```

   A continuación se muestra un ejemplo del resultado esperado:

```
NAME      READY   STATUS    RESTARTS   AGE
gpu-pod   1/1     Running   0          13m
```

7. Compruebe el `ResourceClaim` estado:

```
kubectl get resourceclaims -n gpu-test1
```

   A continuación se muestra un ejemplo del resultado esperado:

```
NAME                 STATE                AGE
gpu-pod-gpu0-l76cg   allocated,reserved   9m6s
```

8. Consulta los registros del pod para ver la información de la GPU:

```
kubectl logs gpu-pod -n gpu-test1
```

   A continuación se muestra un ejemplo del resultado esperado:

```
GPU 0: NVIDIA L4 (UUID: GPU-da7c24d7-c7e3-ed3b-418c-bcecc32af7c5)
```

Continúe con the section called “Técnicas de optimización de GPU con asignación dinámica de recursos” nosotros para obtener técnicas de optimización de GPU más avanzadas mediante DRA.

Técnicas de optimización de GPU con asignación dinámica de recursos

Las cargas de trabajo de GPU modernas requieren una administración de recursos sofisticada para lograr una utilización óptima y una rentabilidad óptimas. DRA permite varias técnicas de optimización avanzadas que abordan diferentes casos de uso y capacidades del hardware:

- La división del tiempo permite que varias cargas de trabajo compartan los recursos de cómputo de la GPU a lo largo del tiempo, lo que la hace ideal para cargas de trabajo de inferencia con un uso esporádico de la GPU. Para ver un ejemplo, consulta the section called “Optimice las cargas de trabajo de la GPU mediante la división del tiempo”.
- El servicio multiproceso (MPS) permite la ejecución simultánea de varios procesos de CUDA en una sola GPU con un mejor aislamiento que la división temporal. Para ver un ejemplo, consulta the section called “Optimice las cargas de trabajo de la GPU con MPS”.
- La GPU de instancias múltiples (MIG) proporciona particiones a nivel de hardware, lo que crea instancias de GPU aisladas con recursos de cómputo y memoria dedicados. Para ver un ejemplo, consulta the section called “Optimice las cargas de trabajo de la GPU con una GPU de instancias múltiples”.
- El intercambio de memoria entre nodos (IMEX) permite una comunicación coherente con la memoria entre los nodos para una formación distribuida en los sistemas NVIDIA 00. GB2 Para ver un ejemplo, consulta the section called “Optimice las cargas de trabajo de la GPU con IMEX mediante GB2 00 instancias P6e”.

Estas técnicas pueden mejorar considerablemente la utilización de los recursos. Las organizaciones informan que la utilización de la GPU aumenta del 30 al 40% con la asignación tradicional al 80 o 90% con estrategias de uso compartido optimizadas. La elección de la técnica depende de las características de la carga de trabajo, los requisitos de aislamiento y las capacidades del hardware.

Optimice las cargas de trabajo de la GPU mediante la división del tiempo

La división del tiempo permite que varias cargas de trabajo compartan los recursos informáticos de la GPU al programarlos para que se ejecuten secuencialmente en la misma GPU física. Es ideal para cargas de trabajo de inferencia con un uso esporádico de la GPU.

Realice los siguientes pasos.

1. Defina una `ResourceClaimTemplate` división temporal con un archivo llamado: `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. Defina un pod dividiendo el tiempo con un archivo llamado: `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. Aplica la plantilla y el pod:

```
kubectl apply -f timeslicing-claim-template.yaml
kubectl apply -f timeslicing-pod.yaml
```

4. Supervise las solicitudes de recursos:

```
kubectl get resourceclaims -n timeslicing-gpu -w
```

   A continuación, se muestra un ejemplo de la salida:

```
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
```

Primer pod (`inference-pod-1`)

- Estado: `allocated,reserved`
- Significado: DRA encontró una GPU disponible y la reservó para este pod
- Estado del pod: comienza a funcionar inmediatamente

Segundo pod (`training-pod-2`)

- Estado: `pending`
- Significado: Esperando a que DRA configure la segmentación del tiempo en la misma GPU
- Estado del pod: En espera de ser programado
- El estado irá de `pending` a `allocated,reserved` a `running`

Optimice las cargas de trabajo de la GPU con MPS

El servicio multiproceso (MPS) permite la ejecución simultánea de varios contextos CUDA en una sola GPU con un mejor aislamiento que la división temporal.

Realice los siguientes pasos.

1. Defina un archivo `ResourceClaimTemplate` para MPS con el nombre`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. Defina un pod mediante MPS con un archivo llamado`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. Aplica la plantilla y crea varios pods MPS:

```
kubectl apply -f mps-claim-template.yaml
kubectl apply -f mps-pod.yaml
```

4. Supervise las solicitudes de recursos:

```
kubectl get resourceclaims -n mps-gpu -w
```

A continuación, se muestra un ejemplo de la salida:

```
NAME                                                STATE                AGE
mps-multi-container-pod-shared-gpu-claim-2p9kx   allocated,reserved   86s
```

Esta configuración demuestra un verdadero uso compartido de la GPU mediante el servicio multiproceso (MPS) de NVIDIA mediante la asignación dinámica de recursos (DRA). A diferencia de la segmentación temporal, en la que las cargas de trabajo se turnan para utilizar la GPU de forma secuencial, MPS permite que ambos contenedores se ejecuten simultáneamente en la misma GPU física. La idea clave es que el uso compartido de MPS con DRA requiere varios contenedores dentro de un único pod, no varios pods separados. Cuando se implementa, el controlador DRA asigna uno `ResourceClaim` al pod y configura automáticamente el MPS para permitir que los contenedores de inferencia y de entrenamiento se ejecuten simultáneamente.

Cada contenedor tiene su propio espacio de memoria de GPU y recursos de cómputo aislados, y el daemon MPS coordina el acceso al hardware subyacente. Puede comprobar que esto funciona de la siguiente manera:

- Comprobando`nvidia-smi`, que mostrará ambos contenedores como procesos M+C (`MPS + Compute`) que comparten el mismo dispositivo de GPU.
- Supervisión de los registros de ambos contenedores, que mostrarán marcas de tiempo intercaladas que demuestran la ejecución simultánea.

Este enfoque maximiza el uso de la GPU al permitir que las cargas de trabajo complementarias compartan el costoso hardware de la GPU de manera eficiente, en lugar de dejarlo infrautilizado por un solo proceso.

Contenedor 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 |
+-----------------------------------------------------------------------------------------
+
```

Contenedor 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 |
+-----------------------------------------------------------------------------------------
+
```

Optimice las cargas de trabajo de la GPU con una GPU de instancias múltiples

La GPU de varias instancias (MIG) proporciona particiones a nivel de hardware, lo que crea instancias de GPU aisladas con recursos de cómputo y memoria dedicados.

El uso de particiones MIG dinámicas con varios perfiles requiere el operador de GPU NVIDIA. El operador de GPU de NVIDIA usa MIG Manager para crear perfiles MIG y reinicia las instancias de la GPU, como P4D, P4De, P5, P6 y más, para aplicar los cambios de configuración. El GPU Operator incluye funciones integrales de administración de MIG a través del componente MIG Manager, que vigila los cambios en las etiquetas de los nodos y aplica automáticamente la configuración MIG adecuada. Cuando se solicita un cambio en el perfil MIG, el operador cierra correctamente todos

los clientes de la GPU, aplica la nueva geometría de partición y reinicia los servicios afectados. Este proceso requiere el reinicio del nodo de las instancias de GPU para garantizar que las transiciones de estado de la GPU sean limpias. Por eso, habilitar la configuración `WITH0REBOOT=true` de MIG Manager es esencial para que las implementaciones de MIG sean exitosas.

Necesita el controlador DRA de NVIDIA y el operador de GPU de NVIDIA para trabajar con MIG en Amazon EKS. Además, no necesitas el complemento de dispositivo NVIDIA ni el exportador DCGM, ya que forman parte del operador de GPU de NVIDIA. Como las EKS NVIDIA AMIs vienen con los controladores NVIDIA preinstalados, hemos deshabilitado el despliegue de controladores por parte del operador de la GPU para evitar conflictos y aprovechar los controladores optimizados que ya estaban presentes en las instancias. El controlador DRA de NVIDIA gestiona la asignación dinámica de recursos para las instancias MIG, mientras que el operador de GPU gestiona todo el ciclo de vida de la GPU. Esto incluye la configuración MIG, la funcionalidad de los complementos del dispositivo, la supervisión mediante DCGM y la detección de características de los nodos. Este enfoque integrado proporciona una solución completa para la administración de GPU empresariales, con capacidades de aislamiento a nivel de hardware y asignación dinámica de recursos.

Paso 1: Implemente NVIDIA GPU Operator

1. Añada el repositorio NVIDIA GPU Operator:

```
helm repo add nvidia https://nvidia.github.io/gpu-operator
helm repo update
```

2. Cree un `gpu-operator-values.yaml` archivo:

```
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. Instale GPU Operator mediante el `gpu-operator-values.yaml` archivo:

```
helm install gpu-operator nvidia/gpu-operator \
  --namespace gpu-operator \
  --create-namespace \
  --version v25.3.1 \
  --values gpu-operator-values.yaml
```

Este diagrama de Helm despliega los siguientes componentes y varios perfiles MIG:

- Complemento de dispositivo (programación de recursos de GPU)
- Exportador de DCGM (métricas y monitoreo de la GPU)
- Node Feature Discovery (NFD: etiquetado de hardware)
- Descubrimiento de funciones de GPU (GFD: etiquetado específico de la GPU)
- MIG Manager (particionamiento de GPU de varias instancias)
- Container Toolkit (tiempo de ejecución de contenedores de GPU)
- Operator Controller (gestión del ciclo de vida)

4. Verifique los módulos de despliegue:

```
kubectl get pods -n gpu-operator
```

A continuación, se muestra un ejemplo de la salida:

```
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. Cree un clúster de Amazon EKS con un grupo de nodos gestionado por P4de para probar los ejemplos de MIG:

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: dra-eks-cluster
  region: us-east-1
  version: '1.33'

managedNodeGroups:
# P4DE MIG Node Group with Capacity Block Reservation
- name: p4de-mig-nodes
  amiFamily: AmazonLinux2023
  instanceType: p4de.24xlarge

  # Capacity settings
  desiredCapacity: 0
  minSize: 0
  maxSize: 1

  # Use specific subnet in us-east-1b for capacity reservation
  subnets:
    - us-east-1b

  # AL2023 NodeConfig for RAID0 local storage only
  nodeadmConfig:
    apiVersion: node.eks.aws/v1alpha1
    kind: NodeConfig
    spec:
      instance:
        localStorage:
          strategy: RAID0

  # Node labels for MIG configuration
  labels:
    nvidia.com/gpu.present: "true"
    nvidia.com/gpu.product: "A100-SXM4-80GB"
    nvidia.com/mig.config: "p4de-half-balanced"
    node-type: "p4de"
    vpc.amazonaws.com/efa.present: "true"
    accelerator: "nvidia"
```

```
  # Node taints
  taints:
    - key: nvidia.com/gpu
      value: "true"
      effect: NoSchedule

  # EFA support
  efaEnabled: true

  # Placement group for high-performance networking
  placementGroup:
    groupName: p4de-placement-group
    strategy: cluster

  # Capacity Block Reservation (CBR)
  # Ensure CBR ID matches the subnet AZ with the Nodegroup subnet
  spot: false
  capacityReservation:
    capacityReservationTarget:
      capacityReservationId: "cr-abcdefghij"  # Replace with your capacity
 reservation ID
```

El operador de GPU de NVIDIA usa la etiqueta que se añade a los nodos `nvidia.com/mig.config: "p4de-half-balanced"` y particiona la GPU con el perfil indicado.

6. Inicie sesión en la `p4de` instancia.
7. Ejecuta el siguiente comando:

```
nvidia-smi -L
```

Deberías ver el siguiente resultado de ejemplo:

```
[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)
```

El operador de GPU de NVIDIA ha aplicado correctamente el perfil `p4de-half-balanced` MIG a tu instancia P4DE y ha creado particiones de GPU a nivel de hardware tal y como estaban configuradas. El particionamiento funciona de la siguiente manera:

El operador de la GPU aplicó esta configuración desde tu perfil MIG integrado:

```
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
```

A partir del `nvidia-smi -L` resultado, esto es lo que creó el operador de la GPU:

- Compatible con MiG GPUs (0-3): particionado por hardware
  - GPU 0: NVIDIA A100- -80 GB SXM4
    - Dispositivo MIG 3g.40 gb: cargas de trabajo grandes (40 GB de memoria, 42) SMs
    - Dispositivo 1 MIG de 2 GB y 20 GB: cargas de trabajo medianas (20 GB de memoria, 28) SMs
    - Dispositivo 2 MIG de 10 GB: cargas de trabajo pequeñas (10 GB de memoria, 14) SMs

    - Dispositivo 3 MIG de 1 GB y 10 GB: cargas de trabajo pequeñas (10 GB de memoria, 14) SMs
  - GPU 1: NVIDIA A100- -80 GB SXM4
    - Dispositivo MIG 3g.40gb 0: diseño de particiones idéntico
    - Dispositivo MIG 2g.20gb 1
    - Dispositivo MIG 1g.10gb 2
    - Dispositivo MIG 1g.10gb 3
  - GPU 2 y GPU 3: el mismo patrón que la GPU 0 y la GPU 1
- Completo GPUs (4-7): sin particiones MIG
  - GPU 4: NVIDIA A100- SXM4 -80 GB — GPU completa de 80 GB
  - GPU 5: NVIDIA A100- SXM4 -80 GB — GPU completa de 80 GB
  - GPU 6: NVIDIA A100- SXM4 -80 GB — GPU completa de 80 GB
  - GPU 7: NVIDIA A100- SXM4 -80 GB — GPU completa de 80 GB

Una vez que el operador de GPU de NVIDIA crea las particiones MIG, el controlador DRA de NVIDIA detecta automáticamente estas instancias aisladas por hardware y las pone a disposición para la asignación dinámica de recursos en Kubernetes. El controlador DRA detecta cada instancia MIG con su perfil específico (1 g.10 gb, 2 g.20 gb, 3 g.40 gb) y las expone como recursos programables según la clase de dispositivo. `mig.nvidia.com`

El controlador DRA monitorea continuamente la topología MIG y mantiene un inventario de todas las instancias disponibles. GPUs Cuando un pod solicita un perfil MIG específico a través de un`ResourceClaimTemplate`, el controlador DRA selecciona de forma inteligente una instancia MIG adecuada de cualquier GPU disponible, lo que permite una verdadera multitenencia a nivel de hardware. Esta asignación dinámica permite que varias cargas de trabajo aisladas se ejecuten simultáneamente en la misma GPU física y, al mismo tiempo, mantiene estrictos límites de recursos y garantías de rendimiento.

Paso 2: Probar la asignación de recursos de MIG

Ahora veamos algunos ejemplos para demostrar cómo DRA asigna dinámicamente las instancias MIG a diferentes cargas de trabajo. Implemente los módulos `resourceclaimtemplates` y pruébelos para ver cómo el controlador DRA coloca las cargas de trabajo en las particiones MIG disponibles, lo que permite que varios contenedores compartan los recursos de la GPU con aislamiento a nivel de hardware.

1. Crea para contener `mig-claim-template.yaml` el MIG: `resourceclaimtemplates`

```
apiVersion: v1
kind: Namespace
metadata:
  name: mig-gpu

---
# Template for 3g.40gb MIG instance (Large training)
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  name: mig-large-template
  namespace: mig-gpu
spec:
  spec:
    devices:
      requests:
      - name: mig-large
        deviceClassName: mig.nvidia.com
        selectors:
        - cel:
            expression: |
              device.attributes['gpu.nvidia.com'].profile == '3g.40gb'

---
# Template for 2g.20gb MIG instance (Medium training)
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  name: mig-medium-template
  namespace: mig-gpu
spec:
  spec:
    devices:
      requests:
      - name: mig-medium
        deviceClassName: mig.nvidia.com
        selectors:
        - cel:
            expression: |
              device.attributes['gpu.nvidia.com'].profile == '2g.20gb'
```

```
---
# Template for 1g.10gb MIG instance (Small inference)
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  name: mig-small-template
  namespace: mig-gpu
spec:
  spec:
    devices:
      requests:
      - name: mig-small
        deviceClassName: mig.nvidia.com
        selectors:
        - cel:
            expression: |
              device.attributes['gpu.nvidia.com'].profile == '1g.10gb'
```

2. Aplique las tres plantillas:

```
kubectl apply -f mig-claim-template.yaml
```

3. Ejecuta el siguiente comando:

```
kubectl get resourceclaimtemplates -n mig-gpu
```

   A continuación, se muestra un ejemplo de la salida:

```
NAME                 AGE
mig-large-template   71m
mig-medium-template  71m
mig-small-template   71m
```

4. Cree `mig-pod.yaml` para programar varios trabajos y aprovechar esto`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. Aplica esta especificación, que debería implementar tres pods:

```
kubctl apply -f mig-pod.yaml
```

   El controlador DRA debe programar estos pods.

6. Revisa los registros de los pods del controlador DRA y verás un resultado similar al siguiente:

```
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. Verifica el estado del pod `resourceclaims` para ver el estado del pod:

```
kubectl get resourceclaims -n mig-gpu -w
```

   A continuación, se muestra un ejemplo de la salida:

```
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
```

Como puedes ver, todos los pods pasaron de estar pendientes a estar pendientes `allocated,reserved` por el conductor del DRA.

8. Ejecute `nvidia-smi` desde el nodo. Observará que se están ejecutando tres procesadores Python:

```
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 |**
+-----------------------------------------------------------------------------------------
+
```

Optimice las cargas de trabajo de la GPU con IMEX mediante GB2 00 instancias P6e

El IMEX (Internode Memory Exchange) permite una comunicación coherente con la memoria entre los nodos para una formación distribuida en NVIDIA 00. GB2 UltraServers

Realice los siguientes pasos.

1. Defina un entrenamiento `ComputeDomain` para varios nodos con un archivo llamado`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. Defina un pod mediante canales IMEX con un archivo llamado: `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**
>
> Esto requiere 00 instancias de P6e. GB2

3. Implemente IMEX aplicando las plantillas y: `ComputeDomain`

```
kubectl apply -f imex-claim-template.yaml
kubectl apply -f imex-compute-domain.yaml
kubectl apply -f imex-pod.yaml
```

4. Compruebe el `ComputeDomain` estado.

```
kubectl get computedomain distributed-training-domain
```

5. Supervise el despliegue del daemon IMEX.

```
kubectl get pods -n nvidia-dra-driver -l resource.nvidia.com/computeDomain
```

6. Comprueba los canales IMEX del pod:

```
kubectl exec imex-distributed-training -- ls -la /dev/nvidia-caps-imex-channels/
```

7. Consulta los registros del Pod:

```
kubectl logs imex-distributed-training
```

   A continuación se muestra un ejemplo del resultado esperado:

```
=== 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
```

Para obtener más información, consulta el ejemplo de NVIDIA en GitHub.

Red

Considere un mayor ancho de banda de red o un adaptador de estructura elástica para aplicaciones con alta comunicación entre nodos

Para cargas de trabajo de entrenamiento distribuidas en Amazon EKS con altas demandas de comunicación entre nodos, considere la posibilidad de seleccionar instancias con mayor ancho de banda de red o Elastic Fabric Adapter (EFA). Un rendimiento de red insuficiente puede obstaculizar la transferencia de datos y ralentizar las tareas de aprendizaje automático, como el entrenamiento distribuido con varias GPU. Tenga en cuenta que las cargas de trabajo de inferencia no suelen tener una alta comunicación entre nodos.

Ejemplo

Por ejemplo, con 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
```

Asegúrese de que herramientas como MPI y NCCL estén instaladas en la imagen de su contenedor para aprovechar la EFA en trabajos de formación.

Aumente la cantidad de direcciones IP disponibles para acelerar los tiempos de lanzamiento de los pods

En EKS, cada pod necesita una dirección IP del bloque CIDR de la VPC. A medida que el clúster se amplía con más nodos y pods, se corre el riesgo de que se agoten las direcciones IP o se reduzca el rendimiento, pero habilitar la delegación de prefijos puede mitigar estos problemas al asignar previamente los rangos de IP y reducir las llamadas a la EC2 API, lo que se traduce en tiempos de lanzamiento de los pods más rápidos y en una mejor escalabilidad.

Al habilitar la delegación de prefijos después de crear el clúster, la interfaz de red de contenedores (CNI) de la VPC puede asignar prefijos IP (/28, cada uno con 16 direcciones IP) a las interfaces de red de las instancias. EC2 Esto significa que cada nodo puede admitir más pods, lo que reduce el riesgo de escasez de IP. Por ejemplo, en una `c5.4xlarge` instancia, puedes admitir hasta 110 pods con delegación de prefijos.

Si bien la delegación de prefijos es fundamental para optimizar el uso de la IP en entornos con muchos pods pequeños, AI/ML las cargas de trabajo suelen utilizar menos pods y más grandes (por ejemplo, un pod por GPU). Al habilitar la delegación de prefijos, la CNI de la VPC puede preasignar un prefijo para acelerar el inicio del pod al mantener una piscina caliente. Esto significa que las direcciones IP están fácilmente disponibles, lo que reduce el tiempo necesario para la inicialización del pod en comparación con la asignación bajo demanda en el modo sin prefijo. En estos casos, los ahorros en IP que se obtienen al permitir la delegación de prefijos ofrecen beneficios de rendimiento

para las cargas de trabajo. AI/ML AI reducir el número de llamadas a la EC2 API necesarias para configurar las direcciones IP y asignar previamente los rangos de IP, el uso de la delegación de prefijos permite lanzar los pods más rápidamente, lo que resulta especialmente beneficioso para escalar las cargas de trabajo con rapidez. AI/ML

Para habilitar la delegación de prefijos:

```
kubectl set env daemonset/aws-node -n kube-system ENABLE_PREFIX_DELEGATION=true
```

Garantice una planificación adecuada de las subredes de VPC para evitar el agotamiento de las direcciones IP, especialmente en implementaciones de gran tamaño, y gestione los bloques de CIDR para evitar superposiciones. VPCs Para obtener más información, consulte Optimización del uso de direcciones IP y Asignación de más direcciones IP a los nodos de Amazon EKS con prefijos.

Seguridad

Seguridad y conformidad

Considere S3 con KMS para un almacenamiento compatible con el cifrado

A menos que especifique lo contrario, todos los depósitos de S3 utilizan SSE-S3 de forma predeterminada para cifrar los objetos en reposo. Sin embargo, puede optar por configurar los buckets para que utilicen el cifrado del lado del servidor con claves de AWS Key Management Service (AWS KMS) (SSE-KMS) en su lugar. Los controles de seguridad de AWS KMS pueden ayudarle a cumplir los requisitos de conformidad relacionados con el cifrado. Puede usar estas claves de KMS para proteger sus datos en buckets de Amazon S3. Cuando utiliza el cifrado SSE-KMS con un bucket de S3, las claves de AWS KMS deben estar en la misma región que el bucket.

Configure sus buckets de uso general para utilizar las claves de bucket de S3 para SSE-KMS, a fin de reducir los costes de las solicitudes de AWS KMS hasta en un 99 por ciento al reducir el tráfico de solicitudes de Amazon S3 a AWS KMS. Las claves de bucket de S3 siempre están habilitadas `GET` y `PUT` funcionan en un bucket de directorio, por lo que no se pueden deshabilitar.

Tenga en cuenta que Amazon S3 Express One Zone utiliza un tipo específico de depósito denominado depósito de directorio S3. Los depósitos de directorio son exclusivamente para la clase de almacenamiento S3 Express One Zone y permiten un acceso de alto rendimiento y baja latencia. Para configurar el cifrado de bucket predeterminado en un bucket de directorio de S3, utilice la AWS CLI y especifique el ARN o ID de clave de KMS, no el alias, como en el siguiente ejemplo:

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"}}]}'
```

Asegúrese de que la función de IAM de su pod EKS tenga permisos de KMS (por ejemplo`kms:Decrypt`) para acceder a objetos cifrados. Pruébelo en un entorno provisional cargando un modelo de muestra en el depósito, montándolo en un módulo (p. ej., mediante el controlador CSI de Mountpoint S3) y verificando que el módulo pueda leer los datos cifrados sin errores. Audite los registros a través de AWS CloudTrail para confirmar el cumplimiento de los requisitos de cifrado. Consulte la documentación de KMS para obtener información sobre la configuración y la administración de claves.

Almacenamiento

Administración y almacenamiento de datos

Implemente modelos de IA en los pods mediante un controlador CSI

Las cargas de trabajo de inteligencia artificial y aprendizaje automático suelen requerir el acceso a artefactos de modelos de gran tamaño (por ejemplo, pesos entrenados o configuraciones), y los pods necesitan una forma confiable y escalable de acceder a ellos sin necesidad de incorporarlos en imágenes de contenedores, lo que puede aumentar el tamaño de las imágenes y los tiempos de extracción del registro de contenedores. Para reducir la sobrecarga operativa que supone gestionar los montajes de volúmenes, recomendamos implementar modelos de IA en los pods montando los servicios de almacenamiento de Amazon (p. ej., S3, FSx para Lustre, FSx para OpenZFS, EFS) como volúmenes persistentes PVs () utilizando sus respectivos controladores CSI. Para obtener información detallada sobre la implementación, consulte los temas siguientes de esta sección.

Optimice el almacenamiento de las cachés de modelos ML en EKS

Aprovechar una solución de almacenamiento óptima es fundamental para minimizar la latencia de inicio de los módulos y las aplicaciones, reducir el uso de memoria, obtener los niveles de rendimiento deseados para acelerar las cargas de trabajo y garantizar la escalabilidad de las cargas de trabajo de aprendizaje automático. Las cargas de trabajo de aprendizaje automático

suelen basarse en archivos modelo (pesos), que pueden ser de gran tamaño y requieren un acceso compartido a los datos entre los módulos o nodos. La selección de la solución de almacenamiento óptima depende de las características de la carga de trabajo, como la eficiencia de un solo nodo, el acceso a varios nodos, los requisitos de latencia, las restricciones de costes y también los requisitos de integración de datos (por ejemplo, con un repositorio de datos de Amazon S3). Recomendamos comparar diferentes soluciones de almacenamiento con sus cargas de trabajo para saber cuál cumple con sus requisitos, y le ofrecemos las siguientes opciones para ayudarlo a realizar una evaluación en función de sus requisitos de carga de trabajo.

El controlador CSI de EKS es compatible con los siguientes servicios de almacenamiento de AWS, cada uno con su propio controlador de CSI y tiene sus propias ventajas para los flujos de trabajo de IA y aprendizaje automático:

- Mountpoint para Amazon S3
- Amazon FSx para Lustre
- Amazon FSx para OpenZFS
- Amazon EFS
- Amazon EBS

La elección del servicio de almacenamiento de AWS depende de la arquitectura de implementación, la escala, los requisitos de rendimiento y la estrategia de costes. Los controladores CSI de almacenamiento deben estar instalados en el clúster de EKS, lo que permite al controlador CSI crear y gestionar volúmenes persistentes (PV) fuera del ciclo de vida de un pod. Con el controlador CSI, puede crear definiciones de PV de los servicios de almacenamiento de AWS compatibles como recursos de clúster de EKS. Luego, los pods pueden acceder a estos volúmenes de almacenamiento para sus volúmenes de datos mediante la creación de una notificación de volumen persistente (PVC) para el PV. Según el servicio de almacenamiento de AWS y el escenario de implementación, se puede conectar un único PVC (y el PV asociado) a varios pods para una carga de trabajo. Por ejemplo, para el entrenamiento de aprendizaje automático, los datos de entrenamiento compartidos se almacenan en un PV y varios pods acceden a ellos; para realizar inferencias en línea en tiempo real, los modelos LLM se almacenan en caché en un PV y se accede a ellos desde varios pods. A continuación, se proporcionan ejemplos de archivos YAML de PVC y PVC para los servicios de almacenamiento de AWS para ayudarle a empezar.

Supervisión del rendimiento Un rendimiento deficiente del disco puede retrasar las lecturas de las imágenes del contenedor, aumentar la latencia de inicio del pod y reducir el rendimiento de las inferencias o el entrenamiento. Utilice Amazon CloudWatch para supervisar las métricas de

rendimiento de sus servicios de almacenamiento de AWS. Cuando identifique los cuellos de botella en el rendimiento, modifique los parámetros de configuración del almacenamiento para optimizarlo.

Escenario: carga de trabajo de varias instancias de GPU

Amazon FSx for Lustre: en situaciones en las que tiene un entorno de instancias de procesamiento de varias EC2 GPU con cargas de trabajo dinámicas sensibles a la latencia y con un alto rendimiento de ancho de banda, como la formación distribuida y el servicio de modelos, y necesita una integración nativa del repositorio de datos de Amazon S3, le recomendamos Amazon for Lustre. FSx FSx for Lustre proporciona un sistema de archivos paralelo de alto rendimiento totalmente gestionado que está diseñado para cargas de trabajo con uso intensivo de cómputo, como la computación de alto rendimiento (HPC) y Machine Learning.

Puede instalar el controlador CSI de FSx for Lustre para montar FSx los sistemas de archivos en EKS como un volumen persistente (PV) y, a continuación, implementarlo FSx para el sistema de archivos Lustre como una caché de alto rendimiento independiente o como un sistema de archivos vinculado a S3 para que actúe como una caché de alto rendimiento para los datos de S3, lo que proporciona un rápido I/O y alto rendimiento para el acceso a los datos en todas las instancias de cómputo de la GPU. FSx for Lustre se puede implementar con las opciones de almacenamiento Scratch-SSD o Persistent-SSD:

- Almacenamiento Scratch-SSD: se recomienda para cargas de trabajo efímeras o de corta duración (horas), con una capacidad de rendimiento fija por TIB aprovisionado.
- Almacenamiento SSD persistente: recomendado para cargas de trabajo críticas y de larga duración que requieren el mayor nivel de disponibilidad, por ejemplo, simulaciones de HPC, análisis de macrodatos o formación en Machine Learning. Con el almacenamiento SSD persistente, puede configurar tanto la capacidad de almacenamiento como la capacidad de rendimiento (por TIB) que se requieren.

Consideraciones sobre el rendimiento:

- Módulo administrativo FSx para gestionar el sistema de archivos Lustre: configure un módulo «administrativo» que tenga el cliente lustre instalado y el sistema de FSx archivos montado. Esto habilitará un punto de acceso para ajustar el sistema de FSx archivos y también en situaciones en las que necesite precalentar el sistema de FSx archivos con sus datos de entrenamiento de aprendizaje automático o modelos LLM antes de iniciar las instancias de cómputo de la GPU. Esto es especialmente importante si su arquitectura utiliza instancias de GPU/Compute de EC2 Amazon basadas en Spot, donde puede utilizar el pod administrativo para «calentar» o «precargar» los

datos deseados en FSx el sistema de archivos, de modo que estén listos para procesarse cuando ejecute sus instancias de Amazon basadas en Spot. EC2

- Elastic Fabric Adapter (EFA): los tipos de implementación de almacenamiento SSD persistente admiten el Elastic Fabric Adapter (EFA), donde el uso del EFA es ideal para cargas de trabajo basadas en GPU de alto rendimiento y rendimiento. Tenga en cuenta que, FSx para Lustre, es compatible con el GPUDirect almacenamiento NVIDIA (GDS), donde el GDS es una tecnología que crea una ruta de datos directa entre el almacenamiento local o remoto y la memoria de la GPU, para permitir un acceso más rápido a los datos.
- Compresión: habilite la compresión de datos en el sistema de archivos si tiene tipos de archivos que se puedan comprimir. Esto puede ayudar a aumentar el rendimiento, ya que la compresión de datos reduce la cantidad de datos que se transfieren entre FSx los servidores de archivos Lustre y el almacenamiento.
- Configuración de división del sistema de archivos Lustre:
  - División de datos: permite FSx a Luster distribuir los datos de un archivo entre varios destinos de almacenamiento de objetos (OSTs) dentro de un sistema de archivos Lustre, lo que maximiza el acceso paralelo y el rendimiento, especialmente para trabajos de formación de aprendizaje automático a gran escala.
  - Sistema de archivos independiente: de forma predeterminada, se crea automáticamente una configuración de división de Lustre de 4 componentes mediante la función de diseño progresivo de archivos (PFL) de for Lustre. FSx En la mayoría de los casos, no es necesario actualizar el número o tamaño predeterminados de las franjas de PFL Lustre. Si necesita ajustar la división de datos de Lustre, puede ajustarla manualmente consultando los parámetros de división de un sistema de archivos de Lustre. FSx
  - Sistema de archivos vinculado a S3: los archivos importados al sistema de FSx archivos mediante la integración nativa de Amazon S3 (Data Repository Association o DRA) no utilizan el diseño de PFL predeterminado, sino que utilizan el diseño del parámetro del sistema de archivos. `ImportedFileChunkSize` Los archivos importados en S3 con un tamaño superior al 1 se `ImportedFileChunkSize` almacenarán en varios archivos OSTs con un recuento de franjas basado en el valor `ImportedFileChunkSize` definido (por defecto, 1 GiB). Si tiene archivos de gran tamaño, le recomendamos ajustar este parámetro a un valor superior.
  - Ubicación: Implemente un sistema de archivos FSx for Lustre en la misma zona de disponibilidad que sus nodos de procesamiento o GPU para permitir el acceso a los datos con la latencia más baja y evitar los patrones de acceso entre zonas de disponibilidad. Si tiene varios nodos de GPU ubicados en distintas zonas de disponibilidad, le recomendamos implementar

un sistema de FSx archivos en cada zona de disponibilidad para acceder a los datos con baja latencia.

Ejemplo

Definición de volumen persistente (PV) para un sistema de archivos FSx para Lustre, mediante aprovisionamiento estático (cuando la FSx instancia ya se ha aprovisionado).

```
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]
```

Ejemplo

Definición de reclamación de volumen persistente para PV denominada: `fsx-pv`

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: fsx-claim
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
```

```
      storage: 1200Gi
  volumeName: fsx-pv
```

Ejemplo

Configure un pod para que utilice una declaración de volumen persistente de`fsx-claim`:

```
apiVersion: v1
kind: Pod
metadata:
  name: fsx-app
spec:
  containers:
  - name: app
    image: amazonlinux:2023
    command: ["/bin/sh"]
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: fsx-claim
```

Para ver ejemplos completos, consulte los ejemplos FSx de controladores Lustre en GitHub. Supervisa las métricas FSx de rendimiento de Amazon Lustre con Amazon CloudWatch. Cuando se identifiquen cuellos de botella en el rendimiento, ajuste los parámetros de configuración según sea necesario.

Escenario: carga de trabajo de una sola instancia de GPU

Mountpoint para Amazon S3 con controlador CSI: puede montar un bucket S3 como volumen en sus pods mediante el controlador CSI de Mountpoint para Amazon S3. Este método permite un control de acceso detallado sobre qué pods pueden acceder a buckets S3 específicos. Cada pod tiene su propia instancia de punto de montaje y caché local (de 5 a 10 GB), lo que aísla el rendimiento de carga y lectura de modelos entre los pods. Esta configuración admite la autenticación a nivel de pod con funciones de IAM para cuentas de servicio (IRSA) y el control de versiones de modelos independientes para diferentes modelos o clientes. La contrapartida es el aumento del uso de la memoria y del tráfico de la API, ya que cada módulo emite llamadas a la API de S3 y mantiene su propia caché.

Ejemplo parcial de implementación de un pod (YAML) con un controlador CSI:

```
# CSI driver dynamically mounts the S3 bucket for each pod

volumes:
  - name: s3-mount
    csi:
      driver: s3.csi.aws.com
      volumeAttributes:
        bucketName: your-s3-bucket-name
        mountOptions: "--allow-delete"  # Optional
        region: us-west-2

containers:
  - name: inference
    image: your-inference-image
    volumeMounts:
      - mountPath: /models
        name: s3-mount
volumeMounts:
  - name: model-cache
    mountPath: /models
volumes:
  - name: model-cache
    hostPath:
      path: /mnt/s3-model-cache
```

Consideraciones de rendimiento:

- Almacenamiento en caché de datos: Mountpoint para S3 puede almacenar contenido en caché para reducir los costos y mejorar el rendimiento en caso de lecturas repetidas del mismo archivo. Consulte la configuración del almacenamiento en caché para conocer las opciones y los parámetros de almacenamiento en caché.
- Tamaño parcial del objeto: cuando almacene archivos de más de 72 GB y acceda a ellos, consulte Configuración del rendimiento de Mountpoint para saber cómo configurar los parámetros de la `--write-part-size` línea de comandos `--read-part-size` y de forma que se ajusten a sus requisitos de perfil de datos y carga de trabajo.
- La caché compartida está diseñada para objetos de hasta 1 MB de tamaño. No admite objetos grandes. Utilice la opción de caché local para almacenar objetos en caché NVMe o volúmenes de EBS en el nodo EKS.

- Cargos por solicitud de API: al realizar un gran número de operaciones de archivos con el Mountpoint para S3, los cargos por solicitud de API pueden convertirse en una parte de los costos de almacenamiento. Para mitigar esta situación, si no se requiere una coherencia sólida, habilite siempre el almacenamiento en caché de los metadatos y establezca un `metadata-ttl` período para reducir el número de operaciones de la API a S3.

Para obtener más información, consulte el controlador CSI de Mountpoint for Amazon S3 en la documentación oficial de Amazon EKS. Recomendamos monitorizar las métricas de rendimiento de Amazon S3 con las CloudWatch métricas de Amazon en caso de que se produzcan cuellos de botella y ajustar la configuración cuando sea necesario.

Almacenamiento compartido persistente de Amazon FSx para OpenZFS

Para escenarios que implican varias instancias de procesamiento de EC2 GPU con cargas de trabajo sensibles a la latencia que requieren alta disponibilidad, alto rendimiento, sensibilidad a los costes e implementaciones de varios pods para diferentes aplicaciones, recomendamos Amazon para OpenZFS. FSx Algunos ejemplos de cargas de trabajo incluyen la inferencia en tiempo real, el aprendizaje por refuerzo y el entrenamiento de redes generativas antagónicas. FSx para OpenZFS es particularmente beneficioso para las cargas de trabajo que necesitan un acceso de alto rendimiento a una estructura de directorios específica con archivos pequeños que utilizan patrones de acceso a datos de E/S pequeños. Además, OpenZFS proporciona la flexibilidad necesaria FSx para escalar el rendimiento de forma independiente de la capacidad de almacenamiento, lo que le ayuda a lograr una rentabilidad óptima al adaptar el tamaño del almacenamiento a las necesidades reales y, al mismo tiempo, mantener los niveles de rendimiento requeridos

El controlador CSI nativo FSx de OpenZFS permite la creación de varios sistemas de archivos o uno solo PVCs mediante la creación de varios volúmenes. Esto reduce la sobrecarga de administración y maximiza la utilización del rendimiento y las IOPS del sistema de archivos mediante la implementación consolidada de los módulos de aplicaciones en un solo sistema de archivos. Además, incluye funciones empresariales, como las instantáneas sin copias, los clones sin copias y las cuotas de usuarios y grupos, que se pueden aprovisionar de forma dinámica mediante el controlador CSI.

FSx para OpenZFS, admite tres tipos de implementación diferentes en el momento de su creación:

- Single-AZ: la opción más económica con latencias inferiores a un milisegundo, pero no proporciona alta disponibilidad a nivel del sistema de archivos o de la zona de disponibilidad. Se

recomienda para cargas de trabajo de desarrollo y prueba o para aquellas que tienen una alta disponibilidad en la capa de aplicación.

- Single-AZ (HA): proporciona alta disponibilidad a nivel de sistema de archivos con latencias inferiores a un milisegundo. Se recomienda para cargas de trabajo de alto rendimiento que requieren alta disponibilidad.
- Multi-AZ: proporciona alta disponibilidad a nivel de sistema de archivos y en todas las zonas de disponibilidad. Se recomienda para cargas de trabajo de alto rendimiento que requieren disponibilidad adicional en todas las zonas de disponibilidad.

Consideraciones sobre el rendimiento:

- Tipo de implementación: si la disponibilidad adicional en todas las zonas de disponibilidad no es un requisito, considere usar el tipo de implementación Single-AZ (HA). Este tipo de implementación proporciona hasta el 100% del rendimiento de las escrituras, mantiene latencias inferiores a un milisegundo y los sistemas de archivos Gen2 tienen una NVMe memoria caché adicional para almacenar hasta terabytes de datos a los que se accede con frecuencia. Los sistemas de archivos Multi-AZ proporcionan hasta un 75% del rendimiento de las escrituras con una latencia mayor para adaptarse al tráfico entre zonas de disponibilidad.
- Rendimiento e IOPS: tanto el rendimiento como las IOPS configuradas para el sistema de archivos se pueden ampliar o reducir después de la implementación. Puede aprovisionar hasta un 10% del acceso a los datos en cachéGB/s of disk throughput providing up to 21GB/s. Las IOPS se pueden escalar hasta 400 000 desde el disco y la memoria caché puede proporcionar más de 1 millón de IOPS. Tenga en cuenta que la ampliación del rendimiento de un sistema de archivos Single-AZ provoca una breve interrupción del sistema de archivos, ya que no existe una alta disponibilidad. El escalado del rendimiento de un sistema de archivos Single-AZ (HA) o Multi-AZ se puede realizar de forma no disruptiva. Las IOPS del SSD se pueden escalar una vez cada seis horas.
- Clase de almacenamiento: FSx para OpenZFS, admite tanto la clase de almacenamiento SSD como la clase de almacenamiento Intelligent-Tiering. Para AI/ML las cargas de trabajo, se recomienda utilizar la clase de almacenamiento SSD, que proporciona un rendimiento uniforme a la carga de trabajo y mantiene las CPU/GPU lo más ocupadas posible.
- Compresión: habilite el algoritmo LZ4 de compresión si tiene una carga de trabajo que se pueda comprimir. Esto reduce la cantidad de datos que cada archivo consume en la memoria caché, lo que permite almacenar más datos directamente desde la memoria caché, ya que el rendimiento de la red y las IOPS reducen la carga en el disco SSD.

- Tamaño del registro: la mayoría de las AI/ML cargas de trabajo se beneficiarán de mantener el tamaño de registro predeterminado de 128 KB. Este valor solo debe reducirse si el conjunto de datos consta de archivos grandes (de más de 10 GiB) con acceso constante a bloques pequeños por debajo de los 128 KB desde la aplicación.

Una vez creado el sistema de archivos, el servicio crea automáticamente un volumen raíz asociado. Se recomienda almacenar los datos dentro de los volúmenes secundarios del volumen raíz del sistema de archivos. Con el FSx controlador CSI de OpenZFS, se crea una notificación de volumen persistente asociada para crear de forma dinámica el volumen secundario.

Ejemplos:

Definición de clase de almacenamiento (SC) FSx para un volumen de OpenZFS, que se utiliza para crear un volumen secundario del volumen raíz ($ROOT_VOL_ID) en un sistema de archivos existente y exportar el volumen al CIDR de VPC ($VPC_CIDR) mediante el protocolo NFS v4.2.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fsxz-vol-sc
provisioner: fsx.openzfs.csi.aws.com
parameters:
  ResourceType: "volume"
  ParentVolumeId: '"$ROOT_VOL_ID"'
  CopyTagsToSnapshots: 'false'
  DataCompressionType: '"LZ4"'
  NfsExports: '[{"ClientConfigurations": [{"Clients": "$VPC_CIDR", "Options":
 ["rw","crossmnt","no_root_squash"]}]}]'
  ReadOnly: 'false'
  RecordSizeKiB: '128'
  Tags: '[{"Key": "Name", "Value": "AI-ML"}]'
  OptionsOnDeletion: '["DELETE_CHILD_VOLUMES_AND_SNAPSHOTS"]'
reclaimPolicy: Delete
allowVolumeExpansion: false
mountOptions:
  - nfsvers=4.2
  - rsize=1048576
  - wsize=1048576
  - timeo=600
  - nconnect=16
  - async
```

Una reclamación de fsxz-vol-sc volumen persistente (PVC) creada de forma dinámica en función de la anterior. Tenga en cuenta que la capacidad de almacenamiento asignada es de 1 GB, algo obligatorio FSx para los volúmenes de OpenZFS, tal y como se indica en las preguntas frecuentes sobre los controladores CSI. El volumen dispondrá de toda la capacidad aprovisionada al sistema de archivos con esta configuración. Si es necesario restringir la capacidad del volumen, puede hacerlo mediante cuotas de usuarios o grupos.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: dynamic-vol-pvc
  namespace: example
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: fsxz-vol-sc
  resources:
    requests:
      storage: 1Gi
```

Configure un pod para montar un volumen mediante la declaración de volumen persistente (PVC) de dynamic-vol-pvc:

```
kind: Pod
apiVersion: v1
metadata:
  name: fsx-app
  namespace: example
spec:
  volumes:
    - name: dynamic-vol-pv
      persistentVolumeClaim:
        claimName: dynamic-vol-pvc
  containers:
    - name: app
      image: amazonlinux:2023
      command: ["/bin/sh"]
      volumeMounts:
        - mountPath: "/mnt/fsxz"
          name: dynamic-vol-pv
```

Amazon EFS para cachés de modelos compartidos

En los escenarios en los que tiene un entorno de instancias de procesamiento de varias EC2 GPU y cargas de trabajo dinámicas que requieren acceso a modelos compartidos en varios nodos y zonas de disponibilidad (por ejemplo, inferencia en línea en tiempo real con Karpenter) con necesidades moderadas de rendimiento y escalabilidad, le recomendamos utilizar un sistema de archivos Amazon Elastic File System (EFS) como volumen persistente a través del controlador CSI de EFS. Amazon EFS es un sistema de archivos NFS basado en la nube escalable, de alta disponibilidad y totalmente gestionado que permite que EC2 las instancias y los contenedores cuenten con almacenamiento de archivos compartido, con un rendimiento uniforme y en los que no es necesario aprovisionar el almacenamiento por adelantado. Utilice EFS como volumen modelo y monte el volumen como un sistema de archivos compartido mediante la definición de un volumen persistente en el clúster EKS. Cada notificación de volumen persistente (PVC) respaldada por un sistema de archivos EFS se crea como un punto de acceso de EFS al sistema de archivos EFS. EFS permite que varios nodos y pods accedan a los mismos archivos de modelo, lo que elimina la necesidad de sincronizar los datos con el sistema de archivos de cada nodo. Instale el controlador CSI de EFS para integrar EFS con EKS.

Puede implementar un sistema de archivos Amazon EFS con los siguientes modos de rendimiento:

- Rendimiento en ráfaga: escala el rendimiento en función del tamaño del sistema de archivos, lo que resulta adecuado para cargas de trabajo variables con ráfagas ocasionales.
- Rendimiento aprovisionado: rendimiento dedicado, ideal para trabajos de formación en aprendizaje automático consistentes con necesidades de rendimiento predecibles dentro de unos límites.
- Rendimiento elástico (recomendado para el aprendizaje automático): se escala automáticamente en función de la carga de trabajo y es rentable para diferentes cargas de trabajo de aprendizaje automático.

Para ver las especificaciones de rendimiento, consulte las especificaciones de rendimiento de Amazon EFS.

Consideraciones sobre el rendimiento:

- Usa Elastic Throughput para diferentes cargas de trabajo.
- Utilice la clase de almacenamiento estándar para las cargas de trabajo de aprendizaje automático activas.

Para ver ejemplos completos del uso del sistema de archivos Amazon EFS como volumen persistente en el clúster y los pods de EKS, consulte los ejemplos de controladores CSI de EFS en GitHub. Supervise las métricas de rendimiento de Amazon EFS con Amazon CloudWatch. Cuando se identifiquen cuellos de botella en el rendimiento, ajuste los parámetros de configuración según sea necesario.

Utilice S3 Express One Zone para flujos de trabajo orientados a objetos y sensibles a la latencia

Para AI/ML las cargas de trabajo sensibles a la latencia en Amazon EKS, como el entrenamiento de modelos a gran escala, la inferencia o el análisis de alto rendimiento, recomendamos usar S3 Express One Zone para almacenar y recuperar modelos de alto rendimiento. S3 Express One Zone ofrece un espacio de nombres jerárquico, similar a un sistema de archivos, en el que basta con subirlos a un depósito de directorios, lo que permite «guardar todo» y, al mismo tiempo, mantener una alta velocidad. Esto resulta especialmente útil si está acostumbrado a los flujos de trabajo orientados a objetos. Como alternativa, si está más acostumbrado a los sistemas de archivos (por ejemplo, compatibles con POSIX), puede preferir Amazon FSx for Lustre u OpenZFS. Amazon S3 Express One Zone almacena los datos en una única zona de disponibilidad (AZ) mediante buckets de directorio y ofrece una latencia más baja que los buckets S3 estándar, que distribuyen los datos entre varias. AZs Para obtener los mejores resultados, asegúrese de ubicar su equipo de EKS en la misma zona de disponibilidad que su depósito de Express One Zone. Para obtener más información sobre las diferencias de S3 Express One Zone, consulte Diferencias entre los depósitos de directorio.

Para acceder a S3 Express One Zone con la semántica del sistema de archivos, recomendamos utilizar el controlador CSI Mountpoint S3, que monta los depósitos de S3 (incluido Express One Zone) como un sistema de archivos local. Esto traduce las operaciones de archivos (por ejemplo, abrir, leer o escribir) en llamadas a la API de S3, lo que proporciona un acceso de alto rendimiento optimizado para cargas de trabajo de lectura intensivas desde varios clientes y escrituras secuenciales en nuevos objetos. Para obtener más información sobre las operaciones y limitaciones admitidas (p. ej., no cumplen totalmente con POSIX, pero Express One Zone admite la incorporación y el cambio de nombre), consulte la documentación de semántica de Mountpoint.

Ventajas de rendimiento

- Proporciona un acceso a los datos hasta 10 veces más rápido que el estándar S3, con una latencia uniforme de milisegundos de un solo dígito y una reducción de los costes de solicitud de hasta un 80%.

- Se amplía para gestionar de cientos de miles a millones de solicitudes por segundo por segmento de directorio, lo que evita las limitaciones o las interrupciones que se producen en el S3 estándar durante las cargas extremas (por ejemplo, en clústeres con decenas o cientos de miles de redes saturadas). GPUs/CPUs
- Utiliza un mecanismo de autenticación basado en sesiones. Autentíquese una vez para obtener un token de sesión y, a continuación, realice operaciones repetidas a alta velocidad sin sobrecargar la autenticación por solicitud. Esto está optimizado para cargas de trabajo como los puntos de control frecuentes o la carga de datos.

Casos de uso recomendados

- Almacenamiento en caché: uno de los principales casos de uso del controlador CSI Mountpoint S3 con S3 Express One Zone es el almacenamiento en caché. La primera instancia lee los datos de S3 Standard (uso general) y los almacena en caché en Express One Zone, de baja latencia. Las lecturas posteriores realizadas por otros clientes acceden más rápido a los datos almacenados en caché, lo que resulta ideal para situaciones con varios nodos de EKS en las que varios nodos de EKS leen los mismos datos (por ejemplo, conjuntos de datos de entrenamiento compartidos). Esto puede mejorar el rendimiento hasta 7 veces en el caso de los accesos repetidos y reducir los costes informáticos. Para las cargas de trabajo que requieren una conformidad total con POSIX (por ejemplo, bloqueo de archivos y modificaciones in situ), considere Amazon FSx for Lustre u OpenZFS como alternativas.
- AI/ML Capacitación e inferencia a gran escala: ideal para cargas de trabajo con cientos o miles de nodos de cómputo (por ejemplo, GPUs en clústeres de EKS) en los que la limitación del S3 de uso general podría provocar demoras y desperdiciar costosos recursos de cómputo. Por ejemplo, los investigadores de LLM o las organizaciones que utilizan un modelo diario se tests/checkpoints benefician de un acceso rápido y fiable sin interrumpir el S3 regional. Para cargas de trabajo de menor escala (p. ej., decenas de nodos), puede ser suficiente el S3 Standard u otras clases de almacenamiento.
- Canalizaciones de datos: Load/prepare modelos, archivos de datos de entrenamiento o puntos de control de transmisiones. Si su equipo prefiere el almacenamiento de objetos en lugar de los sistemas de archivos tradicionales (por ejemplo, debido a que está familiarizado con S3), utilícelo en lugar de introducir cambios de diseño para opciones compatibles con POSIX, como Lustre. FSx

Consideraciones

- Resiliencia: el diseño con una sola zona de disponibilidad ofrece una durabilidad del 99,19% (igual que el S3 estándar, mediante la redundancia dentro de la zona de disponibilidad), pero una disponibilidad inferior (diseño del 99,95% y SLA del 99,9%) en comparación con las clases con zonas de disponibilidad múltiples (disponibilidad del 99,99%). Es menos resistente a los fallos en las zonas de disponibilidad. Úselo para datos recreables o en caché. Considere la posibilidad de realizar copias de seguridad o replicaciones en zonas de disponibilidad múltiples (Multi-AZ) para cargas
- Soporte de API y funciones: admite un subconjunto de S3 APIs (por ejemplo, sin políticas de ciclo de vida ni replicación); puede requerir cambios menores en la aplicación para la autenticación de la sesión o el manejo de objetos.
- Integración con EKS: coloque su EKS pods/nodes en la misma zona de disponibilidad que el depósito del directorio para minimizar la latencia de la red. Utilice Mountpoint para Amazon S3 o los controladores CSI para el acceso nativo de Kubernetes.
- Pruebas: pruebe la latencia de recuperación en un entorno que no sea de producción para validar las mejoras de rendimiento. Supervisa la presencia de estrangulamientos en escenarios S3 estándar (p. ej., alta saturación de la GPU) y compara.

La clase de almacenamiento S3 Express One Zone está disponible en varias regiones y se integra con EKS para las cargas de trabajo que necesitan acceder a objetos sin tener que esperar a ser almacenadas. Para obtener más información, consulte Cómo empezar a usar S3 Express One Zone.

Observabilidad

Monitoreo y observabilidad

Explicación de las métricas de GPU

La métrica de utilización de la GPU muestra si la GPU realizó algún trabajo durante la ventana de muestra. Esta métrica captura el porcentaje de tiempo que la GPU ejecutó al menos una instrucción, pero no revela la eficiencia con la que la GPU utilizó su hardware. Una GPU contiene varios multiprocesadores de transmisión (SMs), que son las unidades de procesamiento en paralelo que ejecutan las instrucciones. Una lectura de utilización del 100% puede significar que la GPU ejecutó cargas de trabajo paralelas pesadas en todas sus SMs unidades, o puede significar que una sola instrucción pequeña activó la GPU durante el período de muestra. Para comprender el uso real, es necesario examinar las métricas de la GPU en varios niveles de la arquitectura del hardware. Cada multiprocesador de streaming está creado a partir de distintos tipos de núcleos y cada capa

presenta características de rendimiento diferentes. Las métricas de nivel superior (utilización de la GPU, utilización de la memoria, potencia de la GPU y temperatura de la GPU, visibles a través de nvidia-smi) muestran si el dispositivo está activo. Las métricas más detalladas (uso del SM, actividad del SM y uso del núcleo tensor) revelan la eficiencia con la que la GPU utiliza sus recursos.

Concéntrese en un uso elevado de la energía

La subutilización GPUs desperdicia la capacidad de cómputo y aumenta los costes porque las cargas de trabajo no utilizan todos los componentes de la GPU de forma simultánea. Para AI/ML las cargas de trabajo en Amazon EKS, realice un seguimiento del uso de energía de la GPU como proxy para identificar la actividad real de la GPU. El uso de la GPU indica el porcentaje de tiempo que la GPU ejecuta cualquier núcleo, pero no revela si los multiprocesadores de streaming, los controladores de memoria y los núcleos tensoriales están todos activos al mismo tiempo. El uso de energía pone de manifiesto esta brecha, ya que el hardware totalmente equipado consume mucha más energía que el hardware que ejecuta núcleos livianos o permanece inactivo entre tareas. Compare el consumo de energía con la potencia de diseño térmico (TDP) de la GPU para detectar la infrautilización y, a continuación, investigue si su carga de trabajo se ve obstaculizada por el preprocesamiento de la CPU, las E/S de la red o los tamaños de lote ineficientes.

CloudWatch Configure Container Insights en Amazon EKS para identificar los pods, los nodos o las cargas de trabajo con bajo consumo de energía de la GPU. Esta herramienta se integra directamente con Amazon EKS y le permite supervisar el consumo de energía de la GPU y ajustar la programación de los pods o los tipos de instancias cuando el consumo de energía cae por debajo de los niveles objetivo. Si necesita una visualización avanzada o paneles personalizados, utilice el exportador DCGM de NVIDIA con Prometheus y Grafana para la supervisión nativa de Kubernetes. Ambos enfoques muestran métricas clave de NVIDIA, como (consumo de energía de la GPU) y (temperatura de la GPU). `nvidia_smi_power_draw` `nvidia_smi_temperature_gpu` Para ver una lista de métricas, consulta https://docs.aws.amazon.com/AmazonCloudWatch/ latest/monitoring/CloudWatch -Agent-NVIDIA-GPU.htm. Busque patrones como el bajo consumo de energía constante durante horas específicas o para trabajos específicos. Estas tendencias le ayudan a identificar dónde consolidar las cargas de trabajo o ajustar la asignación de recursos.

Los límites estáticos de los recursos en Kubernetes (como el número de CPU, memoria y GPU) suelen provocar un aprovisionamiento excesivo o una subutilización, especialmente en el caso de las cargas de trabajo dinámicas AI/ML , como las inferencias, en las que la demanda fluctúa. Analice sus tendencias de uso y consolide las cargas de trabajo en menos cantidad. GPUs Asegúrese de que cada GPU alcance su máxima utilización antes de asignar otras adicionales. Este enfoque reduce el desperdicio y los costos. Para obtener una guía detallada sobre cómo optimizar las estrategias

de programación y uso compartido, consulte las prácticas recomendadas de cálculo y escalado automático de EKS

Observabilidad y métricas

Uso de herramientas de monitoreo y observabilidad para sus cargas de trabajo AI/ML

Los AI/ML servicios modernos requieren la coordinación entre la infraestructura, el modelado y la lógica de las aplicaciones. Los ingenieros de plataformas administran la infraestructura y el conjunto de observabilidad. Recopilan, almacenan y visualizan las métricas. AI/ML los ingenieros definen métricas específicas del modelo y supervisan el rendimiento en condiciones variables de carga y distribución de datos. Los desarrolladores de aplicaciones consumen APIs, direccionan las solicitudes y realizan un seguimiento de las métricas de nivel de servicio y las interacciones de los usuarios. Sin prácticas de observabilidad unificadas, estos equipos trabajan en silos y pasan por alto las señales críticas sobre el estado y el rendimiento del sistema. Establecer una visibilidad compartida en todos los entornos garantiza que todas las partes interesadas puedan detectar los problemas a tiempo y mantener un servicio fiable.

La optimización de los clústeres de Amazon EKS para AI/ML las cargas de trabajo presenta desafíos de supervisión únicos, especialmente en lo que respecta a la administración de la memoria de la GPU. Sin una supervisión adecuada, las organizaciones se enfrentan a errores out-of-memory (OOM), ineficiencias de recursos y costes innecesarios. Un monitoreo efectivo garantiza un mejor rendimiento, resiliencia y menores costos para los clientes de EKS. Utilice un enfoque holístico que combine tres capas de monitoreo. En primer lugar, supervise las métricas granulares de la GPU con el exportador DCGM de NVIDIA para realizar un seguimiento del consumo de energía de la GPU, la temperatura de la GPU, la actividad de la SM, la ocupación de la SM y los errores XID. En segundo lugar, supervise los marcos de servicio de inferencia, como Ray y vLLM, para obtener información sobre las cargas de trabajo distribuidas a través de sus métricas nativas. En tercer lugar, recopile información a nivel de aplicación para realizar un seguimiento de las métricas personalizadas específicas de su carga de trabajo. Este enfoque por capas le brinda visibilidad desde la utilización del hardware hasta el rendimiento de las aplicaciones.

Herramientas y marcos

Varias herramientas y marcos proporcionan out-of-the-box métricas nativas para monitorear las AI/ML cargas de trabajo. Estas métricas integradas eliminan la necesidad de una instrumentación personalizada y reducen el tiempo de configuración. Las métricas se centran en aspectos del rendimiento, como la latencia, el rendimiento y la generación de tokens, que son fundamentales

para la elaboración de inferencias y la evaluación comparativa. El uso de métricas nativas le permite comenzar a monitorear de inmediato sin crear canales de recopilación personalizados.

- vLLM: un motor de servidor de alto rendimiento para modelos de lenguaje de gran tamaño (LLMs) que proporciona métricas nativas, como la latencia de las solicitudes y el uso de memoria.
- Ray: un marco de computación distribuida que emite métricas para cargas de trabajo de IA escalables, incluidos los tiempos de ejecución de las tareas y la utilización de los recursos.
- Hugging Face Text Generation Inference (TGI): un conjunto de herramientas para implementar y LLMs servir, con métricas integradas para el rendimiento de las inferencias.
- NVIDIA genai-perf: una herramienta de línea de comandos para comparar modelos de IA generativa y medir el rendimiento, la latencia y las métricas específicas de la LLM, como las solicitudes completadas en intervalos de tiempo específicos.

Métodos de observabilidad

Recomendamos implementar cualquier mecanismo de observabilidad adicional de una de las siguientes maneras.

CloudWatch Container Insights Si su organización prefiere herramientas nativas de AWS con una configuración mínima, le recomendamos CloudWatch Container Insights. Se integra con el exportador DCGM de NVIDIA para recopilar métricas de la GPU y ofrece paneles prediseñados para obtener información rápida. Gracias a la instalación del complemento CloudWatch Observability en el clúster, Container Insights implementa y gestiona el ciclo de vida del exportador DCGM de NVIDIA, que recopila las métricas de GPU de los controladores de Nvidia y las expone a ellas. CloudWatch

Tras instalar Container Insights, detecta CloudWatch automáticamente a NVIDIA GPUs en su entorno y recopila métricas críticas de estado y rendimiento. Estas métricas aparecen en out-of-the-box paneles seleccionados. También puede integrar Ray y vLLM CloudWatch con el CloudWatch agente unificado para enviar sus métricas nativas. Este enfoque unificado simplifica la observabilidad en los entornos de EKS y permite a los equipos centrarse en el ajuste del rendimiento y la optimización de los costes en lugar de en crear una infraestructura de supervisión.

Para obtener una lista completa de las métricas disponibles, consulte las métricas de Amazon EKS y Kubernetes Container Insights. Para obtener step-by-step orientación sobre la implementación de la supervisión de GPU, consulte Obtenga información operativa para las cargas de trabajo de GPU de NVIDIA mediante Amazon CloudWatch Container Insights. Para ver ejemplos prácticos de cómo optimizar la latencia de inferencia, consulte Optimización de la capacidad de respuesta de la IA: una guía práctica sobre la inferencia optimizada para la latencia de Amazon Bedrock.

Prometheus y Grafana gestionados Si su organización necesita paneles personalizados y funciones de visualización avanzadas, implemente Prometheus con el exportador de DCGM de NVIDIA y Grafana para la supervisión nativa de Kubernetes. Prometheus extrae y almacena las métricas de la GPU del exportador de DCGM, mientras que Grafana proporciona capacidades flexibles de visualización y alerta. Este enfoque te da más control sobre el diseño del panel de control y la retención de métricas en comparación con Container Insights. CloudWatch

Puede ampliar este conjunto de monitoreo integrando marcos de código abierto como Ray y vLLM Ray y vLLM para exportar sus métricas nativas a Prometheus. También puede conectar Grafana a una fuente de datos de AWS X-Ray para visualizar las trazas distribuidas e identificar los cuellos de botella en el rendimiento en su proceso de inferencias. Esta combinación proporciona end-to-end visibilidad desde las métricas a nivel de la GPU hasta los flujos de solicitudes a nivel de la aplicación.

Para obtener step-by-step orientación sobre la implementación de esta pila de monitoreo, consulte Monitorización de cargas de trabajo de GPU en Amazon EKS mediante los servicios de código abierto gestionados por AWS.

Considere la posibilidad de monitorizar la formación básica y ajustar las métricas

Supervise las métricas de entrenamiento principales para realizar un seguimiento del estado y el rendimiento de su clúster de Amazon EKS y de las cargas de trabajo de aprendizaje automático que se ejecutan en él. Las cargas de trabajo de entrenamiento tienen requisitos de monitoreo diferentes a los de las cargas de trabajo de inferencia porque se ejecutan durante períodos prolongados, consumen los recursos de manera diferente y requieren visibilidad de la convergencia de modelos y la eficiencia de la canalización de datos. Las siguientes métricas le ayudan a identificar los cuellos de botella, optimizar la asignación de recursos y garantizar que los trabajos de formación se completen correctamente. Para obtener step-by-step orientación sobre la implementación de este enfoque de monitoreo, consulte Introducción a la observación de las cargas de trabajo de aprendizaje automático en Amazon EKS.

Métricas de uso de recursos

Supervise las métricas de uso de recursos para validar que sus recursos se consumen correctamente. Estas métricas le ayudan a identificar los cuellos de botella y las causas principales de los problemas de rendimiento.

- CPU, memoria, red, potencia de la GPU y temperatura de la GPU: supervise estas métricas para garantizar que los recursos asignados satisfagan las demandas de carga de trabajo e identificar las oportunidades de optimización. Realiza un seguimiento de métricas como

gpu_memory_usage_bytes para identificar los patrones de consumo de memoria y detectar los picos de uso. Calcula percentiles como el percentil 95 (P95) para comprender las demandas de memoria más altas durante el entrenamiento. Este análisis le ayuda a optimizar los modelos y la infraestructura para evitar errores de OOM y reducir los costes.

- Ocupación de SM, FPxx actividad de SM, actividad: supervise estas métricas para comprender cómo se utiliza el recurso subyacente de la GPU. Como regla general, márcate como meta 0,8 para SM Activity.
- Utilización de recursos de nodo y pod: realice un seguimiento del uso de los recursos a nivel de nodo y pod para identificar la contención de recursos y los posibles cuellos de botella. Supervise si los nodos se acercan a los límites de capacidad, lo que puede retrasar la programación de los módulos y ralentizar las tareas de entrenamiento.
- Utilización de recursos en comparación con las solicitudes y los límites: compare el uso real de los recursos con las solicitudes y los límites configurados para determinar si su clúster puede gestionar las cargas de trabajo actuales y adaptarse a las futuras. Esta comparación revela si es necesario ajustar las asignaciones de recursos para evitar errores de OOM o desperdiciar recursos.
- Métricas internas de los marcos de aprendizaje automático: capture las métricas internas de formación y convergencia de los marcos de aprendizaje automático, como TensorFlow y. PyTorch Estas métricas incluyen las curvas de pérdida, la tasa de aprendizaje, el tiempo de procesamiento por lotes y la duración de los pasos de entrenamiento. Visualice estas métricas con herramientas TensorBoard o herramientas similares para realizar un seguimiento de la convergencia de los modelos e identificar las ineficiencias de la formación.

Modele las métricas de rendimiento

Supervise las métricas de rendimiento de los modelos para validar que su proceso de formación produzca modelos que cumplan con los requisitos empresariales y de precisión. Estas métricas le ayudan a determinar cuándo interrumpir la formación, comparar las versiones de los modelos e identificar la degradación del rendimiento.

- Exactitud, precisión, recuperación y puntuación de F1: realice un seguimiento de estas métricas para comprender el rendimiento de su modelo con los datos de validación. Calcule la puntuación F1 en un conjunto de validación después de cada período de entrenamiento para evaluar si el modelo está mejorando y cuándo alcanza los niveles de rendimiento aceptables.
- Métricas específicas de la empresa y KPIs: defina y realice un seguimiento de las métricas que midan directamente el valor empresarial de sus iniciativas. AI/ML En el caso de un sistema de

recomendaciones, haz un seguimiento de métricas como la tasa de clics o la tasa de conversión para asegurarte de que el modelo genere los resultados empresariales esperados.

- Rendimiento a lo largo del tiempo: compare las métricas de rendimiento entre las versiones del modelo y las sesiones de entrenamiento para identificar las tendencias y detectar la degradación. Controle si las versiones más recientes de los modelos mantienen o mejoran el rendimiento en comparación con los modelos de referencia. Esta comparación histórica le ayuda a decidir si desea implementar nuevos modelos o investigar los problemas de formación.

Calidad de los datos y métricas de desviación

Supervisa la calidad de los datos y las métricas de desviación para asegurarte de que los datos de entrenamiento sean consistentes y representativos. La desviación de los datos puede provocar que el rendimiento del modelo se degrade con el tiempo, mientras que los problemas de calidad de los datos pueden impedir la convergencia de los modelos o producir resultados poco fiables.

- Propiedades estadísticas de los datos de entrada: realice un seguimiento de las propiedades estadísticas, como la media, la desviación estándar y la distribución de las entidades de entrada a lo largo del tiempo, para detectar desviaciones o anomalías en los datos. Supervisa si las distribuciones de las funciones varían significativamente con respecto a tus datos de entrenamiento de referencia. Por ejemplo, si la media de una característica crítica cambia en más de dos desviaciones estándar, investigue si su flujo de datos ha cambiado o si la fuente de datos subyacente ha cambiado.
- Detección de desviaciones de datos y alertas: implemente mecanismos automatizados para detectar y alertar sobre los problemas de calidad de los datos antes de que afecten a la formación. Usa pruebas estadísticas, como la prueba de Kolmogorov-Smirnov o la prueba de chi-cuadrado, para comparar las distribuciones de datos actuales con tus datos de entrenamiento originales. Configura alertas cuando las pruebas detecten una desviación significativa para poder volver a entrenar los modelos con datos actualizados o investigar los problemas relacionados con la canalización de datos.

Métricas de latencia y rendimiento

Supervisa las métricas de latencia y rendimiento para identificar los cuellos de botella en tu proceso de formación y optimizar la utilización de los recursos. Estas métricas te ayudan a entender a qué se dedica el tiempo durante la formación y dónde centrar los esfuerzos de optimización.

- End-to-End Latencia de los procesos de formación en aprendizaje automático: mide el tiempo total que tardan los datos en fluir por todo el proceso de formación, desde la ingesta de datos hasta la actualización del modelo. Realiza un seguimiento de esta métrica en todas las sesiones de entrenamiento para identificar si los cambios en el proceso mejoran o degradan el rendimiento. Una latencia alta suele indicar cuellos de botella en la carga de datos, el preprocesamiento o la comunicación de red entre los nodos.
- Rendimiento y tasa de procesamiento de la formación: haga un seguimiento del volumen de datos que procesa su proceso de formación por unidad de tiempo para garantizar una utilización eficiente de los recursos. Supervisa métricas como las muestras procesadas por segundo o los lotes completados por minuto. El bajo rendimiento en relación con la capacidad del hardware sugiere ineficiencias en la carga de datos, el preprocesamiento o la computación de modelos que desperdician los ciclos de la GPU.
- Latencia de almacenamiento y restauración de puntos de control: supervise el tiempo necesario para guardar los puntos de control del modelo en el almacenamiento (S3, EFS FSx) y restaurarlos en la memoria de la GPU o la CPU cuando se reanuden los trabajos o se recuperen de errores. Las operaciones lentas de los puntos de control prolongan el tiempo de recuperación del trabajo y aumentan los costes. Realice un seguimiento del tamaño de los puntos de control, ahorre la duración, la duración de la restauración y el recuento de errores para identificar oportunidades de optimización, como la compresión o los niveles de almacenamiento más rápidos.
- Tiempo de carga y preprocesamiento de los datos: mida el tiempo dedicado a cargar los datos del almacenamiento y a aplicar las transformaciones previas al procesamiento. Compare este tiempo con el tiempo de cálculo del modelo para determinar si su entrenamiento se basa en los datos o en el cómputo. Si la carga de datos consume más del 20% del tiempo total de entrenamiento, considere la posibilidad de optimizar la canalización de datos mediante el almacenamiento en caché, la captura previa o un almacenamiento más rápido.

Tasas de errores y errores

Supervise las tasas de error y las fallas a lo largo de su proceso de capacitación para mantener la confiabilidad y evitar el desperdicio de recursos informáticos. Los errores no detectados pueden provocar que los trabajos de formación fallen silenciosamente, que se produzcan modelos no válidos o que se desperdicien horas de tiempo en la GPU antes de que se detecten problemas.

- Supervisión de errores en la canalización: realiza un seguimiento de los errores en todas las etapas de tu proceso de aprendizaje automático, incluido el preprocesamiento de datos, el entrenamiento con modelos y las operaciones de puntos de control. Registra los tipos de errores,

las frecuencias y los componentes afectados para identificar rápidamente los problemas. Entre los errores más comunes se incluyen las discordancias en el formato de los datos, out-of-memory los errores durante el preprocesamiento y los errores en los puntos de control debidos a los límites de almacenamiento. Configure alertas cuando las tasas de error superen los umbrales de referencia para poder investigar antes de que los errores se generen en cascada.

- Análisis de errores recurrentes: identifique e investigue los patrones de los errores recurrentes para evitar futuros fallos y mejorar la fiabilidad de las tuberías. Analice los registros para determinar si las muestras de datos específicas, los tamaños de los lotes o las configuraciones de entrenamiento causan errores de manera constante. Por ejemplo, si determinados tipos de datos de entrada provocan errores de preprocesamiento, añada comprobaciones de validación en una fase más temprana del proceso o actualice la lógica de limpieza de datos. Realiza un seguimiento del tiempo medio entre fallos (MTBF) para medir si la fiabilidad de tu canalización mejora con el tiempo. »

Métricas específicas de Kubernetes y EKS

Supervise las métricas de Kubernetes y EKS para garantizar que la infraestructura de su clúster se mantenga en buen estado y pueda soportar sus cargas de trabajo de formación. Estas métricas le ayudan a detectar problemas de infraestructura antes de que provoquen fallos en las tareas de formación o una degradación del rendimiento.

- Métricas del estado de los clústeres de Kubernetes: supervise el estado y el estado de los objetos de Kubernetes, incluidos los pods, los nodos, las implementaciones y los servicios. Realice un seguimiento del estado de los pods para identificar los pods que se encuentran en estado pendiente, fallido o en bucle de bloqueo. Supervisa el estado de los nodos para detectar problemas como la presión del disco, la presión de la memoria o la falta de disponibilidad de la red. Usa kubectl o las herramientas de monitoreo para comprobar estas métricas de forma continua y configurar alertas cuando los pods no se inicien o los nodos no se puedan programar.
- Métricas de ejecución de la canalización de formación: realice un seguimiento de las ejecuciones de canalizaciones correctas y fallidas, la duración de los trabajos, los tiempos de finalización de los pasos y los errores de orquestación. Supervise si los trabajos de formación se completan dentro de los plazos esperados y si las tasas de fallos aumentan con el tiempo. Realice un seguimiento de métricas como la tasa de éxito laboral, la duración media del trabajo y el tiempo transcurrido hasta el fracaso. Estas métricas le ayudan a identificar si los problemas de infraestructura, los problemas de configuración o los problemas de calidad de los datos provocan fallos en la formación.

- Estadísticas de servicios de AWS: realice un seguimiento de las métricas de los servicios de AWS que respaldan su infraestructura de EKS y sus cargas de trabajo de formación. Supervise las métricas de S3, como la latencia de las solicitudes, las tasas de error y el rendimiento, para garantizar que el rendimiento de la carga de datos se mantenga constante. Realice un seguimiento de las métricas de volumen de EBS, incluidas las IOPS, el rendimiento y la longitud de las colas, para detectar cuellos de botella en el almacenamiento. Supervise los registros de flujo de la VPC y las métricas de la red para identificar los problemas de conectividad entre los nodos o con los servicios externos.
- Métricas del plano de control de Kubernetes: supervise el servidor de la API, el programador, el administrador del controlador y la base de datos, etc., para detectar problemas o fallas de rendimiento que afecten a las operaciones del clúster. Realice un seguimiento de la latencia de las solicitudes del servidor API, la tasa de solicitudes y la tasa de errores para garantizar que el plano de control responda rápidamente a las solicitudes de programación. Supervise el tamaño de la base de datos etcd, la duración de las confirmaciones y los cambios principales para detectar problemas de estabilidad. La alta latencia del servidor API o los cambios frecuentes de líderes de etcd pueden retrasar la programación de los módulos y prolongar los tiempos de inicio de los trabajos de formación.

Registros de aplicaciones e instancias

Recopile y analice los registros de aplicaciones e instancias para diagnosticar problemas que las métricas por sí solas no pueden explicar. Los registros proporcionan un contexto detallado sobre los errores, los cambios de estado y los eventos del sistema que le ayudan a comprender por qué los trabajos de formación fallan o tienen un rendimiento deficiente. Al correlacionar los registros con las métricas, puede identificar las causas fundamentales con mayor rapidez.

- Registros de aplicaciones: Recopile los registros de las aplicaciones de sus trabajos de formación, flujos de datos y marcos de aprendizaje automático para identificar los cuellos de botella y diagnosticar los fallos. Estos registros recopilan información detallada sobre la ejecución de los trabajos, incluidos los errores de carga de datos, los errores de inicialización del modelo, los errores al guardar los puntos de control y las advertencias específicas del marco. Correlaciona las marcas temporales del registro con los picos de las métricas para entender qué causó la degradación o los fallos del rendimiento. Por ejemplo, si el uso de la GPU disminuye repentinamente, compruebe los registros de la aplicación para ver si hay errores que indiquen un estancamiento en la canalización de datos o errores de preprocesamiento. Utilice herramientas de registro centralizadas, como CloudWatch Logs o Fluent Bit, para agregar los registros de todos los pods y hacer que se puedan buscar en ellos.

- Registros de instancias: Recopile registros a nivel de instancia, como los registros del diario del sistema y los resultados de dmesg, para detectar problemas de hardware y problemas a nivel de núcleo. Estos registros revelan problemas como los errores del controlador de la GPU, los errores de asignación de memoria, los errores de disco I/O y los problemas de la interfaz de red, que pueden no aparecer en los registros de las aplicaciones. Correlaciona los registros de las instancias con los registros y las métricas de las aplicaciones para determinar si los errores de entrenamiento se deben a problemas de hardware o de aplicaciones. Por ejemplo, si un trabajo de entrenamiento no funciona debido a un out-of-memory error, compruebe en los registros dmesg los mensajes que indican si el sistema se ha quedado sin memoria o si la aplicación ha superado sus límites de contenedores. Configura alertas en caso de errores de hardware críticos, como errores XID de la GPU o fallos de disco, para poder sustituir las instancias defectuosas antes de que provoquen interrupciones generalizadas en el entrenamiento.

En las siguientes secciones se muestra cómo recopilar las métricas descritas anteriormente mediante dos enfoques recomendados por AWS: CloudWatch Container Insights y Amazon Managed Prometheus Amazon Managed Prometheus with Amazon Managed Grafana. Elija CloudWatch Container Insights si prefiere herramientas nativas de AWS con una configuración mínima y paneles prediseñados. Elija Amazon Managed Prometheus con Amazon Managed Grafana si necesita paneles personalizados, capacidades de visualización avanzadas o si desea integrarlos con la infraestructura de monitoreo existente basada en Prometheus. Para obtener una lista completa de las métricas de Container Insights disponibles, consulte las métricas de Amazon EKS y Kubernetes Container Insights.

Considere la posibilidad de monitorear las métricas de inferencias en línea en tiempo real

En los sistemas en tiempo real, la baja latencia es fundamental para proporcionar respuestas oportunas a los usuarios u otros sistemas dependientes. La alta latencia puede degradar la experiencia del usuario o infringir los requisitos de rendimiento. Los componentes que influyen en la latencia de inferencia incluyen el tiempo de carga del modelo, el tiempo de preprocesamiento, el tiempo de predicción real, el tiempo de posprocesamiento y el tiempo de transmisión de la red. Recomendamos monitorear la latencia de inferencia para garantizar que las respuestas de baja latencia cumplan con los acuerdos de nivel de servicio (SLAs) y desarrollar métricas personalizadas para lo siguiente. Realice pruebas con la carga esperada, incluya la latencia de la red, contabilice las solicitudes simultáneas y realice pruebas con lotes de distintos tamaños.

- Tiempo transcurrido hasta el primer token (TTFT): tiempo transcurrido desde que un usuario envía una solicitud hasta que recibe el principio de la respuesta (la primera palabra, símbolo o fragmento). Por ejemplo, en los chatbots, debes comprobar cuánto tiempo se tarda en generar el primer resultado (token) después de que el usuario haga una pregunta.
- End-to-End Latencia: es el tiempo total desde que se recibe una solicitud hasta que se devuelve la respuesta. Por ejemplo, mida el tiempo transcurrido desde la solicitud hasta la respuesta.
- Producción de fichas por segundo (TPS): indica la rapidez con la que su modelo genera nuevas fichas una vez que comienza a responder. Por ejemplo, en los chatbots, harías un seguimiento de la velocidad de generación de los modelos de lenguaje para un texto de referencia.
- Tasa de errores: hace un seguimiento de las solicitudes fallidas, lo que puede indicar problemas de rendimiento. Por ejemplo, supervisa las solicitudes fallidas de documentos grandes o de determinados caracteres.
- Rendimiento: mida la cantidad de solicitudes u operaciones que el sistema puede gestionar por unidad de tiempo. Por ejemplo, realice un seguimiento de las solicitudes por segundo para gestionar los picos de carga.

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/Vo una sesión (uso de caché):

- hit/miss Proporción de caché: en el caso de las configuraciones de inferencia que utilizan el almacenamiento en caché (cachés K/V o de incrustación), mida la frecuencia con la que la caché ayuda. Las bajas tasas de aciertos pueden indicar cambios en la carga de trabajo o en la configuración de la caché que no son óptimos, lo que puede aumentar la latencia.

En los temas siguientes, demostraremos la recopilación de datos para algunas de las métricas mencionadas anteriormente. Proporcionaremos ejemplos con los dos enfoques recomendados por AWS: _CloudWatch Container Insights, nativo de AWS, y Prometheus gestionado por Amazon de código abierto con Amazon Managed Grafana. Escogería una de estas soluciones en función de sus necesidades generales de observabilidad. Consulte las métricas de Amazon EKS y Kubernetes Container Insights para ver la lista completa de las métricas de Container Insights.

Seguimiento del uso de la memoria de la GPU

Como se explica en el the section called “Considere la posibilidad de monitorizar la formación básica y ajustar las métricas” tema, el uso de la memoria de la GPU es esencial para evitar errores out-of-memory (OOM) y garantizar un uso eficiente de los recursos. Los siguientes ejemplos muestran cómo configurar una aplicación de entrenamiento para mostrar una métrica de histograma personalizada y cómo calcular el uso de memoria del P95 para identificar los picos de consumo. `gpu_memory_usage_bytes` Asegúrese de realizar la prueba con un ejemplo de trabajo de formación (por ejemplo, ajustar un modelo de transformador) en un entorno de ensayo.

Ejemplo de AWS Native CloudWatch Container Insights

En este ejemplo se muestra cómo instrumentar su aplicación de formación para exponerla `gpu_memory_usage_bytes` como un histograma utilizando el enfoque nativo de AWS. Tenga en cuenta que su AI/ML contenedor debe estar configurado para emitir registros estructurados en formato CloudWatch Embedded Metrics Format (EMF). CloudWatch registra, analiza los EMF y publica las métricas. Usa aws_embedded_metrics en tu aplicación de entrenamiento para enviar registros estructurados en formato EMF a Logs, que extrae las métricas de la GPU. CloudWatch

```
from aws_embedded_metrics import metric_scope
import torch
import numpy as np

memory_usage = []

@metric_scope
def log_gpu_memory(metrics):
    # Record current GPU memory usage
    mem = torch.cuda.memory_allocated()
    memory_usage.append(mem)

    # Log as histogram metric
    metrics.set_namespace("MLTraining/GPUMemory")
    metrics.put_metric("gpu_memory_usage_bytes", mem, "Bytes", "Histogram")

    # Calculate and log P95 if we have enough data points
    if len(memory_usage) >= 10:
        p95 = np.percentile(memory_usage, 95)
        metrics.put_metric("gpu_memory_p95_bytes", p95, "Bytes")
        print(f"Current memory: {mem} bytes, P95: {p95} bytes")
```

```
# Example usage in training loop
for epoch in range(20):
    # Your model training code would go here
    log_gpu_memory()
```

Ejemplo de Prometheus y Grafana

En este ejemplo se muestra cómo instrumentar su aplicación de entrenamiento para que se exponga gpu_memory_usage_bytes` como un histograma mediante la biblioteca cliente Prometheus en Python.

```
from prometheus_client import Histogram
from prometheus_client import start_http_server
import pynvml

start_http_server(8080)
memory_usage = Histogram(
    'gpu_memory_usage_bytes',
    'GPU memory usage during training',
    ['gpu_index'],
    buckets=[1e9, 2e9, 4e9, 8e9, 16e9, 32e9]
)

# Function to get GPU memory usage
def get_gpu_memory_usage():
    if torch.cuda.is_available():
        # Get the current GPU device
        device = torch.cuda.current_device()

        # Get memory usage in bytes
        memory_allocated = torch.cuda.memory_allocated(device)
        memory_reserved = torch.cuda.memory_reserved(device)

        # Total memory usage (allocated + reserved)
        total_memory = memory_allocated + memory_reserved

        return device, total_memory
    else:
        return None, 0

# Get GPU memory usage
gpu_index, memory_used = get_gpu_memory_usage()
```

Realice un seguimiento de la duración de la solicitud de inferencia para realizar inferencias en línea en tiempo real

Como se discutió en el the section called “Considere la posibilidad de monitorizar la formación básica y ajustar las métricas” tema, la baja latencia es fundamental para proporcionar respuestas oportunas a los usuarios u otros sistemas dependientes. Los siguientes ejemplos muestran cómo realizar un seguimiento de una métrica de histograma personalizada`inference_request_duration_seconds`, expuesta por la aplicación de inferencia. Calcule la latencia del percentil 95 (P95) para centrarse en los peores escenarios posibles, realice pruebas con solicitudes de inferencias sintéticas (por ejemplo, mediante Locust) en un entorno de ensayo y establezca umbrales de alerta (por ejemplo, más de 500 ms) para detectar las infracciones del SLA.

Ejemplo de AWS Native CloudWatch Container Insights

En este ejemplo, se muestra cómo crear una métrica de histograma personalizada en su aplicación de inferencia para inference_request_duration_seconds con el formato de métricas integradas de AWS. CloudWatch

```
import boto3
import time
from aws_embedded_metrics import metric_scope, MetricsLogger

cloudwatch = boto3.client('cloudwatch')

@metric_scope
def log_inference_duration(metrics: MetricsLogger, duration: float):
    metrics.set_namespace("ML/Inference")
    metrics.put_metric("inference_request_duration_seconds", duration, "Seconds",
 "Histogram")
    metrics.set_property("Buckets", [0.1, 0.5, 1, 2, 5])

@metric_scope
def process_inference_request(metrics: MetricsLogger):
    start_time = time.time()

    # Your inference processing code here
    # For example:
    # result = model.predict(input_data)

    duration = time.time() - start_time
```

```
    log_inference_duration(metrics, duration)

    print(f"Inference request processed in {duration} seconds")

# Example usage
process_inference_request()
```

Ejemplo de Prometheus y Grafana

En este ejemplo se muestra cómo crear una métrica de histograma personalizada en la aplicación de inferencia para inference_request_duration_seconds mediante la biblioteca cliente de Prometheus en Python:

```
from prometheus_client import Histogram
from prometheus_client import start_http_server
import time

start_http_server(8080)
request_duration = Histogram(
    'inference_request_duration_seconds',
    'Inference request latency',
    buckets=[0.1, 0.5, 1, 2, 5]
)
start_time = time.time()
# Process inference request
request_duration.observe(time.time() - start_time)
```

En Grafana, utilice la consulta `histogram_quantile(0.95, sum(rate(inference_request_duration_seconds_bucket[5m])) by (le, pod))` para visualizar las tendencias de latencia del P95. Para obtener más información, consulte la documentación del histograma de Prometheus y la documentación del cliente de Prometheus.

Realice un seguimiento del rendimiento de los tokens para obtener inferencias en línea en tiempo real

Como se discutió en el the section called “Considere la posibilidad de monitorizar la formación básica y ajustar las métricas” tema, recomendamos monitorear el tiempo de procesamiento de los tokens para evaluar el rendimiento del modelo y optimizar las decisiones de escalado. Los siguientes ejemplos muestran cómo realizar un seguimiento de una métrica de histograma personalizada`token_processing_duration_seconds`, expuesta por su aplicación de inferencia.

Calcule la duración del percentil 95 (P95) para analizar la eficiencia del procesamiento, realice pruebas con cargas de solicitudes simuladas (por ejemplo, de 100 a 1000 solicitudes por segundo) en un clúster que no sea de producción y ajuste los activadores de KEDA para optimizar el escalado.

Ejemplo de AWS Native CloudWatch Container Insights

En este ejemplo, se muestra cómo crear una métrica de histograma personalizada en su aplicación de inferencia para token_processing_duration_seconds con el formato de métricas integradas de AWS. CloudWatch Utiliza dimensiones `() with a custom `get_duration_bucket`función `set_dimension») para clasificar las duraciones en grupos (p. ej., «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
```

Ejemplo de Prometheus y Grafana

En este ejemplo, se muestra cómo crear una métrica de histograma personalizada en la aplicación de inferencia para token_processing_duration_seconds mediante la biblioteca cliente Prometheus en Python.

```
from prometheus_client import Histogram
from prometheus_client import start_http_server
import time

start_http_server(8080)
token_duration = Histogram(
    'token_processing_duration_seconds',
    'Token processing time per request',
    buckets=[0.01, 0.05, 0.1, 0.5, 1]
)
start_time = time.time()
# Process tokens
token_duration.observe(time.time() - start_time)
```

En Grafana, utilice la consulta `histogram_quantile(0.95, sum(rate(token_processing_duration_seconds_bucket[5m])) by (le, pod))` para visualizar las tendencias del tiempo de procesamiento del P95. Para obtener más información, consulte la documentación del histograma de Prometheus y la documentación del cliente de Prometheus.

Mida la latencia de restauración de los puntos de control

Como se explica en el the section called “Considere la posibilidad de monitorizar la formación básica y ajustar las métricas” tema, la latencia de los puntos de control es una métrica fundamental durante varias fases del ciclo de vida del modelo. Los siguientes ejemplos muestran cómo realizar un seguimiento de una métrica de histograma personalizada`checkpoint_restore_duration_seconds`, expuesta por su aplicación. Calcule la duración del percentil 95 (P95) para supervisar el rendimiento de la restauración, realice pruebas con interrupciones puntuales en un clúster que no esté en producción y establezca umbrales de alerta (por ejemplo, menos de 30 segundos) para detectar retrasos.

Ejemplo de AWS Native CloudWatch Container Insights

En este ejemplo, se muestra cómo instrumentar una aplicación por lotes para mostrar checkpoint_restore_duration_seconds como un histograma mediante Insights: CloudWatch

```
import boto3
import time
import torch
from aws_embedded_metrics import metric_scope, MetricsLogger

@metric_scope
def log_checkpoint_restore(metrics: MetricsLogger, duration: float):
    metrics.set_namespace("ML/ModelOperations")
    metrics.put_metric("checkpoint_restore_duration_seconds", duration, "Seconds",
 "Histogram")
    metrics.set_property("Buckets", [1, 5, 10, 30, 60])
    metrics.set_property("CheckpointSource", "s3://my-bucket/checkpoint.pt")

@metric_scope
def load_checkpoint(model, checkpoint_path: str, metrics: MetricsLogger):
    start_time = time.time()

    # Load model checkpoint
    model.load_state_dict(torch.load(checkpoint_path))

    duration = time.time() - start_time
    log_checkpoint_restore(metrics, duration)

    print(f"Checkpoint restored in {duration} seconds")
```

Ejemplo de Prometheus y Grafana

En este ejemplo se muestra cómo instrumentar una aplicación por lotes para que se exponga `checkpoint_restore_duration_seconds` como un histograma mediante la biblioteca cliente Prometheus en Python:

```
from prometheus_client import Histogram
from prometheus_client import start_http_server
import torch

start_http_server(8080)
restore_duration = Histogram(
    'checkpoint_restore_duration_seconds',
    'Time to restore checkpoint',
    buckets=[1, 5, 10, 30, 60]
```

```
)
with restore_duration.time():
    model.load_state_dict(torch.load("s3://my-bucket/checkpoint.pt"))
```

En Grafana, utilice la consulta `histogram_quantile(0.95, sum(rate(checkpoint_restore_duration_seconds_bucket[5m]) by (le))` para visualizar las tendencias de latencia de restauración del P95. Para obtener más información, consulte la documentación del histograma de Prometheus y la documentación del cliente de Prometheus.

Escalado y rendimiento de las aplicaciones

Administración de artefactos de aprendizaje automático, marcos de servicio y optimización de empresas emergentes

La implementación de modelos de aprendizaje automático (ML) en Amazon EKS requiere considerar detenidamente cómo se integran los modelos en las imágenes de los contenedores y en los entornos de ejecución. Esto garantiza la escalabilidad, la reproducibilidad y la utilización eficiente de los recursos. En este tema se describen los diferentes enfoques para gestionar los artefactos de los modelos de aprendizaje automático, seleccionar los marcos de servicio y optimizar los tiempos de inicio de los contenedores mediante técnicas como el almacenamiento previo en caché, todas diseñadas para reducir los tiempos de inicio de los contenedores.

Manejo de los artefactos de los modelos de aprendizaje automático en las implementaciones

Una decisión clave es cómo gestionar por sí mismos los artefactos del modelo de aprendizaje automático (como los pesos y las configuraciones). La elección afecta al tamaño de la imagen, a la velocidad de despliegue, a la frecuencia de actualización del modelo y a la sobrecarga operativa. Tenga en cuenta que cuando hablamos de almacenar el «modelo», nos referimos a los artefactos del modelo (como los parámetros entrenados y los pesos del modelo). Existen diferentes enfoques para gestionar los artefactos de los modelos de aprendizaje automático en Amazon EKS. Cada uno tiene sus ventajas y desventajas, y el mejor depende del tamaño del modelo, la cadencia de actualización y las necesidades de infraestructura. Considera los siguientes enfoques, desde el más mínimo hasta el más recomendado:

- Insertar el modelo en la imagen del contenedor: Copie los archivos del modelo (p. ej., .safetensors, .pth, .h5) en la imagen del contenedor (p. ej., Dockerfile) durante la creación de

la imagen. El modelo forma parte de la imagen inmutable. Recomendamos usar este enfoque para modelos más pequeños con actualizaciones poco frecuentes. Esto garantiza la coherencia y la reproducibilidad, no retrasa la carga y simplifica la gestión de las dependencias, pero se traduce en tamaños de imagen más grandes, lo que ralentiza las compilaciones y las inserciones, requiere la reconstrucción y el redespliegue para las actualizaciones de los modelos, y no es ideal para modelos de gran tamaño debido al alto rendimiento del registro.

- Descarga del modelo en tiempo de ejecución: al iniciar el contenedor, la aplicación descarga el modelo del almacenamiento externo (por ejemplo, Amazon S3, respaldado por S3 CRT para optimizar las transferencias de alto rendimiento mediante métodos como el controlador CSI Mountpoint for S3, la CLI de AWS S3 o la CLI de OSS de s5cmd) mediante scripts en un contenedor de inicio o punto de entrada. Recomendamos empezar con este enfoque para modelos grandes con actualizaciones frecuentes. Esto hace que las imágenes de los contenedores se centren en el código y el tiempo de ejecución, permite actualizar fácilmente los modelos sin necesidad de volver a compilarlos, admite el control de versiones mediante metadatos de almacenamiento, pero introduce posibles fallos de red (requiere una lógica de reintento), requiere autenticación y almacenamiento en caché.

Para obtener más información, consulte Acelerar el proceso de extracción en el taller sobre IA sobre EKS.

Sirviendo modelos de aprendizaje automático

Para implementar y ofrecer modelos de aprendizaje automático (ML) en Amazon EKS, es necesario seleccionar un enfoque de servicio de modelos adecuado para optimizar la latencia, el rendimiento, la escalabilidad y la simplicidad operativa. La elección depende del tipo de modelo (por ejemplo, el idioma o el modelo de visión), las exigencias de la carga de trabajo (por ejemplo, la inferencia en tiempo real) y la experiencia del equipo. Los enfoques comunes incluyen configuraciones basadas en Python para la creación de prototipos, servidores de modelos dedicados para funciones de nivel de producción y motores de inferencia especializados para un alto rendimiento y eficiencia. Cada método implica compensaciones en cuanto a la complejidad de la configuración, el rendimiento y la utilización de los recursos. Tenga en cuenta que los marcos de servicio pueden aumentar el tamaño de las imágenes de los contenedores (varios GBs) debido a las dependencias, lo que podría afectar a los tiempos de inicio; considere la posibilidad de disociarlos mediante técnicas de manejo de artefactos para mitigar esta situación. Las opciones se enumeran de las menos a las más recomendadas:

Uso de marcos de Python (por ejemplo, FastAPI, HuggingFace Transformers with PyTorch) Desarrolle una aplicación personalizada utilizando marcos de Python, incrustando archivos de modelo (pesos, configuración, tokenizador) dentro de una configuración de nodos en contenedores.

- Ventajas: creación sencilla de prototipos, solo para Python sin infraestructura adicional, compatible con todos los HuggingFace modelos, implementación sencilla de Kubernetes.
- Contras: se limita a un solo procesamiento por request/simple lotes, es lenta la generación de tokens (no hay núcleos optimizados), la memoria es ineficiente, no tiene capacidad de escalado ni supervisión e implica tiempos de inicio prolongados.
- Recomendación: Úselo para la creación inicial de prototipos o para tareas de un solo nodo que requieran una integración lógica personalizada.

Utilice marcos de servicio de modelos dedicados (por ejemplo, TensorRT-LLM, TGI) Adopte servidores especializados como TensorRT-LLM o TGI para la inferencia de ML, gestionando la carga, el enrutamiento y la optimización de modelos. Son compatibles con formatos como safetensors, con compilaciones o complementos opcionales.

- Ventajas: Ofrece procesamiento por lotes (paralelismo). static/in-flight or continuous), quantization (INT8, FP8, GPTQ), hardware optimizations (NVIDIA, AMD, Intel, Inferentia), and multi-GPU support (Tensor/Pipeline TensorRT-LLM admite diversos modelos (codificador-decodificador)LLMs, mientras que TGI aprovecha la integración. HuggingFace
- Contras: TensorRT-LLM necesita compilación y es solo para NVIDIA; es posible que TGI sea menos eficiente en el procesamiento por lotes; ambos añaden una sobrecarga de configuración y es posible que no se adapten a todos los tipos de modelos (por ejemplo, los que no son transformadores).
- Recomendación: adecuado para modelos que necesitan capacidades de producción como pruebas o un alto rendimiento con hardware PyTorch/TensorFlow compatible. A/B

Utilice motores de inferencia especializados de alto rendimiento (por ejemplo, el vLLM) Utilice motores de inferencia avanzados, como el vLLM, para optimizar el servicio de LLM, el procesamiento por lotes durante el vuelo y la cuantificación (, -KV, AWQ) PagedAttention, integrables con el escalado automático de EKS. INT8 FP8

- Ventajas: Alto rendimiento y eficiencia de memoria (entre un 40 y un 60% de ahorro en VRAM), gestión dinámica de solicitudes, transmisión de tokens, compatibilidad con múltiples GPU Tensor Parallel de un solo nodo y amplia compatibilidad de hardware.

- Desventajas: Optimizado para transformadores que solo utilizan decodificadores (p. ej., LLa MA), es menos eficaz para los modelos sin transformadores, pero requiere hardware compatible (p. ej., NVIDIA) y un esfuerzo de configuración. GPUs
- Recomendación: la mejor opción para realizar inferencias LLM de gran volumen y baja latencia en EKS, ya que maximiza la escalabilidad y el rendimiento.

Optimización de los tiempos de extracción de imágenes de contenedores

Las imágenes de contenedores de gran tamaño pueden provocar retrasos en el arranque en frío, lo que repercute en la latencia de arranque del módulo. Para las cargas de trabajo sensibles a la latencia, como las cargas de trabajo de inferencia en tiempo real escaladas horizontalmente, es fundamental iniciar rápidamente los pods. Tenga en cuenta los siguientes enfoques para optimizar los tiempos de extracción de imágenes de los contenedores:

Reducir el tamaño de las imágenes de los contenedores

Reducir el tamaño de las imágenes del contenedor durante el inicio es otra forma de reducir el tamaño de las imágenes. Puede realizar reducciones en cada paso del proceso de creación de la imagen del contenedor. Para empezar, elija imágenes base que contengan el menor número de dependencias necesario. Durante la creación de imágenes, incluya solo las bibliotecas y artefactos esenciales que sean necesarios. Al crear imágenes, intente combinar varios `COPY` comandos `RUN` o comandos para crear un número menor de capas más grandes. Para AI/ML los marcos, utilice compilaciones de varias etapas para separar la compilación y el tiempo de ejecución, copiando solo los artefactos necesarios (por ejemplo, mediante `COPY –from=` registros o contextos locales) y seleccione variantes como imágenes solo en tiempo de ejecución (por ejemplo, de 3,03 GB frente `pytorch/pytorch:2.7.1-cuda11.8-cudnn9-runtime` a 6,66 GB de desarrollo). Para obtener más información, consulte Reducir el tamaño de las imágenes de los contenedores en el taller sobre IA en EKS.

Uso del snapshotter SOCI para extraer imágenes previamente

Para imágenes muy grandes que no pueda minimizar fácilmente, puede utilizar el snapshotter Seekable OCI (SOCI) de código abierto configurado en modo paralelo de extracción y desempaquetado. Esta solución le permite utilizar las imágenes existentes sin tener que volver a crear ni modificar los procesos de creación. Esta opción resulta especialmente eficaz cuando se despliegan cargas de trabajo con imágenes muy grandes en instancias EC2 informáticas de alto rendimiento. Funciona bien con redes de alto rendimiento y configuraciones de almacenamiento de alto rendimiento, como es habitual en las cargas de trabajo escaladas. AI/ML

El pull/unpack modo paralelo SOCI mejora el rendimiento de extracción end-to-end de imágenes mediante estrategias de paralelización configurables. La obtención y preparación más rápidas de las imágenes afectan directamente a la rapidez con la que puede implementar nuevas cargas de trabajo y escalar su clúster de manera eficiente. La extracción de imágenes consta de dos fases principales:

1. Extraer capas del registro al nodo

   Para optimizar la captura de capas, SOCI crea varias conexiones HTTP simultáneas por capa, lo que multiplica el rendimiento de descarga más allá del límite de conexión única. Divide las capas grandes en fragmentos y las descarga simultáneamente a través de varias conexiones. Este enfoque ayuda a saturar el ancho de banda de la red disponible y a reducir considerablemente los tiempos de descarga. Esto resulta especialmente útil para AI/ML las cargas de trabajo en las que una sola capa puede ocupar varios gigabytes.

2. Desempacar y preparar esas capas para crear contenedores

   Para optimizar el desempaquetado de capas, SOCI procesa varias capas simultáneamente. En lugar de esperar a que cada capa se descomprima por completo antes de empezar con la siguiente, utiliza los núcleos de CPU disponibles para descomprimir y extraer varias capas simultáneamente. Este procesamiento paralelo transforma la tradicional fase de desempaquetado vinculada a la E/S en una operación optimizada para la CPU que se amplía con los núcleos disponibles. El sistema organiza cuidadosamente esta paralelización para mantener la coherencia del sistema de archivos y, al mismo tiempo, maximizar el rendimiento.

El modo de tracción paralela SOCI utiliza un sistema de control de doble umbral con parámetros configurables tanto para la concurrencia de descargas como para el desempaquetado del paralelismo. Este control granular le permite ajustar el comportamiento del SOCI para adaptarlo a sus requisitos de rendimiento específicos y a las condiciones del entorno. Comprender estos parámetros le ayuda a optimizar su tiempo de ejecución para obtener el mejor rendimiento de tracción.

Referencias

- Para obtener más información sobre la solución y las ventajas y desventajas de su ajuste, consulte la documentación sobre las funciones en el repositorio de proyectos de SOCI en. GitHub
- Para ver un ejemplo práctico de Karpenter en Amazon EKS, consulte el Karpenter Blueprint con el modo paralelo de SOCI snapshotter. pull/unpack

- Para obtener información sobre cómo configurar Bottlerocket para tracción paralela, consulte el modo de desembalaje con tracción paralela soci-snapshotter en la documentación de Bottlerocket.o

Uso de instantáneas de EBS para extraer imágenes previamente

Puede tomar una instantánea de Amazon Elastic Block Store (EBS) de las imágenes de contenedores almacenadas en caché y reutilizarla para los nodos de trabajo de EKS. Esto garantiza que las imágenes se precarguen localmente al iniciar el nodo, lo que reduce el tiempo de inicialización del pod. Consulte Reducir el tiempo de inicio de los contenedores en Amazon EKS con el volumen de datos de Bottlerocket para obtener más información sobre el uso de los planos de Karpenter y EKS Terraform para grupos de nodos gestionados.

Para obtener más información, consulte Uso de instantáneas de contenedores y precarga de imágenes de contenedores en volúmenes de datos de Bottlerocket con instantáneas de EBS en el taller AI on EKS.

Uso de la caché de ejecución de Container para extraer imágenes previamente

Puedes colocar previamente las imágenes de los contenedores en los nodos mediante los recursos de Kubernetes (p. ej., DaemonSet o Deployment) para rellenar la caché de tiempo de ejecución del contenedor del nodo. La caché de tiempo de ejecución del contenedor es el almacenamiento local administrado por el tiempo de ejecución del contenedor (por ejemplo, containerd), donde se almacenan las imágenes después de extraerlas de un registro. La extracción previa garantiza que las imágenes estén disponibles localmente, lo que evita demoras en la descarga durante el arranque del pod. Este enfoque resulta especialmente útil cuando las imágenes cambian con frecuencia (por ejemplo, cuando se actualizan con frecuencia), cuando las instantáneas de EBS no están preconfiguradas, cuando crear un volumen de EBS llevaría más tiempo que extraerlo directamente de un registro de contenedores, o cuando los nodos ya están en el clúster y necesitan hacer girar los pods bajo demanda utilizando una de las varias imágenes posibles.

La extracción previa de todas las variantes garantiza un tiempo de arranque rápido, independientemente de la imagen que se necesite. Por ejemplo, en una carga de trabajo de aprendizaje automático masivamente paralela que requiere 100 000 modelos pequeños creados con 10 técnicas diferentes, la extracción previa de 10 imágenes a DaemonSet través de un clúster grande (por ejemplo, miles de nodos) minimiza el tiempo de inicio del pod y permite completarlos en menos de 10 segundos al evitar las extracciones bajo demanda. El uso del enfoque de caché en tiempo de ejecución del contenedor elimina la necesidad de administrar las instantáneas de EBS y

garantiza que siempre se obtenga la última versión de la imagen del contenedor DaemonSets, pero en el caso de las cargas de trabajo de inferencia en tiempo real en las que los nodos escalan los 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 patrones, esto puede desalojar las imágenes de los nodos inactivos, lo que requiere volver a extraerlas durante las posteriores ampliaciones y reducir la confiabilidad de la memoria caché para cargas de trabajo con ráfagas de trabajo.

Consulte el GitHub repositorio de AWS para ver ejemplos de cómo extraer imágenes previamente en la caché de tiempo de ejecución del contenedor.

NVMe Considérelo para kubelet y almacenamiento en contenedores

Considere la posibilidad de configurar `kubelet` y `containerd` usar discos de almacenamiento de NVMe instancias efímeros para obtener un mayor rendimiento del disco. El proceso de extracción de contenedores implica descargar una imagen de contenedor de un registro y descomprimir sus capas en un formato utilizable. Para optimizar I/O las operaciones durante la descompresión, debe evaluar qué es lo que proporciona niveles más altos de I/O rendimiento y rendimiento para el tipo de instancia del host del contenedor: NVMe instancias respaldadas con almacenamiento local frente a IOPs/rendimiento por volumen de EBS. Para EC2 las instancias con almacenamiento NVMe local, considera configurar el sistema de archivos subyacente del nodo para que kubelet ()`/var/lib/kubelet`, containerd () y Pod logs (`/var/lib/containerd/var/log/pods`) usen discos de almacenamiento de NVMe instancias efímeros para obtener niveles más altos de rendimiento y rendimiento. I/O

El almacenamiento efímero del nodo se puede compartir entre los pods que soliciten almacenamiento efímero y las imágenes de contenedores que se descargan en el nodo. Si utilizas Karpenter con Bottlerocket o el AL2 023 EKS Optimized, puedes AMIs configurarlo en RAID0 o, si EC2NodeClassutilizas grupos de nodos gestionados, instanceStorePolicy configurándolo como parte de los datos del usuario. localStoragePolicy NodeConfig

Contribuye a esta guía

Cualquier persona puede contribuir a la guía de mejores prácticas. La Guía de mejores prácticas de EKS está escrita en el AsciiDoc formato que aparece en GitHub.

Resumen para los colaboradores actuales

- Ábrelo `bpg-docs.code-workspace`con VS Code para instalar automáticamente la AsciiDoc extensión.
  - Obtenga más información sobre la AsciiDoc extensión en Visual Studio Marketplace.
- Los archivos fuente del sitio web de AWS Docs se almacenan en `latest/bpg`
- La sintaxis es muy similar a la de markdown.
  - Revisa la referencia de sintaxis en los AsciiDoctor documentos.
- La plataforma de documentos solo se implementa. `latest/bpg/images` Cada una de las secciones de la guía tiene un enlace simbólico a este directorio. Por ejemplo, `latest/bpg/networking/images` apunta `alatest/bpg/images`.

Configure un entorno de edición local

Si planea editar la guía con frecuencia, configure un entorno de edición local.

Bifurca y clona el repositorio

Debe estar familiarizado con `git`github, y los editores de texto. Para obtener información sobre cómo empezar a usar tu cuenta `git` ygithub, consulta la sección Cómo empezar con tu GitHub cuenta en los GitHub documentos.

1. Consulte la guía de mejores prácticas de EKS en GitHub.
2. Cree una bifurcación del repositorio del proyecto. Obtén información sobre cómo bifurcar un repositorio en los GitHub documentos.
3. Clona tu bifurcación del repositorio del proyecto. Aprenda a clonar su repositorio bifurcado.

Abra el espacio de trabajo de VS Code

AWS recomienda usar Visual Studio Code de Microsoft para editar la guía. Para obtener más información sobre VS Code, consulte Descargar Visual Studio Code y empezar a utilizar Visual Studio Code en la documentación de Visual Studio Code.

1. Abra VS Code.
2. Abre el `bpg-docs.code-workspace` archivo desde el repositorio clonado.
3. Si es la primera vez que abre este espacio de trabajo, acepte la solicitud para instalar la AsciiDoc extensión. Esta extensión comprueba la sintaxis de AsciiDoc los archivos y genera una vista previa en directo.
4. Navegue hasta el `latest/bpg` directorio. Este directorio contiene los archivos fuente que se implementan en el sitio de documentación de AWS. Los archivos fuente están organizados por secciones de la guía, como «seguridad» o «redes».

Editar un archivo

1. Abre un archivo en el editor.
   - Consulta la AsciiDoc sintaxis para aprender a crear encabezados, enlaces y listas.
   - Puedes usar la sintaxis de Markdown para dar formato al texto, crear listas y encabezados. No puedes usar la sintaxis de Markdown para crear enlaces.
2. Abre una vista previa en vivo de la página.
   - Primero, presiona `ctrl-k` o `cmd-k` (según el teclado). En segundo lugar, pulse`v`. Esto abre una vista previa en vista dividida.

AWS sugiere usar ramas de funciones para organizar los cambios. Aprenda a crear ramas con git.

Envía una solicitud de extracción

Puedes crear una solicitud de extracción desde el GitHub sitio web o desde el GitHub cli.

Aprende a crear una solicitud de extracción a partir de una bifurcación utilizando el GitHub sitio web.

Obtén información sobre cómo crear una solicitud de incorporación de cambios mediante la GitHub interfaz de comandos.

Usa el editor web github.dev

El editor `github.dev` basado en la web se basa en VS Code. Esta es una excelente manera de editar varios archivos y previsualizar el contenido sin necesidad de configuración alguna.

Tiene soporte para la AsciiDoc extensión. Puede realizar operaciones de git mediante la interfaz gráfica de usuario. El editor basado en la web no tiene un shell o terminal para ejecutar comandos.

Debe tener una GitHub cuenta. Se le pedirá que inicie sesión si es necesario.

Inicie el editor GitHub basado en la web.

Edita una sola página

Puede actualizar rápidamente páginas individuales utilizando GitHub. Cada página contiene un enlace GitHub "# Editar esta página en» en la parte inferior.

1. Navegue hasta la página de esta guía que desee editar
2. Haz clic en el enlace «Editar esta página en GitHub» situado en la parte inferior
3. Haz clic en el icono del lápiz de edición situado en la parte superior derecha del visor de GitHub archivos o pulsa e
4. Edita el archivo
5. Envíe los cambios mediante el botón «Confirmar cambios...». Este botón crea una solicitud GitHub de cambios. Los encargados del mantenimiento de la guía revisarán esta solicitud de extracción. Un revisor aprobará la solicitud de extracción o solicitará cambios.

Ver y configurar el ID de una página

En esta página se explica cómo ver y configurar el ID de página.

El ID de página es una cadena única que identifica cada página del sitio de documentación. Puede ver el ID de la página en la barra de direcciones del navegador cuando se encuentre en una página específica. El ID de página se usa para la URL, el nombre del archivo y para crear enlaces de referencias cruzadas.

Por ejemplo, si estás viendo esta página, la URL de la barra de direcciones del navegador tendrá un aspecto similar al siguiente:

```
https://docs.aws.amazon.com/view-set-page-id.html
```

La última parte de la URL (`view-set-page-id`) es el ID de la página.

Establece el ID de la página

Al crear una página nueva, debe establecer el ID de la página en el archivo fuente. El identificador de página debe ser una cadena concisa y separada por guiones que describa el contenido de la página.

1. Abre el archivo fuente de la nueva página en un editor de texto.
2. En la parte superior del archivo, añade la siguiente línea. Debe estar por encima del primer encabezado.

   ```
   [#my-new-page]
   ```

   `my-new-page`Sustitúyalo por el identificador de página de la nueva página.
3. Guarde el archivo.

> **Note**
>
> La página IDs debe ser única en todo el sitio de documentación. Si intentas usar un identificador de página existente, aparecerá un error de compilación.

Cómo crear una nueva página

Obtén información sobre cómo crear una página nueva y actualizar el índice de la guía.

Crea metadatos de página

1. Determine el título de la página y el título abreviado de la página. El título abreviado de la página es opcional, pero se recomienda si el título de la página tiene más de unas pocas palabras.
2. Determina el identificador de la página. Debe ser exclusivo de la Guía de mejores prácticas de EKS. La convención es usar todas las minúsculas y separar las palabras con. -
3. Cree un nuevo archivo asciidoc, en una carpeta si es necesario, y añada el siguiente texto al archivo:

Example

[.» tema "] [#<page-id>] = <page-title>:info_titleabbrev: < > page-short-title

Por ejemplo:

Example

[.» topic "] [#scalability] = Mejores prácticas de escalabilidad de EKS: info_titleabbrev: Escalabilidad

Añadir a la tabla de contenido

1. Abra el archivo de la página principal en la tabla de contenido. Para las nuevas secciones de la guía de nivel superior, el archivo principal es`book.adoc`.
2. En la parte inferior del archivo principal, actualice e inserte la siguiente directiva:

   Example

   incluye: <new-filename>[leveloffset=+1]

   Por ejemplo,

   Example

   include: :dataplane.adoc [leveloffset=+1]

Inserta una imagen

1. Busca el prefijo de imagen de la página que estás editando. Revise la `:imagesdir:` propiedad del encabezado del archivo. Por ejemplo, `` `:imagesdir: images/reliability/ ``
2. Coloca la imagen en esta ruta, por ejemplo `latest/bpg/images/reliability`
3. Determina el texto alternativo adecuado para tu imagen. Escribe una breve descripción de alto nivel de la imagen. Por ejemplo, «diagrama de VPC con tres zonas de disponibilidad» es un texto alternativo adecuado.
4. Actualice el siguiente ejemplo con el texto alternativo y el nombre de archivo de la imagen. Inserte en la ubicación deseada.

Example

imagen: <image-filename>[< image-alt-text >]

Por ejemplo:

Example

imagen:: eks-data-plane-connectivity .jpeg [Diagrama de red]

Compruebe el estilo con Vale

1. Instale la CLI de Vale.
2. Ejecute `vale sync`
3. Instale la extensión Vale desde Visual Studio Marketplace.
4. Reinicie VS Code y abra un AsciiDoc archivo
5. VS Code subraya el texto problemático. Aprenda a trabajar con errores y advertencias en los documentos de VS Code.

Crea una vista previa local

1. Instale la `asciidoctor` herramienta `brew` en Linux o macOS
   - Aprenda a instalar asciidoctor cli en los AsciiDoctor documentos.
   - Aprenda a instalar el administrador de paquetes brew.
2. Abre una terminal y navega hasta `latest/bpg/`
3. Ejecute `asciidoctor book.adoc`
   - Revise las advertencias y los errores de sintaxis
4. Abra el archivo `book.html` de salida.
   - En macOS, puedes ejecutar `open book.html` para abrir la vista previa en tu navegador predeterminado.

AsciiDoc Hoja de trucos

Formato básico

```
*bold text*
_italic text_
`monospace text`
```

Encabezados

```
= Document Title (Header 1)
== Header 2
=== Header 3
==== Header 4
===== Header 5
====== Header 6
```

Listas

Listas sin ordenar:

```
- Item 1
- Item 2
-- Subitem 2.1
-- Subitem 2.2
- Item 3
```

Listas ordenadas:

```
. 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]
```

Imágenes

```
image::image-file.jpg[Alt text]
```

Bloques de código

```
 [source,python]
 ----
 def hello_world():
     print("Hello, World!")
 ----
```

Tablas

Aprende a crear una tabla básica.

```
[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
|===
```

Advertencias

```
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.
```

Vista previa:

> **Note**
>
> Esta es una nota de advertencia.

Incluye

```
include::filename.adoc[]
```

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.