Guia de melhores práticas

Amazon EKS



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

Amazon EKS: Guia de melhores práticas

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

As marcas comerciais e imagens de marcas da Amazon não podem ser usadas no contexto de nenhum produto ou serviço que não seja da Amazon, nem de qualquer maneira que possa gerar confusão entre os clientes ou que deprecie ou desprestigie a Amazon. Todas as outras marcas comerciais não pertencentes à Amazon são propriedade de seus respectivos proprietários, que podem ou não ser afiliados, conectados ou patrocinados pela Amazon

Table of Contents

Introdução 1
Guias relacionados 1
contribuição 2
Segurança 3
Como usar este guia 3
Entendendo o modelo de responsabilidade compartilhada 3
Introdução 1
Feedback 6
Outras fontes de leitura 6
Ferramentas e recursos 6
Modo EKS Auto - Segurança 6
Arquitetura de segurança 7
Perguntas frequentes 17
Gerenciamento de Identidade e Acesso 20
Controlando o acesso aos clusters EKS 20
Recomendações de acesso ao cluster 28
Identidades e credenciais para pods EKS 34
Recomendações de identidades e credenciais para pods EKS 43
Ferramentas e recursos 52
Segurança do pod 52
Capacidades do Linux 53
Soluções de segurança para Pod 54
Recomendações 63
Ferramentas e recursos 52
Multilocação 70
Multilocação suave 71
Construções do Kubernetes 73
Controles de mitigação 76
Multilocação rígida 83
Direções futuras 84
Ferramentas e recursos de gerenciamento de vários clusters 84
Controles de Detective 84
Recomendações 63
Ferramentas e recursos 52

Segurança de rede 92
Controle de tráfego 93
Criptografia de rede 93
Política de rede 93
Recomendações 63
Grupos de segurança 98
Quando usar a política de rede versus o grupo de segurança para pods? 101
Aplicação da política de Service Mesh ou política de rede Kubernetes 102
ThirdParty Mecanismos de política de rede 103
Criptografia em trânsito 105
Ferramentas e recursos 52
Criptografia de dados e gerenciamento de segredos 115
Criptografia em repouso 115
Gerenciamento de segredos 117
Ferramentas e recursos 52
Segurança de runtime 120
Contextos de segurança e controles Kubernetes integrados 121
Recomendações 63
Ferramentas e recursos 124
Segurança da infraestrutura 125
Recomendações 63
Alternativas 129
Ferramentas e recursos 52
Conformidade regulatória 133
Deslocando para a esquerda 134
Ferramentas e recursos 52
Resposta a incidentes e análise forense 136
Exemplo de plano de resposta a incidentes 136
Recomendações 63
Ferramentas e recursos 52
Segurança de imagem 141
Recomendações 63
Ferramentas e recursos 52
Estratégia de várias contas 152
Planejando uma estratégia de conta com várias cargas de trabalho para clusters com vários inquilinos 153

Cluster EKS centralizado 153
Clusters EKS descentralizados 160
Clusters EKS centralizados versus descentralizados 162
Gerenciamento de acesso ao cluster 165
Opções de gerenciamento de acesso EKS 165
Opção 1: AWS IAM Identity Center com API de gerenciamento de acesso ao cluster (CAM) 165
Opção 2: AWS IAM Users/Roles mapeado para grupos do Kubernetes 168
Opção 3: Provedores do OIDC 168
AWS EKS Pod Identity versus IRSA para cargas de trabalho 169
Recomendação 170
Escalonamento automático de clusters 171
Modo Automático do EKS 171
Razões para usar o Modo Automático 172
Perguntas frequentes 172
Karpenter 178
Recomendações 63
Melhores práticas da Karpenter 179
Criando NodePools 183
Pods de agendamento 187
Recomendações do CoreDNS 191
Plantas de Karpenter 191
Recursos adicionais 92
Autoscaler do cluster 192
Visão geral do 192
Otimizando o desempenho e a escalabilidade 196
Otimizando o custo e a disponibilidade 199
Casos de uso avançados 202
Parâmetros adicionais 205
Recursos adicionais 92
Referências 83
Confiabilidade 209
Como usar este guia 3
Introdução 1
Feedback 6
Aplicativos 212

Recomendações 63
Autoescalador horizontal de cápsulas (HPA) 217
Autoescalador vertical de pods (VPA) 218
Atualizar aplicativos 219
Exames de saúde e autocura 221
Recomendações 221
Lidando com interrupções 223
Recomendações 224
Observabilidade 226
Recomendações 227
Plano de controle 229
Arquitetura EKS 229
Recomendações 231
Monitore as métricas do plano de controle 231
autenticação de cluster 234
Webhooks de admissão 235
Lidando com atualizações de clusters 236
Conectividade de endpoint de cluster 237
Executando grandes clusters 238
Recursos adicionais: 238
Plano de dados 238
Recomendações 63
Redes 246
Modelo de rede Kubernetes 246
Interface de rede de contêineres (CNI) 247
Amazon Virtual Private Cloud (VPC) CNI 248
Calculadora de sub-rede 249
Considerações sobre VPC e sub-redes 250
Visão geral do 192
Recomendações 63
CNI da Amazon VPC 261
Visão geral do 192
Recomendações 63
Otimizando a utilização do endereço IP 274
Otimize o consumo de IP em nível de nó 275
Mitigar exaustão de IP 275

Executando IPv6 clusters 281
Visão geral do 192
Recomendações 63
Rede personalizada 291
Exemplo de configuração 292
Recomendações 63
Modo de prefixo para Linux 299
Recomendações 63
Modo de prefixo para Windows 304
Recomendações 63
Grupos de segurança por pod 309
Recomendações 63
Balanceamento de carga 319
Escolhendo o tipo de Load Balancer 320
Provisionamento de balanceadores de carga 321
Escolhendo o tipo de destino do Load Balancer 322
Configurando as verificações de integridade do Load Balancer 324
Disponibilidade e ciclo de vida do pod 325
Referências 83
Apêndice 330
Monitoramento de problemas de desempenho de rede 332
Monitorando o tráfego do CoreDNS para problemas de limitação de DNS 332
Monitorando atrasos nas consultas de DNS usando métricas do Conntrack 333
Outras métricas importantes de desempenho da rede 333
Capturando as métricas para monitorar cargas de trabalho em busca de problemas de desempenho de rede 334
Executando o kube-proxy no modo IPVS 342
Visão geral do 192
Escalabilidade 347
Como usar este guia 347
Entendendo as dimensões de escalabilidade 347
Dimensionamento extra grande 348
Plano de controle 348
Limite a carga de trabalho e a explosão de nós 349
Reduza a escala de nós e pods com segurança 349
Use o cache do lado do cliente ao executar o Kubectl 350

Desativar a compactação kubectl 350
Autoescalador de clusters de fragmentos 351
Prioridade e imparcialidade da API 353
Recuperação de recursos no servidor da API 360
Plano de dados 367
Escalonamento automático de nós 367
Use vários tipos de EC2 instância diferentes 368
Prefira nós maiores para reduzir a carga do servidor de API 368
Use tamanhos de nós semelhantes para um desempenho consistente da carga de trabalho 370
Use os recursos computacionais de forma eficiente 370
Automatize as atualizações do Amazon Machine Image (AMI) 371
Use vários volumes do EBS para contêineres 372
Evite instâncias com limites baixos de conexão do EBS se as cargas de trabalho usarem volumes do EBS 373
Desative o registro desnecessário no disco 374
Instâncias de patch em vigor quando a velocidade de atualização do sistema operacional é uma necessidade 374
Serviços de cluster 375
CoreDNS de escala 375
Dimensione o servidor de métricas do Kubernetes verticalmente 377
Duração do CoreDNS lamduck 377
Sonda de prontidão para CoreDNS 378
Agentes de registro e monitoramento 378
Workloads 379
Uso IPv6 para rede de pods 380
Limitar o número de serviços por namespace 380
Entenda as cotas do Elastic Load Balancer 380
Use o Route 53, o Global Accelerator ou CloudFront 381
Use EndpointSlices em vez de endpoints 382
Use segredos imutáveis e externos, se possível 382
Limitar o histórico de implantação 383
Desativar enableServiceLinks por padrão 383
Limite os webhooks de admissão dinâmica por recurso 383
Compare cargas de trabalho em vários clusters 384
A teoria por trás do escalonamento 385

Nodes versus taxa de rotatividade 385
Pensando em consultas por segundo 385
Dimensionamento de componentes distribuídos 386
Gargalos a montante e a jusante 387
Dimensionar por métricas 389
Monitoramento do plano de controle 392
Servidor da API 392
Onde está o problema? 392
Agendador 398
Gerenciador de controladores Kube 400
ETC. 402
Eficiência e escalabilidade do nó 403
Seleção de nós 403
Embalagem Node Bin 405
Utilização versus saturação 412
Definindo limites de CPU 420
Kubernetes SLOs 424
Kubernetes SLOs 424
Métricas de SLI do Kubernetes 427
SLOs em seu cluster 431
Limites e Quotas de Serviço Conhecidos 432
Outras Cotas de Serviços da AWS 432
Limitação de solicitações da AWS 440
Outros limites conhecidos 440
Atualizações de cluster 442
Visão geral do 442
Antes da atualização 442
Mantenha seu cluster up-to-date 443
Revise o calendário de lançamentos do EKS 444
Entenda como o modelo de responsabilidade compartilhada se aplica às atualizações de clusters 444
Atualize clusters no local 445
Atualize seu plano de controle e plano de dados em sequência 445
Use a documentação do EKS para criar uma lista de verificação de atualização 446
Atualize complementos e componentes usando a API Kubernetes 446
Verifique os requisitos básicos do EKS antes da atualização 448

Verifique os endereços IP disponíveis 448
Verifique a função do EKS IAM 449
Migrar para complementos do EKS 450
Identifique e corrija o uso removido da API antes de atualizar o plano de controle 451
Informações sobre clusters 451
K ube-no-trouble 453
Plutão 454
Recursos 454
Atualize as cargas de trabalho do Kubernetes. Use kubectl-convert para atualizar manifestos . 455
Configure PodDisruptionBudgets e garanta topologySpreadConstraints a disponibilidade de suas cargas de trabalho enquanto o plano de dados é atualizado 456
Use grupos de nós gerenciados ou Karpenter para simplificar as atualizações do plano de dados 457
Confirme a compatibilidade da versão com os nós existentes e o plano de controle 458
Habilitar a expiração de nós para nós gerenciados pelo Karpenter 459
Use o recurso Drift para nós gerenciados pelo Karpenter 459
Use eksctl para automatizar atualizações para grupos de nós autogerenciados 460
Faça backup do cluster antes de fazer o upgrade 460
Reinicie as implantações do Fargate após atualizar o plano de controle 460
Avalie Blue/Green os clusters como uma alternativa aos upgrades de clusters no local 461
Acompanhe as principais mudanças planejadas no projeto Kubernetes — Pense no futuro 462
Orientação específica sobre remoções de recursos 462
Remoção do Dockershim na versão 1.25 - Use o detector para Docker Socket (DDS) 462
Remoção da PodSecurityPolicy versão 1.25 - Migre para os padrões de segurança do Pod ou uma solução policy-as-code 462
Depreciação do driver de armazenamento em árvore na versão 1.23 - Migração para drivers de interface de armazenamento de contêiner (CSI) 463
Recursos adicionais 463
ClowdHaus Orientação de atualização do EKS 463
GoNoGo 463
Otimização de custo 464
Diretrizes gerais 464
Melhores práticas de otimização de custos do EKS 464
Como usar este guia 3
Principais recursos dos serviços da AWS e do Kubernetes 465
Feedback 6

Framework 466
O pilar See: mensuração e responsabilidade 466
O pilar Economizar: Otimização de custos 467
O pilar do Plano: planejamento e previsão 468
O pilar Run 468
Referências 83
Consciência 469
Recomendações 63
Outras ferramentas 477
Computação 478
Dimensione corretamente suas cargas de trabalho 478
Reduzir o consumo 479
Reduza a capacidade não utilizada 480
Otimize os tipos de capacidade computacional 486
Otimize o uso da computação 490
Rede 492
Comunicação de pod a pod 492
Comunicação entre o Load Balancer e o pod 504
Transferência de dados do Container Registry 506
Transferência de dados para a Internet e serviços da AWS 507
Transferência de dados entre VPCs 510
Usando um Service Mesh 513
Recursos adicionais 92
Armazenamento 522
Visão geral 192
Volumes efêmeros 522
Volumes persistentes 523
Outras considerações 530
Observabilidade 531
Introdução 531
Registro em log 532
Plano de controle EKS 532
Plano de dados EKS 533
Métricas 537
Rastreamento 541
Recursos adicionais: 92

Windows 544
Gerenciamento de AMI 544
Gerenciando sua própria AMI Windows otimizada para Amazon EKS 545
Configurando um lançamento mais rápido para EKS personalizado otimizado AMIs 546
Armazenando camadas base do Windows em cache de forma personalizada AMIs 547
Publicação no blog 548
gMSA para contêineres Windows 548
O que é uma conta gMSA 548
Caso de uso do contêiner Windows e gMSA 549
Fortalecimento do Windows Server 551
Reduzindo a superfície de ataque com o Windows Server Core 551
Evitando conexões RDP 552
Amazon Inspector 553
Amazon GuardDuty 554
Segurança na Amazon EC2 para Windows 554
Digitalizando imagens do Windows 555
Versões e licenciamento do Windows 555
Versão Windows Server 555
Licenciamento 556
Registro em log 556
Recomendações de registro 556
Monitorando contêineres Windows 557
Rede Windows 562
Visão geral do Windows Container Networking 562
Gerenciamento de endereços IP 563
Opções de interface de rede de contêiner (CNI) 566
Políticas de rede 566
Gerenciamento de memória e sistemas 567
Sistema de reserva e memória kubelet 567
Requisitos de memória de contêiner do Windows 568
Conclusão 568
Gerenciamento da infraestrutura 568
Empurrando e puxando imagens do Windows 569
Referência 571
Agendamento 571
Atribuição PODs às melhores práticas de atribuição aos nós 571

Garantindo que cargas de trabalho específicas do sistema operacional cheguem ao host de contêiner apropriado ... 572
Manipulando várias compilações do Windows no mesmo cluster ... 573
Simplificação NodeSelector e tolerância em manifestos de Pod usando RuntimeClass ... 574
Managed Node Group Support ... 575
Documentações adicionais ... 576
Segurança de Pod para contêineres Windows ... 576
Opções de armazenamento ... 579
O que é um plug-in em árvore versus um plug-in out-of-tree de volume? ... 579
Plug-in de volume em árvore para Windows ... 580
Out-of-tree para Windows ... 584
Servidor FSx de arquivos Amazon para Windows ... 584
Protegendo imagens de contêineres do Windows ... 585
1. Configurar políticas de conta (senha ou bloqueio) usando políticas de segurança e registro locais ... 586
2. Políticas de auditoria ... 589
3. Melhores práticas de segurança do IIS para contêineres do Windows ... 592
4. Princípio do privilégio mínimo ... 597
Considerações finais: Por que proteger seus contêineres do Windows é essencial no cenário atual de ameaças ... 600
Híbrida ... 602
Desconexão de rede ... 602
Práticas recomendadas ... 603
Failover do pod Kubernetes ... 608
Tráfego de rede de aplicativos ... 614
Credenciais do anfitrião ... 619
AI/ML ... 622
Feedback ... 622
Computação ... 622
Otimização de recursos de GPU e gerenciamento de custos ... 622
Resiliência de nós e gerenciamento de tarefas de treinamento ... 637
Dimensionamento e desempenho de aplicativos ... 640
Alocação dinâmica de recursos para gerenciamento avançado de GPU ... 643
Redes ... 689
Considere maior largura de banda de rede ou adaptador de malha elástica para aplicativos com alta comunicação entre nós ... 689

Aumente o número de endereços IP disponíveis para permitir tempos de inicialização mais rápidos do pod 690
Segurança 691
Segurança e conformidade 691
Armazenamento 692
Gerenciamento e armazenamento de dados 692
Observabilidade 706
Monitoramento e observabilidade 706
Observabilidade e métricas 708
desempenho 724
Gerenciamento de artefatos de ML, estruturas de serviço e otimização de startups 724
Otimizando os tempos de extração da imagem do contêiner 727
Considere o NVMe armazenamento de kubelet e containerd 730
Contribuir 731
Resumo para colaboradores existentes 731
Configurar um ambiente de edição local 731
Bifurque e clone o repositório 731
Abra o espaço de trabalho do VS Code 732
Editar um arquivo 732
Enviar um Pull Request 732
Use o editor baseado na web github.dev 733
Editar uma única página 733
Exibir e definir o ID de uma página 733
Definir o ID da página 734
Criação de uma nova página 734
Criar metadados da página 734
Adicionar ao índice 735
Inserir uma imagem 735
Verifique o estilo com a Vale 736
Crie uma prévia local 736
AsciiDoc Folha de dicas 737
Formatação básica 737
Cabeçalhos 737
Listas 737
Links 737
Imagens 738

Blocos de código 738
Tabelas 738
Advertências 738
Inclui 739
..... dccxl

Guia de melhores práticas do Amazon EKS

Bem-vindo aos guias de melhores práticas do EKS. O objetivo principal desse projeto é oferecer um conjunto de melhores práticas para as operações do segundo dia para o Amazon EKS. Optamos por publicar essa orientação para GitHub que pudéssemos iterar rapidamente, fornecer recomendações oportunas e eficazes para diversas questões e incorporar facilmente sugestões da comunidade em geral.

Atualmente, publicamos guias para os seguintes tópicos:

- Melhores práticas de segurança
- Melhores práticas de confiabilidade
- Melhores práticas para escalonamento automático de clusters: Karpenter
- Melhores práticas para escalonamento automático de clusters: cluster-autoscaler
- Melhores práticas para escalonamento automático de clusters: EKS Auto Mode
- Melhores práticas para redes
- Melhores práticas para escalabilidade
- Práticas recomendadas para upgrades de clusters
- Melhores práticas para otimização de custos
- Práticas recomendadas para executar contêineres do Windows
- Melhores práticas para implantações híbridas
- Práticas recomendadas para executar AI/ML cargas de trabalho

Também abrimos o código de uma CLI (interface de linha de comando) baseada em Python chamada hardeneks para verificar algumas das recomendações deste guia.

No futuro, publicaremos diretrizes de melhores práticas para desempenho, otimização de custos e excelência operacional.

Guias relacionados

Além do Guia do usuário do EKS, a AWS publicou vários outros guias que podem ajudá-lo na implementação do EKS.

- Guias de melhores práticas de contêineres do EMR

- Dados sobre o EKS
- Melhores práticas de observabilidade da AWS
- Amazon EKS Blueprints for Terraform
- Início rápido do Amazon EKS Blueprints

contribuição

Nós encorajamos você a contribuir com esses guias. Se você implementou uma prática que provou ser eficaz, compartilhe-a conosco abrindo um problema ou uma pull request. Da mesma forma, se você descobrir um erro ou falha na orientação que já publicamos, envie um PR para corrigi-la. As diretrizes para envio PRs podem ser encontradas em nossas Diretrizes de contribuição.

Melhores práticas de segurança

Este guia fornece conselhos sobre a proteção de informações, sistemas e ativos que dependem do EKS e, ao mesmo tempo, agrega valor comercial por meio de avaliações de risco e estratégias de mitigação. A orientação aqui contida faz parte de uma série de guias de melhores práticas que a AWS está publicando para ajudar os clientes a implementar o EKS de acordo com as melhores práticas. Guias de desempenho, excelência operacional, otimização de custos e confiabilidade estarão disponíveis nos próximos meses.

Como usar este guia

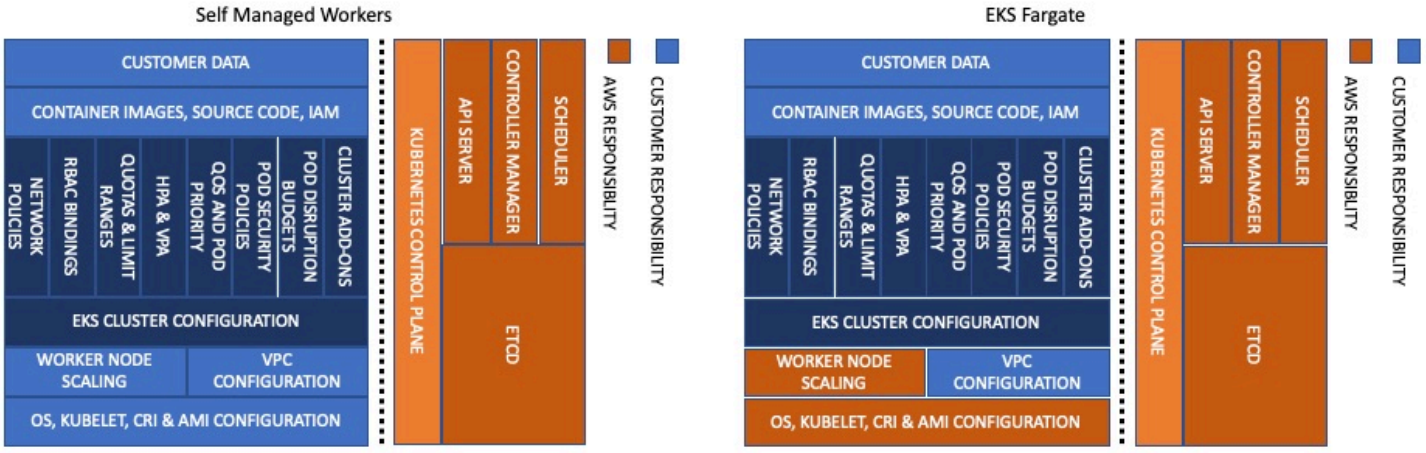
Este guia é destinado a profissionais de segurança responsáveis por implementar e monitorar a eficácia dos controles de segurança para clusters EKS e as cargas de trabalho que eles suportam. O guia está organizado em diferentes áreas temáticas para facilitar o consumo. Cada tópico começa com uma breve visão geral, seguida por uma lista de recomendações e melhores práticas para proteger seus clusters EKS. Os tópicos não precisam ser lidos em uma ordem específica.

Entendendo o modelo de responsabilidade compartilhada

Segurança e conformidade são consideradas responsabilidades compartilhadas ao usar um serviço gerenciado como o EKS. De um modo geral, a AWS é responsável pela segurança “da” nuvem, enquanto você, o cliente, é responsável pela segurança “na” nuvem. Com o EKS, a AWS é responsável pelo gerenciamento do plano de controle do Kubernetes gerenciado pelo EKS. Isso inclui os nós do plano de controle do Kubernetes, o banco de dados ETCD e outras infraestruturas necessárias para que a AWS forneça um serviço seguro e confiável. Como consumidor do EKS, você é o principal responsável pelos tópicos deste guia, por exemplo, IAM, segurança de pod, segurança de tempo de execução, segurança de rede e assim por diante.

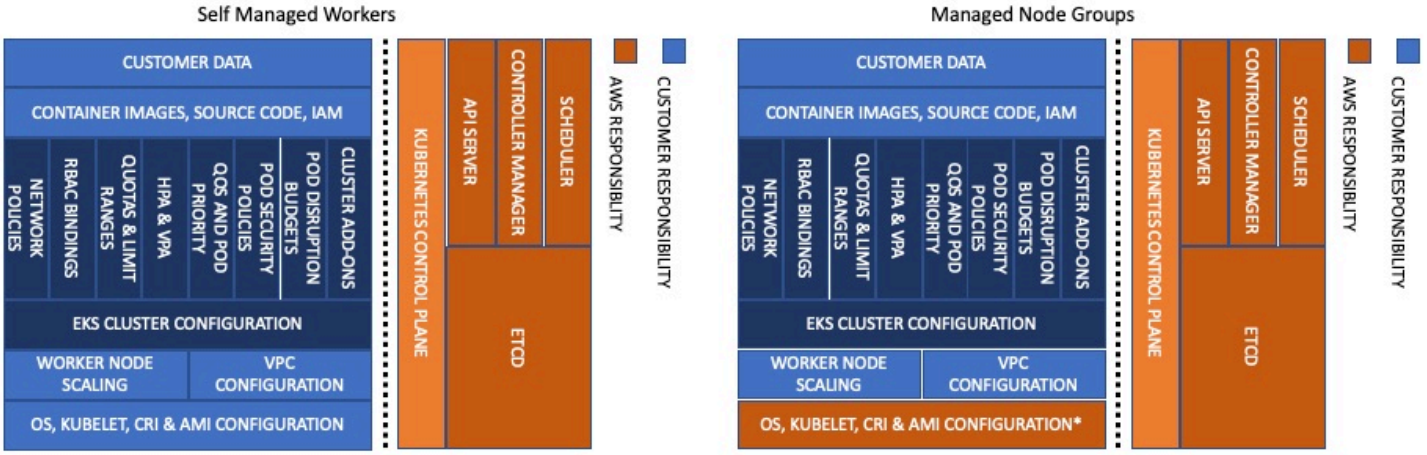
Quando se trata de segurança da infraestrutura, a AWS assumirá responsabilidades adicionais à medida que você migrar de trabalhadores autogerenciados para grupos de nós gerenciados e para o Fargate. Por exemplo, com o Fargate, a AWS se torna responsável por proteger o subjacente instance/runtime usado para executar seus pods.

Modelo de responsabilidade compartilhada - Fargate



A AWS também assumirá a responsabilidade de manter a AMI otimizada do EKS atualizada com as versões de patch e patches de segurança do Kubernetes. Os clientes que usam grupos de nós gerenciados (MNG) são responsáveis por atualizar seus grupos de nós para a AMI mais recente por meio da API EKS, CLI, Cloudformation ou console da AWS. Além disso, ao contrário do Fargate, não MNGs escalará automaticamente sua infraestrutura/cluster. Isso pode ser feito pelo autoescalador de cluster ou por outras tecnologias, como Karpenter, escalonamento automático nativo da AWS,'s Ocean ou Atlassian's Escalator. SpotInst

Modelo de responsabilidade compartilhada - MNG



Antes de projetar seu sistema, é importante saber onde está a linha de demarcação entre suas responsabilidades e o provedor do serviço (AWS).

Para obter informações adicionais sobre o modelo de responsabilidade compartilhada, consulte https://aws.amazon.com/compliance/shared-responsibility-model/

Introdução

Há várias áreas de melhores práticas de segurança que são pertinentes ao usar um serviço gerenciado do Kubernetes, como o EKS:

- Gerenciamento de Identidade e Acesso
- Segurança do pod
- Segurança em tempo de execução
- Segurança de rede
- Multilocação
- Conta múltipla para locação múltipla
- Controles de Detective
- Segurança da infraestrutura
- Criptografia de dados e gerenciamento de segredos
- Conformidade regulatória
- Resposta a incidentes e análise forense
- Segurança de imagem

Como parte do projeto de qualquer sistema, você precisa pensar em suas implicações de segurança e nas práticas que podem afetar sua postura de segurança. Por exemplo, você precisa controlar quem pode realizar ações em relação a um conjunto de recursos. Você também precisa da capacidade de identificar rapidamente incidentes de segurança, proteger seus sistemas e serviços contra acesso não autorizado e manter a confidencialidade e a integridade dos dados por meio da proteção de dados. Ter um conjunto de processos bem definido e ensaiado para responder a incidentes de segurança também melhorará sua postura de segurança. Essas ferramentas e técnicas são importantes porque ajudam a sustentar objetivos como evitar perdas financeiras ou cumprir obrigações regulatórias.

A AWS ajuda as organizações a atingirem suas metas de segurança e conformidade oferecendo um rico conjunto de serviços de segurança que evoluíram com base no feedback de um amplo conjunto de clientes preocupados com a segurança. Ao oferecer uma base altamente segura, os clientes podem passar menos tempo realizando “trabalhos pesados indiferenciados” e mais tempo alcançando seus objetivos comerciais.

Feedback

Este guia está sendo lançado GitHub para coletar feedback direto e sugestões da EKS/Kubernetes comunidade em geral. Se você tem uma prática recomendada que acha que devemos incluir no guia, registre um problema ou envie um PR no GitHub repositório. Nossa intenção é atualizar o guia periodicamente à medida que novos recursos são adicionados ao serviço ou quando uma nova prática recomendada evolui.

Outras fontes de leitura

Whitepaper de segurança do Kubernetes, patrocinado pelo Grupo de Trabalho de Auditoria de Segurança, este whitepaper descreve os principais aspectos da superfície de ataque e da arquitetura de segurança do Kubernetes com o objetivo de ajudar os profissionais de segurança a tomar decisões sólidas de design e implementação.

O CNCF também publicou um white paper sobre segurança nativa na nuvem. O paper examina como o cenário tecnológico evoluiu e defende a adoção de práticas de segurança que se alinhem aos DevOps processos e metodologias ágeis.

Ferramentas e recursos

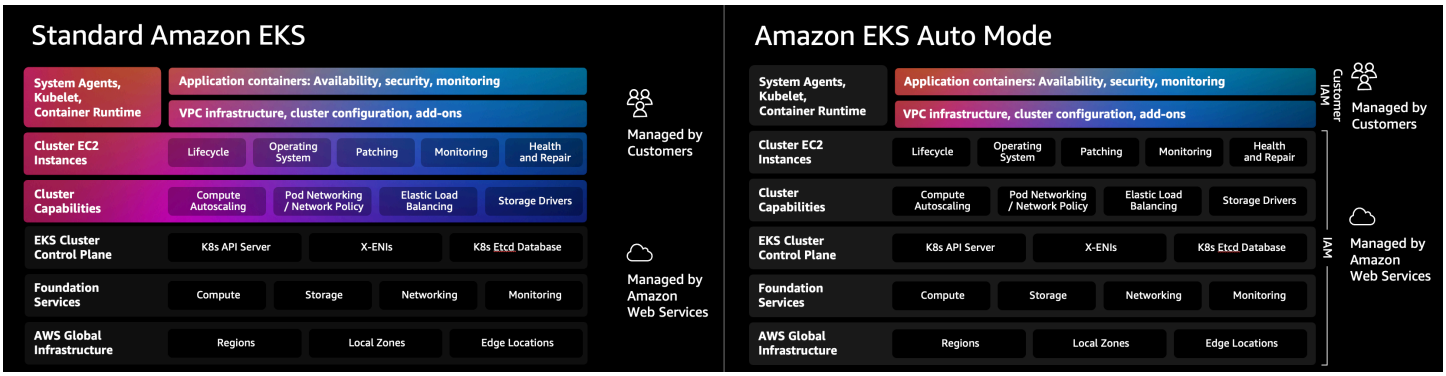
Workshop de imersão em segurança do Amazon EKS

Modo EKS Auto - Segurança

> **Tip**
>
> Interessado em adquirir experiência prática com o Amazon EKS Auto Mode? Participe de um próximo workshop virtual sobre Kubernetes conduzido por especialistas da AWS inscrevendo-se na AWS Connected Community.

O Amazon EKS Auto Mode introduz recursos de segurança aprimorados ao estender o gerenciamento de segurança da AWS além do plano de controle para incluir nós de trabalho e componentes principais do cluster. Esse modelo de segurança abrangente ajuda as organizações a manter uma postura de segurança sólida e, ao mesmo tempo, reduzir a sobrecarga operacional.

Modelo de responsabilidade compartilhada - EKS Auto Mode



Os principais aprimoramentos de segurança no EKS Auto Mode incluem:

- Sistema operacional mínimo otimizado para contêineres com superfície de ataque reduzida
- Melhores práticas de segurança impostas por meio de instâncias EC2 gerenciadas
- Gerenciamento automatizado de patches de segurança com rotação obrigatória de nós
- Isolamento de nós e limites de segurança integrados
- Integração simplificada do IAM com o mínimo de permissões necessárias

O EKS Auto Mode aplica esses controles de segurança por padrão, ajudando as organizações a atender aos requisitos de segurança e conformidade e, ao mesmo tempo, simplificando as operações de cluster. Essa abordagem se alinha aos defense-in-depth princípios, fornecendo várias camadas de controles de segurança nos níveis de infraestrutura, nó e carga de trabalho.

> **Important**
>
> Embora o EKS Auto Mode forneça recursos de segurança aprimorados, as organizações ainda devem implementar controles de segurança apropriados na camada de aplicação e seguir as melhores práticas de segurança para suas cargas de trabalho em execução no cluster.

Arquitetura de segurança

O EKS Auto Mode implementa controles de segurança em várias camadas da infraestrutura EKS, do plano de controle aos nós individuais. Compreender essa arquitetura é crucial para gerenciar e proteger com eficácia seus clusters EKS.

Segurança do avião de controle

O plano de controle EKS no modo automático EKS mantém os mesmos padrões de alta segurança dos clusters EKS tradicionais, ao mesmo tempo em que adiciona novos recursos de segurança:

- Criptografia de envelope: todos os dados da API Kubernetes são criptografados automaticamente usando criptografia de envelope.
- Integração com o KMS: usa o AWS KMS com o provedor Kubernetes KMS v2, com opções para chaves de propriedade da AWS ou chaves gerenciadas pelo cliente (CMK).
- Gerenciamento aprimorado de componentes: componentes essenciais, como auto-scaling, gerenciamento de ENI e controladores do EBS, são movidos para fora do cluster e gerenciados pela AWS.
- Controles de segurança e recursos de auditoria aprimorados: as permissões necessárias do EKS Auto Mode, além dos clusters EKS padrão, são totalmente gerenciadas por meio da função IAM do cluster, em vez de funções de nós individuais.

Integração e gerenciamento de acesso do IAM

O EKS Auto Mode fornece integração aprimorada com o AWS Identity and Access Management (IAM) por meio do EKS Access Entries e do EKS Pod Identity.

Gerenciamento de acesso ao cluster

O EKS Auto Mode introduz melhorias no gerenciamento de acesso ao cluster por meio da API Cluster Access Management (CAM):

- Modos de autenticação padronizados por meio de `EKS_API`
- Segurança aprimorada por meio do gerenciamento de acesso baseado em API
- Controle de acesso simplificado usando entradas de acesso e políticas de acesso

As entradas de acesso podem ser criadas para gerenciar o acesso ao cluster:

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

> **Important**
>
> Embora ainda seja possível criar um cluster EKS Auto Mode com o modo de `CONFIG_MAP_AND_API` autenticação, essa não é a abordagem padrão e é altamente recomendável usar o modo de `API` autenticação padrão para novos clusters. `APIa` autenticação baseada fornece segurança aprimorada e gerenciamento de acesso simplificado em comparação com a abordagem ConfigMap baseada em legados.

EKS Pod Identity

O EKS Auto Mode vem com o Pod Identity Agent já implantado, permitindo uma forma simplificada de conceder permissões do AWS IAM aos pods:

- Gerenciamento simplificado de permissões do IAM sem configuração do provedor OIDC
- Redução da sobrecarga operacional em comparação com o IRSA
- Segurança aprimorada por meio de marcação de sessão e suporte a 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**
>
> O Pod Identity é a abordagem recomendada para permissões do IAM no modo automático do EKS, pois fornece recursos de segurança aprimorados e gerenciamento simplificado em comparação com o IRSA.

Perfil do IAM do nó

O EKS Auto Mode usa um novo `AmazonEKSWorkerNodeMinimalPolicy` que fornece apenas as permissões necessárias para que os nós do EKS Auto Mode operem. Essas permissões:

- Forneça um conjunto reduzido de permissões em comparação com as políticas de nós tradicionais
- Siga o princípio do menor privilégio

- São gerenciados e atualizados automaticamente pela AWS

Essa abordagem política mínima ajuda a melhorar a postura de segurança ao limitar as permissões disponíveis para o nó e suas cargas de trabalho.

Segurança do Node

O EKS Auto Mode introduz várias melhorias significativas de segurança no nível do nó:

EC2 Segurança de instâncias gerenciadas

Os nós do EKS Auto Mode usam instâncias EC2 gerenciadas da Amazon com propriedades de segurança aprimoradas:

- Restrições impostas pelo IAM que impedem operações que poderiam comprometer a capacidade da AWS de operar os nós
- Padrões de infraestrutura imutáveis em que as alterações de configuração exigem a substituição do nó
- Substituição obrigatória do nó em 21 dias para garantir atualizações de segurança regulares
- Acesso restrito aos metadados da instância usando limites IMDSv2 de salto controlados

Segurança do sistema operacional

O sistema operacional é uma variante personalizada do Bottlerocket, otimizada para o modo automático EKS, que inclui:

- Sistema de arquivos raiz somente para leitura
- SELinux ativado por padrão com controles de acesso obrigatórios
- Isolamento automático de pods usando rótulos SELinux MCS exclusivos
- Acesso SSH desativado e remoção de serviços desnecessários
- Patches de segurança automatizados por meio da rotação de nós

Segurança de componentes do Node

Os componentes do Node são configurados com as melhores práticas de segurança:

- Kubelet configurado com padrões seguros

- Configuração reforçada de tempo de execução do contêiner
- Gerenciamento e rotação automatizados de certificados
- node-to-control-planeComunicação restrita

Segurança de rede

O EKS Auto Mode implementa vários recursos de segurança de rede para garantir a comunicação segura dentro do cluster e com recursos externos:

Política de rede VPC CNI

O EKS Auto Mode aproveita o suporte nativo à política de rede Kubernetes do plug-in CNI da Amazon VPC:

- Integra-se à API upstream de políticas de rede do Kubernetes
- Permite um controle refinado sobre a comunicação pod-to-pod
- Suporta regras de entrada e saída

Para habilitar o suporte à política de rede no EKS Auto Mode, você precisa configurar o complemento VPC CNI com um manifesto. `configMap` Exemplo:

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

Também é necessário definir se o suporte à Política de Rede está configurado na Classe Node, conforme ilustrado aqui:

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

```
  networkPolicyEventLogs: Enabled
```

Depois de ativado, você pode criar políticas de rede para controlar o tráfego:

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

Gerenciamento aprimorado de ENI

O EKS Auto Mode fornece segurança aprimorada para o gerenciamento da Elastic Network Interface (ENI):

- Conexão e configuração do ENI gerenciados pela AWS
- Separação do tráfego de controle do tráfego de dados
- Gerenciamento automatizado de endereços IP com privilégios reduzidos necessários nos nós

Segurança de armazenamento

O EKS Auto Mode fornece recursos de segurança aprimorados para armazenamento temporário e persistente:

Armazenamento efêmero

- Todos os dados gravados em volumes temporários são criptografados automaticamente
- Usa o algoritmo criptográfico AES-256 padrão do setor
- Criptografia e descriptografia gerenciadas perfeitamente pelo serviço

Volumes do EBS

- Os volumes raiz e de dados do EBS são sempre criptografados
- Os volumes são configurados para serem excluídos após o encerramento da instância

- Há uma opção para especificar chaves KMS personalizadas para criptografia

Integração com o EFS

- Support para criptografia em trânsito com o EFS
- Criptografia automática em repouso para sistemas de arquivos EFS
- Integração com pontos de acesso EFS para controle de acesso aprimorado

> **Important**
>
> Ao usar o EFS com o Modo Automático do EKS, certifique-se de que as configurações de criptografia apropriadas estejam definidas no nível do sistema de arquivos do EFS, pois o Modo Automático do EKS não gerencia a criptografia do EFS diretamente.

Monitoramento e registro em log

O EKS Auto Mode fornece recursos aprimorados de monitoramento e registro para ajudar você a manter a visibilidade da postura de segurança e da integridade operacional do seu cluster.

Registro em log do plano de controle

O EKS Auto Mode mantém os mesmos recursos de registro do plano de controle do EKS padrão, mas habilita todos os registros por padrão para monitoramento aprimorado.

- Os registros são enviados para o Amazon CloudWatch Logs
- Por padrão, o EKS Auto Mode habilita todos os registros do plano de controle: servidor de API, auditoria, autenticador, gerenciador de controladores e agendador
- O EKS Auto Mode permite visibilidade detalhada das operações do cluster e dos eventos de segurança

> **Important**
>
> O registro no plano de controle incorre em custos adicionais para o armazenamento de registros. CloudWatch Considere sua estratégia de registro com cuidado para equilibrar as necessidades de segurança com o gerenciamento de custos.

Registro em nível de nó

O EKS Auto Mode aprimora o registro em nível de nó:

- Os registros do sistema são coletados automaticamente e podem ser acessados por meio de CloudWatch registros
- Os registros do nó são retidos mesmo após o término do nó, auxiliando na análise pós-incidente
- Visibilidade aprimorada de eventos de segurança em nível de nó e problemas operacionais

GuardDuty Integração com a Amazon

Os clusters do EKS Auto Mode se integram perfeitamente à Amazon GuardDuty para melhorar a detecção de ameaças. Os recursos incluem:

- Escaneamento automatizado para registros de auditoria do plano de controle
- Monitoramento de tempo de execução que pode ser ativado para monitoramento de cargas de trabalho
- Integração com GuardDuty descobertas e mecanismos de alerta existentes

Para ativar a proteção do EKS Auto Mode no Amazon GuardDuty for Kubernetes Audit Logs, você pode executar o seguinte comando:

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

GuardDuty Integração com a Amazon para segurança de tempo de execução

GuardDuty A Amazon fornece monitoramento essencial de segurança em tempo de execução para clusters do EKS Auto Mode, oferecendo recursos abrangentes de detecção de ameaças e monitoramento de segurança. Essa integração é particularmente importante, pois ajuda a identificar possíveis ameaças à segurança e atividades maliciosas em tempo real.

Principais GuardDuty recursos do EKS Auto Mode

- Monitoramento do tempo de execução
  - Monitoramento contínuo do comportamento do tempo de execução

  - Detecção de atividades maliciosas ou suspeitas
  - Identificação de possíveis tentativas de fuga de contêineres
  - Monitoramento de execução de processos ou conexões de rede incomuns
- Detecção de ameaças específicas do Kubernetes:
  - Identificação de tentativas suspeitas de implantação do pod
  - Detecção de contêineres comprometidos
  - Monitoramento de lançamentos de contêineres privilegiados
  - Identificação do uso suspeito da conta de serviço
- Tipos de descoberta abrangentes:
  - Policy:Kubernetes/* - Detecta violações das melhores práticas de segurança
  - Impacto: Kubernetes/* - Identifica recursos potencialmente afetados
  - Discovery:Kubernetes/* - Detecta atividades de reconhecimento
  - Execution:Kubernetes/* - Identifica padrões de execução suspeitos
  - Persistence:Kubernetes/* - Detecta possíveis ameaças persistentes

Para ativar a proteção do EKS Auto Mode na Amazon GuardDuty para registros de auditoria e monitoramento de tempo de execução do Kubernetes, você pode executar o seguinte comando:

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

> **Important**
>
> GuardDuty O monitoramento de tempo de execução é automaticamente suportado nos clusters do EKS Auto Mode, fornecendo maior visibilidade de segurança sem configuração adicional no nível do nó.

GuardDuty Integração de descobertas

GuardDuty as descobertas podem ser integradas a outros serviços da AWS para uma resposta automatizada:

- EventBridge Regras:

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

- Integração com o Security Hub:

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

Melhores práticas para o GuardDuty EKS Auto Mode

1. Ativar todos os tipos de descoberta:
   - Habilite o monitoramento do log de auditoria e o monitoramento do tempo de execução do Kubernetes
   - Configure descobertas para todos os níveis de severidade
2. Implemente a resposta automatizada:
   - Crie EventBridge regras para descobertas de alta severidade
   - Integre-se ao AWS Security Hub para gerenciamento centralizado de segurança
   - Configure ações de remediação automatizadas quando apropriado
3. Revisão e ajuste regulares:
   - Revise regularmente GuardDuty as descobertas

- Ajuste os limites de detecção com base em seu ambiente
- Atualize os procedimentos de resposta com base em novos tipos de descoberta

4. Gerenciamento de várias contas:
   - Considere usar uma conta de GuardDuty administrador para gerenciamento centralizado
   - Permita a agregação de descobertas em várias contas

> **Warning**
>
> Embora GuardDuty forneça monitoramento de segurança abrangente, ele deve fazer parte de uma defense-in-depth estratégia que inclua outros controles de segurança, como políticas de rede, padrões de segurança de pods e configuração adequada do RBAC.

Perguntas frequentes

P: Como o EKS Auto Mode difere do EKS padrão em termos de segurança? R: O EKS Auto Mode fornece segurança aprimorada por meio de instâncias EC2 gerenciadas, aplicação automática de patches, rotação obrigatória de nós e controles de segurança integrados. Isso reduz a sobrecarga operacional e, ao mesmo tempo, mantém uma forte postura de segurança ao fazer com que a AWS gerencie mais aspectos de segurança.

P: Ainda posso usar as ferramentas e políticas de segurança existentes com o EKS Auto Mode? R: Sim, o EKS Auto Mode é compatível com a maioria das ferramentas e políticas de segurança existentes. No entanto, algumas ferramentas de segurança em nível de nó podem exigir adaptação devido à natureza gerenciada dos nós do EKS Auto Mode.

P: Como faço para implantar agentes de segurança e ferramentas de monitoramento no Modo Automático do EKS? R: No modo automático do EKS, os agentes de segurança e as ferramentas de monitoramento devem ser implantados como cargas de trabalho do Kubernetes (normalmente DaemonSets, o que implanta uma instância do pod em cada nó por padrão) em vez de serem instalados diretamente no sistema operacional do nó. Essa abordagem se alinha ao modelo de infraestrutura imutável do EKS Auto Mode. Exemplo:

```
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: As soluções de segurança de terceiros são compatíveis com o EKS Auto Mode? R: Muitas soluções populares de segurança de terceiros foram atualizadas para oferecer suporte ao Modo Automático EKS, no entanto, é sempre recomendável verificar a versão específica e os requisitos de implantação com seu fornecedor de segurança, pois o suporte ao Modo Automático EKS pode exigir versões atualizadas ou configurações de implantação específicas.

P: Quais são as limitações dos agentes de segurança no modo automático do EKS? R: As principais limitações incluem:

- Sem acesso direto para modificar o sistema operacional do nó
- Sem persistência nas rotações dos nós
- Deve ser compatível com a implantação baseada em contêiner
- Necessidade de respeitar a imutabilidade do nó
- Pode exigir configurações de privilégios diferentes
- Qualquer alteração persistente nos nós deve ser feita por meio `NodePools` de `NodeClasses` recursos.

> **Note**
>
> Embora o EKS Auto Mode possa exigir ajustes em sua estratégia de implantação de ferramentas de segurança, essas mudanças geralmente resultam em configurações mais fáceis de manter e seguras, alinhadas às melhores práticas nativas da nuvem. O EKS Auto Mode espera assumir completamente a maioria dos recursos que gerencia. Portanto, quaisquer alterações manuais feitas nesses recursos, se você puder acessá-los, podem ser substituídas ou descartadas pelo Modo Automático do EKS.

P: Posso usar o modo personalizado AMIs com o EKS Auto Mode? R: No momento, o EKS Auto Mode não oferece suporte personalizado AMIs. Isso ocorre intencionalmente, pois a AWS gerencia a segurança, a aplicação de patches e a manutenção dos nós como parte do modelo de responsabilidade compartilhada. Os nós do EKS Auto Mode usam uma variante especializada do Bottlerocket que é otimizada e mantida pela AWS.

P: Com que frequência os nós são girados automaticamente no modo automático do EKS? R: Os nós no modo automático EKS têm uma vida útil máxima de 21 dias. Eles serão substituídos automaticamente antes desse limite, garantindo atualizações regulares de segurança e aplicação de patches.

P: Posso usar SSH nos nós do EKS Auto Mode para solucionar problemas? R: Não, o acesso direto ao SSH não está disponível no modo automático do EKS. Em vez disso, você pode usar a Definição de Recursos NodeDiagnostic Personalizados (CRD) para coletar registros do sistema e informações de depuração.

P: O suporte à política de rede está ativado por padrão no modo automático do EKS? R: Por enquanto, o suporte à política de rede precisa ser explicitamente ativado por meio da configuração do complemento VPC CNI. Depois de habilitado, você pode usar as políticas de rede padrão do Kubernetes.

P: Devo usar o IRSA ou o Pod Identity com o EKS Auto Mode? R: Embora ambos sejam compatíveis, o Pod Identity é a abordagem recomendada no Modo Automático do EKS, pois ele já inclui o complemento do agente Pod Identity Security e fornece recursos de segurança aprimorados e gerenciamento simplificado.

P: Ainda posso usar o aws-auth ConfigMap no modo automático do EKS? R: Esse `aws-auth` ConfigMap é um recurso obsoleto. É recomendável usar a abordagem padrão de autenticação baseada em API para aumentar a segurança e simplificar o gerenciamento de acesso.

P: Como posso monitorar eventos de segurança no EKS Auto Mode? R: O EKS Auto Mode se integra a várias soluções de monitoramento GuardDuty, incluindo CloudWatch, e. CloudTrail GuardDuty fornece monitoramento aprimorado da segurança do tempo de execução, especificamente para cargas de trabalho do EKS.

P: Como faço para coletar registros dos nós do EKS Auto Mode? R: Use o NodeDiagnostic CRD, que carrega automaticamente os registros em um bucket do S3. Você também pode usar o CloudWatch Container Insights e o AWS Distro para OpenTelemetry.

> **Note**
>
> Esta seção de perguntas frequentes é atualizada regularmente à medida que novos recursos são adicionados ao Modo Automático do EKS e à medida que recebemos perguntas comuns dos clientes.

Gerenciamento de Identidade e Acesso

O Identity and Access Management (IAM) é um serviço da AWS que executa duas funções essenciais: autenticação e autorização. A autenticação envolve a verificação de uma identidade, enquanto a autorização rege as ações que podem ser executadas pelos recursos da AWS. Dentro da AWS, um recurso pode ser outro serviço da AWS, por exemplo EC2, ou um diretor da AWS, como um usuário ou função do IAM. As regras que regem as ações que um recurso pode realizar são expressas como políticas do IAM.

Controlando o acesso aos clusters EKS

O projeto Kubernetes oferece suporte a uma variedade de estratégias diferentes para autenticar solicitações no serviço kube-apiserver, por exemplo, Bearer Tokens, certificados X.509, OIDC etc. Atualmente, o EKS tem suporte nativo para autenticação de token de webhook, tokens de conta de serviço e, a partir de 21 de fevereiro de 2021, autenticação OIDC.

A estratégia de autenticação do webhook chama um webhook que verifica os tokens do portador. No EKS, esses tokens portadores são gerados pela AWS CLI ou pelo cliente quando você aws-iam-authenticator`kubectl`executa comandos. Conforme você executa os comandos, o token é passado para o kube-apiserver, que o encaminha para o webhook de autenticação. Se a solicitação for bem formada, o webhook chamará uma URL pré-assinada incorporada no corpo do token. Esse URL valida a assinatura da solicitação e retorna informações sobre o usuário, por exemplo, a conta do usuário, Arn, e UserId para o kube-apiserver.

Para gerar manualmente um token de autenticação, digite o seguinte comando em uma janela do terminal:

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

A saída deve ser semelhante a esta:

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

Você também pode obter um token programaticamente. Abaixo está um exemplo escrito em 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)
}
```

A saída deve ser semelhante a esta:

```
{
  "kind": "ExecCredential",
  "apiVersion": "client.authentication.k8s.io/v1alpha1",
```

```
  "spec": {},
  "status": {
    "expirationTimestamp": "2020-02-19T16:08:27Z",
    "token": "k8s-aws-
v1.aHR0cHM6Ly9zdHMuYW1hem9uYXdzLmNvbS8_QWN0aW9uPUdldENhbGxlcklkZW50aXR5JlZlcnNpb249MjAxMS0wNi0x
  }
}
```

Cada token começa com `k8s-aws-v1.` seguido por uma string codificada em base64. A string, quando decodificada, deve se parecer com algo parecido com isso:

```
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
```

O token consiste em uma URL pré-assinada que inclui uma credencial e uma assinatura da Amazon. Para obter detalhes adicionais, consulte https://docs.aws.amazon.com/STS/ latest/APIReference/API _ GetCallerIdentity .html.

O token tem um tempo de vida (TTL) de 15 minutos, após o qual um novo token precisará ser gerado. Isso é tratado automaticamente quando você usa um cliente`kubectl`, como, no entanto, se você estiver usando o painel do Kubernetes, precisará gerar um novo token e se autenticar novamente sempre que o token expirar.

Depois que a identidade do usuário é autenticada pelo serviço AWS IAM, o kube-apiserver lê o `aws-auth` ConfigMap no `kube-system` namespace para determinar o grupo RBAC a ser associado ao usuário. O `aws-auth` ConfigMap é usado para criar um mapeamento estático entre os principais do IAM, ou seja, usuários e funções do IAM e grupos RBAC do Kubernetes. Os grupos RBAC podem ser referenciados no Kubernetes ou. RoleBindings ClusterRoleBindings Eles são semelhantes aos papéis do IAM, pois definem um conjunto de ações (verbos) que podem ser executadas em uma coleção de recursos (objetos) do Kubernetes.

CloudWatch consulta para ajudar os usuários a identificar clientes que enviam solicitações para o endpoint STS global

Execute a CloudWatch consulta abaixo para obter o endpoint sts. Se stsendpoint for igual a “sts.amazonaws.com”, então é um endpoint STS global. Se stsendpoint for igual a “sts.<region>.amazonaws.com”, então é um endpoint regional do STS.

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

Gerenciador de acesso ao cluster

O Cluster Access Manager, agora a forma preferida de gerenciar o acesso dos diretores do AWS IAM aos clusters do Amazon EKS, é uma funcionalidade da API da AWS e é um recurso opcional para clusters do EKS v1.23 e posteriores (novos ou existentes). Ele simplifica o mapeamento de identidade entre o AWS IAM e o KubernetesRBACs, eliminando a necessidade de alternar entre a AWS e o Kubernetes APIs ou editá-los `aws-auth` ConfigMap para gerenciamento de acesso, reduzindo a sobrecarga operacional e ajudando a resolver configurações incorretas. A ferramenta também permite que os administradores do cluster revoguem ou refinem `cluster-admin` as permissões concedidas automaticamente ao principal do AWS IAM usadas para criar o cluster.

Essa API se baseia em dois conceitos:

- Entradas de acesso: uma identidade de cluster diretamente vinculada a um principal (usuário ou função) do AWS IAM com permissão para se autenticar em um cluster Amazon EKS.
- Políticas de acesso: são políticas específicas do Amazon EKS que fornecem a autorização para que uma entrada de acesso execute ações no cluster do Amazon EKS.

No lançamento, o Amazon EKS oferece suporte somente a políticas predefinidas e gerenciadas pela AWS. As políticas de acesso não são entidades do IAM e são definidas e gerenciadas pelo Amazon EKS.

O Cluster Access Manager permite a combinação do RBAC upstream com políticas de acesso que permitem e aprovam (mas não negam) as decisões do Kubernetes AuthZ em relação às solicitações do servidor de API. Uma decisão de negação ocorrerá quando os autorizadores do RBAC upstream e do Amazon EKS não conseguirem determinar o resultado da avaliação de uma solicitação.

Com esse recurso, o Amazon EKS oferece suporte a três modos de autenticação:

1. `CONFIG_MAP`para continuar usando o `aws-auth` ConfigMap exclusivamente.

2. `API_AND_CONFIG_MAP`para obter os principais autenticados do IAM do EKS Access Entry APIs e do `aws-auth` ConfigMap, priorizando as entradas de acesso. Ideal para migrar as `aws-auth` permissões existentes para o Access Entries.
3. `API`confiar exclusivamente no EKS Access Entry APIs. Essa é a nova abordagem recomendada.

Para começar, os administradores de cluster podem criar ou atualizar clusters do Amazon EKS, definindo a autenticação `API_AND_CONFIG_MAP` ou o `API` método preferido e definindo entradas de acesso para conceder acesso aos principais do AWS IAM desejados.

```
$ 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
```

O comando acima é um exemplo para criar um cluster Amazon EKS sem as permissões de administrador do criador do cluster.

É possível atualizar a configuração de clusters do Amazon EKS para habilitar o `API` AuthenticationMode usando o `update-cluster-config` comando. Para fazer isso em clusters existentes, `CONFIG_MAP` você precisará primeiro atualizar para `API_AND_CONFIG_MAP` e depois para. `API` Essas operações não podem ser revertidas, o que significa que não é possível alternar de `API` para `API_AND_CONFIG_MAP` ou `CONFIG_MAP` e também de `API_AND_CONFIG_MAP` para`CONFIG_MAP`.

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

A API suporta comandos para adicionar e revogar o acesso ao cluster, bem como validar as políticas de acesso e entradas de acesso existentes para o cluster especificado. As políticas padrão são criadas para corresponder ao Kubernetes RBACs da seguinte maneira.

Política de acesso ao EKS	Kubernetes RBAC
Amazon EKSCluster AdminPolicy	administrador de cluster
EKSAdminPolítica da Amazon	admin
EKSEditPolítica da Amazon	editar
EKSViewPolítica da Amazon	visualizar

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

Nenhuma entrada de acesso está disponível quando o cluster é criado sem a permissão de administrador do criador do cluster, que é a única entrada criada por padrão.

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

Uma forma pela qual a integração do Kubernetes com a autenticação da AWS pode ser feita é por meio do `aws-auth` ConfigMap, que reside no Namespace. `kube-system` Ela é responsável por mapear a autenticação de identidades (usuários, grupos e funções) do AWS IAM para a autorização de controle de acesso baseado em funções (RBAC) do Kubernetes. O `aws-auth` ConfigMap é criado automaticamente em seu cluster Amazon EKS durante a fase de provisionamento. Ele foi criado inicialmente para permitir que os nós se juntassem ao seu cluster, mas, conforme mencionado, você também pode usá-lo ConfigMap para adicionar RBACs acesso aos principais do IAM.

Para verificar o do seu cluster `aws-auth` ConfigMap, você pode usar o comando a seguir.

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

Este é um exemplo de uma configuração padrão do `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
```

A sessão principal desta ConfigMap, está abaixo `data` do `mapRoles` bloco, que é composto basicamente por 3 parâmetros.

- grupos: o Kubernetes group/groups para o qual mapear a função do IAM. Isso pode ser um grupo padrão ou um grupo personalizado especificado em um `clusterrolebinding` ou`rolebinding`. No exemplo acima, temos apenas grupos de sistemas declarados.

- rolearn: O ARN da função AWS IAM deve ser mapeado para a adição de grupos/grupos do Kubernetes, usando o formato a seguir. `arn:<PARTITION>:iam::<AWS_ACCOUNT_ID>:role/role-name`
- nome de usuário: o nome de usuário no Kubernetes a ser mapeado para a função AWS IAM. Isso pode ser qualquer nome personalizado.

Também é possível mapear permissões para usuários do AWS IAM, definindo um novo bloco de configuração para`mapUsers`, abaixo `data` de `aws-auth`ConfigMap, substituir o parâmetro rolearn por userarn. No entanto, como melhor prática, é sempre recomendável que o usuário use. `mapRoles`

Para gerenciar permissões, você pode editar a `aws-auth` ConfigMap adição ou remoção do acesso ao seu cluster Amazon EKS. Embora seja possível editá-los `aws-auth` ConfigMap manualmente, é recomendável usar ferramentas como`eksctl`, já que essa é uma configuração muito sensível, e uma configuração imprecisa pode bloquear você fora do seu cluster Amazon EKS. Consulte a subseção Usar ferramentas para fazer alterações no aws-auth ConfigMap abaixo para obter mais detalhes.

Benefícios ConfigMap em relação ao gerenciamento de acesso baseado

1. Risco reduzido de configurações incorretas: o gerenciamento direto baseado em API elimina erros comuns associados à edição manual. ConfigMap Isso ajuda a evitar exclusões acidentais ou erros de sintaxe que podem impedir que os usuários saiam do cluster.
2. Princípio aprimorado de privilégios mínimos: elimina a necessidade de permissão de administrador de cluster da identidade do criador do cluster e permite uma atribuição de permissões mais granular e apropriada. Você pode optar por adicionar essa permissão para casos de uso de quebra-vidros.
3. Modelo de segurança aprimorado: fornece validação integrada das entradas de acesso antes de serem aplicadas. Além disso, oferece maior integração com o AWS IAM para autenticação.
4. Operações simplificadas: oferece uma maneira mais intuitiva de gerenciar permissões por meio de ferramentas nativas da AWS.

Recomendações de acesso ao cluster

Combine o IAM Identity Center com a API CAM

- Gerenciamento simplificado: ao usar a API Cluster Access Management em conjunto com o IAM Identity Center, os administradores podem gerenciar o acesso ao cluster EKS junto com outros serviços da AWS, reduzindo a necessidade de alternar entre diferentes interfaces ou editar ConfigMaps manualmente.
- Use entradas de acesso para gerenciar as permissões do Kubernetes das entidades principais do IAM de fora do cluster. Você pode adicionar e gerenciar o acesso ao cluster usando a API EKS, a interface de linha de comando da AWS, a AWS SDKs, a AWS CloudFormation e o AWS Management Console. Isso significa que você pode gerenciar usuários com as mesmas ferramentas com as quais criou o cluster.
- Aproveite a automação, conforme demonstrado neste exemplo, para implantar clusters com o AWS IAM Identity Center como IdP e a API CAM como ponto de entrada.
- As permissões granulares do Kubernetes podem ser aplicadas com o mapeamento de usuários ou grupos do Kubernetes com diretores do IAM associados a identidades de SSO por meio de entradas de acesso e políticas de acesso.
- Para começar, siga Alterar o modo de autenticação para usar entradas de acesso e, em seguida, Migrar entradas aws-auth existentes para ConfigMap entradas de acesso.

Torne o EKS Cluster Endpoint privado

Por padrão, quando você provisiona um cluster EKS, o endpoint do cluster da API é definido como público, ou seja, ele pode ser acessado pela Internet. Apesar de ser acessível pela Internet, o endpoint ainda é considerado seguro porque exige que todas as solicitações de API sejam autenticadas pelo IAM e depois autorizadas pelo Kubernetes RBAC. Dito isso, se sua política de segurança corporativa exigir que você restrinja o acesso à API pela Internet ou impeça o roteamento do tráfego para fora do cluster VPC, você pode:

- Configure o endpoint do cluster EKS para ser privado. Consulte Modificando o acesso ao Cluster Endpoint para obter mais informações sobre esse tópico.
- Deixe o endpoint do cluster público e especifique quais blocos CIDR podem se comunicar com o endpoint do cluster. Os blocos são, na verdade, um conjunto de endereços IP públicos na lista branca que têm permissão para acessar o endpoint do cluster.

- Configure o acesso público com um conjunto de blocos CIDR na lista branca e defina o acesso ao endpoint privado como ativado. Isso permitirá o acesso público de uma faixa específica de público e, IPs ao mesmo tempo, forçará todo o tráfego de rede entre os kubelets (trabalhadores) e a API do Kubernetes por meio da conta cruzada que é provisionada ENIs na VPC do cluster quando o plano de controle é provisionado.

Não use um token de conta de serviço para autenticação

Um token de conta de serviço é uma credencial estática de longa duração. Se ele for comprometido, perdido ou roubado, um invasor poderá realizar todas as ações associadas a esse token até que a conta de serviço seja excluída. Às vezes, talvez seja necessário conceder uma exceção para aplicativos que precisam consumir a API Kubernetes de fora do cluster, por exemplo, um aplicativo de pipeline de CI/CD. Se esses aplicativos forem executados na infraestrutura da AWS, como EC2 instâncias, considere usar um perfil de instância e mapeá-lo para uma função RBAC do Kubernetes.

Empregue o acesso menos privilegiado aos recursos da AWS

Um usuário do IAM não precisa ter privilégios atribuídos aos recursos da AWS para acessar a API do Kubernetes. Se você precisar conceder a um usuário do IAM acesso a um cluster EKS, crie uma entrada `aws-auth` ConfigMap para esse usuário que mapeie para um grupo RBAC específico do Kubernetes.

Remova as permissões de administrador do cluster do criador do cluster

Por padrão, os clusters do Amazon EKS são criados com uma `cluster-admin` permissão permanente vinculada ao principal criador do cluster. Com a API Cluster Access Manager, é possível criar clusters sem essa configuração de permissão`false`, `--access-config` `bootstrapClusterCreatorAdminPermissions` ao usar `API_AND_CONFIG_MAP` nosso modo de `API` autenticação. Revogar esse acesso é considerado uma prática recomendada para evitar alterações indesejadas na configuração do cluster. O processo para revogar esse acesso segue o mesmo processo para revogar qualquer outro acesso ao cluster.

A API oferece flexibilidade para dissociar apenas um principal do IAM de uma política de acesso, neste caso o. `AmazonEKSClusterAdminPolicy`

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

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

Ou remover completamente a entrada de acesso associada à `cluster-admin` permissão.

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

{
    "accessEntries": []
}

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

Esse acesso pode ser concedido novamente, se necessário, durante um cenário de incidente, emergência ou quebra de vidro em que o cluster esteja inacessível.

Se o cluster ainda estiver configurado com o método de `CONFIG_MAP` autenticação, todos os usuários adicionais deverão ter acesso ao cluster por meio do `aws-auth` ConfigMap, e depois de `aws-auth` ConfigMap configurado, a função atribuída à entidade que criou o cluster poderá ser excluída e recriada somente em caso de incidente, emergência ou cenário de quebra de vidro, ou quando a função `aws-auth` ConfigMap estiver corrompida e o cluster estiver inacessível. Isso pode ser particularmente útil em clusters de produção.

Use funções do IAM quando vários usuários precisarem de acesso idêntico ao cluster

Em vez de criar uma entrada para cada usuário individual do IAM, permita que esses usuários assumam uma função do IAM e mapeiem essa função para um grupo RBAC do Kubernetes. Isso será mais fácil de manter, especialmente à medida que o número de usuários que precisam de acesso aumentar.

> **Important**
>
> Ao acessar o cluster EKS com a entidade IAM mapeada por `aws-auth` ConfigMap, o nome de usuário descrito é registrado no campo de usuário do registro de auditoria do Kubernetes. Se você estiver usando uma função do IAM, os usuários reais que assumem essa função não são registrados e não podem ser auditados.

Se ainda estiver usando o `aws-auth` ConfigMap como método de autenticação, ao atribuir permissões RBAC do K8s a uma função do IAM, você deve incluir\ {{}} em seu nome de usuário. SessionName Dessa forma, o registro de auditoria registrará o nome da sessão para que você possa acompanhar quem o usuário real assume essa função junto com o CloudTrail registro.

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

> No Kubernetes 1.20 e versões posteriores, essa alteração não é mais necessária, pois `user.extra.sessionName.0` foi adicionada ao registro de auditoria do Kubernetes.

Empregue o acesso menos privilegiado ao criar e RoleBindings ClusterRoleBindings

Como no ponto anterior sobre a concessão de acesso aos recursos da AWS, RoleBindings ClusterRoleBindings deve incluir apenas o conjunto de permissões necessárias para realizar uma função específica. Evite usar `["*"]` em suas funções, a ClusterRoles menos que seja absolutamente necessário. Se você não tiver certeza de quais permissões atribuir, considere usar uma ferramenta como audit2rbac para gerar automaticamente funções e vinculações com base nas chamadas de API observadas no registro de auditoria do Kubernetes.

Crie um cluster usando um processo automatizado

Conforme visto nas etapas anteriores, ao criar um cluster Amazon EKS, se não estiver usando o modo de uso `API_AND_CONFIG_MAP` ou `API` autenticação e não optar por não delegar `cluster-admin` permissões ao criador do cluster, o usuário ou a função da entidade IAM, como um usuário federado que cria o cluster, recebe automaticamente `system:masters` permissões na configuração RBAC do cluster. Mesmo sendo uma prática recomendada remover essa permissão, conforme descrito aqui, se você usar o método de `CONFIG_MAP` autenticação, confie nele `aws-auth` ConfigMap, esse acesso não pode ser revogado. Portanto, é uma boa ideia criar o cluster com um pipeline de automação de infraestrutura vinculado à função dedicada do IAM, sem permissões a serem assumidas por outros usuários ou entidades, e auditar regularmente as permissões e políticas dessa função e quem tem acesso para acionar o pipeline. Além disso, essa função não deve ser usada para realizar ações rotineiras no cluster e ser usada exclusivamente para ações em nível de cluster acionadas pelo pipeline, por meio de alterações no código SCM, por exemplo.

Crie o cluster com uma função IAM dedicada

Quando você cria um cluster Amazon EKS, o usuário ou a função da entidade IAM, como um usuário federado que cria o cluster, recebe automaticamente `system:masters` permissões na configuração RBAC do cluster. Esse acesso não pode ser removido e não é gerenciado por meio do `aws-auth` ConfigMap. Portanto, é uma boa ideia criar o cluster com uma função dedicada do IAM e auditar regularmente quem pode assumir essa função. Essa função não deve ser usada para realizar ações rotineiras no cluster e, em vez disso, usuários adicionais devem ter acesso ao cluster por meio do `aws-auth` ConfigMap para essa finalidade. Depois de `aws-auth` ConfigMap configurada, a função deve ser protegida e usada somente no modo temporário de privilégios elevados /break glass para cenários em que o cluster estaria inacessível. Isso pode ser particularmente útil em clusters que não têm o acesso direto do usuário configurado.

Audite regularmente o acesso ao cluster

Quem precisa de acesso provavelmente mudará com o tempo. Planeje auditar periodicamente o `aws-auth` ConfigMap para ver a quem foi concedido acesso e os direitos que lhes foram atribuídos. Você também pode usar ferramentas de código aberto kubectl-who-can, como rbac-lookup, para examinar as funções vinculadas a uma conta de serviço, usuário ou grupo específico. Exploraremos esse tópico mais detalhadamente quando chegarmos à seção sobre auditoria. Ideias adicionais podem ser encontradas neste artigo do NCC Group.

Se estiver confiando no **`aws-auth`** ConfigMap, use ferramentas para fazer alterações

Um aws-auth formatado incorretamente ConfigMap pode fazer com que você perca o acesso ao cluster. Se você precisar fazer alterações no ConfigMap, use uma ferramenta.

eksctl A `eksctl` CLI inclui um comando para adicionar mapeamentos de identidade ao aws-auth. ConfigMap

Veja a ajuda da CLI:

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

Verifique as identidades mapeadas para o seu cluster Amazon EKS.

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

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

Transforme uma função do IAM em administrador de cluster:

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

Para obter mais informações, revise os `eksctl`documentos

aws-auth por keikoproj

`aws-auth`by keikoproj inclui uma biblioteca cli e uma biblioteca go.

Baixe e veja a ajuda da CLI de ajuda:

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

Como alternativa, instale `aws-auth` com o gerenciador de plug-ins krew para kubectl.

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

Consulte a documentação do aws-auth em GitHub para obter mais informações, incluindo a biblioteca go.

CLI do AWS IAM Authenticator

O `aws-iam-authenticator` projeto inclui uma CLI para atualizar o. ConfigMap

Baixe uma versão em GitHub.

Adicione permissões de cluster a uma função do 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
...
```

Abordagens alternativas para autenticação e gerenciamento de acesso

Embora o IAM seja a forma preferida de autenticar usuários que precisam acessar um cluster EKS, é possível usar um provedor de identidade OIDC, como GitHub usar um proxy de autenticação e a representação do Kubernetes. Publicações sobre duas dessas soluções foram publicadas no blog de código aberto da AWS:

- Autenticação no EKS usando GitHub credenciais com teleporte
- Autenticação OIDC consistente em vários clusters EKS usando kube-oidc-proxy

> **Important**
>
> O EKS oferece suporte nativo à autenticação OIDC sem usar um proxy. Para obter mais informações, leia o blog de lançamento, Apresentando a autenticação do provedor de identidade OIDC para o Amazon EKS. Para ver um exemplo de como configurar o EKS com o Dex, um popular provedor OIDC de código aberto com conectores para uma variedade de métodos de autenticação diferentes, consulte Uso do Dex e do autenticador dex-k8s-para autenticação no Amazon EKS. Conforme descrito nos blogs, os username/group usuários autenticados por um provedor OIDC aparecerão no registro de auditoria do Kubernetes.

Você também pode usar o AWS SSO para federar a AWS com um provedor de identidade externo, por exemplo, Azure AD. Se você decidir usar isso, a AWS CLI v2.0 inclui uma opção para criar um perfil nomeado que facilita a associação de uma sessão de SSO à sua sessão atual da CLI e a assunção de uma função do IAM. Saiba que você deve assumir uma função antes de executar, `kubectl` pois a função do IAM é usada para determinar o grupo RBAC do Kubernetes do usuário.

Identidades e credenciais para pods EKS

Certos aplicativos executados em um cluster Kubernetes precisam de permissão para chamar a API do Kubernetes para funcionar corretamente. Por exemplo, o AWS Load Balancer Controller precisa ser capaz de listar os endpoints de um serviço. O controlador também precisa ser capaz de invocar

a AWS APIs para provisionar e configurar um ALB. Nesta seção, exploraremos as melhores práticas para atribuir direitos e privilégios aos pods.

Contas de serviço do Kubernetes

Uma conta de serviço é um tipo especial de objeto que permite atribuir uma função RBAC do Kubernetes a um pod. Uma conta de serviço padrão é criada automaticamente para cada namespace em um cluster. Quando você implanta um pod em um namespace sem fazer referência a uma conta de serviço específica, a conta de serviço padrão desse Namespace será automaticamente atribuída ao pod e o segredo, ou seja, o token da conta de serviço (JWT) dessa conta de serviço, será montado no pod como um volume em. `/var/run/secrets/kubernetes.io/serviceaccount` A decodificação do token da conta de serviço nesse diretório revelará os seguintes metadados:

```
{
  "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"
}
```

A conta de serviço padrão tem as seguintes permissões para a API Kubernetes.

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

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

Essa função autoriza usuários não autenticados e autenticados a ler as informações da API e é considerada segura para ser acessível ao público.

Quando um aplicativo executado em um pod chama o Kubernetes APIs, é necessário atribuir ao pod uma conta de serviço que conceda explicitamente permissão para chamá-los. APIs Semelhante às diretrizes para acesso do usuário, a função ou a ClusterRole vinculação a uma conta de serviço devem ser restritas aos recursos e métodos da API de que o aplicativo precisa para funcionar e nada mais. Para usar uma conta de serviço não padrão, basta definir o `spec.serviceAccountName` campo de um pod com o nome da conta de serviço que você deseja usar. Para obter informações adicionais sobre a criação de contas de serviço, consulte https://kubernetes. io/docs/reference/access-authn-authz/rbac/#service-account-permissions.

> **Note**
>
> Antes do Kubernetes 1.24, o Kubernetes criava automaticamente um segredo para cada conta de serviço. Esse segredo foi montado no pod em/var/run/secrets/kubernetes.io/serviceaccounte seria usado pelo pod para se autenticar no servidor da API Kubernetes. No Kubernetes 1.24, um token de conta de serviço é gerado dinamicamente quando o pod é executado e, por padrão, só é válido por uma hora. Um segredo para a conta de serviço não será criado. Se você tiver um aplicativo executado fora do cluster que precisa se autenticar na API Kubernetes, por exemplo, Jenkins, precisará criar um segredo do tipo `kubernetes.io/service-account-token` junto com uma anotação que faça referência à conta de serviço, como. `metadata.annotations.kubernetes.io/service-account.name: <SERVICE_ACCOUNT_NAME>` Os segredos criados dessa maneira não expiram.

Funções do IAM para contas de serviço (IRSA)

O IRSA é um recurso que permite atribuir uma função do IAM a uma conta de serviço do Kubernetes. Ele funciona aproveitando um recurso do Kubernetes conhecido como Projeção de Volume do Token da Conta de Serviço. Quando os pods são configurados com uma conta de serviço que faz referência a uma função do IAM, o servidor da API Kubernetes chamará o endpoint público de descoberta do OIDC para o cluster na inicialização. O endpoint assina criptograficamente o token OIDC emitido pelo Kubernetes e o token resultante montado como um volume. Esse token assinado permite que o pod chame a função IAM APIs associada à AWS. Quando uma API da AWS é invocada, a AWS SDKs chama`sts:AssumeRoleWithWebIdentity`. Depois de validar a assinatura do token, o IAM troca o token emitido pelo Kubernetes por uma credencial de função temporária da AWS.

Ao usar o IRSA, é importante reutilizar as sessões do AWS SDK para evitar chamadas desnecessárias para o AWS STS.

A decodificação do token (JWT) para IRSA produzirá uma saída semelhante ao exemplo que você vê abaixo:

```
{
  "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"
}
```

Esse token específico concede privilégios somente de visualização do Pod ao S3 ao assumir uma função do IAM. Quando o aplicativo tenta ler do S3, o token é trocado por um conjunto temporário de credenciais do IAM semelhante a este:

```
{
    "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"
    }
}
```

Um webhook mutante que é executado como parte do plano de controle do EKS injeta o ARN da função da AWS e o caminho para um arquivo de token de identidade da web no pod como variáveis de ambiente. Esses valores também podem ser fornecidos 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
```

O kubelet rotacionará automaticamente o token projetado quando ele tiver mais de 80% do TTL total ou após 24 horas. A AWS SDKs é responsável por recarregar o token quando ele gira. Para obter mais informações sobre o IRSA, consulte https://docs.aws.amazon.com/eks/ latest/userguide/iam - roles-for-service-accounts -technical-overview.html.

EKS Pod Identities

O EKS Pod Identities é um recurso lançado no re:Invent 2023 que permite atribuir uma função do IAM a uma conta de serviço do Kubernetes, sem a necessidade de configurar um provedor de identidade (IDP) do Open Id Connect (OIDC) para cada cluster em sua conta da AWS. Para usar o EKS Pod Identity, você deve implantar um agente que seja executado como um DaemonSet pod em cada nó de trabalho qualificado. Esse agente é disponibilizado para você como um complemento do EKS e é um pré-requisito para usar o recurso EKS Pod Identity. Seus aplicativos devem usar uma versão compatível do AWS SDK para usar esse recurso.

Quando as identidades de pod do EKS são configuradas para um pod, o EKS monta e atualiza um token de identidade do pod em. `/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token` Esse token será usado pelo AWS SDK para se comunicar com o EKS Pod Identity Agent, que usa o token de identidade do pod e a função IAM do agente para criar credenciais temporárias para seus pods chamando a API. AssumeRoleForPodIdentity O token de identidade do pod entregue aos seus pods é um JWT emitido do seu cluster EKS e assinado criptograficamente, com declarações JWT apropriadas para uso com o EKS Pod Identities.

Para saber mais sobre o EKS Pod Identities, consulte este blog.

Você não precisa fazer nenhuma modificação no código do aplicativo para usar o EKS Pod Identities. As versões compatíveis do AWS SDK descobrirão automaticamente as credenciais disponibilizadas com o EKS Pod Identities usando a cadeia de fornecedores de credenciais. Assim como o IRSA, as identidades de pods do EKS definem variáveis em seus pods para orientá-los a encontrar as credenciais da AWS.

Trabalhando com funções do IAM para EKS Pod Identities

- O EKS Pod Identities só pode assumir diretamente uma função do IAM que pertença à mesma conta da AWS do cluster EKS. Para acessar uma função do IAM em outra conta da AWS, você deve assumir essa função configurando um perfil na configuração do SDK ou no código do seu aplicativo.
- Quando as identidades de pod do EKS estão sendo configuradas para contas de serviço, a pessoa ou o processo que configura a associação de identidade do pod deve ter o `iam:PassRole` direito para essa função.
- Cada conta de serviço pode ter apenas uma função do IAM associada a ela por meio do EKS Pod Identities, mas você pode associar a mesma função do IAM a várias contas de serviço.

- As funções do IAM usadas com o EKS Pod Identities devem permitir que o responsável pelo `pods.eks.amazonaws.com` serviço as assuma e defina tags de sessão. Veja a seguir um exemplo de política de confiança de função que permite que o EKS Pod Identities use uma função do 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}"
        }
      }
    }
  ]
}
```

A AWS recomenda o uso de chaves de condição `aws:SourceOrgId` para ajudar a se proteger contra o problema confuso de representantes entre serviços. No exemplo acima, a política de confiança da função `ResourceOrgId` é uma variável igual ao ID da organização do AWS Organizations da organização da AWS à qual a conta da AWS pertence. O EKS passará um valor `aws:SourceOrgId` igual a esse ao assumir uma função com o EKS Pod Identities.

Identidades de pod ABAC e EKS

Quando o EKS Pod Identities assume uma função do IAM, ele define as seguintes tags de sessão:

Tag de sessão EKS Pod Identities	Valor
namespace kubernetes	O namespace em que o pod associado ao EKS Pod Identities é executado.

Tag de sessão EKS Pod Identities	Valor
kubernetes-service-account	O nome da conta de serviço kubernetes associada ao EKS Pod Identities
eks-cluster-arn	O ARN do cluster EKS, por exemplo `arn:${Partition}:eks:${Region}:${Account}:cluster/${ClusterName}` O ARN do cluster é exclusivo, mas se um cluster for excluído e recriado na mesma região com o mesmo nome, dentro da mesma conta da AWS, ele terá o mesmo ARN.
eks-cluster-name	O nome do cluster EKS. Observe que os nomes dos clusters do EKS podem ser os mesmos na sua conta da AWS e dos clusters do EKS em outras contas da AWS.
kubernetes-pod-name	O nome do pod no EKS.
kubernetes-pod-uid	O UID do pod no EKS.

Essas tags de sessão permitem que você use o Attribute Based Access Control (ABAC) para conceder acesso aos seus recursos da AWS somente para contas específicas do serviço Kubernetes. Ao fazer isso, é muito importante entender que as contas de serviço do Kubernetes são exclusivas somente dentro de um namespace, e os namespaces do Kubernetes são exclusivos somente dentro de um cluster EKS. Essas tags de sessão podem ser acessadas nas políticas da AWS usando a chave de condição `aws:PrincipalTag/<tag-key>` global, como `aws:PrincipalTag/eks-cluster-arn`

Por exemplo, se você quiser conceder acesso somente a uma conta de serviço específica para acessar um recurso da AWS em sua conta com um IAM ou uma política de recursos, você precisaria verificar as `kubernetes-namespace` tags `eks-cluster-arn` e também as tags `kubernetes-service-account` para garantir que somente as contas de serviço do cluster pretendido tenham acesso a esse recurso, pois outros clusters poderiam ter acesso idêntico `kubernetes-service-accounts` `kubernetes-namespaces` e.

Este exemplo de política do S3 Bucket só concede acesso aos objetos no bucket do S3 ao qual está anexado, somente se`kubernetes-service-account`,`kubernetes-namespace`, `eks-cluster-arn` todos atenderem aos valores esperados, onde o cluster EKS está hospedado na conta da AWS. `111122223333`

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

Identidades do EKS Pod em comparação com a IRSA

Tanto o EKS Pod Identities quanto o IRSA são formas preferidas de fornecer credenciais temporárias da AWS para seus pods EKS. A menos que você tenha casos de uso específicos para o IRSA, recomendamos que você use o EKS Pod Identities ao usar o EKS. Essa tabela ajuda a comparar os dois recursos.

#	EKS Pod Identities	IRSA
Requer permissão para criar um IDP do OIDC em suas contas da AWS?	Não	Sim
Requer configuração exclusiva de IDP por cluster	Não	Sim
Define tags de sessão relevantes para uso com o ABAC	Sim	Não
Requer um objetivo: PassRole Verificar?	Sim	Não
Usa a cota do AWS STS da sua conta da AWS?	Não	Sim
Pode acessar outras contas da AWS	Indiretamente com o encadeamento de funções	Diretamente com sts: AssumeRoleWithWebIdentity
Compatível com AWS SDKs	Sim	Sim
Requer o Pod Identity Agent Daemonset nos nós?	Sim	Não

Recomendações de identidades e credenciais para pods EKS

Atualize o daemonset aws-node para usar o IRSA

No momento, o daemonset aws-node está configurado para usar uma função atribuída às EC2 instâncias a serem atribuídas aos pods. IPs Essa função inclui várias políticas gerenciadas pela AWS, por exemplo, Amazoneks_CNI_Policy, que permitem efetivamente que todos os pods executados em um nó acessem EC2 ContainerRegistryReadOnly endereços IP ou extraiam imagens do ECR. attach/detach ENIs assign/unassign Como isso representa um risco para seu cluster, é recomendável que você atualize o daemonset aws-node para usar o IRSA. Um script para fazer isso pode ser encontrado no repositório deste guia.

O daemonset aws-node é compatível com EKS Pod Identities nas versões v1.15.5 e posteriores.

Restrinja o acesso ao perfil da instância atribuído ao nó de trabalho

Quando você usa o IRSA ou o EKS Pod Identities, ele atualiza a cadeia de credenciais do pod para usar o IRSA ou o EKS Pod Identities primeiro. No entanto, o pod ainda pode herdar os direitos do perfil da instância atribuído ao nó de trabalho. Para pods que não precisam dessas permissões, você pode bloquear o acesso aos metadados da instância para ajudar a garantir que seus aplicativos tenham apenas as permissões necessárias, e não seus nós.

> ⚠ Warning
>
> Bloquear o acesso aos metadados da instância impedirá que pods que não usam identidades de pod IRSA ou EKS herdem a função atribuída ao node de trabalho.

Você pode bloquear o acesso aos metadados da instância exigindo que a instância use IMDSv2 somente e atualizando a contagem de saltos para 1, como no exemplo abaixo. Você também pode incluir essas configurações no modelo de lançamento do grupo de nós. Não desative os metadados da instância, pois isso impedirá que componentes como o manipulador de terminação do nó e outras coisas que dependem dos metadados da instância funcionem corretamente.

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

Se você estiver usando o Terraform para criar modelos de lançamento para uso com grupos de nós gerenciados, adicione o bloco de metadados para configurar a contagem de saltos, conforme visto neste trecho 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" } …
```

Você também pode bloquear o acesso de um pod aos EC2 metadados manipulando iptables no nó. Para obter mais informações sobre esse método, consulte Limitar o acesso ao serviço de metadados da instância.

Se você tem um aplicativo que está usando uma versão mais antiga do AWS SDK que não é compatível com IRSA ou EKS Pod Identities, você deve atualizar a versão do SDK.

Defina o escopo da política de confiança do IAM Role para IRSA Roles para o nome da conta de serviço, namespace e cluster

A política de confiança pode ter como escopo um namespace ou uma conta de serviço específica dentro de um namespace. Ao usar o IRSA, é melhor tornar a política de confiança da função o mais explícita possível, incluindo o nome da conta de serviço. Isso evitará efetivamente que outros pods no mesmo namespace assumam a função. A CLI `eksctl` fará isso automaticamente quando você a usar para criar funções de serviço accounts/IAM . Consulte https://eksctl. io/usage/iamserviceaccounts/para obter mais informações.

Ao trabalhar diretamente com o IAM, isso significa adicionar uma condição à política de confiança da função que usa condições para garantir que a `:sub` declaração seja o namespace e a conta de serviço que você espera. Como exemplo, antes tínhamos um token IRSA com uma subdeclaração de “system:serviceaccount:default:s3-read-only”. Esse é o `default` namespace e a conta de serviço é. `s3-read-only` Você usaria uma condição como a seguinte para garantir que somente sua conta de serviço em um determinado namespace do seu cluster possa assumir essa função:

```
 "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"
     }
 }
```

Use uma função do IAM por aplicativo

Com o IRSA e o EKS Pod Identity, é uma prática recomendada dar a cada aplicativo sua própria função de IAM. Isso proporciona um isolamento aprimorado, pois você pode modificar um aplicativo sem afetar outro e permite aplicar o princípio de menor privilégio concedendo a um aplicativo apenas as permissões necessárias.

Ao usar o ABAC com o EKS Pod Identity, você pode usar uma função comum do IAM em várias contas de serviço e confiar nos atributos de sessão para controle de acesso. Isso é especialmente útil ao operar em grande escala, pois o ABAC permite que você opere com menos funções do IAM.

Quando seu aplicativo precisar acessar o IMDS, use IMDSv2 e aumente o limite de salto nas EC2 instâncias para 2

IMDSv2exige que você use uma solicitação PUT para obter um token de sessão. A solicitação PUT inicial precisa incluir um TTL para o token da sessão. As versões mais recentes da AWS SDKs lidarão com isso e com a renovação do referido token automaticamente. Também é importante estar ciente de que o limite de salto padrão nas EC2 instâncias é definido intencionalmente como 1 para evitar o encaminhamento de IP. Como consequência, os pods que solicitam um token de sessão que é executado em EC2 instâncias podem eventualmente expirar e voltar a usar o fluxo de IMDSv1 dados. O EKS adiciona suporte IMDSv2 habilitando a v1 e a v2 e alterando o limite de saltos para 2 nos nós provisionados pelo eksctl ou com os modelos oficiais. CloudFormation

Desativar a montagem automática de tokens da conta de serviço

Se seu aplicativo não precisar chamar a API Kubernetes, defina o `automountServiceAccountToken` atributo como `false` in PodSpec para seu aplicativo ou corrija a conta de serviço padrão em cada namespace para que ela não seja mais montada automaticamente nos pods. Por exemplo:

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

Use contas de serviço dedicadas para cada aplicativo

Cada aplicativo deve ter sua própria conta de serviço dedicada. Isso se aplica às contas de serviço da API Kubernetes, bem como ao IRSA e ao EKS Pod Identity.

> **Important**
>
> Se você empregar uma blue/green abordagem para atualizações de cluster em vez de realizar uma atualização de cluster no local ao usar o IRSA, precisará atualizar a política de confiança de cada uma das funções do IRSA IAM com o endpoint OIDC do novo cluster. Uma atualização de blue/green cluster é quando você cria um cluster executando uma versão mais recente do Kubernetes junto com o cluster antigo e usa um balanceador de carga ou uma malha de serviços para transferir facilmente o tráfego dos serviços executados no cluster antigo para o novo cluster. Ao usar atualizações de blue/green cluster com o EKS Pod Identity, você criaria associações de identidade de pod entre as funções do IAM e as contas de serviço no novo cluster. E atualize a política de confiança da função do IAM se você tiver uma `sourceArn` condição.

Execute o aplicativo como um usuário não root

Os contêineres são executados como root por padrão. Embora isso permita que eles leiam o arquivo de token de identidade da web, executar um contêiner como root não é considerado uma prática recomendada. Como alternativa, considere adicionar o `spec.securityContext.runAsUser` atributo ao PodSpec. O valor de `runAsUser` é um valor arbitrário.

No exemplo a seguir, todos os processos dentro do pod serão executados com o ID de usuário especificado no `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" ]
```

Quando você executa um contêiner como usuário não root, isso impede que o contêiner leia o token da conta de serviço IRSA porque o token recebe permissões 0600 [root] por padrão. Se você atualizar o SecurityContext do seu contêiner para incluir fsgroup=65534 [Ninguém], isso permitirá que o contêiner leia o token.

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

No Kubernetes 1.19 e versões posteriores, essa alteração não é mais necessária e os aplicativos podem ler o token da conta de serviço IRSA sem adicioná-los ao grupo Ninguém.

Conceda acesso menos privilegiado aos aplicativos

O Action Hero é um utilitário que você pode executar junto com seu aplicativo para identificar as chamadas de API da AWS e as permissões de IAM correspondentes que seu aplicativo precisa para funcionar adequadamente. É semelhante ao IAM Access Advisor, pois ajuda você a limitar

gradualmente o escopo das funções do IAM atribuídas aos aplicativos. Consulte a documentação sobre como conceder acesso menos privilegiado aos recursos da AWS para obter mais informações.

Considere definir um limite de permissões nas funções do IAM usadas com IRSA e Pod Identities. Você pode usar o limite de permissões para garantir que as funções usadas pelo IRSA ou pelo Pod Identities não excedam o nível máximo de permissões. Para obter um exemplo de guia sobre como começar com limites de permissões com um exemplo de política de limite de permissões, consulte este repositório do github.

Revise e revogue o acesso anônimo desnecessário ao seu cluster EKS

Idealmente, o acesso anônimo deve ser desativado para todas as ações da API. O acesso anônimo é concedido criando um RoleBinding ou ClusterRoleBinding para o sistema de usuário integrado do Kubernetes:anonymous. Você pode usar a ferramenta rbac-lookup para identificar as permissões que o usuário system:anonymous tem em seu cluster:

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

Qualquer função ou ClusterRole outra que não seja system: não public-info-viewer deve ser vinculada ao system:anonymous user ou system:unauthenticated group.

Pode haver alguns motivos legítimos para permitir o acesso anônimo em determinadas áreas APIs. Se esse for o caso do seu cluster, certifique-se de que somente aqueles específicos APIs sejam acessíveis por usuários anônimos e expô-los APIs sem autenticação não tornará seu cluster vulnerável.

Antes da Kubernetes/EKS versão 1.14, o grupo system:unauthenticated era associado a system:discovery e system:basic-user por padrão. ClusterRoles Observe que, mesmo que você tenha atualizado seu cluster para a versão 1.14 ou superior, essas permissões ainda podem estar habilitadas em seu cluster, já que as atualizações do cluster não revogam essas permissões. Para verificar quais ClusterRoles têm “system:unauthenticated”, exceto system: public-info-viewer você pode executar o seguinte comando (requer jq util):

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

E “system:unauthenticated” pode ser removido de todas as funções, exceto “system:” usando: 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 -
```

Como alternativa, você pode verificá-la e removê-la manualmente usando kubectl describe e kubectl edit. Para verificar se system:unauthenticated group tem permissões system:discovery em seu cluster, execute o seguinte 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 verificar se system:unauthenticated group tem permissão system:basic-user em seu cluster, execute o seguinte 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
```

Se o grupo system:unauthenticated estiver vinculado a system:discovery e/ou system:basic-user em seu cluster, você deverá desassociar essas funções do grupo ClusterRoles system:unauthenticated. Edite system:discovery ClusterRoleBinding usando o seguinte comando:

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

O comando acima abrirá a definição atual de system:discovery ClusterRoleBinding em um editor, conforme mostrado abaixo:

```
# 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
```

Exclua a entrada para system:unauthenticated group da seção “assuntos” na tela do editor acima.

Repita as mesmas etapas para ClusterRoleBinding system:basic-user.

Reutilize sessões do AWS SDK com a IRSA

Quando você usa o IRSA, os aplicativos criados usando o SDK da AWS usam o token entregue aos seus pods para fazer chamadas `sts:AssumeRoleWithWebIdentity` para gerar credenciais temporárias da AWS. Isso é diferente de outros serviços computacionais da AWS, nos quais o serviço computacional entrega credenciais temporárias da AWS diretamente para o recurso computacional da AWS, como uma função lambda. Isso significa que toda vez que uma sessão do AWS SDK é inicializada, uma chamada para o AWS STS for `AssumeRoleWithWebIdentity` é feita. Se seu aplicativo escalar rapidamente e inicializar muitas sessões do AWS SDK, você poderá enfrentar a limitação do AWS STS, pois seu código fará muitas chamadas. `AssumeRoleWithWebIdentity`

Para evitar esse cenário, recomendamos reutilizar as sessões do AWS SDK em seu aplicativo para que chamadas desnecessárias não `AssumeRoleWithWebIdentity` sejam feitas.

No código de exemplo a seguir, uma sessão é criada usando o SDK boto3 python, e essa mesma sessão é usada para criar clientes e interagir com o Amazon S3 e o Amazon SQS. `AssumeRoleWithWebIdentity`é chamado apenas uma vez, e o SDK da AWS atualizará as credenciais de `my_session` quando elas expirarem automaticamente.

```
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)
```

Se você estiver migrando um aplicativo de outro serviço computacional da AWS, como EC2, para o EKS com o IRSA, esse é um detalhe particularmente importante. Em outros serviços de computação, a inicialização de uma sessão do AWS SDK não chama o AWS STS, a menos que você o instrua.

Abordagens alternativas

Embora as identidades de pod IRSA e EKS sejam as formas preferidas de atribuir uma identidade da AWS a um pod, elas exigem que você inclua uma versão recente da AWS SDKs em seu aplicativo. Para obter uma lista completa dos SDKs que atualmente oferecem suporte ao 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. Se você tem um aplicativo que não pode ser atualizado imediatamente com um SDK compatível, há várias soluções criadas pela comunidade disponíveis para atribuir funções do IAM aos pods do Kubernetes, incluindo kube2iam e kiam. Embora a AWS não endosse, tolere nem apoie o uso dessas soluções, elas são frequentemente usadas pela comunidade em geral para obter resultados semelhantes aos das identidades IRSA e EKS Pod.

Se você precisar usar uma dessas soluções não fornecidas pela AWS, faça a devida diligência e certifique-se de entender as implicações de segurança de fazer isso.

Ferramentas e recursos

- Workshop de imersão em segurança do Amazon EKS — Identity and Access Management
- Padrão de projetos do Terraform EKS - Cluster Amazon EKS totalmente privado
- Padrão de blueprints do Terraform EKS - Login único do IAM Identity Center para o cluster Amazon EKS
- Padrão de blueprints do Terraform EKS - Login único Okta para o cluster Amazon EKS
- auditoria 2rbac
- rbac.dev Uma lista de recursos adicionais, incluindo blogs e ferramentas, para o Kubernetes RBAC
- Herói de ação
- kube2iam
- kiam

Segurança do pod

A especificação do pod inclui uma variedade de atributos diferentes que podem fortalecer ou enfraquecer sua postura geral de segurança. Como praticante do Kubernetes, sua principal preocupação deve ser evitar que um processo executado em um contêiner escape dos limites de isolamento do tempo de execução do contêiner e tenha acesso ao host subjacente.

Capacidades do Linux

Os processos executados em um contêiner são executados no contexto do usuário raiz [Linux] por padrão. Embora as ações do root em um contêiner sejam parcialmente restringidas pelo conjunto de recursos do Linux que o tempo de execução do contêiner atribui aos contêineres, esses privilégios padrão podem permitir que um invasor aumente seus privilégios .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 e os pods Fargate recebem os recursos mencionados acima por padrão. Além disso, os recursos do Linux só podem ser removidos dos pods do Fargate.

Os pods que são executados com privilégios herdam todos os recursos do Linux associados ao root no host. Isso deve ser evitado, se possível.

Autorização do Node

Todos os nós de trabalho do Kubernetes usam um modo de autorização chamado Node Authorization. A autorização de nós autoriza todas as solicitações de API originadas do kubelet e permite que os nós executem as seguintes ações:

Operações de leitura:

- serviços
- endpoints
- nós
- pods
- segredos, configmaps, declarações de volume persistentes e volumes persistentes relacionados a pods vinculados ao nó do kubelet

Operações de gravação:

- nós e status do nó (habilite o plug-in de `NodeRestriction` admissão para limitar um kubelet a modificar seu próprio nó)
- pods e status do pod (habilite o plug-in de `NodeRestriction` admissão para limitar um kubelet a modificar pods vinculados a si mesmo)
- eventos

Operações relacionadas à autenticação:

- Acesso de leitura/gravação à API CertificateSigningRequest (CSR) para inicialização de TLS
- a capacidade de criar TokenReview e SubjectAccessReview para cheques delegados authentication/authorization

O EKS usa o controlador de admissão de restrição de nós, que só permite que o nó modifique um conjunto limitado de atributos do nó e objetos do pod que estão vinculados ao nó. No entanto, um invasor que conseguir acessar o host ainda poderá obter informações confidenciais sobre o ambiente a partir da API do Kubernetes, o que poderia permitir que ele se movesse lateralmente dentro do cluster.

Soluções de segurança para Pod

Política de segurança do Pod (PSP)

No passado, os recursos da Política de Segurança do Pod (PSP) eram usados para especificar um conjunto de requisitos que os pods precisavam atender antes de serem criados. A partir da versão 1.21 do Kubernetes, o PSP foi descontinuado. Eles estão programados para remoção na versão 1.25 do Kubernetes.

> **Important**
>
> PSPs estão obsoletos na versão 1.21 do Kubernetes. Você terá até a versão 1.25 ou aproximadamente 2 anos para fazer a transição para uma alternativa. Este documento explica a motivação para essa suspensão de uso.

Migração para uma nova solução de segurança de pod

Como PSPs foram removidos a partir do Kubernetes v1.25, os administradores e operadores de cluster devem substituir esses controles de segurança. Duas soluções podem preencher essa necessidade:

- Policy-as-code Soluções (PAC) do ecossistema Kubernetes
- Padrões de segurança (PSS) do Kubernetes Pod

As soluções PAC e PSS podem coexistir com o PSP; elas podem ser usadas em clusters antes que o PSP seja removido. Isso facilita a adoção ao migrar do PSP. Consulte este documento ao considerar a migração do PSP para o PSS.

A Kyverno, uma das soluções de PAC descritas abaixo, tem orientações específicas descritas em uma postagem no blog ao migrar PSPs para sua solução, incluindo políticas análogas, comparações de recursos e um procedimento de migração. Informações e orientações adicionais sobre a migração para Kyverno com relação ao Pod Security Admission (PSA) foram publicadas no blog da AWS aqui.

Policy-as-code (PAC)

Policy-as-code As soluções (PAC) fornecem proteções para orientar os usuários do cluster e evitar comportamentos indesejados por meio de controles prescritos e automatizados. O PAC usa Kubernetes Dynamic Admission Controllers para interceptar o fluxo de solicitações do servidor da API Kubernetes, por meio de uma chamada de webhook, e alterar e validar as cargas de solicitação, com base em políticas escritas e armazenadas como código. A mutação e a validação acontecem antes que a solicitação do servidor da API resulte em uma alteração no cluster. As soluções PAC usam políticas para combinar e agir sobre as cargas de solicitação do servidor de API, com base na taxonomia e nos valores.

Há várias soluções PAC de código aberto disponíveis para o Kubernetes. Essas soluções não fazem parte do projeto Kubernetes; elas são provenientes do ecossistema Kubernetes. Algumas soluções de PAC estão listadas abaixo.

- OPA/Gatekeeper
- Agente de políticas abertas (OPA)
- Kyverno
- Kubewarden
- Política JS

Para obter mais informações sobre as soluções PAC e como ajudá-lo a selecionar a solução adequada às suas necessidades, consulte os links abaixo.

- Contramedidas baseadas em políticas para Kubernetes — Parte 1
- Contramedidas baseadas em políticas para Kubernetes — Parte 2

Padrões de segurança do pod (PSS) e admissão de segurança do pod (PSA)

Em resposta à descontinuação do PSP e à necessidade contínua de controlar a segurança do pod out-of-the-box, com uma solução Kubernetes integrada, o Kubernetes Auth Special Interest Group criou o Pod Security Standards (PSS) e o Pod Security Admission (PSA). O esforço do PSA inclui um projeto de webhook do controlador de admissão que implementa os controles definidos no PSS. Essa abordagem de controlador de admissão é semelhante à usada nas soluções PAC.

De acordo com a documentação do Kubernetes, o PSS "`define três políticas diferentes para cobrir amplamente o espectro de segurança. Essas políticas são cumulativas e variam de altamente permissivas a altamente restritivas. `"

Essas políticas são definidas como:

- Privilegiado: política irrestrita (não segura), fornecendo o maior nível possível de permissões. Essa política permite escalonamentos de privilégios conhecidos. É a ausência de uma política. Isso é bom para aplicativos como agentes de registro CNIs, drivers de armazenamento e outros aplicativos de todo o sistema que precisam de acesso privilegiado.
- Linha de base: política minimamente restritiva que evita escalonamentos de privilégios conhecidos. Permite a configuração padrão (minimamente especificada) do pod. A política básica proíbe o uso de HostNetwork, HostPid, HostPC, HostPath, HostPort, a incapacidade de adicionar recursos Linux, além de várias outras restrições.
- Restrito: política altamente restrita, seguindo as melhores práticas atuais de fortalecimento de Pod. Essa política é herdada da linha de base e adiciona mais restrições, como a incapacidade de ser executada como raiz ou como grupo-raiz. Políticas restritas podem afetar a capacidade de funcionamento de um aplicativo. Eles são voltados principalmente para a execução de aplicativos essenciais de segurança.

Essas políticas definem perfis para execução de pods, organizados em três níveis de acesso privilegiado versus acesso restrito.

Para implementar os controles definidos pelo PSS, o PSA opera em três modos:

- impor: violações da política farão com que o pod seja rejeitado.
- auditoria: as violações da política acionarão a adição de uma anotação de auditoria ao evento registrado no registro de auditoria, mas são permitidas de outra forma.
- aviso: as violações da política acionarão um aviso voltado para o usuário, mas, de outra forma, são permitidas.

Esses modos e os níveis de perfil (restrição) são configurados no nível do Kubernetes Namespace, usando rótulos, conforme mostrado no exemplo abaixo.

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

Quando usados de forma independente, esses modos operacionais têm respostas diferentes que resultam em experiências de usuário diferentes. O modo de imposição impedirá a criação de pods se os respectivos PodSpecs violarem o nível de restrição configurado. No entanto, nesse modo, objetos Kubernetes que não sejam de pod que criam pods, como Deployments, não serão impedidos de serem aplicados ao cluster, mesmo que o PodSpec nele viole o PSS aplicado. Nesse caso, a implantação será aplicada, enquanto os pods serão impedidos de serem aplicados.

Essa é uma experiência difícil para o usuário, pois não há indicação imediata de que o objeto de implantação aplicado com sucesso oculte a falha na criação do pod. Os PodSpecs ofensivos não criarão pods. A inspeção do recurso de implantação com `kubectl get deploy <DEPLOYMENT_NAME> -oyaml` exibirá a mensagem do (s) `.status.conditions` elemento (s) do pod (s) com falha, conforme mostrado abaixo.

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

Nos modos de auditoria e aviso, as restrições de pods não impedem que pods violadores sejam criados e iniciados. No entanto, nesses modos, as anotações de auditoria nos eventos do log de auditoria do servidor de API e os avisos para clientes do servidor de API, como kubectl, são acionados, respectivamente, quando pods, assim como objetos que criam pods, contêm podSpecs com violações. Uma mensagem de `kubectl` aviso é exibida abaixo.

```
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
```

Os modos de auditoria e aviso do PSA são úteis ao introduzir o PSS sem afetar negativamente as operações do cluster.

Os modos operacionais do PSA não são mutuamente exclusivos e podem ser usados de forma cumulativa. Conforme mostrado abaixo, os vários modos podem ser configurados em um único namespace.

```
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
```

No exemplo acima, os avisos fáceis de usar e as anotações de auditoria são fornecidos ao aplicar Deployments, enquanto a aplicação de violações também é fornecida no nível do pod. Na verdade, vários rótulos de PSA podem usar diferentes níveis de perfil, conforme mostrado abaixo.

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

No exemplo acima, o PSA é configurado para permitir a criação de todos os pods que satisfaçam o nível básico do perfil e, em seguida, avisar sobre pods (e objetos que criam pods) que violam o nível de perfil restrito. Essa é uma abordagem útil para determinar os possíveis impactos ao mudar da linha de base para perfis restritos.

Pods existentes

Se um namespace com pods existentes for modificado para usar um perfil PSS mais restritivo, os modos de auditoria e aviso produzirão mensagens apropriadas; no entanto, o modo de aplicação não excluirá os pods. As mensagens de aviso são vistas abaixo.

```
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
```

Isenções

O PSA usa isenções para excluir a aplicação de violações contra grupos que, de outra forma, teriam sido aplicadas. Essas isenções estão listadas abaixo.

- Nomes de usuário: solicitações de usuários com um nome de usuário autenticado (ou personificado) isento são ignoradas.
- RuntimeClassNames: pods e recursos de carga de trabalho que especificam um nome de classe de tempo de execução isento são ignorados.

- Namespaces: pods e recursos de carga de trabalho em um namespace isento são ignorados.

Essas isenções são aplicadas estaticamente na configuração do controlador de admissão do PSA como parte da configuração do servidor da API.

Na implementação do Validating Webhook, as isenções podem ser configuradas em um ConfigMaprecurso do Kubernetes que é montado como um volume no contêiner. 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"]
```

Conforme visto no ConfigMap YAML acima, o nível PSS padrão em todo o cluster foi definido como restrito para todos os modos de PSA, auditoria, aplicação e aviso. Isso afeta todos os namespaces, exceto aqueles isentos:. `namespaces: ["kube-system","policy-test1"]` Além disso, no ValidatingWebhookConfigurationrecurso, visto abaixo, o pod-security-webhooknamespace também está isento do 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**
>
> O Pod Security Admissions passou para estável no Kubernetes v1.25. Se você quiser usar o recurso Pod Security Admission antes de ser ativado por padrão, precisará instalar o controlador de admissão dinâmico (webhook mutante). As instruções para instalar e configurar o webhook podem ser encontradas aqui.

Escolhendo entre policy-as-code os padrões de segurança do Pod

Os Pod Security Standards (PSS) foram desenvolvidos para substituir a Política de Segurança do Pod (PSP), fornecendo uma solução integrada ao Kubernetes e que não exigia soluções do ecossistema Kubernetes. Dito isso, as soluções policy-as-code (PAC) são consideravelmente mais flexíveis.

A lista de prós e contras a seguir foi projetada para ajudá-lo a tomar uma decisão mais informada sobre sua solução de segurança de pod.

Policy-as-code (em comparação com os padrões de segurança do Pod)

Prós:

- Mais flexível e mais granular (até os atributos dos recursos, se necessário)
- Não é apenas focado em cápsulas, pode ser usado contra diferentes recursos e ações
- Não é aplicado apenas no nível do namespace
- Mais maduro do que os padrões de segurança do Pod

- As decisões podem ser baseadas em qualquer coisa na carga de solicitação do servidor da API, bem como nos recursos de cluster existentes e nos dados externos (dependendo da solução)
- Suporta solicitações mutantes do servidor de API antes da validação (depende da solução)
- Pode gerar políticas complementares e recursos do Kubernetes (depende da solução) - A partir das políticas de pod, o Kyverno pode gerar políticas automaticamente para controladores de nível superior, como implantações. O Kyverno também pode gerar recursos adicionais do Kubernetes "`quando um novo recurso é criado ou quando a fonte é atualizada`" usando Generate Rules.)
- Pode ser usado para mudar para a esquerda, para os pipelines do CICD, antes de fazer chamadas para o servidor da API Kubernetes (dependente da solução)
- Pode ser usado para implementar comportamentos que não estão necessariamente relacionados à segurança, como melhores práticas, padrões organizacionais etc.
- Pode ser usado em casos de uso que não sejam do Kubernetes (depende da solução)
- Devido à flexibilidade, a experiência do usuário pode ser ajustada às necessidades dos usuários

Contras:

- Não incorporado ao Kubernetes
- Mais complexo de aprender, configurar e oferecer suporte
- A criação de políticas pode exigir novas skills/languages/capabilities

Admissão de segurança do Pod (em comparação com policy-as-code)

Prós:

- Integrado ao Kubernetes
- Mais simples de configurar
- Sem novos idiomas para usar ou políticas para criar
- Se o nível de admissão padrão do cluster estiver configurado como privilegiado, os rótulos de namespace poderão ser usados para incluir namespaces nos perfis de segurança do pod.

Contras:

- Não é tão flexível ou granular quanto policy-as-code
- Apenas 3 níveis de restrições

- Focado principalmente em cápsulas

Resumo

Se você atualmente não tem uma solução de segurança de pod, além do PSP, e sua postura de segurança de pod necessária se encaixa no modelo definido nos Padrões de Segurança de Pod (PSS), um caminho mais fácil pode ser adotar o PSS, em vez de uma solução. policy-as-code No entanto, se a postura de segurança do seu pod não se encaixa no modelo PSS, ou se você pretende adicionar controles adicionais, além dos definidos pelo PSS, uma policy-as-code solução pareceria mais adequada.

Recomendações

Use vários modos de admissão de segurança do Pod (PSA) para uma melhor experiência do usuário

Conforme mencionado anteriormente, o modo PSA enforce impede que pods com violações de PSS sejam aplicados, mas não interrompe controladores de nível superior, como Deployments. Na verdade, a implantação será aplicada com sucesso sem qualquer indicação de que os pods não foram aplicados. Embora você possa usar o kubectl para inspecionar o objeto de implantação e descobrir a mensagem de pods com falha do PSA, a experiência do usuário poderia ser melhor. Para melhorar a experiência do usuário, vários modos de PSA (auditoria, aplicação, aviso) devem ser usados.

```
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
```

No exemplo acima, com o modo de imposição definido, quando um manifesto de implantação com violações de PSS no respectivo PodSpec tentar ser aplicado ao servidor da API Kubernetes, a implantação será aplicada com sucesso, mas os pods não. E, como os modos de auditoria e aviso também estão habilitados, o cliente do servidor da API receberá uma mensagem de aviso e o evento do log de auditoria do servidor da API também será anotado com uma mensagem.

Restrinja os contêineres que podem ser executados como privilegiados

Conforme mencionado, os contêineres que são executados com privilégios herdam todos os recursos do Linux atribuídos ao root no host. Raramente os contêineres precisam desses tipos de privilégios para funcionar adequadamente. Há vários métodos que podem ser usados para restringir as permissões e os recursos dos contêineres.

> **Important**
>
> O Fargate é um tipo de lançamento que permite que você execute contêineres "sem servidor", onde os contêineres de um pod são executados na infraestrutura gerenciada pela AWS. Com o Fargate, você não pode executar um contêiner privilegiado nem configurar seu pod para usar HostNetwork ou HostPort.

Não execute processos em contêineres como root

Todos os contêineres são executados como root por padrão. Isso pode ser problemático se um invasor conseguir explorar uma vulnerabilidade no aplicativo e obter acesso ao shell ao contêiner em execução. Você pode mitigar esse risco de várias maneiras. Primeiro, removendo a casca da imagem do contêiner. Segundo, adicionar a diretiva USER ao seu Dockerfile ou executar os contêineres no pod como um usuário não root. O Kubernetes PodSpec inclui um conjunto de campos, em`spec.securityContext`, que permite especificar o and/or grupo de usuários sob o qual executar seu aplicativo. Esses campos são `runAsUser` e `runAsGroup` respectivamente.

Para impor o uso do e de seus elementos associados no `spec.securityContext` Kubernetes, o PodSpec policy-as-code ou os Padrões de Segurança do Pod podem ser adicionados aos clusters. Essas soluções permitem que você escreva e/ou use políticas ou perfis que possam validar cargas de solicitação de entrada do servidor da API Kubernetes, antes que elas persistam no etcd. Além disso, policy-as-code as soluções podem alterar as solicitações recebidas e, em alguns casos, gerar novas solicitações.

Nunca execute o Docker no Docker nem monte o soquete no contêiner

Embora isso permita que você crie build/run imagens em contêineres do Docker, você está basicamente cedendo o controle total do nó ao processo em execução no contêiner. Se você precisar criar imagens de contêiner no Kubernetes, use Kaniko, buildah ou um serviço de compilação, como alternativa. CodeBuild

> **Note**
>
> Os clusters Kubernetes usados para processamento de CICD, como a criação de imagens de contêineres, devem ser isolados dos clusters que executam cargas de trabalho mais generalizadas.

Restrinja o uso do HostPath ou, se o HostPath for necessário, restrinja quais prefixos podem ser usados e configure o volume como somente para leitura

`hostPath`é um volume que monta um diretório do host diretamente no contêiner. Raramente os pods precisarão desse tipo de acesso, mas se precisarem, você precisa estar ciente dos riscos. Por padrão, os pods executados como root terão acesso de gravação ao sistema de arquivos exposto pelo HostPath. Isso pode permitir que um invasor modifique as configurações do kubelet, crie links simbólicos para diretórios ou arquivos não expostos diretamente pelo HostPath, por exemplo, /etc/shadow, instale chaves ssh, leia segredos montados no host e outras coisas maliciosas. Para mitigar os riscos do HostPath, configure o `spec.containers.volumeMounts` com`readOnly`, por exemplo:

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

Você também deve usar policy-as-code soluções para restringir os diretórios que podem ser usados por `hostPath` volumes ou impedir totalmente o `hostPath` uso. Você pode usar a linha de base dos padrões de segurança do Pod ou as políticas restritas para evitar o uso de`hostPath`.

Para obter mais informações sobre os perigos da escalação privilegiada, leia o blog Bad Pods: Kubernetes Pod Privilege Escalation de Seth Art.

Defina solicitações e limites para cada contêiner para evitar contenção de recursos e ataques de DoS

Um pod sem solicitações ou limites pode, teoricamente, consumir todos os recursos disponíveis em um host. À medida que pods adicionais são programados em um nó, o nó pode sofrer pressão de CPU ou memória, o que pode fazer com que o Kubelet encerre ou expulse pods do nó. Embora você não possa evitar que isso aconteça completamente, definir solicitações e limites ajudará a minimizar

a contenção de recursos e mitigar o risco de aplicativos mal escritos que consomem uma quantidade excessiva de recursos.

O `podSpec` permite que você especifique solicitações e limites para CPU e memória. A CPU é considerada um recurso compressível porque pode ter excesso de assinaturas. A memória é incompressível, ou seja, não pode ser compartilhada entre vários contêineres.

Ao especificar solicitações de CPU ou memória, você está basicamente designando a quantidade de memória que os contêineres têm a garantia de receber. O Kubernetes agrega as solicitações de todos os contêineres em um pod para determinar em qual nó programar o pod. Se um contêiner exceder a quantidade de memória solicitada, ele poderá estar sujeito ao encerramento se houver pressão de memória no nó.

Os limites são a quantidade máxima de recursos de CPU e memória que um contêiner pode consumir e correspondem diretamente ao `memory.limit_in_bytes` valor do cgroup criado para o contêiner. Um contêiner que exceda o limite de memória será eliminado pelo OOM. Se um contêiner exceder seu limite de CPU, ele será acelerado.

> **Note**
>
> Ao usar o contêiner`resources.limits`, é altamente recomendável que o uso dos recursos do contêiner (também conhecido como Resource Footprints) seja orientado por dados e preciso, com base em testes de carga. Sem uma pegada de recursos precisa e confiável, o contêiner `resources.limits` pode ser acolchoado. Por exemplo, `resources.limits.memory` pode ser preenchido de 20 a 30% a mais do que os máximos observáveis, para considerar possíveis imprecisões no limite de recursos de memória.

O Kubernetes usa três classes de Qualidade de Serviço (QoS) para priorizar as cargas de trabalho em execução em um nó. Isso inclui:

- garantidos
- estourável
- melhor esforço

Se os limites e as solicitações não forem definidos, o pod será configurado como melhor esforço (prioridade mais baixa). As cápsulas de melhor esforço são as primeiras a serem mortas quando não há memória suficiente. Se os limites forem definidos em todos os contêineres dentro do pod, ou se

as solicitações e os limites forem definidos com os mesmos valores e não forem iguais a 0, o pod será configurado como garantido (prioridade mais alta). Os pods garantidos não serão eliminados a menos que excedam os limites de memória configurados. Se os limites e as solicitações estiverem configurados com valores diferentes e não forem iguais a 0, ou se um contêiner dentro do pod definir limites e os outros não tiverem ou tiverem limites definidos para recursos diferentes, os pods serão configurados como intermitentes (prioridade média). Esses pods têm algumas garantias de recursos, mas podem ser eliminados quando excedem a memória solicitada.

> **Important**
>
> As solicitações não afetam o `memory_limit_in_bytes` valor do cgroup do contêiner; o limite do cgroup é definido para a quantidade de memória disponível no host. No entanto, definir o valor da solicitação muito baixo pode fazer com que o pod seja direcionado para o encerramento pelo kubelet se o nó sofrer pressão de memória.

Classe	Prioridade	Condição	Condição de morte
Garantido	highest	limite = solicitação! = 0	Exceda apenas os limites de memória
Performance expansível	medium	limite! = solicitação! = 0	Pode ser eliminado se exceder a memória solicitada
Melhor esforço	lowest	limite e solicitação não definidos	O primeiro a ser morto quando não há memória suficiente

Para obter informações adicionais sobre QoS de recursos, consulte a documentação do Kubernetes.

Você pode forçar o uso de solicitações e limites definindo uma cota de recursos em um namespace ou criando um intervalo limite. Uma cota de recursos permite que você especifique a quantidade total de recursos, por exemplo, CPU e RAM, alocados para um namespace. Quando aplicado a um namespace, ele força você a especificar solicitações e limites para todos os contêineres implantados nesse namespace. Por outro lado, os intervalos de limite oferecem um controle mais granular da alocação de recursos. Com intervalos de limite, você pode usar min/max recursos de CPU e

memória por pod ou por contêiner em um namespace. Você também pode usá-los para definir valores de solicitação/limite padrão, caso nenhum seja fornecido.

Policy-as-code as soluções podem ser usadas para impor solicitações e limites. ou até mesmo para criar cotas de recursos e intervalos de limite quando os namespaces são criados.

Não permita escalonamento privilegiado

O escalonamento privilegiado permite que um processo altere o contexto de segurança sob o qual está sendo executado. O Sudo é um bom exemplo disso, assim como os binários com o bit SUID ou SGID. O escalonamento privilegiado é basicamente uma forma de os usuários executarem um arquivo com as permissões de outro usuário ou grupo. Você pode impedir que um contêiner use o escalonamento privilegiado policy-as-code implementando uma política mutante definida `allowPrivilegeEscalation` como `false` ou configurando o. `securityContext.allowPrivilegeEscalation` `podSpec` Policy-as-code as políticas também podem ser usadas para impedir que as solicitações do servidor de API sejam bem-sucedidas se configurações incorretas forem detectadas. Os padrões de segurança de pods também podem ser usados para impedir que os pods usem o escalonamento de privilégios.

Desativar montagens de ServiceAccount tokens

Para pods que não precisam acessar a API Kubernetes, você pode desativar a montagem automática de um ServiceAccount token em uma especificação de pod ou para todos os pods que usam um determinado token. 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
```

Desativar descoberta de serviços

Para pods que não precisam pesquisar ou chamar serviços no cluster, você pode reduzir a quantidade de informações fornecidas a um pod. Você pode definir a política de DNS do pod para não usar CoreDNS e não expor serviços no namespace do pod como variáveis de ambiente. Consulte a documentação do Kubernetes sobre variáveis de ambiente para obter mais informações sobre links de serviços. O valor padrão da política de DNS de um pod é "ClusterFirst", que usa DNS no cluster, enquanto o valor não padrão “Default” usa a resolução DNS do nó subjacente. Consulte a documentação do Kubernetes sobre a política de DNS do Pod para obter mais informações.

Example

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

Configure suas imagens com sistema de arquivos raiz somente para leitura

Configurar suas imagens com um sistema de arquivos raiz somente para leitura impede que um invasor sobrescreva um binário no sistema de arquivos que seu aplicativo usa. Se seu aplicativo precisar gravar no sistema de arquivos, considere gravar em um diretório temporário ou anexar e montar um volume. Você pode aplicar isso configurando os pods da SecurityContext seguinte forma:

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

Policy-as-code e os Padrões de Segurança do Pod podem ser usados para impor esse comportamento.

Example

De acordo com o Windows, os contêineres no Kubernetes `securityContext.readOnlyRootFilesystem` não podem ser configurados `true` para um

contêiner em execução no Windows, pois o acesso de gravação é necessário para que os processos de registro e do sistema sejam executados dentro do contêiner.

Ferramentas e recursos

- Workshop de imersão em segurança do Amazon EKS — Pod Security
- open-policy-agent/gatekeeper-library: A biblioteca de políticas do OPA Gatekeeper, uma biblioteca de OPA/Gatekeeper políticas que você pode usar como substituta. PSPs
- Biblioteca de políticas Kyverno
- Uma coleção de políticas comuns de OPA e Kyverno para EKS.
- Contramedidas baseadas em políticas: parte 1
- Contramedidas baseadas em políticas: parte 2
- Pod Security Policy Migrator, uma ferramenta que se converte PSPs em políticas OPA/Gatekeeper ou Kyverno KubeWarden
- NeuVector da SUSE, a plataforma de segurança de contêineres de código aberto e de confiança zero fornece políticas de processo e sistema de arquivos, bem como regras de controle de admissão.

Isolamento de inquilinos

Quando pensamos em multilocação, geralmente queremos isolar um usuário ou aplicativo de outros usuários ou aplicativos executados em uma infraestrutura compartilhada.

O Kubernetes é um único orquestrador de inquilinos, ou seja, uma única instância do plano de controle é compartilhada entre todos os inquilinos em um cluster. No entanto, existem vários objetos do Kubernetes que você pode usar para criar a aparência de multilocação. Por exemplo, namespaces e controles de acesso baseados em funções (RBAC) podem ser implementados para isolar logicamente os inquilinos uns dos outros. Da mesma forma, as cotas e os intervalos de limite podem ser usados para controlar a quantidade de recursos do cluster que cada inquilino pode consumir. No entanto, o cluster é a única construção que fornece um forte limite de segurança. Isso ocorre porque um invasor que consegue obter acesso a um host dentro do cluster pode recuperar todos os segredos e volumes montados nesse host. ConfigMaps Eles também poderiam se passar pelo Kubelet, o que lhes permitiria manipular os atributos do nó and/or se movendo lateralmente dentro do cluster.

As seções a seguir explicarão como implementar o isolamento de inquilinos e, ao mesmo tempo, reduzir os riscos de usar um único orquestrador de inquilinos, como o Kubernetes.

Multilocação suave

Com a multilocação flexível, você usa construções nativas do Kubernetes, por exemplo, namespaces, funções e associações de funções e políticas de rede, para criar uma separação lógica entre os locatários. O RBAC, por exemplo, pode impedir que os inquilinos acessem ou manipulem os recursos uns dos outros. As cotas e os intervalos de limite controlam a quantidade de recursos do cluster que cada locatário pode consumir, enquanto as políticas de rede podem ajudar a impedir que aplicativos implantados em namespaces diferentes se comuniquem entre si.

Nenhum desses controles, no entanto, impede que pods de diferentes locatários compartilhem um nó. Se for necessário um isolamento mais forte, você pode usar um seletor de nós, regras de antiafinidade, and/or manchas e tolerâncias para forçar pods de diferentes inquilinos a serem programados em nós separados; geralmente chamados de nós de locatários únicos. Isso pode ser bastante complicado e ter um custo proibitivo em um ambiente com muitos inquilinos.

> **Important**
>
> A multilocação flexível implementada com namespaces não permite que você forneça aos locatários uma lista filtrada de namespaces porque os namespaces são um tipo com escopo global. Se um locatário tiver a capacidade de visualizar um determinado namespace, ele poderá visualizar todos os namespaces dentro do cluster.

> **Warning**
>
> Com soft-multi-tenancy, os locatários mantêm a capacidade de consultar o CoreDNS para todos os serviços executados no cluster por padrão. Um invasor pode explorar isso executando o dig SRV a `..svc.cluster.local` partir de qualquer pod no cluster. Se você precisar restringir o acesso aos registros DNS dos serviços executados em seus clusters, considere usar os plug-ins Firewall ou Policy para CoreDNS. Para obter informações adicionais, consulte https://github.com/coredns/ policy# -policykubernetes-metadata-multi-tenancy.

O Kiosk é um projeto de código aberto que pode auxiliar na implementação da multilocação flexível. Ele é implementado como uma série CRDs de controladores que fornecem os seguintes recursos:

- Contas e usuários de contas para separar inquilinos em um cluster Kubernetes compartilhado
- Provisionamento de namespace de autoatendimento para usuários da conta
- Limites de conta para garantir a qualidade do serviço e a imparcialidade ao compartilhar um cluster
- Modelos de namespace para isolamento seguro de inquilinos e inicialização de namespace de autoatendimento

O Loft é uma oferta comercial dos mantenedores do Kiosk e DevSpaceque adiciona os seguintes recursos:

- Acesso a vários clusters para conceder acesso a espaços em diferentes clusters
- O modo de suspensão reduz as implantações em um espaço durante períodos de inatividade
- Login único com provedores de autenticação OIDC, como GitHub

Há três casos de uso principais que podem ser resolvidos por meio de multilocação flexível.

Configuração empresarial

A primeira é em um ambiente corporativo em que os “inquilinos” são semiconfiáveis, pois são funcionários, contratados ou estão autorizados pela organização. Cada inquilino normalmente se alinha a uma divisão administrativa, como um departamento ou equipe.

Nesse tipo de configuração, um administrador de cluster geralmente será responsável pela criação de namespaces e pelo gerenciamento de políticas. Eles também podem implementar um modelo de administração delegada em que determinados indivíduos supervisionam um namespace, permitindo que eles executem operações CRUD para objetos não relacionados a políticas, como implantações, serviços, pods, trabalhos etc.

O isolamento fornecido pelo tempo de execução de um contêiner pode ser aceitável dentro dessa configuração ou pode precisar ser aumentado com controles adicionais para a segurança do pod. Também pode ser necessário restringir a comunicação entre serviços em namespaces diferentes se for necessário um isolamento mais rigoroso.

Kubernetes como serviço

Por outro lado, a multilocação flexível pode ser usada em configurações nas quais você deseja oferecer o Kubernetes como um serviço (KaaS). Com o KaaS, seu aplicativo é hospedado em um cluster compartilhado junto com uma coleção de controladores e CRDs que fornecem um conjunto de serviços PaaS. Os locatários interagem diretamente com o servidor da API Kubernetes e têm permissão para realizar operações CRUD em objetos que não sejam de política. Há também um elemento de autoatendimento em que os inquilinos podem criar e gerenciar seus próprios namespaces. Nesse tipo de ambiente, presume-se que os inquilinos estejam executando código não confiável.

Para isolar os inquilinos nesse tipo de ambiente, você provavelmente precisará implementar políticas de rede rígidas, bem como o sandboxing de pods. O sandboxing é onde você executa os contêineres de um pod dentro de uma micro VM como o Firecracker ou em um kernel de espaço de usuário. Hoje, você pode criar pods em sandbox com o EKS Fargate.

Software como serviço (SaaS)

O caso de uso final da multilocação flexível está em uma configuração (Software-as-a-ServiceSaaS). Nesse ambiente, cada locatário está associado a uma instância específica de um aplicativo em execução no cluster. Cada instância geralmente tem seus próprios dados e usa controles de acesso separados que geralmente são independentes do Kubernetes RBAC.

Diferentemente dos outros casos de uso, o locatário em uma configuração SaaS não interage diretamente com a API Kubernetes. Em vez disso, o aplicativo SaaS é responsável pela interface com a API Kubernetes para criar os objetos necessários para dar suporte a cada locatário.

Construções do Kubernetes

Em cada um desses casos, as seguintes construções são usadas para isolar os inquilinos uns dos outros:

Namespaces

Os namespaces são fundamentais para implementar a multilocação flexível. Eles permitem que você divida o cluster em partições lógicas. Cotas, políticas de rede, contas de serviço e outros objetos necessários para implementar a multilocação têm como escopo um namespace.

Políticas de rede

Por padrão, todos os pods em um cluster Kubernetes podem se comunicar entre si. Esse comportamento pode ser alterado usando políticas de rede.

As políticas de rede restringem a comunicação entre pods usando rótulos ou intervalos de endereços IP. Em um ambiente multilocatário em que o isolamento estrito da rede entre os locatários é necessário, recomendamos começar com uma regra padrão que negue a comunicação entre os pods e outra regra que permita que todos os pods consultem o servidor DNS para resolução de nomes. Com isso estabelecido, você pode começar a adicionar mais regras permissivas que permitem a comunicação dentro de um namespace. Isso pode ser refinado ainda mais conforme necessário.

> **Note**
>
> O Amazon VPC CNI agora oferece suporte às políticas de rede do Kubernetes para criar políticas que possam isolar cargas de trabalho confidenciais e protegê-las do acesso não autorizado ao executar o Kubernetes na AWS. Isso significa que você pode usar todos os recursos da API de política de rede em seu cluster Amazon EKS. Esse nível de controle granular permite que você implemente o princípio do privilégio mínimo, o que garante que somente pods autorizados possam se comunicar entre si.

> **Important**
>
> As políticas de rede são necessárias, mas não suficientes. A aplicação de políticas de rede requer um mecanismo de políticas como Calico ou Cilium.

Regras de controle de acesso com base em função (RBAC)

Funções e associações de funções são os objetos do Kubernetes usados para aplicar o controle de acesso baseado em funções (RBAC) no Kubernetes. As funções contêm listas de ações que podem ser executadas em objetos em seu cluster. As vinculações de funções especificam os indivíduos ou grupos aos quais as funções se aplicam. Nas configurações corporativas e de KaaS, o RBAC pode ser usado para permitir a administração de objetos por grupos ou indivíduos selecionados.

Cotas

As cotas são usadas para definir limites nas cargas de trabalho hospedadas em seu cluster. Com as cotas, você pode especificar a quantidade máxima de CPU e memória que um pod pode consumir ou limitar o número de recursos que podem ser alocados em um cluster ou namespace. Os intervalos de limite permitem que você declare valores mínimos, máximos e padrão para cada limite.

O comprometimento excessivo de recursos em um cluster compartilhado geralmente é benéfico porque permite que você maximize seus recursos. No entanto, o acesso ilimitado a um cluster pode causar falta de recursos, o que pode levar à degradação do desempenho e à perda da disponibilidade do aplicativo. Se as solicitações de um pod forem definidas como muito baixas e a utilização real dos recursos exceder a capacidade do nó, o nó começará a sentir pressão na CPU ou na memória. Quando isso acontece, os pods podem ser reiniciados e and/or removidos do nó.

Para evitar que isso aconteça, você deve planejar impor cotas em namespaces em um ambiente multilocatário para forçar os inquilinos a especificar solicitações e limites ao programar seus pods no cluster. Isso também mitigará uma possível negação de serviço ao restringir a quantidade de recursos que um pod pode consumir.

Você também pode usar cotas para distribuir os recursos do cluster de acordo com os gastos do inquilino. Isso é particularmente útil no cenário de KaaS.

Prioridade e preempção do pod

A prioridade e a preempção de pods podem ser úteis quando você quer dar mais importância a um pod em relação a outros pods. Por exemplo, com o pod priority, você pode configurar os pods do cliente A para serem executados com uma prioridade mais alta do que o cliente B. Quando não houver capacidade disponível suficiente, o programador despejará os pods de menor prioridade do cliente B para acomodar os pods de maior prioridade do cliente A. Isso pode ser especialmente útil em um ambiente SaaS em que os clientes dispostos a pagar um prêmio recebem uma prioridade maior.

> **Important**
>
> A prioridade dos pods pode ter um efeito indesejado em outros pods com prioridade mais baixa. Por exemplo, embora os grupos de vítimas sejam encerrados normalmente, mas isso não PodDisruptionBudget seja garantido, o que pode interromper um aplicativo com prioridade mais baixa que depende de um quórum de pods, consulte Limitações de preempção.

Controles de mitigação

Sua principal preocupação como administrador de um ambiente multilocatário é impedir que um invasor tenha acesso ao host subjacente. Os seguintes controles devem ser considerados para mitigar esse risco:

Ambientes de execução em sandbox para contêineres

O sandboxing é uma técnica pela qual cada contêiner é executado em sua própria máquina virtual isolada. As tecnologias que realizam o sandboxing de pods incluem o Firecracker e o Firekube da Weave.

Para obter informações adicionais sobre o esforço de tornar o Firecracker um tempo de execução compatível com o EKS, consulte 1238496944684597248.html. https://threadreaderapp.com/thread/

Agente de política aberta (OPA) e porteiro

O Gatekeeper é um controlador de admissão do Kubernetes que aplica políticas criadas com o OPA. Com o OPA, você pode criar uma política que executa pods de inquilinos em instâncias separadas ou com maior prioridade do que outros locatários. Uma coleção de políticas comuns de OPA pode ser encontrada no GitHub repositório deste projeto.

Há também um plug-in OPA experimental para CoreDNS que permite usar o OPA nos registros retornados filter/control pelo CoreDNS.

Kyverno

O Kyverno é um mecanismo de políticas nativo do Kubernetes que pode validar, alterar e gerar configurações com políticas como recursos do Kubernetes. O Kyverno usa sobreposições no estilo Kustomize para validação, suporta JSON Patch e patch de mesclagem estratégica para mutação e pode clonar recursos em namespaces com base em acionadores flexíveis.

Você pode usar o Kyverno para isolar namespaces, reforçar a segurança do pod e outras práticas recomendadas e gerar configurações padrão, como políticas de rede. Vários exemplos estão incluídos no GitHub repositório deste projeto. Muitos outros estão incluídos na biblioteca de políticas no site da Kyverno.

Isolando cargas de trabalho de inquilinos em nós específicos

Restringir as cargas de trabalho do inquilino para execução em nós específicos pode ser usado para aumentar o isolamento no modelo flexível de multilocação. Com essa abordagem, as cargas de

trabalho específicas do inquilino são executadas somente em nós provisionados para os respectivos inquilinos. Para obter esse isolamento, as propriedades nativas do Kubernetes (afinidade de nós, manchas e tolerações) são usadas para direcionar nós específicos para o agendamento de pods e evitar que pods de outros locatários sejam programados nos nós específicos do inquilino.

Parte 1 - Afinidade de nós

A afinidade de nós do Kubernetes é usada para direcionar nós para agendamento, com base nos rótulos dos nós. Com as regras de afinidade de nós, os pods são atraídos por nós específicos que correspondem aos termos do seletor. Na especificação do pod abaixo, a afinidade do `requiredDuringSchedulingIgnoredDuringExecution` nó é aplicada ao respectivo pod. O resultado é que o pod terá como alvo nós rotulados com a seguinte chave/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
...
```

Com essa afinidade de nó, o rótulo é necessário durante o agendamento, mas não durante a execução. Se os rótulos dos nós subjacentes mudarem, os pods não serão removidos apenas devido à alteração do rótulo. No entanto, o agendamento futuro pode ser afetado.

> **Warning**
>
> O prefixo do rótulo de `node-restriction.kubernetes.io/` tem um significado especial no Kubernetes. NodeRestrictionque está habilitado para clusters EKS `kubelet` evita adding/removing/updating rótulos com esse prefixo. Os atacantes não podem usar o `kubelet's credentials to update the node object or modify the system setup to pass these labels into `kubelet` anúncio não `kubelet` é permitido para modificar esses rótulos. Se esse prefixo for usado para todo o agendamento de pod para nó, ele

evitará cenários em que um invasor queira atrair um conjunto diferente de cargas de trabalho para um nó modificando os rótulos dos nós.

Example

Em vez da afinidade de nós, poderíamos ter usado o seletor de nós. No entanto, a afinidade do nó é mais expressiva e permite que mais condições sejam consideradas durante o agendamento do pod. Para obter informações adicionais sobre as diferenças e as opções de agendamento mais avançadas, consulte esta postagem no blog da CNCF sobre o agendamento avançado de pod para node do Kubernetes.

Parte 2 - Contaminações e tolerâncias

Atrair grupos para nós é apenas a primeira parte dessa abordagem em três partes. Para que essa abordagem funcione, devemos impedir que os pods sejam programados para nós para os quais os pods não estão autorizados. Para repelir pods indesejados ou não autorizados, o Kubernetes usa manchas de nós. As manchas são usadas para impor condições nos nós que impedem que os pods sejam programados. A mancha abaixo usa um par de valores-chave de. `tenant: tenants-x`

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

Dado o nó acima`taint`, somente cápsulas que toleram a contaminação poderão ser programadas no nó. Para permitir que os pods autorizados sejam programados no nó, as respectivas especificações do pod devem incluir a `toleration` até a contaminação, conforme mostrado abaixo.

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

Os pods com os itens acima não `toleration` serão impedidos de serem programados no nó, pelo menos não por causa dessa contaminação específica. As impurezas também são usadas pelo Kubernetes para interromper temporariamente o agendamento de pods durante determinadas condições, como a pressão dos recursos do nó. Com afinidade de nós, manchas e tolerâncias, podemos atrair efetivamente os frutos desejados para nós específicos e repelir os frutos indesejados.

> **Important**
>
> É necessário que determinados pods do Kubernetes sejam executados em todos os nós. Exemplos desses pods são aqueles iniciados pela Container Network Interface (CNI) e pelos daemonsets kube-proxy. Para esse fim, as especificações desses frutos contêm tolerâncias muito permissivas, para tolerar diferentes contaminações. Deve-se tomar cuidado para não alterar essas tolerações. A alteração dessas tolerâncias pode resultar em uma operação incorreta do cluster. Além disso, ferramentas de gerenciamento de políticas, como OPA/Gatekeeper e Kyverno, podem ser usadas para criar políticas de validação que impeçam que pods não autorizados usem essas tolerâncias permissivas.

Parte 3 - Gerenciamento baseado em políticas para seleção de nós

Há várias ferramentas que podem ser usadas para ajudar a gerenciar a afinidade dos nós e as tolerâncias das especificações dos pods, incluindo a aplicação de regras nos pipelines do CICD. No entanto, a aplicação do isolamento também deve ser feita no nível do cluster Kubernetes. Para esse fim, as ferramentas de gerenciamento de políticas podem ser usadas para alterar as solicitações de entrada do servidor da API Kubernetes, com base nas cargas de solicitação, para aplicar as respectivas regras e tolerâncias de afinidade de nós mencionadas acima.

Por exemplo, pods destinados ao namespace tenants-x podem ser marcados com a afinidade e a tolerância corretas do nó para permitir o agendamento nos nós tenants-x. Utilizando ferramentas de gerenciamento de políticas configuradas usando o Webhook de admissão mutante do Kubernetes, as políticas podem ser usadas para alterar as especificações do pod de entrada. As mutações adicionam os elementos necessários para permitir o agendamento desejado. Um exemplo OPA/Gatekeeper de política que adiciona uma afinidade de nó é visto abaixo.

```
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"
```

A política acima é aplicada a uma solicitação do servidor da API Kubernetes para aplicar um pod ao namespace tenants-x. A política adiciona a regra de afinidade do `requiredDuringSchedulingIgnoredDuringExecution` nó, para que os pods sejam atraídos pelos nós com o rótulo. `tenant: tenants-x`

Uma segunda política, vista abaixo, adiciona a tolerância à mesma especificação de pod, usando os mesmos critérios de correspondência de namespace e grupos, tipos e versões 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"
```

As políticas acima são específicas para pods; isso se deve aos caminhos para os elementos mutantes nos elementos das políticas`location`. Políticas adicionais podem ser criadas para lidar com recursos que criam pods, como recursos de Deployment e Job. As políticas listadas e outros exemplos podem ser vistos no GitHubprojeto complementar deste guia.

O resultado dessas duas mutações é que os frutos são atraídos para o nódulo desejado e, ao mesmo tempo, não são repelidos pela mancha específica do nódulo. Para verificar isso, podemos ver os trechos da saída de duas `kubectl` chamadas para obter os nós rotulados e colocar os `tenant=tenants-x` pods no namespace. `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 nas saídas acima, todos os pods estão programados nos nós rotulados com. `tenant=tenants-x` Simplificando, os pods funcionarão apenas nos nós desejados, e os outros

pods (sem a afinidade e as tolerâncias exigidas) não. As cargas de trabalho dos inquilinos são efetivamente isoladas.

Um exemplo de especificação de pod com mutação é visto abaixo.

```
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**
>
> As ferramentas de gerenciamento de políticas integradas ao fluxo de solicitações do servidor da API Kubernetes, usando webhooks de admissão de mutação e validação, são projetadas para responder à solicitação do servidor da API dentro de um prazo especificado. Isso geralmente é de 3 segundos ou menos. Se a chamada do webhook não retornar uma resposta dentro do tempo configurado, a and/or validação de mutação da solicitação de entrada do servidor da API poderá ou não ocorrer. Esse comportamento se baseia no fato de as configurações do webhook de admissão estarem definidas como Falha na Abertura ou Falha no Fechamento.

Nos exemplos acima, usamos políticas escritas para OPA/Gatekeeper. No entanto, existem outras ferramentas de gerenciamento de políticas que também lidam com nosso caso de uso de seleção de nós. Por exemplo, essa política do Kyverno pode ser usada para lidar com a mutação de afinidade do nó.

> **Note**
>
> Se estiverem operando corretamente, as políticas mutantes afetarão as alterações desejadas nas cargas de solicitação do servidor de API de entrada. No entanto, políticas de validação também devem ser incluídas para verificar se as alterações desejadas ocorrem, antes que as alterações possam persistir. Isso é especialmente importante ao usar essas políticas para tenant-to-node isolamento. Também é uma boa ideia incluir políticas de auditoria para verificar rotineiramente seu cluster em busca de configurações indesejadas.

Referências

- k-rail Projetado para ajudar você a proteger um ambiente multilocatário por meio da aplicação de determinadas políticas.
- Práticas de segurança para aplicativos MultiTenant SaaS usando o Amazon EKS

Multilocação rígida

A multilocação rígida pode ser implementada provisionando clusters separados para cada inquilino. Embora isso forneça um isolamento muito forte entre os inquilinos, tem várias desvantagens.

Primeiro, quando você tem muitos inquilinos, essa abordagem pode rapidamente se tornar cara. Além de pagar pelos custos do plano de controle de cada cluster, você não poderá compartilhar recursos computacionais entre clusters. Isso acabará por causar fragmentação, na qual um subconjunto de seus clusters é subutilizado, enquanto outros são superutilizados.

Em segundo lugar, você provavelmente precisará comprar ou criar ferramentas especiais para gerenciar todos esses clusters. Com o tempo, gerenciar centenas ou milhares de clusters pode simplesmente se tornar muito complicado.

Por fim, a criação de um cluster por inquilino será lenta em relação à criação de um namespace. No entanto, uma abordagem de locação rígida pode ser necessária em setores altamente regulamentados ou em ambientes SaaS onde é necessário um forte isolamento.

Direções futuras

A comunidade Kubernetes reconheceu as deficiências atuais da multilocação flexível e os desafios da multilocação rígida. O Multi-Tenancy Special Interest Group (SIG) está tentando resolver essas deficiências por meio de vários projetos de incubação, incluindo o Hierarchical Namespace Controller (HNC) e o Virtual Cluster.

A proposta do HNC (KEP) descreve uma maneira de criar relacionamentos pai-filho entre namespaces com herança de objetos [policy], além da capacidade de administradores de inquilinos criarem subnamespaces.

A proposta do Virtual Cluster descreve um mecanismo para criar instâncias separadas dos serviços do plano de controle, incluindo o servidor de API, o gerenciador do controlador e o agendador, para cada inquilino dentro do cluster (também conhecido como “Kubernetes on Kubernetes”).

A proposta de benchmarks de multilocação fornece diretrizes para compartilhar clusters usando namespaces para isolamento e segmentação, e uma ferramenta de linha de comando kubectl-mtb para validar a conformidade com as diretrizes.

Ferramentas e recursos de gerenciamento de vários clusters

- Nuvem Banzai
- Comandante
- Lente
- Nirmata
- Rafay
- Fazendeiro
- Fluxo de tecelagem

Auditoria e registro

Coletar e analisar registros [de auditoria] é útil por vários motivos diferentes. Os registros podem ajudar na análise e atribuição da causa raiz, ou seja, atribuir uma alteração a um usuário específico. Quando registros suficientes tiverem sido coletados, eles também podem ser usados para detectar comportamentos anômalos. No EKS, os registros de auditoria são enviados para o Amazon Cloudwatch Logs. A política de auditoria do EKS é a seguinte:

```
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"
```

Recomendações

Ativar registros de auditoria

Os registros de auditoria fazem parte dos registros do plano de controle do Kubernetes gerenciado pelo EKS, gerenciados pelo EKS. As instruções para enabling/disabling os registros do plano de controle, que incluem os registros do servidor da API Kubernetes, do gerenciador do controlador e do

agendador, junto com o registro de auditoria, podem ser encontradas aqui, -plane-logs.html# -export. https://docs.aws.amazon.com/eks/ latest/userguide/control enabling-control-plane-log

> **Note**
>
> Ao ativar o registro no plano de controle, você incorrerá em CloudWatchcustos para armazenar os registros. Isso levanta uma questão mais ampla sobre o custo contínuo da segurança. Em última análise, você terá que pesar esses custos com o custo de uma violação de segurança, por exemplo, perda financeira, danos à sua reputação, etc. Talvez você descubra que pode proteger adequadamente seu ambiente implementando somente algumas das recomendações deste guia.

> **Warning**
>
> O tamanho máximo de uma entrada de CloudWatch registros é de 1 MB, enquanto o tamanho máximo da solicitação da API Kubernetes é de 1,5 MiB. As entradas de registro maiores que 1 MB serão truncadas ou incluirão apenas os metadados da solicitação.

Utilize metadados de auditoria

Os registros de auditoria do Kubernetes incluem duas anotações que indicam se uma solicitação foi autorizada ou não `authorization.k8s.io/decision` e o motivo da decisão. `authorization.k8s.io/reason` Use esses atributos para verificar por que uma chamada de API específica foi permitida.

Crie alarmes para eventos suspeitos

Crie um alarme para alertá-lo automaticamente quando houver um aumento nas 403 respostas proibidas e 401 não autorizadas e, em seguida, use atributos como `hostsourceIPs`, e `k8s_user.username` para descobrir de onde vêm essas solicitações.

Analise registros com o Log Insights

Use o CloudWatch Log Insights para monitorar alterações nos objetos RBAC, por exemplo, Roles, RoleBindings ClusterRoles, e. ClusterRoleBindings Alguns exemplos de consultas aparecem abaixo:

Lista as atualizações do `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
```

Lista a criação de novos webhooks ou alterações nos webhooks de validação:

```
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
```

Lista as operações de criação, atualização e exclusão de funções:

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

Listas de operações de criação, atualização e exclusão para RoleBindings:

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

Listas de operações de criação, atualização e exclusão para ClusterRoles:

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

Listas de operações de criação, atualização e exclusão para ClusterRoleBindings:

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

Traça operações de leitura não autorizada contra Segredos:

```
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 solicitações anônimas que falharam:

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

Audite seus CloudTrail registros

A AWS APIs chamada por pods que utilizam funções do IAM para contas de serviço (IRSA) é automaticamente registrada CloudTrail junto com o nome da conta de serviço. Se o nome de uma conta de serviço que não foi explicitamente autorizada a chamar uma API aparecer no registro, isso pode ser uma indicação de que a política de confiança da função do IAM foi configurada incorretamente. De um modo geral, o Cloudtrail é uma ótima maneira de atribuir chamadas de API da AWS a diretores específicos do IAM.

Use o CloudTrail Insights para descobrir atividades suspeitas

CloudTrail O Insights analisa automaticamente eventos de gerenciamento de gravação a partir de CloudTrail trilhas e alerta você sobre atividades incomuns. Isso pode ajudar você a identificar quando há um aumento no volume de chamadas gravadas APIs em sua conta da AWS, inclusive de pods que usam o IRSA para assumir uma função do IAM. Consulte Anunciando CloudTrail insights: identificação e resposta a atividades incomuns de API para obter mais informações.

Recursos adicionais do

À medida que o volume de registros aumenta, analisá-los e filtrá-los com o Log Insights ou outra ferramenta de análise de registros pode se tornar ineficaz. Como alternativa, você pode considerar a execução do Sysdig Falco e do ekscloudwatch. A Falco analisa registros de auditoria e sinaliza anomalias ou abusos por um longo período de tempo. O projeto ekscloudwatch encaminha os eventos do log de auditoria para a Falco para análise. CloudWatch A Falco fornece um conjunto de regras de auditoria padrão, além da capacidade de adicionar suas próprias.

Outra opção pode ser armazenar os registros de auditoria no S3 e usar o algoritmo SageMaker Random Cut Forest para comportamentos anômalos que justifiquem uma investigação mais aprofundada.

Ferramentas e recursos

Os seguintes projetos comerciais e de código aberto podem ser usados para avaliar o alinhamento do seu cluster com as melhores práticas estabelecidas:

- Workshop de imersão em segurança do Amazon EKS — Detective Controls
- auditoria do kube
- kube-scan Atribui uma pontuação de risco às cargas de trabalho em execução em seu cluster de acordo com a estrutura do Kubernetes Common Configuration Scoring System
- kubesec.io
- polaris
- Estibordo
- Snyk
- Kubescape O Kubescape é uma ferramenta de segurança de kubernetes de código aberto que verifica clusters, arquivos YAML e gráficos do Helm. Ele detecta configurações incorretas de acordo com várias estruturas (incluindo NSA-CISA e MITRE ATT&CK®).

Segurança de rede

A segurança da rede tem várias facetas. O primeiro envolve a aplicação de regras que restringem o fluxo de tráfego de rede entre os serviços. A segunda envolve a criptografia do tráfego enquanto ele está em trânsito. Os mecanismos para implementar essas medidas de segurança no EKS são variados, mas geralmente incluem os seguintes itens:

Controle de tráfego

- Políticas de rede
- Grupos de segurança

Criptografia de rede

- Malha de serviços
- Interfaces de rede de contêineres (CNIs)
- Controladores de entrada e balanceadores de carga
- Instâncias Nitro
- CA privada do ACM com cert-manager

Política de rede

Em um cluster Kubernetes, toda comunicação de pod a pod é permitida por padrão. Embora essa flexibilidade possa ajudar a promover a experimentação, ela não é considerada segura. As políticas de rede do Kubernetes oferecem um mecanismo para restringir o tráfego de rede entre os pods (geralmente chamado de tráfego leste/oeste), bem como entre os pods e os serviços externos. As políticas de rede do Kubernetes operam nas camadas 3 e 4 do modelo OSI. As políticas de rede usam pod, seletores de namespace e rótulos para identificar pods de origem e destino, mas também podem incluir endereços IP, números de porta, protocolos ou uma combinação desses. As políticas de rede podem ser aplicadas às conexões de entrada e saída do pod, geralmente chamadas de regras de entrada e saída.

Com o suporte nativo à política de rede do Amazon VPC CNI Plugin, você pode implementar políticas de rede para proteger o tráfego de rede em clusters kubernetes. Isso se integra à API upstream de políticas de rede do Kubernetes, garantindo compatibilidade e aderência aos padrões do Kubernetes. Você pode definir políticas usando identificadores diferentes compatíveis com a API upstream. Por padrão, todo o tráfego de entrada e saída é permitido para um pod. Quando uma política de rede com um PolicyType Ingress é especificada, somente as conexões permitidas no pod são aquelas do nó do pod e aquelas permitidas pelas regras de entrada. O mesmo se aplica às regras de saída. Se várias regras forem definidas, a união de todas as regras será levada em consideração ao tomar a decisão. Assim, a ordem de avaliação não afeta o resultado da política.

> **Important**
>
> Quando você provisiona pela primeira vez um cluster EKS, a funcionalidade de política de rede VPC CNI não está habilitada por padrão. Certifique-se de ter implantado a versão compatível do complemento VPC CNI e `ENABLE_NETWORK_POLICY` defina a bandeira no complemento vpc-cni `true` para habilitá-la. Consulte o guia do usuário do Amazon EKS para obter instruções detalhadas.

Recomendações

Introdução às políticas de rede - siga o princípio do menor privilégio

Crie uma política de negação padrão

Assim como nas políticas de RBAC, é recomendável seguir os princípios de acesso menos privilegiado com as políticas de rede. Comece criando uma política de negar tudo que restringe todo o tráfego de entrada e saída em um namespace.

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

negação padrão

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

> **Note**
>
> A imagem acima foi criada pelo visualizador de políticas de rede da Tufin.

Crie uma regra para permitir consultas de DNS

Depois de estabelecer a regra padrão de negar tudo, você pode começar a criar regras adicionais, como uma regra que permite que os pods consultem o CoreDNS para resolução de nomes.

```
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	

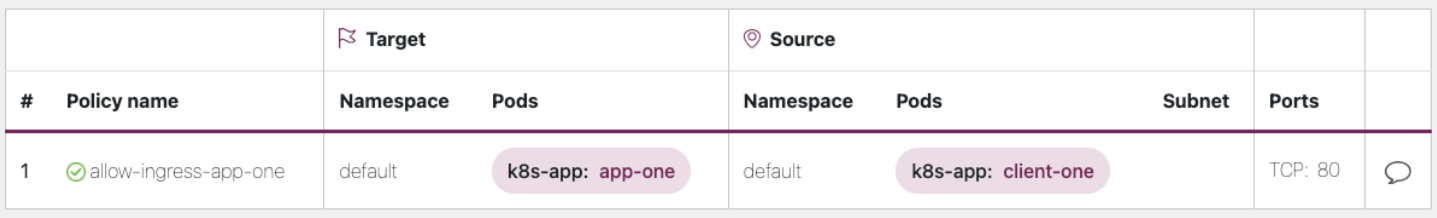
Adicione regras de forma incremental para permitir seletivamente o fluxo de tráfego entre namespaces/pods

Entenda os requisitos do aplicativo e crie regras de entrada e saída refinadas conforme necessário. O exemplo abaixo mostra como restringir o tráfego de entrada na porta 80 para `app-one` `declient-one`. Isso ajuda a minimizar a superfície de ataque e reduz o risco de acesso não 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	

Monitorando a aplicação da política de rede

- Use o editor de políticas de rede
  - O editor de políticas de rede ajuda com visualizações, pontuação de segurança e geração automática a partir de registros de fluxo de rede
  - Crie políticas de rede de forma interativa
- Registros de auditoria
  - Revise regularmente os registros de auditoria do seu cluster EKS
  - Os registros de auditoria fornecem muitas informações sobre quais ações foram realizadas em seu cluster, incluindo alterações nas políticas de rede.
  - Use essas informações para rastrear alterações em suas políticas de rede ao longo do tempo e detectar quaisquer alterações não autorizadas ou inesperadas.
- Teste automatizado

  - Implemente testes automatizados criando um ambiente de teste que espelhe seu ambiente de produção e implante periodicamente cargas de trabalho que tentem violar suas políticas de rede.
- Métricas de monitoramento
  - Configure seus agentes de observabilidade para extrair as métricas do prometheus dos agentes do nó CNI da VPC, o que permite monitorar a integridade do agente e os erros do SDK.
- Audite as políticas de rede regularmente
  - Audite periodicamente suas políticas de rede para garantir que elas atendam aos requisitos atuais de seus aplicativos. À medida que seu aplicativo evolui, uma auditoria oferece a oportunidade de remover regras redundantes de entrada e saída e garantir que seus aplicativos não tenham permissões excessivas.
- Verifique se as políticas de rede existem usando o Open Policy Agent (OPA)
  - Use a Política de OPA, conforme mostrado abaixo, para garantir que a Política de Rede sempre exista antes de integrar os pods de aplicativos. Essa política nega a integração de pods k8s com um rótulo `k8s-app: sample-app` se a política de rede correspondente não existir.

```
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
}
```

Solução de problemas

Monitore os vpc-network-policy-controller registros do node-agent

Ative os registros do gerenciador do controlador do plano de controle EKS para diagnosticar a funcionalidade da política de rede. Você pode transmitir os registros do plano de controle para um

grupo de CloudWatch registros e usar o CloudWatchLog Insights para realizar consultas avançadas. Nos registros, você pode ver quais objetos de endpoint do pod estão resolvidos em uma política de rede, o status de reconciliação das políticas e depurar se a política está funcionando conforme o esperado.

Além disso, o Amazon VPC CNI permite que você habilite a coleta e a exportação de registros de aplicação de políticas para o Amazon Cloudwatch a partir dos nós de trabalho do EKS. Depois de ativado, você pode aproveitar o CloudWatchContainer Insights para fornecer informações sobre seu uso relacionado às políticas de rede.

O Amazon VPC CNI também envia um SDK que fornece uma interface para interagir com os programas eBPF no nó. O SDK é instalado quando `aws-node` é implantado nos nós. Você pode encontrar o binário do SDK instalado no `/opt/cni/bin` diretório no nó. No lançamento, o SDK fornece suporte para funcionalidades fundamentais, como a inspeção de programas e mapas do eBPF.

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

Registrar metadados de tráfego de rede

O AWS VPC Flow Logs captura metadados sobre o tráfego que flui por uma VPC, como endereço IP e porta de origem e destino, juntamente com pacotes aceitos/descartados. Essas informações podem ser analisadas para procurar atividades suspeitas ou incomuns entre os recursos da VPC, incluindo pods. No entanto, como os endereços IP dos pods mudam frequentemente à medida que são substituídos, os registros de fluxo podem não ser suficientes por si só. O Calico Enterprise estende os registros de fluxo com rótulos de pod e outros metadados, facilitando a decifração dos fluxos de tráfego entre os pods.

Grupos de segurança

O EKS usa AWS VPC Security Groups (SGs) para controlar o tráfego entre o plano de controle do Kubernetes e os nós de trabalho do cluster. Os grupos de segurança também são usados para controlar o tráfego entre os nós de trabalho, outros recursos da VPC e endereços IP externos. Quando você provisiona um cluster EKS (com a versão 1.14-eks.3 ou superior do Kubernetes), um grupo de segurança de cluster é criado automaticamente para você. Esse grupo de segurança permite a comunicação sem restrições entre o plano de controle do EKS e os nós dos grupos de nós gerenciados. Para simplificar, é recomendável adicionar o cluster SG a todos os grupos de nós, incluindo grupos de nós não gerenciados.

Antes da versão 1.14 do Kubernetes e da versão eks.3 do EKS, havia grupos de segurança separados configurados para o plano de controle e os grupos de nós do EKS. As regras mínimas e sugeridas para os grupos de segurança do plano de controle e do grupo de nós podem ser encontradas em https://docs.aws.amazon.com/eks/ latest/userguide/sec -group-reqs.html. As regras mínimas para o grupo de segurança do plano de controle permitem a entrada da porta 443 do nó de trabalho SG. Essa regra é o que permite que os kubelets se comuniquem com o servidor da API Kubernetes. Também inclui a porta 10250 para tráfego de saída para o nó de trabalho SG; 10250 é a porta na qual os kubelets escutam. Da mesma forma, as regras mínimas de grupo de nós permitem a entrada da porta 10250 do plano de controle SG e a saída 443 do plano de controle SG. Finalmente, há uma regra que permite a comunicação irrestrita entre os nós dentro de um grupo de nós.

Se você precisar controlar a comunicação entre os serviços executados dentro do cluster e os serviços executados fora do cluster, como um banco de dados do RDS, considere grupos de segurança para pods. Com grupos de segurança para pods, você pode atribuir um grupo de segurança existente a uma coleção de pods.

> **Warning**
>
> Se você fizer referência a um grupo de segurança que não existia antes da criação dos pods, os pods não serão agendados.

Você pode controlar quais pods são atribuídos a um grupo de segurança criando um `SecurityGroupPolicy` objeto e especificando um `PodSelector` ou a. `ServiceAccountSelector` Definir os seletores como `{}` atribuirá o SGs referenciado em a todos os pods em um namespace ou `SecurityGroupPolicy` a todas as contas de serviço em um namespace. Certifique-se de ter se familiarizado com todas as considerações antes de implementar grupos de segurança para pods.

> **Important**
>
> Se você usa SGs para pods, deve criar uma porta SGs que permita a saída da porta 53 para o grupo de segurança do cluster. Da mesma forma, você deve atualizar o grupo de segurança do cluster para aceitar o tráfego de entrada da porta 53 do grupo de segurança do pod.

> **Important**
>
> Os limites para grupos de segurança ainda se aplicam ao usar grupos de segurança para pods, portanto, use-os criteriosamente.

> **Important**
>
> Você deve criar regras para o tráfego de entrada do grupo de segurança do cluster (kubelet) para todos os testes configurados para o pod.

> **Important**
>
> Os grupos de segurança para pods dependem de um recurso conhecido como entroncamento ENI, criado para aumentar a densidade de ENI de uma instância. EC2 Quando um pod é atribuído a um SG, um controlador de VPC associa uma ENI de ramificação do grupo de nós ao pod. Se não houver ramificações suficientes ENIs disponíveis em um grupo de nós no momento em que o pod for agendado, o pod permanecerá no estado pendente. O número de ramificações que ENIs uma instância pode suportar varia de acordo com a instância type/family. See https://docs.aws.amazon.com/eks/latest/userguide/security - groups-for-pods .html# supported-instance-types para obter mais detalhes.

Embora os grupos de segurança para pods ofereçam uma forma nativa da AWS de controlar o tráfego de rede dentro e fora do seu cluster sem a sobrecarga de um daemon de políticas, outras opções estão disponíveis. Por exemplo, o mecanismo de política do Cilium permite que você faça referência a um nome DNS em uma política de rede. O Calico Enterprise inclui uma opção para mapear políticas de rede para grupos de segurança da AWS. Se você implementou uma malha de serviços como o Istio, pode usar um gateway de saída para restringir a saída da rede a domínios ou endereços IP específicos e totalmente qualificados. Para obter mais informações sobre essa opção, leia a série de três partes sobre controle de tráfego de saída no Istio.

Quando usar a política de rede versus o grupo de segurança para pods?

Quando usar a política de rede do Kubernetes

- Controlando o pod-to-pod tráfego
  - Adequado para controlar o tráfego de rede entre pods dentro de um cluster (tráfego leste-oeste)
- Controle o tráfego no endereço IP ou no nível da porta (camada OSI 3 ou 4)

Quando usar grupos de segurança da AWS para pods (SGP)

- Aproveite as configurações existentes da AWS
  - Se você já tem um conjunto complexo de grupos de EC2 segurança que gerenciam o acesso aos serviços da AWS e está migrando aplicativos de EC2 instâncias para o EKS, SGPs pode ser uma ótima opção, permitindo que você reutilize recursos do grupo de segurança e os aplique aos seus pods.
- Controle o acesso aos serviços da AWS
  - Seus aplicativos executados em um cluster EKS desejam se comunicar com outros serviços da AWS (banco de dados RDS) e usá-los SGPs como um mecanismo eficiente para controlar o tráfego dos pods para os serviços da AWS.
- Isolamento do tráfego de Pod e Node
  - Se você quiser separar completamente o tráfego do pod do resto do tráfego do nó, use o SGP no `POD_SECURITY_GROUP_ENFORCING_MODE=strict` modo.

Melhores práticas de uso de grupos de segurança para pods e políticas de rede

- Segurança em camadas
  - Use uma combinação de políticas de rede SGP e kubernetes para uma abordagem de segurança em camadas
  - Use SGPs para limitar o acesso no nível da rede aos serviços da AWS que não fazem parte de um cluster, enquanto as políticas de rede do Kubernetes podem restringir o tráfego de rede entre os pods dentro do cluster
- Princípio do privilégio mínimo
  - Permita apenas o tráfego necessário entre pods ou namespaces
- Segmente seus aplicativos

  - Sempre que possível, segmente os aplicativos de acordo com a política de rede para reduzir o raio de explosão se um aplicativo for comprometido
- Mantenha as políticas simples e claras
  - As políticas de rede do Kubernetes podem ser bastante granulares e complexas; é melhor mantê-las o mais simples possível para reduzir o risco de configuração incorreta e facilitar a sobrecarga de gerenciamento
- Reduza a superfície de ataque
  - Minimize a superfície de ataque limitando a exposição de seus aplicativos

> **Important**
>
> Os grupos de segurança para pods oferecem dois modos de imposição: e. `strict` `standard` Você deve usar o `standard` modo ao usar a Política de Rede e os Grupos de Segurança para recursos de pods em um cluster EKS.

Quando se trata de segurança de rede, uma abordagem em camadas geralmente é a solução mais eficaz. Usar a política de rede kubernetes e o SGP em combinação pode fornecer uma defense-in-depth estratégia robusta para seus aplicativos executados no EKS.

Aplicação da política de Service Mesh ou política de rede Kubernetes

`service mesh`A é uma camada de infraestrutura dedicada que você pode adicionar aos seus aplicativos. Ele permite que você adicione de forma transparente recursos como observabilidade, gerenciamento de tráfego e segurança, sem adicioná-los ao seu próprio código.

O service mesh impõe políticas na camada 7 (aplicativo) do modelo OSI, enquanto as políticas de rede kubernetes operam na camada 3 (rede) e na camada 4 (transporte). Há muitas ofertas neste espaço, como AWSAppMesh, Istio, Linkerd etc.,

Quando usar o Service Mesh para aplicação de políticas

- Tenha um investimento existente em uma malha de serviços
- Precisa de recursos mais avançados, como gerenciamento de tráfego, observabilidade e segurança
  - Controle de tráfego, balanceamento de carga, interrupção de circuito, limitação de taxa, tempos limite, etc.

- Informações detalhadas sobre o desempenho de seus serviços (latência, taxas de erro, solicitações por segundo, volumes de solicitações etc.)
- Você deseja implementar e aproveitar o service mesh para recursos de segurança como mTLS

Escolha a política de rede do Kubernetes para casos de uso mais simples

- Limite quais pods podem se comunicar entre si
- As políticas de rede exigem menos recursos do que uma malha de serviços, o que as torna adequadas para casos de uso mais simples ou para clusters menores, nos quais a sobrecarga de executar e gerenciar uma malha de serviços pode não ser justificada.

> **Note**
>
> As políticas de rede e o Service Mesh também podem ser usados juntos. Use políticas de rede para fornecer um nível básico de segurança e isolamento entre seus pods e, em seguida, use uma malha de serviços para adicionar recursos adicionais, como gerenciamento de tráfego, observabilidade e segurança.

ThirdParty Mecanismos de política de rede

Considere um mecanismo de política de rede de terceiros quando você tiver requisitos de política avançados, como políticas de rede globais, suporte para regras baseadas em nomes de host DNS, regras de camada 7, regras ServiceAccount baseadas e deny/log ações explícitas, etc. O Calico é um mecanismo de políticas de código aberto da Tigera que funciona bem com o EKS. Além de implementar o conjunto completo de recursos de política de rede do Kubernetes, o Calico oferece suporte a políticas de rede estendidas com um conjunto mais rico de recursos, incluindo suporte para regras de camada 7, por exemplo, HTTP, quando integrado ao Istio. As políticas do Calico podem ser definidas para namespaces, pods, contas de serviço ou globalmente. Quando as políticas têm como escopo uma conta de serviço, ela associa um conjunto de ingress/egress regras a essa conta de serviço. Com as regras de RBAC adequadas em vigor, você pode impedir que as equipes ignorem essas regras, permitindo que os profissionais de segurança de TI deleguem com segurança a administração de namespaces. A Isovalent, mantenedora do Cilium, também estendeu as políticas de rede para incluir suporte parcial às regras da camada 7, por exemplo, HTTP. O Cilium também tem suporte para nomes de host DNS, que podem ser úteis para restringir o tráfego entre o Kubernetes Services/Pods e os recursos executados dentro ou fora da sua VPC. Por outro lado, o

Calico Enterprise inclui um recurso que permite mapear uma política de rede do Kubernetes para um grupo de segurança da AWS, bem como nomes de host DNS.

Você pode encontrar uma lista de políticas de rede comuns do Kubernetes em. https://github.com/ahmetb/ kubernetes-network-policy-recipes Um conjunto similar de regras para Calico está disponível em https://docs.projectcalico. org/security/calico-política de rede.

Migração para o Amazon VPC CNI Network Policy Engine

Para manter a consistência e evitar um comportamento inesperado de comunicação entre pods, é recomendável implantar somente um Network Policy Engine em seu cluster. Se você quiser migrar do 3P para o VPC CNI Network Policy Engine, recomendamos converter seu 3P existente para os recursos do Kubernetes antes de ativar NetworkPolicy CRDs o suporte à política de rede VPC CNI. NetworkPolicy E teste as políticas migradas em um cluster de teste separado antes de aplicá-las em seu ambiente de produção. Isso permite que você identifique e resolva possíveis problemas ou inconsistências no comportamento de comunicação do pod.

Ferramenta de migração

Para auxiliar em seu processo de migração, desenvolvemos uma ferramenta chamada K8s Network Policy Migrator que converte sua política de rede existente em políticas CRDs de Calico/Cilium rede nativas do Kubernetes. Após a conversão, você pode testar diretamente as políticas de rede convertidas em seus novos clusters executando o controlador de política de rede VPC CNI. A ferramenta foi projetada para ajudar você a simplificar o processo de migração e garantir uma transição tranquila.

> **Important**
>
> A ferramenta de migração converterá apenas políticas 3P compatíveis com a API de política de rede Kubernetes nativa. Se você estiver usando recursos avançados de política de rede oferecidos pelos plug-ins 3P, a ferramenta de migração os ignorará e os reportará.

Observe que a ferramenta de migração atualmente não é suportada pela equipe de engenharia de políticas da rede AWS VPC CNI. Ela é disponibilizada aos clientes com base no melhor esforço possível. Recomendamos que você utilize essa ferramenta para facilitar seu processo de migração. Caso você encontre algum problema ou bug com a ferramenta, pedimos que você crie um GitHubproblema. Seu feedback é inestimável para nós e ajudará na melhoria contínua de nossos serviços.

Recursos adicionais

- Kubernetes e Tigera: políticas de rede, segurança e auditoria
- Empresa Calico
- Cilium
- NetworkPolicyEditor: um editor de políticas interativo da Cilium
- Inspektor Gadget advise gadget gadget de política de rede Sugere políticas de rede com base em uma análise do tráfego de rede

Criptografia em trânsito

Os aplicativos que precisam estar em conformidade com PCI, HIPAA ou outras regulamentações podem precisar criptografar os dados enquanto eles estão em trânsito. Atualmente, o TLS é a escolha de fato para criptografar o tráfego na rede. O TLS, assim como seu antecessor SSL, fornece comunicações seguras em uma rede usando protocolos criptográficos. O TLS usa criptografia simétrica em que as chaves para criptografar os dados são geradas com base em um segredo compartilhado que é negociado no início da sessão. Veja a seguir algumas maneiras de criptografar dados em um ambiente Kubernetes.

Instâncias Nitro

O tráfego trocado entre os seguintes tipos de instância Nitro, por exemplo, C5n, G4, I3en, M5dn, M5n, P3dn, R5dn e R5n, é criptografado automaticamente por padrão. Quando há um salto intermediário, como um gateway de trânsito ou um balanceador de carga, o tráfego não é criptografado. Consulte Criptografia em trânsito para obter mais detalhes sobre criptografia em trânsito, bem como a lista completa dos tipos de instâncias que oferecem suporte à criptografia de rede por padrão.

Interfaces de rede de contêineres (CNIs)

WeaveNetpode ser configurado para criptografar automaticamente todo o tráfego usando NaCl criptografia para tráfego intermediário e IPsec ESP para tráfego rápido de caminhos de dados.

Malha de serviços

A criptografia em trânsito também pode ser implementada com uma malha de serviços como App Mesh, Linkerd v2 e Istio. AppMesh suporta mTLS com certificados X.509 ou Envoy's Secret Discovery Service (SDS). Tanto o Linkerd quanto o Istio têm suporte para mTLS.

O aws-app-mesh-examplesGitHub repositório fornece orientações para configurar o mTLS usando certificados X.509 e o SPIRE como provedor de SDS com seu contêiner Envoy:

- Configurando mTLS usando certificados X.509
- Configurando o TLS usando o SPIRE (SDS)

O App Mesh também oferece suporte à criptografia TLS com um certificado privado emitido pelo AWS Certificate Manager (ACM) ou um certificado armazenado no sistema de arquivos local do nó virtual.

O aws-app-mesh-examplesGitHub repositório fornece orientações para configurar o TLS usando certificados emitidos pelo ACM e certificados fornecidos com seu contêiner Envoy:

- Configurando o TLS com certificados TLS fornecidos pelo arquivo
- Configurando o TLS com o AWS Certificate Manager

Controladores de entrada e balanceadores de carga

Os controladores de entrada são uma forma de rotear de forma inteligente o 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 processamento que consome muita CPU. Consequentemente, se você tiver flexibilidade, tente realizar o descarregamento de SSL no Ingress ou no balanceador de carga.

Use criptografia com balanceadores de carga elásticos da AWS

O AWS Application Load Balancer (ALB) e o Network Load Balancer (NLB) têm suporte para criptografia de transporte (SSL e TLS). A `alb.ingress.kubernetes.io/certificate-arn` anotação para o ALB permite que você especifique quais certificados adicionar ao ALB. Se você omitir a anotação, o controlador tentará adicionar certificados aos ouvintes que a exigirem, combinando os certificados disponíveis do AWS Certificate Manager (ACM) usando o campo host. A partir do EKS v1.15, você pode usar a `service.beta.kubernetes.io/aws-load-balancer-ssl-cert` anotação com o NLB, conforme mostrado no exemplo abaixo.

```
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
```

A seguir estão exemplos adicionais de SSL/TLS rescisão.

- Protegendo o EKS Ingress com Contour e vamos criptografar o caminho GitOps
- Como encerro o tráfego HTTPS nas cargas de trabalho do Amazon EKS com o ACM?

> **Important**
>
> Algumas entradas, como o controlador AWS LB, implementam o SSL/TLS uso de anotações em vez de como parte da especificação de entrada.

CA privada do ACM com cert-manager

Você pode habilitar o TLS e o mTLS para proteger suas cargas de trabalho do aplicativo EKS na entrada, no pod e entre os pods usando a ACM Private Certificate Authority (CA) e o cert-manager, um complemento popular do Kubernetes para distribuir, renovar e revogar certificados. A CA privada do ACM é uma CA altamente disponível, segura e gerenciada, sem os custos iniciais e de manutenção de gerenciar sua própria CA. Se você estiver usando a autoridade de certificação padrão do Kubernetes, há uma oportunidade de melhorar sua segurança e atender aos requisitos de conformidade com a CA privada do ACM. A CA privada do ACM protege chaves privadas nos módulos de segurança de hardware FIPS 140-2 de nível 3 (muito seguros), em comparação com a CA padrão que armazena chaves codificadas na memória (menos segura). Uma CA centralizada também oferece mais controle e melhor auditabilidade para certificados privados dentro e fora de um ambiente Kubernetes.

Modo CA de curta duração para TLS mútuo entre cargas de trabalho

Ao usar a CA privada do ACM para mTLS no EKS, é recomendável usar certificados de curta duração com o modo CA de curta duração. Embora seja possível emitir certificados de curta duração no modo CA de uso geral, usar o modo CA de curta duração é mais econômico (~ 75% mais barato que o modo geral) para casos de uso em que novos certificados precisam ser emitidos com frequência. Além disso, você deve tentar alinhar o período de validade dos certificados privados com a vida útil dos pods em seu cluster EKS. Saiba mais sobre o ACM Private CA e seus benefícios aqui.

Instruções de configuração do ACM

Comece criando uma CA privada seguindo os procedimentos fornecidos nos documentos técnicos da CA privada do ACM. Depois de ter uma CA privada, instale o cert-manager usando as instruções de instalação regulares. Depois de instalar o cert-manager, instale o plug-in privado do CA Kubernetes cert-manager seguindo as instruções de configuração em. GitHub O plug-in permite que o cert-manager solicite certificados privados da CA privada do ACM.

Agora que você tem uma CA privada e um cluster EKS com cert-manager e o plug-in instalados, é hora de definir as permissões e criar o emissor. Atualize as permissões do IAM da função do nó EKS para permitir o acesso à CA privada do ACM. <CA_ARN>Substitua o pelo valor da sua 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"
        }
    ]
}
```

As funções de serviço para contas do IAM ou IRSA também podem ser usadas. Consulte a seção Recursos adicionais abaixo para ver exemplos completos.

Crie um emissor no Amazon EKS criando um arquivo de definição de recurso personalizado chamado cluster-issuer.yaml com o texto a seguir, substituindo as informações por sua CA privada. <CA_ARN> <Region>

```
apiVersion: awspca.cert-manager.io/v1beta1
kind: AWSPCAClusterIssuer
metadata:
          name: demo-test-root-ca
spec:
```

```
        arn: <CA_ARN>
        region: <Region>
```

Implante o emissor que você criou.

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

Seu cluster EKS está configurado para solicitar certificados da CA privada. Agora você pode usar o `Certificate` recurso do cert-manager para emitir certificados alterando os valores do `issuerRef` campo para o emissor de CA privado que você criou acima. Para obter mais detalhes sobre como especificar e solicitar recursos de certificado, consulte o guia de recursos de certificados do cert-manager. Veja exemplos aqui.

CA privada do ACM com Istio e cert-manager

Se você estiver executando o Istio em seu cluster EKS, poderá desativar o plano de controle do Istio (especificamente`istiod`) de funcionar como a Autoridade Certificadora (CA) raiz e configurar a CA privada do ACM como a CA raiz para mTLS entre cargas de trabalho. Se você optar por essa abordagem, considere usar o modo CA de curta duração na CA privada do ACM. Consulte a seção anterior e esta postagem do blog para obter mais detalhes.

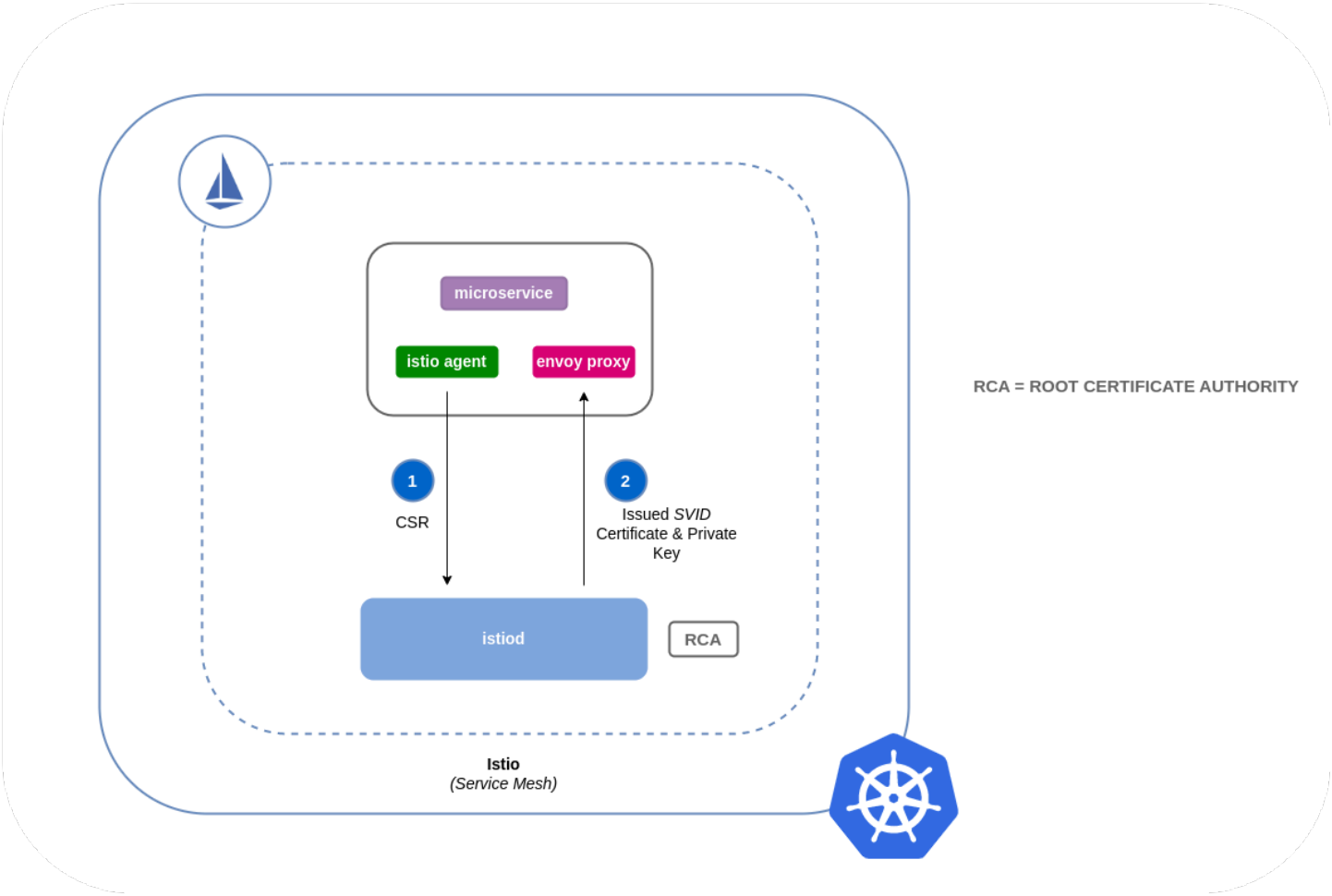
Como funciona a assinatura de certificados no Istio (padrão)

As cargas de trabalho no Kubernetes são identificadas usando contas de serviço. Se você não especificar uma conta de serviço, o Kubernetes atribuirá automaticamente uma à sua carga de trabalho. Além disso, as contas de serviço montam automaticamente um token associado. Esse token é usado pela conta de serviço para que as cargas de trabalho sejam autenticadas na API Kubernetes. A conta de serviço pode ser suficiente como identidade para o Kubernetes, mas o Istio tem seu próprio sistema de gerenciamento de identidade e CA. Quando uma carga de trabalho é iniciada com seu proxy auxiliar de envio, ela precisa de uma identidade atribuída pelo Istio para ser considerada confiável e ter permissão para se comunicar com outros serviços na malha.

Para obter essa identidade do Istio, `istio-agent` ele envia uma solicitação conhecida como solicitação de assinatura de certificado (ou CSR) para o plano de controle do Istio. Essa CSR contém o token da conta de serviço para que a identidade da carga de trabalho possa ser verificada antes de ser processada. Esse processo de verificação é realizado por`istiod`, que atua como Autoridade de Registro (ou RA) e CA. O RA serve como um guardião que garante que somente o CSR verificado chegue à CA. Depois que a CSR for verificada, ela será encaminhada para a CA, que emitirá um

certificado contendo uma identidade SPIFFE com a conta de serviço. Esse certificado é chamado de documento de identidade verificável SPIFFE (ou SVID). O SVID é atribuído ao serviço solicitante para fins de identificação e para criptografar o tráfego em trânsito entre os serviços de comunicação.

Fluxo padrão para solicitações de assinatura de certificados do Istio:



Como funciona a assinatura de certificados no Istio com ACM Private CA

Você pode usar um complemento cert-manager chamado agente de solicitação de assinatura de certificado do Istio (istio-csr) para integrar o Istio à CA privada do ACM. Esse agente permite que as cargas de trabalho e os componentes do plano de controle do Istio sejam protegidos pelos emissores do gerenciador de certificados, neste caso, o ACM Private CA. O agente istio-csr expõe o mesmo serviço que o istiod serve na configuração padrão de validação de entrada. CSRs Exceto que, após a verificação, ele converterá as solicitações em recursos que o cert manager suporta (ou seja, integrações com emissores externos de CA).

Sempre que houver uma CSR de uma carga de trabalho, ela será encaminhada para istio-csr, que solicitará certificados da CA privada do ACM. Essa comunicação entre istio-csr e a CA privada

do ACM é ativada pelo plug-in do emissor da CA privada da AWS. O gerenciador de certificados usa esse plug-in para solicitar certificados TLS da CA privada do ACM. O plug-in do emissor se comunicará com o serviço ACM Private CA para solicitar um certificado assinado para a carga de trabalho. Depois que o certificado for assinado, ele será devolvido ao istio-csr, que lerá a solicitação assinada e a retornará à carga de trabalho que iniciou a CSR.

Fluxo para solicitações de assinatura de certificados do Istio com istio-csr

imagem: istio-csr-with-acm -private-ca.png [Fluxo para solicitações de assinatura de certificados do Istio com istio-csr]

Instruções de configuração do Istio com CA privada

1. Comece seguindo as mesmas instruções de configuração nesta seção para concluir o seguinte:
2. Criar uma CA privada
3. Instalar cert-manager
4. Instale o plug-in do emissor
5. Defina permissões e crie um emissor. O emissor representa a CA e é usado para assinar `istiod` e combinar certificados de carga de trabalho. Ele se comunicará com o ACM Private CA.
6. Crie um `istio-system` namespace. É aqui que o `istiod certificate` e outros recursos do Istio serão implantados.
7. Instale o Istio CSR configurado com o plug-in AWS Private CA Issuer. Você pode preservar as solicitações de assinatura de certificados para cargas de trabalho para verificar se elas foram aprovadas e assinadas (`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 o Istio com configurações personalizadas para `istiod` substituí-las `cert-manager istio-csr` como provedor de certificados para a malha. Esse processo pode ser realizado usando o Istio Operator.

```
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. Implante o recurso personalizado acima que você criou.

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

10 Agora você pode implantar uma carga de trabalho na malha do seu cluster EKS e aplicar o mTLS.

Solicitações de assinatura de certificados do Istio

imagem: istio-csr-requests .png [Solicitações de assinatura de certificado Istio]

Ferramentas e recursos

- Workshop de imersão em segurança do Amazon EKS — Segurança de rede
- Como implementar o cert-manager e o plug-in ACM Private CA para habilitar o TLS no EKS.
- Configurando a criptografia end-to-end TLS no Amazon EKS com o novo AWS Load Balancer Controller e o ACM Private CA.
- Plug-in privado do gerenciador de certificados CA Kubernetes ativado. GitHub
- Guia do usuário do plug-in privado do CA Kubernetes cert-manager.
- Como usar o modo de certificado de curta duração da Autoridade de Certificação Privada da AWS
- egress-operator Um operador e um plug-in de DNS para controlar o tráfego de saída do seu cluster sem inspeção de protocolo

- NeuVector da SUSE, a plataforma de segurança de contêineres de código aberto e de confiança zero fornece regras de rede de políticas, prevenção de perda de dados (DLP), firewall de aplicativos web (WAF) e assinaturas de ameaças à rede.

Criptografia de dados e gerenciamento de segredos

Criptografia em repouso

Há três opções diferentes de armazenamento nativo da AWS que você pode usar com o Kubernetes: EBS, EFS e for Lustre. FSx Todos os três oferecem criptografia em repouso usando uma chave gerenciada por serviço ou uma chave mestra do cliente (CMK). Para o EBS, você pode usar o driver de armazenamento em árvore ou o driver EBS CSI. Ambos incluem parâmetros para criptografar volumes e fornecer uma CMK. Para o EFS, você pode usar o driver EFS CSI, no entanto, diferentemente do EBS, o driver EFS CSI não oferece suporte ao provisionamento dinâmico. Se você quiser usar o EFS com o EKS, precisará provisionar e configurar a criptografia em repouso para o sistema de arquivos antes de criar um PV. Para obter mais informações sobre a criptografia de arquivos EFS, consulte Criptografando dados em repouso. Além de oferecer criptografia em repouso, o EFS e FSx o Lustre incluem uma opção para criptografar dados em trânsito. FSx for Lustre faz isso por padrão. Para o EFS, você pode adicionar criptografia de transporte adicionando o `tls` parâmetro ao `mountOptions` seu PV, como neste exemplo:

```
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>
```

O driver FSx CSI oferece suporte ao provisionamento dinâmico dos sistemas de arquivos Lustre. Por padrão, ele criptografa os dados com uma chave gerenciada por serviço, embora haja a opção de fornecer sua própria CMK, como neste exemplo:

```
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**
>
> Em 28 de maio de 2020, todos os dados gravados no volume efêmero nos pods do EKS Fargate são criptografados por padrão usando um algoritmo criptográfico AES-256 padrão do setor. Nenhuma modificação em seu aplicativo é necessária, pois a criptografia e a descriptografia são tratadas sem problemas pelo serviço.

Criptografia de dados em repouso

Criptografar dados em repouso é considerada uma prática recomendada. Se você não tiver certeza se a criptografia é necessária, criptografe seus dados.

Gire seu periodicamente CMKs

Configure o KMS para girar automaticamente seu. CMKs Isso girará suas chaves uma vez por ano e salvará as chaves antigas indefinidamente para que seus dados ainda possam ser descriptografados. Para obter informações adicionais, consulte Chaves mestras rotativas do cliente

Use pontos de acesso EFS para simplificar o acesso a conjuntos de dados compartilhados

Se você tiver conjuntos de dados compartilhados com permissões de arquivo POSIX diferentes ou quiser restringir o acesso a parte do sistema de arquivos compartilhado criando pontos de montagem

diferentes, considere usar pontos de acesso do EFS. Para saber mais sobre como trabalhar com pontos de acesso, consulte https://docs.aws.amazon.com/efs/ latest/ug/efs -access-points.html. Hoje, se você quiser usar um ponto de acesso (AP), precisará referenciar o AP no `volumeHandle` parâmetro do PV.

> **Important**
>
> A partir de 23 de março de 2021, o driver EFS CSI oferece suporte ao provisionamento dinâmico de pontos de acesso EFS. Os pontos de acesso são pontos de entrada específicos do aplicativo em um sistema de arquivos EFS que facilitam o compartilhamento de um sistema de arquivos entre vários pods. Cada sistema de arquivos EFS pode ter até 120 PVs. Consulte Introdução ao provisionamento dinâmico CSI do Amazon EFS para obter informações adicionais.

Gerenciamento de segredos

Os segredos do Kubernetes são usados para armazenar informações confidenciais, como certificados de usuário, senhas ou chaves de API. Eles são mantidos no etcd como strings codificadas em base64. No EKS, os volumes do EBS para nós etcd são criptografados com a criptografia do EBS. Um pod pode recuperar objetos secretos do Kubernetes referenciando o segredo no. `podSpec` Esses segredos podem ser mapeados para uma variável de ambiente ou montados como volume. Para obter informações adicionais sobre a criação de segredos, consulte https://kubernetes. io/docs/concepts/configuration/secret/.

> **Warning**
>
> Os segredos em um namespace específico podem ser referenciados por todos os pods no namespace do segredo.

> **Warning**
>
> O autorizador do nó permite que o Kubelet leia todos os segredos montados no nó.

Use o AWS KMS para criptografia de envelopes de segredos do Kubernetes

Isso permite que você criptografe seus segredos com uma chave de criptografia de dados (DEK) exclusiva. Em seguida, a DEK é criptografada usando uma chave de criptografia de chave (KEK) do AWS KMS, que pode ser rotacionada automaticamente de acordo com uma programação recorrente. Com o plug-in KMS para Kubernetes, todos os segredos do Kubernetes são armazenados no etcd em texto cifrado em vez de texto simples e só podem ser descriptografados pelo servidor da API Kubernetes. Para obter detalhes adicionais, consulte Como usar o suporte do provedor de criptografia EKS para defesa em detalhes

Audite o uso dos segredos do Kubernetes

No EKS, ative o registro de auditoria e crie um filtro de CloudWatch métricas e um alarme para alertá-lo quando um segredo for usado (opcional). Veja a seguir um exemplo de filtro de métricas para o registro de auditoria do Kubernetes,. `{($.verb="get") && ($.objectRef.resource="secret")}` Você também pode usar as seguintes consultas com o 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"
```

A consulta acima exibirá o número de vezes que um segredo foi acessado em um período 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
```

Essa consulta exibirá o segredo, junto com o namespace e o nome de usuário do usuário que tentou acessar o segredo e o código de resposta.

Alterne seus segredos periodicamente

O Kubernetes não alterna automaticamente os segredos. Se você precisar alternar segredos, considere usar um armazenamento secreto externo, por exemplo, Vault ou AWS Secrets Manager.

Use namespaces separados como forma de isolar segredos de diferentes aplicativos

Se você tiver segredos que não podem ser compartilhados entre aplicativos em um namespace, crie um namespace separado para esses aplicativos.

Use montagens de volume em vez de variáveis de ambiente

Os valores das variáveis de ambiente podem aparecer acidentalmente nos registros. Os segredos montados como volumes são instanciados como volumes tmpfs (um sistema de arquivos baseado em RAM) que são automaticamente removidos do nó quando o pod é excluído.

Use um provedor externo de segredos

Há várias alternativas viáveis para usar segredos do Kubernetes, incluindo o AWS Secrets Manager e o Hashicorp's Vault. Esses serviços oferecem recursos como controles de acesso refinados, criptografia forte e rotação automática de segredos que não estão disponíveis com o Kubernetes Secrets. O Bitnami's Sealed Secrets é outra abordagem que usa criptografia assimétrica para criar “segredos selados”. Uma chave pública é usada para criptografar o segredo, enquanto a chave privada usada para descriptografar o segredo é mantida no cluster, permitindo que você armazene segredos selados com segurança em sistemas de controle de origem como o Git. Consulte Gerenciando a implantação de segredos no Kubernetes usando segredos selados para obter mais informações.

À medida que o uso de armazenamentos externos de segredos cresceu, também aumentou a necessidade de integrá-los ao Kubernetes. O Secret Store CSI Driver é um projeto comunitário que usa o modelo de driver CSI para buscar segredos de lojas secretas externas. Atualmente, o driver tem suporte para AWS Secrets Manager, Azure, Vault e GCP. O provedor da AWS oferece suporte ao AWS Secrets Manager e ao AWS Parameter Store. Ele também pode ser configurado para alternar segredos quando eles expirarem e pode sincronizar segredos do AWS Secrets Manager com segredos do Kubernetes. A sincronização de segredos pode ser útil quando você precisa referenciar um segredo como uma variável de ambiente em vez de lê-lo em um volume.

> **Note**
>
> Quando o driver CSI do armazenamento secreto precisa buscar um segredo, ele assume a função IRSA atribuída ao pod que faz referência a um segredo. O código dessa operação pode ser encontrado aqui.

Para obter informações adicionais sobre o AWS Secrets & Configuration Provider (ASCP), consulte os seguintes recursos:

- Como usar o AWS Secrets Configuration Provider com o driver CSI do Kubernetes Secret Store
- Integrando segredos do Secrets Manager com o driver CSI do Kubernetes Secrets Store

external-secrets é outra forma de usar um armazenamento secreto externo com o Kubernetes. Assim como o CSI Driver, os segredos externos funcionam em uma variedade de back-ends diferentes, incluindo o AWS Secrets Manager. A diferença é que, em vez de recuperar segredos do armazenamento secreto externo, os segredos externos copiam os segredos desses back-ends para o Kubernetes como segredos. Isso permite que você gerencie segredos usando seu repositório secreto preferido e interaja com os segredos de uma forma nativa do Kubernetes.

Ferramentas e recursos

- Workshop de imersão em segurança do Amazon EKS — Criptografia de dados e gerenciamento de segredos

Segurança de runtime

A segurança de tempo de execução fornece proteção ativa para seus contêineres enquanto eles estão em execução. A ideia é detectar e and/or evitar que atividades maliciosas ocorram dentro do contêiner. Isso pode ser feito com vários mecanismos no kernel Linux ou nas extensões do kernel integradas ao Kubernetes, como recursos do Linux, computação segura (seccomp) ou. AppArmor SELinux Também há opções como a Amazon GuardDuty e ferramentas de terceiros que podem ajudar a estabelecer linhas de base e detectar atividades anômalas com menos configuração manual dos mecanismos do kernel Linux.

> **Important**
>
> Atualmente, o Kubernetes não fornece nenhum mecanismo nativo para carregar seccomp ou perfis nos nós AppArmor. SELinux Eles precisam ser carregados manualmente ou instalados nos Nodes quando são inicializados. Isso precisa ser feito antes de referenciá-los em seus pods, pois o agendador não sabe quais nós têm perfis. Veja abaixo como ferramentas como o Security Profiles Operator podem ajudar a automatizar o provisionamento de perfis em nós.

Contextos de segurança e controles Kubernetes integrados

Muitos mecanismos de segurança de tempo de execução do Linux estão totalmente integrados ao Kubernetes e podem ser configurados por meio de contextos de segurança do Kubernetes. Uma dessas opções é o `privileged` sinalizador, que é `false` por padrão e, se ativado, é essencialmente equivalente à raiz no host. Quase sempre é inapropriado ativar o modo privilegiado em cargas de trabalho de produção, mas há muitos outros controles que podem fornecer privilégios mais granulares aos contêineres, conforme apropriado.

Capacidades do Linux

Os recursos do Linux permitem que você conceda determinados recursos a um pod ou contêiner sem fornecer todas as habilidades do usuário root. Os exemplos incluem`CAP_NET_ADMIN`, que permite configurar interfaces de rede ou firewalls, ou`CAP_SYS_TIME`, que permite a manipulação do relógio do sistema.

Seccomp

Com a computação segura (seccomp), você pode impedir que um aplicativo em contêiner faça determinadas chamadas de sistema para o kernel do sistema operacional hospedeiro subjacente. Embora o sistema operacional Linux tenha algumas centenas de chamadas de sistema, a maior parte delas não é necessária para executar contêineres. Ao restringir quais syscalls podem ser feitas por um contêiner, você pode reduzir efetivamente a superfície de ataque do seu aplicativo.

O Seccomp funciona interceptando chamadas de sistema e permitindo a passagem apenas daquelas que foram listadas na lista de permissões. O Docker tem um perfil seccomp padrão que é adequado para a maioria das cargas de trabalho de uso geral, e outros tempos de execução de contêineres, como containerd, fornecem padrões comparáveis. Você pode configurar seu contêiner ou pod para usar o perfil seccomp padrão do tempo de execução do contêiner adicionando o seguinte à `securityContext` seção da especificação do pod:

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

A partir de 1.22 (em alfa, estável a partir de 1.27), o código acima `RuntimeDefault` pode ser usado para todos os pods em um Node usando um único sinalizador kubelet,. `--seccomp-default` Então, o perfil especificado em só `securityContext` é necessário para outros perfis.

Também é possível criar seus próprios perfis para coisas que exigem privilégios adicionais. Isso pode ser muito tedioso de fazer manualmente, mas existem ferramentas como o Inspektor Gadget (também recomendado na seção de segurança de rede para gerar políticas de rede) e o Security Profiles Operator que suportam o uso de ferramentas como eBPF ou logs para registrar os requisitos básicos de privilégios como perfis seccomp. O Security Profiles Operator permite ainda automatizar a implantação de perfis gravados em nós para uso por pods e contêineres.

AppArmor and SELinux

AppArmor e SELinux são conhecidos como sistemas de controle de acesso obrigatório ou MAC. Eles têm um conceito semelhante ao seccomp, mas com habilidades diferentes APIs , permitindo o controle de acesso, por exemplo, para caminhos específicos do sistema de arquivos ou portas de rede. O suporte para essas ferramentas depende da distribuição Linux, com Debian/Ubuntu suporte AppArmor e suporte para RHEL/CentOS/Bottlerocket/Amazon Linux 2023 SELinux. Consulte também a seção de segurança da infraestrutura para uma discussão mais aprofundada sobre SELinux.

Ambos AppArmor SELinux estão integrados ao Kubernetes, mas a partir do Kubernetes 1.28, os AppArmor perfis devem ser especificados por meio de anotações, enquanto os SELinux rótulos podem ser definidos diretamente no campo Opções no contexto de segurança. SELinux

Assim como nos perfis seccomp, o operador de perfis de segurança mencionado acima pode ajudar na implantação de perfis nos nós do cluster. (No futuro, o projeto também visa gerar perfis para AppArmor e SELinux como faz para o seccomp.)

Recomendações

Use a Amazon GuardDuty para monitorar o tempo de execução e detectar ameaças aos seus ambientes EKS

Se você atualmente não tem uma solução para monitorar continuamente os tempos de execução do EKS, analisar os registros de auditoria do EKS e verificar se há malware e outras atividades suspeitas, a Amazon recomenda fortemente o uso da Amazon GuardDuty para clientes que desejam uma forma simples, rápida, segura, escalável e econômica de proteger seus ambientes da AWS com apenas um clique. GuardDuty A Amazon é um serviço de monitoramento de segurança que analisa e processa fontes de dados fundamentais, como eventos de CloudTrail gerenciamento da AWS, registros de eventos da AWS CloudTrail , registros de fluxo de VPC (de EC2 instâncias da Amazon), registros de auditoria do Kubernetes e registros de DNS. Também inclui monitoramento de tempo

de execução do EKS. Ele usa feeds de inteligência de ameaças continuamente atualizados, como listas de endereços IP e domínios maliciosos e aprendizado de máquina para identificar atividades inesperadas, potencialmente não autorizadas e maliciosas em seu ambiente da AWS. Isso pode incluir problemas como escalonamento de privilégios, uso de credenciais expostas ou comunicação com endereços IP e domínios maliciosos, presença de malware em suas EC2 instâncias da Amazon e cargas de trabalho de contêineres EKS ou descoberta de atividades suspeitas de API. GuardDuty informa você sobre o status do seu ambiente da AWS produzindo descobertas de segurança que você pode visualizar no GuardDuty console ou por meio da Amazon EventBridge. GuardDuty também fornece suporte para você exportar suas descobertas para um bucket do Amazon Simple Storage Service (S3) e integrá-las a outros serviços, como o AWS Security Hub e o Detective.

Assista à palestra técnica on-line da AWS "Detecção aprimorada de ameaças para o Amazon EKS com a Amazon GuardDuty — AWS Online Tech Talks" para ver como habilitar esses recursos adicionais de segurança do EKS step-by-step em minutos.

Opcionalmente: use uma solução terceirizada para monitoramento do tempo de execução

Criar e gerenciar perfis seccomp e Apparmor pode ser difícil se você não estiver familiarizado com a segurança do Linux. Se você não tiver tempo para se tornar proficiente, considere usar uma solução comercial terceirizada. Muitos deles foram além dos perfis estáticos, como Apparmor e seccomp, e começaram a usar o aprendizado de máquina para bloquear ou alertar sobre atividades suspeitas. Algumas dessas soluções podem ser encontradas abaixo na seção de ferramentas. Opções adicionais podem ser encontradas no AWS Marketplace for Containers.

Considere os recursos do add/dropping Linux antes de escrever políticas seccomp

Os recursos envolvem várias verificações nas funções do kernel acessíveis por syscalls. Se a verificação falhar, a syscall normalmente retornará um erro. A verificação pode ser feita logo no início de uma syscall específica ou mais profundamente no kernel em áreas que podem ser acessadas por meio de várias syscalls diferentes (como gravar em um arquivo privilegiado específico). O Seccomp, por outro lado, é um filtro de syscall que é aplicado a todas as syscalls antes de serem executadas. Um processo pode configurar um filtro que permita revogar seu direito de executar determinadas syscalls ou argumentos específicos para determinadas syscalls.

Antes de usar o seccomp, considere se os recursos do adding/removing Linux oferecem o controle de que você precisa. Consulte Configuração de recursos para contêineres para obter mais informações.

Veja se você pode atingir seus objetivos usando as Políticas de Segurança do Pod (PSPs)

As políticas de segurança do Pod oferecem várias maneiras diferentes de melhorar sua postura de segurança sem introduzir complexidade indevida. Explore as opções disponíveis PSPs antes de se aventurar na criação de perfis seccomp e Apparmor.

> **Warning**
>
> A partir do Kubernetes 1.25, PSPs foram removidos e substituídos pelo controlador Pod Security Admission. As alternativas de terceiros que existem incluem OPA/Gatekeeper Kyverno. Uma coleção de restrições e modelos de restrições do Gatekeeper para implementar políticas comumente encontradas em PSPs pode ser extraída do repositório da biblioteca do Gatekeeper em. GitHub E muitos substitutos para PSPs podem ser encontrados na biblioteca de políticas do Kyverno, incluindo a coleção completa de padrões de segurança do Pod.

Ferramentas e recursos

- 7 coisas que você deve saber antes de começar
- AppArmorCarregador
- Configurando nós com perfis
- O Security Profiles Operator é um aprimoramento do Kubernetes que visa facilitar o uso dos usuários SELinux, seccomp e em clusters do Kubernetes. AppArmor Ele fornece recursos para gerar perfis a partir da execução de cargas de trabalho e carregar perfis nos nós do Kubernetes para uso em pods.
- O Inspektor Gadget permite inspecionar, rastrear e traçar o perfil de muitos aspectos do comportamento do tempo de execução no Kubernetes, incluindo assistência na geração de perfis seccomp.
- Aqua
- Qualys
- Stackrox
- Sysdig Secure
- Prisma

- NeuVector da SUSE open source, a plataforma de segurança de contêineres de confiança zero fornece regras de perfil de processo e regras de acesso a arquivos.

Protegendo a infraestrutura (hosts)

Por mais que seja importante proteger suas imagens de contêiner, é igualmente importante proteger a infraestrutura que as executa. Esta seção explora maneiras diferentes de mitigar os riscos de ataques lançados diretamente contra o host. Essas diretrizes devem ser usadas em conjunto com as descritas na seção Runtime Security.

Recomendações

Use um sistema operacional otimizado para executar contêineres

Considere usar o Flatcar Linux, o Project Atomic, o RancherOS e o Bottlerocket, um sistema operacional especial da AWS projetado para executar contêineres Linux. Ele inclui uma superfície de ataque reduzida, uma imagem de disco que é verificada na inicialização e limites de permissão impostos usando SELinux.

Como alternativa, use a AMI otimizada para EKS para seus nós de trabalho do Kubernetes. A AMI otimizada para EKS é lançada regularmente e contém um conjunto mínimo de pacotes de sistema operacional e binários necessários para executar suas cargas de trabalho em contêineres.

Consulte a especificação de compilação do Amazon EKS AMI RHEL para obter um exemplo de script de configuração que pode ser usado para criar uma AMI personalizada do Amazon EKS em execução no Red Hat Enterprise Linux usando o Hashicorp Packer. Esse script pode ser ainda mais aproveitado para criar um EKS personalizado compatível com STIG. AMIs

Mantenha o sistema operacional do seu node de trabalho atualizado

Independentemente de você usar um sistema operacional host otimizado para contêineres, como o Bottlerocket, ou um Amazon Machine Image maior, mas ainda minimalista, como o EKS AMIs, é uma prática recomendada manter essas imagens do sistema operacional hospedeiro atualizadas com os patches de segurança mais recentes.

Para otimizar o EKS AMIs, verifique regularmente o canal CHANGELOG e/ou as notas de versão e automatize a implantação de imagens atualizadas de nós de trabalho em seu cluster.

Trate sua infraestrutura como imutável e automatize a substituição de seus nós de trabalho

Em vez de realizar atualizações no local, substitua seus trabalhadores quando um novo patch ou atualização estiver disponível. Isso pode ser abordado de duas maneiras. Você pode adicionar instâncias a um grupo de escalonamento automático existente usando a AMI mais recente ao isolar e drenar os nós sequencialmente até que todos os nós do grupo sejam substituídos pela AMI mais recente. Como alternativa, você pode adicionar instâncias a um novo grupo de nós enquanto isola e drena sequencialmente os nós do grupo de nós antigo até que todos sejam substituídos. Os grupos de nós gerenciados do EKS usam a primeira abordagem e exibirão uma mensagem no console para atualizar seus trabalhadores quando uma nova AMI estiver disponível. `eksctl`também tem um mecanismo para criar grupos de nós com a AMI mais recente e para isolar e drenar facilmente os pods dos grupos de nós antes que as instâncias sejam encerradas. Se você decidir usar um método diferente para substituir seus nós de trabalho, é altamente recomendável automatizar o processo para minimizar a supervisão humana, pois provavelmente precisará substituir trabalhadores regularmente à medida que novos updates/patches forem lançados e quando o plano de controle for atualizado.

Com o EKS Fargate, a AWS atualizará automaticamente a infraestrutura subjacente à medida que as atualizações forem disponibilizadas. Muitas vezes, isso pode ser feito sem problemas, mas pode haver momentos em que uma atualização faça com que seu pod seja reprogramado. Portanto, recomendamos que você crie implantações com várias réplicas ao executar seu aplicativo como um pod Fargate.

Execute periodicamente o kube-bench para verificar a conformidade com os benchmarks do CIS para Kubernetes

O kube-bench é um projeto de código aberto da Aqua que avalia seu cluster em relação aos benchmarks do CIS para Kubernetes. O benchmark descreve as melhores práticas para proteger clusters Kubernetes não gerenciados. O CIS Kubernetes Benchmark abrange o plano de controle e o plano de dados. Como o Amazon EKS fornece um plano de controle totalmente gerenciado, nem todas as recomendações do CIS Kubernetes Benchmark são aplicáveis. Para garantir que esse escopo reflita como o Amazon EKS é implementado, a AWS criou o CIS Amazon EKS Benchmark. O benchmark EKS é herdado do CIS Kubernetes Benchmark com informações adicionais da comunidade com considerações de configuração específicas para clusters EKS.

Ao executar o kube-bench em um cluster EKS, siga estas instruções do Aqua Security. Para obter mais informações, consulte Apresentando o CIS Amazon EKS Benchmark.

Minimize o acesso aos nós de trabalho

Em vez de habilitar o acesso SSH, use o SSM Session Manager quando precisar se conectar remotamente a um host. Ao contrário das chaves SSH que podem ser perdidas, copiadas ou compartilhadas, o Session Manager permite que você controle o acesso às EC2 instâncias usando o IAM. Além disso, ele fornece uma trilha de auditoria e um registro dos comandos que foram executados na instância.

A partir de 19 de agosto de 2020, os grupos de nós gerenciados oferecem suporte a modelos personalizados AMIs e de EC2 lançamento. Isso permite que você incorpore o agente SSM na AMI ou o instale enquanto o nó de trabalho está sendo inicializado. Se você preferir não modificar a AMI otimizada ou o modelo de execução do ASG, você pode instalar o agente SSM com um, DaemonSet como neste exemplo.

Política mínima de IAM para acesso SSH baseado em SSM

A política gerenciada pela `AmazonSSMManagedInstanceCore` AWS contém várias permissões que não são necessárias para o SSM Session Manager /SSM RunCommand se você estiver apenas tentando evitar o acesso ao SSH. Especificamente, são preocupantes as * permissões `ssm:GetParameter(s)` que permitiriam que a função acessasse todos os parâmetros no Parameter Store (inclusive SecureStrings com a chave KMS gerenciada pela AWS configurada).

A política do IAM a seguir contém o conjunto mínimo de permissões para permitir o acesso ao nó por meio do 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": "*"
    }
  ]
}
```

Com essa política em vigor e o plug-in do Gerenciador de Sessões instalado, você pode então executar

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

para acessar o nó.

> **Note**
>
> Você também pode considerar a adição de permissões para ativar o registro no Gerenciador de Sessões.

Implante trabalhadores em sub-redes privadas

Ao implantar trabalhadores em sub-redes privadas, você minimiza a exposição deles à Internet, onde os ataques geralmente se originam. A partir de 22 de abril de 2020, a atribuição de endereços IP públicos aos nós em grupos de nós gerenciados será controlada pela sub-rede na qual eles serão implantados. Antes disso, os nós em um grupo de nós gerenciados recebiam automaticamente um IP público. Se você optar por implantar seus nós de trabalho em sub-redes públicas, implemente regras restritivas de grupos de segurança da AWS para limitar sua exposição.

Execute o Amazon Inspector para avaliar os anfitriões quanto à exposição, vulnerabilidades e desvios das melhores práticas

Você pode usar o Amazon Inspector para verificar se há acesso não intencional à rede aos seus nós e vulnerabilidades nas instâncias subjacentes da Amazon. EC2

O Amazon Inspector pode fornecer dados comuns de vulnerabilidades e exposições (CVE) para suas instâncias da Amazon EC2 somente se o agente do Amazon EC2 Systems Manager (SSM) estiver instalado e habilitado. Esse agente está pré-instalado em várias Amazon Machine Images (AMIs), incluindo Amazon Linux AMIs otimizado para EKS. Independentemente do status do agente SSM, todas as suas EC2 instâncias da Amazon são verificadas em busca de problemas de acessibilidade da rede. Para obter mais informações sobre como configurar escaneamentos para a Amazon EC2, consulte Verificando instâncias da Amazon EC2 .

> **Important**
>
> O Inspector não pode ser executado na infraestrutura usada para executar os pods do Fargate.

Alternativas

Corra SELinux

> **Note**
>
> Disponível no Red Hat Enterprise Linux (RHEL), CentOS, Bottlerocket e Amazon Linux 2023

SELinux fornece uma camada adicional de segurança para manter os contêineres isolados uns dos outros e do host. SELinux permite que os administradores apliquem controles de acesso obrigatórios (MAC) para cada usuário, aplicativo, processo e arquivo. Pense nisso como um backstop que restringe as operações que podem ser executadas a recursos específicos com base em um conjunto de rótulos. No EKS, SELinux pode ser usado para impedir que os contêineres acessem os recursos uns dos outros.

As SELinux políticas de contêiner são definidas no pacote container-selinux. O Docker CE requer esse pacote (junto com suas dependências) para que os processos e arquivos criados pelo Docker (ou outros tempos de execução de contêineres) sejam executados com acesso limitado ao sistema.

Os contêineres utilizam o `container_t` rótulo, que é um alias para`svirt_lxc_net_t`. Essas políticas impedem efetivamente que os contêineres acessem determinados recursos do host.

Quando você configura SELinux para o Docker, o Docker rotula automaticamente as cargas de trabalho `container_t` como um tipo e dá a cada contêiner um nível MCS exclusivo. Isso isolará os contêineres uns dos outros. Se precisar de restrições mais flexíveis, você pode criar seu próprio perfil no SElinux qual concede permissões a um contêiner para áreas específicas do sistema de arquivos. Isso é semelhante ao fato PSPs de que você pode criar perfis diferentes para diferentes contêineres/cápsulas. Por exemplo, você pode ter um perfil para cargas de trabalho gerais com um conjunto de controles restritivos e outro para coisas que exigem acesso privilegiado.

SELinux for Containers tem um conjunto de opções que podem ser configuradas para modificar as restrições padrão. Os seguintes SELinux booleanos podem ser ativados ou desativados de acordo com suas necessidades:

Booleano	Padrão	Description
`container_connect_any`	`off`	Permita que os contêineres acessem portas privilegiadas no host. Por exemplo, se você tiver um contêiner que precise mapear portas para 443 ou 80 no host.
`container_manage_cgroup`	`off`	Permita que os contêineres gerenciem a configuração do cgroup. Por exemplo, um contêiner executando o systemd precisará que isso seja habilitado.
`container_use_cephfs`	`off`	Permita que os contêineres usem um sistema de arquivos ceph.

Por padrão, os contêineres podem read/execute subtrair `/usr` e ler a maior parte do conteúdo`/etc`. Os arquivos abaixo `/var/lib/docker` e `/var/lib/containers` têm o

rótulo`container_var_lib_t`. Para ver uma lista completa dos rótulos padrão, consulte o arquivo 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
```

Os arquivos marcados com `container_file_t` são os únicos arquivos que podem ser gravados por contêineres. Se você quiser que uma montagem de volume seja gravável, você precisará especificar `:z` ou `:Z` no final.

- `:z`rotulará novamente os arquivos para que o contêiner possa ler/gravar
- `:Z`rotulará novamente os arquivos para que somente o contêiner possa ler/gravar

```
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
```

No Kubernetes, a renomeação é um pouco diferente. Em vez de fazer com que o Docker renomeie automaticamente os arquivos, você pode especificar um rótulo MCS personalizado para executar o pod. Os volumes que oferecem suporte à nova rotulagem serão automaticamente renomeados para que possam ser acessados. Os pods com uma etiqueta MCS correspondente poderão acessar o volume. Se você precisar de isolamento estrito, defina um rótulo MCS diferente para cada pod.

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

Neste exemplo, `s0:c144:c154` corresponde a um rótulo MCS atribuído a um arquivo que o contêiner tem permissão para acessar.

No EKS, você pode criar políticas que permitam a execução de contêineres privilegiados, como o FluentD, e criar SELinux uma política para permitir a leitura de /var/log no host sem precisar renomear o diretório do host. Pods com o mesmo rótulo poderão acessar os mesmos volumes do host.

Implementamos uma amostra AMIs para o Amazon EKS que foi SELinux configurada no CentOS 7 e no RHEL 7. Eles AMIs foram desenvolvidos para demonstrar exemplos de implementações que atendem aos requisitos de clientes altamente regulamentados.

> **Warning**
>
> SELinux ignorará os contêineres em que o tipo não está confinado.

Ferramentas e recursos

- SELinuxKubernetes RBAC e políticas de segurança de envio para aplicativos locais
- Fortalecimento iterativo do Kubernetes
- Auditoria 2 Permitir
- SEAlert
- Gerar SELinux políticas para contêineres com o Udica descreve uma ferramenta que analisa os arquivos de especificação do contêiner para recursos, portas e pontos de montagem do Linux e gera um conjunto de SELinux regras que permitem que o contêiner seja executado corretamente
- Manuais do AMI Hardening para fortalecer o sistema operacional para atender a diferentes requisitos regulatórios
- Keiko Upgrade Manager, um projeto de código aberto da Intuit que orquestra a rotação dos nós de trabalho.

- Sysdig Secure
- eksctl

Compliance

A conformidade é uma responsabilidade compartilhada entre a AWS e os consumidores de seus serviços. De um modo geral, a AWS é responsável pela “segurança da nuvem”, enquanto seus usuários são responsáveis pela “segurança na nuvem”. A linha que define o que a AWS e seus usuários são responsáveis variará de acordo com o serviço. Por exemplo, com o Fargate, a AWS é responsável por gerenciar a segurança física de seus datacenters, do hardware, da infraestrutura virtual (Amazon EC2) e do tempo de execução do contêiner (Docker). Os usuários do Fargate são responsáveis por proteger a imagem do contêiner e seu aplicativo. Saber quem é responsável pelo quê é uma consideração importante ao executar cargas de trabalho que devem seguir os padrões de conformidade.

A tabela a seguir mostra os programas de conformidade com os quais os diferentes serviços de contêiner estão em conformidade.

Programa de conformidade	Orquestrador Amazon ECS	Orquestrador Amazon EKS	ECS Fargate	Amazon ECR
PCI DSS de Nível 1.	1	1	1	1
Elegível para HIPAA	1	1	1	1
MEIA É	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

Programa de conformidade	Orquestrador Amazon ECS	Orquestrador Amazon EKS	ECS Fargate	Amazon ECR
IRAP	1	1	1	1
FedRAMP Moderado (Leste/Oeste)	1	1	0	1
FedRAMP High () GovCloud	1	1	0	1
DOD CC SRG	1	Avaliação da DISA () IL5	0	1
HIPAA BAA	1	1	1	1
MTCS	1	1	0	1
C5	1	1	0	1
K-ISMS	1	1	0	1
ENS High	1	1	0	1
OSPAR	1	1	0	1
HITRUST CSF	1	1	1	1

O status de conformidade muda com o tempo. Para obter o status mais recente, sempre consulte https://aws.amazon.com/compliance/ services-in-scope /.

Para obter mais informações sobre modelos de credenciamento de nuvem e melhores práticas, consulte o whitepaper da AWS, Modelos de credenciamento para adoção segura da nuvem

Deslocando para a esquerda

O conceito de mudar para a esquerda envolve detectar violações e erros de políticas no início do ciclo de vida do desenvolvimento de software. Do ponto de vista da segurança, isso pode ser muito benéfico. Um desenvolvedor, por exemplo, pode corrigir problemas com sua configuração antes que

o aplicativo seja implantado no cluster. Detectar erros como esse mais cedo ajudará a impedir que configurações que violem suas políticas sejam implantadas.

Política como código

A política pode ser considerada como um conjunto de regras para governar comportamentos, ou seja, comportamentos permitidos ou proibidos. Por exemplo, você pode ter uma política que diz que todos os Dockerfiles devem incluir uma diretiva USER que faça com que o contêiner seja executado como um usuário não root. Como documento, uma política como essa pode ser difícil de descobrir e aplicar. Também pode ficar desatualizado à medida que seus requisitos mudam. Com as soluções Policy as Code (PaC), você pode automatizar os controles de segurança, conformidade e privacidade que detectam, previnem, reduzem e neutralizam ameaças conhecidas e persistentes. Além disso, eles fornecem um mecanismo para codificar suas políticas e gerenciá-las da mesma forma que você faz com outros artefatos de código. A vantagem dessa abordagem é que você pode reutilizar suas GitOps estratégias DevOps e estratégias para gerenciar e aplicar políticas de forma consistente em frotas de clusters Kubernetes. Consulte o Pod Security para obter informações sobre as opções do PaC e o futuro do PSPs.

Use policy-as-code ferramentas em pipelines para detectar violações antes da implantação

- O OPA é um mecanismo de políticas de código aberto que faz parte do CNCF. Ele é usado para tomar decisões políticas e pode ser executado de várias maneiras diferentes, por exemplo, como uma biblioteca de idiomas ou um serviço. As políticas de OPA são escritas em uma linguagem específica de domínio (DSL) chamada Rego. Embora geralmente seja executado como parte de um Kubernetes Dynamic Admission Controller como o projeto Gatekeeper, o OPA também pode ser incorporado ao seu pipeline. CI/CD Isso permite que os desenvolvedores obtenham feedback sobre sua configuração no início do ciclo de lançamento, o que pode posteriormente ajudá-los a resolver problemas antes de começarem a produção. Uma coleção de políticas comuns de OPA pode ser encontrada no GitHub repositório deste projeto.
- O Conftest é construído com base no OPA e fornece uma experiência focada no desenvolvedor para testar a configuração do Kubernetes.
- O Kyverno é um mecanismo de políticas projetado para o Kubernetes. Com o Kyverno, as políticas são gerenciadas como recursos do Kubernetes e nenhuma nova linguagem é necessária para criar políticas. Isso permite usar ferramentas conhecidas, como kubectl, git e kustomize, para gerenciar políticas. As políticas da Kyverno podem validar, alterar e gerar recursos do Kubernetes, além de garantir a segurança da cadeia de suprimentos de imagens da OCI. A CLI do Kyverno

pode ser usada para testar políticas e validar recursos como parte de um pipeline. CI/CD Todas as políticas da comunidade Kyverno podem ser encontradas no site da Kyverno e, para exemplos de como usar a CLI do Kyverno para escrever testes em pipelines, consulte o repositório de políticas.

Ferramentas e recursos

- Workshop de imersão em segurança do Amazon EKS — Conformidade regulatória
- banco de cubos
- docker-bench-security
- Inspector da AWS
- Análise de segurança do Kubernetes Uma avaliação de segurança terceirizada do Kubernetes 1.13.4 (2019)
- NeuVector da SUSE, plataforma de segurança de contêineres de código aberto e de confiança zero, fornece relatórios de conformidade e verificações de conformidade personalizadas

Resposta a incidentes e análise forense

Sua capacidade de reagir rapidamente a um incidente pode ajudar a minimizar os danos causados por uma violação. Ter um sistema de alerta confiável que possa alertá-lo sobre comportamentos suspeitos é a primeira etapa de um bom plano de resposta a incidentes. Quando ocorre um incidente, você precisa decidir rapidamente se deseja destruir e substituir o contêiner afetado ou isolar e inspecionar o contêiner. Se você optar por isolar o contêiner como parte de uma investigação forense e análise da causa raiz, o seguinte conjunto de atividades deve ser seguido:

Exemplo de plano de resposta a incidentes

Identifique o pod e o nó de trabalho ofensivos

Seu primeiro curso de ação deve ser isolar o dano. Comece identificando onde a violação ocorreu e isole esse pod e seu nó do resto da infraestrutura.

Identifique os pods e os nós de trabalho ofensivos usando o nome da carga de trabalho

Se você souber o nome e o namespace do pod ofensivo, você pode identificar o nó de trabalho que executa o pod da seguinte forma:

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

Se um recurso de carga de trabalho, como uma implantação, tiver sido comprometido, é provável que todos os pods que fazem parte do recurso de carga de trabalho estejam comprometidos. Use o comando a seguir para listar todos os pods do recurso de carga de trabalho e os nós em que eles estão sendo executados:

```
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)"'
```

O comando acima é para implantações. Você pode executar o mesmo comando para outros recursos de carga de trabalho, como replicasets, statefulsets etc.

Identifique os pods e os nós de trabalho ofensivos usando o nome da conta de serviço

Em alguns casos, você pode identificar que uma conta de serviço está comprometida. É provável que os pods que usam a conta de serviço identificada estejam comprometidos. Você pode identificar todos os pods usando a conta de serviço e os nós em que eles estão sendo executados com o seguinte comando:

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

Identifique pods com imagens e nós de trabalho vulneráveis ou comprometidos

Em alguns casos, você pode descobrir que uma imagem de contêiner usada em pods no seu cluster é maliciosa ou está comprometida. Uma imagem de contêiner é maliciosa ou comprometida, se for descoberto que contém malware, é uma imagem incorreta conhecida ou tem um CVE que foi explorado. Você deve considerar comprometidos todos os pods que usam a imagem do contêiner. Você pode identificar os pods usando a imagem e os nós em que eles estão sendo executados com o seguinte 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)"'
```

Isole o pod criando uma política de rede que negue todo o tráfego de entrada e saída para o pod

Uma regra de negar todo o tráfego pode ajudar a interromper um ataque que já está em andamento, cortando todas as conexões com o pod. A política de rede a seguir se aplicará a um pod com o rótulo`app=web`.

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

> **Important**
>
> Uma política de rede pode se mostrar ineficaz se um invasor tiver acesso ao host subjacente. Se você suspeitar que isso aconteceu, você pode usar os grupos de segurança da AWS para isolar um host comprometido de outros hosts. Ao alterar o grupo de segurança de um host, saiba que isso afetará todos os contêineres em execução nesse host.

Revogue as credenciais de segurança temporárias atribuídas ao pod ou ao nó de trabalho, se necessário

Se o node de trabalho tiver recebido uma função do IAM que permite que os pods tenham acesso a outros recursos da AWS, remova essas funções da instância para evitar mais danos causados pelo ataque. Da mesma forma, se o pod tiver recebido uma função do IAM, avalie se você pode remover com segurança as políticas do IAM da função sem afetar outras workloads.

Isole o nó do trabalhador

Ao isolar o nó de trabalho afetado, você está informando o agendador para evitar o agendamento de pods no nó afetado. Isso permitirá que você remova o nó para estudo forense sem interromper outras cargas de trabalho.

> **Note**
>
> Essa orientação não se aplica ao Fargate, onde cada pod do Fargate é executado em seu próprio ambiente de sandbox. Em vez de isolar, sequestre os pods Fargate afetados aplicando uma política de rede que nega todo o tráfego de entrada e saída.

Ative a proteção contra rescisão no nó de trabalho afetado

Um atacante pode tentar apagar seus delitos encerrando um nó afetado. Ativar a proteção contra rescisão pode impedir que isso aconteça. A proteção de escalabilidade da instância protegerá o nó de um evento de expansão.

> **Warning**
>
> Você não pode ativar a proteção contra encerramento em uma instância spot.

Identifique a ofensa Pod/Node com uma etiqueta indicando que ela faz parte de uma investigação ativa

Isso servirá como um aviso para os administradores do cluster não adulterarem os afetados Pods/Nodes até que a investigação seja concluída.

Capture artefatos voláteis no nó de trabalho

- Capture a memória do sistema operacional. Isso capturará o daemon do Docker (ou outro tempo de execução do contêiner) e seus subprocessos por contêiner. Isso pode ser feito usando ferramentas como LiME e Volatility, ou por meio de ferramentas de alto nível, como o Automated Forensics Orchestrator for Amazon EC2, que se baseiam nelas.
- Execute um despejo em árvore netstat dos processos em execução e das portas abertas. Isso capturará o daemon do docker e seu subprocesso por contêiner.

- Execute comandos para salvar o estado no nível do contêiner antes que as evidências sejam alteradas. Você pode usar os recursos do tempo de execução do contêiner para capturar informações sobre os contêineres em execução no momento. Por exemplo, com o Containerd, você pode fazer o seguinte:
  - `crictl ps`para processos em execução.
  - `crictl logs CONTAINER`para registros mantidos em nível de daemon.

    O mesmo poderia ser obtido com containerd usando a CLI nerdctl, no lugar de (por exemplo). `docker` `nerdctl inspect` Alguns comandos adicionais estão disponíveis dependendo do tempo de execução do contêiner. Por exemplo, o Docker `docker diff` precisa ver as alterações no sistema de arquivos do contêiner ou salvar todo o estado do contêiner, incluindo `docker checkpoint` a memória volátil (RAM). Veja esta postagem no blog do Kubernetes para discutir recursos semelhantes com ambientes de execução containerd ou CRI-O.
- Pausa o contêiner para captura forense.
- Faça um snapshot dos volumes do EBS da instância.

Reimplante o pod ou o recurso de carga de trabalho comprometido

Depois de coletar dados para análise forense, você pode reimplantar o pod comprometido ou o recurso de carga de trabalho.

Primeiro, implante a correção para a vulnerabilidade que foi comprometida e inicie novos pods de substituição. Em seguida, exclua os pods vulneráveis.

Se os pods vulneráveis forem gerenciados por um recurso de carga de trabalho de alto nível do Kubernetes (por exemplo, um Deployment ou DaemonSet), excluí-los agendará novos. Portanto, os pods vulneráveis serão lançados novamente. Nesse caso, você deve implantar um novo recurso de carga de trabalho substituto após corrigir a vulnerabilidade. Em seguida, você deve excluir a carga de trabalho vulnerável.

Recomendações

Leia o whitepaper de resposta a incidentes de segurança da AWS

Embora esta seção forneça uma breve visão geral e algumas recomendações para lidar com suspeitas de violações de segurança, o tópico é abordado exaustivamente no white paper AWS Security Incident Response.

Pratique dias de jogos de segurança

Divida seus profissionais de segurança em duas equipes: vermelha e azul. A equipe vermelha se concentrará em investigar diferentes sistemas em busca de vulnerabilidades, enquanto a equipe azul será responsável por se defender contra elas. Se você não tiver profissionais de segurança suficientes para criar equipes separadas, considere contratar uma entidade externa que tenha conhecimento das explorações do Kubernetes.

O Kubesploit é uma estrutura de teste de penetração CyberArk que você pode usar para conduzir dias de jogo. Ao contrário de outras ferramentas que examinam seu cluster em busca de vulnerabilidades, o kubesploit simula um ataque no mundo real. Isso dá à sua equipe azul a oportunidade de praticar sua resposta a um ataque e avaliar sua eficácia.

Execute testes de penetração em seu cluster

Atacar periodicamente seu próprio cluster pode ajudá-lo a descobrir vulnerabilidades e configurações incorretas. Antes de começar, siga as diretrizes do teste de penetração antes de realizar um teste em seu cluster.

Ferramentas e recursos

- kube-hunter, uma ferramenta de teste de penetração para Kubernetes.
- Gremlin, um kit de ferramentas de engenharia de caos que você pode usar para simular ataques contra seus aplicativos e infraestrutura.
- Atacando e defendendo instalações do Kubernetes
- kubesploit
- NeuVector da SUSE, a plataforma de segurança de contêineres de código aberto e de confiança zero fornece relatórios de vulnerabilidade e risco, bem como notificação de eventos de segurança
- Ameaças persistentes avançadas
- Ataque e defesa práticos do Kubernetes
- Comprometendo o cluster Kubernetes explorando as permissões do RBAC

Segurança de imagem

Você deve considerar a imagem do contêiner como sua primeira linha de defesa contra um ataque. Uma imagem insegura e mal construída pode permitir que um invasor escape dos limites do

contêiner e tenha acesso ao host. Uma vez no host, um invasor pode obter acesso a informações confidenciais ou se mover lateralmente dentro do cluster ou com sua conta da AWS. As melhores práticas a seguir ajudarão a mitigar o risco de isso acontecer.

Recomendações

Crie imagens mínimas

Comece removendo todos os binários estranhos da imagem do contêiner. Se você estiver usando uma imagem desconhecida do Dockerhub, inspecione a imagem usando um aplicativo como o Dive, que pode mostrar o conteúdo de cada uma das camadas do contêiner. Remova todos os binários com os bits SETUID e SETGID, pois eles podem ser usados para aumentar os privilégios e considere a remoção de todos os shells e utilitários, como nc e curl, que podem ser usados para fins nefastos. Você pode encontrar os arquivos com os bits SETUID e SETGID com o seguinte comando:

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

Para remover as permissões especiais desses arquivos, adicione a seguinte diretiva à imagem do contêiner:

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

Coloquialmente, isso é conhecido como desobstruir sua imagem.

Use compilações de vários estágios

Usar compilações de vários estágios é uma forma de criar imagens mínimas. Muitas vezes, compilações de vários estágios são usadas para automatizar partes do ciclo de integração contínua. Por exemplo, compilações de vários estágios podem ser usadas para lintar seu código-fonte ou realizar análises estáticas de código. Isso dá aos desenvolvedores a oportunidade de obter feedback quase imediato, em vez de esperar pela execução de um pipeline. As compilações de vários estágios são atraentes do ponto de vista da segurança porque permitem minimizar o tamanho da imagem final enviada ao seu registro de contêiner. Imagens de contêiner sem ferramentas de criação e outros binários estranhos melhoram sua postura de segurança ao reduzir a superfície de ataque da imagem. Para obter informações adicionais sobre compilações em vários estágios, consulte a documentação de compilações em vários estágios do Docker.

Crie uma lista de materiais de software (SBOMs) para sua imagem de contêiner

Uma “lista de materiais de software” (SBOM) é um inventário aninhado dos artefatos de software que compõem a imagem do seu contêiner. O SBOM é um alicerce fundamental na segurança de software e no gerenciamento de riscos da cadeia de suprimentos de software. A geração e o armazenamento de SBOMS em um repositório central e a verificação SBOMs de vulnerabilidades ajudam a resolver as seguintes preocupações:

- Visibilidade: entenda quais componentes compõem a imagem do seu contêiner. O armazenamento em um repositório central permite que você seja auditado e verificado SBOMs a qualquer momento, mesmo após a implantação, para detectar e responder a novas vulnerabilidades, como vulnerabilidades de dia zero.
- Verificação de proveniência: garantia de que as suposições existentes de onde e como um artefato se origina são verdadeiras e de que o artefato ou os metadados que o acompanham não foram adulterados durante os processos de construção ou entrega.
- Confiabilidade: garantia de que um determinado artefato e seu conteúdo podem ser confiáveis para fazer o que se pretende fazer, ou seja, é adequado para uma finalidade. Isso envolve avaliar se o código é seguro de executar e tomar decisões informadas sobre os riscos associados à execução do código. A confiabilidade é garantida pela criação de um relatório atestado de execução do pipeline junto com o SBOM atestado e o relatório de digitalização CVE atestado para garantir aos consumidores da imagem que essa imagem foi de fato criada por meios seguros (pipeline) com componentes seguros.
- Verificação de confiança de dependência: verificação recursiva da árvore de dependências de um artefato quanto à confiabilidade e proveniência dos artefatos que ele usa. O Drift in SBOMs pode ajudar a detectar atividades maliciosas, incluindo dependências não autorizadas e não confiáveis e tentativas de infiltração.

As seguintes ferramentas podem ser usadas para gerar o SBOM:

- O Amazon Inspector pode ser usado para criar e exportar. SBOMs
- O Syft da Anchore também pode ser usado para geração de SBOM. Para varreduras de vulnerabilidade mais rápidas, o SBOM gerado para uma imagem de contêiner pode ser usado como uma entrada para escanear. O SBOM e o relatório de digitalização são então atestados e anexados à imagem antes de enviá-la para um repositório central da OCI, como o Amazon ECR, para fins de revisão e auditoria.

Saiba mais sobre como proteger sua cadeia de suprimentos de software revisando o guia de melhores práticas da cadeia de suprimentos de software CNCF.

Examine imagens em busca de vulnerabilidades regularmente

Assim como suas contrapartes de máquinas virtuais, as imagens de contêiner podem conter binários e bibliotecas de aplicativos com vulnerabilidades ou desenvolver vulnerabilidades ao longo do tempo. A melhor maneira de se proteger contra ataques é verificar regularmente suas imagens com um verificador de imagens. As imagens armazenadas no Amazon ECR podem ser digitalizadas por push ou sob demanda (uma vez durante um período de 24 horas). Atualmente, o ECR suporta dois tipos de escaneamento: básico e aprimorado. A digitalização básica utiliza a Clair, uma solução de digitalização de imagens de código aberto, sem nenhum custo. O escaneamento aprimorado usa o Amazon Inspector para fornecer escaneamentos contínuos automáticos por um custo adicional. Depois que uma imagem é digitalizada, os resultados são registrados no fluxo de eventos para o ECR in. EventBridge Você também pode ver os resultados de uma verificação no console do ECR. Imagens com uma vulnerabilidade ALTA ou CRÍTICA devem ser excluídas ou reconstruídas. Se uma imagem que foi implantada desenvolver uma vulnerabilidade, ela deverá ser substituída assim que possível.

Saber onde as imagens com vulnerabilidades foram implantadas é essencial para manter seu ambiente seguro. Embora você mesmo possa criar uma solução de rastreamento de imagens, já existem várias ofertas comerciais que fornecem esse e outros recursos avançados prontos para uso, incluindo:

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

Um webhook de validação do Kubernetes também pode ser usado para validar se as imagens estão livres de vulnerabilidades críticas. Os webhooks de validação são invocados antes da API Kubernetes. Eles geralmente são usados para rejeitar solicitações que não estão em conformidade com os critérios de validação definidos no webhook. Esse é um exemplo de um webhook sem servidor que chama a API ECR describeImageScan Findings para determinar se um pod está

obtendo uma imagem com vulnerabilidades críticas. Se forem encontradas vulnerabilidades, o pod será rejeitado e uma mensagem com a lista de será CVEs retornada como um evento.

Use atestados para validar a integridade do artefato

Um atestado é uma “declaração” assinada criptograficamente que afirma que algo - um “predicado”, por exemplo, uma execução de pipeline ou o SBOM ou o relatório de verificação de vulnerabilidade é verdadeiro sobre outra coisa - um “assunto”, ou seja, a imagem do contêiner.

Os atestados ajudam os usuários a validar se um artefato vem de uma fonte confiável na cadeia de suprimentos de software. Como exemplo, podemos usar uma imagem de contêiner sem conhecer todos os componentes ou dependências de software incluídos nessa imagem. No entanto, se confiarmos em tudo o que o produtor da imagem do contêiner diz sobre qual software está presente, podemos usar o atestado do produtor para confiar nesse artefato. Isso significa que podemos continuar usando o artefato com segurança em nosso fluxo de trabalho, em vez de termos feito a análise sozinhos.

- Os atestados podem ser criados usando o AWS Signer ou o Sigstore cosign.
- Os controladores de admissão do Kubernetes, como o Kyverno, podem ser usados para verificar os atestados.
- Consulte este workshop para saber mais sobre as melhores práticas de gerenciamento da cadeia de suprimentos de software na AWS usando ferramentas de código aberto com tópicos como criar e anexar atestados a uma imagem de contêiner.

Crie políticas do IAM para repositórios ECR

Atualmente, não é incomum que uma organização tenha várias equipes de desenvolvimento operando de forma independente em uma conta compartilhada da AWS. Se essas equipes não precisarem compartilhar ativos, talvez você queira criar um conjunto de políticas do IAM que restrinjam o acesso aos repositórios com os quais cada equipe pode interagir. Uma boa maneira de implementar isso é usando namespaces ECR. Os namespaces são uma forma de agrupar repositórios semelhantes. Por exemplo, todos os registros da equipe A podem ser prefaciados com a equipe a/, enquanto os da equipe B podem usar o prefixo equipe-b/. A política para restringir o acesso pode ter a seguinte aparência:

```
{
  "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 o uso de endpoints privados ECR

A API ECR tem um endpoint público. Consequentemente, os registros ECR podem ser acessados pela Internet, desde que a solicitação tenha sido autenticada e autorizada pelo IAM. Para aqueles que precisam operar em um ambiente de área restrita em que o cluster VPC não tem um Internet Gateway (IGW), você pode configurar um endpoint privado para ECR. A criação de um endpoint privado permite que você acesse de forma privada a API ECR por meio de um endereço IP privado, em vez de rotear o tráfego pela Internet. Para obter informações adicionais sobre esse tópico, consulte VPC endpoints da interface Amazon ECR.

Implemente políticas de endpoint para ECR

A política de endpoint padrão permite o acesso a todos os repositórios ECR em uma região. Isso pode permitir que um attacker/insider exfiltre dados empacotando-os como uma imagem de contêiner e enviando-os para um registro em outra conta da AWS. A mitigação desse risco envolve a criação de uma política de endpoint que limita o acesso da API aos repositórios ECR. Por exemplo, a política a seguir permite que todos os princípios da AWS em sua conta executem todas as ações em seus e somente em seus repositórios ECR:

```
{
  "Statement": [
```

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

Você pode aprimorar isso ainda mais definindo uma condição que use o novo `PrincipalOrgID` atributo, que evitará imagens pushing/pulling de acordo com um princípio do IAM que não faz parte da sua organização da AWS. Consulte, aws: PrincipalOrg ID para obter detalhes adicionais. Recomendamos aplicar a mesma política a ambos os endpoints `com.amazonaws.<region>.ecr.dkr` e `com.amazonaws.<region>.ecr.api`. Como o EKS extrai imagens para kube-proxy, coredns e aws-node do ECR, você precisará adicionar o ID da conta do registro, por exemplo, à lista de recursos na política de endpoint ou alterar a política `602401143452.dkr.ecr.us-west-2.amazonaws.com/` para permitir retiradas e restringir envios ao ID da sua conta. A tabela abaixo revela o mapeamento entre as contas da AWS de onde as imagens EKS são vendidas e a região do cluster.

Número da conta	Região
602401143452	Todas as regiões comerciais, exceto as listadas abaixo
—	—
800184023465	ap-east-1 - Ásia-Pacífico (Hong Kong)
558608220178	me-south-1 - Oriente Médio (Bahrein)
918309763551	cn-north-1 - China (Pequim)
961992271922	cn-northwest-1 - China (Ningxia)

Para obter mais informações sobre o uso de políticas de endpoint, consulte Usando políticas de endpoint VPC para controlar o acesso ao Amazon ECR.

Implemente políticas de ciclo de vida para ECR

O Guia de Segurança de Contêineres de Aplicativos do NIST alerta sobre o risco de “imagens obsoletas nos registros”, observando que, com o tempo, imagens antigas e vulneráveis, os pacotes de out-of-date software devem ser removidos para evitar implantação e exposição acidentais. Cada repositório ECR pode ter uma política de ciclo de vida que define regras para quando as imagens expiram. A documentação oficial da AWS descreve como configurar regras de teste, avaliá-las e depois aplicá-las. Há vários exemplos de políticas de ciclo de vida nos documentos oficiais que mostram maneiras diferentes de filtrar as imagens em um repositório:

- Filtragem por idade ou número de imagens
- Filtrando por imagens marcadas ou não marcadas
- Filtragem por tags de imagem, em várias regras ou em uma única regra

??? + aviso Se a imagem de um aplicativo de longa execução for removida do ECR, isso poderá causar erros de extração de imagem quando o aplicativo for reimplantado ou dimensionado horizontalmente. Ao usar políticas de ciclo de vida de imagens, certifique-se de ter boas CI/CD práticas para manter as implantações e as imagens às quais elas fazem referência atualizadas e sempre criar regras de expiração de [imagem] que considerem a frequência com que você faz lançamentos/implantações.

Criar um conjunto de imagens selecionadas

Em vez de permitir que os desenvolvedores criem suas próprias imagens, considere criar um conjunto de imagens verificadas para as diferentes pilhas de aplicativos em sua organização. Ao fazer isso, os desenvolvedores podem prescindir de aprender a compor Dockerfiles e se concentrar em escrever código. À medida que as alterações são mescladas no Master, um CI/CD pipeline pode compilar automaticamente o ativo, armazená-lo em um repositório de artefatos e copiar o artefato na imagem apropriada antes de enviá-lo para um registro do Docker, como o ECR. No mínimo, você deve criar um conjunto de imagens básicas a partir das quais os desenvolvedores criem seus próprios Dockerfiles. Idealmente, você deve evitar extrair imagens do Dockerhub porque 1/você nem sempre sabe o que está na imagem e 2/cerca de um quinto das 1000 imagens principais têm vulnerabilidades. Uma lista dessas imagens e suas vulnerabilidades pode ser encontrada aqui.

Adicione a diretiva USER aos seus Dockerfiles para ser executada como um usuário não root

Conforme mencionado na seção de segurança do pod, você deve evitar executar o contêiner como root. Embora você possa configurar isso como parte do PodSpec, é um bom hábito usar a `USER` diretiva em seus Dockerfiles. A `USER` diretiva define o UID a ser usado durante a execução `RUNENTRYPOINT`, ou a `CMD` instrução que aparece após a diretiva USER.

Lint seus Dockerfiles

O Linting pode ser usado para verificar se seus Dockerfiles estão aderindo a um conjunto de diretrizes predefinidas, por exemplo, a inclusão da `USER` diretiva ou a exigência de que todas as imagens sejam marcadas. dockerfile_lint é um projeto de código aberto RedHat que verifica as melhores práticas comuns e inclui um mecanismo de regras que você pode usar para criar suas próprias regras para linting de Dockerfiles. Ele pode ser incorporado a um pipeline de CI, pois compilações com Dockerfiles que violam uma regra falharão automaticamente.

Crie imagens do zero

Reduzir a superfície de ataque das imagens do contêiner deve ser o objetivo principal ao criar imagens. A maneira ideal de fazer isso é criando imagens mínimas sem binários que possam ser usadas para explorar vulnerabilidades. Felizmente, o Docker tem um mecanismo a partir do `scratch`qual criar imagens. Com linguagens como Go, você pode criar um binário vinculado estático e referenciá-lo em seu Dockerfile, como neste exemplo:

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

Isso cria uma imagem de contêiner que consiste em seu aplicativo e nada mais, tornando-o extremamente seguro.

Use tags imutáveis com ECR

As tags imutáveis forçam você a atualizar a tag da imagem em cada envio para o repositório de imagens. Isso pode impedir que um invasor substitua uma imagem por uma versão maliciosa sem alterar as tags da imagem. Além disso, oferece uma maneira de identificar uma imagem de forma fácil e exclusiva.

Assine suas imagens SBOMs, execuções de pipeline e relatórios de vulnerabilidade

Quando o Docker foi introduzido pela primeira vez, não havia um modelo criptográfico para verificar imagens de contêineres. Com a v2, o Docker adicionou resumos ao manifesto da imagem. Isso permitiu que a configuração de uma imagem fosse codificada e que o hash fosse usado para gerar um ID para a imagem. Quando a assinatura de imagem está ativada, o mecanismo Docker verifica a assinatura do manifesto, garantindo que o conteúdo tenha sido produzido de uma fonte confiável e que nenhuma violação tenha ocorrido. Depois que cada camada é baixada, o mecanismo verifica o resumo da camada, garantindo que o conteúdo corresponda ao conteúdo especificado no manifesto. A assinatura de imagens permite que você crie efetivamente uma cadeia de suprimentos segura, por meio da verificação das assinaturas digitais associadas à imagem.

Podemos usar o AWS Signer ou o Sigstore Cosign para assinar imagens de contêineres, criar atestados, relatórios de verificação de vulnerabilidade e relatórios de SBOMs execução de pipeline. Esses atestados garantem a confiabilidade e a integridade da imagem, que ela é de fato criada pelo pipeline confiável sem qualquer interferência ou adulteração e que contém somente os componentes de software documentados (no SBOM) que são verificados e confiáveis pelo editor da imagem. Esses atestados podem ser anexados à imagem do contêiner e enviados ao repositório.

Na próxima seção, veremos como usar os artefatos atestados para auditorias e verificação do controlador de admissões.

Verificação da integridade da imagem usando o controlador de admissão Kubernetes

Podemos verificar assinaturas de imagens e artefatos atestados de forma automatizada antes de implantar a imagem no cluster Kubernetes de destino usando o controlador de admissão dinâmico

e admitir implantações somente quando os metadados de segurança dos artefatos estiverem em conformidade com as políticas do controlador de admissão.

Por exemplo, podemos escrever uma política que verifica criptograficamente a assinatura de uma imagem, um SBOM atestado, um relatório de execução de pipeline atestado ou um relatório de verificação CVE atestado. Podemos escrever condições na política para verificar os dados no relatório, por exemplo, um escaneamento CVE não deve ter nenhum aspecto críticoCVEs. A implantação é permitida somente para imagens que satisfaçam essas condições e todas as outras implantações serão rejeitadas pelo controlador de admissões.

Exemplos de controlador de admissão incluem:

- Kyverno
- Guardião da OPA
- Portieris
- Ratificar
- Kritis
- Tutorial de grafeas
- Voucher

Atualize os pacotes nas imagens do seu contêiner

Você deve incluir RUN `apt-get update && apt-get upgrade` em seus Dockerfiles para atualizar os pacotes em suas imagens. Embora a atualização exija que você seja executada como root, isso ocorre durante a fase de criação da imagem. O aplicativo não precisa ser executado como root. Você pode instalar as atualizações e depois mudar para um usuário diferente com a diretiva USER. Se sua imagem base for executada como um usuário não root, alterne para root e vice-versa; não confie apenas nos mantenedores da imagem base para instalar as atualizações de segurança mais recentes.

Execute `apt-get clean` para excluir os arquivos do instalador do`/var/cache/apt/archives/`. Você também pode executar `rm -rf /var/lib/apt/lists/*` depois de instalar os pacotes. Isso remove os arquivos de índice ou as listas de pacotes que estão disponíveis para instalação. Esteja ciente de que esses comandos podem ser diferentes para cada gerenciador de pacotes. Por exemplo:

```
RUN apt-get update && apt-get install -y \
```

```
    curl \
    git \
    libsqlite3-dev \
    && apt-get clean && rm -rf /var/lib/apt/lists/*
```

Ferramentas e recursos

- Workshop de imersão em segurança do Amazon EKS — Segurança de imagem
- docker-slim Crie imagens mínimas seguras
- dockle Verifica se seu Dockerfile está alinhado com as melhores práticas para criar imagens seguras
- dockerfile-lint Linter baseado em regras para Dockerfiles
- hadolint Um linter inteligente de dockerfile
- Gatekeeper e OPA Um controlador de admissão baseado em políticas
- Kyverno Um mecanismo de políticas nativo do Kubernetes
- in-toto Permite que o usuário verifique se uma etapa na cadeia de suprimentos foi planejada para ser executada e se a etapa foi executada pelo ator certo
- Notário Um projeto para assinar imagens de contêineres
- Notário v2
- Grafeas Uma API aberta de metadados de artefatos para auditar e governar sua cadeia de suprimentos de software
- NeuVector da SUSE, a plataforma de segurança de contêineres de confiança zero, de código aberto, fornece varredura de contêineres, imagens e registros em busca de vulnerabilidades, segredos e conformidade.

Estratégia de várias contas

A AWS recomenda usar uma estratégia de várias contas e organizações da AWS para ajudar a isolar e gerenciar seus aplicativos e dados comerciais. Há muitos benefícios em usar uma estratégia de várias contas:

- Aumento das cotas de serviços de API da AWS. As cotas são aplicadas às contas da AWS, e o uso de várias contas para suas cargas de trabalho aumenta a cota geral disponível para suas cargas de trabalho.

- Políticas mais simples de Identity and Access Management (IAM). Conceder às cargas de trabalho e aos operadores que as apoiam acesso somente às suas próprias contas da AWS significa menos tempo criando políticas de IAM refinadas para alcançar o princípio do menor privilégio.
- Isolamento aprimorado dos recursos da AWS. Por padrão, todos os recursos provisionados em uma conta são logicamente isolados dos recursos provisionados em outras contas. Esse limite de isolamento fornece uma maneira de limitar os riscos de um problema relacionado à aplicação, a uma configuração incorreta ou a ações mal-intencionadas. Se ocorrer um problema em uma conta, os impactos nas workloads contidas em outras contas poderão ser reduzidos ou eliminados.
- Mais benefícios, conforme descrito no whitepaper da AWS Multi Account Strategy

As seções a seguir explicarão como implementar uma estratégia de várias contas para suas cargas de trabalho do EKS usando uma abordagem de cluster EKS centralizada ou descentralizada.

Planejando uma estratégia de conta com várias cargas de trabalho para clusters com vários inquilinos

Em uma estratégia multiconta da AWS, os recursos que pertencem a uma determinada carga de trabalho, como buckets S3, clusters ElastiCache e tabelas do DynamoDB, são todos criados em uma conta da AWS que contém todos os recursos dessa carga de trabalho. Elas são chamadas de conta de carga de trabalho, e o cluster EKS é implantado em uma conta chamada conta de cluster. As contas de cluster serão exploradas na próxima seção. A implantação de recursos em uma conta de carga de trabalho dedicada é semelhante à implantação de recursos do kubernetes em um namespace dedicado.

As contas de carga de trabalho podem então ser mais detalhadas por ciclo de vida de desenvolvimento de software ou outros requisitos, se apropriado. Por exemplo, uma determinada carga de trabalho pode ter uma conta de produção, uma conta de desenvolvimento ou contas para hospedar instâncias dessa carga de trabalho em uma região específica. Mais informações estão disponíveis neste whitepaper da AWS.

Você pode adotar as seguintes abordagens ao implementar a estratégia EKS Multi Account:

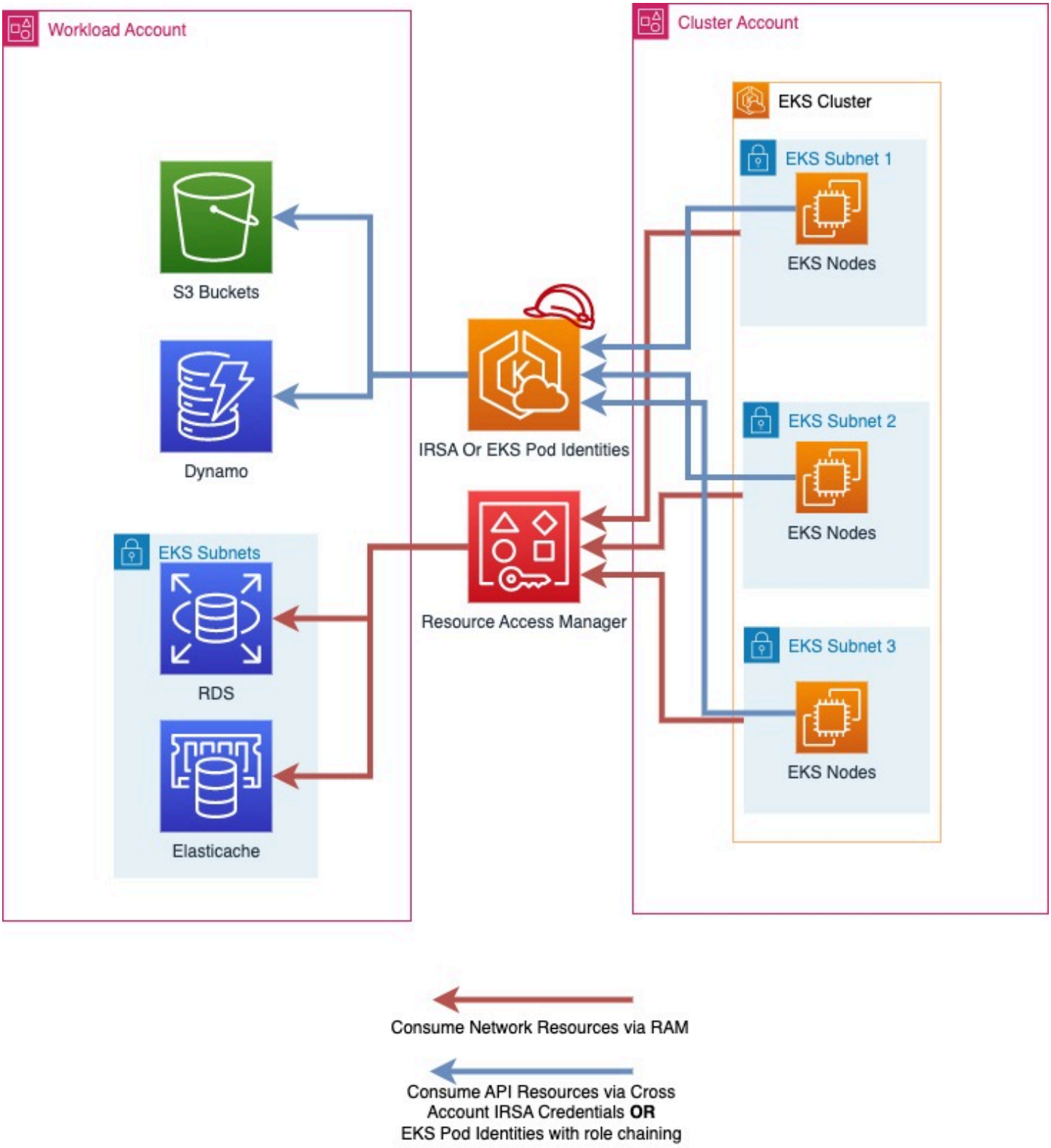
Cluster EKS centralizado

Nessa abordagem, seu cluster EKS será implantado em uma única conta da AWS chamada de`Cluster Account`. Usando as funções do IAM para contas de serviço (IRSA) ou EKS Pod Identities para fornecer credenciais temporárias da AWS e o AWS Resource Access Manager

(RAM) para simplificar o acesso à rede, você pode adotar uma estratégia de várias contas para seu cluster EKS multilocatário. A conta do cluster conterá a VPC, as sub-redes, o cluster EKS, os recursos computacionais EC2 /Fargate (nós de trabalho) e todas as configurações de rede adicionais necessárias para executar seu cluster EKS.

Em uma estratégia de contas com várias cargas de trabalho para clusters multilocatários, as contas da AWS normalmente se alinham aos namespaces do Kubernetes como um mecanismo para isolar grupos de recursos. As melhores práticas para isolamento de inquilinos em um cluster EKS ainda devem ser seguidas ao implementar uma estratégia de várias contas para clusters EKS de vários inquilinos.

É possível ter vários `Cluster Accounts` em sua organização da AWS, e é uma prática recomendada ter vários `Cluster Accounts` que se alinhem às suas necessidades do ciclo de vida de desenvolvimento de software. Para cargas de trabalho que operam em uma escala muito grande, você pode precisar de várias `Cluster Accounts` para garantir que haja cotas de serviços de Kubernetes e AWS suficientes disponíveis para todas as suas cargas de trabalho.



¡No diagrama acima, a AWS RAM é usada para compartilhar sub-redes de uma conta de cluster em uma conta de carga de trabalho. Em seguida, as cargas de trabalho executadas em pods EKS usam

IRSA ou EKS Pod Identities e o encadeamento de funções para assumir uma função em sua conta de carga de trabalho e acessar seus recursos da AWS.

Implementando uma estratégia de conta com várias cargas de trabalho para clusters multilocatários

Compartilhamento de sub-redes com o AWS Resource Access Manager

O AWS Resource Access Manager (RAM) permite que você compartilhe recursos entre contas da AWS.

Se a RAM estiver habilitada para sua organização da AWS, você poderá compartilhar as sub-redes VPC da conta Cluster com suas contas de carga de trabalho. Isso permitirá que os recursos da AWS pertencentes às suas contas de carga de trabalho, como Amazon ElastiCache Clusters ou bancos de dados Amazon Relational Database Service (RDS), sejam implantados na mesma VPC do seu cluster EKS e possam ser consumidos pelas cargas de trabalho em execução no seu cluster EKS.

Para compartilhar um recurso via RAM, abra a RAM no console AWS da conta do cluster e selecione “Compartilhamentos de recursos” e “Criar compartilhamento de recursos”. Nomeie seu compartilhamento de recursos e selecione as sub-redes que você deseja compartilhar. Selecione Avançar novamente e insira a conta de 12 dígitos IDs para as contas de carga de trabalho com as quais você deseja compartilhar as sub-redes, selecione Avançar novamente e clique em Criar compartilhamento de recursos para concluir. Após essa etapa, a conta de carga de trabalho pode implantar recursos nessas sub-redes.

Os compartilhamentos de RAM também podem ser criados programaticamente ou com a infraestrutura como código.

Escolhendo entre o EKS Pod Identities e o IRSA

No re:Invent 2023, a AWS lançou o EKS Pod Identities como uma forma mais simples de entregar credenciais temporárias da AWS para seus pods no EKS. Tanto o IRSA quanto o EKS Pod Identities são métodos válidos para entregar credenciais temporárias da AWS aos seus pods EKS e continuarão sendo suportados. Você deve considerar qual método de entrega atende melhor às suas necessidades.

Ao trabalhar com um cluster EKS e várias contas da AWS, a IRSA pode assumir diretamente funções nas contas da AWS que não sejam a conta na qual o cluster EKS está hospedado diretamente, enquanto as identidades do EKS Pod exigem que você configure o encadeamento de funções. Consulte a documentação do EKS para uma comparação detalhada.

Acessando recursos de API da AWS com funções do IAM para contas de serviço

O IAM Roles for Service Accounts (IRSA) permite que você entregue credenciais temporárias da AWS para suas cargas de trabalho executadas no EKS. O IRSA pode ser usado para obter credenciais temporárias para funções do IAM nas contas de carga de trabalho da conta do cluster. Isso permite que suas cargas de trabalho em execução nos clusters do EKS na conta do cluster consumam recursos da API da AWS, como buckets do S3 hospedados na conta de carga de trabalho, sem interrupções, e usem a autenticação do IAM para recursos como bancos de dados do Amazon RDS ou Amazon EFS. FileSystems

Os recursos da API da AWS e outros recursos que usam a autenticação do IAM em uma conta de carga de trabalho só podem ser acessados por credenciais para funções do IAM nessa mesma conta de carga de trabalho, exceto quando o acesso entre contas é possível e foi habilitado explicitamente.

Habilitando o IRSA para acesso entre contas

Para habilitar o IRSA para cargas de trabalho em sua conta de cluster para acessar recursos em suas contas de carga de trabalho, primeiro você deve criar um provedor de identidade IAM OIDC em sua conta de carga de trabalho. Isso pode ser feito com o mesmo procedimento para configurar o IRSA, exceto que o provedor de identidade será criado na conta da carga de trabalho.

Então, ao configurar o IRSA para suas cargas de trabalho no EKS, você pode seguir as mesmas etapas da documentação, mas usar o ID da conta de carga de trabalho de 12 dígitos, conforme mencionado na seção “Exemplo de criação de um provedor de identidade a partir do cluster de outra conta”.

Depois que isso for configurado, seu aplicativo executado no EKS poderá usar diretamente sua conta de serviço para assumir uma função na conta de carga de trabalho e usar os recursos dentro dela.

Acessando recursos de API da AWS com o EKS Pod Identities

O EKS Pod Identities é uma nova forma de fornecer credenciais da AWS para suas cargas de trabalho executadas no EKS. As identidades de pod do EKS simplificam a configuração dos recursos da AWS, pois você não precisa mais gerenciar as configurações do OIDC para entregar as credenciais da AWS aos seus pods no EKS.

Habilitando o EKS Pod Identities para acesso entre contas

Ao contrário do IRSA, o EKS Pod Identities só pode ser usado para conceder acesso direto a uma função na mesma conta do cluster EKS. Para acessar uma função em outra conta da AWS, os pods que usam o EKS Pod Identities devem realizar o encadeamento de funções.

O encadeamento de funções pode ser configurado em um perfil de aplicativos com seu arquivo de configuração da AWS usando o Process Credentials Provider disponível em várias AWS. SDKs `credential_process`pode ser usado como fonte de credencial ao configurar um perfil, como:

```
# 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
```

A fonte do script chamado 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}'
```

Você pode criar um arquivo de configuração aws conforme mostrado acima com as funções da Conta A e B e especificar as variáveis AWS_CONFIG_FILE e AWS_PROFILE env na especificação do seu pod. O webhook de identidade do EKS Pod não será substituído se os env vars já existirem na especificação do 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
```

Ao configurar políticas de confiança de funções para encadeamento de funções com identidades de pod do EKS, você pode referenciar atributos específicos do EKS como tags de sessão e usar o controle de acesso baseado em atributos (ABAC) para limitar o acesso às suas funções do IAM

somente a sessões de identidade específicas do EKS Pod, como a conta de serviço do Kubernetes à qual um pod pertence.

Observe que alguns desses atributos podem não ser universalmente exclusivos, por exemplo, dois clusters EKS podem ter namespaces idênticos e um cluster pode ter contas de serviço com nomes idênticos em todos os namespaces. Portanto, ao conceder acesso via EKS Pod Identities e ABAC, é uma prática recomendada sempre considerar o arn e o namespace do cluster ao conceder acesso a uma conta de serviço.

Identidades ABAC e EKS Pod para acesso entre contas

Ao usar o EKS Pod Identities para assumir funções (encadeamento de funções) em outras contas como parte de uma estratégia de várias contas, você tem a opção de atribuir uma função exclusiva do IAM para cada conta de serviço que precisa acessar outra conta ou usar uma função comum do IAM em várias contas de serviço e usar o ABAC para controlar quais contas ela pode acessar.

Para usar o ABAC para controlar quais contas de serviço podem assumir uma função em outra conta com o encadeamento de funções, você cria uma declaração de política de confiança de função que só permite que uma função seja assumida por uma sessão de função quando os valores esperados estão presentes. A política de confiança de função a seguir só permitirá que uma função da conta de cluster do EKS (ID da conta 111122223333) assuma uma função se todas as `kubernetes-namespace` tags `eks-cluster-arn` e tiverem o `kubernetes-service-account` 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"
                }
```

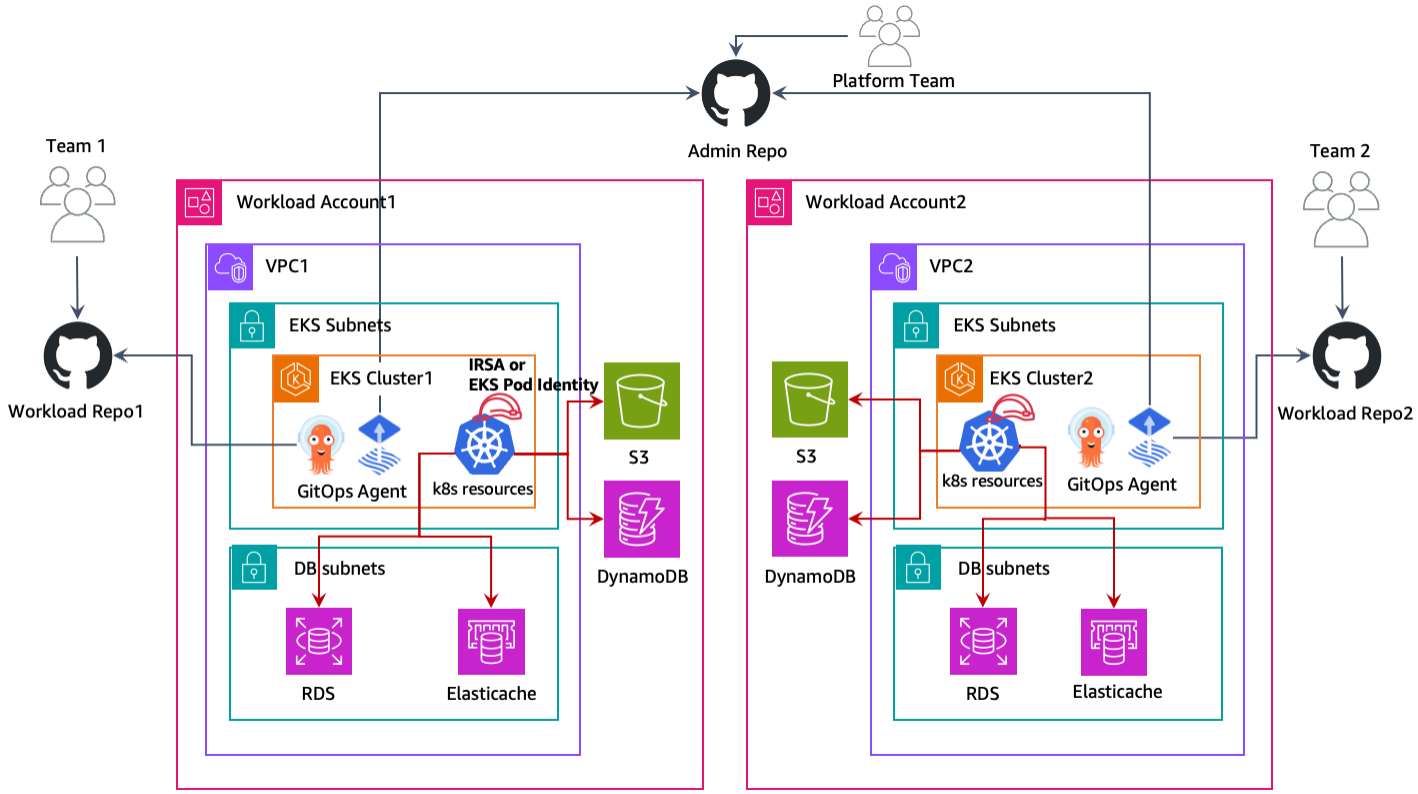
```
            }
        }
    ]
}
```

Ao usar essa estratégia, é uma prática recomendada garantir que a função comum do IAM tenha apenas `sts:AssumeRole` permissões e nenhum outro acesso à AWS.

Ao usar o ABAC, é importante que você controle quem tem a capacidade de marcar funções e usuários do IAM somente para aqueles que têm uma necessidade estrita de fazer isso. Alguém com a capacidade de marcar uma função ou usuário do IAM seria capaz de definir tags roles/users idênticas às definidas pelo EKS Pod Identities e poderia escalar seus privilégios. Você pode restringir quem tem acesso para definir tags e tags na função `kubernetes-` e `eks-` nos usuários do IAM usando a política do IAM ou a Política de Controle de Serviços (SCP).

Clusters EKS descentralizados

Nessa abordagem, os clusters EKS são implantados nas respectivas contas de carga de trabalho da AWS e coexistem com outros recursos da AWS, como buckets do Amazon S3, tabelas do VPCs Amazon DynamoDB etc. Cada conta de carga de trabalho é independente, autossuficiente e operada pelo respectivo cluster de Unit/Application teams. This model allows the creation of reusable blueprints for various cluster capabilities — AI/ML negócios, processamento em lote, uso geral etc., e vende os clusters com base nos requisitos da equipe de aplicativos. As equipes de aplicativos e plataformas operam em seus respectivos GitOpsrepositórios para gerenciar as implantações nos clusters de carga de trabalho.



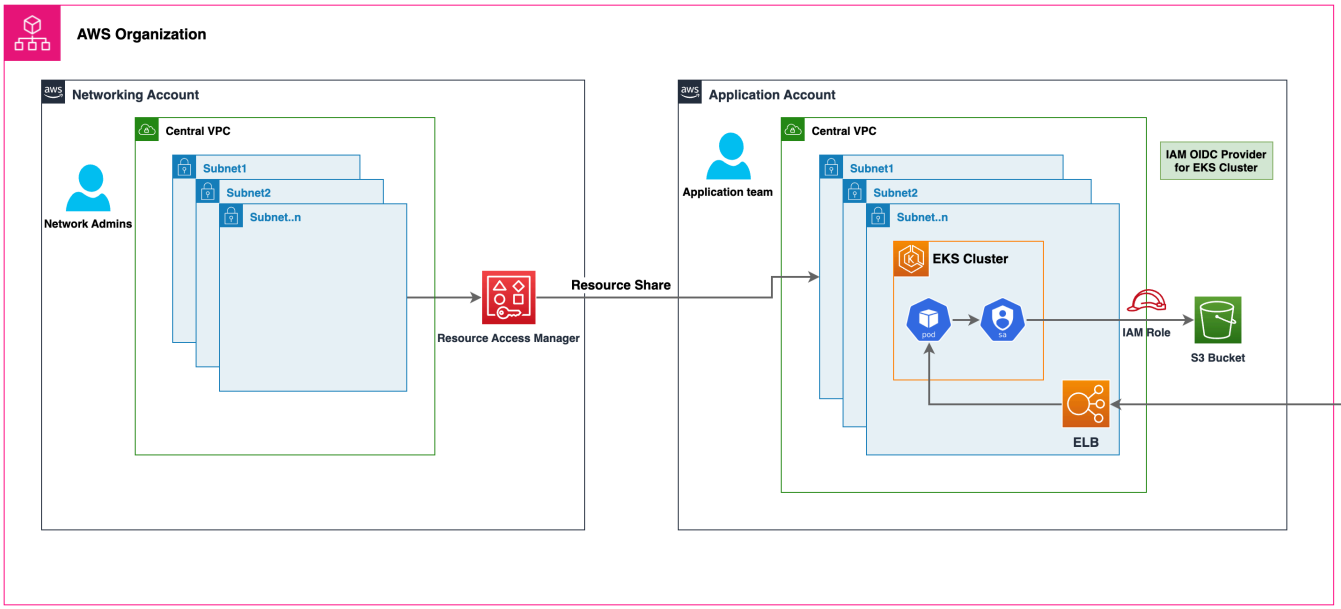
No diagrama acima, os clusters do Amazon EKS e outros recursos da AWS são implantados nas respectivas contas de carga de trabalho. Em seguida, as cargas de trabalho executadas em pods EKS usam IRSA ou EKS Pod Identities para acessar seus recursos da AWS.

GitOps é uma forma de gerenciar a implantação de aplicativos e infraestrutura para que todo o sistema seja descrito declarativamente em um repositório Git. É um modelo operacional que oferece a capacidade de gerenciar o estado de vários clusters Kubernetes usando as melhores práticas de controle de versão, artefatos imutáveis e automação. Nesse modelo de vários clusters, cada cluster de carga de trabalho é inicializado com vários repositórios Git, permitindo que cada equipe (aplicativo, plataforma, segurança etc.) implemente suas respectivas alterações no cluster.

Você utilizaria funções do IAM para contas de serviço (IRSA) ou identidades de pod do EKS em cada conta para permitir que suas cargas de trabalho do EKS obtivessem credenciais temporárias da AWS para acessar com segurança outros recursos da AWS. As funções do IAM são criadas nas respectivas contas da AWS de carga de trabalho e as mapeiam para as contas de serviço k8s para fornecer acesso temporário ao IAM. Portanto, nenhum acesso entre contas é necessário nessa abordagem. Siga a documentação das funções do IAM para contas de serviço sobre como configurar em cada carga de trabalho do IRSA e a documentação do EKS Pod Identities sobre como configurar as identidades de pod do EKS em cada conta.

Rede centralizada

Você também pode utilizar a AWS RAM para compartilhar as sub-redes da VPC com contas de carga de trabalho e lançar clusters do Amazon EKS e outros recursos da AWS nelas. Isso permite gerenciamento/administração de rede centralizados, conectividade de rede simplificada e clusters EKS descentralizados. Consulte este blog da AWS para ver um passo a passo detalhado e considerações sobre essa abordagem.



No diagrama acima, a AWS RAM é usada para compartilhar sub-redes de uma conta de rede central em uma conta de carga de trabalho. Em seguida, o cluster EKS e outros recursos da AWS são lançados nessas sub-redes nas respectivas contas de carga de trabalho. Os pods EKS usam IRSA ou EKS Pod Identities para acessar seus recursos da AWS.

Clusters EKS centralizados versus descentralizados

A decisão de executar com um sistema centralizado ou descentralizado dependerá de seus requisitos. Esta tabela demonstra as principais diferenças com cada estratégia.

#	Cluster EKS centralizado	Clusters EKS descentralizados
Gerenciamento de clusters:	Gerenciar um único cluster EKS é mais fácil do que administrar vários clusters	Uma automação eficiente do gerenciamento de clusters é necessária para reduzir a sobrecarga operacional

#	Cluster EKS centralizado	Clusters EKS descentralizados
		do gerenciamento de vários clusters EKS
Eficiência de custos:	Permite a reutilização do cluster e dos recursos de rede do EKS, o que promove a eficiência de custos	Requer configurações de rede e cluster por carga de trabalho, o que requer recursos adicionais
Resiliência:	Várias cargas de trabalho no cluster centralizado podem ser afetadas se um cluster for prejudicado	Se um cluster for prejudicado, o dano será limitado somente às cargas de trabalho executadas nesse cluster. Todas as outras cargas de trabalho não são afetadas
Isolamento e segurança:	A multilocação isolada/flexível é obtida usando construções nativas do k8s, como. `Namespaces` As cargas de trabalho podem compartilhar os recursos subjacentes, como CPU, memória etc. Os recursos da AWS são isolados em suas próprias contas de carga de trabalho que, por padrão, não são acessíveis a partir de outras contas da AWS.	Maior isolamento dos recursos computacionais, pois as cargas de trabalho são executadas em clusters e nós individuais que não compartilham nenhum recurso. Os recursos da AWS são isolados em suas próprias contas de carga de trabalho que, por padrão, não são acessíveis a partir de outras contas da AWS.

#	Cluster EKS centralizado	Clusters EKS descentralizados
Desempenho e escalabilidade:	À medida que as cargas de trabalho crescem em escalas muito grandes, você pode encontrar kubernetes e cotas de serviços da AWS na conta do cluster. Você pode implantar contas de cluster adicionais para escalar ainda mais	À medida que mais clusters VPCs estão presentes, cada carga de trabalho tem mais k8s disponíveis e cota de serviços da AWS
Redes:	Uma única VPC é usada por cluster, permitindo uma conectividade mais simples para aplicativos nesse cluster	O roteamento deve ser estabelecido entre o cluster EKS descentralizado VPCs
Gerenciamento de acesso ao Kubernetes:	É necessário manter muitas funções e usuários diferentes no cluster para fornecer acesso a todas as equipes de carga de trabalho e garantir que os recursos do Kubernetes sejam segregados adequadamente	Gerenciamento de acesso simplificado, pois cada cluster é dedicado a uma carga de trabalho/equipe
Gerenciamento de acesso da AWS:	Os recursos da AWS são implantados em sua própria conta, que só pode ser acessada por padrão com funções do IAM na conta de carga de trabalho. As funções do IAM nas contas de carga de trabalho são assumidas em várias contas com IRSA ou EKS Pod Identities.	Os recursos da AWS são implantados em sua própria conta, que só pode ser acessada por padrão com funções do IAM na conta de carga de trabalho. As funções do IAM nas contas de carga de trabalho são entregues diretamente aos pods com IRSA ou EKS Pod Identities

Gerenciamento de acesso ao cluster

O gerenciamento eficaz do acesso é crucial para manter a segurança e a integridade dos seus clusters do Amazon EKS. Este guia explora várias opções de gerenciamento de acesso ao EKS, com foco no uso do AWS IAM Identity Center (antigo AWS SSO). Compararemos diferentes abordagens, discutiremos suas vantagens e desvantagens e destacaremos as limitações e considerações conhecidas.

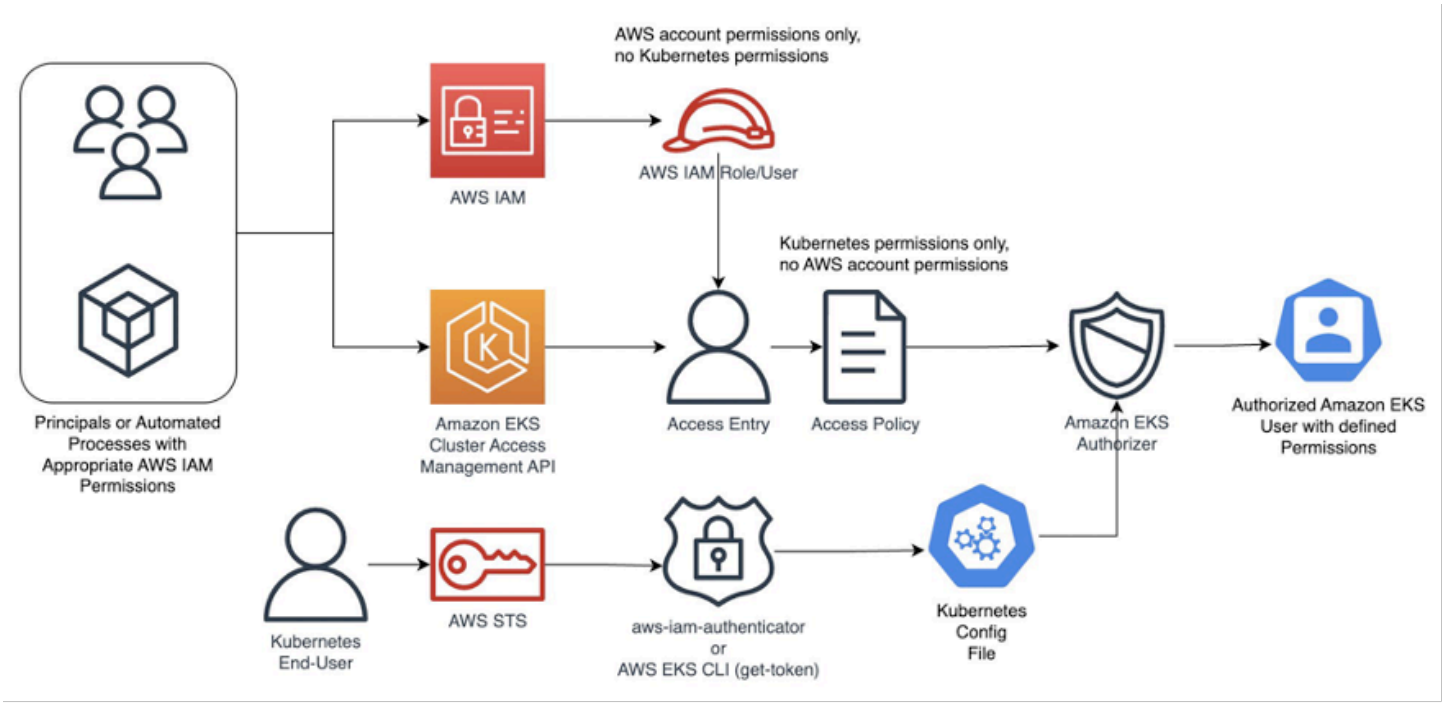
Opções de gerenciamento de acesso EKS

> **Note**
>
> ConfigMapo gerenciamento de acesso baseado (aws-auth ConfigMap) está obsoleto e substituído pela API Cluster Access Management (CAM). Para novos clusters EKS, implemente a API CAM para gerenciar o acesso ao cluster. Para clusters existentes usando aws-auth ConfigMap, migre para o uso da API CAM.

Opção 1: AWS IAM Identity Center com API de gerenciamento de acesso ao cluster (CAM)

- Gerenciamento centralizado de usuários e permissões
- Integração com provedores de identidade existentes (por exemplo, Microsoft AD, Okta PingId e outros)
- A API CAM usa entradas de acesso para vincular os principais do AWS IAM (usuários ou funções) ao cluster EKS. Essas entradas funcionam com as identidades gerenciadas do IAM Identity Center, permitindo que os administradores controlem o acesso ao cluster para usuários e grupos definidos no Identity Center.

Fluxo de autenticação do cluster EKS:



1. Os diretores (usuários humanos) ou processos automatizados se autenticam por meio do AWS IAM apresentando as permissões apropriadas da conta da AWS. Nesta etapa, eles são mapeados para o principal (função ou usuário) apropriado do AWS IAM.
2. Em seguida, uma entrada de acesso do EKS mapeia esse principal do IAM para um principal do Kubernetes RBAC (usuário ou grupo) definindo a política de acesso apropriada, que contém somente as permissões do Kubernetes.
3. Quando um usuário final do Kubernetes tenta acessar um cluster, sua solicitação de autenticação é processada pela CLI do aws-iam-authenticator AWS EKS e validada em relação ao contexto do cluster no arquivo kubeconfig.
4. Por fim, o autorizador do EKS verifica as permissões associadas à entrada de acesso do usuário autenticado e concede ou nega o acesso adequadamente.
    - A API usa políticas de acesso específicas do Amazon EKS para definir o nível de autorização para cada entrada de acesso. Essas políticas podem ser mapeadas para funções e permissões configuradas no IAM Identity Center, garantindo um controle de acesso consistente nos serviços da AWS e nos clusters do EKS.

Benefícios ConfigMap em relação ao gerenciamento de acesso baseado:

1. Risco reduzido de configurações incorretas: o gerenciamento direto baseado em API elimina erros comuns associados à edição manual. ConfigMap Isso ajuda a evitar exclusões acidentais ou erros de sintaxe que podem impedir que os usuários saiam do cluster.
2. Princípio aprimorado de privilégios mínimos: elimina a necessidade de permissão de administrador de cluster da identidade do criador do cluster e permite uma atribuição de permissões mais granular e apropriada. Você pode optar por adicionar essa permissão para casos de uso de quebra-vidros.
3. Modelo de segurança aprimorado: fornece validação integrada das entradas de acesso antes de serem aplicadas. Além disso, oferece maior integração com o AWS IAM para autenticação.
4. Operações simplificadas: oferece uma maneira mais intuitiva de gerenciar permissões por meio de ferramentas nativas da AWS.

Práticas recomendadas:

1. Use o AWS Organizations para gerenciar várias contas e aplicar políticas de controle de serviços (SCPs).
2. Implemente o princípio de privilégio mínimo criando conjuntos de permissões específicos para diferentes funções do EKS (por exemplo, administrador, desenvolvedor, somente leitura).
3. Utilize o controle de acesso baseado em atributos (ABAC) para atribuir dinamicamente permissões aos pods com base nos atributos do usuário.
4. Audite e revise regularmente as permissões de acesso.

Considerações/limitações:

1. As funções ARNs geradas pelo Identity Center têm sufixos aleatórios, o que as torna difíceis de usar em configurações estáticas.
2. Suporte limitado para permissões refinadas no nível de recursos do Kubernetes. É necessária uma configuração adicional para funções RBAC personalizadas do Kubernetes. Junto com o RBAC nativo do Kubernetes, considere usar o Kyverno para gerenciamento avançado de permissões em clusters EKS.

Opção 2: AWS IAM Users/Roles mapeado para grupos do Kubernetes

Prós:

1. Controle refinado sobre as permissões do IAM.
2. Papel previsível e estático ARNs

Contras:

1. Aumento da sobrecarga de gerenciamento das contas de usuário
2. Falta de gerenciamento centralizado de identidade
3. Potencial de proliferação de entidades do IAM

Práticas recomendadas:

1. Use funções do IAM em vez de usuários do IAM para melhorar a segurança e a capacidade de gerenciamento
2. Implemente uma convenção de nomenclatura para funções para garantir consistência e facilidade de gerenciamento
3. Utilize as condições da política do IAM para restringir o acesso com base em tags ou outros atributos.
4. Alterne regularmente as chaves de acesso e revise as permissões.

Considerações/limitações:

1. Problemas de escalabilidade ao gerenciar um grande número de usuários ou funções
2. Sem recursos integrados de login único

Opção 3: Provedores do OIDC

Prós:

1. Integração com sistemas de gerenciamento de identidade existentes
2. Redução da sobrecarga de gerenciamento das contas de usuário

Contras:

1. Complexidade adicional de configuração
2. Potencial para maior latência durante a autenticação
3. Dependência do provedor de identidade externo

Melhores práticas:

1. Configure cuidadosamente o provedor OIDC para garantir a validação segura do token.
2. Use tokens de curta duração e implemente mecanismos de atualização de tokens.
3. Audite e atualize regularmente as configurações do OIDC.

Consulte este guia para obter uma implementação de referência da integração de provedores externos de Single Sign-On com o Amazon EKS

Considerações/limitações:

1. Integração nativa limitada com os serviços da AWS em comparação com o IAM.
2. O URL do emissor do provedor do OIDC deve estar acessível ao público para que o EKS descubra as chaves de assinatura.

AWS EKS Pod Identity versus IRSA para cargas de trabalho

O Amazon EKS fornece duas maneiras de conceder permissões do AWS IAM para cargas de trabalho executadas em clusters do Amazon EKS: funções do IAM para contas de serviço (IRSA) e EKS Pod Identities.

Embora tanto o IRSA quanto o EKS Pod Identities ofereçam os benefícios de acesso com privilégios mínimos, isolamento de credenciais e auditabilidade, o EKS Pod Identity é a forma recomendada de conceder permissões às cargas de trabalho.

Para obter orientações detalhadas sobre identidade e credenciais para pods EKS, consulte a seção Identidades e credenciais das melhores práticas de segurança.

Recomendação

Combine o IAM Identity Center com a API CAM

- Gerenciamento simplificado: ao usar a API Cluster Access Management em conjunto com o IAM Identity Center, os administradores podem gerenciar o acesso ao cluster EKS junto com outros serviços da AWS, reduzindo a necessidade de alternar entre diferentes interfaces ou editar ConfigMaps manualmente.
- Use entradas de acesso para gerenciar as permissões do Kubernetes das entidades principais do IAM de fora do cluster. Você pode adicionar e gerenciar o acesso ao cluster usando a API EKS, a interface de linha de comando da AWS, a AWS SDKs, a AWS CloudFormation e o AWS Management Console. Isso significa que você pode gerenciar usuários com as mesmas ferramentas com as quais criou o cluster.
- As permissões granulares do Kubernetes podem ser aplicadas com o mapeamento de usuários ou grupos do Kubernetes com diretores do IAM associados a identidades de SSO por meio de entradas de acesso e políticas de acesso.
- Para começar, siga Alterar o modo de autenticação para usar entradas de acesso e, em seguida, Migrar entradas aws-auth existentes para ConfigMap entradas de acesso.

Melhores práticas para escalonamento automático de clusters

Este guia fornece conselhos sobre o escalonamento automático de clusters, incluindo orientações sobre o modo automático, o Karpenter e o escalonador automático de cluster do Kubernetes.

Tópicos

- Modo Automático do EKS
- Karpenter
- Autoscaler do cluster

Modo Automático do EKS

> **Tip**
>
> Interessado em adquirir experiência prática com o Amazon EKS Auto Mode? Participe de um próximo workshop virtual sobre Kubernetes conduzido por especialistas da AWS inscrevendo-se na AWS Connected Community.

O Amazon EKS Auto Mode representa uma evolução significativa no gerenciamento da infraestrutura do Kubernetes, combinando uma infraestrutura de cluster segura e escalável com recursos integrados do Kubernetes gerenciados pela AWS. O serviço fornece operações de nós de trabalho totalmente gerenciadas, eliminando a necessidade de os clientes configurarem grupos ou AutoScaling grupos de nós gerenciados.

A principal diferença arquitetônica é que o EKS Auto Mode usa um sistema baseado em Karpenter que provisiona automaticamente as EC2 instâncias em resposta às solicitações do pod. Essas instâncias são executadas no Bottlerocket AMIs com complementos pré-instalados, como drivers EBS CSI, tornando a infraestrutura verdadeiramente gerenciada pela AWS. Em contraste com os métodos tradicionais de escalabilidade:

- O Autoescalador de Cluster (CAS) tradicional exige gerenciamento manual de grupos de nós e só pode criar nós com um único tipo de instância por grupo de nós

- O Karpenter autogerenciado oferece mais flexibilidade ao trabalhar com a API EC2 Fleet e pode provisionar diferentes tipos de instância, mas exige gerenciamento de clientes
- O EKS Auto Mode lida com todas as operações de dimensionamento automaticamente por meio de gerenciamento NodePools e NodeClasses

O novo sistema introduz várias melhorias operacionais:

- Dimensionamento automático baseado em pods sem configuração manual de grupos de nós
- Controladores de balanceador de carga gerenciados integrados que são criados automaticamente ALB/NLB com base nos recursos do Ingress
- Recursos de segurança integrados com identidade de pod pré-configurada
- Tempo máximo de execução do nó de 21 dias com substituição automática

Do ponto de vista do custo, o EKS Auto Mode mantém o EC2 preço padrão enquanto adiciona uma taxa de gerenciamento somente para os nós gerenciados pelo Auto Mode. É importante ressaltar que os clientes ainda podem combinar nós gerenciados do Modo Automático com nós autogerenciados no mesmo cluster.

Embora a AWS cuide da maioria dos aspectos operacionais, os clientes mantêm a responsabilidade pelo gerenciamento de versões do cluster e podem realizar atualizações controladas que acionam atualizações contínuas dos nós de trabalho.

Razões para usar o Modo Automático

O Modo Automático é voltado para usuários que desejam os benefícios do Kubernetes e do EKS, mas precisam minimizar a carga operacional do Kubernetes, como atualizações, e de partes installation/maintenance críticas da plataforma, como auto-scaling, balanceamento de carga e armazenamento. O Modo Automático leva o EKS um passo adiante na minimização do trabalho pesado indiferenciado que acompanha a manutenção do Kubernetes

Perguntas frequentes

Qual é a diferença entre o EKS Auto Mode e o Open Source Karpenter?

O EKS Auto Mode é um grande conjunto de recursos que simplifica a execução do Kubernetes de nível de produção. Um desses recursos são os benefícios de auto-scaling do Karpenter, totalmente

gerenciados. Do ponto de vista operacional, a única diferença é que, no modo automático do EKS, você não precisa gerenciar a implantação, o dimensionamento e a atualização dos próprios pods do Karpenter. Todas as outras operações, como gerenciadas NodeClasses e NodePools funcionam da mesma forma que com o Karpenter de código aberto.

Posso executar grupos de nós gerenciados junto com nós gerenciados pelo Modo Automático?

Sim, você pode executar nós estáticos por meio de grupos de nós gerenciados junto com seus nós de escalonamento automático fornecidos com o Modo Automático

Posso migrar um cluster do EKS padrão para o EKS Auto Mode?

Sim, as instruções para habilitar o EKS Auto Mode em um cluster existente podem ser encontradas na documentação oficial da AWS

Coisas a serem observadas: 1. Depois de ativar o Modo Automático, você desejará desinstalar todos os componentes que você instalou e que agora são gerenciados pelo Modo Automático, como o Karpenter ou o AWS Load Balancer Controller 2. Você precisa ter certeza de que seus complementos instalados estão up-to-date. Veja a documentação.

Como faço para configurar NodePools no modo automático do EKS?

Um novo cluster virá pré-configurado com dois NodePools

de uso geral

```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
```

Isso NodePool instrui o Karpenter a lançar nós com as seguintes características:

1. Tipo de capacidade “On Demand”
2. Tipos de instância de C, M ou R
3. Geração de 4 instâncias
4. Arquitetura AMD
5. SO Linux

Ele também define qual é a lógica de redução ao declarar que apenas 10% de todos os nós podem estar em um estado interrompido a qualquer momento e que a consolidação só deve ocorrer quando os nós estiverem vazios ou subutilizados.

operacional

```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
```

Isso NodePool é semelhante ao “propósito geral”, exceto pelas seguintes diferenças:

1. Ele permite nós com a arquitetura ARM, bem como a arquitetura AMD
2. Isso contamina esses nós com a, NoSchedule a menos que haja uma tolerância para “”CriticalAddonsOnly. Isso é para uso interno por complementos do EKS

personalizado

Você pode criar seu próprio personalizado, NodePools dependendo de suas necessidades. Para saber mais sobreNodePools , consulte a documentação do Karpenter.

Posso personalizar a AMI usada pelo Modo Automático quando novos nós são iniciados?

Não, atualmente, os únicos compatíveis AMIs são para o Bottlerocket fornecido pela Amazon

Como posso instalar ferramentas ou agentes personalizados em meus hosts Kubernetes?

Como a personalização da AMI não é suportada, se você precisar de software em nível de host para coisas como verificação de segurança, você deve implantar a carga de trabalho como um Kubernetes. DaemonSet

Quais componentes estão sendo executados no meu plano de dados de cluster quando eu provisiono um novo cluster EKS Auto Mode?

Se você criar um cluster com o eksctl ou o console AWS, os únicos pods em execução em um cluster do EKS Auto Mode são os pods do Kubernetes Metrics Server. Os outros componentes do EKS Auto Mode, como o Karpenter, o AWS Load Balancer Controller e o driver EBS CSI, são todos executados e gerenciados fora do cluster.

Quais componentes gerenciados estão sendo executados para dar suporte ao meu novo cluster EKS Auto Mode?

O EKS Auto Mode automatiza completamente a implantação da maioria das partes de um plano de dados necessárias para o Kubernetes de nível de produção. Isso inclui:

- Karpenter, para escalar automaticamente a computação do seu cluster

- Controlador do AWS Load Balancer para permitir que você exponha facilmente os serviços do Kubernetes por meio da integração automatizada do Elastic Load Balancer
- WEB CSI
- CNI DE PVC
- Atendente de Identidade de Pods do EKS

Como soluciono problemas dos componentes do Modo Automático que costumavam ser executados como pods no meu cluster?

Com o EKS Auto Mode, muitos dos componentes, como o AWS Load Balancer Controller e o Karpenter, são gerenciados para você fora do seu cluster, portanto, você não terá a mesma visibilidade dos registros com a qual está acostumado quando se autogerencia. Se você estiver em uma situação em que precise solucionar problemas com a funcionalidade de uma parte da funcionalidade do Modo Automático, crie um ticket do AWS Support.

Karpenter

O Karpenter é um projeto de código aberto projetado para aprimorar o gerenciamento do ciclo de vida dos nós nos clusters do Kubernetes. Ele automatiza o provisionamento e o desprovisionamento de nós com base nas necessidades específicas de agendamento dos pods, permitindo escalabilidade eficiente e otimização de custos. Suas principais funções são:

- Monitore pods que o programador do Kubernetes não pode programar devido a restrições de recursos.
- Avalie os requisitos de agendamento (solicitações de recursos, seletores de nós, afinidades, tolerações etc.) dos pods não programáveis.
- Provisione novos nós que atendam aos requisitos desses pods.
- Remova os nós quando eles não forem mais necessários.

Com o Karpenter, você pode definir NodePools com restrições no provisionamento de nós, como manchas, rótulos, requisitos (tipos de instância, zonas etc.) e limites no total de recursos provisionados. Ao implantar cargas de trabalho, você pode especificar várias restrições de agendamento nas especificações do pod, como requests/limits, node selectors, node/pod afinidades de recursos, tolerâncias e restrições de distribuição de topologia. O Karpenter então provisionará nós do tamanho certo com base nessas especificações.

Razões para usar o Karpenter

Antes do lançamento do Karpenter, os usuários do Kubernetes dependiam principalmente dos grupos do Amazon Auto Scaling EC2 e do Kubernetes Cluster Autoscaler (CAS) para ajustar dinamicamente a capacidade computacional de seus clusters. Com o Karpenter, você não precisa criar dezenas de grupos de nós para obter a flexibilidade e a diversidade que você obtém com o Karpenter. Ao contrário do CAS, o Karpenter não é tão acoplado às versões do Kubernetes e não exige que você alterne entre a AWS e o Kubernetes. APIs

O Karpenter consolida as responsabilidades de orquestração de instâncias em um único sistema, que é mais simples, mais estável e reconhece clusters. O Karpenter foi projetado para superar alguns dos desafios apresentados pelo Cluster Autoscaler, fornecendo maneiras simplificadas de:

- Provisione nós com base nos requisitos da carga de trabalho.
- Crie diversas configurações de nós por tipo de instância, usando NodePool opções flexíveis. Em vez de gerenciar muitos grupos de nós personalizados específicos, o Karpenter pode permitir que você gerencie diversas capacidades de carga de trabalho com um único e flexível. NodePool
- Obtenha um melhor agendamento de pods em grande escala lançando nós e agendando pods rapidamente.

Para obter informações e documentação sobre o uso do Karpenter, visite o site karpenter.sh.

Recomendações

As melhores práticas são divididas em seções sobre o próprio Karpenter e o agendamento NodePools de pods.

Melhores práticas da Karpenter

As melhores práticas a seguir abrangem tópicos relacionados ao próprio Karpenter.

Bloqueio AMIs em clusters de produção

É altamente recomendável que você fixe imagens conhecidas da Amazon Machine (AMIs) usadas pelo Karpenter para clusters de produção. Usar `amiSelector` com um alias definido como`@latest`, ou usar algum outro método que resulte em implantações não testadas à AMIs medida que são lançadas, oferece o risco de falhas na carga de trabalho e tempo de inatividade em seus clusters de produção. Como resultado, é altamente recomendável fixar versões de trabalho

testadas do AMIs para seus clusters de produção enquanto você testa versões mais recentes em clusters que não são de produção. Por exemplo, você pode definir um alias no seu da NodeClass seguinte forma:

```
amiSelectorTerms
  - alias: al2023@v20240807
```

Para obter informações sobre como gerenciar e fixar AMIs no Karpenter, consulte Gerenciando AMIs na documentação do Karpenter.

Use o Karpenter para cargas de trabalho com necessidades de capacidade variáveis

O Karpenter aproxima o gerenciamento de escalabilidade do Kubernetes nativo APIs do que os grupos de escalonamento automático () e os grupos de nós gerenciados ()ASGs. MNGs ASGs e MNGs são abstrações nativas da AWS em que a escalabilidade é acionada com base em métricas de nível da AWS, como carga da CPU. EC2 O Cluster Autoscaler une as abstrações do Kubernetes às abstrações da AWS, mas perde alguma flexibilidade por causa disso, como o agendamento para uma zona de disponibilidade específica.

O Karpenter remove uma camada de abstração da AWS para trazer parte da flexibilidade diretamente para o Kubernetes. O Karpenter é melhor usado para clusters com cargas de trabalho que enfrentam períodos de alta demanda ou têm diversos requisitos de computação. MNGs e ASGs são bons para clusters que executam cargas de trabalho que tendem a ser mais estáticas e consistentes. Você pode usar uma combinação de nós gerenciados de forma dinâmica e estática, dependendo dos seus requisitos.

Considere outros projetos de escalonamento automático quando...

Você precisa de recursos que ainda estão sendo desenvolvidos no Karpenter. Como o Karpenter é um projeto relativamente novo, considere outros projetos de escalonamento automático por enquanto, se você precisar de recursos que ainda não façam parte do Karpenter.

Execute o controlador Karpenter no EKS Fargate ou em um nó de trabalho que pertença a um grupo de nós

O Karpenter é instalado usando um gráfico Helm. O gráfico Helm instala o controlador Karpenter e um pod de webhook como uma implantação que precisa ser executada antes que o controlador possa ser usado para escalar seu cluster. Recomendamos no mínimo um pequeno grupo de nós com pelo menos um nó de trabalho. Como alternativa, você pode executar esses pods no EKS

Fargate criando um perfil Fargate para o namespace. `karpenter` Isso fará com que todos os pods implantados nesse namespace sejam executados no EKS Fargate. Não execute o Karpenter em um nó gerenciado pelo Karpenter.

Não há suporte para modelos de lançamento personalizados com o Karpenter

Não há suporte para modelos de lançamento personalizados com a v1 APIs. Você pode usar dados personalizados do usuário especificando and/or diretamente o personalizado AMIs no EC2NodeClass. Mais informações sobre como fazer isso estão disponíveis em NodeClasses.

Exclua tipos de instância que não se adequam à sua carga de trabalho

Considere excluir tipos específicos de instâncias com a `node.kubernetes.io/instance-type` chave se eles não forem exigidos pelas cargas de trabalho em execução no seu cluster.

O exemplo a seguir mostra como evitar o provisionamento de grandes instâncias do Graviton.

```
- key: node.kubernetes.io/instance-type
  operator: NotIn
  values:
  - m6g.16xlarge
  - m6gd.16xlarge
  - r6g.16xlarge
  - r6gd.16xlarge
  - c6g.16xlarge
```

Ative o tratamento de interrupções ao usar o Spot

O Karpenter suporta o tratamento nativo de interrupções e pode lidar com eventos de interrupção involuntários, como interrupções de instâncias spot, eventos de manutenção programada, eventos de terminação/parada de instâncias que podem interromper suas cargas de trabalho. Quando o Karpenter detecta esses eventos nos nós, ele automaticamente contamina, drena e encerra os nós afetados com antecedência para iniciar uma limpeza elegante das cargas de trabalho antes da interrupção. Para interrupções do Spot com 2 minutos de antecedência, o Karpenter inicia rapidamente um novo nó para que os pods possam ser movidos antes que a instância seja recuperada. Para habilitar o tratamento de interrupções, você configura o argumento `--interruption-queue` CLI com o nome da fila SQS provisionada para essa finalidade. Não é aconselhável usar o tratamento de interrupções do Karpenter junto com o Node Termination Handler, conforme explicado aqui.

Os pods que requerem um posto de controle ou outras formas de drenagem cuidadosa, exigindo 2 minutos antes do desligamento, devem permitir o tratamento de interrupções do Karpenter em seus clusters.

Cluster privado Amazon EKS sem acesso de saída à Internet

Ao provisionar um cluster EKS em uma VPC sem rota para a Internet, você precisa se certificar de que configurou seu ambiente de acordo com os requisitos de cluster privado que aparecem na documentação do EKS. Além disso, você precisa ter certeza de que criou um endpoint regional STS VPC em sua VPC. Caso contrário, você verá erros semelhantes aos que aparecem abaixo.

```
{"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"}
```

Essas alterações são necessárias em um cluster privado porque o Karpenter Controller usa funções do IAM para contas de serviço (IRSA). Os pods configurados com o IRSA adquirem credenciais chamando a API do AWS Security Token Service (AWS STS). Se não houver acesso externo à Internet, você deverá criar e usar um endpoint de VPC do AWS STS em sua VPC.

Os clusters privados também exigem que você crie um VPC endpoint para SSM. Quando o Karpenter tenta provisionar um novo nó, ele consulta as configurações do modelo Launch e um parâmetro SSM. Se você não tiver um endpoint SSM VPC em sua VPC, isso causará o seguinte erro:

```
{"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"}
```

Não há um VPC endpoint para a API Price List Query. Como resultado, os dados de preços ficarão obsoletos com o tempo. O Karpenter contorna isso incluindo dados de preços sob demanda em seu

binário, mas só atualiza esses dados quando o Karpenter é atualizado. Solicitações falhadas de dados de preços resultarão nas seguintes mensagens de erro:

```
{"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 documentação para usar o Karpenter em clusters EKS completamente privados e para saber quais endpoints de VPC devem ser criados.

Criando NodePools

As práticas recomendadas a seguir abrangem tópicos relacionados à criação NodePools.

Crie vários NodePools quando...

Quando equipes diferentes estiverem compartilhando um cluster e precisarem executar suas cargas de trabalho em diferentes nós de trabalho ou tiverem requisitos diferentes de sistema operacional ou tipo de instância, crie várias NodePools. Por exemplo, uma equipe pode querer usar o Bottlerocket, enquanto outra pode querer usar o Amazon Linux. Da mesma forma, uma equipe pode ter acesso a um hardware de GPU caro que não seria necessário para outra equipe. O uso de vários NodePools garante que os ativos mais adequados estejam disponíveis para cada equipe.

Crie NodePools que sejam mutuamente exclusivos ou ponderados

É recomendável criar NodePools que sejam mutuamente exclusivas ou ponderadas para fornecer um comportamento de agendamento consistente. Se não estiverem e vários NodePools forem combinados, o Karpenter escolherá aleatoriamente quais usar, causando resultados inesperados. Exemplos úteis para criar vários NodePools incluem o seguinte:

Criar um NodePool com GPU e permitir que cargas de trabalho especiais sejam executadas apenas nesses nós (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"
```

Implantação com tolerância à contaminação:

```
# 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"
```

Para uma implantação geral para outra equipe, a NodePool especificação pode incluir NodeAffinity. Uma implantação poderia então ser usada nodeSelectorTerms para corresponder`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
```

Implantação usando 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"]
```

Use temporizadores (TTL) para excluir automaticamente os nós do cluster

Você pode usar temporizadores em nós provisionados para definir quando excluir nós que estão desprovidos de pods de carga de trabalho ou que atingiram um prazo de expiração. A expiração do nó pode ser usada como um meio de atualização, para que os nós sejam retirados e substituídos por versões atualizadas. Consulte Expiração na documentação do Karpenter para obter informações sobre como usar para configurar `spec.template.spec` a expiração do nó.

Evite restringir excessivamente os tipos de instância que o Karpenter pode provisionar, especialmente ao utilizar o Spot

Ao usar o Spot, a Karpenter usa a estratégia de alocação otimizada para capacidade de preço para provisionar instâncias. EC2 Essa estratégia instrui EC2 a provisionar instâncias dos pools mais profundos para o número de instâncias que você está iniciando e com o menor risco de

interrupção. EC2 Em seguida, o Fleet solicita instâncias spot do menor preço desses pools. Quanto mais tipos de instância você permitir que o Karpenter utilize, melhor EC2 poderá otimizar o tempo de execução da sua instância spot. Por padrão, o Karpenter usará todas as EC2 ofertas de Tipos de Instância na região e nas zonas de disponibilidade em que seu cluster está implantado. O Karpenter escolhe de forma inteligente o conjunto de todos os tipos de instância com base nos pods pendentes para garantir que seus pods sejam programados em instâncias de tamanho e equipamento adequados. Por exemplo, se seu pod não precisar de uma GPU, o Karpenter não o programará para um tipo de EC2 instância compatível com uma GPU. Quando não tiver certeza sobre quais tipos de instância usar, você pode executar o Amazon ec2-instance-selector para gerar uma lista de tipos de instância que correspondam aos seus requisitos computacionais. Por exemplo, a CLI usa memória vCPU, arquitetura e região como parâmetros de entrada e fornece uma lista de EC2 instâncias que satisfazem essas restrições.

```
$ 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
```

Você não deve colocar muitas restrições no Karpenter ao usar instâncias spot, pois isso pode afetar a disponibilidade de seus aplicativos. Digamos, por exemplo, que todas as instâncias de um determinado tipo sejam recuperadas e não haja alternativas adequadas disponíveis para substituí-las. Seus pods permanecerão em um estado pendente até que a capacidade spot dos tipos de instância configurados seja reabastecida. Você pode reduzir o risco de erros de capacidade insuficientes distribuindo suas instâncias em diferentes zonas de disponibilidade, porque os pools spot são diferentes entre si AZs. Dito isso, a melhor prática geral é permitir que o Karpenter use um conjunto diversificado de tipos de instância ao usar o Spot.

Pods de agendamento

As melhores práticas a seguir estão relacionadas à implantação de pods em um cluster usando o Karpenter para provisionamento de nós.

Siga as melhores práticas do EKS para alta disponibilidade

Se você precisar executar aplicativos de alta disponibilidade, siga as recomendações gerais de melhores práticas do EKS. Consulte a documentação Topology Spread in Karpenter para obter detalhes sobre como distribuir pods entre nós e zonas. Use o Disruption Budgets para definir o mínimo de pods disponíveis que precisam ser mantidos, caso haja tentativas de despejar ou excluir os pods.

Use restrições em camadas para restringir os recursos de computação disponíveis no seu provedor de nuvem

O modelo de restrições em camadas do Karpenter permite que você crie um conjunto complexo de restrições de implantação de pods para obter as NodePool melhores correspondências possíveis para o agendamento de pods. Exemplos de restrições que uma especificação de pod pode solicitar incluem o seguinte:

- Necessidade de execução em zonas de disponibilidade onde somente aplicativos específicos estão disponíveis. Digamos, por exemplo, que você tenha um pod que precisa se comunicar com outro aplicativo executado em uma EC2 instância residente em uma zona de disponibilidade específica. Se seu objetivo é reduzir o tráfego entre AZ em sua VPC, talvez você queira colocar os pods na AZ em que a instância está EC2 localizada. Esse tipo de segmentação geralmente é realizado usando seletores de nós. Para obter informações adicionais sobre seletores de nós, consulte a documentação do Kubernetes.
- Exigindo certos tipos de processadores ou outro hardware. Consulte a seção Aceleradores da documentação do Karpenter para ver um exemplo de especificação de pod que exige que o pod seja executado em uma GPU.

Crie alarmes de cobrança para monitorar seus gastos com computação

Ao configurar seu cluster para escalar automaticamente, você deve criar alarmes de cobrança para avisá-lo quando seus gastos excederem um limite e adicionar limites de recursos à sua configuração do Karpenter. Definir limites de recursos com o Karpenter é semelhante a definir a capacidade máxima de um grupo de escalonamento automático da AWS, pois representa a quantidade máxima de recursos computacionais que podem ser instanciados por um Karpenter. NodePool

> **Note**
>
> Não é possível definir um limite global para todo o cluster. Os limites se aplicam a determinados NodePools.

O trecho abaixo diz ao Karpenter para provisionar apenas um máximo de 1000 núcleos de CPU e 1000Gi de memória. O Karpenter deixará de adicionar capacidade somente quando o limite for atingido ou excedido. Quando um limite é excedido, o controlador Karpenter grava uma mensagem semelhante nos registros do controlador. `memory resource usage of 1001 exceeds limit of 1000` Se você estiver roteando seus registros de contêiner para CloudWatch registros, poderá criar um filtro de métricas para procurar padrões ou termos específicos em seus registros e, em seguida, criar um CloudWatchalarme para alertá-lo quando o limite de métricas configurado for violado.

Para obter mais informações sobre como usar limites com o Karpenter, consulte Definindo limites de recursos na documentação do Karpenter.

```
spec:
  limits:
    cpu: 1000
    memory: 1000Gi
```

Se você não usar limites ou restringir os tipos de instância que o Karpenter pode provisionar, o Karpenter continuará adicionando capacidade computacional ao seu cluster conforme necessário. Embora a configuração do Karpenter dessa forma permita que seu cluster seja escalado livremente, ela também pode ter implicações de custo significativas. É por esse motivo que recomendamos a configuração de alarmes de cobrança. Os alarmes de cobrança permitem que você seja alertado e notificado proativamente quando as cobranças estimadas calculadas em sua (s) conta (s) excederem um limite definido. Consulte Configurar um alarme de CloudWatch cobrança da Amazon para monitorar proativamente as cobranças estimadas para obter informações adicionais.

Você também pode habilitar a detecção de anomalias de custos, que é um recurso de gerenciamento de custos da AWS que usa aprendizado de máquina para monitorar continuamente seu custo e uso para detectar gastos incomuns. Mais informações podem ser encontradas no guia de conceitos básicos da AWS Cost Anomaly Detection. Se você chegou ao ponto de criar um orçamento nos Orçamentos da AWS, você também pode configurar uma ação para notificá-lo quando um limite específico for violado. Com as ações orçamentárias, você pode enviar um e-mail, publicar uma

mensagem em um tópico do SNS ou enviar uma mensagem para um chatbot como o Slack. Para obter mais informações, consulte Como configurar ações do AWS Budgets.

Use a do-not-disrupt anotação karpenter.sh/ para evitar que o Karpenter desprovisione um nó

Se você estiver executando um aplicativo crítico em um nó provisionado pelo Karpenter, como um trabalho em lotes de longa execução ou um aplicativo com estado, e o TTL do nó tiver expirado, o aplicativo será interrompido quando a instância for encerrada. Ao adicionar uma `karpenter.sh/do-not-disrupt` anotação ao pod, você está instruindo o Karpenter a preservar o nó até que o pod seja encerrado ou a anotação seja removida. `karpenter.sh/do-not-disrupt` Consulte a documentação de interrupção para obter mais informações.

Se os únicos pods sem daemonset restantes em um nó forem aqueles associados a trabalhos, o Karpenter poderá direcionar e encerrar esses nós, desde que o status do trabalho seja bem-sucedido ou falhe.

Configure solicitações=limites para todos os recursos que não são de CPU ao usar a consolidação

A consolidação e o agendamento em geral funcionam comparando as solicitações de recursos dos pods com a quantidade de recursos alocáveis em um nó. Os limites de recursos não são considerados. Por exemplo, pods com um limite de memória maior do que a solicitação de memória podem ultrapassar a solicitação. Se vários pods no mesmo nó estourarem ao mesmo tempo, isso pode fazer com que alguns deles sejam encerrados devido a uma condição de falta de memória (OOM). A consolidação pode aumentar a probabilidade de que isso ocorra, pois funciona para empacotar pods em nós, considerando apenas suas solicitações.

Use LimitRanges para configurar padrões para solicitações e limites de recursos

Como o Kubernetes não define solicitações ou limites padrão, o consumo de recursos do host, da CPU e da memória subjacentes por um contêiner é ilimitado. O programador do Kubernetes analisa o total de solicitações de um pod (o maior número de solicitações dos contêineres do pod ou o total de recursos dos contêineres Init do pod) para determinar em qual nó de trabalho programar o pod. Da mesma forma, o Karpenter considera as solicitações de um pod para determinar qual tipo de instância ele provisiona. Você pode usar um intervalo limite para aplicar um padrão sensato a um namespace, caso as solicitações de recursos não sejam especificadas por alguns pods.

Consulte Configurar solicitações e limites de memória padrão para um namespace

Aplique solicitações de recursos precisas a todas as cargas de trabalho

O Karpenter é capaz de lançar nós que melhor se adaptam às suas cargas de trabalho quando as informações sobre os requisitos de suas cargas de trabalho são precisas. Isso é particularmente importante ao usar o recurso de consolidação do Karpenter.

Consulte Configurar e dimensionar recursos Requests/Limits para todas as cargas de trabalho

Recomendações do CoreDNS

Atualize a configuração do CoreDNS para manter a confiabilidade

Ao implantar pods CoreDNS em nós gerenciados pelo Karpenter, dada a natureza dinâmica do Karpenter em novos nós terminating/creating rapidamente para se alinhar à demanda, é aconselhável seguir as seguintes melhores práticas:

Duração do CoreDNS lamduck

Sonda de prontidão para CoreDNS

Isso garantirá que as consultas de DNS não sejam direcionadas para um pod CoreDNS que ainda não esteja pronto ou tenha sido encerrado.

Plantas de Karpenter

Como o Karpenter adota uma abordagem que prioriza o aplicativo para provisionar a capacidade computacional para o plano de dados do Kubernetes, há cenários comuns de carga de trabalho nos quais você pode estar se perguntando como configurá-los adequadamente. O Karpenter Blueprints é um repositório que inclui uma lista de cenários comuns de carga de trabalho seguindo as melhores práticas descritas aqui. Você terá todos os recursos necessários até mesmo para criar um cluster EKS com o Karpenter configurado e testar cada um dos blueprints incluídos no repositório. Você pode combinar diferentes esquemas para finalmente criar o que você precisa para sua (s) carga (s) de trabalho.

Recursos adicionais

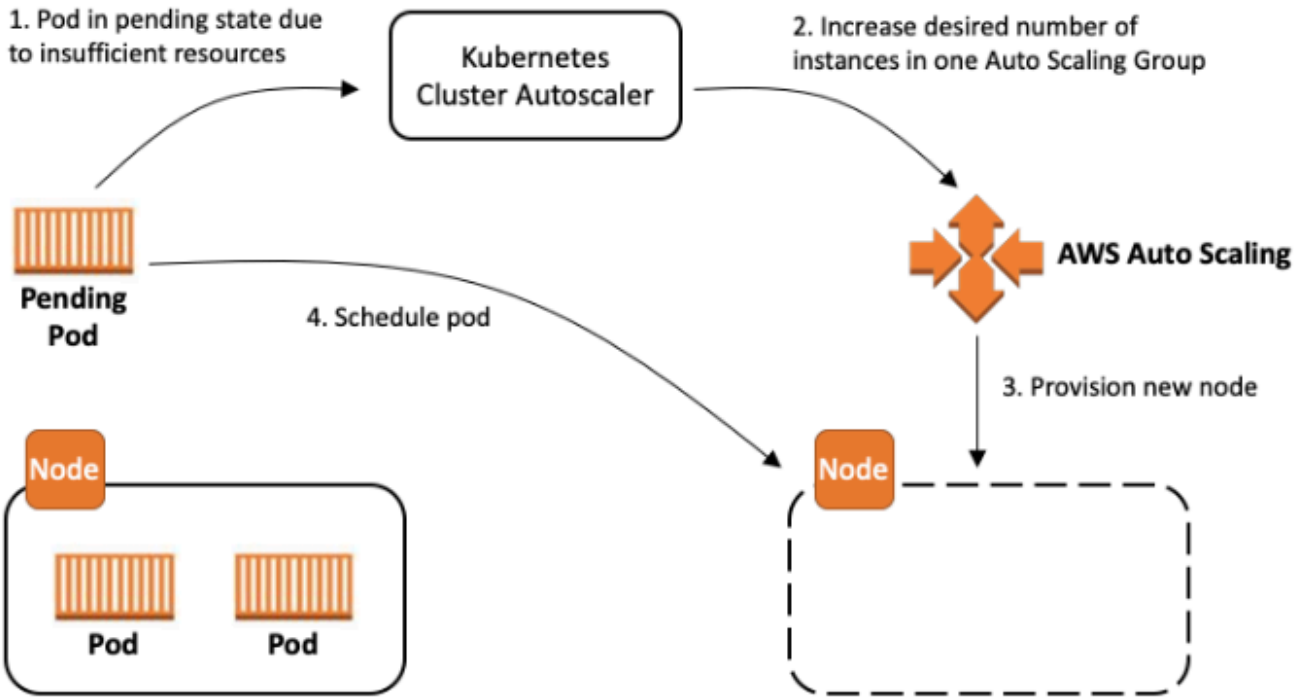
- Workshop do Dia de Imersão em Karpenter
- Workshop de otimização de custos da Karpenter
- Workshop EKS - Karpenter

- Karpenter versus Cluster Autoscaler
- Sessão Karpenter no re:Invent 2023
- Tutorial: execute clusters Kubernetes por menos com o Amazon EC2 Spot e o Karpenter

Autoscaler do cluster

Visão geral do

O Kubernetes Cluster Autoscaler é uma solução popular de escalonamento automático de clusters mantida pelo SIG Autoscaling. Ele é responsável por garantir que seu cluster tenha nós suficientes para programar seus pods sem desperdiçar recursos. Ele observa os pods que falham no agendamento e os nós que estão subutilizados. Em seguida, ele simula a adição ou remoção de nós antes de aplicar a alteração ao seu cluster. A implementação do AWS Cloud Provider no Cluster Autoscaler controla o `.DesiredReplicas` campo dos seus grupos de Auto EC2 Scaling.



Este guia fornecerá um modelo mental para configurar o autoescalador de cluster e escolher o melhor conjunto de compensações para atender aos requisitos da sua organização. Embora não exista uma única melhor configuração, há um conjunto de opções de configuração que permitem que você troque desempenho, escalabilidade, custo e disponibilidade. Além disso, este guia fornecerá dicas e melhores práticas para otimizar sua configuração para a AWS.

Glossário

A terminologia a seguir será usada com frequência em todo este documento. Esses termos podem ter um significado amplo, mas estão limitados às definições abaixo para os fins deste documento.

A escalabilidade se refere ao desempenho do autoescalador de cluster à medida que seu cluster Kubernetes aumenta em número de pods e nós. À medida que os limites de escalabilidade são atingidos, o desempenho e a funcionalidade do Cluster Autoscaler diminuem. Como o autoescalador de cluster excede seus limites de escalabilidade, ele não pode mais adicionar ou remover nós em seu cluster.

O desempenho se refere à rapidez com que o autoescalador de cluster é capaz de tomar e executar decisões de escalabilidade. Um autoescalador de cluster com desempenho perfeito tomaria instantaneamente uma decisão e acionaria uma ação de escalonamento em resposta a estímulos, como um pod se tornando não programável.

A disponibilidade significa que os pods podem ser programados rapidamente e sem interrupções. Isso inclui quando os pods recém-criados precisam ser programados e quando um nó reduzido encerra todos os pods restantes programados nele.

O custo é determinado pela decisão por trás da expansão e da escala nos eventos. Recursos são desperdiçados se um nó existente for subutilizado ou se for adicionado um novo nó muito grande para os pods de entrada. Dependendo do caso de uso, pode haver custos associados ao encerramento prematuro dos pods devido a uma decisão agressiva de redução de escala.

Grupos de nós são um conceito abstrato do Kubernetes para um grupo de nós em um cluster. Ele não é um verdadeiro recurso do Kubernetes, mas existe como uma abstração no autoescalador de cluster, na API de cluster e em outros componentes. Os nós dentro de um grupo de nós compartilham propriedades como rótulos e manchas, mas podem consistir em várias zonas de disponibilidade ou tipos de instância.

EC2 Os grupos de Auto Scaling podem ser usados como uma implementação do Node Groups on. EC2 EC2 Os grupos de Auto Scaling são configurados para iniciar instâncias que se juntam automaticamente aos clusters do Kubernetes e aplicam rótulos e contaminações ao recurso Node correspondente na API do Kubernetes.

EC2 Os grupos de nós gerenciados são outra implementação dos grupos de nós emEC2. Eles eliminam a complexidade da configuração manual dos grupos de EC2 escalabilidade automática e fornecem recursos adicionais de gerenciamento, como atualização da versão do nó e encerramento normal do nó.

Operando o escalador automático de cluster

O Cluster Autoscaler é normalmente instalado como uma implantação no cluster. Ele usa a eleição do líder para garantir alta disponibilidade, mas o trabalho é feito por uma única réplica por vez. Não é escalável horizontalmente. Para configurações básicas, o padrão deve funcionar imediatamente, usando as instruções de instalação fornecidas, mas há algumas coisas que você deve ter em mente.

Verifique se:

- A versão do Cluster Autoscaler corresponde à versão do Cluster. A compatibilidade entre versões não foi testada nem suportada.
- A descoberta automática está ativada, a menos que você tenha casos de uso avançados específicos que impeçam o uso desse modo.

Empregue o acesso menos privilegiado à função do IAM

Quando a Descoberta Automática é usada, é altamente recomendável que você empregue o acesso com privilégios mínimos limitando as Ações `autoscaling:SetDesiredCapacity` e os grupos de `autoscaling:TerminateInstanceInAutoScalingGroup` Auto Scaling que têm como escopo o cluster atual.

Isso impedirá que um autoescalador de cluster executado em um cluster modifique grupos de nós em um cluster diferente, mesmo que o `--node-group-auto-discovery` argumento não tenha sido reduzido aos grupos de nós do cluster usando tags (por exemplo). `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": "*"
        }
    ]
}
```

Configurando seus grupos de nós

O escalonamento automático eficaz começa com a configuração correta de um conjunto de grupos de nós para seu cluster. Selecionar o conjunto certo de grupos de nós é fundamental para maximizar a disponibilidade e reduzir os custos em suas cargas de trabalho. A AWS implementa grupos de nós usando grupos de EC2 Auto Scaling, que são flexíveis para um grande número de casos de uso. No entanto, o escalador automático de clusters faz algumas suposições sobre seus grupos de nós. Manter as configurações do EC2 Auto Scaling Group consistentes com essas suposições minimizará o comportamento indesejado.

Verifique se:

- Cada nó em um grupo de nós tem propriedades de agendamento idênticas, como rótulos, manchas e recursos.
  - Para MixedInstancePolicies isso, os tipos de instância devem ter o mesmo formato para CPU, memória e GPU
  - O primeiro tipo de instância especificado na política será usado para simular o agendamento.

  - Se sua política tiver tipos de instância adicionais com mais recursos, os recursos podem ser desperdiçados após a expansão horizontal.
  - Se sua política tiver tipos de instância adicionais com menos recursos, os pods podem falhar no agendamento das instâncias.
- Grupos de nós com muitos nós são preferidos em relação a muitos grupos de nós com menos nós. Isso terá o maior impacto na escalabilidade.
- Sempre que possível, prefira EC2 recursos quando os dois sistemas fornecerem suporte (por exemplo, regiões MixedInstancePolicy)

> **Note**
>
> Recomendamos usar grupos de nós gerenciados do EKS. Os grupos de nós gerenciados vêm com recursos de gerenciamento poderosos, incluindo recursos para o escalador automático de cluster, como descoberta automática de grupos de Auto EC2 Scaling e encerramento de nós sem problemas.

Otimizando o desempenho e a escalabilidade

Compreender a complexidade do tempo de execução do algoritmo de escalonamento automático ajudará você a ajustar o autoescalador de cluster para continuar operando sem problemas em grandes clusters com mais de 1.000 nós.

Os botões principais para ajustar a escalabilidade do autoescalador de cluster são os recursos fornecidos ao processo, o intervalo de varredura do algoritmo e o número de grupos de nós no cluster. Há outros fatores envolvidos na verdadeira complexidade do tempo de execução desse algoritmo, como a complexidade do plug-in de agendamento e o número de pods. Esses parâmetros são considerados não configuráveis, pois são naturais à carga de trabalho do cluster e não podem ser facilmente ajustados.

O escalador automático de cluster carrega todo o estado do cluster na memória, incluindo pods, nós e grupos de nós. Em cada intervalo de varredura, o algoritmo identifica pods não programáveis e simula o agendamento para cada grupo de nós. O ajuste desses fatores vem com diferentes compensações que devem ser cuidadosamente consideradas para seu caso de uso.

Escalonamento automático vertical do escalonador automático do cluster

A maneira mais simples de escalar o autoescalador de cluster para clusters maiores é aumentar as solicitações de recursos para sua implantação. Tanto a memória quanto a CPU devem ser aumentadas para clusters grandes, embora isso varie significativamente com o tamanho do cluster. O algoritmo de escalonamento automático armazena todos os pods e nós na memória, o que pode resultar em um espaço de memória maior que um gigabyte em alguns casos. O aumento de recursos geralmente é feito manualmente. Se você achar que o ajuste constante de recursos está criando uma carga operacional, considere usar o Addon Resizer ou o Vertical Pod Autoscaler.

Reduzindo o número de grupos de nós

Minimizar o número de grupos de nós é uma forma de garantir que o autoescalador de clusters continue funcionando bem em clusters grandes. Isso pode ser um desafio para algumas organizações que estruturam seus grupos de nós por equipe ou por aplicativo. Embora isso seja totalmente suportado pela API Kubernetes, isso é considerado um antipadrão do autoescalador de cluster com repercussões na escalabilidade. Há muitos motivos para usar vários grupos de nós (por exemplo, Spot ou GPUs), mas em muitos casos existem designs alternativos que alcançam o mesmo efeito ao usar um pequeno número de grupos.

Verifique se:

- O isolamento do pod é feito usando namespaces em vez de grupos de nós.
  - Isso pode não ser possível em clusters multilocatários de baixa confiança.
  - O pod ResourceRequests e o Pod ResourceLimits estão configurados corretamente para evitar a contenção de recursos.
  - Tipos de instância maiores resultarão em um melhor empacotamento de compartimentos e redução da sobrecarga do pod do sistema.
- NodeTaints ou NodeSelectors são usados para programar pods como exceção, não como regra.
- Os recursos regionais são definidos como um único grupo de EC2 Auto Scaling com várias zonas de disponibilidade.

Reduzindo o intervalo de varredura

Um intervalo de varredura baixo (por exemplo, 10 segundos) garantirá que o escalador automático de cluster responda o mais rápido possível quando os pods não puderem ser programados. No entanto, cada verificação resulta em muitas chamadas de API para a API Kubernetes e o EC2 Auto

Scaling Group ou o EKS Managed Node Group. APIs Essas chamadas de API podem resultar em limitação de taxa ou até mesmo indisponibilidade de serviço para seu plano de controle do Kubernetes.

O intervalo de verificação padrão é de 10 segundos, mas na AWS, a execução de um nó leva muito mais tempo para iniciar uma nova instância. Isso significa que é possível aumentar o intervalo sem aumentar significativamente o tempo geral para o aumento da escala. Por exemplo, se forem necessários 2 minutos para iniciar um nó, alterar o intervalo para 1 minuto resultará em uma troca de chamadas de API reduzidas em 6 vezes por aumentos de escala 38% mais lentos.

Fragmentação entre grupos de nós

O escalador automático de cluster pode ser configurado para operar em um conjunto específico de grupos de nós. Usando essa funcionalidade, é possível implantar várias instâncias do autoescalador de cluster, cada uma configurada para operar em um conjunto diferente de grupos de nós. Essa estratégia permite que você use números arbitrariamente grandes de grupos de nós, trocando o custo pela escalabilidade. Recomendamos usar isso apenas como último recurso para melhorar o desempenho.

O escalador automático de cluster não foi originalmente projetado para essa configuração, portanto, há alguns efeitos colaterais. Como os fragmentos não se comunicam, é possível que vários autoescaladores tentem programar um pod não programável. Isso pode resultar em escalabilidade desnecessária de vários grupos de nós. Esses nós extras voltarão a ser ampliados após `scale-down-delay` o.

```
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
```

Verifique se:

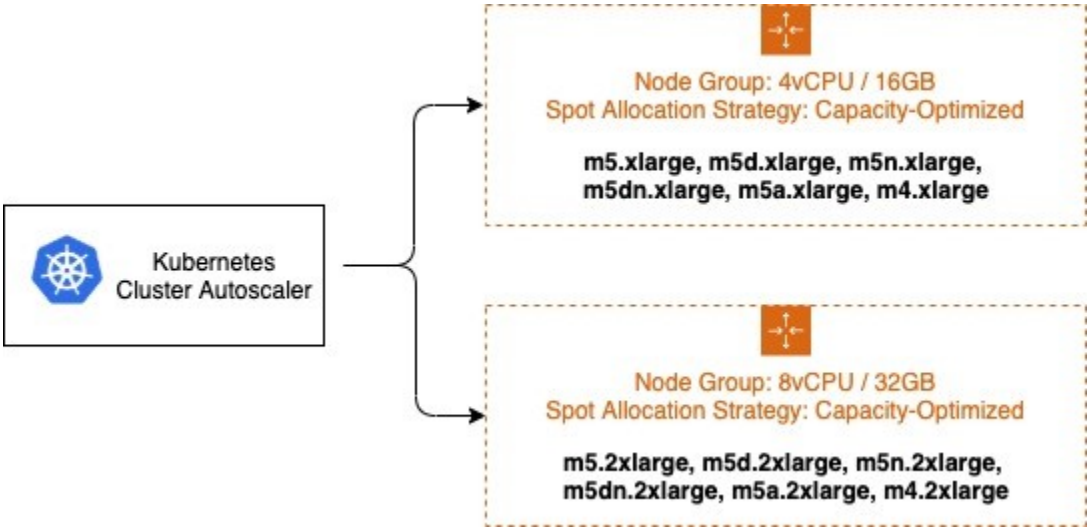
- Cada fragmento é configurado para apontar para um conjunto exclusivo de grupos de EC2 Auto Scaling
- Cada fragmento é implantado em um namespace separado para evitar conflitos na eleição do líder

Otimizando o custo e a disponibilidade

Instâncias spot

Você pode usar Instâncias Spot em seus grupos de nós e economizar até 90% do preço sob demanda. Com a desvantagem, as Instâncias Spot podem ser interrompidas a qualquer momento quando EC2 precisar recuperar a capacidade. Erros de capacidade insuficiente ocorrerão quando seu grupo de EC2 Auto Scaling não puder ser ampliado devido à falta de capacidade disponível. Maximizar a diversidade selecionando muitas famílias de instâncias pode aumentar sua chance de alcançar a escala desejada explorando vários pools de capacidade spot e diminuir o impacto das interrupções de instâncias spot na disponibilidade do cluster. Políticas de instâncias mistas com instâncias spot são uma ótima maneira de aumentar a diversidade sem aumentar o número de grupos de nós. Lembre-se de que, se você precisar de recursos garantidos, use instâncias sob demanda em vez de instâncias spot.

É fundamental que todos os tipos de instância tenham capacidade de recursos semelhante ao configurar políticas de instância mista. O simulador de agendamento do autoescalador usa o primeiro no. InstanceType MixedInstancePolicy Se os tipos de instância subsequentes forem maiores, os recursos poderão ser desperdiçados após um aumento de escala. Se forem menores, seus pods podem não ser programados nas novas instâncias devido à capacidade insuficiente. Por exemplo, todas as instâncias M4, M5, M5a e M5n têm quantidades semelhantes de CPU e memória e são ótimas candidatas para a. MixedInstancePolicy A ferramenta EC2 Instance Selector pode ajudar você a identificar tipos de instância semelhantes.



É recomendável isolar a capacidade sob demanda e spot em grupos separados de Auto EC2 Scaling. Isso é preferível ao uso de uma estratégia de capacidade básica porque as propriedades de agendamento são fundamentalmente diferentes. Como as instâncias spot são interrompidas a qualquer momento (quando é EC2 necessário recuperar a capacidade), os usuários geralmente contaminam seus nós preemptivos, exigindo uma tolerância explícita do pod ao comportamento de preempção. Essas manchas resultam em propriedades de agendamento diferentes para os nós, portanto, eles devem ser separados em vários grupos de Auto EC2 Scaling.

O escalador automático de cluster tem um conceito de expansores, que fornecem estratégias diferentes para selecionar qual grupo de nós escalar. A estratégia `--expander=least-waste` é um bom padrão de uso geral e, se você for usar vários grupos de nós para diversificar instâncias spot (conforme descrito na imagem acima), ela poderá ajudar a otimizar ainda mais os custos dos grupos de nós ao escalar o grupo que seria melhor utilizado após a atividade de escalabilidade.

Priorizando um grupo de nós/ASG

Você também pode configurar o escalonamento automático baseado em prioridade usando o expansor de prioridade. `--expander=priority`permite que seu cluster priorize um grupo de nós / ASG e, se não conseguir escalar por algum motivo, ele escolherá o próximo grupo de nós na lista priorizada. Isso é útil em situações em que, por exemplo, você deseja usar tipos de instância P3 porque a GPU deles fornece desempenho ideal para sua carga de trabalho, mas, como segunda opção, você também pode usar tipos de instância P2.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-autoscaler-priority-expander
```

```
  namespace: kube-system
data:
  priorities: |-
    10:
      - .*p2-node-group.*
    50:
      - .*p3-node-group.*
```

O Cluster Autoscaler tentará escalar o grupo Auto EC2 Scaling correspondente ao nome p3-node-group. Se essa operação não for bem-sucedida`--max-node-provision-time`, ela tentará escalar um grupo de EC2 Auto Scaling correspondente ao nome p2-node-group. Esse valor é padronizado para 15 minutos e pode ser reduzido para uma seleção de grupos de nós mais responsiva, mas se o valor for muito baixo, isso pode causar escalas desnecessárias.

Provisionamento excessivo

O escalador automático de cluster minimiza os custos ao garantir que os nós sejam adicionados ao cluster somente quando necessário e removidos quando não utilizados. Isso afeta significativamente a latência da implantação, pois muitos pods serão forçados a esperar pelo aumento da escala de um nó antes de poderem ser programados. Os nós podem levar vários minutos para ficarem disponíveis, o que pode aumentar a latência do agendamento do pod em uma ordem de grandeza.

Isso pode ser atenuado usando o provisionamento em excesso, o que troca o custo pela latência do agendamento. O superprovisionamento é implementado usando pods temporários com prioridade negativa, que ocupam espaço no cluster. Quando os pods recém-criados não puderem ser programados e tiverem maior prioridade, os pods temporários serão antecipados para liberar espaço. Os pods temporários então se tornam não programáveis, fazendo com que o autoescalador de cluster escale novos nós superprovisionados.

Há outros benefícios menos óbvios no provisionamento excessivo. Sem o provisionamento excessivo, um dos efeitos colaterais de um cluster altamente utilizado é que os pods tomarão decisões de agendamento menos ideais usando a `preferredDuringSchedulingIgnoredDuringExecution` regra de Pod ou Node Affinity. Um caso de uso comum para isso é separar pods para um aplicativo altamente disponível em todas as zonas de disponibilidade usando AntiAffinity. O provisionamento excessivo pode aumentar significativamente a chance de que um nó da zona correta esteja disponível.

A quantidade de capacidade superprovisionada é uma decisão comercial cuidadosa para sua organização. Em essência, é uma troca entre desempenho e custo. Uma forma de tomar essa decisão é determinar sua frequência média de aumento de escala e dividi-la pelo tempo

necessário para escalar um novo nó. Por exemplo, se, em média, você precisar de um novo nó a cada 30 segundos e EC2 levar 30 segundos para provisionar um novo nó, um único nó de superprovisionamento garantirá que sempre haja um nó extra disponível, reduzindo a latência do agendamento em 30 segundos ao custo de uma única instância adicional. EC2 Para melhorar as decisões de agendamento zonal, provisione em excesso um número de nós igual ao número de zonas de disponibilidade em seu Grupo de EC2 Auto Scaling para garantir que o programador possa selecionar a melhor zona para os pods de entrada.

Evite o despejo em escala reduzida

O despejo de algumas workloads são caros. A análise de big data, as tarefas de aprendizado de máquina e os executores de testes acabarão sendo concluídos, mas deverão ser reiniciados se forem interrompidos. O escalador automático de cluster tentará reduzir qualquer nó abaixo do scale-down-utilization-threshold, o que interromperá todos os pods restantes no nó. Isso pode ser evitado garantindo que os pods caros de serem despejados sejam protegidos por um rótulo reconhecido pelo escalador automático de cluster.

Verifique se:

- Cápsulas caras de despejar têm a anotação `cluster-autoscaler.kubernetes.io/safe-to-evict=false`

Casos de uso avançados

Volumes do EBS

O armazenamento persistente é essencial para criar aplicativos com bom estado, como bancos de dados ou caches distribuídos. Os volumes do EBS permitem esse caso de uso no Kubernetes, mas estão limitados a uma zona específica. Esses aplicativos podem estar altamente disponíveis se fragmentados em vários AZs usando um volume EBS separado para cada AZ. O escalonador automático de cluster pode então equilibrar o escalonamento dos grupos de escalonamento automático. EC2

Verifique se:

- O balanceamento do grupo de nós está ativado pela configuração `balance-similar-node-groups=true`.
- Os grupos de nós são configurados com configurações idênticas, exceto para diferentes zonas de disponibilidade e volumes do EBS.

Co-agendamento

Treinamentos distribuídos de machine learning se beneficiam significativamente da latência minimizada das configurações de nós da mesma zona. Essas cargas de trabalho implantam vários pods em uma zona específica. Isso pode ser feito configurando o Pod Affinity para todos os pods co-programados ou usando o Node Affinity. `topologyKey: failure-domain.beta.kubernetes.io/zone` O escalonador automático de cluster então expandirá uma zona específica para atender às demandas. Talvez você queira alocar vários grupos de EC2 Auto Scaling, um por zona de disponibilidade, para permitir o failover de toda a carga de trabalho coagendada.

Verifique se:

- O balanceamento do grupo de nós está ativado pela configuração `balance-similar-node-groups=false`
- O Node Affinity e/ou Pod Preemption é usado quando os clusters incluem grupos de nós regionais e zonais.
  - Use Node Affinity para forçar ou incentivar grupos regionais a evitar grupos de nós zonais e vice-versa.
  - Se os pods zonais forem programados em grupos de nós regionais, isso resultará em um desequilíbrio na capacidade dos pods regionais.
  - Se suas cargas de trabalho zonais puderem tolerar interrupções e realocação, configure o Pod Preemption para permitir que pods escalonados regionalmente forcem a preempção e o reagendamento em uma zona menos contestada.

Aceleradores

Alguns clusters aproveitam aceleradores de hardware especializados, como a GPU. Ao escalar para baixo, o plug-in do dispositivo acelerador pode levar vários minutos para anunciar o recurso no cluster. O escalador automático de cluster simulou que esse nó terá o acelerador, mas até que o acelerador fique pronto e atualize os recursos disponíveis do nó, os pods pendentes não podem ser programados no nó. Isso pode resultar na repetição desnecessária do aumento de escala.

Além disso, nós com aceleradores e alta utilização de CPU ou memória não serão considerados para redução de escala, mesmo que o acelerador não seja usado. Esse comportamento pode ser caro devido ao custo relativo dos aceleradores. Em vez disso, o autoescalador de cluster pode

aplicar regras especiais para considerar os nós a serem reduzidos se eles tiverem aceleradores desocupados.

Para garantir o comportamento correto nesses casos, você pode configurar o kubelet nos nós do acelerador para rotular o nó antes que ele se junte ao cluster. O autoescalador de cluster usará esse seletor de rótulos para acionar o comportamento otimizado do acelerador.

Verifique se:

- O Kubelet para nós de GPU é configurado com `--node-labels k8s.amazonaws.com/accelerator=$ACCELERATOR_TYPE`
- Os nós com aceleradores seguem a mesma regra de propriedades de agendamento mencionada acima.

Escalando a partir de 0

O Cluster Autoscaler é capaz de escalar grupos de nós de e para zero, o que pode gerar economias de custo significativas. Ele detecta os recursos de CPU, memória e GPU de um grupo de Auto Scaling inspecionando InstanceType o especificado em seu ou. LaunchConfiguration LaunchTemplate Alguns pods requerem recursos adicionais, como `WindowsENI` ou específicos, `PrivateIPv4Address` NodeSelectors ou Taints, que não podem ser descobertos no. LaunchConfiguration O escalador automático de cluster pode explicar esses fatores ao descobri-los nas tags do Auto Scaling Group. EC2 Por exemplo:

```
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**
>
> Lembre-se de que, ao escalar para zero, sua capacidade é retornada EC2 e pode ficar indisponível no futuro.

Parâmetros adicionais

Há muitas opções de configuração que podem ser usadas para ajustar o comportamento e a performance do Cluster Autoscaler. Uma lista completa de parâmetros está disponível em GitHub.

Parâmetro	Description	Padrão
intervalo de digitalização	Com que frequência o cluster é reavaliado para aumentar ou diminuir a escala	10 segundos
max-empty-bulk-delete	Número máximo de nós vazios que podem ser excluídos ao mesmo tempo.	10
scale-down-delay-after-adic ionar	Quanto tempo após a ampliação, essa avaliação de redução é retomada	10 minutos
scale-down-delay-after-excluir	Quanto tempo após a exclusão do nó, essa avaliação de redução é retomada, o padrão é o intervalo de varredura	intervalo de digitalização
scale-down-delay-after-falha	Quanto tempo após a falha na redução da escala, a avaliação da redução é retomada	3 minutos
scale-down-unneeded-time	Por quanto tempo um nó deve ser desnecessário antes de ser elegível para redução	10 minutos
scale-down-unready-time	Por quanto tempo um nó não pronto deve ser desnecess ário antes de ser elegível para redução	20 minutos

Parâmetro	Description	Padrão
scale-down-utilization-threshold	Nível de utilização do nó, definido como a soma dos recursos solicitados dividida pela capacidade, abaixo do qual um nó pode ser considerado para redução	0,5
scale-down-non-empty-candidates-count	Número máximo de nós não vazios considerados em uma iteração como candidatos à redução de escala com drenagem. Um valor menor significa melhor capacidade de resposta da CA, mas é possível reduzir a latência. Um valor mais alto pode afetar o desempenho da CA com grandes clusters (centenas de nós). Defina como um valor não positivo para desativar essa heurística - a CA não limitará o número de nós que considera. ”	30

Parâmetro	Description	Padrão
scale-down-candidates-pool-proporção	Uma proporção de nós que são considerados candidato s adicionais não vazios para redução de escala quando alguns candidatos da iteração anterior não são mais válidos. Um valor menor significa melhor capacidad e de resposta da CA, mas é possível reduzir a latência. Um valor mais alto pode afetar o desempenho da CA com grandes clusters (centenas de nós). Defina como 1,0 para desativar essa heurística. A CA considerará todos os nós como candidatos adicionais.	0.1
scale-down-candidates-pool-contagem mínima	Número mínimo de nós que são considerados candidato s adicionais não vazios para redução de escala quando alguns candidatos da iteração anterior não são mais válidos. Ao calcular o tamanho da piscina para candidato s adicionais, tomamos `max(#nodes * scale-down-candidates-pool-ratio, scale-dow n-candidates-pool-min-count)`	50

Recursos adicionais

Esta página contém uma lista de apresentações e demonstrações do Cluster Autoscaler. Se você quiser adicionar uma apresentação ou demonstração aqui, envie um pull request.

Apresentação/Demonstração	Apresentadores
Escalonamento automático e otimização de custos no Kubernetes: de 0 a 100	Guy Templeton, Skyscanner e Jiaxin Shan, Amazon
Análise aprofundada do escalonamento automático do SIG	Maciek Pytel e Marcin Wielgus

Referências

- 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/seletor de instâncias amazon-ec2
- https://github.com/aws/aws-node-termination-handler

Melhores práticas de confiabilidade

Esta seção fornece orientação sobre como tornar as cargas de trabalho em execução no EKS resilientes e altamente disponíveis

Como usar este guia

Este guia é destinado a desenvolvedores e arquitetos que desejam desenvolver e operar serviços altamente disponíveis e tolerantes a falhas no EKS. O guia está organizado em diferentes áreas temáticas para facilitar o consumo. Cada tópico começa com uma breve visão geral, seguida por uma lista de recomendações e melhores práticas para a confiabilidade de seus clusters EKS.

Introdução

As melhores práticas de confiabilidade do EKS foram agrupadas nos seguintes tópicos:

- Aplicativos
- Plano de controle
- Plano de dados

O que torna um sistema confiável? Se um sistema pode funcionar de forma consistente e atender às demandas apesar das mudanças em seu ambiente durante um período de tempo, ele pode ser chamado de confiável. Para conseguir isso, o sistema precisa detectar falhas, se curar automaticamente e ter a capacidade de escalar com base na demanda.

Os clientes podem usar o Kubernetes como base para operar aplicativos e serviços essenciais de forma confiável. Mas, além de incorporar princípios de design de aplicativos baseados em contêineres, executar cargas de trabalho de forma confiável também requer uma infraestrutura confiável. No Kubernetes, a infraestrutura compreende o plano de controle e o plano de dados.

O EKS fornece um plano de controle Kubernetes de nível de produção projetado para ser altamente disponível e tolerante a falhas.

No EKS, a AWS é responsável pela confiabilidade do plano de controle do Kubernetes. O EKS executa o plano de controle do Kubernetes em três zonas de disponibilidade em uma região da AWS. Ele gerencia automaticamente a disponibilidade e a escalabilidade dos servidores da API Kubernetes e do cluster etcd.

A responsabilidade pela confiabilidade do plano de dados é compartilhada entre você, o cliente e a AWS. O EKS oferece quatro opções de nós de trabalho para implantar o plano de dados do Kubernetes.

O EKS Auto Mode, que é a opção mais gerenciada, lida com o provisionamento, o dimensionamento e as atualizações do plano de dados, além de fornecer recursos gerenciados de computação, rede e armazenamento. O Modo Automático AMIs é lançado com frequência e os clusters são atualizados automaticamente para a AMI mais recente para implantar correções de CVE e patches de segurança. Você pode controlar quando isso ocorre configurando os controles de interrupção no Modo Automático. NodePools

O Fargate gerencia o provisionamento e a escalabilidade do plano de dados executando um pod por nó. A terceira opção, grupos de nós gerenciados, gerencia o provisionamento e as atualizações do plano de dados. E, finalmente, os nós autogerenciados são a opção menos gerenciada para o plano de dados. Quanto mais plano de dados gerenciado pela AWS você usa, menos responsabilidade você tem.

Os grupos de nós gerenciados automatizam o provisionamento e o gerenciamento do ciclo de vida dos nós. EC2 Você pode usar a API EKS (usando o console EKS, a API da AWS, a CLI da AWSCloudFormation, o Terraform ou`eksctl`) para criar, escalar e atualizar nós gerenciados. Os nós gerenciados executam EC2 instâncias do Amazon Linux 2 otimizadas para EKS em sua conta, e você pode instalar pacotes de software personalizados ativando o acesso SSH. Quando você provisiona nós gerenciados, eles são executados como parte de um grupo de Auto Scaling gerenciado pelo EKS que pode abranger várias zonas de disponibilidade; você controla isso por meio das sub-redes fornecidas ao criar nós gerenciados. O EKS também marca automaticamente os nós gerenciados para que eles possam ser usados com o escalador automático de cluster.

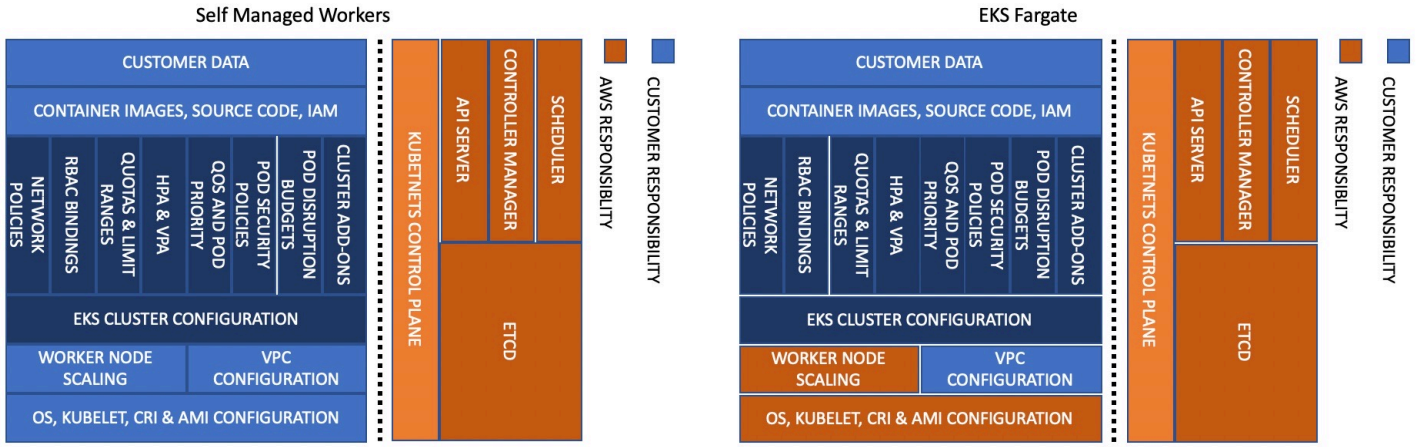
O Amazon EKS segue o modelo de responsabilidade compartilhada CVEs e os patches de segurança em grupos de nós gerenciados. Como os nós gerenciados executam o Amazon EKS otimizado, o AMIs Amazon EKS é responsável por criar versões corrigidas deles AMIs quando há correções de bugs. No entanto, você é responsável por implantar essas versões de AMI com patches nos seus grupos de nós gerenciados.

O EKS também gerencia a atualização dos nós, embora você precise iniciar o processo de atualização. O processo de atualização do nó gerenciado é explicado na documentação do EKS.

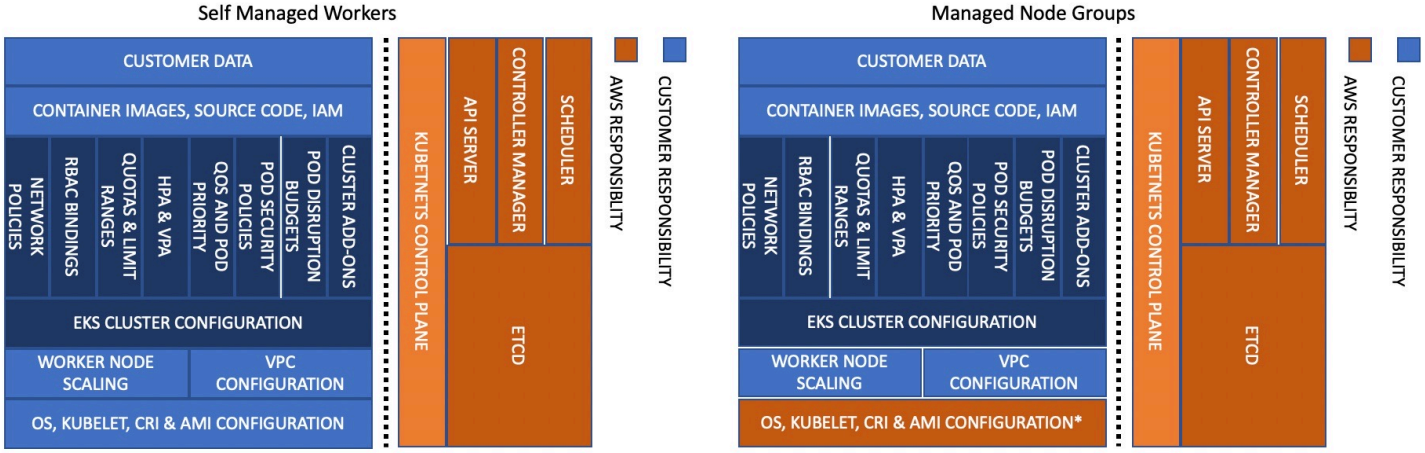
Se você executa nós autogerenciados, pode usar a AMI Linux otimizada para Amazon EKS para criar nós de trabalho. Você é responsável por aplicar patches e atualizar a AMI e os nós. É uma prática recomendada usar `eksctl` nossa infraestrutura como ferramentas de código para provisionar

nós autogerenciados, pois isso facilitará a atualização de nós autogerenciados. CloudFormation Considere migrar para novos nós ao atualizar os nós de trabalho, pois o processo de migração contamina o grupo de nós antigo `NoSchedule` e drena os nós depois que uma nova pilha está pronta para aceitar a carga de trabalho existente do pod. No entanto, você também pode realizar uma atualização local dos nós autogerenciados.

Modelo de responsabilidade compartilhada - Fargate



Modelo de Responsabilidade Compartilhada - MNG



Este guia inclui um conjunto de recomendações que você pode usar para melhorar a confiabilidade do seu plano de dados EKS, dos componentes principais do Kubernetes e dos seus aplicativos.

Feedback

Este guia está sendo lançado GitHub para coletar feedback direto e sugestões da EKS/Kubernetes comunidade em geral. Se você tem uma prática recomendada que acha que devemos incluir no guia, registre um problema ou envie um PR no GitHub repositório. Pretendemos atualizar o guia periodicamente à medida que novos recursos são adicionados ao serviço ou quando uma nova prática recomendada evolui.

Executando aplicativos de alta disponibilidade

Seus clientes esperam que seu aplicativo esteja sempre disponível, inclusive quando você está fazendo alterações e especialmente durante picos de tráfego. Uma arquitetura escalável e resiliente mantém seus aplicativos e serviços funcionando sem interrupções, o que mantém seus usuários satisfeitos. Uma infraestrutura escalável cresce e diminui com base nas necessidades da empresa. Eliminar pontos únicos de falha é uma etapa fundamental para melhorar a disponibilidade de um aplicativo e torná-lo resiliente.

Com o Kubernetes, você pode operar seus aplicativos e executá-los de forma altamente disponível e resiliente. Seu gerenciamento declarativo garante que, depois de configurar o aplicativo, o Kubernetes tente continuamente combinar o estado atual com o estado desejado.

Recomendações

Configurar orçamentos de interrupção do pod

Os orçamentos de interrupção do pod são usados para limitar a quantidade de interrupções simultâneas que um aplicativo experimentará. Eles devem ser configurados para cargas de trabalho se for importante sempre ter uma parte dessa carga de trabalho disponível. O EKS Auto Mode, o Karpenter e o Cluster Autoscaler conhecem e cumprem os orçamentos configurados de interrupção do pod ao reduzir a escala. O EKS Auto Mode, o Karpenter e os grupos de nós gerenciados também aderem aos orçamentos de interrupção do Pod ao atualizar os nós

Evite executar pods singleton

Se todo o seu aplicativo for executado em um único pod, ele ficará indisponível se esse pod for encerrado. Em vez de implantar aplicativos usando pods individuais, crie implantações. Se um pod criado por uma implantação falhar ou for encerrado, o controlador de implantação iniciará um novo pod para garantir que o número especificado de pods de réplica esteja sempre em execução.

Execute várias réplicas

A execução de várias réplicas de pods de um aplicativo usando uma implantação ajuda a executá-lo de maneira altamente disponível. Se uma réplica falhar, as réplicas restantes continuarão funcionando, embora com capacidade reduzida, até que o Kubernetes crie outro pod para compensar a perda. Além disso, você pode usar o escalonador automático do Horizontal Pod para escalar réplicas automaticamente com base na demanda da carga de trabalho.

Programe réplicas em todos os nós

A execução de várias réplicas não será muito útil se todas estiverem em execução no mesmo nó e o nó ficar indisponível. Considere usar a antiafinidade do pod ou as restrições de dispersão da topologia do pod para distribuir réplicas de uma implantação em vários nós de trabalho.

Você pode melhorar ainda mais a confiabilidade de um aplicativo típico executando-o em vários AZs.

Usando as regras de antiafinidade do Pod

O manifesto abaixo diz ao programador do Kubernetes que prefira colocar pods em nós separados e. AZs Ele não requer nós ou AZ distintos porque, se exigisse, o Kubernetes não conseguiria programar nenhum pod quando houver um pod em execução em cada AZ. Se seu aplicativo exigir apenas três réplicas, você poderá usar `requiredDuringSchedulingIgnoredDuringExecution` for`topologyKey: topology.kubernetes.io/zone`, e o programador do Kubernetes não agendará dois pods na mesma AZ.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spread-host-az
  labels:
    app: web-server
spec:
  replicas: 4
  selector:
    matchLabels:
      app: web-server
  template:
    metadata:
      labels:
        app: web-server
    spec:
      affinity:
```

```
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - web-server
              topologyKey: topology.kubernetes.io/zone
            weight: 100
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - web-server
              topologyKey: kubernetes.io/hostname
            weight: 99
      containers:
      - name: web-app
        image: nginx:1.16-alpine
```

Usando restrições de dispersão da topologia do Pod

Semelhantes às regras de antiafinidade do pod, as restrições de dispersão da topologia do pod permitem que você disponibilize seu aplicativo em diferentes domínios de falha (ou topologia), como hosts ou. AZs Essa abordagem funciona muito bem quando você está tentando garantir a tolerância a falhas e a disponibilidade por meio de várias réplicas em cada um dos diferentes domínios de topologia. As regras de antiafinidade de pods, por outro lado, podem facilmente produzir um resultado quando você tem uma única réplica em um domínio de topologia, pois os pods com antiafinidade entre si têm um efeito repelente. Nesses casos, uma única réplica em um nó dedicado não é ideal para tolerância a falhas nem é um bom uso dos recursos. Com as restrições de distribuição de topologia, você tem mais controle sobre a distribuição ou distribuição que o programador deve tentar aplicar nos domínios de topologia. Aqui estão algumas propriedades importantes a serem usadas nessa abordagem:

1. O `maxSkew` é usado para controlar ou determinar o ponto máximo em que as coisas podem ser desiguais nos domínios da topologia. Por exemplo, se um aplicativo tiver 10 réplicas e for implantado em 3 AZs, você não poderá obter uma distribuição uniforme, mas poderá influenciar

o quão desigual será a distribuição. Nesse caso, `maxSkew` pode ser qualquer coisa entre 1 e 10. Um valor de 1 significa que você pode potencialmente acabar com um spread como `4,3,3` `3,4,3` ou `3,3,4` entre 3 AZs. Em contraste, um valor de 10 significa que você pode potencialmente acabar com um spread como`10,0,0`, `0,10,0` ou `0,0,10` entre 3 AZs.

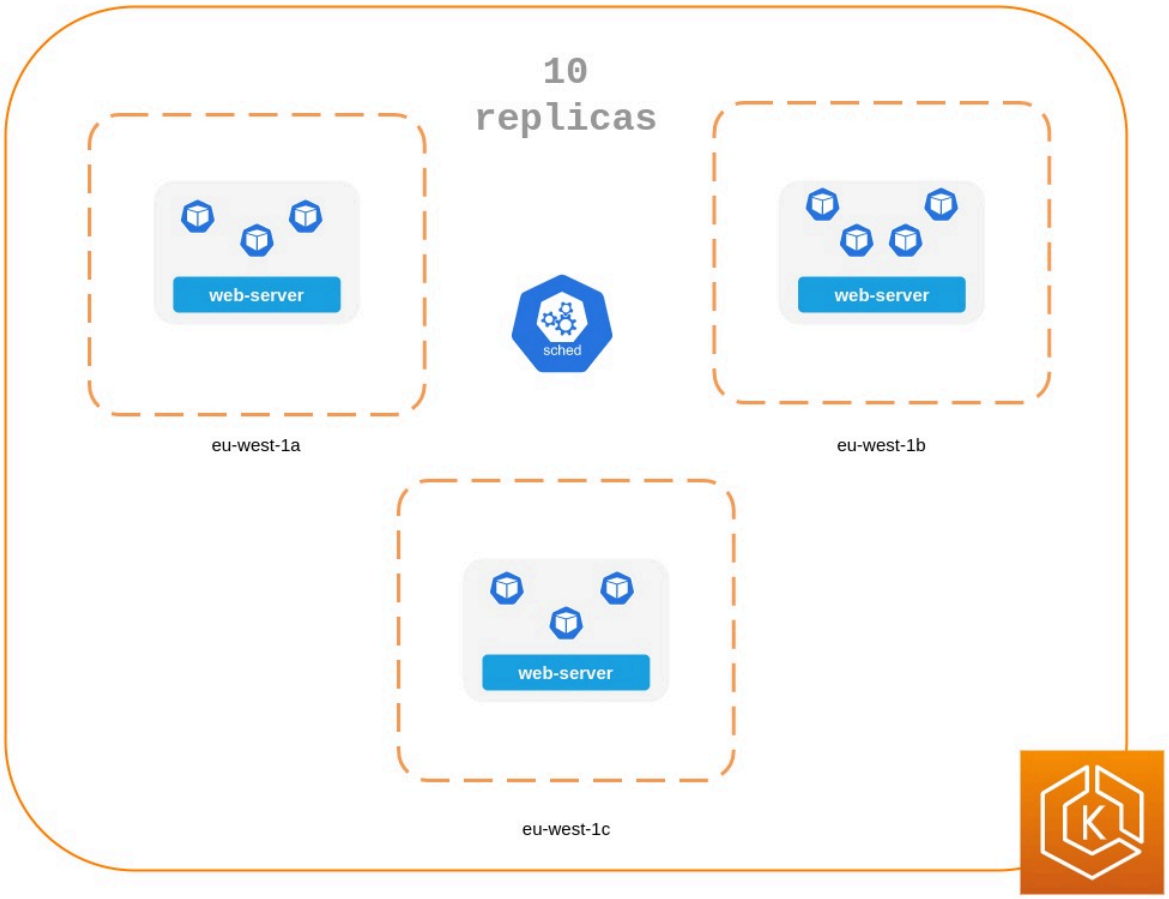
2. `topologyKey`É uma chave para um dos rótulos dos nós e define o tipo de domínio de topologia que deve ser usado para a distribuição do pod. Por exemplo, um spread zonal teria o seguinte par de valores-chave:

```
topologyKey: "topology.kubernetes.io/zone"
```

3. A `whenUnsatisfiable` propriedade é usada para determinar como você deseja que o agendador responda se as restrições desejadas não puderem ser satisfeitas.
4. O `labelSelector` é usado para encontrar pods correspondentes para que o programador possa estar ciente deles ao decidir onde colocar os pods de acordo com as restrições que você especificar.

Além dos campos acima, há outros campos sobre os quais você pode ler mais na documentação do Kubernetes.

A topologia do pod distribui as restrições em 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
```

Execute o Kubernetes Metrics Server

Instale o servidor de métricas do Kubernetes para ajudar a escalar seus aplicativos. Os complementos do autoescalador do Kubernetes, como HPA e VPA, precisam rastrear as métricas dos aplicativos para escalá-los. O servidor de métricas coleta métricas de recursos que podem ser usadas para tomar decisões de escalabilidade. As métricas são coletadas dos kubelets e veiculadas no formato da API Metrics.

O servidor de métricas não retém nenhum dado e não é uma solução de monitoramento. Seu objetivo é expor as métricas de uso da CPU e da memória a outros sistemas. Se você quiser acompanhar o estado do seu aplicativo ao longo do tempo, precisará de uma ferramenta de monitoramento como o Prometheus ou o Amazon. CloudWatch

Siga a documentação do EKS para instalar o metrics-server em seu cluster EKS.

Autoescalador horizontal de cápsulas (HPA)

A HPA pode escalar automaticamente seu aplicativo em resposta à demanda e ajudá-lo a evitar o impacto de seus clientes durante o pico de tráfego. Ele é implementado como um loop de controle no Kubernetes que consulta periodicamente métricas APIs que fornecem métricas de recursos.

O HPA pode recuperar métricas do seguinte APIs: 1. `metrics.k8s.io`também conhecida como Resource Metrics API — fornece uso de CPU e memória para pods 2. `custom.metrics.k8s.io` — Fornece métricas de outros coletores de métricas, como o Prometheus; essas métricas são internas ao seu cluster Kubernetes. 3. `external.metrics.k8s.io` — Fornece métricas externas ao seu cluster Kubernetes (por exemplo, profundidade de fila do SQS, latência do ELB).

Você deve usar um desses três APIs para fornecer a métrica para escalar seu aplicativo.

Dimensionamento de aplicativos com base em métricas personalizadas ou externas

Você pode usar métricas personalizadas ou externas para escalar seu aplicativo em métricas diferentes da utilização da CPU ou da memória. Os servidores da API Custom Metrics fornecem a `custom-metrics.k8s.io` API que a HPA pode usar para escalar aplicativos automaticamente.

Você pode usar o adaptador Prometheus para métricas do Kubernetes para coletar métricas do Prometheus e usá-las APIs com o HPA. Nesse caso, o adaptador Prometheus exporá as métricas do Prometheus no formato da API Metrics.

Depois de implantar o adaptador Prometheus, você pode consultar métricas personalizadas usando kubectl. `kubectl get –raw /apis/custom.metrics.k8s.io/v1beta1/`

As métricas externas, como o nome sugere, fornecem ao Horizontal Pod Autoscaler a capacidade de escalar implantações usando métricas externas ao cluster Kubernetes. Por exemplo, em cargas de trabalho de processamento em lote, é comum escalar o número de réplicas com base no número de trabalhos em andamento em uma fila do SQS.

Para escalonar automaticamente as cargas de trabalho do Kubernetes, você pode usar o KEDA (Kubernetes Event-driven Autoscaling), um projeto de código aberto que pode impulsionar o escalonamento de contêineres com base em vários eventos personalizados. Este blog da AWS descreve como usar o Amazon Managed Service for Prometheus para o auto-scaling da carga de trabalho do Kubernetes.

Autoescalador vertical de pods (VPA)

O VPA ajusta automaticamente a reserva de CPU e memória dos seus pods para ajudá-lo a “dimensionar corretamente” seus aplicativos. Para aplicativos que precisam ser escalados verticalmente, o que é feito aumentando a alocação de recursos, você pode usar o VPA para escalar automaticamente as réplicas do pod ou fornecer recomendações de escalabilidade.

Seu aplicativo pode ficar temporariamente indisponível se o VPA precisar escalá-lo, pois a implementação atual do VPA não realiza ajustes locais nos pods; em vez disso, ele recriará o pod que precisa ser escalado.

A documentação do EKS inclui um passo a passo para configurar o VPA.

O projeto Fairwinds Goldilocks fornece um painel para visualizar as recomendações do VPA para solicitações e limites de CPU e memória. Seu modo de atualização do VPA permite que você escale automaticamente os pods com base nas recomendações do VPA.

Atualizar aplicativos

As aplicações modernas exigem inovação rápida com alto grau de estabilidade e disponibilidade. O Kubernetes oferece as ferramentas para atualizar seus aplicativos continuamente sem interromper seus clientes.

Vejamos algumas das melhores práticas que possibilitam a rápida implantação de mudanças sem sacrificar a disponibilidade.

Tenha um mecanismo para realizar reversões

Ter um botão de desfazer pode evitar desastres. É uma prática recomendada testar as implantações em um ambiente inferior separado (ambiente de teste ou desenvolvimento) antes de atualizar o cluster de produção. Usar um pipeline de CI/CD pode ajudar você a automatizar e testar implantações. Com um pipeline de implantação contínua, você pode reverter rapidamente para a versão mais antiga se a atualização estiver com defeito.

Você pode usar Deployments para atualizar um aplicativo em execução. Isso geralmente é feito atualizando a imagem do contêiner. Você pode usar `kubectl` para atualizar uma implantação como esta:

```
kubectl --record deployment.apps/nginx-deployment set image nginx-deployment
 nginx=nginx:1.16.1
```

O `--record` argumento registra as alterações na implantação e ajuda você se você precisar realizar uma reversão. `kubectl rollout history deployment`mostra as alterações registradas nas implantações em seu cluster. Você pode reverter uma alteração usando. `kubectl rollout undo deployment <DEPLOYMENT_NAME>`

Por padrão, quando você atualiza uma implantação que exige a recriação de pods, a implantação realiza uma atualização contínua. Em outras palavras, o Kubernetes atualizará apenas uma parte dos pods em execução em uma implantação e não todos os pods de uma vez. Você pode controlar como o Kubernetes realiza atualizações contínuas por meio da propriedade. `RollingUpdateStrategy`

Ao realizar uma atualização contínua de uma implantação, você pode usar a `Max Unavailable`propriedade para especificar o número máximo de pods que podem ficar indisponíveis durante a atualização. A `Max Surge` propriedade de Deployment permite que você defina o número máximo de pods que podem ser criados acima do número desejado de pods.

Considere `max unavailable` fazer ajustes para garantir que um lançamento não atrapalhe seus clientes. Por exemplo, o Kubernetes define 25% `max unavailable` por padrão, o que significa que, se você tiver 100 pods, poderá ter apenas 75 pods trabalhando ativamente durante uma implantação. Se seu aplicativo precisar de um mínimo de 80 pods, esse lançamento pode causar interrupções. Em vez disso, você pode definir 20% `max unavailable` para garantir que haja pelo menos 80 pods funcionais durante o lançamento.

Use blue/green implantações

As mudanças são inerentemente arriscadas, mas mudanças que não podem ser desfeitas podem ser potencialmente catastróficas. Os procedimentos de alteração que permitem que você efetivamente volte no tempo por meio de uma reversão tornam os aprimoramentos e a experimentação mais seguros. Blue/green as implantações oferecem um método para retrair rapidamente as alterações se as coisas derem errado. Nessa estratégia de implantação, você cria um ambiente para a nova versão. Esse ambiente é idêntico à versão atual do aplicativo que está sendo atualizado. Depois que o novo ambiente é provisionado, o tráfego é roteado para o novo ambiente. Se a nova versão produzir os resultados desejados sem gerar erros, o ambiente antigo será encerrado. Caso contrário, o tráfego será restaurado para a versão antiga.

Você pode realizar blue/green implantações no Kubernetes criando uma nova implantação idêntica à implantação da versão existente. Depois de verificar se os pods na nova implantação estão sendo executados sem erros, você pode começar a enviar tráfego para a nova implantação alterando a `selector` especificação no serviço que direciona o tráfego para os pods do seu aplicativo.

Muitas ferramentas de integração contínua, como Flux, Jenkins e Spinnaker, permitem automatizar implantações. blue/green O blog de contêineres da AWS inclui um passo a passo usando o AWS Load Balancer Controller: Usando o AWS Load Balancer Controller blue/green para implantação, implantação canária e teste A/B

Use implantações do Canary

As implantações do Canary são uma variante das blue/green implantações que podem remover significativamente os riscos das mudanças. Nessa estratégia de implantação, você cria uma nova implantação com menos pods junto com a implantação antiga e desvia uma pequena porcentagem do tráfego para a nova implantação. Se as métricas indicarem que a nova versão está funcionando tão bem ou melhor do que a versão existente, você aumenta progressivamente o tráfego para a nova implantação enquanto o expande até que todo o tráfego seja desviado para a nova implantação. Se houver algum problema, você pode rotear todo o tráfego para a implantação antiga e parar de enviar tráfego para a nova implantação.

Embora o Kubernetes não ofereça uma maneira nativa de realizar implantações canárias, você pode usar ferramentas como o Flagger com o Istio.

Exames de saúde e autocura

Nenhum software está livre de bugs, mas o Kubernetes pode ajudar você a minimizar o impacto das falhas de software. No passado, se um aplicativo falhasse, alguém precisava remediar a situação reiniciando o aplicativo manualmente. O Kubernetes permite detectar falhas de software em seus pods e substituí-las automaticamente por novas réplicas. Com o Kubernetes, você pode monitorar a integridade de seus aplicativos e substituir automaticamente as instâncias não íntegras.

O Kubernetes oferece suporte a três tipos de verificações de integridade:

1. Sonda de vivacidade
2. Sonda de inicialização (compatível com o Kubernetes versão 1.16+)
3. Sonda de prontidão

O Kubelet, o agente do Kubernetes, é responsável por executar todas as verificações mencionadas acima. O Kubelet pode verificar a integridade de um Pods de três maneiras: o kubelet pode executar um comando shell dentro do contêiner de um Pod, enviar uma solicitação HTTP GET para o contêiner ou abrir um soquete TCP em uma porta especificada.

Se você escolher um teste `exec` baseado, que executa um script de shell dentro de um contêiner, certifique-se de que o comando shell saia antes que o `timeoutSeconds` valor expire. Caso contrário, seu nó terá `<defunct>` processos, levando à falha do nó.

Recomendações

Use o Liveness Probe para remover frutos não saudáveis

A sonda Liveness pode detectar condições de impasse em que o processo continua em execução, mas o aplicativo deixa de responder. Por exemplo, se você estiver executando um serviço web que escuta na porta 80, você pode configurar uma sonda Liveness para enviar uma solicitação HTTP GET na porta 80 do Pod. O Kubelet enviará periodicamente uma solicitação GET ao pod e esperará uma resposta; se o pod responder entre 200 e 399, o kubelet considerará que o pod está íntegro; caso contrário, o pod será marcado como não íntegro. Se um pod falhar continuamente nas verificações de integridade, o kubelet o encerrará.

Você pode usar `initialDelaySeconds` para atrasar a primeira sonda.

Ao usar o Liveness Probe, certifique-se de que seu aplicativo não se depare com uma situação em que todos os pods falhem simultaneamente no Liveness Probe, pois o Kubernetes tentará substituir todos os seus pods, o que deixará seu aplicativo offline. Além disso, o Kubernetes continuará criando novos pods que também falharão no Liveness Probes, sobrecarregando desnecessariamente o plano de controle. Evite configurar o Liveness Probe para depender de um fator externo ao seu pod, por exemplo, um banco de dados externo. Em outras palavras, um external-to-your-Pod banco de dados não responsivo não deve fazer com que seus pods falhem em suas sondas de vida.

O post LIVENESS PROBES ARE DANGEROUS, de Sandor Szücs, descreve problemas que podem ser causados por sondas mal configuradas.

Use o Startup Probe para aplicativos que demoram mais para iniciar

Quando seu aplicativo precisar de mais tempo para inicializar, você pode usar o Startup Probe para atrasar o Liveness and Readiness Probe. Por exemplo, um aplicativo Java que precisa hidratar o cache de um banco de dados pode precisar de até dois minutos antes de estar totalmente funcional. Qualquer sonda de vivacidade ou prontidão até que se torne totalmente funcional pode falhar.
A configuração de um Startup Probe permitirá que o aplicativo Java fique saudável antes que o Liveness ou o Readiness Probe sejam executados.

Até que a sonda de inicialização seja bem-sucedida, todas as outras sondas serão desativadas. Você pode definir o tempo máximo que o Kubernetes deve esperar pela inicialização do aplicativo. Se, após o tempo máximo configurado, o pod ainda falhar no Startup Probes, ele será encerrado e um novo pod será criado.

O Startup Probe é semelhante ao Liveness Probe — se eles falharem, o Pod será recriado. Como Ricardo A. explica em seu post Fantastic Probes And How To Configure Them, Startup Probes deve ser usado quando o tempo de inicialização de um aplicativo é imprevisível. Se você sabe que seu aplicativo precisa de dez segundos para ser iniciado, você deve usar o Liveness/Readiness Probe em vez disso. `initialDelaySeconds`

Use o Readiness Probe para detectar indisponibilidade parcial

Enquanto a sonda Liveness detecta falhas em um aplicativo que são resolvidas com o encerramento do pod (portanto, reiniciando o aplicativo), o Readiness Probe detecta condições em que o aplicativo pode estar temporariamente indisponível. Nessas situações, o aplicativo pode ficar temporariamente sem resposta; no entanto, espera-se que ele volte a funcionar quando a operação for concluída.

Por exemplo, durante I/O operações intensas de disco, os aplicativos podem ficar temporariamente indisponíveis para lidar com solicitações. Aqui, encerrar o pod do aplicativo não é uma solução; ao mesmo tempo, solicitações adicionais enviadas ao pod podem falhar.

Você pode usar o Readiness Probe para detectar a indisponibilidade temporária em seu aplicativo e parar de enviar solicitações ao pod até que ele volte a funcionar. Ao contrário do Liveness Probe, em que uma falha resultaria na recriação do Pod, um Readiness Probe com falha significaria que o Pod não receberia nenhum tráfego do Kubernetes Service. Quando o Readiness Probe for bem-sucedido, o Pod retomará o recebimento do tráfego do Service.

Assim como o Liveness Probe, evite configurar sondas de prontidão que dependam de um recurso externo ao pod (como um banco de dados). Aqui está um cenário em que uma prontidão mal configurada pode tornar o aplicativo não funcional: se o teste de prontidão do pod falhar quando o banco de dados do aplicativo estiver inacessível, outras réplicas do pod também falharão simultaneamente, pois compartilham os mesmos critérios de verificação de integridade. Configurar o teste dessa forma garantirá que, sempre que o banco de dados estiver indisponível, os testes de prontidão do pod falhem e o Kubernetes pare de enviar tráfego para todos os pods.

Um efeito colateral do uso de sondas de prontidão é que elas podem aumentar o tempo necessário para atualizar as implantações. As novas réplicas não receberão tráfego, a menos que as sondas de prontidão sejam bem-sucedidas; até lá, as réplicas antigas continuarão recebendo tráfego.

Lidando com interrupções

Os pods têm uma vida útil finita. Mesmo se você tiver pods de longa duração, é prudente garantir que os pods terminem corretamente quando chegar a hora. Dependendo da sua estratégia de upgrade, os upgrades de cluster do Kubernetes podem exigir que você crie novos nós de trabalho, o que exige que todos os pods sejam recriados em nós mais novos. O tratamento adequado da rescisão e os orçamentos de interrupção do pod podem ajudar você a evitar interrupções no serviço, pois os pods são removidos dos nós mais antigos e recriados nos nós mais novos.

A forma preferida de atualizar os nós de trabalho é criando novos nós de trabalho e encerrando os antigos. Antes de encerrar os nós de trabalho, você deve fazer `drain` isso. Quando um nódulo de trabalho é drenado, todos os seus frutos são despejados com segurança. Segurança é uma palavra-chave aqui; quando as cápsulas de um trabalhador são despejadas, elas não recebem simplesmente um sinal. `SIGKILL` Em vez disso, um `SIGTERM` sinal é enviado para o processo principal (PID 1) de cada contêiner nos pods que estão sendo despejados. Depois que o `SIGTERM` sinal for enviado, o Kubernetes dará ao processo algum tempo (período de carência) antes que o `SIGKILL` sinal seja

enviado. Esse período de carência é de 30 segundos por padrão; você pode substituir o padrão usando `grace-period` flag em kubectl ou declare `terminationGracePeriodSeconds` em seu Podspec.

```
kubectl delete pod <pod name> –grace-period=<seconds>
```

É comum ter contêineres nos quais o processo principal não tenha PID 1. Considere este contêiner de amostra baseado em 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
```

Neste exemplo, o script de shell recebe`SIGTERM`, o processo principal, que por acaso é um aplicativo Python neste exemplo, não recebe um `SIGTERM` sinal. Quando o Pod for encerrado, o aplicativo Python será encerrado abruptamente. Isso pode ser corrigido alterando o `ENTRYPOINT`contêiner para iniciar o aplicativo Python. Como alternativa, você pode usar uma ferramenta como o dumb-init para garantir que seu aplicativo possa lidar com sinais.

Você também pode usar ganchos de contêiner para executar um script ou uma solicitação HTTP no início ou na parada do contêiner. A ação do `PreStop` gancho é executada antes que o contêiner receba um `SIGTERM` sinal e deve ser concluída antes que esse sinal seja enviado. O `terminationGracePeriodSeconds` valor se aplica a partir do momento em que a ação do `PreStop` gancho começa a ser executada, não quando o `SIGTERM` sinal é enviado.

Recomendações

Proteja a carga de trabalho crítica com o Pod Disruption Budgets

O Pod Disruption Budget ou o PDB podem interromper temporariamente o processo de despejo se o número de réplicas de um aplicativo ficar abaixo do limite declarado. O processo de despejo continuará quando o número de réplicas disponíveis ultrapassar o limite. Você pode usar o PDB para declarar o `maxUnavailable` número `minAvailable` e o número de réplicas. Por exemplo, se você quiser que pelo menos três cópias do seu aplicativo estejam disponíveis, você pode criar um PDB.

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
```

```
  name: my-svc-pdb
spec:
  minAvailable: 3
  selector:
    matchLabels:
      app: my-svc
```

A política de PDB acima diz ao Kubernetes que interrompa o processo de despejo até que três ou mais réplicas estejam disponíveis. `PodDisruptionBudgets`Respeita a drenagem do nódulo. Durante uma atualização do grupo de nós gerenciados pelo EKS, os nós são drenados com um tempo limite de quinze minutos. Depois de quinze minutos, se a atualização não for forçada (a opção se chama Atualização contínua no console EKS), a atualização falhará. Se a atualização for forçada, os pods serão excluídos.

Para nós autogerenciados, você também pode usar ferramentas como o AWS Node Termination Handler, que garante que o plano de controle do Kubernetes responda adequadamente a eventos que podem fazer com que sua EC2 instância fique indisponível, como eventos de manutenção e interrupções spot. EC2 EC2 Ele usa a API Kubernetes para isolar o nó para garantir que nenhum novo pods seja programado e, em seguida, o drena, encerrando todos os pods em execução.

Você pode usar a anti-afinidade do Pod para programar os pods de uma implantação em nós diferentes e evitar atrasos relacionados ao PDB durante as atualizações dos nós.

Pratique engenharia do caos

A Engenharia do Caos é a disciplina de fazer experiências em um sistema distribuído para criar confiança na capacidade do sistema de suportar condições turbulentas na produção.

Em seu blog, Dominik Tornow explica que o Kubernetes é um sistema declarativo em que "o usuário fornece uma representação do estado desejado do sistema ao sistema. O sistema então considera o estado atual e o estado desejado para determinar a sequência de comandos para fazer a transição do estado atual para o estado desejado. “ Isso significa que o Kubernetes sempre armazena o estado desejado e, se o sistema se desviar, o Kubernetes tomará medidas para restaurar o estado. Por exemplo, se um nó de trabalho ficar indisponível, o Kubernetes reprogramará os pods em outro nó de trabalho. Da mesma forma, se um `replica` travar, o Deployment Controller criará um novo. `replica` Dessa forma, os controladores do Kubernetes corrigem automaticamente as falhas.

Ferramentas de engenharia de caos, como o Gremlin, ajudam você a testar a resiliência do seu cluster Kubernetes e a identificar pontos únicos de falha. Ferramentas que introduzem o caos artificial em seu cluster (e além) podem revelar pontos fracos sistêmicos, apresentar uma

oportunidade de identificar gargalos e configurações incorretas e corrigir problemas em um ambiente controlado. A filosofia da Chaos Engineering defende quebrar as coisas propositalmente e testar a infraestrutura de estresse para minimizar o tempo de inatividade imprevisto.

Use um Service Mesh

Você pode usar uma malha de serviços para melhorar a resiliência do seu aplicativo. As malhas de serviços permitem a service-to-service comunicação e aumentam a observabilidade da sua rede de microsserviços. A maioria dos produtos de service mesh funciona com um pequeno proxy de rede executado ao lado de cada serviço que intercepta e inspeciona o tráfego de rede do aplicativo. Você pode colocar seu aplicativo em uma malha sem modificá-lo. Usando os recursos integrados do proxy de serviço, você pode fazer com que ele gere estatísticas de rede, crie registros de acesso e adicione cabeçalhos HTTP às solicitações de saída para rastreamento distribuído.

Uma malha de serviços pode ajudar você a tornar seus microsserviços mais resilientes com recursos como novas tentativas automáticas de solicitações, tempos limite, interrupção de circuitos e limitação de taxa.

Se você opera vários clusters, pode usar uma malha de serviços para permitir a service-to-service comunicação entre clusters.

Malhas de serviço

- Istio
- ID vinculado
- Cônsul

Observabilidade

Observabilidade é um termo abrangente que inclui monitoramento, registro e rastreamento. Os aplicativos baseados em microsserviços são distribuídos por natureza. Ao contrário dos aplicativos monolíticos em que monitorar um único sistema é suficiente, em uma arquitetura de aplicativos distribuídos, você precisa monitorar o desempenho de cada componente. Você pode usar sistemas de monitoramento, registro e rastreamento distribuído em nível de cluster para identificar problemas em seu cluster antes que eles interrompam seus clientes.

As ferramentas integradas do Kubernetes para solução de problemas e monitoramento são limitadas. O servidor de métricas coleta métricas de recursos e as armazena na memória, mas não as

persiste. Você pode ver os registros de um pod usando kubectl, mas o Kubernetes não retém automaticamente os registros. E a implementação do rastreamento distribuído é feita no nível do código do aplicativo ou usando malhas de serviços.

A extensibilidade do Kubernetes brilha aqui. O Kubernetes permite que você traga sua solução centralizada preferida de monitoramento, registro e rastreamento.

Recomendações

Monitore seus aplicativos

O número de métricas que você precisa monitorar em aplicativos modernos está crescendo continuamente. Ajuda se você tiver uma forma automatizada de rastrear seus aplicativos para que possa se concentrar na solução dos desafios de seus clientes. Ferramentas de monitoramento em todo o cluster, como Prometheus ou CloudWatchContainer Insights, podem monitorar seu cluster e sua carga de trabalho e fornecer sinais quando, ou de preferência, antes que as coisas dêem errado.

As ferramentas de monitoramento permitem que você crie alertas que sua equipe de operações pode assinar. Considere regras para ativar alarmes para eventos que podem, quando exacerbados, levar a uma interrupção ou afetar o desempenho do aplicativo.

Se você não tiver certeza de quais métricas deve monitorar, pode se inspirar nesses métodos:

- Método RED. Significa solicitações, erros e duração.
- Método USE. Significa utilização, saturação e erros.

A postagem de Sysdig sobre as melhores práticas para alertas no Kubernetes inclui uma lista abrangente de componentes que podem afetar a disponibilidade de seus aplicativos.

Use a biblioteca de cliente Prometheus para expor as métricas do aplicativo

Além de monitorar o estado do aplicativo e agregar métricas padrão, você também pode usar a biblioteca cliente Prometheus para expor métricas personalizadas específicas do aplicativo para melhorar a observabilidade do aplicativo.

Use ferramentas de registro centralizadas para coletar e manter registros

O registro no EKS se enquadra em duas categorias: registros do plano de controle e registros do aplicativo. O registro do plano de controle do EKS fornece registros de auditoria e diagnóstico

diretamente do plano de controle para CloudWatch os registros em sua conta. Os registros do aplicativo são registros produzidos por pods executados dentro do seu cluster. Os registros de aplicativos incluem registros produzidos por pods que executam os aplicativos de lógica de negócios e os componentes do sistema Kubernetes, como CoreDNS, Cluster Autoscaler, Prometheus etc.

O EKS fornece cinco tipos de registros do plano de controle:

1. Registros de componentes do servidor da API Kubernetes
2. Auditoria
3. Autenticador
4. Gerenciador de controladores
5. Agendador

Os registros do gerenciador do controlador e do agendador podem ajudar a diagnosticar problemas no plano de controle, como gargalos e erros. Por padrão, os registros do plano de controle EKS não são enviados para CloudWatch Logs. Você pode ativar o registro do plano de controle e selecionar os tipos de registros do plano de controle do EKS que você gostaria de capturar para cada cluster em sua conta.

A coleta de registros de aplicativos requer a instalação de uma ferramenta agregadora de registros como Fluent Bit, Fluentd ou CloudWatchContainer Insights em seu cluster.

As ferramentas agregadoras de registros do Kubernetes são executadas como DaemonSets e extraem registros de contêineres dos nós. Os registros do aplicativo são então enviados para um destino centralizado para armazenamento. Por exemplo, o CloudWatch Container Insights pode usar o Fluent Bit ou o Fluentd para coletar registros e enviá-los ao CloudWatch Logs para armazenamento. O Fluent Bit e o Fluentd oferecem suporte a muitos sistemas populares de análise de registros, como Elasticsearch e InfluxDB, oferecendo a capacidade de alterar o back-end de armazenamento de seus registros modificando o Fluentbit ou a configuração de log do Fluentd.

Use um sistema de rastreamento distribuído para identificar gargalos

Um aplicativo moderno típico tem componentes distribuídos pela rede e sua confiabilidade depende do funcionamento adequado de cada um dos componentes que compõem o aplicativo. Você pode usar uma solução de rastreamento distribuído para entender como as solicitações fluem e como os sistemas se comunicam. Os rastreamentos podem mostrar onde existem gargalos na sua rede de aplicativos e evitar problemas que podem causar falhas em cascata.

Você tem duas opções para implementar o rastreamento em seus aplicativos: você pode implementar o rastreamento distribuído no nível do código usando bibliotecas compartilhadas ou usar uma malha de serviços.

Implementar o rastreamento no nível do código pode ser desvantajoso. Nesse método, você precisa fazer alterações no seu código. Isso é ainda mais complicado se você tiver aplicativos poliglotas. Você também é responsável por manter outra biblioteca em todos os seus serviços.

Os Service Meshes, como o LinkerD e o Istio, podem ser usados para implementar o rastreamento distribuído em seu aplicativo com alterações mínimas no código do aplicativo. Você pode usar o service mesh para padronizar a geração, o registro e o rastreamento de métricas.

Ferramentas de rastreamento como AWS X-Ray e Jaeger oferecem suporte a implementações de bibliotecas compartilhadas e service mesh.

Considere usar uma ferramenta de rastreamento, como o AWS X-Ray ou o Jaeger, que ofereça suporte a ambas as implementações (biblioteca compartilhada e malha de serviços), para que você não precise trocar de ferramenta se adotar o service mesh posteriormente.

Plano de controle EKS

O Amazon Elastic Kubernetes Service (EKS) é um serviço gerenciado de Kubernetes que facilita a execução do Kubernetes na AWS sem precisar instalar, operar e manter seu próprio plano de controle do Kubernetes ou nós de trabalho. Ele executa o Kubernetes upstream e é certificado em conformidade com o Kubernetes. Essa conformidade garante que o EKS ofereça suporte ao KubernetesAPIs, assim como a versão de código aberto da comunidade que você pode instalar no local ou no local. EC2 Os aplicativos existentes em execução no Kubernetes upstream são compatíveis com o Amazon EKS.

O EKS gerencia automaticamente a disponibilidade e a escalabilidade dos nós do plano de controle do Kubernetes e substitui automaticamente os nós não íntegros do plano de controle.

Arquitetura EKS

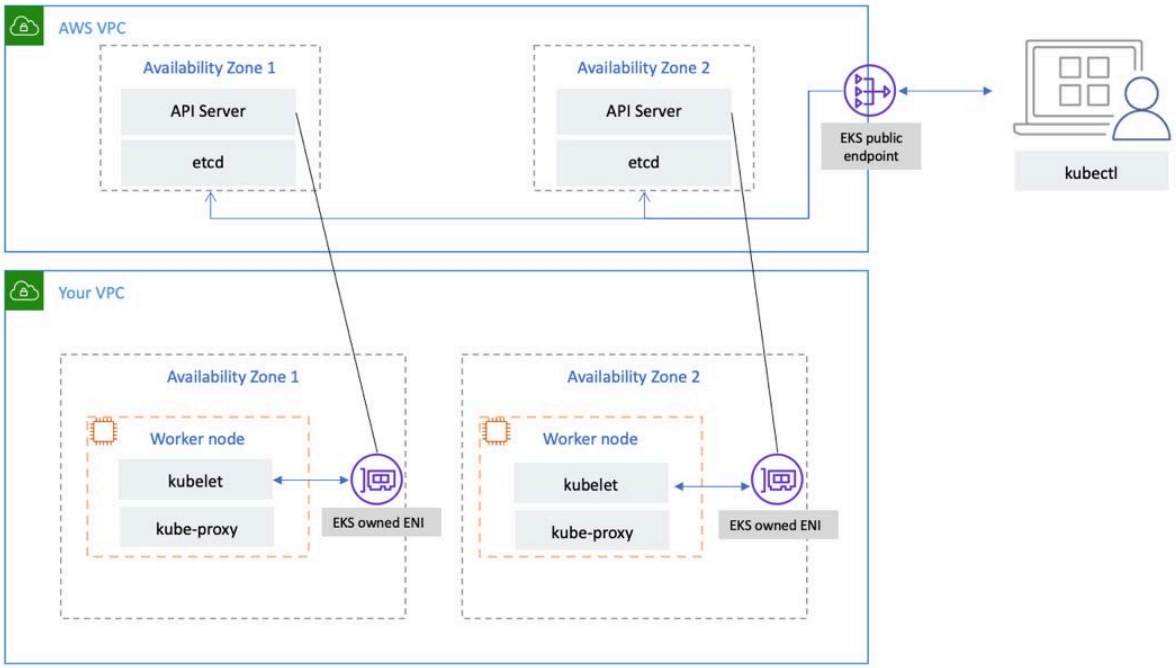
A arquitetura EKS foi projetada para eliminar qualquer ponto único de falha que possa comprometer a disponibilidade e a durabilidade do plano de controle do Kubernetes.

O plano de controle do Kubernetes gerenciado pelo EKS é executado dentro de uma VPC gerenciada pelo EKS. O plano de controle do EKS compreende os nós do servidor da API

Kubernetes, o cluster etcd. Nós de servidor da API Kubernetes que executam componentes como o servidor da API, o agendador e são executados `kube-controller-manager` em um grupo de auto-scaling. O EKS executa no mínimo dois nós de servidor de API em zonas de disponibilidade distintas (AZs) dentro da região da AWS. Da mesma forma, para maior durabilidade, os nós do servidor etcd também são executados em um grupo de auto-scaling que abrange três. AZs O EKS executa um NAT Gateway em cada AZ, e os servidores API e etcd são executados em uma sub-rede privada. Essa arquitetura garante que um evento em uma única AZ não afete a disponibilidade do cluster EKS.

Quando você cria um novo cluster, o Amazon EKS cria um endpoint altamente disponível para o servidor gerenciado da API Kubernetes que você usa para se comunicar com seu cluster (usando ferramentas como). `kubectl` O endpoint gerenciado usa o NLB para balancear a carga dos servidores da API Kubernetes. O EKS também provisiona dois ENIs diferentes AZs para facilitar a comunicação com seus nós de trabalho.

Conectividade de rede do plano de dados EKS



Você pode configurar se o servidor de API do seu cluster Kubernetes pode ser acessado pela Internet pública (usando o endpoint público) ou por meio de sua VPC (usando o EKS gerenciado) ou ambos. ENIs

Independentemente de os usuários e os nós de trabalho se conectarem ao servidor da API usando o endpoint público ou o ENI gerenciado pelo EKS, há caminhos redundantes para conexão.

Recomendações

Analise as recomendações a seguir.

Monitore as métricas do plano de controle

O monitoramento das métricas da API Kubernetes pode fornecer informações sobre o desempenho do plano de controle e identificar problemas. Um plano de controle não íntegro pode comprometer a disponibilidade das cargas de trabalho em execução dentro do cluster. Por exemplo, controladores mal escritos podem sobrecarregar os servidores da API, afetando a disponibilidade do seu aplicativo.

O Kubernetes expõe as métricas do plano de controle no endpoint. `/metrics`

Você pode ver as métricas expostas usando`kubectl`:

```
kubectl get --raw /metrics
```

Essas métricas são representadas em um formato de texto do Prometheus.

Você pode usar o Prometheus para coletar e armazenar essas métricas. Em maio de 2020, CloudWatch foi adicionado suporte para monitorar as métricas CloudWatch do Prometheus no Container Insights. Portanto, você também pode usar CloudWatch a Amazon para monitorar o plano de controle do EKS. Você pode usar o Tutorial para adicionar um novo Prometheus Scrape Target: Prometheus KPI Server Metrics para coletar métricas CloudWatch e criar um painel para monitorar o plano de controle do seu cluster.

Você pode encontrar as métricas do servidor da API Kubernetes aqui. Por exemplo, `apiserver_request_duration_seconds` pode indicar quanto tempo as solicitações de API estão demorando para serem executadas.

Considere monitorar essas métricas do plano de controle:

Servidor da API

Métrica	Description
`apiserver_request_total`	Contador de solicitações do apiserver divididas para cada verbo, valor de simulação, grupo,

Métrica	Description
	versão, recurso, escopo, componente e código de resposta HTTP.
`apiserver_request_duration_seconds*`	Histograma de latência de resposta em segundos para cada verbo, valor de execução seca, grupo, versão, recurso, sub-recurso, escopo e componente.
`apiserver_admission_controller_admission_duration_seconds*`	Histograma de latência do controlador de admissão em segundos, identificado por nome e dividido para cada operação e recurso e tipo de API (validar ou admitir).
`apiserver_admission_webhook_rejection_count`	Contagem de rejeições de webhook de admissão. Identificado por nome, operação, código_de rejeição, tipo (validando ou admitido), tipo_erro (calling_webhook_error, apiserver_internal_error, no_error)
`rest_client_request_duration_seconds*`	Solicite o histograma de latência em segundos. Dividido por verbo e URL.
`rest_client_requests_total`	Número de solicitações HTTP, particionadas por código de status, método e host.

- As métricas do histograma incluem os sufixos _bucket, _sum e _count.

etc.

Métrica	Descri on
`etcd_request_duration_seconds*`	Histog a de

Métrica	Descri on
	latênci da solicita ão Etcd em segun para cada operaç e tipo de objeto.
`apiserver_storage_db_total_size_in_bytes` ou `apiserver_storage_size_bytes` (começando com EKS v1.28)	Tamar do banco de dados Etcd.

- As métricas do histograma incluem os sufixos _bucket, _sum e _count.

Considere usar o Painel de Visão Geral do Monitoramento do Kubernetes para visualizar e monitorar as solicitações do servidor da API Kubernetes e as métricas de latência e latência etcd.

> **Important**
>
> Quando o limite de tamanho do banco de dados é excedido, o etcd emite um alarme sem espaço e para de receber mais solicitações de gravação. Em outras palavras, o cluster se torna somente para leitura e todas as solicitações para alterar objetos, como criar novos pods, escalar implantações etc., serão rejeitadas pelo servidor de API do cluster.

autenticação de cluster

Atualmente, o EKS oferece suporte a dois tipos de autenticação: tokens de portador/conta de serviço e autenticação IAM, que usa autenticação de token de webhook. Quando os usuários chamam a API Kubernetes, um webhook passa um token de autenticação incluído na solicitação para o IAM. O token, uma URL assinada de base 64, é gerado pela AWS Command Line Interface (AWS CLI).

O usuário ou a função do IAM que cria o EKS Cluster obtém automaticamente acesso total ao cluster. Você pode gerenciar o acesso ao cluster EKS editando o configmap aws-auth.

Se você configurar incorretamente o `aws-auth` configmap e perder o acesso ao cluster, ainda poderá usar o usuário ou a função do criador do cluster para acessar seu cluster EKS.

No caso improvável de você não poder usar o serviço IAM na região da AWS, você também pode usar o token portador da conta de serviço do Kubernetes para gerenciar o cluster.

Crie uma `super-admin` conta que tenha permissão para realizar todas as ações no cluster:

```
kubectl -n kube-system create serviceaccount super-admin
```

Crie uma associação de função que dê ao superadmin o papel cluster-admin:

```
kubectl create clusterrolebinding super-admin-rb --clusterrole=cluster-admin --
serviceaccount=kube-system:super-admin
```

Veja o segredo da conta de serviço:

```
SECRET_NAME=`kubectl -n kube-system get serviceaccount/super-admin -o
 jsonpath='{.secrets[0].name}'`
```

Obtenha o token associado ao segredo:

```
TOKEN=`kubectl -n kube-system get secret $SECRET_NAME -o jsonpath='{.data.token}'|
 base64 --decode`
```

Adicione conta de serviço e token a`kubeconfig`:

```
kubectl config set-credentials super-admin --token=$TOKEN
```

Defina o contexto atual `kubeconfig` para usar a conta de superadministrador:

```
kubectl config set-context --current --user=super-admin
```

A final `kubeconfig` deve ficar assim:

```
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 admissão

O Kubernetes tem dois tipos de webhooks de admissão: webhooks de admissão de validação e webhooks de admissão mutantes. Isso permite que um usuário estenda a API kubernetes e valide ou altere objetos antes que eles sejam aceitos pela API. Configurações inadequadas desses webhooks podem desestabilizar o plano de controle do EKS ao bloquear as operações críticas do cluster.

Para evitar o impacto das operações críticas do cluster, evite configurar webhooks “abrangentes”, como os seguintes:

```
- name: "pod-policy.example.com"
  rules:
  - apiGroups:   ["*"]
    apiVersions: ["*"]
    operations:  ["*"]
    resources:   ["*"]
    scope: "*"
```

Ou certifique-se de que o webhook tenha uma política de abertura de falhas com um tempo limite inferior a 30 segundos para garantir que, se o webhook não estiver disponível, ele não prejudique as cargas de trabalho críticas do cluster.

Bloqueie pods com opções inseguras **`sysctls`**

`Sysctl`é um utilitário Linux que permite aos usuários modificar os parâmetros do kernel durante o tempo de execução. Esses parâmetros do kernel controlam vários aspectos do comportamento do sistema operacional, como rede, sistema de arquivos, memória virtual e gerenciamento de processos.

O Kubernetes permite atribuir `sysctl` perfis para pods. O Kubernetes é classificado `systcls` como seguro e inseguro. Os Safe `sysctls` têm namespaces no contêiner ou no pod, e configurá-los não afeta outros pods no nó ou no próprio nó. Por outro lado, os sysctls inseguros são desativados por padrão, pois podem potencialmente interromper outros pods ou tornar o nó instável.

Como os inseguros `sysctls` estão desativados por padrão, o kubelet não criará um pod com perfil inseguro. `sysctl` Se você criar um pod desse tipo, o programador atribuirá repetidamente esses pods aos nós, enquanto o nó não conseguirá iniciá-lo. Esse loop infinito acaba sobrecarregando o plano de controle do cluster, tornando o cluster instável.

Considere usar o OPA Gatekeeper ou o Kyverno para rejeitar pods inseguros. `sysctls`

Lidando com atualizações de clusters

Desde abril de 2021, o ciclo de lançamento do Kubernetes foi alterado de quatro lançamentos por ano (uma vez por trimestre) para três lançamentos por ano. Uma nova versão secundária (como 1. 21 ou 1. 22) é lançado aproximadamente a cada quinze semanas. A partir do Kubernetes 1.19, cada versão secundária é suportada por aproximadamente doze meses após seu lançamento. Com

o advento do Kubernetes v1.28, a distorção de compatibilidade entre o plano de controle e os nós de trabalho se expandiu de n-2 para n-3 versões secundárias. Para saber mais, consulte Melhores práticas para atualizações de clusters.

Conectividade de endpoint de cluster

Ao trabalhar com o Amazon EKS (Elastic Kubernetes Service), você pode encontrar tempos limite de conexão ou erros durante eventos como escalabilidade ou aplicação de patches no plano de controle do Kubernetes. Esses eventos podem fazer com que as instâncias do kube-apiserver sejam substituídas, resultando potencialmente no retorno de endereços IP diferentes ao resolver o FQDN. Este documento descreve as melhores práticas para os consumidores da Kubernetes API manterem uma conectividade confiável.

> **Note**
>
> A implementação dessas melhores práticas pode exigir atualizações nas configurações ou scripts do cliente para lidar com novas estratégias de reresolução e repetição de DNS de forma eficaz.

O principal problema decorre do cache do lado do cliente do DNS e do potencial de endereços IP obsoletos do endpoint EKS - NLB público para endpoint público ou X-ENI para endpoint privado. Quando as instâncias do kube-apiserver são substituídas, o nome de domínio totalmente qualificado (FQDN) pode ser resolvido para novos endereços IP. No entanto, devido às configurações de DNS Time to Live (TTL), que são definidas para 60 segundos na zona do Route 53 gerenciada pela AWS, os clientes podem continuar usando endereços IP desatualizados por um curto período de tempo.

Para mitigar esses problemas, os consumidores da API Kubernetes (como kubectl, CI/CD pipelines e aplicativos personalizados) devem implementar as seguintes práticas recomendadas:

- Implemente a reresolução de DNS
- Implemente novas tentativas com Backoff e Jitter. Por exemplo, veja este artigo intitulado Falhas acontecem
- Implemente tempos limite do cliente. Defina tempos limite apropriados para evitar que solicitações de longa duração bloqueiem seu aplicativo. Lembre-se de que algumas bibliotecas de cliente do Kubernetes, especialmente aquelas geradas por geradores OpenAPI, podem não permitir a configuração fácil de tempos limite personalizados.
  - Exemplo 1 com kubectl:

```
kubectl get pods --request-timeout 10s # default: no timeout
```

- Exemplo 2 com Python: o cliente Kubernetes fornece um parâmetro _request_timeout

Ao implementar essas melhores práticas, você pode melhorar significativamente a confiabilidade e a resiliência de seus aplicativos ao interagir com a API Kubernetes. Lembre-se de testar essas implementações minuciosamente, especialmente sob condições de falha simuladas, para garantir que elas se comportem conforme o esperado durante eventos reais de escalabilidade ou correção.

Executando grandes clusters

O EKS monitora ativamente a carga nas instâncias do plano de controle e as dimensiona automaticamente para garantir alto desempenho. No entanto, você deve considerar possíveis problemas e limites de desempenho no Kubernetes e nas cotas nos serviços da AWS ao executar grandes clusters.

- Clusters com mais de 1000 serviços podem apresentar latência de rede com o uso `kube-proxy` no `iptables` modo de acordo com os testes realizados pela ProjectCalico equipe. A solução é mudar para o `ipvs`modo `kube-proxy` de execução.
- Você também pode enfrentar a limitação de solicitações de EC2 API se a CNI precisar solicitar endereços IP para pods ou se você precisar criar novas instâncias com frequência. EC2 Você pode reduzir a EC2 API de chamadas configurando a CNI para armazenar endereços IP em cache. Você pode usar tipos de EC2 instância maiores para reduzir os eventos EC2 de escalabilidade.

Recursos adicionais:

- Desmistificando a rede de clusters para nós de trabalho do Amazon EKS
- Controle de acesso ao endpoint do cluster Amazon EKS
- AWS re:Invent 2019: Amazon EKS nos bastidores (-R1) CON421

Plano de dados EKS

Para operar aplicativos de alta disponibilidade e resiliência, você precisa de um plano de dados altamente disponível e resiliente. Um plano de dados elástico garante que o Kubernetes possa

escalar e corrigir seus aplicativos automaticamente. Um plano de dados resiliente consiste em dois ou mais nós de trabalho, pode crescer e diminuir com a carga de trabalho e se recuperar automaticamente de falhas.

Você tem várias opções para nós de trabalho com o EKS: nós gerenciados no Modo Automático do EKS, EC2 Instâncias e Fargate.

O EKS Auto Mode oferece o caminho mais fácil para um plano de dados resiliente. O Modo Automático estende o gerenciamento de clusters Kubernetes pela AWS além do próprio cluster, para permitir que a AWS também configure e gerencie a infraestrutura que permite a operação tranquila de suas cargas de trabalho. O Modo Automático aumenta ou diminui automaticamente o plano de dados à medida que o Kubernetes escala os pods e trabalha para garantir continuamente que os nós em seu cluster sejam dimensionados de forma adequada e econômica para as cargas de trabalho em execução no momento.

Se você escolher EC2 instâncias, você mesmo poderá gerenciar os nós de trabalho ou usar grupos de nós gerenciados pelo EKS. Você pode ter um cluster com uma combinação de modo automático, nós de trabalho gerenciados e autogerenciados e Fargate.

O Fargate executa cada pod em um ambiente computacional isolado. Cada pod executado no Fargate tem seu próprio nó de trabalho. O Fargate escala automaticamente o plano de dados à medida que o Kubernetes escala os pods. Você pode escalar o plano de dados e sua carga de trabalho usando o autoescalador horizontal do pod.

A forma preferida de escalar os nós de EC2 trabalho (se não estiver usando o EKS Auto Mode, onde isso é executado automaticamente pela AWS) é usando grupos Karpenter, Kubernetes Cluster Autoscaler ou Auto Scaling. EC2

Recomendações

Distribua os nós de trabalho e as cargas de trabalho em vários AZs

Você pode proteger suas cargas de trabalho contra falhas em uma AZ individual executando nós de trabalho e pods em vários. AZs Você pode controlar a AZ na qual os nós de trabalho são criados usando as sub-redes nas quais você cria os nós.

O método recomendado para distribuir pods AZs é usar restrições de dispersão de topologia para pods. Recursos de escalonamento automático, como o EKS Auto Mode e o Karpenter, estão cientes das restrições de dispersão da topologia e iniciarão automaticamente os Nodes da maneira correta AZs para permitir que suas restrições sejam atendidas.

A implantação abaixo distribui os pods, AZs se possível, permitindo que esses pods funcionem de qualquer maneira, caso contrário:

```
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`só conhece os domínios de topologia por meio de nós que existem com esses rótulos. Se a implantação acima for implantada em um cluster com nós somente em uma única zona, todos os pods serão programados nesses nós, pois `kube-scheduler` não conhece as outras zonas. Para que essa distribuição de topologia funcione conforme o esperado com o agendador, os nós já devem existir em todas as zonas. A `minDomains` propriedade de restrições de dispersão de topologia é usada para informar o agendador

sobre o número de domínios elegíveis, mesmo se houver um Node em execução para evitar esse problema.

> **Warning**
>
> `whenUnsatisfiable`Definir como `DoNotSchedule` fará com que os pods não sejam programáveis se a restrição de dispersão da topologia não puder ser atendida. Ele só deve ser definido se for preferível que os pods não sejam executados em vez de violar a restrição de dispersão da topologia.

Nas versões mais antigas do Kubernetes, você pode usar regras de antiafinidade de pods para programar pods em vários. AZs O manifesto abaixo informa o programador do Kubernetes a preferir o agendamento de pods de forma distinta. 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**
>
> Não exija que os pods sejam programados de AZs forma distinta, caso contrário, o número de pods em uma implantação nunca excederá o número de. AZs

Garanta a capacidade de iniciar nós em cada AZ ao usar volumes do EBS

Se você usa o Amazon EBS para fornecer volumes persistentes, precisará garantir que os pods e o volume associado do EBS estejam localizados na mesma AZ. Um pod não pode acessar volumes persistentes suportados pelo EBS localizados em uma AZ diferente. O programador do Kubernetes sabe em qual AZ um nó de trabalho está localizado a partir dos rótulos que estão no nó e sempre agendará um pod que exija um volume do EBS no mesmo AZ do volume. No entanto, se não houver nós de trabalho disponíveis na AZ em que o volume está localizado, o pod não pode ser programado.

Se estiver usando o EKS Auto Mode ou o Karpenter, você precisará garantir que suas sub-redes sejam NodeClass selecionadas em cada AZ. Se estiver usando grupos de nós gerenciados, você precisa garantir que tenha um grupo de nós em cada AZ.

Um recurso de armazenamento do EBS é incorporado ao EKS Auto Mode, mas se estiver usando Karpenter ou Managed Node Groups, o EBS CSI também precisará ser instalado.

Use o EKS Auto Mode para gerenciar nós de trabalho

O EKS Auto Mode simplifica o gerenciamento do EKS fornecendo clusters prontos para produção com o mínimo de sobrecarga operacional. O modo automático é responsável por aumentar ou diminuir o número de nós, dependendo dos pods em execução no cluster. Os nós são atualizados automaticamente com patches e correções de software, com as atualizações sendo realizadas de acordo com as configurações de NodePoolinterrupção definidas e os orçamentos de interrupção do pod.

Execute o Node Monitoring Agent

O Node Monitoring Agent monitora e reage aos problemas de saúde do Node publicando eventos do Kubernetes e atualizando a condição de status nos Nodes. O agente de monitoramento de nós está incluído nos nós do modo automático do EKS e pode ser instalado como um complemento do EKS para nós que não são gerenciados pelo modo automático.

O EKS Auto Mode, o Managed Node Groups e o Karpenter têm a capacidade de detectar condições fatais de Node relatadas pelo Node Monitoring Agent e reparar esses Nodes automaticamente quando essas condições ocorrerem.

Implemente QoS

Para aplicações críticas, considere definir `requests` = `limits` para o contêiner no pod. Isso garantirá que o contêiner não seja eliminado se outro pod solicitar recursos.

É uma prática recomendada implementar limites de CPU e memória para todos os contêineres, pois isso evita que um contêiner consuma inadvertidamente os recursos do sistema e afete a disponibilidade de outros processos co-localizados.

Configurar e dimensionar recursos Requests/Limits para todas as cargas de trabalho

Algumas orientações gerais podem ser aplicadas ao dimensionamento de solicitações de recursos e limites para cargas de trabalho:

- Não especifique limites de recursos na CPU. Na ausência de limites, a solicitação atua como um peso sobre quanto tempo relativo de CPU os contêineres recebem. Isso permite que suas cargas de trabalho usem toda a CPU sem limites artificiais ou inanição.
- Para recursos que não são de CPU, configurar `requests` = `limits` fornece o comportamento mais previsível. Se`requests`! `=limits`, o contêiner também tem sua QOS reduzida de Garantida para Estável, aumentando a probabilidade de ser despejado em caso de pressão no nó.
- Para recursos que não sejam de CPU, não especifique um limite muito maior do que a solicitação. Quanto maiores `limits` forem configurados em relação a`requests`, maior a probabilidade de os nós ficarem sobrecarregados, levando a grandes chances de interrupção da carga de trabalho.
- Solicitações dimensionadas corretamente são particularmente importantes ao usar uma solução de auto-escalonamento de nós, como Karpenter ou Cluster. AutoScaler Essas ferramentas analisam suas solicitações de carga de trabalho para determinar o número e o tamanho dos nós a serem provisionados. Se suas solicitações forem muito pequenas com limites maiores, você

poderá descobrir que suas cargas de trabalho foram removidas ou o OOM eliminado se estiverem compactadas em um nó.

Determinar as solicitações de recursos pode ser difícil, mas ferramentas como o Vertical Pod Autoscaler podem ajudar você a "dimensionar corretamente" as solicitações observando o uso dos recursos do contêiner em tempo de execução. Outras ferramentas que podem ser úteis para determinar o tamanho das solicitações incluem:

- Cachinhos Dourados
- Parca
- Perfis
- rsg

Configurar cotas de recursos para namespaces

Namespaces são destinados para o uso em ambientes com muitos usuários distribuídos entre várias equipes ou projetos. Eles fornecem um escopo para nomes e são uma forma de dividir os recursos do cluster entre várias equipes, projetos e cargas de trabalho. Você pode limitar o consumo agregado de recursos em um namespace. O `ResourceQuota`objeto pode limitar a quantidade de objetos que podem ser criados em um namespace por tipo, bem como a quantidade total de recursos computacionais que podem ser consumidos pelos recursos desse projeto. Você pode limitar a soma total dos recursos de and/or computação de armazenamento (CPU e memória) que podem ser solicitados em um determinado namespace.

Se a cota de recursos estiver habilitada para um namespace para recursos computacionais, como CPU e memória, os usuários devem especificar solicitações ou limites para cada contêiner nesse namespace.

Considere configurar cotas para cada namespace. Considere usar `LimitRanges` para aplicar automaticamente limites pré-configurados aos contêineres em um namespace.

Limitar o uso de recursos do contêiner em um namespace

As cotas de recursos ajudam a limitar a quantidade de recursos que um namespace pode usar. O `LimitRange`objeto pode ajudá-lo a implementar os recursos mínimos e máximos que um contêiner pode solicitar. Usando `LimitRange` você pode definir uma solicitação e limites padrão para contêineres, o que é útil se definir limites de recursos computacionais não for uma prática padrão

em sua organização. Como o nome sugere, `LimitRange` pode impor o uso mínimo e máximo de recursos computacionais por pod ou contêiner em um namespace. Além disso, imponha a solicitação mínima e máxima de armazenamento PersistentVolumeClaim em um namespace.

Considere usar `LimitRange` em conjunto com `ResourceQuota` para impor limites em um nível de contêiner e namespace. Definir esses limites garantirá que um contêiner ou namespace não interfira nos recursos usados por outros locatários no cluster.

Use NodeLocal DNSCache

Você pode melhorar o desempenho do Cluster DNS executando NodeLocalDNSCache. Esse recurso executa um agente de cache de DNS nos nós do cluster como um. DaemonSet Todos os pods usam o agente de cache DNS em execução no nó para resolução de nomes em vez de usar o Service. `kube-dns` Esse recurso é incluído automaticamente no EKS Auto Mode.

Configurar o CoreDNS de escalonamento automático

Outro método para melhorar o desempenho do Cluster DNS é habilitar o auto-scaling integrado dos pods CoreDNS.

Esse recurso monitora continuamente o estado do cluster, incluindo o número de nós e núcleos de CPU. Com base nessas informações, o controlador adaptará dinamicamente o número de réplicas da implantação do CoreDNS em um cluster do EKS.

Práticas recomendadas para redes

É fundamental entender a rede Kubernetes para operar seu cluster e seus aplicativos com eficiência. A rede Pod, também chamada de rede de cluster, é o centro da rede Kubernetes. O Kubernetes é compatível com plug-ins Container Network Interface (CNI) para redes de clusters.

O Amazon EKS oferece suporte oficial ao plug-in CNI da Amazon Virtual Private Cloud (VPC) para implementar a rede Kubernetes Pod. A VPC CNI fornece integração nativa com a AWS VPC e funciona no modo subjacente. No modo subjacente, os pods e os hosts estão localizados na mesma camada de rede e compartilham o namespace da rede. O endereço IP do pod é consistente do ponto de vista do cluster e da VPC.

Este guia apresenta a Amazon VPC Container Network Interface (VPC CNI) no contexto da rede de clusters Kubernetes. O VPC CNI é o plug-in de rede padrão compatível com o EKS e, portanto, é o foco do guia. O VPC CNI é altamente configurável para oferecer suporte a diferentes casos de uso. Este guia inclui ainda seções dedicadas sobre diferentes casos de uso, modos operacionais e subcomponentes do VPC CNI, seguidos pelas recomendações.

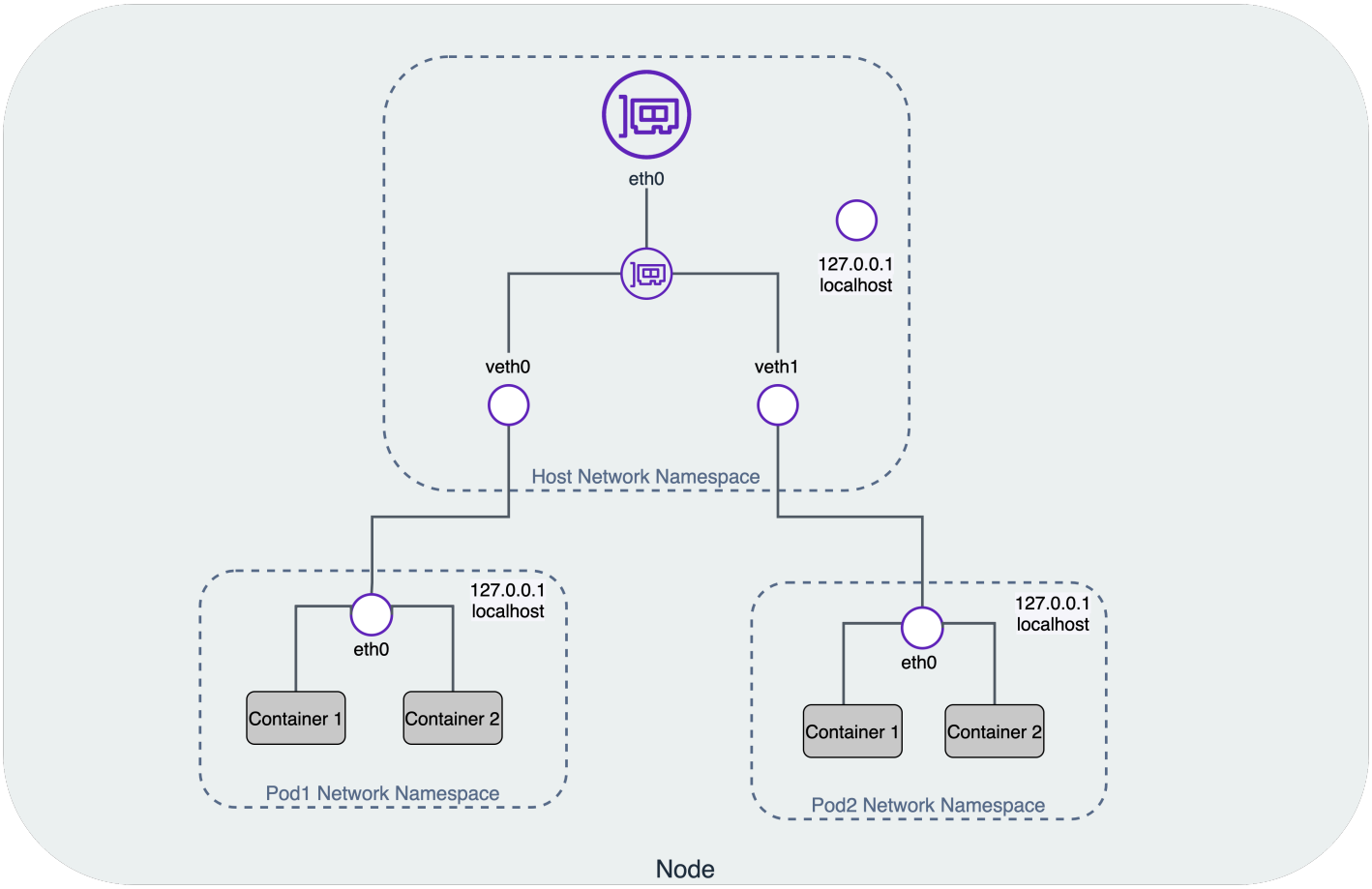
O Amazon EKS executa o Kubernetes upstream e é certificado em conformidade com o Kubernetes. Embora você possa usar plug-ins CNI alternativos, este guia não fornece recomendações para gerenciar alternativas CNIs. Consulte a documentação do EKS Alternate CNI para obter uma lista de parceiros e recursos para gerenciar alternativas de CNIs forma eficaz.

Modelo de rede Kubernetes

O Kubernetes define os seguintes requisitos em redes de cluster:

- Os pods programados no mesmo nó devem ser capazes de se comunicar com outros pods sem usar NAT (Network Address Translation).
- Todos os daemons do sistema (processos em segundo plano, por exemplo, kubelet) em execução em um determinado nó podem se comunicar com os pods em execução no mesmo nó.
- Os pods que usam a rede host devem ser capazes de entrar em contato com todos os outros pods em todos os outros nós sem usar o NAT.

Consulte o modelo de rede Kubernetes para obter detalhes sobre o que o Kubernetes espera de implementações de rede compatíveis. A figura a seguir ilustra a relação entre os namespaces da rede do pod e o namespace da rede do host.



Interface de rede de contêineres (CNI)

O Kubernetes oferece suporte às especificações e plug-ins da CNI para implementar o modelo de rede Kubernetes. Uma CNI consiste em uma especificação (versão atual 1.0.0) e bibliotecas para escrever plug-ins para configurar interfaces de rede em contêineres, junto com vários plug-ins compatíveis. A CNI se preocupa apenas com a conectividade de rede dos contêineres e com a remoção dos recursos alocados quando o contêiner é excluído.

O plug-in CNI é ativado passando ao kubelet a opção de linha de comando. `--network-plugin=cni` O Kubelet lê um arquivo de `--cni-conf-dir` (padrão etc/cni/net /.d) e usa a configuração CNI desse arquivo para configurar a rede de cada pod. O arquivo de configuração CNI deve corresponder à especificação CNI (mínimo v0.4.0) e todos os plug-ins CNI necessários referenciados pela configuração devem estar presentes no `--cni-bin-dir` diretório (padrão/). opt/cni/bin Se houver vários arquivos de configuração CNI no diretório, o kubelet usa o arquivo de configuração que vem primeiro pelo nome em ordem lexicográfica.

Amazon Virtual Private Cloud (VPC) CNI

A VPC CNI fornecida pela AWS é o complemento de rede padrão para clusters EKS. O complemento VPC CNI é instalado por padrão quando você provisiona clusters EKS. O VPC CNI é executado em nós de trabalho do Kubernetes. O complemento VPC CNI consiste no binário CNI e no plug-in IP Address Management (ipamd). A CNI atribui um endereço IP da rede VPC a um pod. O ipamd gerencia as AWS Elastic Networking Interfaces (ENIs) para cada nó do Kubernetes e mantém o pool aquecido de. IPs O VPC CNI fornece opções de configuração para pré-alocação ENIs e endereços IP para tempos rápidos de inicialização do pod. Consulte a CNI da Amazon VPC para ver as melhores práticas recomendadas de gerenciamento de plug-ins.

O Amazon EKS recomenda que você especifique sub-redes em pelo menos duas zonas de disponibilidade ao criar um cluster. O Amazon VPC CNI aloca endereços IP para pods a partir das sub-redes dos nós. É altamente recomendável verificar as sub-redes para ver se há endereços IP disponíveis. Considere as recomendações de VPC e sub-rede antes de implantar clusters EKS.

O Amazon VPC CNI aloca um pool quente de endereços IP ENIs secundários da sub-rede conectada à ENI primária do nó. Esse modo de VPC CNI é chamado de modo IP secundário. O número de endereços IP e, portanto, o número de pods (densidade de pods) são definidos pelo número ENIs e pelo endereço IP por ENI (limites), conforme definido pelo tipo de instância. O modo secundário é o padrão e funciona bem para clusters pequenos com tipos de instância menores. Considere usar o modo de prefixo se você estiver enfrentando desafios de densidade de pods. Você também pode aumentar os endereços IP disponíveis no nó para pods atribuindo prefixos a. ENIs

O Amazon VPC CNI se integra de forma nativa com o AWS VPC e permite que os usuários apliquem as melhores práticas existentes de rede e segurança do AWS VPC para criar clusters Kubernetes. Isso inclui a capacidade de usar registros de fluxo de VPC, políticas de roteamento de VPC e grupos de segurança para isolamento de tráfego de rede. Por padrão, a CNI da Amazon VPC aplica o grupo de segurança associado à ENI primária no nó aos pods. Considere ativar grupos de segurança para pods quando quiser atribuir regras de rede diferentes para um pod.

Por padrão, a VPC CNI atribui endereços IP aos pods da sub-rede atribuída à ENI primária de um nó. É comum sentir falta de IPv4 endereços ao executar grandes clusters com milhares de cargas de trabalho. O AWS VPC permite que você estenda a disponibilidade IPs atribuindo um secundário CIDRs para contornar o esgotamento dos blocos CIDR. IPv4 O AWS VPC CNI permite que você use um intervalo CIDR de sub-rede diferente para pods. Esse recurso da VPC CNI é chamado de rede personalizada. Você pode considerar o uso de redes personalizadas para usar `100.64.0.0/10`

e `198.19.0.0/16` CIDRs (CG-NAT) com o EKS. Isso permite que você crie efetivamente um ambiente em que os pods não consumam mais nenhum RFC1918 endereço IP da sua VPC.

A rede personalizada é uma opção para resolver o problema de esgotamento do IPv4 endereço, mas exige sobrecarga operacional. Recomendamos IPv6 clusters em redes personalizadas para resolver esse problema. Especificamente, recomendamos migrar para IPv6 clusters se você tiver esgotado completamente todo o espaço de IPv4 endereço disponível para sua VPC. Avalie os planos de suporte IPv6 da sua organização e considere se investir neles IPv6 pode ter mais valor a longo prazo.

O suporte do EKS para IPv6 está focado em resolver o problema de exaustão de IP causado por um espaço de IPv4 endereço limitado. Em resposta aos problemas de IPv4 exaustão dos clientes, a EKS priorizou IPv6 apenas pods em vez de pods de pilha dupla. Ou seja, os pods podem acessar IPv4 recursos, mas não recebem um IPv4 endereço do intervalo CIDR da VPC. A VPC CNI atribui IPv6 endereços aos pods do bloco CIDR da VPC gerenciado pela AWS. IPv6

Calculadora de sub-rede

Este projeto inclui um documento Excel de calculadora de sub-rede. Este documento de calculadora simula o consumo de endereço IP de uma carga de trabalho especificada em diferentes opções de configuração de ENI, como e. `WARM_IP_TARGET` `WARM_ENI_TARGET` O documento inclui duas folhas, uma primeira para o modo Warm ENI e uma segunda para o modo Warm IP. Consulte a orientação do VPC CNI para obter mais informações sobre esses modos.

Entradas:

- Tamanho CIDR da sub-rede
- Alvo ENI quente ou alvo IP quente
- Lista de instâncias
  - tipo, número e número de pods de carga de trabalho programados por instância

Saídas:

- Número total de pods hospedados
- Número de sub-rede consumida IPs
- Número de sub-rede restante IPs
- Detalhes do nível da instância

- Número de Warm IPs/ENIs por instância
- Número de ativos IPs/ENIs por instância

Considerações sobre VPC e sub-rede

Operar um cluster EKS requer conhecimento da rede VPC da AWS, além da rede Kubernetes.

Recomendamos que você entenda os mecanismos de comunicação do plano de controle do EKS antes de começar a projetar sua VPC ou implantar clusters em clusters existentes. VPCs

Consulte as considerações do Cluster VPC e as considerações do grupo de segurança do Amazon EKS ao arquitetar uma VPC e sub-redes para serem usadas com o EKS.

Visão geral do

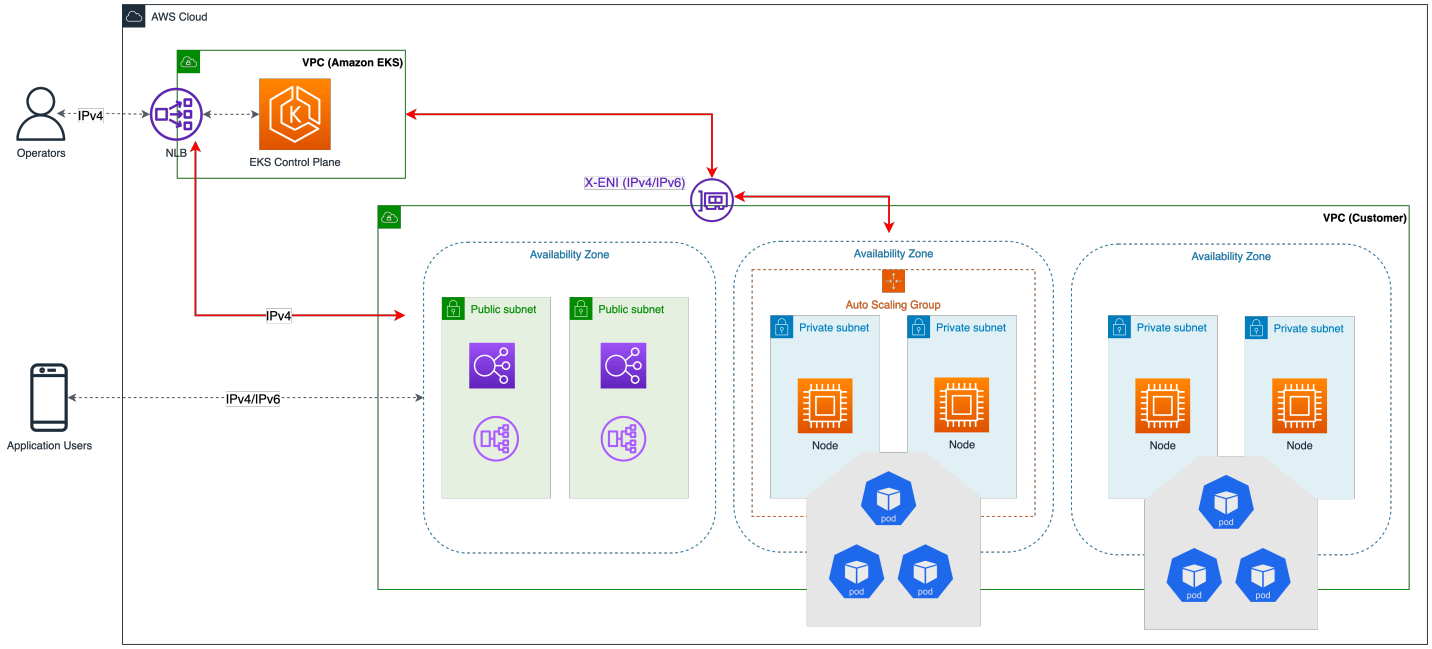
Arquitetura de cluster EKS

Um cluster EKS consiste em dois VPCs:

- Uma VPC gerenciada pela AWS que hospeda o plano de controle do Kubernetes. Essa VPC não aparece na conta do cliente.
- Uma VPC gerenciada pelo cliente que hospeda os nós do Kubernetes. É aqui que os contêineres são executados, assim como outras infraestruturas da AWS gerenciadas pelo cliente, como balanceadores de carga usados pelo cluster. Essa VPC aparece na conta do cliente. Você precisa criar uma VPC gerenciada pelo cliente antes de criar um cluster. O eksctl cria uma VPC se você não fornecer uma.

Os nós na VPC do cliente precisam da capacidade de se conectar ao endpoint do servidor de API gerenciado na VPC da AWS. Isso permite que os nós se registrem no plano de controle do Kubernetes e recebam solicitações para executar pods de aplicativos.

Os nós se conectam ao plano de controle do EKS por meio (a) de um endpoint público do EKS ou (b) de interfaces de rede elástica entre contas (X-ENI) gerenciadas pelo EKS. Quando um cluster é criado, você precisa especificar pelo menos duas sub-redes VPC. O EKS coloca um X-ENI em cada sub-rede especificada durante a criação do cluster (também chamadas de sub-redes de cluster). O servidor da API Kubernetes usa essas contas cruzadas ENIs para se comunicar com os nós implantados nas sub-redes VPC do cluster gerenciado pelo cliente.



Quando o nó é iniciado, o script de bootstrap do EKS é executado e os arquivos de configuração do nó do Kubernetes são instalados. Como parte do processo de inicialização em cada instância, os agentes de tempo de execução do contêiner, o kubelet e os agentes do nó do Kubernetes são iniciados.

Para registrar um nó, o Kubelet entra em contato com o endpoint do cluster Kubernetes. Ele estabelece uma conexão com o endpoint público fora da VPC ou com o endpoint privado dentro da VPC. O Kubelet recebe instruções de API e fornece atualizações de status e pulsações para o endpoint regularmente.

Comunicação do plano de controle EKS

O EKS tem duas maneiras de controlar o acesso ao endpoint do cluster. O controle de acesso ao endpoint permite que você escolha se o endpoint pode ser acessado pela Internet pública ou somente por meio de sua VPC. Você pode ativar o endpoint público (que é o padrão), o endpoint privado ou ambos ao mesmo tempo.

A configuração do endpoint da API do cluster determina o caminho que os nós percorrem para se comunicar com o plano de controle. Observe que essas configurações de endpoint podem ser alteradas a qualquer momento por meio do console EKS ou da API.

Endpoint público

Esse é o comportamento padrão para novos clusters do Amazon EKS. Quando somente o endpoint público do cluster está habilitado, as solicitações da API Kubernetes que se originam da VPC do seu cluster (como o nó de trabalho para controlar a comunicação do plano) saem da VPC, mas não da rede da Amazon. Para que os nós se conectem ao plano de controle, eles devem ter um endereço IP público e uma rota para um gateway da Internet ou uma rota para um gateway NAT, onde possam usar o endereço IP público do gateway NAT.

Endpoint público e privado

Quando os endpoints públicos e privados estão habilitados, as solicitações da API Kubernetes de dentro da VPC se comunicam com o plano de controle por meio do X- dentro da sua VPC. ENIs O servidor de API do cluster é acessível pela internet.

Endpoint privado

Não há acesso público ao seu servidor de API pela Internet quando somente o endpoint privado está ativado. Todo o tráfego para o servidor de API do cluster deve vir da VPC do cluster ou de uma rede conectada. Os nós se comunicam com o servidor da API via X- ENIs em sua VPC. Observe que as ferramentas de gerenciamento de cluster devem ter acesso ao endpoint privado. Saiba mais sobre como se conectar a um endpoint privado do cluster Amazon EKS de fora da Amazon VPC.

Observe que o endpoint do servidor de API do cluster é resolvido por servidores DNS públicos em um endereço IP privado da VPC. No passado, o endpoint só podia ser resolvido a partir da VPC.

Configurações de VPC

Suporte IPv4 e endereçamento do Amazon VPC. IPv6 O Amazon EKS oferece suporte IPv4 por padrão. Uma VPC deve ter um bloco IPv4 CIDR associado a ela. Opcionalmente, você pode associar vários blocos de roteamento entre domínios IPv4 sem classe (CIDR) e vários blocos CIDR à IPv6 sua VPC. Ao criar uma VPC, você deve especificar um bloco IPv4 CIDR para a VPC a partir dos intervalos de IPv4 endereços privados, conforme especificado na RFC 1918. O tamanho de bloco permitido está entre um `/16` prefixo (65.536 endereços IP) e um `/28` prefixo (16 endereços IP).

Ao criar uma nova VPC, você pode anexar um único bloco IPv6 CIDR e até cinco ao alterar uma VPC existente. O comprimento do prefixo do tamanho do bloco IPv6 CIDR pode estar entre /44 e /60 e, para as IPv6 sub-redes, pode estar entre /44/ e /64. Você pode solicitar um bloco IPv6 CIDR do pool de IPv6 endereços mantido pela Amazon. Consulte a seção Blocos CIDR da VPC do Guia do usuário da VPC para obter mais informações.

Os clusters do Amazon EKS são compatíveis com IPv4 IPv6 e. Por padrão, os clusters EKS usam IPv4 IP. Especificar IPv6 no momento da criação do cluster permitirá o uso de IPv6 clusters. IPv6 os clusters exigem pilha dupla VPCs e sub-redes.

O Amazon EKS recomenda que você use pelo menos duas sub-redes que estejam em zonas de disponibilidade diferentes durante a criação do cluster. As sub-redes que você passa durante a criação do cluster são conhecidas como sub-redes de cluster. Quando você cria um cluster, o Amazon EKS cria até 4 contas cruzadas (x-account ou x-ENIs) ENIs nas sub-redes que você especifica. Os x- ENIs são sempre implantados e usados para tráfego de administração de clusters, como entrega de logs, exec e proxy. Consulte o guia do usuário do EKS para obter detalhes completos dos requisitos de VPC e sub-rede.

Os nós de trabalho do Kubernetes podem ser executados nas sub-redes do cluster, mas isso não é recomendado. Durante as atualizações do cluster, o Amazon EKS fornece provisões adicionais ENIs nas sub-redes do cluster. Quando seu cluster se expande, os nós e pods de trabalho podem consumir o disponível IPs na sub-rede do cluster. Portanto, para garantir que haja disponibilidade IPs suficiente, convém considerar o uso de sub-redes de cluster dedicadas com máscara de rede /28.

Os nós de trabalho do Kubernetes podem ser executados em uma sub-rede pública ou privada. O fato de uma sub-rede ser pública ou privada se refere ao fato de o tráfego dentro da sub-rede ser roteado por meio de um gateway da Internet. As sub-redes públicas têm uma entrada na tabela de rotas para a Internet por meio do gateway da Internet, mas as sub-redes privadas não.

O tráfego que se origina em outro lugar e chega aos seus nós é chamado de entrada. O tráfego que se origina dos nós e sai da rede é chamado de saída. Os nós com endereços IP públicos ou elásticos (EIPs) em uma sub-rede configurada com um gateway de internet permitem a entrada de fora da VPC. As sub-redes privadas geralmente têm um roteamento para um gateway NAT, que não permite tráfego de entrada para os nós nas sub-redes de fora da VPC e, ao mesmo tempo, permite que o tráfego dos nós saia da VPC (saída).

No IPv6 mundo, todo endereço é roteável pela Internet. Os IPv6 endereços associados aos nós e pods são públicos. As sub-redes privadas são suportadas pela implementação de gateways de Internet somente de saída (EIGW) em uma VPC, permitindo tráfego de saída e bloqueando todo o tráfego de entrada. As melhores práticas para implementar IPv6 sub-redes podem ser encontradas no guia do usuário da VPC.

Você pode configurar a VPC e as sub-redes de três maneiras diferentes:

Usando somente sub-redes públicas

Nas mesmas sub-redes públicas, tanto os nós quanto os recursos de entrada (como balanceadores de carga) são criados. Marque a sub-rede pública com `kubernetes.io/role/elb`para criar balanceadores de carga voltados para a Internet. Nessa configuração, o endpoint do cluster pode ser configurado para ser público, privado ou ambos (público e privado).

Usando sub-redes públicas e privadas

Os nós são criados em sub-redes privadas, enquanto os recursos do Ingress são instanciados em sub-redes públicas. Você pode habilitar o acesso público, privado ou ambos (público e privado) ao endpoint do cluster. Dependendo da configuração do endpoint do cluster, o tráfego do nó entrará pelo gateway NAT ou pelo ENI.

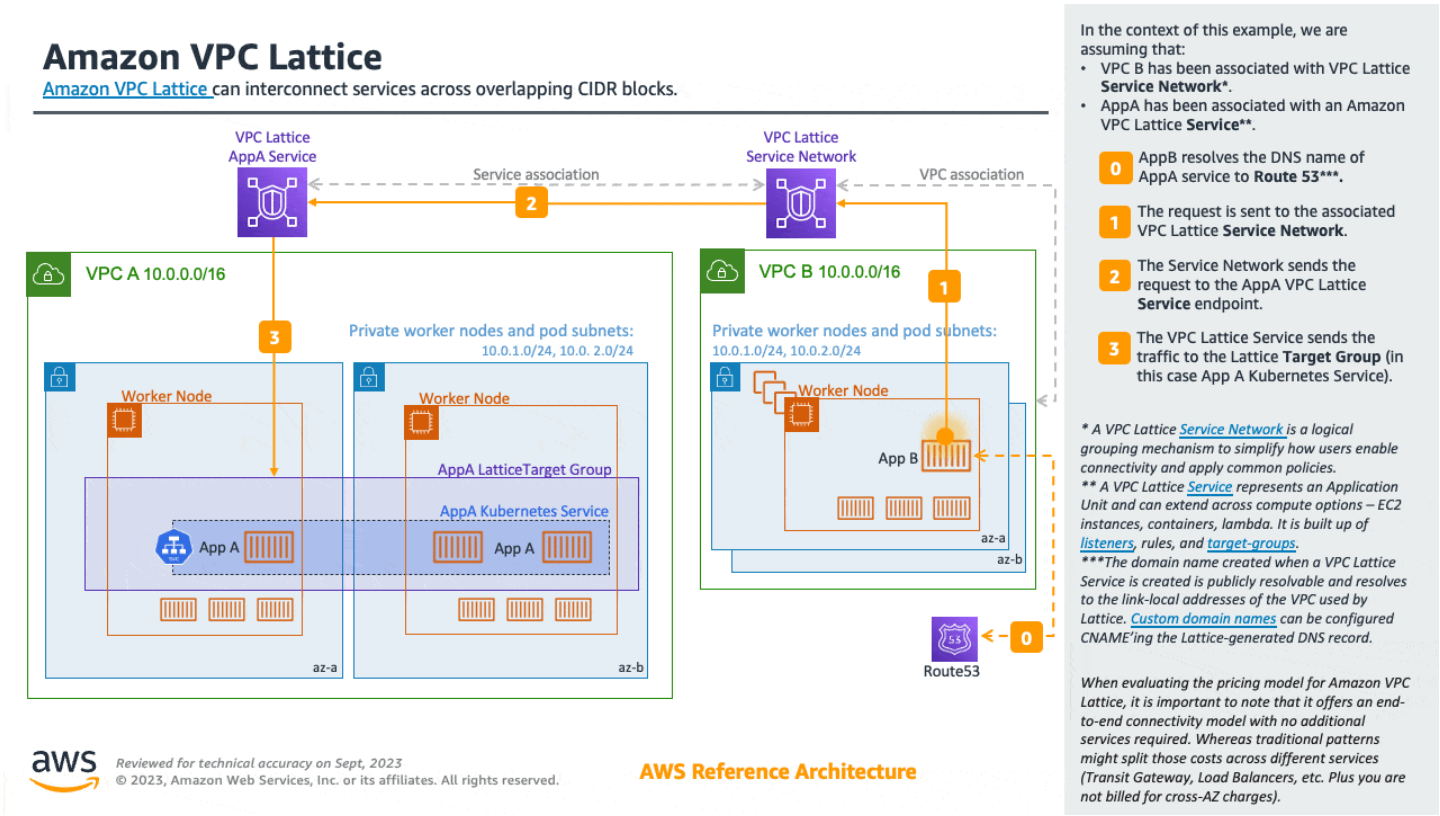
Usando somente sub-redes privadas

Tanto os nós quanto a entrada são criados em sub-redes privadas. Usando a tag de `kubernetes.io/role/internal-elb`sub-rede para criar balanceadores de carga internos. O acesso ao endpoint do seu cluster exigirá uma conexão VPN. Você deve ativar a AWS PrivateLink para todos EC2 os repositórios Amazon ECR e S3. Somente o endpoint privado do cluster deve ser ativado. Sugerimos que você analise os requisitos de cluster privado do EKS antes de provisionar clusters privados.

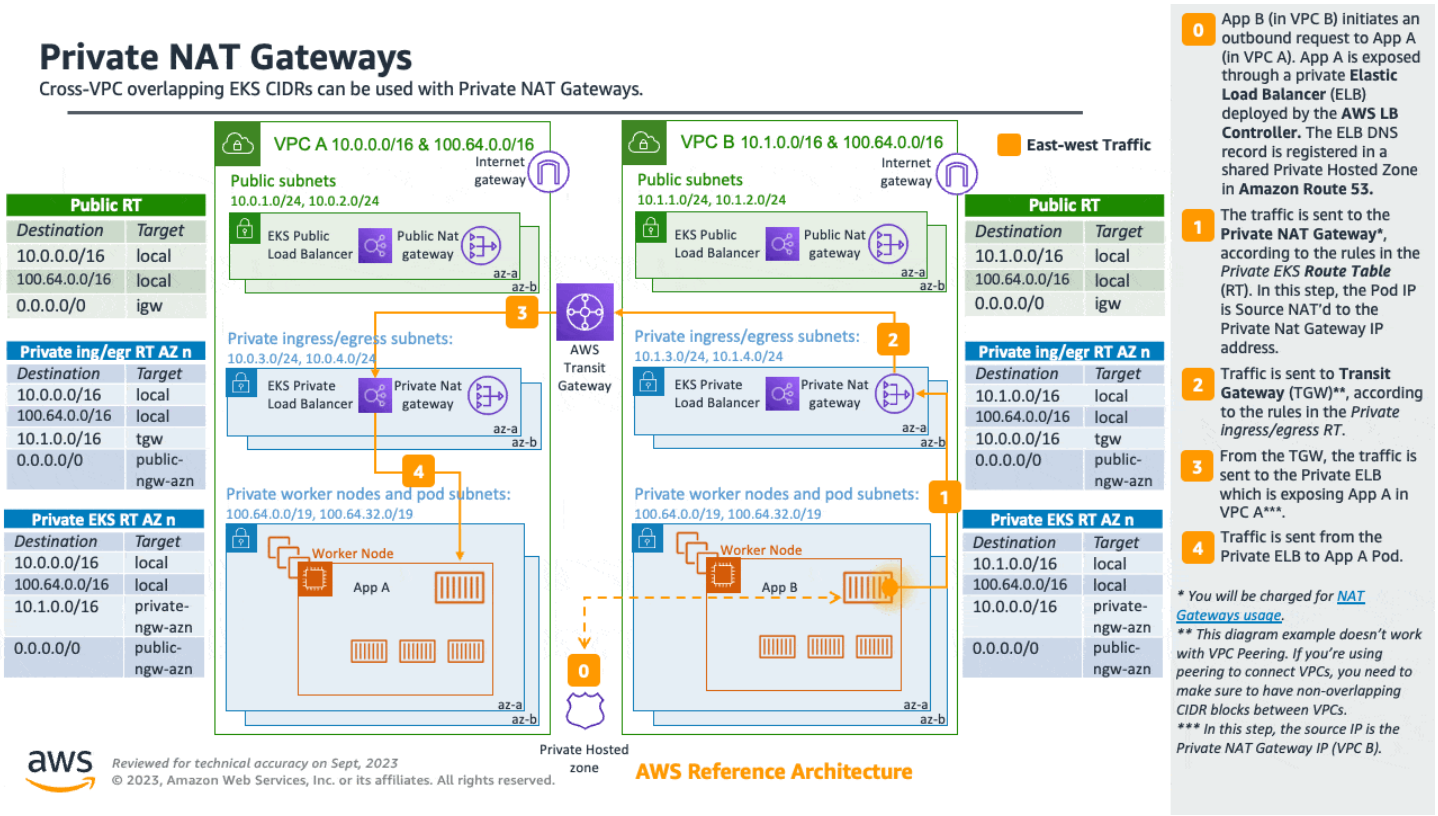
Comunicação entre VPCs

Há muitos cenários em que você precisa de vários VPCs clusters EKS separados implantados neles VPCs.

Você pode usar o Amazon VPC Lattice para conectar serviços de forma consistente e segura em várias VPCs contas (sem exigir que a conectividade adicional seja fornecida por serviços como emparelhamento de VPC, AWS ou AWS Transit Gateway). PrivateLink Saiba mais aqui.



O Amazon VPC Lattice opera no espaço de endereço local do link em IPv4 e IPv6, fornecendo conectividade entre serviços que podem ter endereços sobrepostos. IPv4 Para eficiência operacional, é altamente recomendável implantar clusters e nós EKS em intervalos de IP que não se sobreponham. Caso sua infraestrutura inclua intervalos VPCs de IP sobrepostos, você precisa arquitetar sua rede adequadamente. Sugerimos o Private NAT Gateway ou VPC CNI no modo de rede personalizado em conjunto com o Transit Gateway para integrar cargas de trabalho no EKS a fim de resolver desafios de CIDR sobrepostos e, ao mesmo tempo, preservar os endereços IP roteáveis. RFC1918

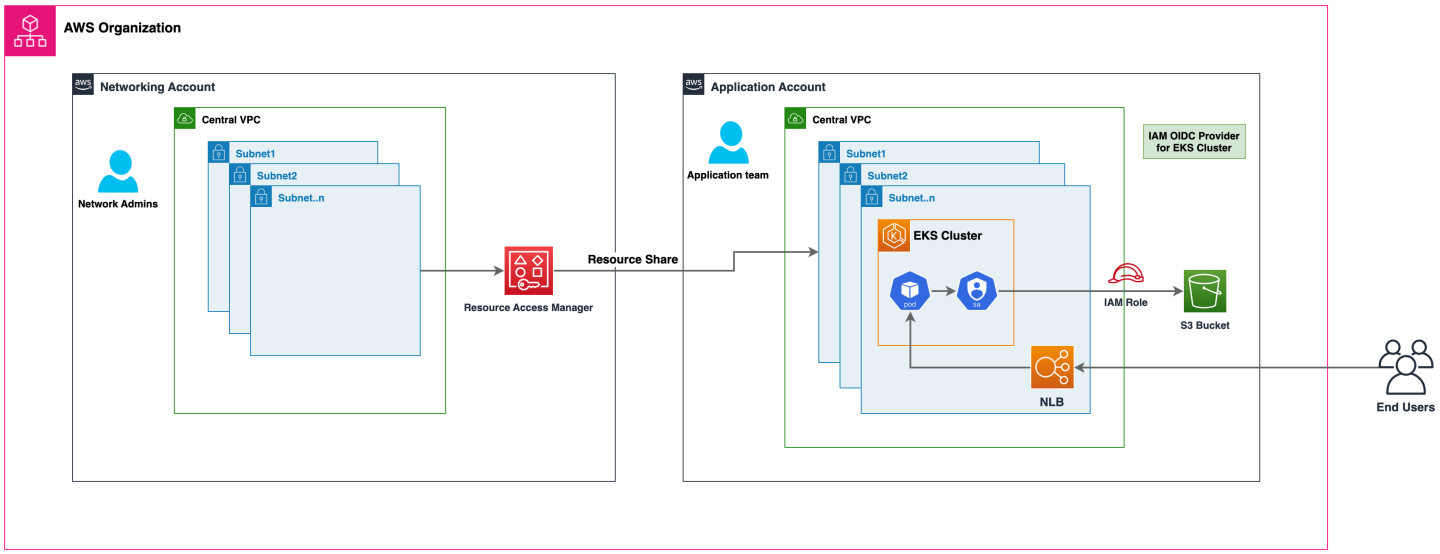


Considere utilizar a AWS PrivateLink, também conhecida como serviço de endpoint, se você for o provedor de serviços e quiser compartilhar seu serviço e entrada do Kubernetes (ALB ou NLB) com a VPC do seu cliente em contas separadas.

Compartilhamento de VPC em várias contas

Muitas empresas adotaram a Amazon compartilhada VPCs como forma de simplificar a administração da rede, reduzir custos e melhorar a segurança em várias contas da AWS em uma organização da AWS. Eles utilizam o AWS Resource Access Manager (RAM) para compartilhar com segurança os recursos compatíveis da AWS com contas individuais da AWS, unidades organizacionais (OUs) ou toda a organização da AWS.

Você pode implantar clusters do Amazon EKS, grupos de nós gerenciados e outros recursos de apoio da AWS (como grupos de segurança LoadBalancers, endpoints etc.) em sub-redes VPC compartilhadas de outra conta da AWS usando a AWS RAM. A figura abaixo mostra um exemplo de arquitetura de alto nível. Isso permite que as equipes de rede central controlem as construções de rede VPCs, como sub-redes, etc., enquanto permite que equipes de aplicativos ou plataformas implantem clusters do Amazon EKS em suas respectivas contas da AWS. Uma explicação completa desse cenário está disponível neste repositório github.



Considerações ao usar sub-redes compartilhadas

- Os clusters e nós de trabalho do Amazon EKS podem ser criados em sub-redes compartilhadas que fazem parte da mesma VPC. O Amazon EKS não oferece suporte à criação de clusters em vários VPCs.
- O Amazon EKS usa AWS VPC Security Groups (SGs) para controlar o tráfego entre o plano de controle do Kubernetes e os nós de trabalho do cluster. Os grupos de segurança também são usados para controlar o tráfego entre os nós de trabalho, outros recursos da VPC e endereços IP externos. Você deve criar esses grupos de segurança na application/participant conta. Certifique-se de que os grupos de segurança que você pretende usar para seus pods também estejam localizados na conta do participante. Você pode configurar as regras de entrada e saída em seus grupos de segurança para permitir o tráfego necessário de e para grupos de segurança localizados na conta VPC central.
- Crie funções do IAM e políticas associadas na conta do participante em que seu cluster Amazon EKS reside. Essas funções e políticas do IAM são essenciais para conceder as permissões necessárias aos clusters Kubernetes gerenciados pelo Amazon EKS, bem como aos nós e pods em execução no Fargate. As permissões permitem que o Amazon EKS faça chamadas para outros serviços da AWS em seu nome.
- Você pode seguir as seguintes abordagens para permitir o acesso entre contas aos recursos da AWS, como buckets do Amazon S3, tabelas do Dynamodb, etc., a partir de pods k8s:
  - Abordagem de política baseada em recursos: se o serviço da AWS oferecer suporte a políticas de recursos, você pode adicionar uma política adequada baseada em recursos para permitir o acesso entre contas às funções do IAM atribuídas aos pods do Kubernetes. Nesse cenário,

o provedor do OIDC, as funções do IAM e as políticas de permissão existem na conta do aplicativo. Para encontrar serviços da AWS que suportam políticas baseadas em recursos, consulte os serviços da AWS que funcionam com o IAM e procure os serviços que têm Sim na coluna Baseado em recursos.

- Abordagem do provedor OIDC: recursos do IAM, como provedor do OIDC, funções do IAM, permissões e políticas de confiança, serão criados em outra conta da AWS participante em que os recursos existam. Essas funções serão atribuídas aos pods do Kubernetes na conta do aplicativo, para que eles possam acessar recursos entre contas. Consulte o blog Cross account IAM roles for Kubernetes service accounts para ver uma explicação completa dessa abordagem.

- Você pode implantar os recursos do Amazon Elastic Loadbalancer (ELB) (ALB ou NLB) para rotear o tráfego para os pods k8s no aplicativo ou em contas de rede central. Consulte o passo a passo do Expose Amazon EKS Pods Through Cross-Account Load Balancer para obter instruções detalhadas sobre a implantação dos recursos do ELB na conta de rede central. Essa opção oferece maior flexibilidade, pois concede à conta Central Networking controle total sobre a configuração de segurança dos recursos do Load Balancer.
- Ao usar o `custom networking feature` Amazon VPC CNI, você precisa usar os mapeamentos de ID da Zona de Disponibilidade (AZ) listados na conta de rede central para criar cada um. `ENIConfig` Isso se deve ao mapeamento aleatório de nomes físicos AZs para os de AZ em cada conta da AWS.

Grupos de segurança

Um grupo de segurança controla o tráfego que tem permissão para acessar e sair dos recursos aos quais está associado. O Amazon EKS usa grupos de segurança para gerenciar a comunicação entre o plano de controle e os nós. Ao criar um cluster, o Amazon EKS cria um grupo de segurança com o nome `eks-cluster-sg-my-cluster-uniqueID`. O EKS associa esses grupos de segurança aos nós gerenciados ENIs e aos nós. As regras padrão permitem que todo o tráfego flua livremente entre o cluster e os nós e aceitam todo o tráfego de saída para qualquer destino.

Ao criar um cluster, você pode especificar seus próprios grupos de segurança. Consulte a recomendação para grupos de segurança ao especificar grupos de segurança próprios.

Recomendações

Considere a implantação do Multi-AZ

As regiões da AWS fornecem várias zonas de disponibilidade (AZ) fisicamente separadas e isoladas, conectadas a redes de baixa latência, alta taxa de transferência e altamente redundantes. Com as Zonas de Disponibilidade, você pode projetar e operar aplicativos que realizam o failover automático entre as Zonas de Disponibilidade sem interrupção. O Amazon EKS recomenda fortemente a implantação de clusters EKS em várias zonas de disponibilidade. Considere especificar sub-redes em pelo menos duas zonas de disponibilidade ao criar o cluster.

O Kubelet executado em nós adiciona automaticamente rótulos ao objeto do nó, como. `topology.kubernetes.io/region=us-west-2` Recomendamos usar rótulos de nós em conjunto com as restrições de distribuição da topologia de pods para controlar como os pods são distribuídos pelas zonas. Essas dicas permitem que o programador do Kubernetes coloque pods para obter uma melhor disponibilidade esperada, reduzindo o risco de que uma falha correlacionada afete toda a sua carga de trabalho. Consulte Atribuição de nós a pods para ver exemplos de seletor de nós e restrições de propagação de AZ.

Você pode definir as sub-redes ou as zonas de disponibilidade ao criar nós. Os nós são colocados em sub-redes de cluster se nenhuma sub-rede estiver configurada. O suporte do EKS para grupos de nós gerenciados distribui automaticamente os nós em várias zonas de disponibilidade de acordo com a capacidade disponível. O Karpenter honrará o posicionamento da distribuição AZ escalando os nós para especificar AZs se as cargas de trabalho definem limites de distribuição de topologia.

Os AWS Elastic Load Balancers são gerenciados pelo AWS Load Balancer Controller para um cluster Kubernetes. Ele provisiona um Application Load Balancer (ALB) para recursos de entrada do Kubernetes e um Network Load Balancer (NLB) para serviços Kubernetes do tipo Loadbalancer. O controlador do Elastic Load Balancer usa tags para descobrir as sub-redes. O controlador ELB requer um mínimo de duas zonas de disponibilidade (AZs) para provisionar o recurso de entrada com sucesso. Considere configurar sub-redes em pelo menos duas AZs para aproveitar a segurança e a confiabilidade da redundância geográfica.

Implante nós em sub-redes privadas

Uma VPC incluindo sub-redes públicas e privadas é o método ideal para implantar cargas de trabalho do Kubernetes no EKS. Considere definir no mínimo duas sub-redes públicas e duas sub-redes privadas em duas zonas de disponibilidade distintas. A tabela de rotas relacionada de uma

sub-rede pública contém uma rota para um gateway da Internet. Os pods podem interagir com a Internet por meio de um gateway NAT. As sub-redes privadas são suportadas por gateways de internet somente de saída no ambiente (EIGW). IPv6

A instanciação de nós em sub-redes privadas oferece controle máximo sobre o tráfego para os nós e é eficaz para a grande maioria dos aplicativos Kubernetes. Recursos de entrada (como balanceadores de carga) são instanciados em sub-redes públicas e direcionam o tráfego para pods que operam em sub-redes privadas.

Considere o modo somente privado se você exigir segurança estrita e isolamento de rede. Nessa configuração, três sub-redes privadas são implantadas em zonas de disponibilidade distintas dentro da VPC da região da AWS. Os recursos implantados nas sub-redes não podem acessar a Internet, nem a Internet pode acessar os recursos nas sub-redes. Para que seu aplicativo Kubernetes acesse outros serviços da AWS, você deve configurar PrivateLink interfaces and/or e endpoints de gateway. Você pode configurar balanceadores de carga internos para redirecionar o tráfego para pods usando o AWS Load Balancer Controller. As sub-redes privadas devem ser marcadas com (`kubernetes.io/role/internal-elb: 1`) para que o controlador provisione balanceadores de carga. Para que os nós se registrem no cluster, o endpoint do cluster deve ser definido para o modo privado. Visite o guia de clusters privados para ver os requisitos e considerações completos.

Considere o modo público e privado para o cluster Endpoint

O Amazon EKS oferece modos de endpoint public-and-private de cluster somente público e privado. O modo padrão é somente público, no entanto, recomendamos configurar o endpoint do cluster nos modos público e privado. Essa opção permite que as chamadas da API Kubernetes na VPC do seu cluster (como node-to-control-plane comunicação) utilizem o endpoint privado da VPC e o tráfego permaneça na VPC do seu cluster. Seu servidor de API de cluster, por outro lado, pode ser acessado pela Internet. No entanto, é altamente recomendável limitar os blocos CIDR que podem usar o endpoint público. Saiba como configurar o acesso a endpoints públicos e privados, incluindo a limitação de blocos CIDR.

Sugerimos um endpoint somente privado quando você precisar de segurança e isolamento de rede. Recomendamos usar qualquer uma das opções listadas no guia do usuário do EKS para se conectar a um servidor de API de forma privada.

Configure grupos de segurança com cuidado

O Amazon EKS oferece suporte ao uso de grupos de segurança personalizados. Qualquer grupo de segurança personalizado deve permitir a comunicação entre os nós e o plano de controle

do Kubernetes. Verifique os requisitos de porta e configure as regras manualmente quando sua organização não permitir a comunicação aberta.

O EKS aplica os grupos de segurança personalizados que você fornece durante a criação do cluster às interfaces gerenciadas (X-ENIs). No entanto, ele não os associa imediatamente aos nós. Ao criar grupos de nós, é altamente recomendável associar grupos de segurança personalizados manualmente. Considere ativar securityGroupSelectoros Termos para permitir que o modelo de nó do Karpenter descubra grupos de segurança personalizados durante o escalonamento automático dos nós.

É altamente recomendável criar um grupo de segurança para permitir todo o tráfego de comunicação entre nós. Durante o processo de bootstrap, os nós exigem conectividade de saída com a Internet para acessar o endpoint do cluster. Avalie os requisitos de acesso externo, como conexão local e acesso ao registro de contêineres, e defina as regras apropriadamente. Antes de colocar as mudanças em produção, sugerimos que você verifique cuidadosamente as conexões em seu ambiente de desenvolvimento.

Implemente gateways NAT em cada zona de disponibilidade

Se você implantar nós em sub-redes privadas (IPv4 e IPv6), considere criar um gateway NAT em cada zona de disponibilidade (AZ) para garantir uma arquitetura independente da zona e reduzir os gastos entre AZ. Cada gateway NAT em uma AZ é implementado com redundância.

CNI da Amazon VPC

O Amazon EKS implementa redes de cluster por meio do plug-in Amazon VPC Container Network Interface, também conhecido como VPC CNI. O plug-in CNI permite que os Kubernetes Pods tenham o mesmo endereço IP da rede VPC. Mais especificamente, todos os contêineres dentro do Pod compartilham um namespace de rede e podem se comunicar entre si usando portas locais.

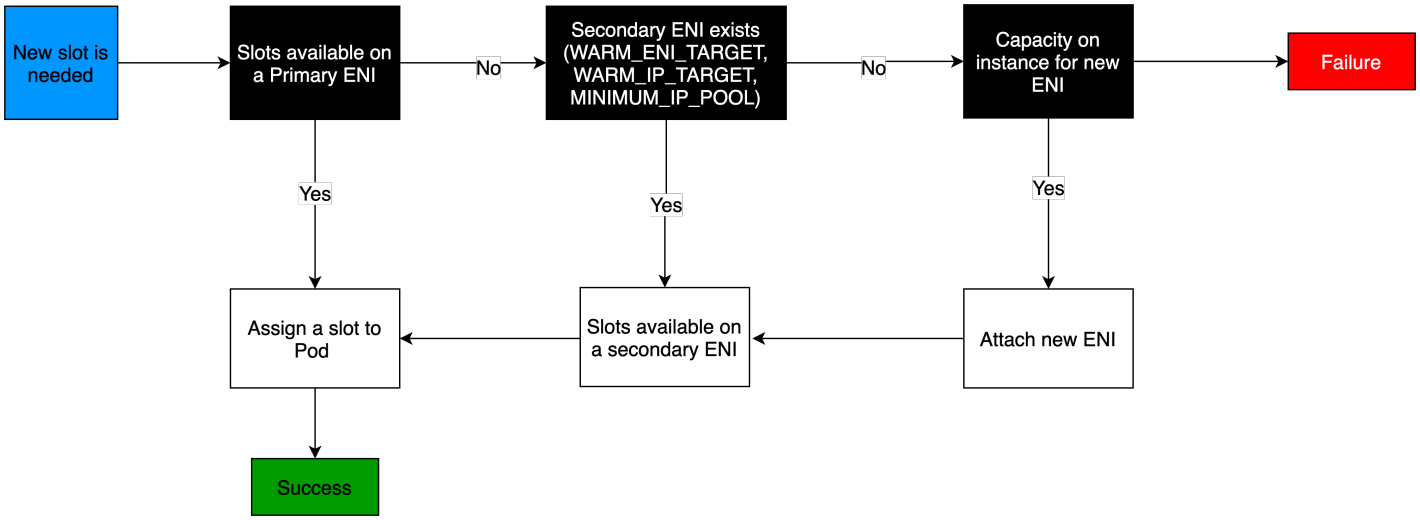
O Amazon VPC CNI tem dois componentes:

- CNI Binary, que configurará a rede Pod para permitir a Pod-to-Pod comunicação. O binário CNI é executado em um sistema de arquivos raiz do nó e é invocado pelo kubelet quando um novo pod é adicionado ou um pod existente é removido do nó.
- ipamd, um daemon de gerenciamento de endereços IP (IPAM) de nó local de longa duração e é responsável por:

- gerenciando ENIs em um nó e
- mantendo um conjunto aquecido de endereços IP ou prefixos disponíveis

Quando uma instância é criada, EC2 cria e anexa uma ENI primária associada a uma sub-rede primária. A sub-rede primária pode ser pública ou privada. Os pods executados no modo HostNetwork usam o endereço IP primário atribuído à ENI primária do nó e compartilham o mesmo namespace de rede do host.

O plug-in CNI gerencia as interfaces de rede elásticas (ENI) no nó. Quando um nó é provisionado, o plug-in CNI aloca automaticamente um pool de slots (IPs ou prefixos) da sub-rede do nó para a ENI primária. Esse pool é conhecido como pool quente e seu tamanho é determinado pelo tipo de instância do nó. Dependendo das configurações da CNI, um slot pode ser um endereço IP ou um prefixo. Quando um slot em um ENI é atribuído, o CNI pode anexar mais slots ENIs com um pool aquecido de slots aos nós. Esses adicionais ENIs são chamados de secundários ENIs. Cada ENI só pode suportar um determinado número de slots, com base no tipo de instância. A CNI atribui mais ENIs às instâncias com base no número de slots necessários, o que geralmente corresponde ao número de pods. Esse processo continua até que o nó não possa mais suportar ENI adicional. O CNI também pré-aloca “quentes” ENIs e slots para uma inicialização mais rápida do Pod. Observe que cada tipo de instância tem um número máximo ENIs que pode ser anexado. Essa é uma restrição na densidade de pods (número de pods por nó), além dos recursos computacionais.



O número máximo de interfaces de rede e o número máximo de slots que você pode usar variam de acordo com o tipo de EC2 instância. Como cada pod consome um endereço IP em um slot, o número de pods que você pode executar em uma EC2 instância específica depende de quantos ENIs podem ser conectados a ela e de quantos slots cada ENI suporta. Sugerimos definir o máximo de pods por

guia do usuário do EKS para evitar o esgotamento dos recursos de CPU e memória da instância. Os pods usados `hostNetwork` são excluídos desse cálculo. Você pode considerar usar um script chamado max-pod-calculator.sh para calcular o máximo de pods recomendados pelo EKS para um determinado tipo de instância.

Visão geral do

O modo IP secundário é o modo padrão para VPC CNI. Este guia fornece uma visão geral genérica do comportamento da VPC CNI quando o modo IP secundário está ativado. A funcionalidade do ipamd (alocação de endereços IP) pode variar dependendo das configurações da VPC CNI, como, e. the section called “Modo de prefixo para Linux” the section called “Grupos de segurança por pod” the section called “Rede personalizada”

O Amazon VPC CNI é implantado como um Daemonset do Kubernetes chamado aws-node em nós de trabalho. Quando um nó de trabalho é provisionado, ele tem uma ENI padrão, chamada ENI primária, anexada a ele. A CNI aloca um pool quente de endereços IP ENIs secundários da sub-rede conectada à ENI primária do nó. Por padrão, o ipamd tenta alocar uma ENI adicional para o nó. O IPAMD aloca ENI adicional quando um único pod é programado e recebe um endereço IP secundário do ENI primário. Essa ENI “quente” permite uma rede de Pod mais rápida. À medida que o pool de endereços IP secundários se esgota, a CNI adiciona outra ENI para atribuir mais.

O número ENIs e os endereços IP em um pool são configurados por meio de variáveis de ambiente chamadas WARM_ENI_TARGET, WARM_IP_TARGET, MINIMUM_IP_TARGET. O `aws-node` Daemonset verificará periodicamente se um número suficiente de ENIs está conectado. Um número suficiente de ENIs é anexado quando todas as `WARM_ENI_TARGET` `MINIMUM_IP_TARGET` condições ou `WARM_IP_TARGET` e são atendidas. Se não houver ENIs conexão suficiente, a CNI fará uma chamada de API para EC2 anexar mais até que o `MAX_ENI` limite seja atingido.

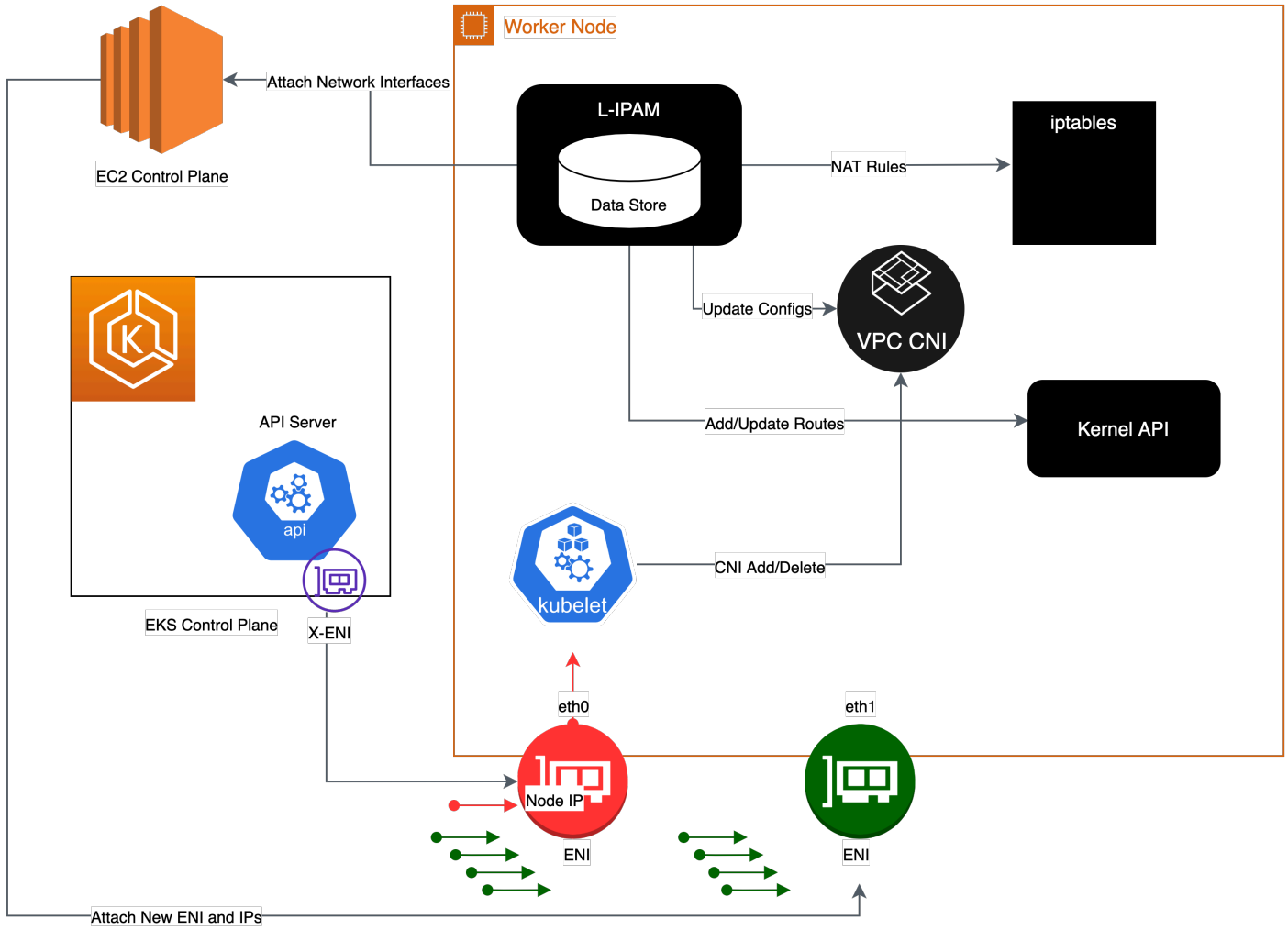
- `WARM_ENI_TARGET`- Inteiro, valores maiores que 0 indicam o requisito Ativado
  - O número de Warm ENIs a ser mantido. Uma ENI fica “quente” quando conectada como uma ENI secundária a um nó, mas não está sendo usada por nenhum pod. Mais especificamente, nenhum endereço IP da ENI foi associado a um pod.
  - Exemplo: considere uma instância com 2 ENIs, cada ENI suportando 5 endereços IP. WARM_ENI_TARGET está definido como 1. Se exatamente 5 endereços IP estiverem associados à instância, a CNI manterá 2 ENIs anexados à instância. A primeira ENI está em uso e todos os 5 endereços IP possíveis dessa ENI são usados. A segunda ENI é “quente” com todos os 5 endereços IP no pool. Se outro pod for executado na instância, será necessário um sexto endereço IP. A CNI atribuirá a esse 6º Pod um endereço IP do segundo ENI e do 5 IPs do

pool. O segundo ENI está agora em uso e não está mais em um status “quente”. O CNI alocará um 3º ENI para manter pelo menos 1 ENI aquecido.

> **Note**
>
> O warm ENIs ainda consome endereços IP do CIDR da sua VPC. Os endereços IP são “não utilizados” ou “inativos” até serem associados a uma carga de trabalho, como um pod.

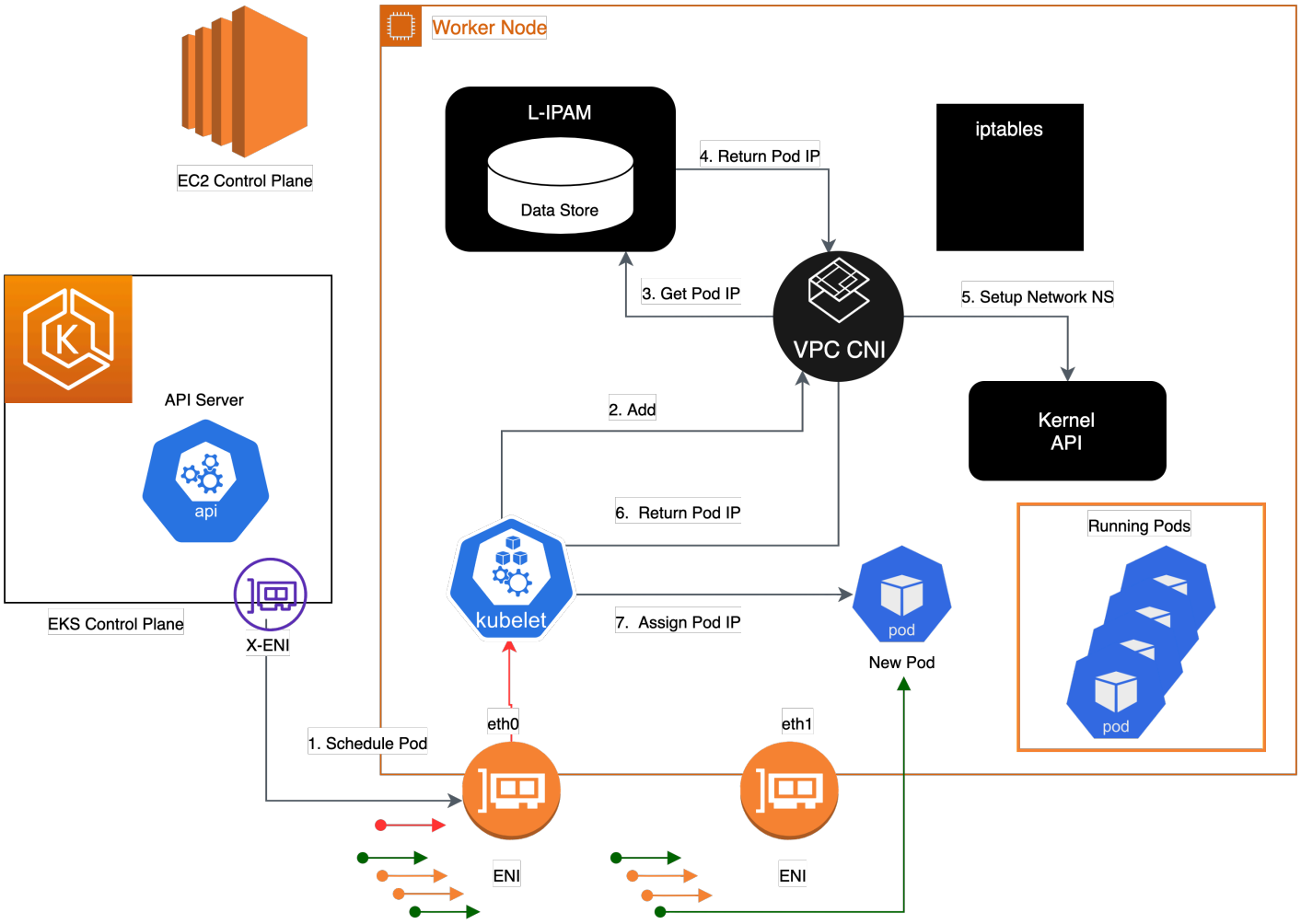
- `WARM_IP_TARGET`, Inteiro, Valores maiores que 0 indicam o requisito Ativado
  - O número de endereços IP quentes a serem mantidos. Um IP quente está disponível em um ENI conectado ativamente, mas não foi atribuído a um pod. Em outras palavras, o número de Warm IPs disponível é o número IPs que pode ser atribuído a um Pod sem exigir um ENI adicional.
- Exemplo: considere uma instância com 1 ENI, cada ENI suportando 20 endereços IP. WARM_IP_TARGET está definido como 5. WARM_ENI_TARGET está definido como 0. Somente 1 ENI será anexada até que um 16º endereço IP seja necessário. Em seguida, a CNI anexará uma segunda ENI, consumindo 20 endereços possíveis do CIDR da sub-rede.
- `MINIMUM_IP_TARGET`, Inteiro, Valores maiores que 0 indicam o requisito Ativado
  - O número mínimo de endereços IP a serem alocados a qualquer momento. Isso geralmente é usado para antecipar a atribuição de vários ENIs na inicialização da instância.
  - Exemplo: considere uma instância recém-lançada. Ele tem 1 ENI e cada ENI suporta 10 endereços IP. MINIMUM_IP_TARGET está definido como 100. O ENI anexa imediatamente mais 9 ENIs para um total de 100 endereços. Isso acontece independentemente de qualquer valor de WARM_IP_TARGET ou WARM_ENI_TARGET.

Este projeto inclui um documento Excel de calculadora de sub-rede. Este documento de calculadora simula o consumo de endereço IP de uma carga de trabalho especificada em diferentes opções de configuração de ENI, como e. `WARM_IP_TARGET` `WARM_ENI_TARGET`

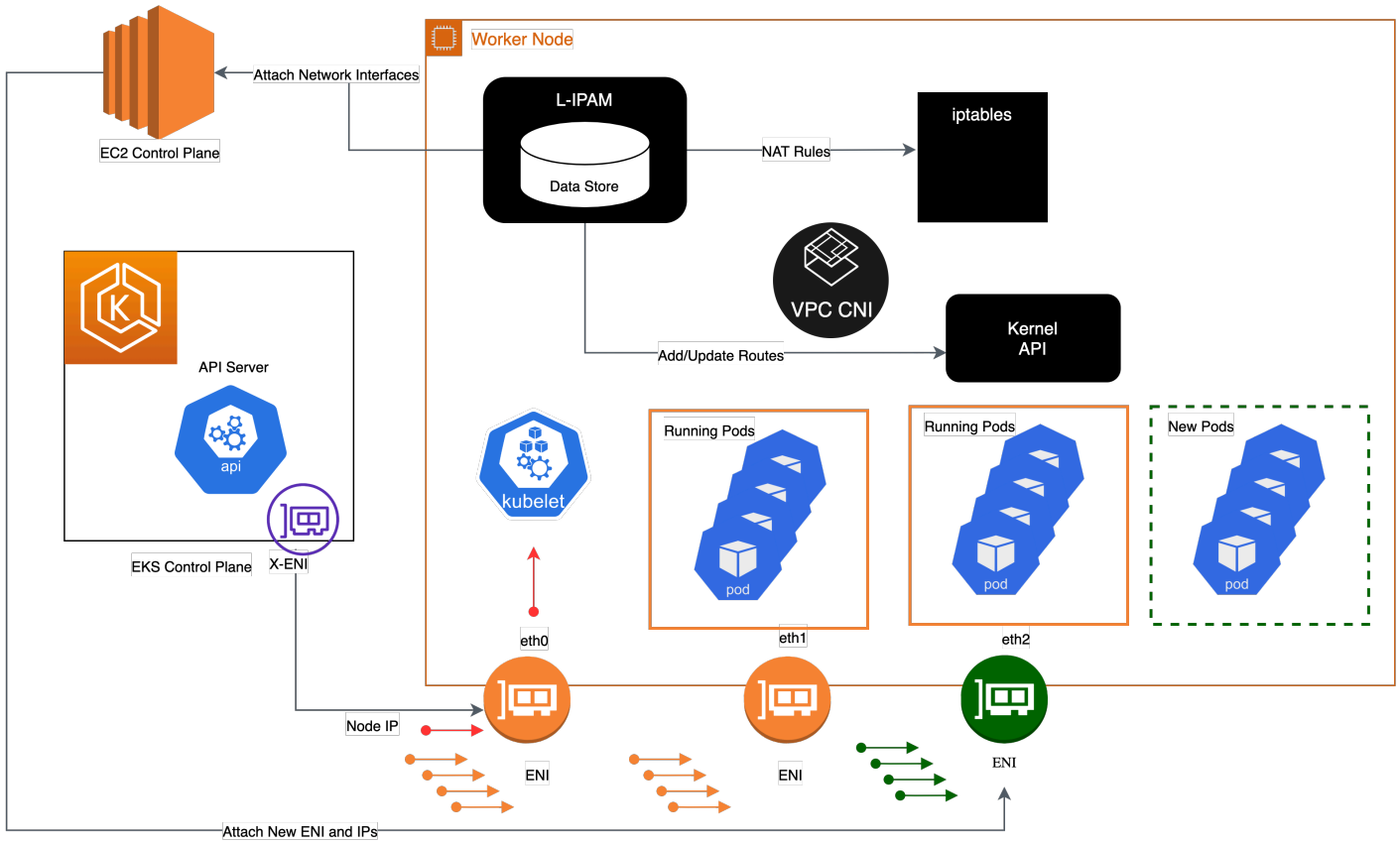


Quando o Kubelet recebe uma solicitação de adição de Pod, o binário CNI consulta ipamd para obter um endereço IP disponível, que o ipamd então fornece ao Pod. O binário CNI conecta o host e a rede Pod.

Os pods implantados em um nó são, por padrão, atribuídos aos mesmos grupos de segurança da ENI primária. Como alternativa, os pods podem ser configurados com grupos de segurança diferentes.



À medida que o grupo de endereços IP é esgotado, o plugin anexa automaticamente outra interface de rede elástica à instância e aloca outro conjunto de endereços IP secundários a essa interface. Esse processo continua até que o nó não possa mais oferecer suporte a interfaces de rede elástica adicionais.



Quando um pod é excluído, o VPC CNI coloca o endereço IP do pod em um cache de resfriamento de 30 segundos. Os IPs em um cache de resfriamento não são atribuídos a novos pods. Quando o período de resfriamento termina, o VPC CNI move o Pod IP de volta para a piscina aquecida. O período de reflexão evita que os endereços IP do pod sejam reciclados prematuramente e permite que o kube-proxy em todos os nós do cluster conclua a atualização das regras do iptables. Quando o número IPs ou ENIs excede o número de configurações de pool quente, o plug-in ipamd retorna ENIs para IPs a VPC.

Conforme descrito acima no modo IP secundário, cada pod recebe um endereço IP privado secundário de um dos ENIs anexados a uma instância. Como cada pod usa um endereço IP, o número de pods que você pode executar em uma EC2 instância específica depende de quantos ENIs podem ser anexados a ela e de quantos endereços IP ela suporta. A VPC CNI verifica o arquivo de limites para descobrir quantos ENIs endereços IP são permitidos para cada tipo de instância.

Você pode usar a fórmula a seguir para determinar o número máximo de pods que você pode implantar em um nó.

```
(Number of network interfaces for the instance type * (the number of IP addresses per
 network interface - 1)) + 2
```

O +2 indica pods que exigem rede de host, como kube-proxy e VPC CNI. O Amazon EKS exige que o kube-proxy e o VPC CNI operem em cada nó, e esses requisitos são considerados no valor de max-pods. Se você quiser executar mais pods de rede de host, considere atualizar o valor de max-pods. Você pode especificar `--kubelet-extra-args "–max-pods=110"` como dados do usuário no modelo de lançamento.

Por exemplo, em um cluster com 3 nós c5.large (3 ENIs e máximo 10 IPs por ENI), quando o cluster é inicializado e tem 2 pods CoreDNS, a CNI consome 50 endereços IP e mantém 43 em um pool aquecido. IPs A piscina aquecida permite uma inicialização mais rápida do Pod quando o aplicativo é implantado.

Nó 1 (com pod CoreDNS): ENIs 2, 20 atribuídos IPs

Nó 2 (com pod CoreDNS): ENIs 2, 20 atribuídos IPs

Node 3 (sem Pod): 1 ENI. 10 IPs atribuídos.

Para o Node 1 e o Node 2 (configuração idêntica):

- 2 ENIs × 10 IPs por ENI = 20 IPs no total
- Subtraia 2 primários IPs (1 por ENI) = 18 IPs
- Subtraia 1 IP para o pod CoreDNS = 17 disponíveis IPs
- Então, cada um desses nós tem 17 IPs na piscina quente

Para o Node 3:

- 1 ENI × 10 IPs = 10 IPs no total
- Subtraia 1 IP primário = 9 IPs disponível na piscina aquecida

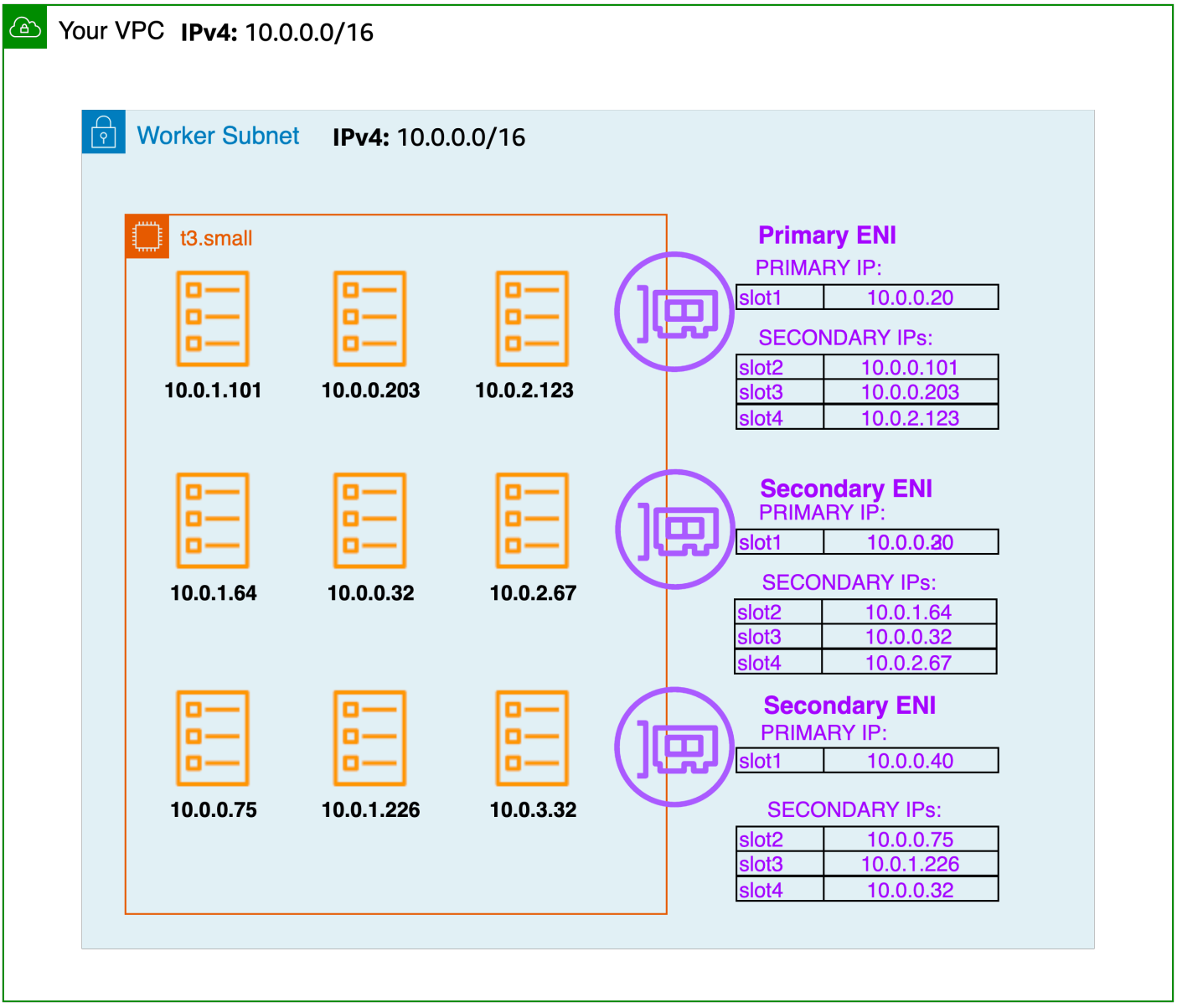
Cálculo total da piscina aquecida: - 17 (Node 1) + 17 (Node 2) + 9 (Node 3) = 43 IPs

Lembre-se de que cada um dos pods de infraestrutura, geralmente executados como conjuntos de daemons, contribui para a contagem máxima de pods. Isso pode incluir:

- CoreDNS

- Amazon Elastic LoadBalancer
- Pods operacionais para servidor de métricas

Sugerimos que você planeje sua infraestrutura combinando as capacidades desses Pods. Para ver uma lista do número máximo de pods compatíveis com cada tipo de instância, consulte eni-max-Pods.txt em GitHub.



Recomendações

Implemente o cluster EKS com o modo automático

Quando você usa o EKS Auto Mode para criar um cluster, a AWS gerencia a configuração da VPC Container Network Interface (CNI) para seu cluster. Com o Modo automático do Amazon EKS, você não precisa instalar ou atualizar complementos de rede. No entanto, certifique-se de que suas cargas de trabalho sejam compatíveis com a configuração gerenciada da VPC CNI.

Implemente o complemento gerenciado VPC CNI

Quando você provisiona um cluster, o Amazon EKS instala a VPC CNI automaticamente. No entanto, o Amazon EKS oferece suporte a complementos gerenciados que permitem que o cluster interaja com os recursos subjacentes da AWS, como computação, armazenamento e rede. É altamente recomendável que você implante clusters com complementos gerenciados, incluindo VPC CNI.

O complemento gerenciado do Amazon EKS oferece instalação e gerenciamento de VPC CNI para clusters do Amazon EKS. Os complementos do Amazon EKS incluem os patches de segurança e correções de erros mais recentes e são validados pela AWS para funcionar com o Amazon EKS. O complemento VPC CNI permite que você garanta continuamente a segurança e a estabilidade de seus clusters do Amazon EKS e diminua o esforço necessário para instalar, configurar e atualizar complementos. Além disso, um complemento gerenciado pode ser adicionado, atualizado ou excluído por meio da API Amazon EKS, do AWS Management Console, da AWS CLI e do eksctl.

Você pode encontrar os campos gerenciados da VPC CNI usando `--show-managed-fields` flag com o comando. `kubectl get`

```
kubectl get daemonset aws-node --show-managed-fields -n kube-system -o yaml
```

Os complementos gerenciados evitam desvios de configuração ao sobrescrever automaticamente as configurações a cada 15 minutos. Isso significa que todas as alterações nos complementos gerenciados, feitas por meio da API Kubernetes após a criação do complemento, serão substituídas pelo processo automatizado de prevenção de desvios e também serão definidas como padrões durante o processo de atualização do complemento.

Os campos gerenciados pelo EKS estão listados em ManagedFields com o gerente como EKS. Os campos gerenciados pelo EKS incluem conta de serviço, imagem, URL da imagem, sonda de vivacidade, sonda de prontidão, rótulos, volumes e montagens de volume.

> **Note**
>
> Os campos usados com mais frequência, como WARM_ENI_TARGET, WARM_IP_TARGET e MINIMUM_IP_TARGET, não são gerenciados e não serão reconciliados. As alterações nesses campos serão preservadas após a atualização do complemento.

Sugerimos testar o comportamento do complemento em seus clusters de não produção para uma configuração específica antes de atualizar os clusters de produção. Além disso, siga as etapas no guia do usuário do EKS para configurações complementares.

Migrar para o complemento gerenciado

Você gerenciará a compatibilidade de versões e atualizará os patches de segurança da VPC CNI autogerenciada. Para atualizar um complemento autogerenciado, você deve usar o Kubernetes APIs e as instruções descritas no guia do usuário do EKS. Recomendamos migrar para um complemento gerenciado para clusters EKS existentes e é altamente recomendável criar um backup das configurações atuais da CNI antes da migração. Para configurar complementos gerenciados, você pode utilizar a API Amazon EKS, o AWS Management Console ou a AWS Command Line Interface.

```
kubectl apply view-last-applied daemonset aws-node -n kube-system  aws-k8s-cni-old.yaml
```

O Amazon EKS substituirá as configurações da CNI se o campo estiver listado como gerenciado com as configurações padrão. Advertimos contra a modificação dos campos gerenciados. O complemento não reconcilia campos de configuração, como as variáveis de ambiente quente e os modos CNI. Os pods e os aplicativos continuarão em execução enquanto você migra para uma CNI gerenciada.

Backup das configurações do CNI antes da atualização

O VPC CNI é executado no plano de dados do cliente (nós) e, portanto, o Amazon EKS não atualiza automaticamente o complemento (gerenciado e autogerenciado) quando novas versões são lançadas ou depois que você atualiza seu cluster para uma nova versão secundária do Kubernetes. Para atualizar o complemento para um cluster existente, você deve acionar uma atualização por meio da API update-addon ou clicar no link Atualizar agora no console do EKS para complementos. Se você implantou o complemento autogerenciado, siga as etapas mencionadas em Atualização do complemento VPC CNI autogerenciado.

É altamente recomendável que você atualize uma versão secundária por vez. Por exemplo, se a sua versão atual for a `1.9` e você quiser atualizá-la para a `1.11`, deverá primeiro atualizá-la para a versão de patch mais recente da `1.10` primeiro e, em seguida, para a versão de patch mais recente da `1.11`.

Execute uma inspeção do daemonset do aws-node antes de atualizar a CNI da Amazon VPC. Faça um backup das configurações existentes. Se estiver usando um complemento gerenciado, confirme que você não atualizou nenhuma configuração que o Amazon EKS possa substituir. Recomendamos um gancho de pós-atualização em seu fluxo de trabalho de automação ou uma etapa de aplicação manual após uma atualização do complemento.

```
kubectl apply view-last-applied daemonset aws-node -n kube-system  aws-k8s-cni-old.yaml
```

Para um complemento autogerenciado, compare o backup com o `releases` ativado GitHub para ver as versões disponíveis e se familiarizar com as alterações na versão para a qual você deseja atualizar. Recomendamos usar o Helm para gerenciar complementos autogerenciados e aproveitar arquivos de valores para aplicar configurações. Qualquer operação de atualização envolvendo a exclusão do Daemonset resultará em tempo de inatividade do aplicativo e deverá ser evitada.

Entenda o contexto de segurança

É altamente recomendável que você entenda os contextos de segurança configurados para gerenciar a VPC CNI de forma eficiente. O Amazon VPC CNI tem dois componentes: CNI binário e ipamd (aws-node) Daemonset. O CNI é executado como um binário em um nó e tem acesso ao sistema de arquivos raiz do nó, além de ter acesso privilegiado, pois lida com iptables no nível do nó. O binário CNI é invocado pelo kubelet quando os pods são adicionados ou removidos.

O aws-node Daemonset é um processo de longa execução responsável pelo gerenciamento de endereços IP no nível do nó. O aws-node é executado no `hostNetwork` modo e permite acesso ao dispositivo de loopback e à atividade de rede de outros pods no mesmo nó. O aws-node init-container é executado no modo privilegiado e monta o soquete CRI, permitindo que o Daemonset monitore o uso do IP pelos pods em execução no nó. O Amazon EKS está trabalhando para remover o requisito privilegiado do contêiner de inicialização aws-node. Além disso, o aws-node precisa atualizar as entradas NAT e carregar os módulos iptables e, portanto, é executado com privilégios NET_ADMIN.

O Amazon EKS recomenda implantar as políticas de segurança conforme definido pelo manifesto aws-node para gerenciamento de IP para os pods e as configurações de rede. Considere atualizar

para a versão mais recente do VPC CNI. Além disso, considere abrir um GitHub problema se você tiver um requisito de segurança específico.

Use uma função separada do IAM para a CNI

O AWS VPC CNI exige permissões do AWS Identity and Access Management (IAM). A política da CNI precisa ser configurada antes que a função do IAM possa ser usada. Você pode usar `AmazonEKS_CNI_Policy`, que é uma política gerenciada pela AWS para IPv4 clusters. A política gerenciada da CNI do Amazoneks só tem permissões para clusters. IPv4 Você deve criar uma política de IAM separada para IPv6 clusters com as permissões listadas aqui.

Por padrão, a VPC CNI herda a função IAM do nó Amazon EKS (grupos de nós gerenciados e autogerenciados).

É altamente recomendável configurar uma função separada do IAM com as políticas relevantes para a CNI da Amazon VPC. Caso contrário, os pods do Amazon VPC CNI recebem a permissão atribuída à função IAM do nó e têm acesso ao perfil da instância atribuído ao nó.

O plug-in VPC CNI cria e configura uma conta de serviço chamada aws-node. Por padrão, a conta de serviço se vincula à função IAM do nó Amazon EKS com a política CNI do Amazon EKS anexada. Para usar a função separada do IAM, recomendamos que você crie uma nova conta de serviço com a política CNI do Amazon EKS anexada. Para usar uma nova conta de serviço, você deve reimplantar os pods da CNI. Considere especificar um complemento gerenciado `--service-account-role-arn` para VPC CNI ao criar novos clusters. Certifique-se de remover a política CNI do Amazon EKS para ambas IPv4 e IPv6 da função de nó do Amazon EKS.

É recomendável bloquear os metadados da instância de acesso para minimizar o raio de explosão da violação de segurança.

Lidar com Liveness/Readiness falhas na sonda

Recomendamos aumentar os valores de tempo limite da sondagem de atividade e prontidão (padrão`timeoutSeconds: 10`) para clusters EKS 1.20 e posteriores para evitar que falhas na sondagem façam com que o pod do seu aplicativo fique preso no estado ContainerCreating. Esse problema foi observado em clusters com uso intensivo de dados e processamento em lote. O alto uso da CPU causa falhas na integridade da sonda aws-node, levando a solicitações não atendidas da CPU do Pod. Além de modificar o tempo limite da sondagem, certifique-se de que as solicitações de recursos da CPU (padrão`CPU: 25m`) para aws-node estejam configuradas corretamente. Não sugerimos atualizar as configurações, a menos que seu nó esteja com problemas.

É altamente recomendável que você execute o sudo `bash /opt/cni/bin/aws-cni-support.sh` em um nó enquanto se envolve com o suporte do Amazon EKS. O script ajudará na avaliação dos registros do kubelet e da utilização da memória no nó. Considere instalar o SSM Agent nos nós de trabalho do Amazon EKS para executar o script.

Configurar a política de IPTables encaminhamento em instâncias de AMI não otimizadas para EKS

Se você estiver usando uma AMI personalizada, certifique-se de definir a política de encaminhamento de iptables como ACCEPT em kubelet.service. Muitos sistemas definem a política de encaminhamento do iptables como DROP. Você pode criar uma AMI personalizada usando o HashiCorp Packer e uma especificação de compilação com recursos e scripts de configuração do repositório Amazon EKS AMI na AWS. GitHub Você pode atualizar o kubelet.service e seguir as instruções especificadas aqui para criar uma AMI personalizada.

Atualize rotineiramente a versão CNI

O VPC CNI é compatível com versões anteriores. A versão mais recente funciona com todas as versões do Kubernetes compatíveis com o Amazon EKS. Além disso, o VPC CNI é oferecido como um complemento do EKS (consulte “Implantar o complemento gerenciado do VPC CNI” acima). Embora os complementos do EKS organizem atualizações de complementos, eles não atualizarão automaticamente os complementos, como o CNI, porque eles são executados no plano de dados. Você é responsável por atualizar o complemento VPC CNI após as atualizações de nós de trabalho gerenciados e autogerenciados.

Otimizando a utilização do endereço IP

Os ambientes em contêineres estão crescendo em escala em um ritmo acelerado, graças à modernização dos aplicativos. Isso significa que cada vez mais nós e pods de trabalho estão sendo implantados.

O plug-in CNI da Amazon VPC atribui a cada pod um endereço IP do (s) CIDR (s) da VPC. Essa abordagem fornece visibilidade total dos endereços do pod com ferramentas como VPC Flow Logs e outras soluções de monitoramento. Dependendo do tipo de carga de trabalho, isso pode fazer com que um número substancial de endereços IP seja consumido pelos pods.

Ao projetar sua arquitetura de rede da AWS, é importante otimizar o consumo de IP do Amazon EKS no VPC e no nível do nó. Isso ajudará você a mitigar os problemas de exaustão de IP e a aumentar a densidade do pod por nó.

Nesta seção, discutiremos técnicas que podem ajudá-lo a atingir essas metas.

Otimize o consumo de IP em nível de nó

A delegação de prefixos é um recurso da Amazon Virtual Private Cloud (Amazon VPC) que permite IPv4 atribuir IPv6 ou prefixar suas instâncias do Amazon Elastic Compute Cloud (EC2Amazon). Ele aumenta os endereços IP por interface de rede (ENI), o que aumenta a densidade do pod por nó e melhora sua eficiência computacional. A delegação de prefixo também é compatível com redes personalizadas.

Para obter informações detalhadas, consulte as seções Delegação de prefixo com nós Linux e Delegação de prefixo com nós Windows.

Mitigar exaustão de IP

Para evitar que seus clusters consumam todos os endereços IP disponíveis, é altamente recomendável dimensionar suas VPCs e suas sub-redes pensando no crescimento.

A adoção IPv6é uma ótima maneira de evitar esses problemas desde o início. No entanto, para organizações cujas necessidades de escalabilidade excedem o planejamento inicial e não podem ser adotadas IPv6, melhorar o design da VPC é a resposta recomendada ao esgotamento do endereço IP. A técnica mais usada entre os clientes do Amazon EKS é adicionar um secundário não roteável CIDRs à VPC e configurar a CNI da VPC para usar esse espaço IP adicional ao alocar endereços IP aos pods. Isso geralmente é chamado de rede personalizada.

Abordaremos quais variáveis do Amazon VPC CNI você pode usar para otimizar o pool de aquecimento IPs atribuído aos seus nós. Encerraremos esta seção com alguns outros padrões de arquitetura que não são intrínsecos ao Amazon EKS, mas que podem ajudar a mitigar o esgotamento de IP.

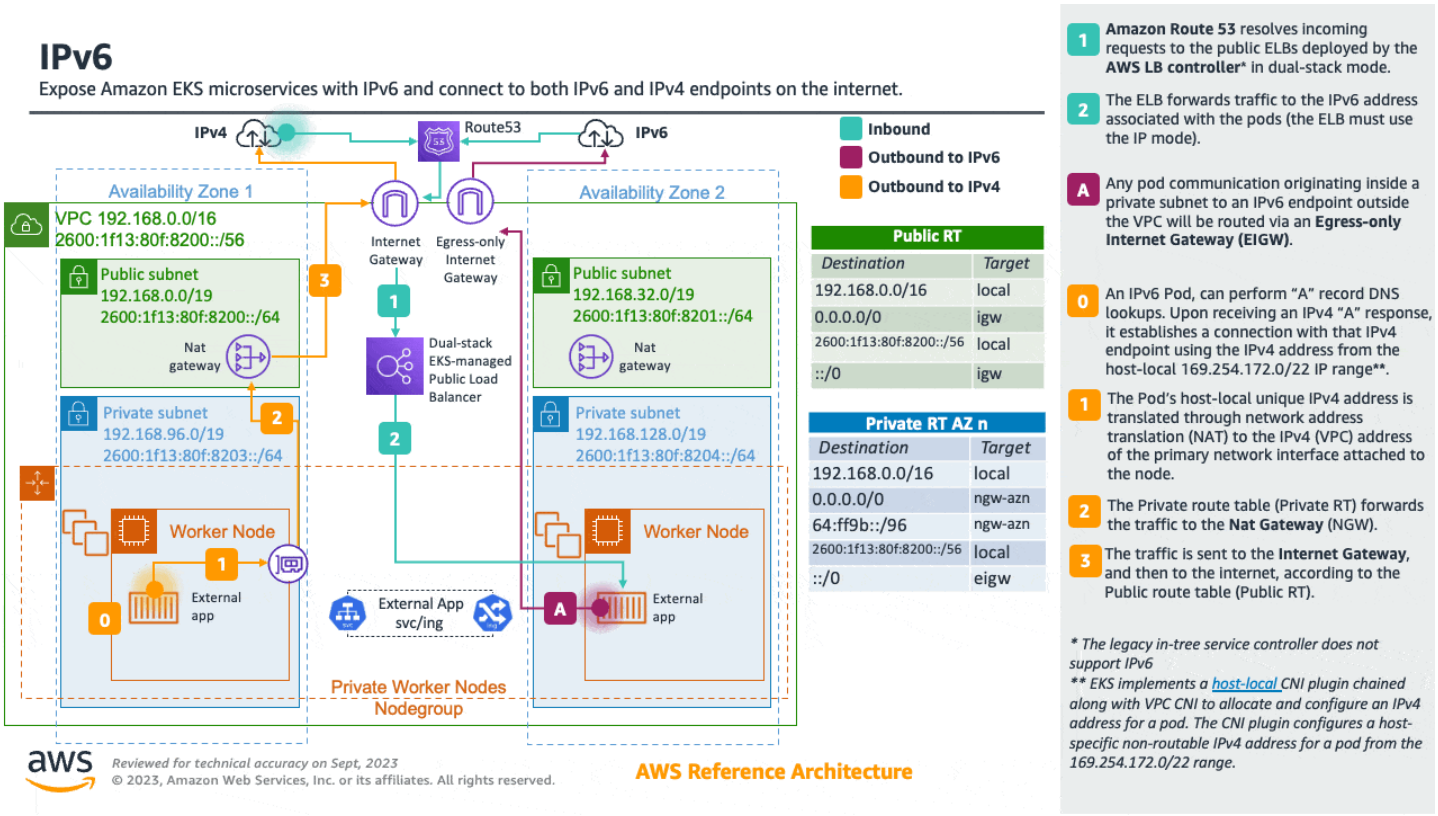
Uso IPv6 (recomendado)

A adoção IPv6 é a maneira mais fácil de contornar as RFC1918 limitações; é altamente recomendável que você considere a adoção IPv6 como primeira opção ao escolher uma arquitetura de rede. IPv6 fornece um espaço total de endereço IP significativamente maior, e os administradores de cluster podem se concentrar em migrar e escalar aplicativos sem se esforçar para contornar os limites. IPv4

Os clusters do Amazon EKS são compatíveis com IPv4 IPv6 e. Por padrão, os clusters EKS usam espaço IPv4 de endereço. Especificar um espaço de endereço IPv6 baseado no momento da

criação do cluster permitirá o uso de IPv6. Em um cluster IPv6 EKS, pods e serviços recebem IPv6 endereços enquanto mantêm a capacidade de IPv4 endpoints legados se conectarem a serviços IPv6 executados em clusters e vice-versa. Toda a pod-to-pod comunicação dentro de um cluster sempre termina IPv6. Em uma VPC (/56), o tamanho do bloco IPv6 CIDR para IPv6 sub-redes é fixado em /64. Isso fornece 2^64 (aproximadamente 18 quintilhões) de IPv6 endereços, permitindo escalar suas implantações no EKS.

Para obter informações detalhadas, consulte a seção Executando clusters do IPv6 EKS e, para obter experiência prática, consulte a seção Entendendo o IPv6 Amazon EKS do workshop Get hands-on with. IPv6



Otimize o consumo de IP em IPv4 clusters

Esta seção é dedicada aos clientes que estão executando aplicativos legados e para os and/or quais não estão prontos para migrar IPv6. Embora incentivemos todas as organizações a migrarem para IPv6 lá o mais rápido possível, reconhecemos que algumas ainda precisam procurar abordagens alternativas para escalar suas cargas de trabalho de contêineres. IPv4 Por esse motivo, também explicaremos os padrões arquitetônicos para otimizar IPv4 (RFC1918) o consumo de espaço de endereço com os clusters do Amazon EKS.

Plano de crescimento

Como primeira linha de defesa contra o esgotamento de IP, é altamente recomendável dimensionar suas sub-redes IPv4 VPCs e suas sub-redes pensando no crescimento, para evitar que seus clusters consumam todos os endereços IP disponíveis. Você não poderá criar novos pods ou nós se as sub-redes não tiverem endereços IP disponíveis suficientes.

Antes de criar a VPC e as sub-redes, é recomendável retroceder a partir da escala de carga de trabalho necessária. Por exemplo, quando os clusters são criados usando eksctl (uma ferramenta CLI simples para criar e gerenciar clusters no EKS), as sub-redes /19 são criadas por padrão. Uma máscara de rede de /19 é adequada para a maioria dos tipos de carga de trabalho, permitindo que mais de 8.000 endereços sejam alocados.

> **Important**
>
> Quando você VPCs dimensiona as sub-redes, pode haver vários elementos (além de pods e nós) que podem consumir endereços IP, por exemplo, balanceadores de carga, bancos de dados RDS e outros serviços in-vpc.

Além disso, o Amazon EKS pode criar até 4 interfaces de rede elásticas (X-ENI) que são necessárias para permitir a comunicação com o plano de controle (mais informações aqui). Durante as atualizações do cluster, o Amazon EKS cria novos X- ENIs e exclui os antigos quando a atualização é bem-sucedida. Por esse motivo, recomendamos uma máscara de rede de pelo menos /28 (16 endereços IP) para sub-redes associadas a um cluster EKS.

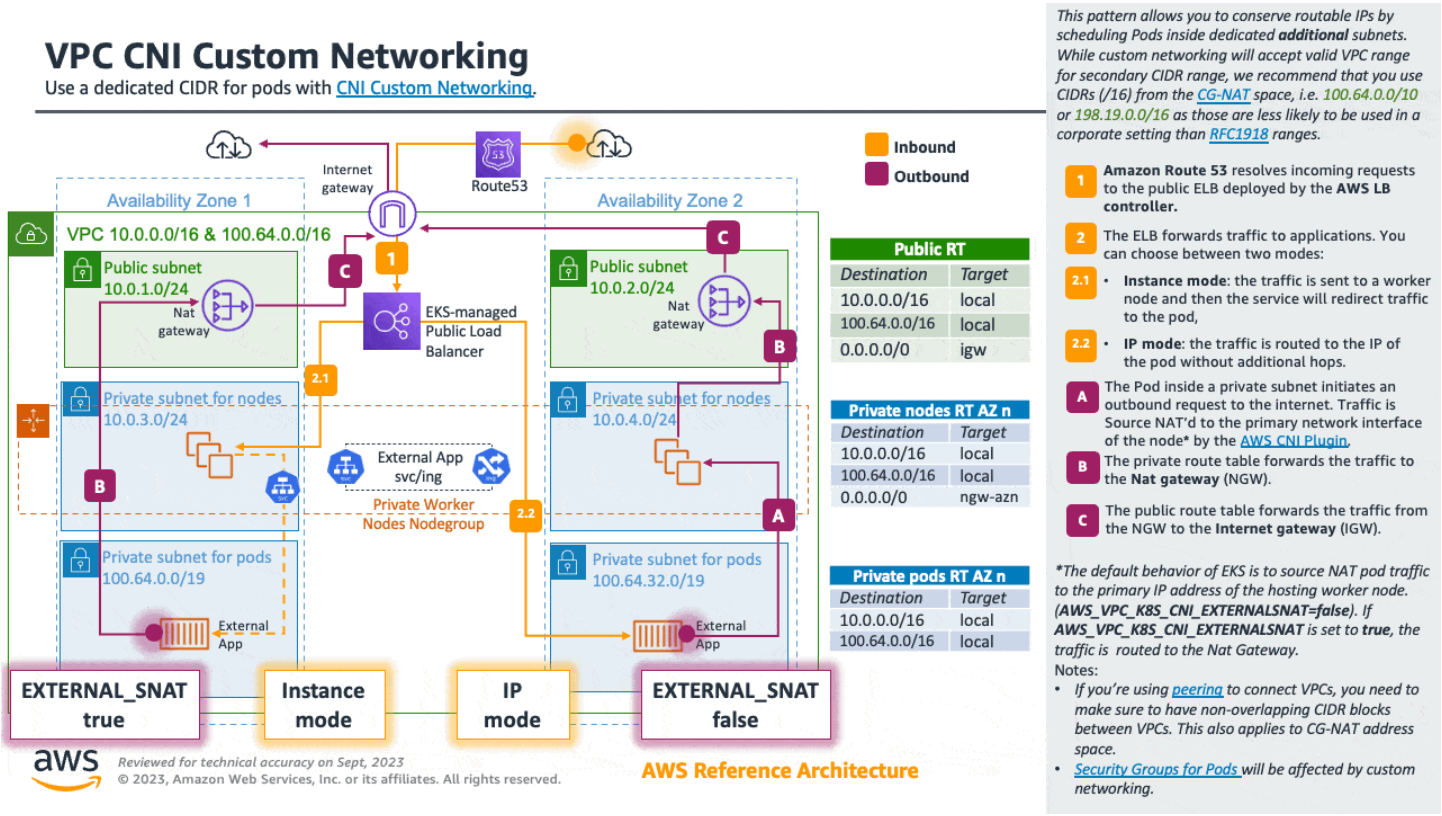
Você pode usar o exemplo da planilha EKS Subnet Calculator para planejar sua rede. A planilha calcula o uso do IP com base nas cargas de trabalho e na configuração do VPC ENI. O uso do IP é comparado a uma IPv4 sub-rede para determinar se a configuração e o tamanho da sub-rede são suficientes para sua carga de trabalho. Lembre-se de que, se as sub-redes em sua VPC ficarem sem endereços IP disponíveis, sugerimos criar uma nova sub-rede usando os blocos CIDR originais da VPC. Observe que agora o Amazon EKS agora permite a modificação de sub-redes de cluster e grupos de segurança.

Rede personalizada

Se você estiver prestes a esgotar o espaço RFC1918 IP, poderá usar o padrão de rede personalizada para conservar o roteável IPs agendando pods em sub-redes adicionais dedicadas. Embora a rede personalizada aceite um intervalo VPC válido para o intervalo CIDR secundário,

recomendamos que você use CIDRs (/16) do espaço CG-NAT, `100.64.0.0/10` ou seja, `198.19.0.0/16` porque é menos provável que sejam usados em um ambiente corporativo do que os intervalos. RFC1918

Para obter informações detalhadas, consulte a seção dedicada à rede personalizada.



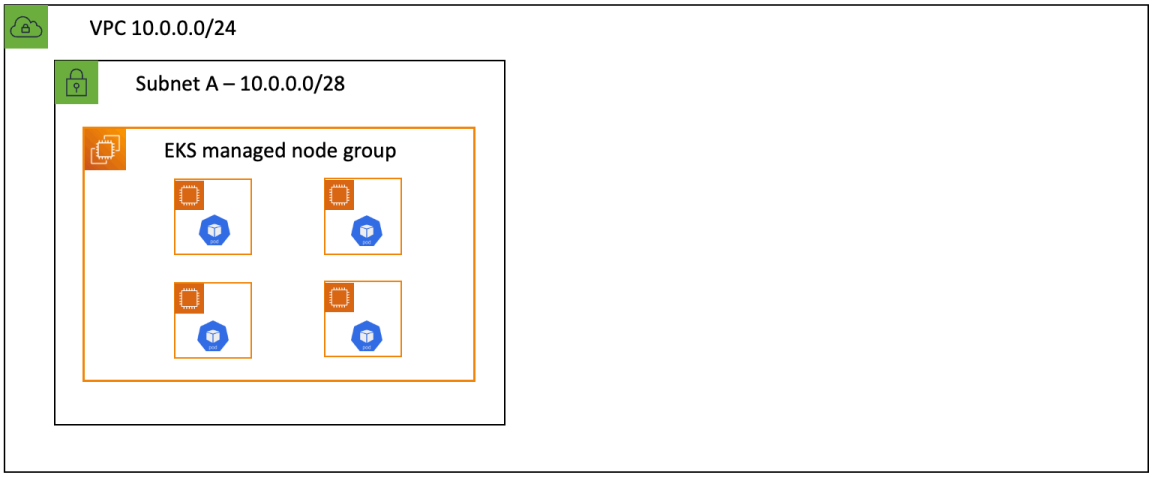
Descoberta aprimorada de sub-rede

O Enhanced Subnet Discovery fornece uma alternativa simplificada de configuração de rede para o esgotamento de IP, marcando novas sub-redes para que possam ser descobertas pela CNI da Amazon VPC. Com o Enhanced Subnet Discovery, as cargas de trabalho atuais podem continuar em execução nas mesmas sub-redes e o Amazon Elastic Kubernetes Service (Amazon EKS) agora pode programar pods adicionais nas novas "sub-redes utilizáveis".

Se as sub-redes atuais do seu cluster estiverem ficando sem endereços IP, você pode simplesmente adicionar outras sub-redes ao seu cluster Amazon EKS da seguinte forma:

1. Associe um novo bloco CIDR à sua VPC.
2. Crie uma nova sub-rede no novo bloco CIDR e marque-a com "kubernetes". io/role/cni" = "1".
3. Ative a configuração ENABLE_SUBNET_DISCOVERY do complemento CNI da Amazon VPC como "true" (padrão desde a versão 1.18.0).

Depois que o Enhanced Subnet Discovery estiver habilitado em seus clusters VPC e Amazon EKS, novas interfaces de rede elástica ENIs () serão anexadas aos seus nós do Amazon EKS, conforme descrito no diagrama a seguir:



Para obter mais informações, consulte Amazon VPC CNI apresenta o Enhanced Subnet Discovery no blog de contêineres da AWS.

Otimize a piscina IPs aquecida

Com a configuração padrão, a VPC CNI mantém uma ENI inteira (e associada IPs) no pool aquecido. Isso pode consumir um grande número de IPs, especialmente em tipos de instância maiores.

Se sua sub-rede de cluster tiver um número limitado de endereços IP disponíveis, examine estas variáveis de ambiente de configuração da VPC CNI:

- `WARM_IP_TARGET`
- `MINIMUM_IP_TARGET`
- `WARM_ENI_TARGET`

Você pode configurar o valor de `MINIMUM_IP_TARGET` para corresponder aproximadamente ao número de pods que você espera executar em seus nós. Isso garantirá que, à medida que os pods forem criados, a CNI possa atribuir endereços IP do pool aquecido sem chamar a EC2 API.

Lembre-se de que definir um valor `WARM_IP_TARGET` muito baixo causará chamadas adicionais para a EC2 API e isso poderá causar limitação das solicitações. Para clusters grandes, use junto com `MINIMUM_IP_TARGET` para evitar a limitação das solicitações.

Para configurar essas opções, você pode baixar o `aws-k8s-cni.yaml` manifesto e definir as variáveis de ambiente. No momento em que este artigo foi escrito, a versão mais recente está localizada aqui. Verifique se a versão do valor da configuração corresponde à versão da VPC CNI instalada.

> **Warning**
>
> Essas configurações serão redefinidas para os padrões quando você atualizar o CNI. Faça um backup do CNI antes de atualizá-lo. Revise as configurações para determinar se você precisa reaplicá-las após a atualização ser bem-sucedida.

Você pode ajustar os parâmetros da CNI em tempo real, sem tempo de inatividade para seus aplicativos existentes, mas deve escolher valores que atendam às suas necessidades de escalabilidade. Por exemplo, se você estiver trabalhando com cargas de trabalho em lote, recomendamos atualizar o padrão `WARM_ENI_TARGET` para atender às necessidades de escala do pod. `WARM_ENI_TARGET`Definir um valor alto sempre mantém o pool de IP aquecido necessário para executar grandes cargas de trabalho em lotes e, portanto, evitar atrasos no processamento de dados.

> **Warning**
>
> Melhorar seu design de VPC é a resposta recomendada para o esgotamento do endereço IP. Considere soluções como IPv6 Secondary CIDRs. Ajustar esses valores para minimizar o número de Warm IPs deve ser uma solução temporária após a exclusão de outras opções. A configuração incorreta desses valores pode interferir na operação do cluster. Antes de fazer qualquer alteração em um sistema de produção, não deixe de revisar as considerações nesta página.

Monitore o inventário de endereços IP

Além das soluções descritas acima, também é importante ter visibilidade sobre a utilização do IP. Você pode monitorar o inventário de endereços IP das sub-redes usando o CNI Metrics Helper. Algumas das métricas disponíveis são:

- número máximo que ENIs o cluster pode suportar
- número de ENIs já alocados
- número de endereços IP atualmente atribuídos aos pods
- número total e máximo de endereços IP disponíveis

Você também pode definir CloudWatch alarmes para ser notificado se uma sub-rede estiver ficando sem endereços IP.

> **Warning**
>
> Certifique-se de que a `DISABLE_METRICS` variável para VPC CNI esteja definida como falsa.

Outras considerações

Há outros padrões de arquitetura não intrínsecos ao Amazon EKS que podem ajudar no esgotamento do IP. Por exemplo, você pode otimizar a comunicação VPCs ou compartilhar uma VPC em várias contas para limitar a alocação de IPv4 endereços.

Saiba mais sobre esses padrões aqui:

- Projetando redes Amazon VPC em hiperescala,
- Crie conectividade segura de várias contas com várias VPCs com o Amazon VPC Lattice.

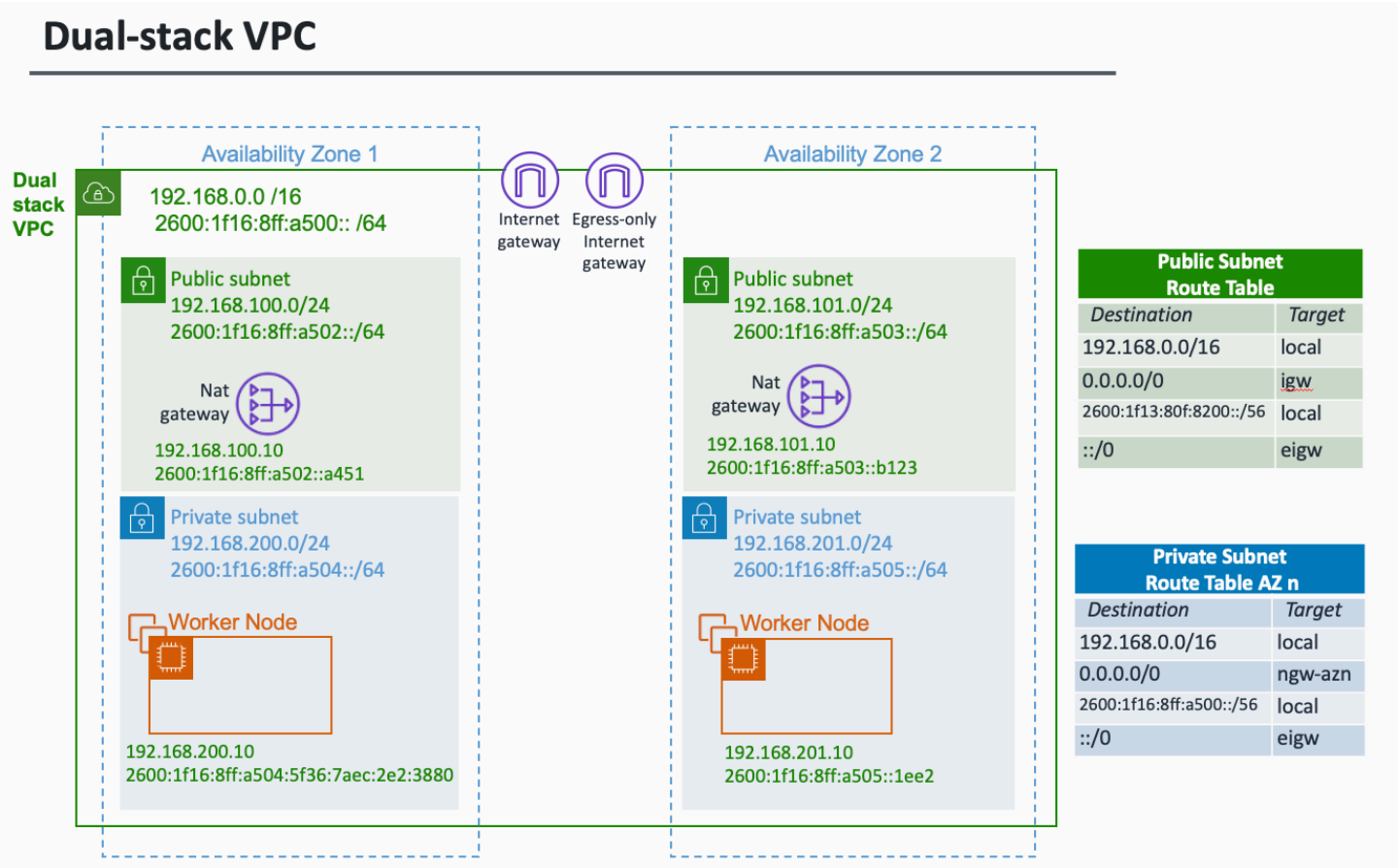
Executando clusters IPv6 EKS

O IPv6 modo EKS in resolve o desafio de IPv4 exaustão que geralmente se manifesta em clusters EKS de grande escala. O suporte do EKS para IPv6 está focado em resolver o problema de IPv4 exaustão, que decorre do tamanho limitado do espaço de endereço. IPv4 Essa é uma preocupação significativa levantada por vários de nossos clientes e é diferente do recurso IPv4IPv6 Kubernetes/dual-stack. O IPv6 EKS/também fornecerá a flexibilidade de interconectar os limites da rede,

minimizando IPv6 CIDRs assim as chances de sofrer sobreposição de CIDR, resolvendo assim um problema duplo (dentro do cluster, entre clusters). Ao implantar clusters EKS no IPv6 modo (--ip-family ipv6), a ação não é reversível. Em palavras simples, o IPv6 suporte ao EKS está habilitado durante toda a vida útil do seu cluster.

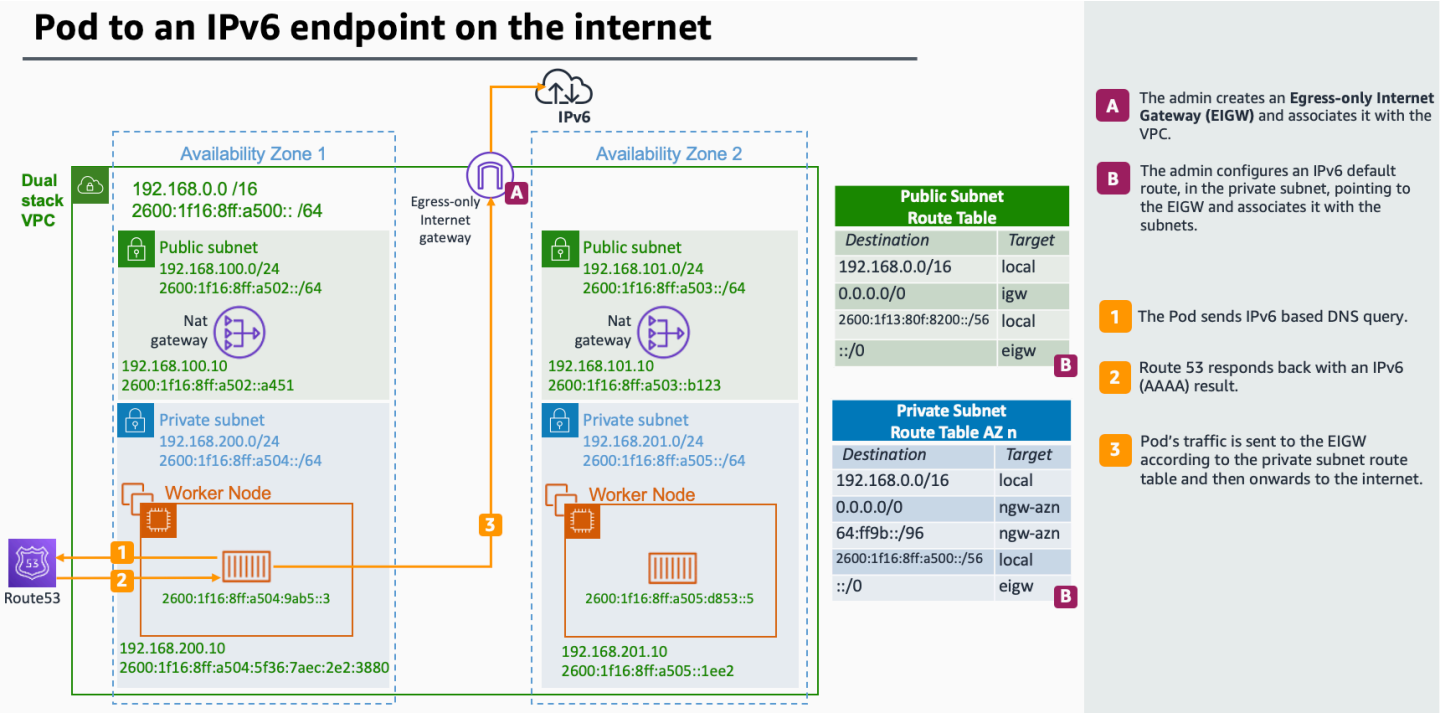
Em um cluster IPv6 EKS, os pods e os serviços receberão IPv6 endereços enquanto mantêm a compatibilidade com IPv4 endpoints antigos. Isso inclui a capacidade de IPv4 endpoints externos acessarem serviços no cluster e pods acessarem endpoints externos. IPv4

O IPv6 suporte do Amazon EKS aproveita os recursos nativos da IPv6 VPC. Cada VPC é alocada com um prefixo de IPv4 endereço (o tamanho do bloco CIDR pode ser de /16 a /28) e um prefixo de endereço /56 exclusivo (fixo) de dentro do GUA (IPv6 endereço unicast global) da Amazon; você pode atribuir um prefixo de endereço /64 a cada sub-rede em sua VPC. IPv4 recursos, como tabelas de rotas, listas de controle de acesso à rede, peering e resolução de DNS, funcionam da mesma forma em uma IPv6 VPC habilitada. A VPC é então chamada de VPC de pilha dupla, seguindo as sub-redes de pilha dupla. O diagrama a seguir mostra o padrão básico da VPC que oferece suporte a clusters baseados em EKS/: IPV4 IPv6 IPv6



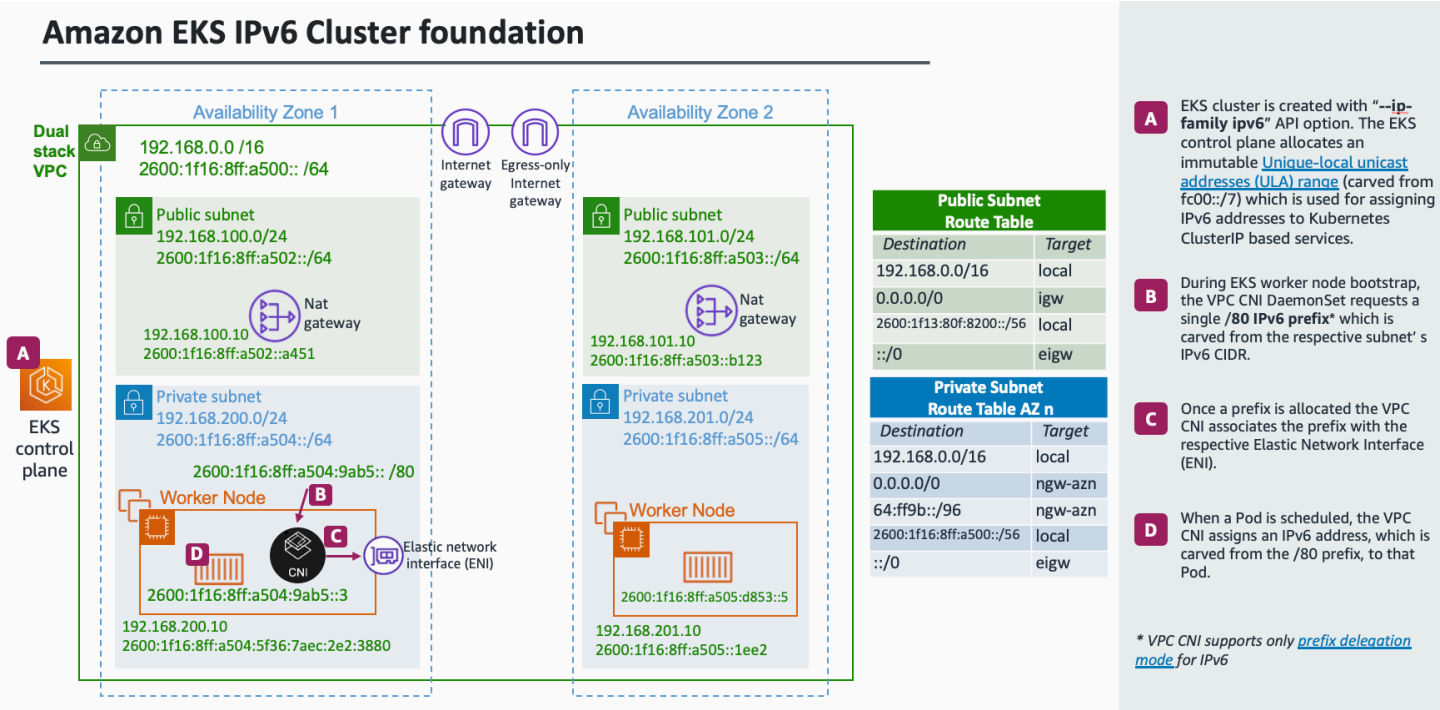
No IPv6 mundo, todo endereço é roteável pela Internet. Por padrão, a VPC aloca IPv6 CIDR do intervalo público do GUA. No entanto, desde agosto de 2024, você também pode usar IPv6 endereçamento privado VPCs e sub-redes com o Amazon VPC IP Address Manager (IPAM). Consulte esta postagem no blog da AWS Networking e a documentação da VPC para obter mais informações.

O diagrama a seguir mostra um fluxo de saída de IPv6 Internet do Pod dentro de um cluster IPv6 EKS/:



As melhores práticas para implementar IPv6 sub-redes podem ser encontradas no guia do usuário da VPC.

Em um cluster IPv6 EKS, nós e pods recebem IPv6 endereços públicos. O EKS atribui IPv6 endereços a serviços com base em endereços IPv6 unicast locais exclusivos (ULA). O CIDR do serviço ULA para um IPv6 cluster é atribuído automaticamente durante o estágio de criação do cluster e não pode ser especificado, ao contrário IPv4 de. O diagrama a seguir mostra um padrão básico de plano de dados do plano de controle de cluster IPv6 baseado em EKS/S:



Visão geral do

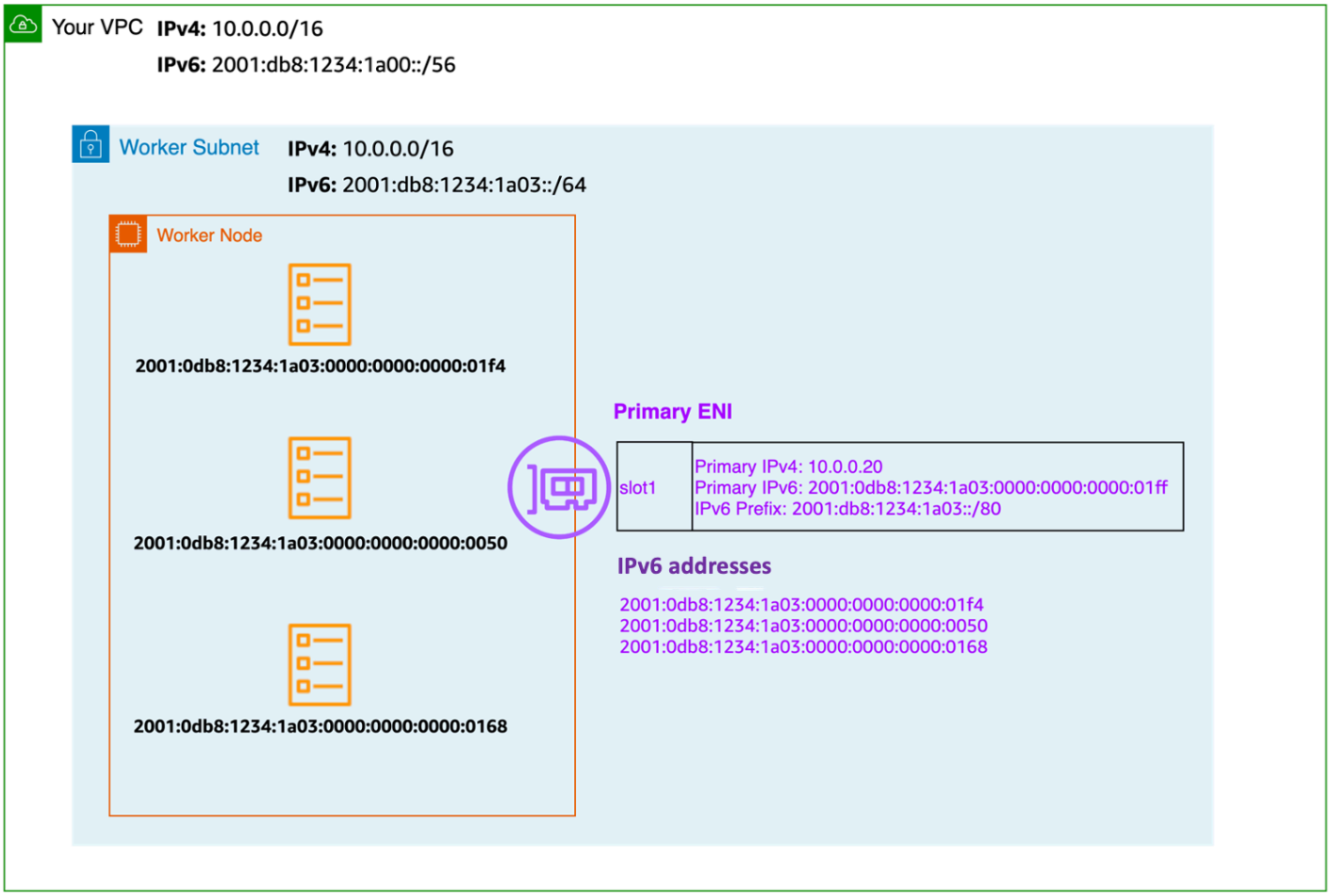
O EKS/ só IPv6 é suportado no modo de prefixo (modo de atribuição de IP ENI do plug-in VPC-CNI). Saiba mais sobre o Modo de prefixo.

A atribuição de prefixo só funciona em EC2 instâncias baseadas em Nitro, portanto, o EKS/ só IPv6 é suportado quando o plano de dados do cluster usa instâncias baseadas em Nitro. EC2

Em palavras simples, um IPv6 prefixo de /80 (por nó de trabalho) produzirá aproximadamente 10^14 IPv6 endereços. O fator limitante não será IPs mais a densidade do pod (em termos de recursos).

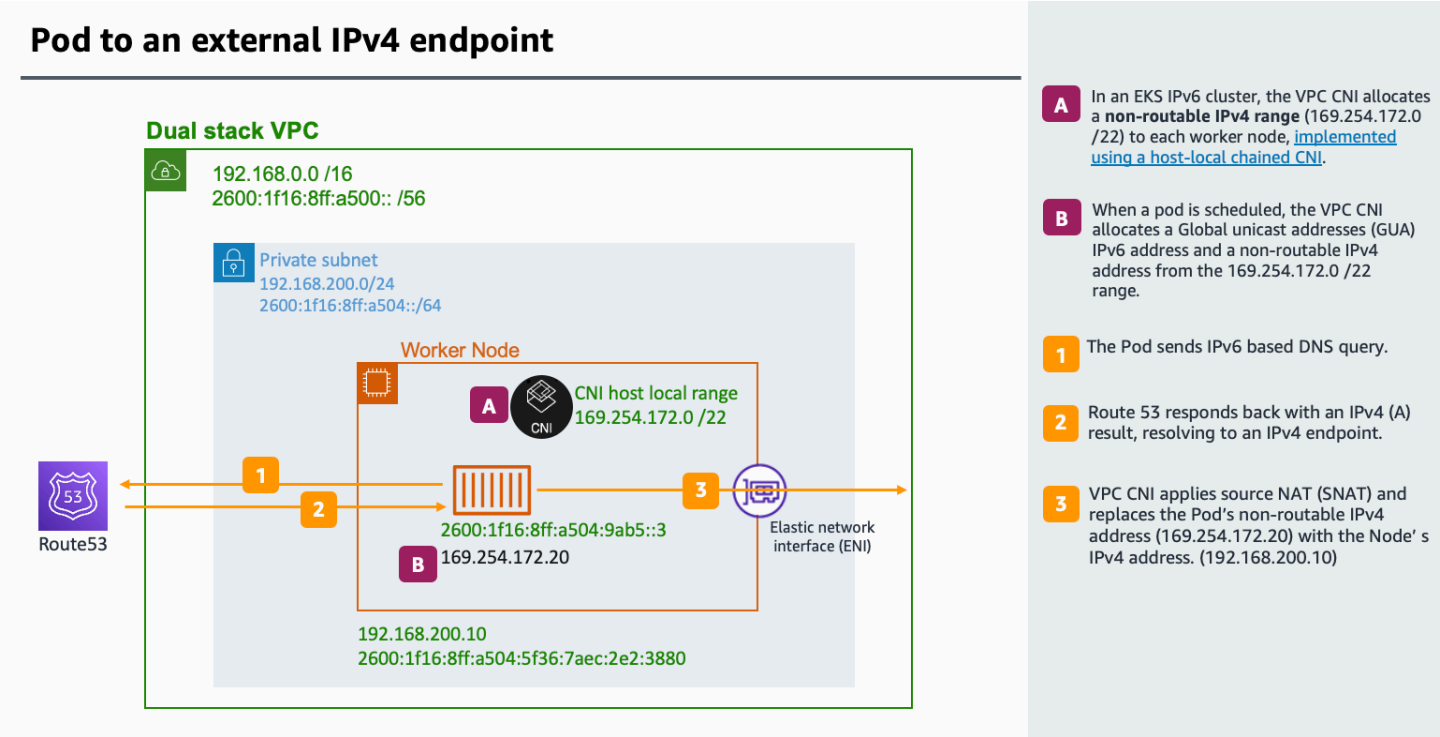
IPv6 a atribuição de prefixo ocorre apenas no momento do bootstrap do EKS worker-node. Sabe-se que esse comportamento atenua cenários em que IPv4 clusters EKS/S de alta rotatividade de Pod geralmente atrasam o agendamento de Pod devido a chamadas de API limitadas geradas pelo plug-in VPC CNI (ipamd) com o objetivo de alocar endereços privados em tempo hábil. IPv4 Também é conhecido por fazer com que os botões avançados do plug-in VPC-CNI ajustem WARM_IP/ENI, MINIMUM_IP desnecessariamente.

O diagrama a seguir amplia uma interface de rede elástica (IPv6 ENI) de nó de trabalho:



Cada nó de trabalho do EKS é atribuído com IPv6 endereços IPv4 e, juntamente com as entradas de DNS correspondentes. Para um determinado nó de trabalho, somente um único IPv4 endereço da sub-rede de pilha dupla é consumido. O suporte do EKS IPv6 permite que você se comunique com IPv4 endpoints (AWS, no local, Internet) por meio de um modelo somente de saída altamente opinativo. IPv4 O EKS implementa um plug-in CNI local do host, secundário ao plug-in VPC CNI, que aloca e configura um endereço para um pod. IPv4 O plug-in CNI configura um IPv4 endereço não roteável específico do host para um pod da faixa 169.254.172.0/22. O IPv4 endereço atribuído ao pod é exclusivo para o nó de trabalho e não é anunciado além do nó de trabalho. 169.254.172.0/22 fornece até 1024 endereços exclusivos que podem oferecer suporte a tipos de instâncias grandes. IPv4

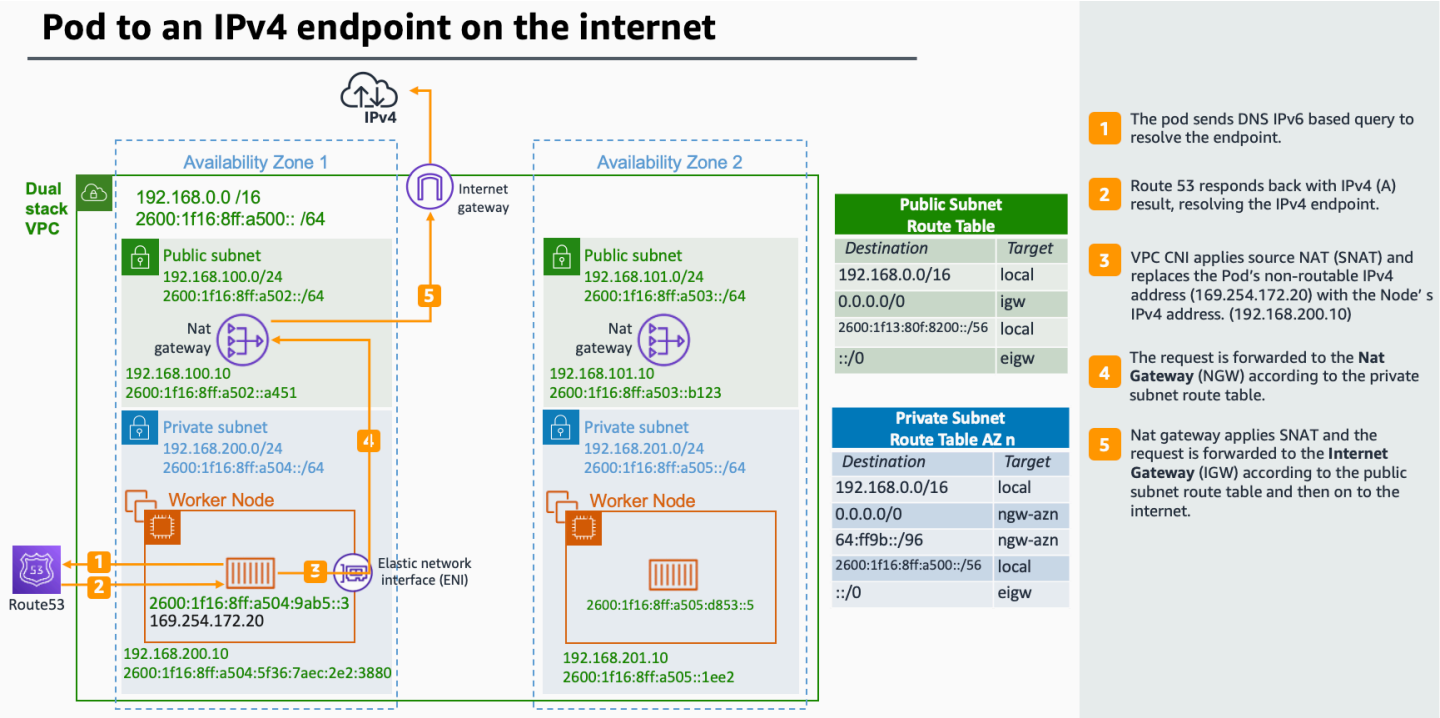
O diagrama a seguir mostra o fluxo de um IPv6 pod conectado a um IPv4 endpoint fora do limite do cluster (sem internet):



No diagrama acima, os Pods realizarão uma pesquisa de DNS para o endpoint e, ao receber uma resposta IPv4 “A”, o IPv4 endereço exclusivo exclusivo do Pod será traduzido por meio da tradução de endereços de rede de origem (SNAT) para o endereço privado (IPv4 VPC) da interface de rede primária conectada ao nó de trabalho. EC2

> **Note**
>
> O padrão acima requer DNS64 ser desativado em sub-redes em que o EKS/Pods está sendo executadoIPv6 . Quando DNS64 ativado, o resolvedor de DNS retorna um endereço sintetizado IPv4 somente para endpoints junto com um IPv6 endereço. IPv4 Como resultado, o tráfego passa pela NAT64 funcionalidade do NAT Gateway (se incluída na arquitetura) em vez de permanecer dentro da VPC, conforme mostrado no padrão acima. Isso pode levar ao uso inesperado do NAT Gateway e aos custos associados.

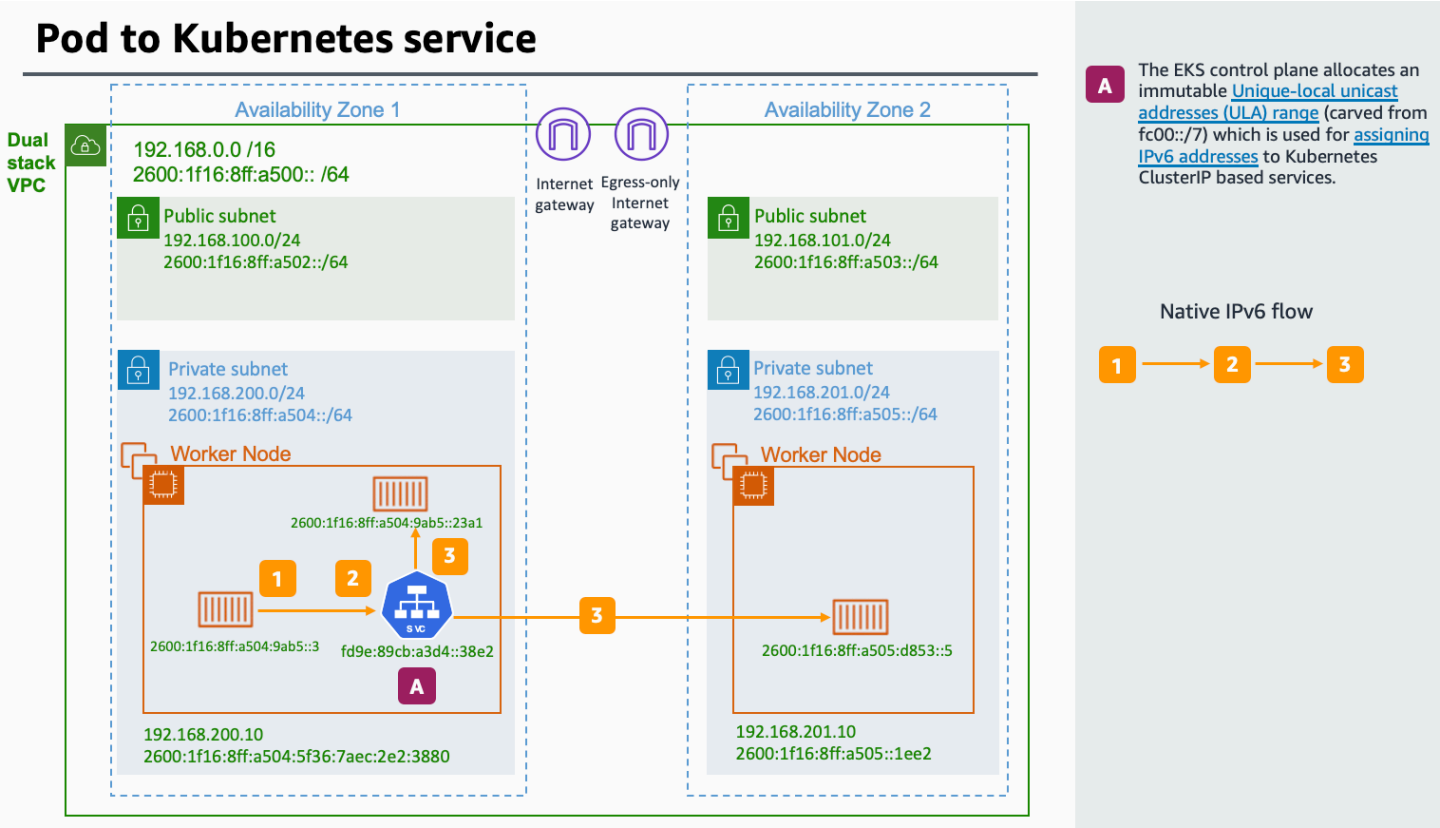
O IPv6 EKS/Pods também precisará se conectar a IPv4 endpoints pela Internet usando IPv4 endereços públicos, para que exista um fluxo semelhante. O diagrama a seguir mostra o fluxo de um IPv6 pod conectado a um IPv4 endpoint fora do limite do cluster (roteável pela Internet):



No diagrama acima, os Pods realizarão uma pesquisa de DNS para o endpoint e, ao receber uma resposta IPv4 “A”, o IPv4 endereço exclusivo exclusivo do Pod será traduzido por meio da tradução de endereços de rede de origem (SNAT) para o endereço privado (IPv4 VPC) da interface de rede primária conectada ao nó de trabalho. EC2 O IPv4 endereço do pod (fonte IPv4: IP EC2 primário) é então roteado para o gateway IPv4 NAT, onde o IP EC2 primário é traduzido (SNAT) em um endereço IP público roteável pela Internet válido (IP IPv4 público atribuído ao gateway NAT).

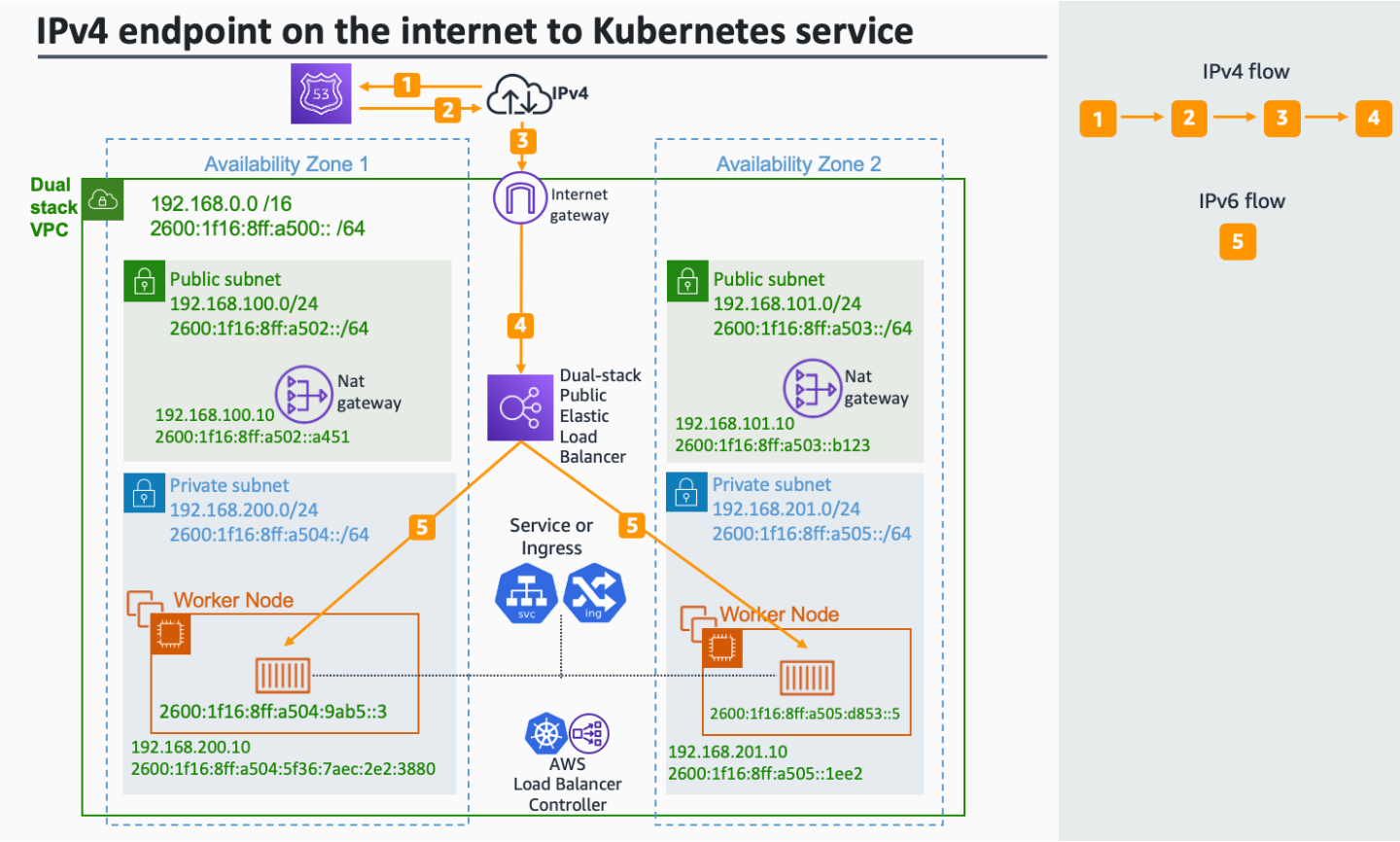
Qualquer Pod-to-Pod comunicação entre os nós sempre usa um IPv6 endereço. O VPC CNI configura iptables para IPv6 manipular enquanto bloqueia qualquer conexão. IPv4

Os serviços do Kubernetes receberão somente endereços IPv6 (ClusterIP) de endereços unicast locais exclusivos (ULA). IPv6 O CIDR do serviço ULA para um IPv6 cluster é atribuído automaticamente durante o estágio de criação do cluster EKS e não pode ser modificado. O diagrama a seguir mostra o fluxo do Pod to Kubernetes Service:



Os serviços são expostos à Internet usando um balanceador de carga da AWS. O balanceador de carga recebe IPv6 endereços públicos IPv4 e, também conhecido como balanceador de carga de pilha dupla. Para IPv4 clientes que acessam serviços de IPv6 cluster kubernetes, o balanceador de carga faz a tradução. IPv4 IPv6

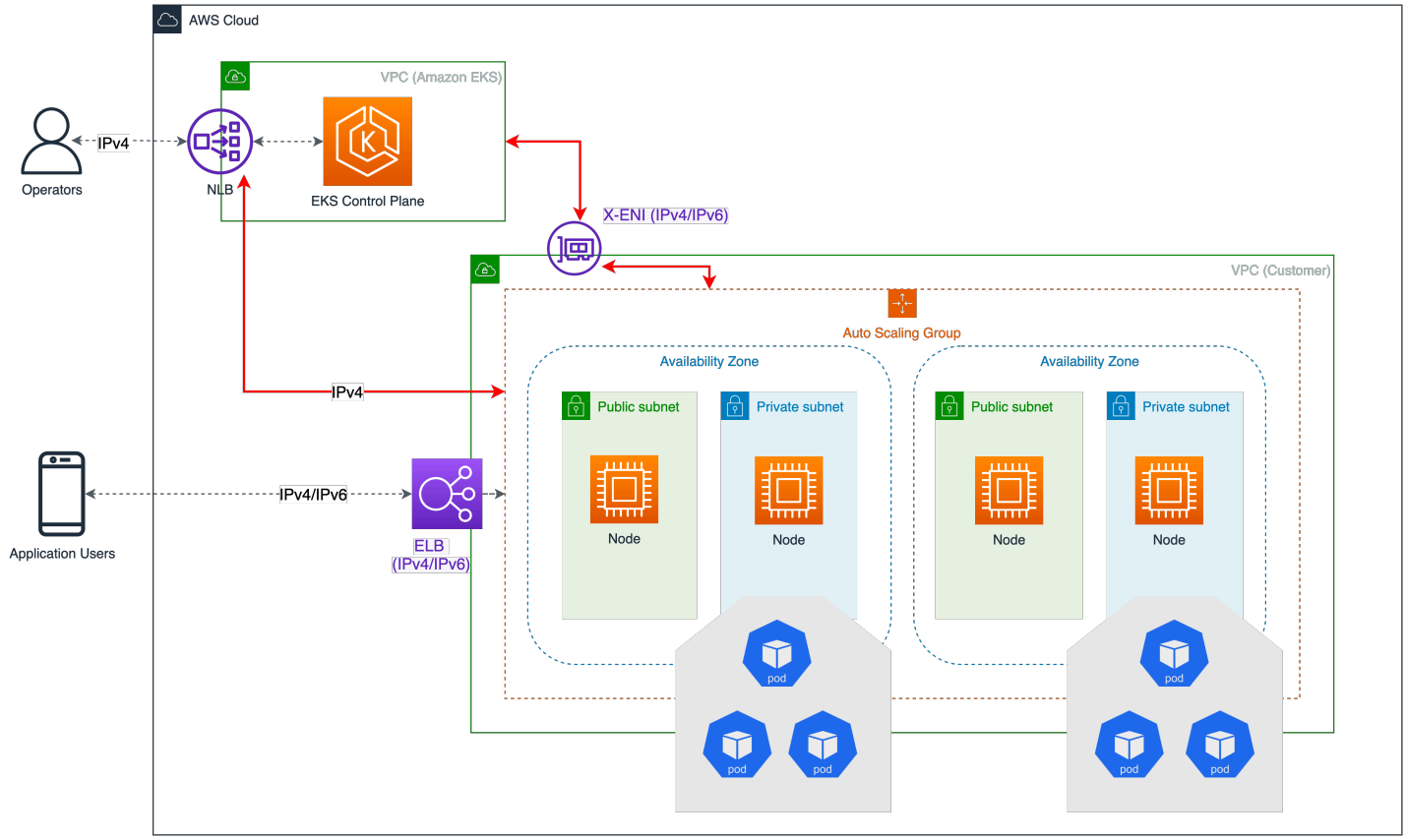
O Amazon EKS recomenda executar nós de trabalho e pods em sub-redes privadas. Você pode criar balanceadores de carga públicos nas sub-redes públicas que balanceiam a carga do tráfego para os pods executados em nós que estão em sub-redes privadas. O diagrama a seguir mostra um IPv4 usuário da Internet acessando um serviço baseado em IPv6 EKS/Ingress:



> **Note**
>
> O padrão acima exige a implantação da versão mais recente do controlador do balanceador de carga da AWS.

Comunicação do plano de dados do plano de controle EKS

O EKS provisionará contas cruzadas ENIs (X-ENIs) no modo de pilha dupla (IPv4/IPv6). Os componentes do nó do Kubernetes, como kubelet e kube-proxy, são configurados para oferecer suporte a pilha dupla. O Kubelet e o kube-proxy são executados no modo HostNetwork e se vinculam a ambos IPv4 e aos IPv6 endereços conectados à interface de rede primária de um nó. O servidor de API do Kubernetes se comunica com os pods e os componentes do node por meio do X-is. ENIs IPv6 Os pods se comunicam com os servidores de API por meio do X-ENIs, e a comunicação de pod para servidor de API sempre usa o modo. IPv6



Recomendações

Programação com base em recursos computacionais

Um único IPv6 prefixo é suficiente para executar vários pods em um único nó. Isso também remove efetivamente as limitações de ENI e IP do número máximo de pods em um nó. Embora IPv6 remova a dependência direta dos Max-pods, ao usar anexos de prefixo com tipos de instância menores, como o m5.large, é provável que você esgote os recursos de CPU e memória da instância muito antes de esgotar seus endereços IP. Você deve definir manualmente o valor máximo de pod recomendado pelo EKS se estiver usando grupos de nós autogerenciados ou um grupo de nós gerenciados com uma ID de AMI personalizada.

Você pode usar a fórmula a seguir para determinar o número máximo de pods que você pode implantar em um nó para um cluster IPv6 EKS.

```
((Number of network interfaces for instance type (number of prefixes per network
 interface-1)* 16) + 2
```

```
((3 ENIs)_((10 secondary IPs per ENI-1)_ 16)) + 2 = 460 (real)
```

Os grupos de nós gerenciados calculam automaticamente o número máximo de pods para você. Evite alterar o valor recomendado do EKS para o número máximo de pods para evitar falhas no agendamento de pods devido a limitações de recursos.

Avalie a finalidade da rede personalizada existente

Se a rede personalizada estiver habilitada no momento, o Amazon EKS recomenda reavaliar sua necessidade com. IPv6 Se você optar por usar uma rede personalizada para resolver o problema de IPv4 exaustão, ela não será mais necessária. IPv6 Se você estiver utilizando uma rede personalizada para atender a um requisito de segurança, como uma rede separada para nós e pods, recomendamos que envie uma solicitação de roteiro do EKS.

Fargate Pods em EKS/Cluster IPv6

O EKS é compatível com IPv6 pods executados no Fargate. Os pods executados no Fargate consumirão IPv4 endereços IPv6 privados roteáveis da VPC extraídos dos intervalos CIDR da VPC (). IPv4 IPv6 Em palavras simples, a ampla densidade do cluster de EKS/Fargate Pods será limitada à disponibilidade IPv4 e aos IPv6 endereços. É recomendável dimensionar sua pilha dupla para crescimento subnets/VPC CIDRs futuro. Você não poderá agendar novos Fargate Pods se a sub-rede subjacente não contiver um endereço disponível, independentemente dos IPv4 endereços disponíveis. IPv6

Implante o AWS Load Balancer Controller (LBC)

O controlador de serviço Kubernetes upstream em árvore não é compatível. IPv6 Recomendamos usar a versão mais recente do complemento AWS Load Balancer Controller. O LBC só implantará um NLB de pilha dupla ou um ALB de pilha dupla após consumir a definição de kubernetes correspondente anotada com: e service/ingress `"alb.ingress.kubernetes.io/ip-address-type: dualstack"` `"alb.ingress.kubernetes.io/target-type: ip"`

Rede personalizada

Por padrão, o Amazon VPC CNI atribuirá aos pods um endereço IP selecionado na sub-rede primária. A sub-rede primária é a CIDR à qual a ENI primária está conectada, geralmente a sub-rede do nó/host.

Se o CIDR da sub-rede for muito pequeno, talvez a CNI não consiga adquirir endereços IP secundários suficientes para atribuir aos seus pods. Esse é um desafio comum para IPv4 clusters EKS.

A rede personalizada é uma solução para esse problema.

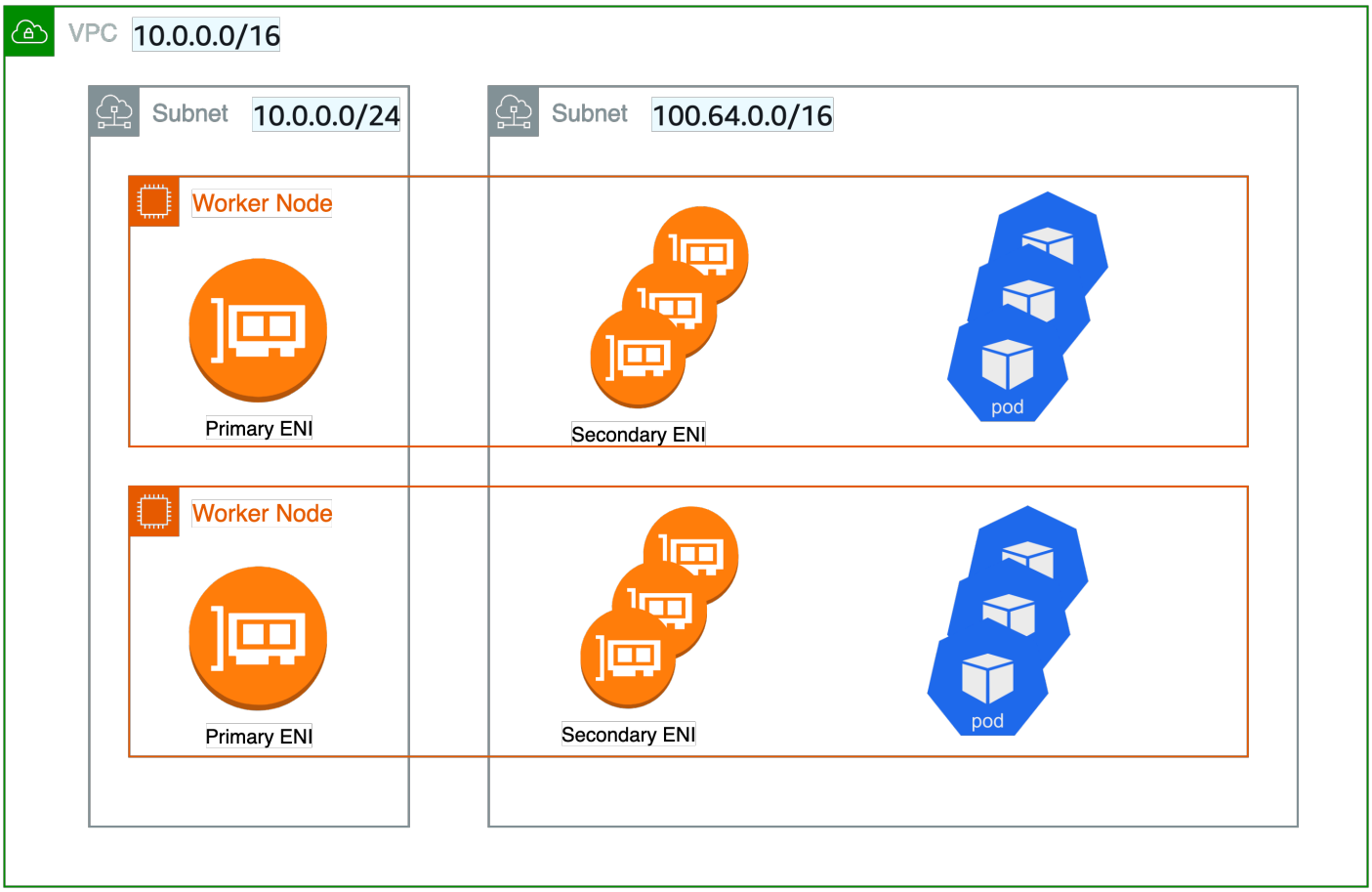
A rede personalizada resolve o problema de exaustão de IP atribuindo o nó e o pod dos espaços de endereço VPC IPs secundários (CIDR). O suporte de rede personalizado oferece suporte a recursos ENIConfig personalizados. ENIConfig Isso inclui um intervalo alternativo de CIDR de sub-rede (extraído de um CIDR VPC secundário), junto com os grupos de segurança aos quais os pods pertencerão. Quando a rede personalizada está habilitada, a VPC CNI cria uma secundária na sub-rede ENIs definida em. ENIConfig A CNI atribui aos pods endereços IP de um intervalo CIDR definido em um CRD. ENIConfig

Como a ENI primária não é usada pela rede personalizada, o número máximo de pods que você pode executar em um nó é menor. Os pods da rede host continuam usando o endereço IP atribuído à ENI primária. Além disso, a ENI primária é usada para lidar com a tradução da rede de origem e rotear o tráfego de pods para fora do nó.

Exemplo de configuração

Embora a rede personalizada aceite um intervalo VPC válido para o intervalo CIDR secundário, recomendamos que você use CIDRs (/16) do espaço CG-NAT, ou seja, 100.64.0.0/10 ou 198.19.0.0/16, pois é menos provável que sejam usados em uma configuração corporativa do que outros intervalos. RFC1918 Para obter informações adicionais sobre as associações de blocos CIDR permitidas e restritas que você pode usar com sua VPC, IPv4 consulte Restrições de associação de blocos CIDR na seção Dimensionamento de sub-rede e VPC da documentação da VPC.

Conforme mostrado no diagrama abaixo, a interface de rede elástica (ENI) primária do nó de trabalho ainda usa a faixa CIDR da VPC primária (nesse caso, 10.0.0.0/16), mas a secundária ENIs usa a faixa CIDR da VPC secundária (nesse caso, 100.64.0.0/16). Agora, para que os pods usem o intervalo CIDR 100.64.0.0/16, você deve configurar o plug-in CNI para usar redes personalizadas. Você pode seguir as etapas conforme documentado aqui.



Se você quiser que a CNI use uma rede personalizada, defina a variável de `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG` ambiente como`true`.

```
kubectl set env daemonset aws-node -n kube-system
 AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

Quando`AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true`, a CNI atribuirá o endereço IP do pod a partir de uma sub-rede definida em. `ENIConfig` O recurso `ENIConfig` personalizado é usado para definir a sub-rede na qual os pods serão programados.

```
apiVersion : crd.k8s.amazonaws.com/v1alpha1
kind : ENIConfig
metadata:
  name: us-west-2a
spec:
  securityGroups:
    - sg-0dff111a1d11c1c11
```

```
  subnet: subnet-011b111c1f11fdf11
```

Ao criar os recursos `ENIconfig` personalizados, você precisará criar novos nós de trabalho e drenar os nós existentes. Os nós de trabalho e os pods existentes permanecerão inalterados.

Recomendações

Use redes personalizadas quando

Recomendamos que você considere a criação de redes personalizadas se estiver com IPv4 exaustão e ainda não puder usá-la. IPv6 O suporte do Amazon EKS para RFC6598espaço permite que você escale pods além de RFC1918enfrentar os desafios de exaustão. Considere usar a delegação de prefixos com redes personalizadas para aumentar a densidade de pods em um nó.

Você pode considerar uma rede personalizada se tiver um requisito de segurança para executar pods em uma rede diferente com requisitos de grupo de segurança diferentes. Quando a rede personalizada é ativada, os pods usam grupos de sub-rede ou de segurança diferentes, conforme definido na ENIConfig interface de rede primária do nó.

A rede personalizada é, de fato, uma opção ideal para implantar vários clusters e aplicativos EKS para conectar serviços de datacenter local. Você pode aumentar o número de endereços privados (RFC1918) acessíveis ao EKS em sua VPC para serviços como Amazon Elastic Load Balancing e NAT-GW, enquanto usa espaço CG-NAT não roteável para seus pods em vários clusters. A rede personalizada com o gateway de trânsito e uma VPC de Serviços Compartilhados (incluindo gateways NAT em várias zonas de disponibilidade para alta disponibilidade) permite que você forneça fluxos de tráfego escaláveis e previsíveis. Esta postagem do blog descreve um padrão arquitetônico que é uma das formas mais recomendadas de conectar EKS Pods a uma rede de datacenter usando redes personalizadas.

Evite redes personalizadas quando

Pronto para implementar IPv6

A rede personalizada pode mitigar os problemas de exaustão de IP, mas exige sobrecarga operacional adicional. Se você está implantando atualmente uma IPv6 VPC de pilha dupla IPv4 (/) ou se seu plano inclui IPv6 suporte, recomendamos implementar clusters. IPv6 Você pode configurar clusters IPv6 EKS e migrar seus aplicativos. Em um cluster IPv6 EKS, tanto o Kubernetes quanto os pods recebem um IPv6 endereço e podem se comunicar de entrada e saída com ambos os endpoints. IPv4 IPv6 Consulte as melhores práticas para executar clusters IPv6 EKS.

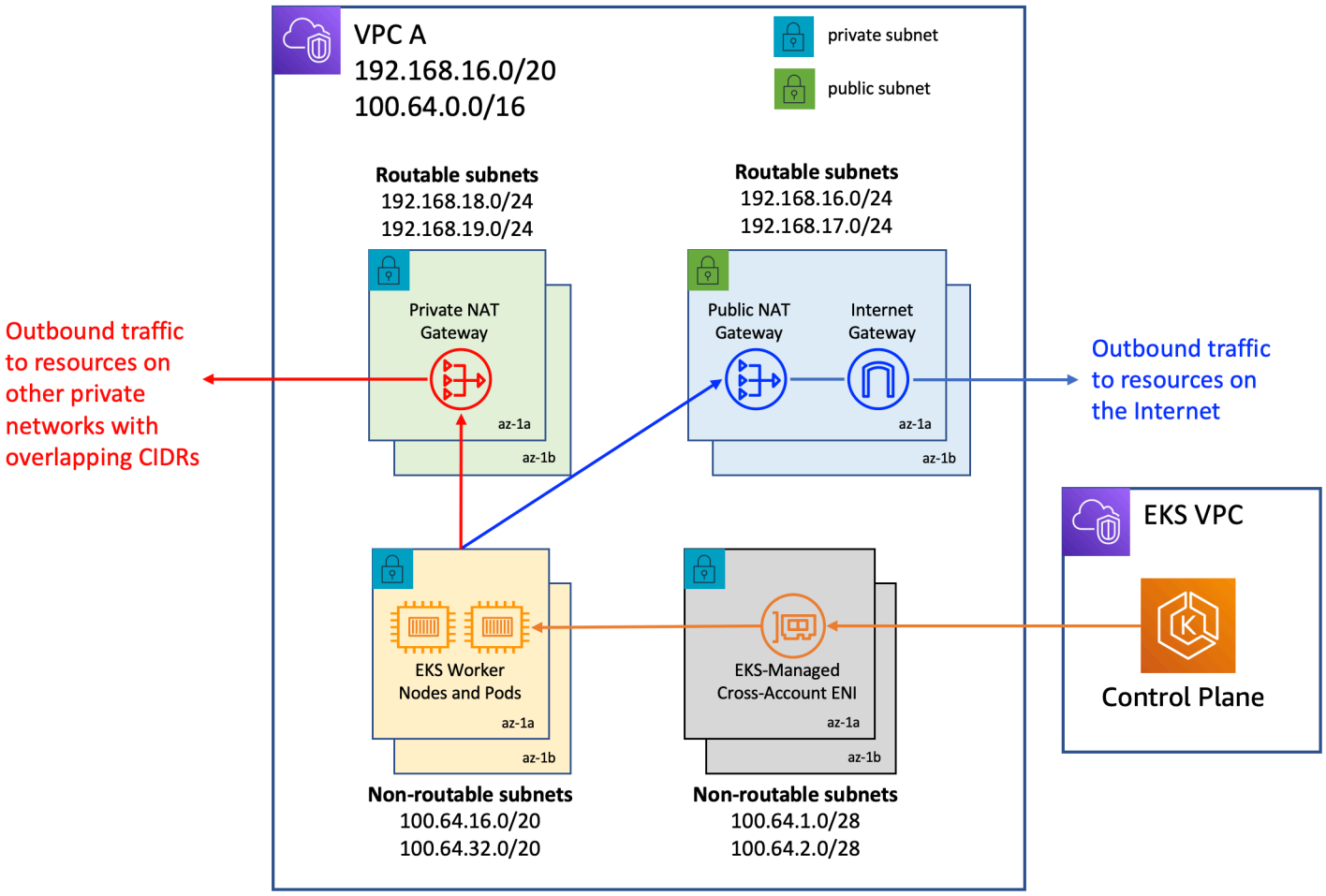
Espaço CG-NAT esgotado

Além disso, se você está utilizando atualmente a CIDRs partir do espaço CG-NAT ou não consegue vincular um CIDR secundário à sua VPC de cluster, talvez seja necessário explorar outras opções, como usar uma CNI alternativa. É altamente recomendável que você obtenha suporte comercial ou possua o conhecimento interno para depurar e enviar patches para o projeto de plug-in CNI de código aberto. Consulte o guia do usuário de plug-ins CNI alternativos para obter mais detalhes.

Use o gateway NAT privado

A Amazon VPC agora oferece recursos de gateway NAT privado. O NAT Gateway privado da Amazon permite que instâncias em sub-redes privadas se conectem a outras VPCs redes locais com sobreposição. CIDRs Considere utilizar o método descrito nesta postagem do blog para empregar um gateway NAT privado para superar os problemas de comunicação das cargas de trabalho do EKS causados pela sobreposição CIDRs, uma reclamação significativa expressa por nossos clientes. A rede personalizada não pode resolver sozinha as dificuldades sobrepostas do CIDR e aumenta os desafios de configuração.

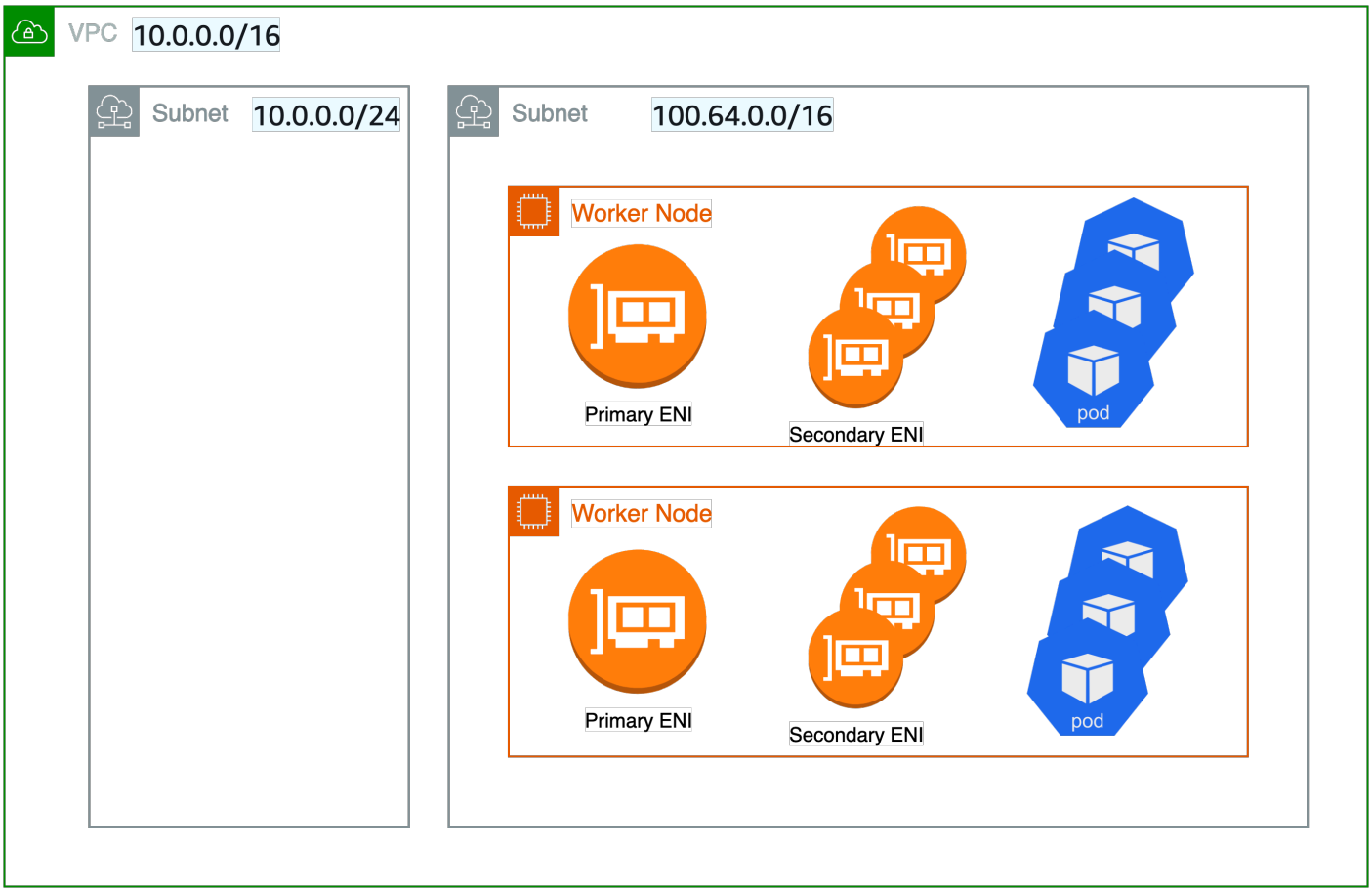
A arquitetura de rede usada na implementação desta postagem do blog segue as recomendações em Habilitar a comunicação entre redes sobrepostas na documentação da Amazon VPC. Conforme demonstrado nesta postagem do blog, você pode expandir o uso do NAT Gateway privado em conjunto com RFC6598 endereços para gerenciar os problemas de exaustão de IP privado dos clientes. Os clusters EKS e os nós de trabalho são implantados no intervalo CIDR secundário não roteável de 100.64.0.0/16 VPC, enquanto o gateway NAT privado e o gateway NAT são implantados nos intervalos CIDR roteáveis. RFC1918 O blog explica como um gateway de trânsito é usado para se conectar a VPCs fim de facilitar a comunicação VPCs com intervalos CIDR não roteáveis sobrepostos. Para casos de uso em que os recursos do EKS no intervalo de endereços não roteáveis de uma VPC precisam se comunicar com outros VPCs que não tenham intervalos de endereços sobrepostos, os clientes têm a opção de usar o VPC Peering para interconectá-los. VPCs Esse método pode proporcionar uma possível economia de custos, pois todo o trânsito de dados dentro de uma zona de disponibilidade por meio de uma conexão de emparelhamento de VPC agora é gratuito.



Rede exclusiva para nós e pods

Se você precisar isolar seus nós e pods em uma rede específica por motivos de segurança, recomendamos que você implante nós e pods em uma sub-rede a partir de um bloco CIDR secundário maior (por exemplo, 100.64.0.0/8). Após a instalação do novo CIDR em sua VPC, você pode implantar outro grupo de nós usando o CIDR secundário e drenar os nós originais para reimplantar automaticamente os pods nos novos nós de trabalho. Para obter mais informações sobre como implementar isso, consulte esta postagem no blog.

A rede personalizada não é usada na configuração representada no diagrama abaixo. Em vez disso, os nós de trabalho do Kubernetes são implantados em sub-redes do intervalo CIDR de VPC secundário da sua VPC, como 100.64.0.0/10. Você pode manter o cluster EKS funcionando (o plano de controle permanecerá no original)subnet/s), but the nodes and Pods will be moved to a secondary subnet/s. Essa é outra técnica, embora não convencional, para mitigar o perigo de exaustão de IP em uma VPC. Propomos drenar os nós antigos antes de reimplantar os pods nos novos nós de trabalho.



Automatize a configuração com rótulos de zona de disponibilidade

Você pode permitir que o Kubernetes aplique automaticamente o correspondente ENIConfig à Zona de Disponibilidade (AZ) do nó de trabalho.

O Kubernetes adiciona automaticamente a tag `topology.kubernetes.io/zone` aos seus nós de trabalho. O Amazon EKS recomenda usar a zona de disponibilidade como seu nome de configuração de ENI quando você tem apenas uma sub-rede secundária (CIDR alternativo) por AZ. Em seguida, você pode definir o rótulo usado para descobrir o nome da configuração ENI como. `topology.kubernetes.io/zone` Observe que a tag `failure-domain.beta.kubernetes.io/zone` está obsoleta e foi substituída pela tag. `topology.kubernetes.io/zone`

1. Defina o `name` campo como a zona de disponibilidade da sua VPC.
2. Ative a configuração automática por meio do seguinte comando
3. Defina o rótulo de configuração por meio do seguinte 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"
```

Se você tiver várias sub-redes secundárias por zona de disponibilidade, precisará criar uma específica. `ENI_CONFIG_LABEL_DEF` Você pode considerar a configuração de nós `ENI_CONFIG_LABEL_DEF` como `k8s.amazonaws.com/eniConfig`e label com nomes EniConfig personalizados, como e. `k8s.amazonaws.com/eniConfig=us-west-2a-subnet-1k8s.amazonaws.com/eniConfig=us-west-2a-subnet-2`

Substitua os pods ao configurar a rede secundária

A ativação da rede personalizada não modifica os nós existentes. A rede personalizada é uma ação disruptiva. Em vez de fazer uma substituição contínua de todos os nós de trabalho em seu cluster depois de ativar a rede personalizada, sugerimos atualizar o CloudFormation modelo da AWS no Guia de introdução do EKS com um recurso personalizado que chama uma função Lambda para atualizar o `aws-node` Daemonset com a variável de ambiente para habilitar a rede personalizada antes que os nós de trabalho sejam provisionados.

Se você tinha algum nó em seu cluster com pods em execução antes de mudar para o recurso de rede CNI personalizado, você deve isolar e drenar os nós para desligar os pods normalmente e, em seguida, encerrar os nós. Somente novos nós que correspondam ao ENIConfig rótulo ou às anotações usam redes personalizadas e, portanto, os pods programados nesses novos nós podem receber um IP do CIDR secundário.

Calcule o máximo de pods por nó

Como a ENI principal do nó não é mais usada para atribuir endereços IP de pod, há uma diminuição no número de pods que você pode executar em um determinado tipo de EC2 instância. Para contornar essa limitação, você pode usar a atribuição de prefixo com redes personalizadas. Com a atribuição de prefixo, cada IP secundário é substituído por um prefixo /28 no secundário. ENIs

Considere o número máximo de pods para uma instância m5.large com rede personalizada.

O número máximo de pods que você pode executar sem atribuição de prefixo é 29

- `3 ENIs - 1) * (10 secondary IPs per ENI - 1 + 2 = 20`

Ativar anexos de prefixo aumenta o número de pods para 290.

- `(3 ENIs - 1) * ((10 secondary IPs per ENI - 1) * 16 + 2 = 290`

No entanto, sugerimos definir max-pods para 110 em vez de 290 porque a instância tem um número bem pequeno de virtuais. CPUs Em instâncias maiores, o EKS recomenda um valor máximo de pods de 250. Ao utilizar anexos de prefixo com tipos de instância menores (por exemplo, m5.large), é possível que você esgote os recursos de CPU e memória da instância bem antes dos endereços IP.

> **Note**
>
> Quando o prefixo CNI aloca um prefixo /28 para uma ENI, ele precisa ser um bloco contíguo de endereços IP. Se a sub-rede da qual o prefixo é gerado estiver altamente fragmentada, o anexo do prefixo poderá falhar. Você pode evitar que isso aconteça criando uma nova VPC dedicada para o cluster ou reservando um conjunto de CIDR para a sub-rede exclusivamente para anexos de prefixo. Visite Reservas CIDR de sub-rede para obter mais informações sobre esse tópico.

Identifique o uso existente do espaço CG-NAT

A rede personalizada permite que você reduza o problema de exaustão de IP, mas não pode resolver todos os desafios. Se você já usa o espaço CG-NAT para seu cluster ou simplesmente não tem a capacidade de associar um CIDR secundário à sua VPC de cluster, sugerimos que explore outras opções, como usar uma CNI alternativa ou migrar para clusters. IPv6

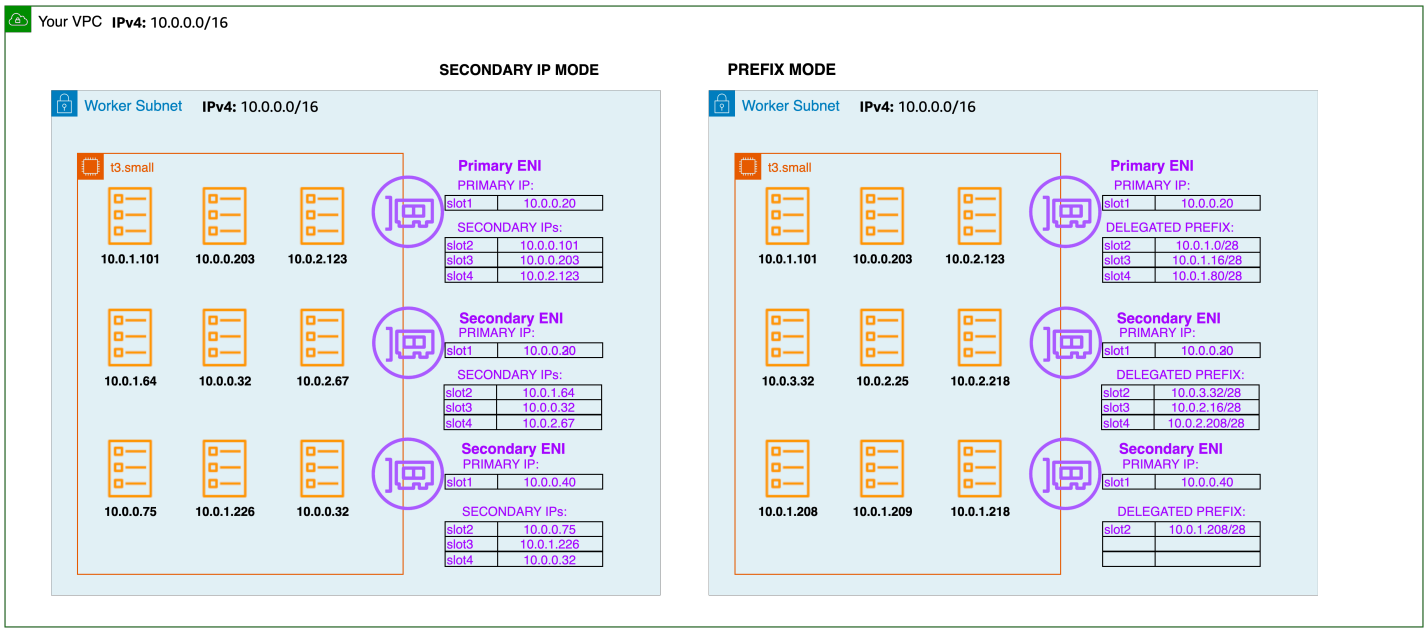
Modo de prefixo para Linux

O Amazon VPC CNI atribui prefixos de rede às interfaces de EC2 rede da Amazon para aumentar o número de endereços IP disponíveis para os nós e aumentar a densidade de pods por nó. Você pode configurar a versão 1.9.0 ou posterior do complemento CNI da Amazon VPC para atribuir IPv4 e em IPv6 CIDRs vez de atribuir endereços IP secundários individuais às interfaces de rede.

O modo de prefixo é ativado por padrão nos IPv6 clusters e é a única opção suportada. O VPC CNI atribui um IPv6 prefixo /80 a um slot em um ENI. Consulte a IPv6 seção deste guia para obter mais informações.

Com o modo de atribuição de prefixo, o número máximo de interfaces de rede elástica por tipo de instância permanece o mesmo, mas agora você pode configurar o Amazon VPC CNI para atribuir

prefixos de IPv4 endereço /28 (16 endereços IP), em vez de atribuir endereços individuais IPv4 aos slots nas interfaces de rede. Quando `ENABLE_PREFIX_DELEGATION` definido como verdadeiro, o VPC CNI aloca um endereço IP para um pod a partir do prefixo atribuído a um ENI. Siga as instruções mencionadas no guia do usuário do EKS para ativar o modo Prefix IP.

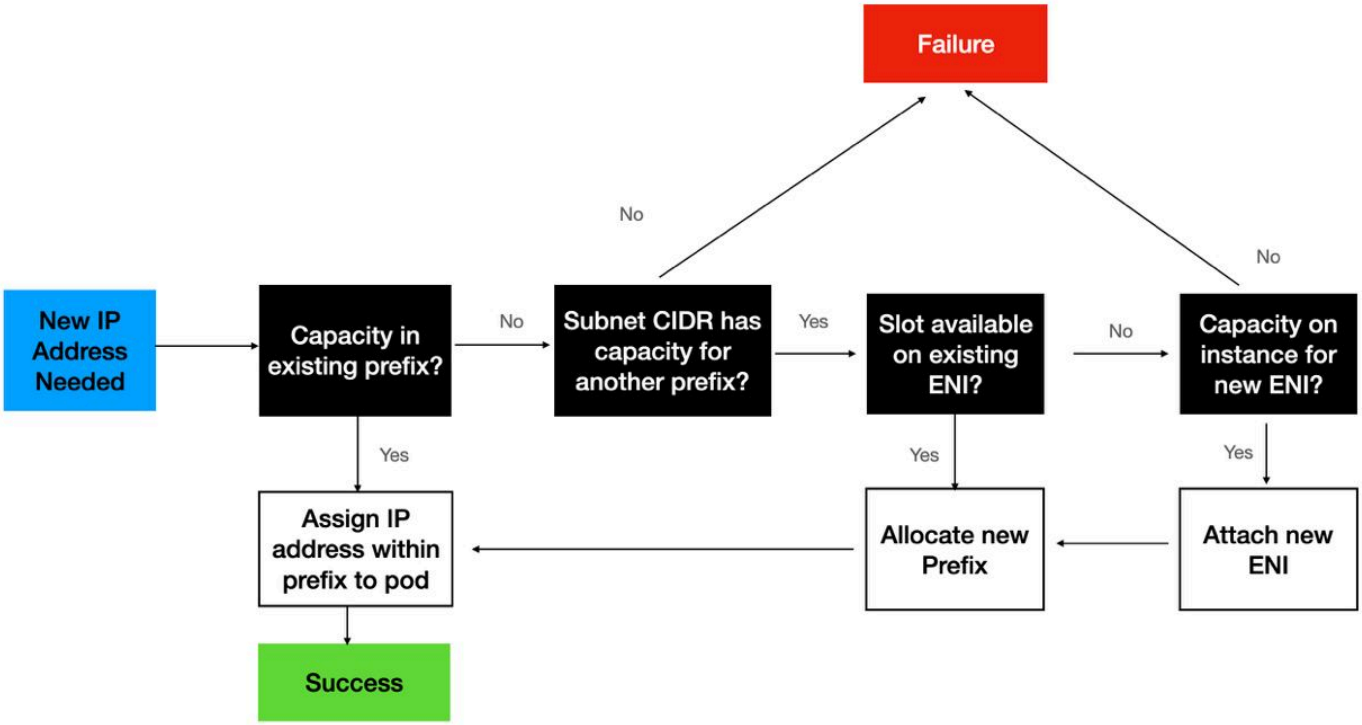


O número máximo de endereços IP que podem ser atribuídos a uma interface de rede depende do tipo de instância. Todo prefixo que você atribui a uma interface de rede conta como um endereço IP. Por exemplo, uma `c5.large` instância tem um limite de `10` IPv4 endereços por interface de rede. Cada interface de rede dessa instância tem um IPv4 endereço primário. Se uma interface de rede não tiver IPv4 endereços secundários, você poderá atribuir até 9 prefixos à interface de rede. Para cada IPv4 endereço adicional atribuído a uma interface de rede, você pode atribuir um prefixo a menos à interface de rede. Analise a EC2 documentação da AWS sobre endereços IP por interface de rede por tipo de instância e atribuição de prefixos às interfaces de rede.

Durante a inicialização do nó de trabalho, a CNI da VPC atribui um ou mais prefixos à ENI primária. O CNI pré-aloca um prefixo para uma inicialização mais rápida do pod, mantendo uma piscina aquecida. O número de prefixos a serem mantidos em uma piscina aquecida pode ser controlado definindo variáveis de ambiente.

- `WARM_PREFIX_TARGET`, o número de prefixos a serem alocados além da necessidade atual.
- `WARM_IP_TARGET`, o número de endereços IP a serem alocados além da necessidade atual.
- `MINIMUM_IP_TARGET`, o número mínimo de endereços IP disponíveis a qualquer momento.
- `WARM_IP_TARGETe`, `MINIMUM_IP_TARGET` se definido, `WARM_PREFIX_TARGET` substituirá.

À medida que mais pods forem programados, prefixos adicionais serão solicitados para o ENI existente. Primeiro, a VPC CNI tenta alocar um novo prefixo para uma ENI existente. Se a ENI estiver cheia, a VPC CNI tentará alocar uma nova ENI para o nó. O novo ENIs será anexado até que o limite máximo de ENI (definido pelo tipo de instância) seja atingido. Quando uma nova ENI é anexada, o ipamd alocará um ou mais prefixos necessários para manter a configuraçãoWARM_PREFIX_TARGET, e. WARM_IP_TARGET MINIMUM_IP_TARGET



Recomendações

Use o modo de prefixo quando

Use o modo de prefixo se você estiver enfrentando problemas de densidade de pods nos nós de trabalho. Para evitar erros de VPC CNI, recomendamos examinar as sub-redes em busca de blocos contíguos de endereços para o prefixo /28 antes de migrar para o modo de prefixo. Consulte a seção "Usar reservas de sub-rede para evitar a fragmentação da sub-rede (IPv4)" para obter detalhes da reserva de sub-rede.

Para compatibilidade com versões anteriores, o limite máximo de pods é definido para suportar o modo IP secundário. Para aumentar a densidade do pod, especifique o `max-pods` valor para Kubelet e `--use-max-pods=false` como dados do usuário para os nós. Você pode considerar usar o max-pod-calculatorscript.sh para calcular o número máximo recomendado de pods do EKS

para um determinado tipo de instância. Consulte o guia do usuário do EKS para ver, por exemplo, os dados do usuário.

```
./max-pods-calculator.sh --instance-type m5.large --cni-version ``1.9``.0 --cni-prefix-delegation-enabled
```

O modo de atribuição de prefixo é especialmente relevante para usuários da rede personalizada CNI, em que a ENI primária não é usada para pods. Com a atribuição de prefixos, você ainda pode anexar mais IPs em quase todos os tipos de instância do Nitro, mesmo sem a ENI primária usada para pods.

Evite o modo de prefixo quando

Se sua sub-rede estiver muito fragmentada e não tiver endereços IP disponíveis suficientes para criar prefixos /28, evite usar o modo de prefixo. O anexo do prefixo pode falhar se a sub-rede da qual o prefixo é produzido estiver fragmentada (uma sub-rede muito usada com endereços IP secundários dispersos). Esse problema pode ser evitado criando uma nova sub-rede e reservando um prefixo.

No modo prefixo, o grupo de segurança atribuído aos nós de trabalho é compartilhado pelos pods. Considere usar grupos de segurança para pods se você tiver um requisito de segurança para obter conformidade executando aplicativos com diferentes requisitos de segurança de rede em recursos computacionais compartilhados.

Use tipos de instância semelhantes no mesmo grupo de nós

Seu grupo de nós pode conter instâncias de vários tipos. Se uma instância tiver uma contagem máxima baixa de pods, esse valor será aplicado a todos os nós no grupo de nós. Considere usar tipos de instância semelhantes em um grupo de nós para maximizar o uso dos nós. Recomendamos configurar node.kubernetes.io/instance-type na parte de requisitos da API do provisionador se você estiver usando o Karpenter para escalonamento automatizado de nós.

> **Warning**
>
> A contagem máxima de pods para todos os nós em um determinado grupo de nós é definida pela menor contagem máxima de pods de qualquer tipo de instância única no grupo de nós.

Configure **WARM_PREFIX_TARGET** para conservar endereços IPv4

O valor padrão do manifesto de instalação `WARM_PREFIX_TARGET` é 1. Na maioria dos casos, o valor recomendado de 1 for `WARM_PREFIX_TARGET` fornecerá uma boa combinação de tempos rápidos de inicialização do pod e minimizará os endereços IP não utilizados atribuídos à instância.

Se você precisar conservar ainda mais os IPv4 endereços por nó, use `WARM_IP_TARGET` e `MINIMUM_IP_TARGET` configurações, que são substituídos `WARM_PREFIX_TARGET` quando configurados. Ao `WARM_IP_TARGET` definir um valor menor que 16, você pode impedir que o CNI mantenha um prefixo inteiro em excesso anexado.

Prefiro alocar novos prefixos em vez de anexar um novo ENI

Alocar um prefixo adicional a uma ENI existente é uma operação de EC2 API mais rápida do que criar e anexar uma nova ENI à instância. O uso de prefixos melhora o desempenho e, ao mesmo tempo, reduz a alocação de IPv4 endereços. A anexação de um prefixo normalmente é concluída em menos de um segundo, enquanto a anexação de uma nova ENI pode levar até 10 segundos. Para a maioria dos casos de uso, a CNI precisará apenas de uma única ENI por nó de trabalho ao ser executada no modo prefixo. Se você puder pagar (na pior das hipóteses) até 15 não utilizados IPs por nó, é altamente recomendável usar o novo modo de rede de atribuição de prefixos e obter os ganhos de desempenho e eficiência decorrentes dele.

Use reservas de sub-rede para evitar a fragmentação da sub-rede () IPv4

Quando EC2 aloca um IPv4 prefixo /28 para uma ENI, ele precisa ser um bloco contíguo de endereços IP da sua sub-rede. Se a sub-rede da qual o prefixo é gerado estiver fragmentada (uma sub-rede muito usada com endereços IP secundários dispersos), o anexo do prefixo poderá falhar e você verá a seguinte mensagem de erro nos registros da VPC CNI:

```
failed to allocate a private IP/Prefix address: InsufficientCidrBlocks: There are not
 enough free cidr blocks in the specified subnet to satisfy the request.
```

Para evitar a fragmentação e ter espaço contíguo suficiente para criar prefixos, você pode usar reservas CIDR de sub-rede VPC para reservar espaço IP em uma sub-rede para uso exclusivo de prefixos. Depois de criar uma reserva, o plug-in VPC CNI chamará EC2 APIs para atribuir prefixos que são alocados automaticamente do espaço reservado.

É recomendável criar uma nova sub-rede, reservar espaço para prefixos e habilitar a atribuição de prefixos com VPC CNI para nós de trabalho executados nessa sub-rede. Se a nova sub-rede for

dedicada somente aos pods em execução no seu cluster EKS com a atribuição de prefixo VPC CNI ativada, você poderá pular a etapa de reserva do prefixo.

Evite fazer o downgrade da VPC CNI

O modo de prefixo funciona com o VPC CNI versão 1.9.0 e posterior. O downgrade do complemento CNI da Amazon VPC para uma versão inferior à 1.9.0 deve ser evitado quando o modo de prefixo estiver ativado e os prefixos forem atribuídos a. ENIs Você deve excluir e recriar nós se decidir fazer o downgrade da VPC CNI.

Substitua todos os nós durante a transição para a Delegação de Prefixo

É altamente recomendável criar novos grupos de nós para aumentar o número de endereços IP disponíveis, em vez de fazer a substituição contínua dos nós de trabalho existentes. Isole e drene todos os nós existentes para expulsar com segurança todos os seus pods existentes. Para evitar interrupções no serviço, sugerimos implementar Pod Disruption Budgets em seus clusters de produção para cargas de trabalho críticas. Os pods em novos nós receberão um IP a partir de um prefixo atribuído a uma ENI. Depois de confirmar que os pods estão em execução, você pode excluir os nós e grupos de nós antigos. Se você estiver usando grupos de nós gerenciados, siga as etapas mencionadas aqui para excluir com segurança um grupo de nós.

Modo de prefixo para Windows

No Amazon EKS, por padrão, cada pod executado em um host Windows recebe um endereço IP secundário pelo controlador de recursos da VPC. Esse endereço IP é um endereço roteável por VPC que é alocado da sub-rede do host. No Linux, cada ENI anexada à instância tem vários slots que podem ser preenchidos por um endereço IP secundário ou um CIDR /28 (um prefixo). No entanto, os hosts Windows suportam apenas uma única ENI e seus slots disponíveis. Usar somente endereços IP secundários pode limitar artificialmente o número de pods que você pode executar em um host Windows, mesmo quando há muitos endereços IP disponíveis para atribuição.

Para aumentar a densidade de pods em hosts Windows, especialmente ao usar tipos de instância menores, você pode habilitar a Delegação de Prefixo para nós do Windows. Quando a delegação de prefixos está habilitada, IPv4 os prefixos /28 são atribuídos aos slots ENI em vez de endereços IP secundários. A delegação de prefixo pode ser habilitada adicionando a `enable-windows-prefix-delegation: "true"` entrada ao mapa de `amazon-vpc-cni` configuração. Esse é o mesmo mapa de configuração em que você precisa definir a `enable-windows-ipam: "true"` entrada para habilitar o suporte ao Windows.

Siga as instruções mencionadas no guia do usuário do EKS para ativar o modo de delegação de prefixo para nós do Windows.

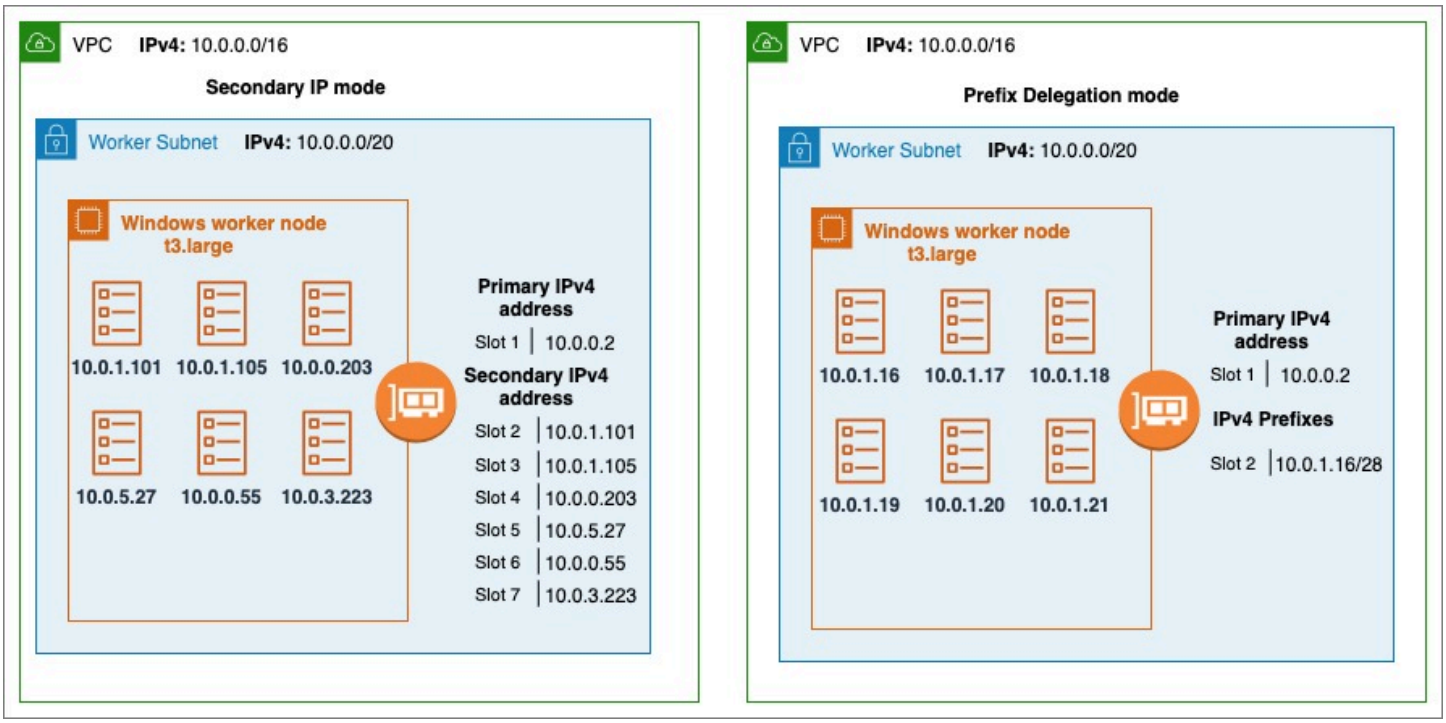


Figura: Comparação do modo IP secundário com o modo de delegação de prefixo

O número máximo de endereços IP que você pode atribuir a uma interface de rede depende do tipo e do tamanho da instância. Cada prefixo atribuído a uma interface de rede consome um slot disponível. Por exemplo, uma `c5.large` instância tem um limite de `10` slots por interface de rede. O primeiro slot em uma interface de rede é sempre consumido pelo endereço IP principal da interface, deixando você com 9 slots para prefixos de endereços IP and/or secundários. Se esses slots receberem prefixos atribuídos, o nó poderá suportar (9 * 16) 144 endereços IP, enquanto que, se forem atribuídos endereços IP secundários, ele poderá suportar apenas 9 endereços IP. Consulte a documentação sobre endereços IP por interface de rede por tipo de instância e como atribuir prefixos às interfaces de rede para obter mais informações.

Durante a inicialização do nó de trabalho, o VPC Resource Controller atribui um ou mais prefixos à ENI primária para acelerar a inicialização do pod, mantendo um pool aquecido de endereços IP. O número de prefixos a serem mantidos na piscina aquecida pode ser controlado definindo os seguintes parâmetros de configuração no mapa de `amazon-vpc-cni` configuração.

- `warm-prefix-target`, o número de prefixos a serem alocados além da necessidade atual.
- `warm-ip-target`, o número de endereços IP a serem alocados além da necessidade atual.

- `minimum-ip-target`, o número mínimo de endereços IP disponíveis a qualquer momento.
- `warm-ip-target` and/or `minimum-ip-target`se definido, `warm-prefix-target` substituirá.

À medida que mais pods forem programados no nó, prefixos adicionais serão solicitados para o ENI existente. Quando um pod é programado no nó, o VPC Resource Controller primeiro tenta atribuir um IPv4 endereço a partir dos prefixos existentes no nó. Se isso não for possível, um novo IPv4 prefixo será solicitado, desde que a sub-rede tenha a capacidade necessária.

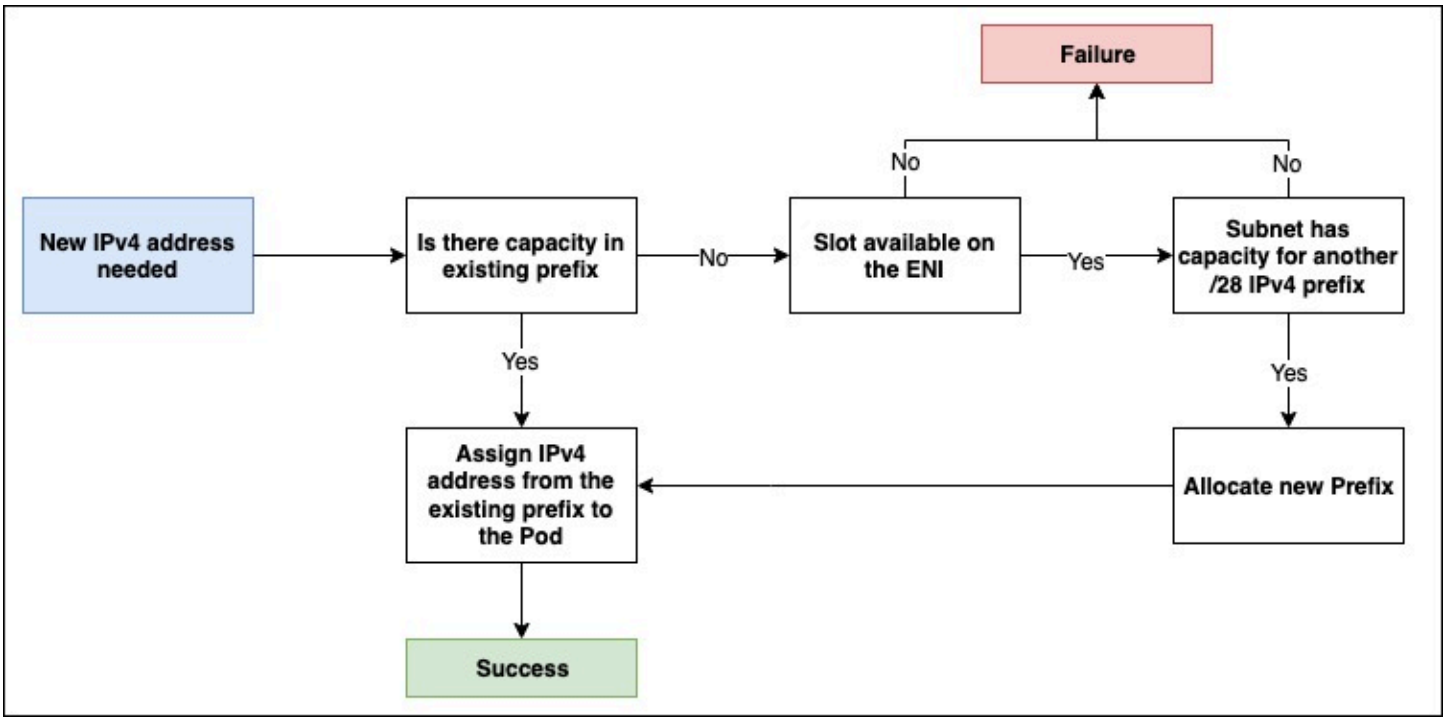


Figura: Fluxo de trabalho durante a atribuição do IPv4 endereço ao pod

Recomendações

Use Delegação de Prefixo quando

Use a delegação de prefixo se você estiver enfrentando problemas de densidade de pods nos nós de trabalho. Para evitar erros, recomendamos examinar as sub-redes em busca de blocos contíguos de endereços para o prefixo /28 antes de migrar para o modo de prefixo. Consulte a seção "Usar reservas de sub-rede para evitar a fragmentação da sub-rede (IPv4)" para obter detalhes da reserva de sub-rede.

Por padrão, os nós `max-pods` no Windows estão definidos como`110`. Para a grande maioria dos tipos de instância, isso deve ser suficiente. Se você quiser aumentar ou diminuir esse limite, adicione o seguinte ao comando bootstrap nos dados do usuário:

```
-KubeletExtraArgs '--max-pods=example-value'
```

Para obter mais detalhes sobre os parâmetros de configuração de bootstrap para nós do Windows, visite a documentação aqui.

Evite a delegação de prefixo quando

Se sua sub-rede estiver muito fragmentada e não tiver endereços IP disponíveis suficientes para criar prefixos /28, evite usar o modo de prefixo. O anexo do prefixo pode falhar se a sub-rede da qual o prefixo é produzido estiver fragmentada (uma sub-rede muito usada com endereços IP secundários dispersos). Esse problema pode ser evitado criando uma nova sub-rede e reservando um prefixo.

Configure parâmetros para delegação de prefixo para conservar endereços IPv4

`warm-prefix-target`,`warm-ip-target`, e `minimum-ip-target` pode ser usado para ajustar o comportamento do pré-escalonamento e do escalonamento dinâmico com prefixos. Por padrão, os seguintes valores são usados:

```
warm-ip-target: "1"
minimum-ip-target: "3"
```

Ao ajustar esses parâmetros de configuração, você pode obter um equilíbrio ideal entre conservar os endereços IP e garantir a diminuição da latência do pod devido à atribuição do endereço IP. Para obter mais informações sobre esses parâmetros de configuração, visite a documentação aqui.

Use reservas de sub-rede para evitar a fragmentação da sub-rede () IPv4

Quando EC2 aloca um IPv4 prefixo /28 para uma ENI, ele precisa ser um bloco contíguo de endereços IP da sua sub-rede. Se a sub-rede da qual o prefixo é gerado estiver fragmentada (uma sub-rede muito usada com endereços IP secundários dispersos), o anexo do prefixo poderá falhar e você verá o seguinte evento de nó:

```
InsufficientCidrBlocks: The specified subnet does not have enough free cidr blocks to
 satisfy the request
```

Para evitar a fragmentação e ter espaço contíguo suficiente para criar prefixos, use reservas CIDR de sub-rede VPC para reservar espaço IP em uma sub-rede para uso exclusivo por prefixos. Depois de criar uma reserva, os endereços IP dos blocos reservados não serão atribuídos a outros recursos. Dessa forma, o VPC Resource Controller poderá obter prefixos disponíveis durante a chamada de atribuição para o nó ENI.

É recomendável criar uma nova sub-rede, reservar espaço para prefixos e habilitar a atribuição de prefixos para nós de trabalho executados nessa sub-rede. Se a nova sub-rede for dedicada somente aos pods em execução no seu cluster EKS com a delegação de prefixo ativada, você poderá pular a etapa de reserva do prefixo.

Substitua todos os nós ao migrar do modo IP secundário para o modo de delegação de prefixo ou vice-versa

É altamente recomendável criar novos grupos de nós para aumentar o número de endereços IP disponíveis, em vez de fazer a substituição contínua dos nós de trabalho existentes.

Ao usar grupos de nós autogerenciados, as etapas da transição seriam:

- Aumente a capacidade do seu cluster para que os novos nós possam acomodar suas cargas de trabalho
- Ativar/desativar o recurso de delegação de prefixo para Windows
- Isole e drene todos os nós existentes para expulsar com segurança todos os seus pods existentes. Para evitar interrupções no serviço, sugerimos implementar Pod Disruption Budgets em seus clusters de produção para cargas de trabalho críticas.
- Depois de confirmar que os pods estão em execução, você pode excluir os nós e grupos de nós antigos. Os pods em novos nós receberão um IPv4 endereço de um prefixo atribuído ao nó ENI.

Ao usar grupos de nós gerenciados, as etapas da transição seriam:

- Ativar/desativar o recurso de delegação de prefixo para Windows
- Atualize o grupo de nós usando as etapas mencionadas aqui. Isso executa etapas semelhantes às anteriores, mas é gerenciado pelo EKS.

> ⚠ Warning
>
> Execute todos os pods em um nó no mesmo modo

Para Windows, recomendamos que você evite executar pods no modo IP secundário e no modo de delegação de prefixo ao mesmo tempo. Essa situação pode surgir quando você migra do modo IP secundário para o modo de delegação de prefixo ou vice-versa com a execução de cargas de trabalho do Windows.

Embora isso não afete seus pods em execução, pode haver inconsistência com relação à capacidade do endereço IP do nó. Por exemplo, considere um nó t3.xlarge que tem 14 slots para endereços secundários. IPv4 Se você estiver executando 10 pods, 10 slots na ENI serão consumidos por endereços IP secundários. Depois de habilitar a delegação de prefixo, a capacidade anunciada para o servidor kube-api seria (14 slots * 16 endereços IP por prefixo) 244, mas a capacidade real naquele momento seria (4 slots restantes * 16 endereços por prefixo) 64. Essa inconsistência entre a quantidade de capacidade anunciada e a quantidade real de capacidade (slots restantes) pode causar problemas se você executar mais pods do que endereços IP disponíveis para atribuição.

Dito isso, você pode usar a estratégia de migração conforme descrito acima para fazer a transição segura dos pods do endereço IP secundário para os endereços obtidos dos prefixos. Ao alternar entre os modos, os Pods continuarão funcionando normalmente e:

- Ao alternar do modo IP secundário para o modo de delegação de prefixo, os endereços IP secundários atribuídos aos pods em execução não serão liberados. Os prefixos serão atribuídos aos slots gratuitos. Quando um pod for encerrado, o IP secundário e o slot que ele estava usando serão liberados.
- Ao alternar do modo de delegação de prefixo para o modo IP secundário, um prefixo será liberado quando todos os itens IPs dentro de seu intervalo não estiverem mais alocados aos pods. Se algum IP do prefixo for atribuído a um pod, esse prefixo será mantido até que os pods sejam encerrados.

Problemas de depuração com delegação de prefixo

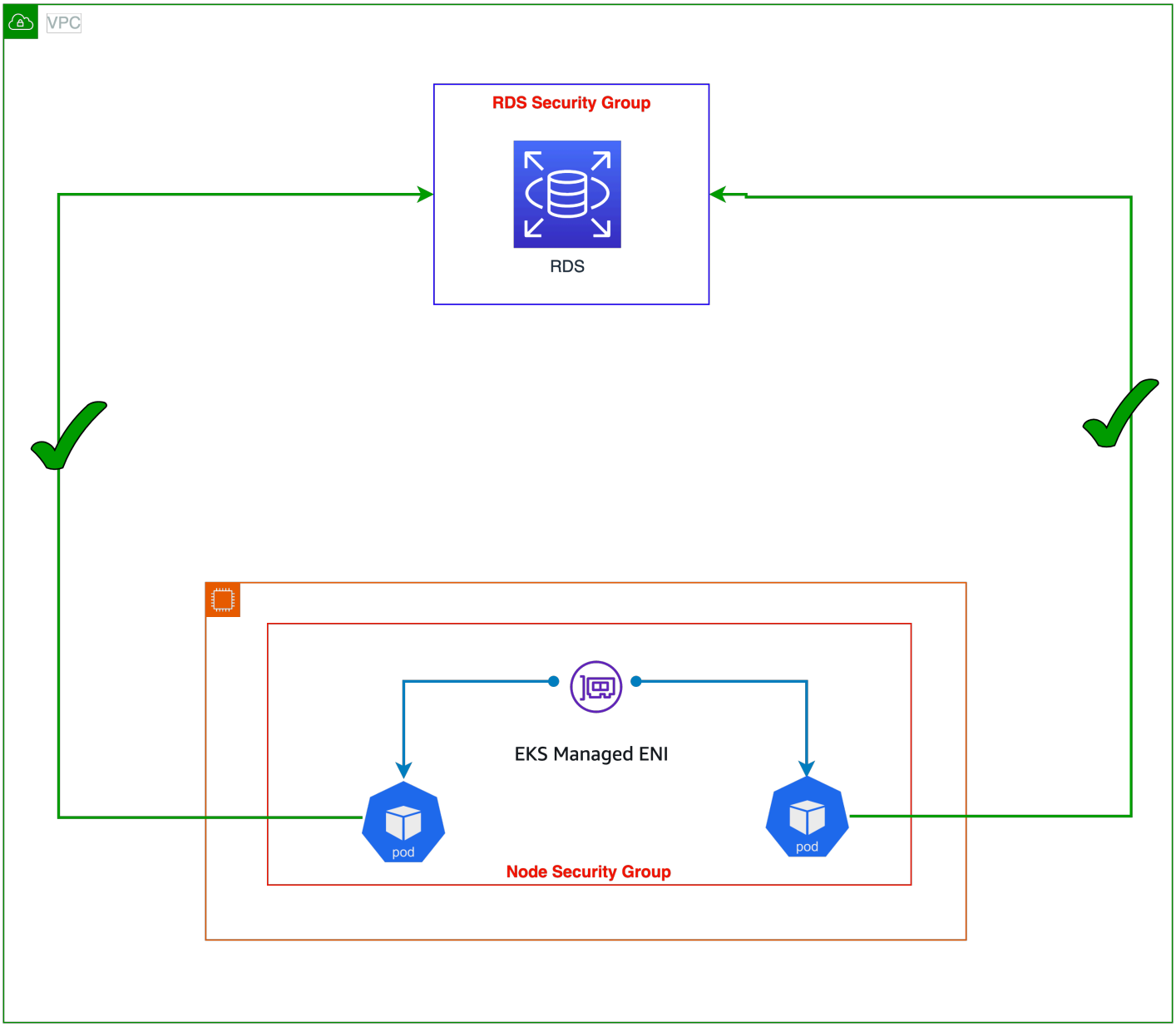
Você pode usar nosso guia de depuração aqui para se aprofundar no problema que você enfrenta com a delegação de prefixos no Windows.

Grupos de segurança por pod

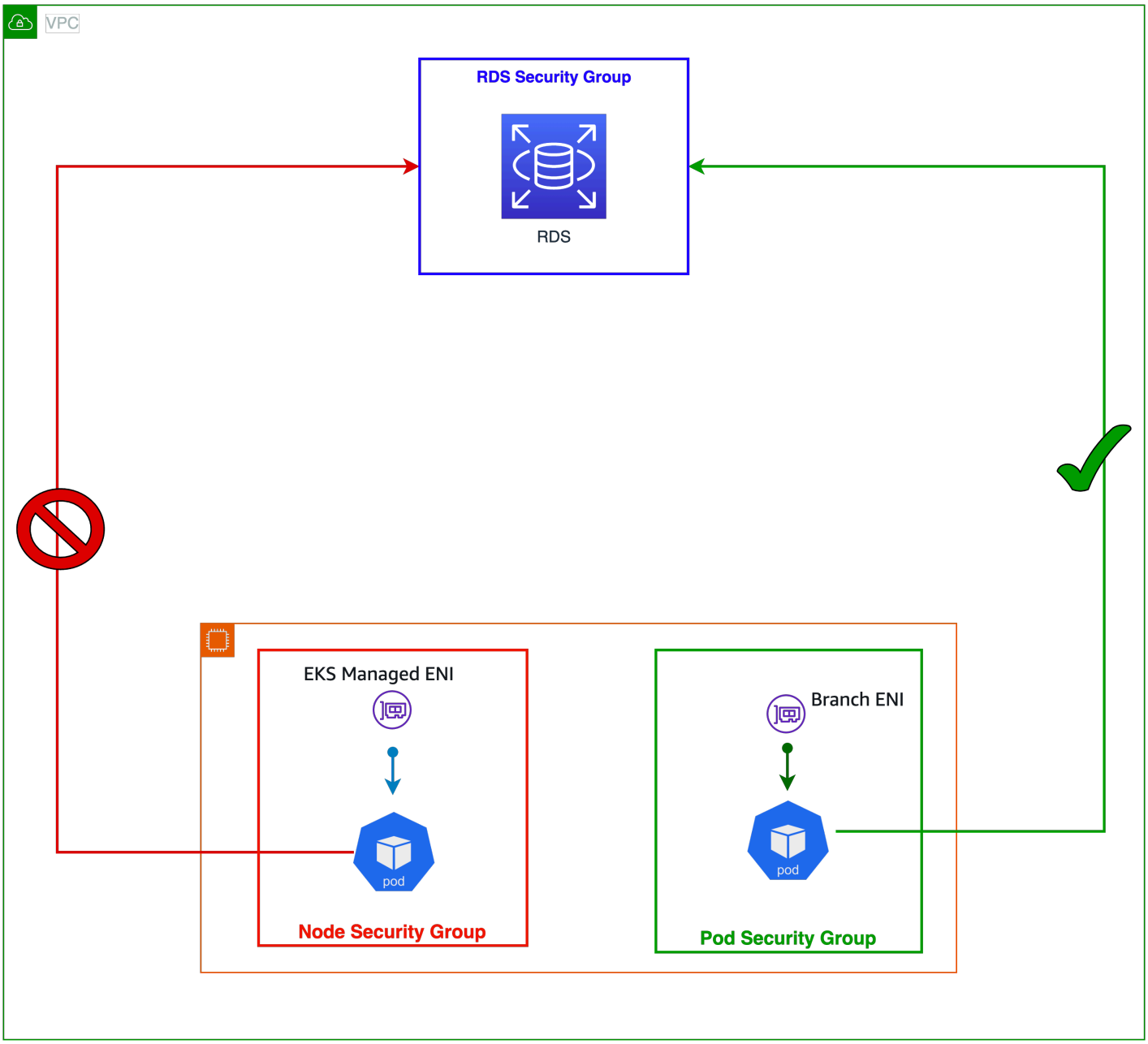
Um grupo de segurança da AWS atua como um firewall virtual para que EC2 as instâncias controlem o tráfego de entrada e saída. Por padrão, a CNI da Amazon VPC usará grupos de segurança associados à ENI primária no nó. Mais especificamente, cada ENI associada à instância terá os

mesmos grupos EC2 de segurança. Assim, cada pod em um nó compartilha os mesmos grupos de segurança do nó em que é executado.

Conforme visto na imagem abaixo, todos os pods de aplicativos que operam nos nós de trabalho terão acesso ao serviço de banco de dados do RDS (considerando que a entrada do RDS permite o grupo de segurança do nó). Os grupos de segurança são muito granulados porque se aplicam a todos os pods em execução em um nó. Os grupos de segurança para pods fornecem segmentação de rede para cargas de trabalho, o que é uma parte essencial de uma boa estratégia de defesa em profundidade.

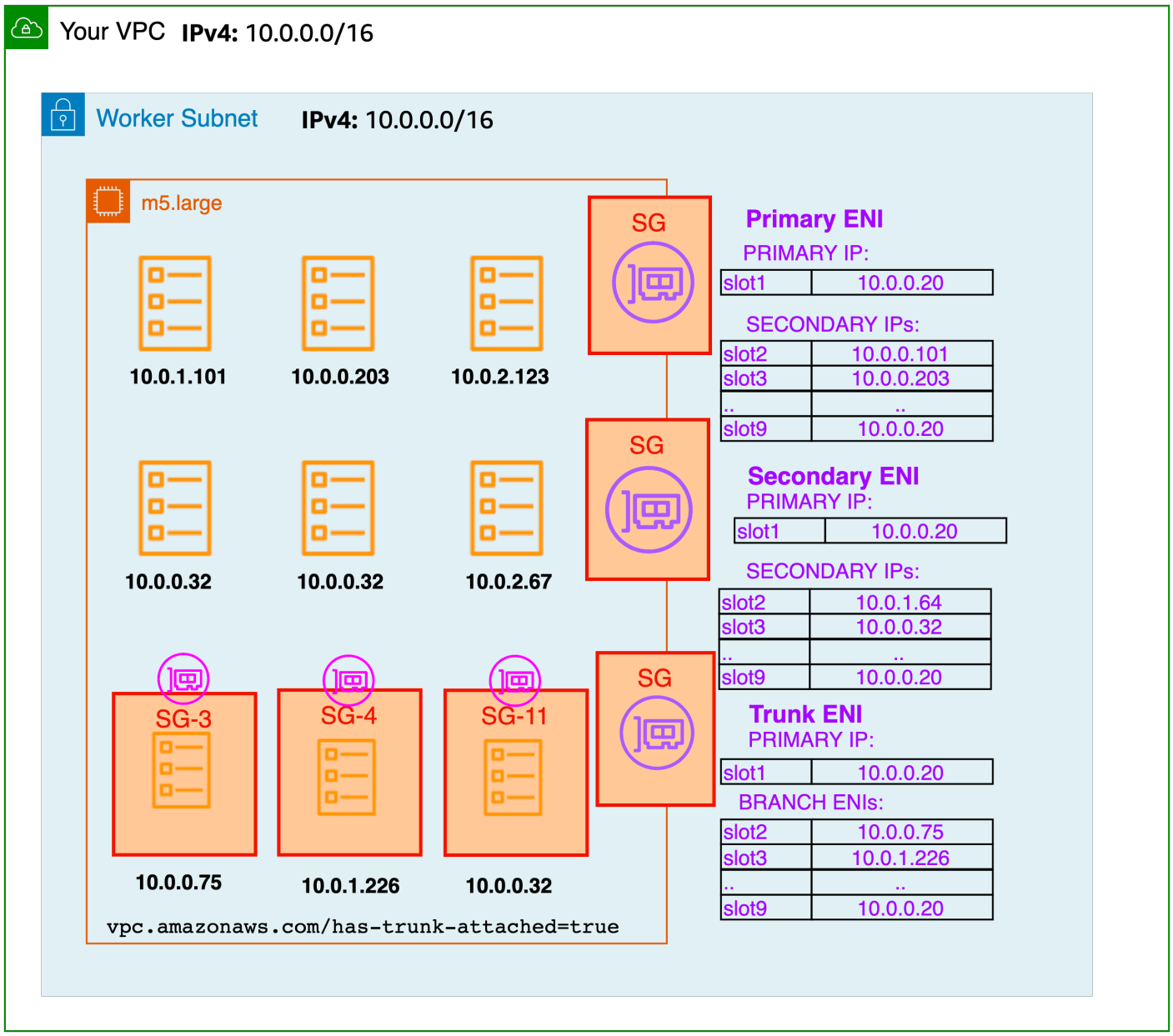


Com grupos de segurança para pods, você pode melhorar a eficiência computacional executando aplicativos com diferentes requisitos de segurança de rede em recursos computacionais compartilhados. Vários tipos de regras de segurança, como Pod-to-Pod serviços da Pod-to-External AWS, podem ser definidos em um único local com grupos de EC2 segurança e aplicados a cargas de trabalho com o Kubernetes nativo. APIs A imagem abaixo mostra grupos de segurança aplicados no nível do pod e como eles simplificam a implantação do aplicativo e a arquitetura do nó. O Pod agora pode acessar o banco de dados do Amazon RDS.



Você pode ativar grupos de segurança para pods configurando `ENABLE_POD_ENI=true` para VPC CNI. Depois de ativado, o VPC Resource Controller executado no plano de controle (gerenciado pelo EKS) cria e anexa uma interface de tronco chamada “`aws-k8 “`s-trunk-eniao nó. A interface de tronco atua como uma interface de rede padrão conectada à instância. Para gerenciar interfaces de tronco, você deve adicionar a política `AmazonEKSVPCResourceController` gerenciada à função de cluster que acompanha seu cluster Amazon EKS.

O controlador também cria interfaces de ramificação chamadas “aws-k8s-branch-eni" e as associa à interface de tronco. Os pods recebem um grupo de segurança usando o recurso SecurityGroupPolicypersonalizado e são associados a uma interface de ramificação. Como os grupos de segurança são especificados com interfaces de rede, agora podemos programar pods que exigem grupos de segurança específicos nessas interfaces de rede adicionais. Consulte a seção do Guia do usuário do EKS sobre grupos de segurança para pods, incluindo os pré-requisitos de implantação.



A capacidade da interface de filial é adicional aos limites de tipo de instância existentes para endereços IP secundários. Os pods que usam grupos de segurança não são contabilizados na fórmula max-pods e, ao usar o grupo de segurança para pods, é necessário considerar aumentar o valor de max-pods ou aceitar a execução de menos pods do que o nó realmente suporta.

Um m5.large pode ter até 9 interfaces de rede de filiais e até 27 endereços IP secundários atribuídos às suas interfaces de rede padrão. Conforme mostrado no exemplo abaixo, o máximo de pods padrão para um m5.large é 29, e o EKS conta os pods que usam grupos de segurança em relação

ao máximo de pods. Consulte o guia do usuário do EKS para obter instruções sobre como alterar os max-pods para nós.

Quando grupos de segurança para pods são usados em combinação com redes personalizadas, o grupo de segurança definido nos grupos de segurança para pods é usado em vez do grupo de segurança especificado no. ENIConfig Como resultado, quando a rede personalizada estiver ativada, avalie cuidadosamente a ordem dos grupos de segurança ao usar grupos de segurança por pod.

Recomendações

Desative o TCP Early Demux para Liveness Probe

Se você estiver usando sondas de vivacidade ou prontidão, também precisará desativar o TCP early demux, para que o kubelet possa se conectar aos pods nas interfaces de rede da filial via TCP. Isso só é necessário no modo estrito. Para fazer isso, execute o seguinte comando:

```
kubectl edit daemonset aws-node -n kube-system
```

Na `initContainer` seção, altere o valor de `DISABLE_TCP_EARLY_DEMUX` para `true`.

Use o Security Group For Pods para aproveitar o investimento existente em configuração da AWS.

Os grupos de segurança facilitam a restrição do acesso à rede aos recursos da VPC, como bancos de dados ou instâncias do RDS. EC2 Uma vantagem clara dos grupos de segurança por pod é a oportunidade de reutilizar os recursos existentes do grupo de segurança da AWS. Se você estiver usando grupos de segurança como um firewall de rede para limitar o acesso aos seus serviços da AWS, propomos aplicar grupos de segurança aos pods usando o branch ENIs. Considere usar grupos de segurança para pods se você estiver transferindo aplicativos de EC2 instâncias para o EKS e limite o acesso a outros serviços da AWS com grupos de segurança.

Configurar o modo de imposição do grupo de segurança do Pod

A versão 1.11 do plug-in CNI da Amazon VPC adicionou uma nova configuração chamada `POD_SECURITY_GROUP_ENFORCING_MODE` (“modo de aplicação”). O modo de imposição controla quais grupos de segurança se aplicam ao pod e se o NAT de origem está habilitado. Você pode especificar o modo de imposição como estrito ou padrão. Rigoroso é o padrão, refletindo o comportamento anterior da VPC CNI `ENABLE_POD_ENI` com definido como. `true`

No Modo Estrito, somente os grupos de segurança ENI da ramificação são aplicados. O NAT de origem também está desativado.

No Modo Padrão, os grupos de segurança associados à ENI primária e à ENI de ramificação (associada ao pod) são aplicados. O tráfego de rede deve estar em conformidade com os dois grupos de segurança.

> **Warning**
>
> Qualquer alteração no modo afetará apenas os pods recém-lançados. Os pods existentes usarão o modo que foi configurado quando o pod foi criado. Os clientes precisarão reciclar os pods existentes com grupos de segurança se quiserem mudar o comportamento do tráfego.

Modo de aplicação: use o modo estrito para isolar o tráfego de pods e nós:

Por padrão, os grupos de segurança para pods estão definidos como “modo estrito”. Use essa configuração se precisar separar completamente o tráfego do pod do resto do tráfego do nó. No modo estrito, o NAT de origem é desativado para que os grupos de segurança de saída ENI da ramificação possam ser usados.

> **Warning**
>
> Quando o modo estrito estiver ativado, todo o tráfego de saída de um pod sairá do nó e entrará na rede VPC. O tráfego entre pods no mesmo nó passará pela VPC. Isso aumenta o tráfego da VPC e limita os recursos baseados em nós. O não NodeLocal DNSCache é compatível com o modo estrito.

Modo obrigatório: use o modo Padrão nas seguintes situações

IP de origem do cliente visível para os contêineres no pod

Se você precisar manter o IP de origem do cliente visível para os contêineres no pod, considere `POD_SECURITY_GROUP_ENFORCING_MODE` configurar como`standard`. Os serviços do Kubernetes suportam externalTrafficPolicy =local para suportar a preservação do IP de origem do cliente (tipo padrão de cluster). Agora você pode executar serviços do Kubernetes do tipo NodePort e LoadBalancer usando destinos de instância com um externalTrafficPolicy definido como Local no

modo padrão. `Local`preserva o IP de origem do cliente e evita um segundo salto para LoadBalancer e NodePort digite os Serviços.

Implantando NodeLocal DNSCache

Ao usar grupos de segurança para pods, configure o modo padrão para suportar pods que usam. NodeLocal DNSCache NodeLocal DNSCache melhora o desempenho do Cluster DNS executando um agente de cache de DNS nos nós do cluster como a. DaemonSet Isso ajudará os pods que têm os mais altos requisitos de DNS QPS a consultar o Kube-DNS/CoreDNS local com um cache local, o que melhorará a latência.

NodeLocal DNSCache não é suportado no modo estrito, pois todo o tráfego de rede, até mesmo para o nó, entra na VPC.

Apoiando a política de rede Kubernetes

Recomendamos usar o modo de imposição padrão ao usar a política de rede com pods que tenham grupos de segurança associados.

É altamente recomendável utilizar grupos de segurança para pods para limitar o acesso em nível de rede aos serviços da AWS que não fazem parte de um cluster. Considere políticas de rede para restringir o tráfego de rede entre pods dentro de um cluster, geralmente conhecido como East/West tráfego.

Identifique incompatibilidades com grupos de segurança por pod

Instâncias baseadas em Windows e não nitro não oferecem suporte a grupos de segurança para pods. Para utilizar grupos de segurança com pods, as instâncias precisam ser marcadas com isTrunkingEnabled. Use políticas de rede para gerenciar o acesso entre pods em vez de grupos de segurança se seus pods não dependerem de nenhum serviço da AWS dentro ou fora da sua VPC.

Use grupos de segurança por pod para controlar com eficiência o tráfego para os serviços da AWS

Se um aplicativo em execução no cluster EKS precisar se comunicar com outro recurso dentro da VPC, por exemplo, um banco de dados do RDS, considere usar SGs para pods. Embora existam mecanismos de política que permitem especificar um CIDR ou um nome DNS, eles são uma opção menos ideal para se comunicar com serviços da AWS que têm endpoints que residem em uma VPC.

Por outro lado, as políticas de rede do Kubernetes fornecem um mecanismo para controlar o tráfego de entrada e saída dentro e fora do cluster. As políticas de rede do Kubernetes devem ser

consideradas se seu aplicativo tiver dependências limitadas de outros serviços da AWS. Você pode configurar políticas de rede que especifiquem regras de saída com base em intervalos de CIDR para limitar o acesso aos serviços da AWS, em oposição à semântica nativa da AWS, como. SGs Você pode usar as políticas de rede do Kubernetes para controlar o tráfego de rede entre pods (geralmente chamado de East/West tráfego) e entre pods e serviços externos. As políticas de rede do Kubernetes são implementadas nos níveis 3 e 4 do OSI.

O Amazon EKS permite que você use mecanismos de política de rede, como Calico e Cilium. Por padrão, os mecanismos de política de rede não estão instalados. Consulte os respectivos guias de instalação para obter instruções sobre como configurar. Para obter mais informações sobre como usar a política de rede, consulte as melhores práticas de segurança do EKS. O recurso de nomes de host DNS está disponível nas versões corporativas dos mecanismos de política de rede, o que pode ser útil para controlar o tráfego entre o Kubernetes Services/Pods e os recursos executados fora da AWS. Além disso, você pode considerar o suporte de nome de host DNS para serviços da AWS que não oferecem suporte a grupos de segurança por padrão.

Marque um único grupo de segurança para usar o AWS Loadbalancer Controller

Quando vários grupos de segurança são alocados em um pod, o Amazon EKS recomenda marcar um único grupo de segurança como `kubernetes.io/cluster/$name`compartilhado ou próprio. A tag permite que o AWS Loadbalancer Controller atualize as regras dos grupos de segurança para rotear o tráfego para os pods. Se apenas um grupo de segurança for fornecido a um pod, a atribuição de uma tag é opcional. As permissões definidas em um grupo de segurança são aditivas, portanto, marcar um único grupo de segurança é suficiente para que o controlador do balanceador de carga localize e reconcilie as regras. Também ajuda a aderir às cotas padrão definidas pelos grupos de segurança.

Configurar NAT para tráfego de saída

O NAT de origem está desativado para tráfego de saída de pods aos quais são atribuídos grupos de segurança. Para pods que usam grupos de segurança que exigem acesso aos nós de trabalho de inicialização da Internet em sub-redes privadas configuradas com um gateway ou instância NAT e habilite o SNAT externo na CNI.

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

Implante pods com grupos de segurança em sub-redes privadas

Os pods atribuídos a grupos de segurança devem ser executados em nós implantados em sub-redes privadas. Observe que os pods com grupos de segurança atribuídos implantados em sub-redes públicas não conseguirão acessar a Internet.

Verifique os terminationGracePeriodsegundos no arquivo de especificação do pod

Certifique-se de que `terminationGracePeriodSeconds` seja diferente de zero no arquivo de especificação do seu pod (30 segundos padrão). Isso é essencial para que o Amazon VPC CNI exclua a rede Pod do nó de trabalho. Quando definido como zero, o plug-in CNI não remove a rede Pod do host e a ENI da ramificação não é efetivamente limpa.

Usando grupos de segurança para pods com o Fargate

Os grupos de segurança para pods executados no Fargate funcionam de forma muito semelhante aos pods EC2 executados em nós de trabalho. Por exemplo, você precisa criar o grupo de segurança antes de referenciá-lo no que SecurityGroupPolicy você associa ao seu Fargate Pod. Por padrão, o grupo de segurança do cluster é atribuído a todos os Fargate Pods quando você não atribui explicitamente um a um Fargate Pod. SecurityGroupPolicy Para simplificar, talvez você queira adicionar o grupo de segurança do cluster ao Fagate Pod, SecurityGroupPolicy caso contrário, você precisará adicionar as regras mínimas do grupo de segurança ao seu grupo de segurança. Você pode encontrar o grupo de segurança do cluster usando a 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
```

As regras mínimas do grupo de segurança estão listadas aqui. Essas regras permitem que os Fargate Pods se comuniquem com serviços no cluster, como kube-apiserver, kubelet e CoreDNS. Você também precisa adicionar regras para permitir conexões de entrada e saída de e para seu Fargate Pod. Isso permitirá que seu pod se comunique com outros pods ou recursos em sua VPC. Além disso, você precisa incluir regras para que o Fargate extraia imagens de contêineres do Amazon ECR ou de outros registros de contêineres, como. DockerHub Para obter mais informações, consulte Intervalos de endereços IP da AWS na Referência geral da AWS.

Você pode usar os comandos abaixo para encontrar os grupos de segurança aplicados a um Fargate Pod.

```
kubectl get pod FARGATE_POD -o jsonpath='{.metadata.annotations.fargate\.amazonaws
\.com/pod-sg}{"\n"}'
```

Anote o comando eNIID acima.

```
aws ec2 describe-network-interfaces --network-interface-ids ENI_ID --query
 'NetworkInterfaces[*].Groups[*]'
```

Os pods Fargate existentes devem ser excluídos e recriados para que novos grupos de segurança sejam aplicados. Por exemplo, o comando a seguir inicia a implantação do aplicativo de exemplo. Para atualizar pods específicos, você pode alterar o namespace e o nome da implantação no comando abaixo.

```
kubectl rollout restart -n example-ns deployment example-pod
```

Balanceamento de carga

Os balanceadores de carga recebem tráfego de entrada e o distribuem entre os destinos do aplicativo pretendido hospedado em um cluster EKS. Isso melhora a resiliência do aplicativo. Quando implantado em um cluster EKS, o controlador do AWS Load Balancer criará e gerenciará os AWS Elastic Load Balancers para esse cluster. Quando um serviço Kubernetes do tipo LoadBalancer é criado, o AWS Load Balancer Controller cria um Network Load Balancer (NLB) que balanceia a carga do tráfego recebido na camada 4 do modelo OSI. Já quando um objeto Kubernetes Ingress é criado,

o AWS Load Balancer Controller cria um Application Load Balancer (ALB) que equilibra a carga do tráfego na camada 7 do modelo OSI.

Escolhendo o tipo de Load Balancer

O portfólio do AWS Elastic Load Balancing (ELB) suporta os seguintes balanceadores de carga: Application Load Balancers (ALB), Network Load Balancers (NLB), Gateway Load Balancers (GWLB) e Classic Load Balancers (CLB). Esta seção de melhores práticas se concentrará no ALB e no NLB, que são os dois mais relevantes para os clusters EKS.

A principal consideração na escolha do tipo de balanceador de carga são os requisitos de carga de trabalho.

Para obter informações mais detalhadas e como referência para todos os balanceadores de carga da AWS, consulte Comparações de produtos

Escolha o Application Load Balancer (ALB) se sua carga de trabalho for HTTP/HTTPS

Se uma carga de trabalho exigir balanceamento de carga na camada 7 do modelo OSI, o AWS Load Balancer Controller pode ser usado para provisionar um ALB; abordaremos o provisionamento na seção a seguir. O ALB é controlado e configurado pelo recurso Ingress mencionado anteriormente e encaminha o tráfego HTTP ou HTTPS para diferentes pods dentro do cluster. O ALB oferece aos clientes a flexibilidade de alterar o algoritmo de roteamento de tráfego do aplicativo; o algoritmo de roteamento padrão é round robin, com o algoritmo de roteamento de solicitações menos pendentes também uma alternativa.

Escolha o Network Load Balancer (NLB) se sua carga de trabalho for TCP ou se sua carga de trabalho exigir a preservação do IP de origem dos clientes

Um Network Load Balancer funciona na quarta camada (Transporte) do modelo de Interconexão de Sistemas Abertos (OSI). É adequado para cargas de trabalho baseadas em TCP e UDP. O Network Load Balancer também preserva, por padrão, o IP de origem do endereço dos clientes ao apresentar o tráfego ao pod.

Escolha o Network Load Balancer (NLB) se sua carga de trabalho não puder utilizar o DNS

Outro motivo importante para usar o NLB é se seus clientes não podem utilizar o DNS. Nesse caso, o NLB pode ser mais adequado à sua carga de trabalho, pois os do Network Load IPs Balancer são

estáticos. Embora seja recomendável que os clientes usem o DNS ao resolver nomes de domínio em endereços IP ao se conectar a balanceadores de carga, se o aplicativo de um cliente não oferecer suporte à resolução de DNS e aceitar apenas códigos embutidos, um NLB é mais adequado IPs , pois IPs eles são estáticos e permanecem os mesmos durante toda a vida útil do NLB.

Provisionamento de balanceadores de carga

Depois de determinar o Load Balancer mais adequado para suas cargas de trabalho, os clientes têm várias opções para provisionar um balanceador de carga.

Provisione balanceadores de carga implantando o AWS Load Balancer Controller

Há dois métodos principais de provisionamento de balanceadores de carga em um cluster EKS.

- Aproveitando o controlador de serviços no provedor de nuvem da AWS (antigo)
- Aproveitando o AWS Load Balancer Controller (recomendado)

Por padrão, o Kubernetes Service Controller, também conhecido como controlador em árvore, reconcilia o recurso do tipo Kubernetes Service. LoadBalancer Esse controlador é incorporado ao componente AWS Cloud Provider, que funciona como o Kubernetes Cloud Controller Manager.

A configuração do Elastic Load Balancer provisionado é controlada por anotações que devem ser adicionadas ao manifesto do Kubernetes Service. As anotações usadas pelo Service Controller e pelo AWS Load Balancer Controller são diferentes.

O Service Controller é antigo e atualmente só está recebendo correções de erros críticos. Quando você cria um serviço Kubernetes do tipo LoadBalancer, o controlador de serviço cria um AWS CLB por padrão, mas também pode criar o AWS NLB se você usar a anotação correta. É importante notar que o Service Controller não oferece suporte aos recursos do Kubernetes Ingress e também não oferece suporte. IPv6

Recomendamos usar o AWS Load Balancer Controller em seus clusters EKS para reconciliar os recursos do Kubernetes Service e do Ingress. Você deve usar as anotações corretas em seu serviço Kubernetes ou manifesto do Ingress para que o AWS Load Balancer Controller seja o proprietário do processo de reconciliação. (em vez de Service Controller)

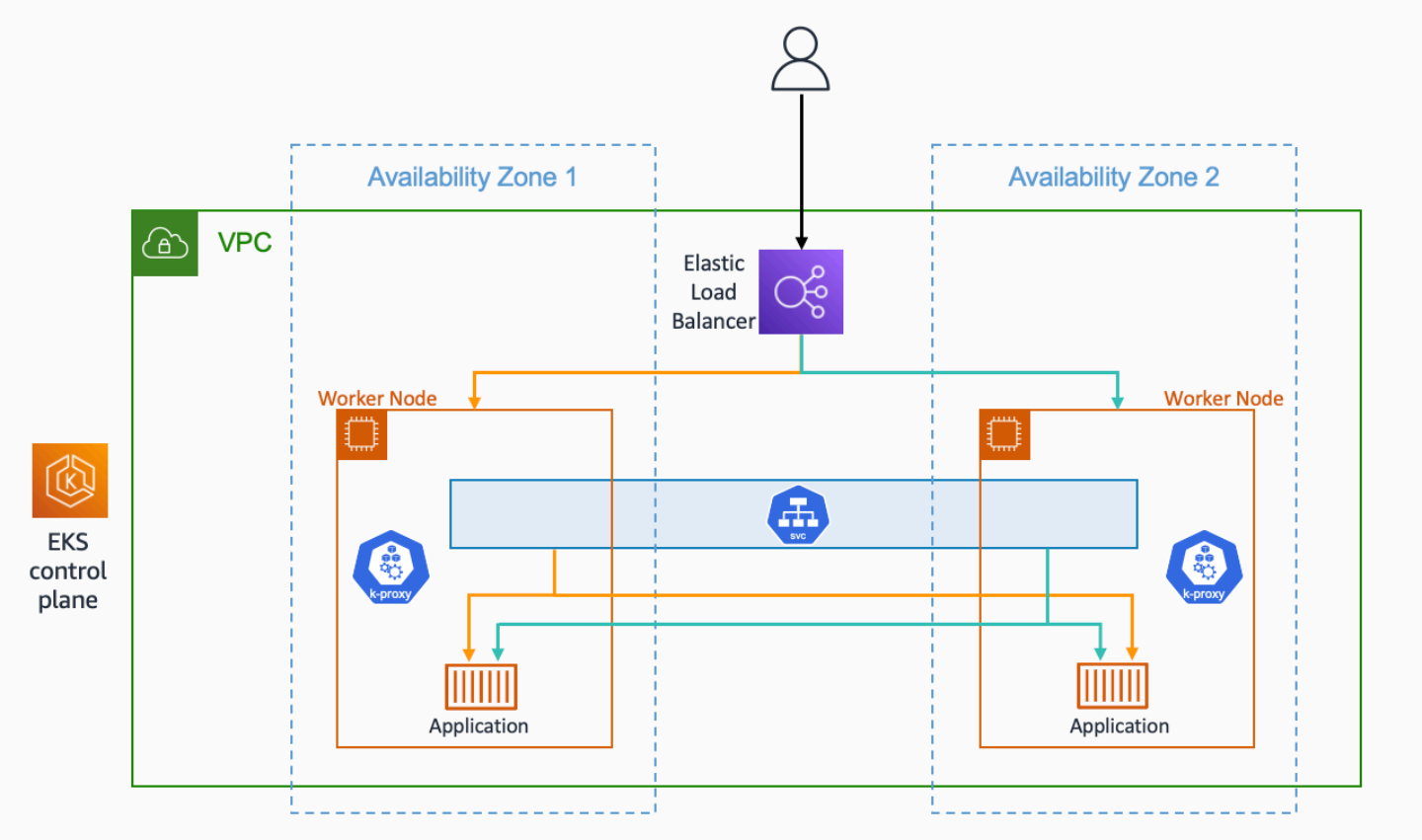
Se você estiver utilizando o EKS Auto Mode, o AWS Load Balancer Controller é fornecido automaticamente; nenhuma instalação é necessária.

Escolhendo o tipo de destino do Load Balancer

Registre pods como destinos usando o tipo de destino IP

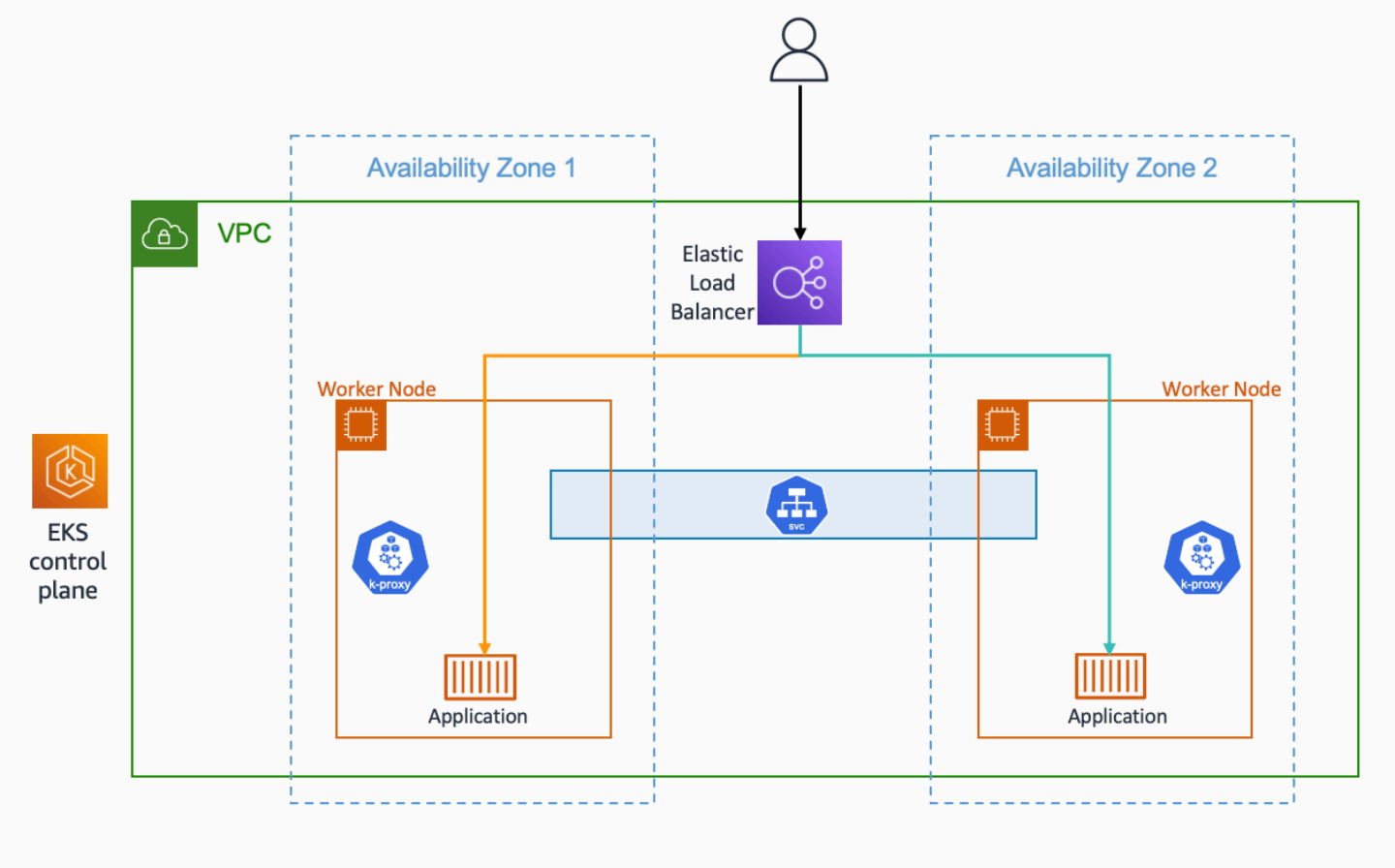
Um AWS Elastic Load Balancer: Network & Application envia o tráfego recebido para alvos registrados em um grupo-alvo. Para um cluster EKS, há dois tipos de destinos que você pode registrar no grupo de destino: Instância e IP, que tipo de destino é usado, têm implicações sobre o que é registrado e como o tráfego é roteado do Load Balancer para o pod. Por padrão, o controlador do AWS Load Balancer registrará destinos usando o tipo “Instância” e esse destino será o IP do Worker Node. As implicações NodePort disso incluem:

- O tráfego do Load Balancer será encaminhado para o Worker Node no NodePort, processado pelas regras do iptables (configuradas pelo kube-proxy em execução no nó) e encaminhado para o Serviço em seu ClusteriP (ainda no nó). Finalmente, o Serviço seleciona aleatoriamente um pod registrado nele e encaminha o tráfego para ele. Esse fluxo envolve vários saltos e pode ocorrer latência extra, especialmente porque o Serviço às vezes seleciona um pod em execução em outro nó de trabalho que também pode estar em outro AZ.
- Como o Load Balancer registra o Worker Node como destino, isso significa que a verificação de integridade enviada ao destino não será recebida diretamente pelo pod, mas pelo Worker Node nele, NodePort e o tráfego da verificação de integridade seguirá o mesmo caminho descrito acima.
- O monitoramento e a solução de problemas são mais complexos, pois o tráfego encaminhado pelo Load Balancer não é enviado diretamente aos pods e você precisaria correlacionar cuidadosamente o pacote recebido no Worker Node com o Service ClusterIP e, eventualmente, com o pod para ter visibilidade end-to-end total do caminho do pacote para a solução de problemas adequada.



Por outro lado, se você configurar o tipo de destino como “IP”, conforme recomendamos, a implicação será a seguinte:

- O tráfego do Load Balancer será encaminhado diretamente para o pod. Isso simplifica o caminho da rede, pois ignora os saltos extras anteriores dos Worker Nodes e do Service Cluster IP, reduz a latência que, de outra forma, teria ocorrido se o serviço encaminhasse o tráfego para um pod em outro AZ e, por fim, remove o processamento indireto das regras de iptables nos Worker Nodes.
- A verificação de integridade do Load Balancer é recebida e respondida diretamente pelo pod. Isso significa que o status alvo “saudável” ou “não íntegro” é uma representação direta do status de saúde do pod.
- O monitoramento e a solução de problemas são mais fáceis e qualquer ferramenta usada para capturar o endereço IP do pacote revelará diretamente o tráfego bidirecional entre o Load Balancer e o pod nos campos de origem e destino.



Para criar um AWS Elastic Load Balancing que usa destinos IP, você adiciona:

- `alb.ingress.kubernetes.io/target-type: ip`anotação no manifesto do seu Ingress ao configurar seu Kubernetes Ingress (Application Load Balancer)
- `service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip`anotação no manifesto do seu serviço ao configurar seu serviço Kubernetes do tipo (Network Load LoadBalancer Balancer).

Configurando as verificações de integridade do Load Balancer

Embora o Kubernetes forneça seus próprios mecanismos de verificação de integridade (detalhados na próxima seção), recomendamos implementar as verificações de integridade do ELB como uma proteção complementar que funciona fora do plano de controle do Kubernetes. Essa camada independente continua monitorando sua aplicação mesmo durante:

- Interrupções no plano de controle do Kubernetes
- Atrasos na execução da sonda

- Partições de rede entre kubelet e pod

Para cargas de trabalho críticas que exigem disponibilidade máxima e recuperação acelerada durante os cenários mencionados acima, as verificações de integridade do ELB fornecem uma rede de segurança essencial que funciona ao lado, e não no lugar, dos mecanismos nativos do Kubernetes.

Para configurar e ajustar as verificações de saúde em seu ELB, você deve usar anotações no serviço Kubernetes ou no manifesto do Ingress que seriam reconciliadas pelo Service Controller ou pelo AWS Load Balancer Controller.

Disponibilidade e ciclo de vida do pod

Durante uma atualização de aplicativo, você deve garantir que seu aplicativo esteja sempre disponível para processar solicitações, para que os usuários não tenham nenhum tempo de inatividade. Um desafio comum nesse cenário é sincronizar o status de disponibilidade de suas cargas de trabalho entre a camada Kubernetes e a infraestrutura, por exemplo, balanceadores de carga externos. As próximas seções destacam as melhores práticas para lidar com esses cenários.

> **Note**
>
> As explicações abaixo são baseadas na EndpointSlicessubstituição recomendada dos endpoints no Kubernetes. As diferenças entre os dois são insignificantes no contexto dos cenários abordados abaixo. Por padrão, o AWS Load Balancer Controller consome endpoints. Você pode habilitá-los EndpointSlices habilitando-os no enable-endpoint-sliceflagcontrolador.

Use verificações de saúde

Por padrão, o Kubernetes executa a verificação de integridade do processo, na qual o processo kubelet no nó verifica se o processo principal do contêiner está em execução. Caso contrário, por padrão, ele reinicia esse contêiner. No entanto, você também pode configurar sondas do Kubernetes para identificar quando um processo de contêiner está em execução, mas em um estado de impasse, ou se um aplicativo foi iniciado com sucesso ou não. As sondas podem ser baseadas nos mecanismos exec, grpc, HttpGet e TcpSocket. Com base no tipo e no resultado da sonda, o contêiner pode ser reiniciado.

Consulte a seção Criação do pod no Apêndice abaixo para rever a sequência de eventos no processo de criação do pod.

Use sondas de prontidão

Por padrão, quando todos os contêineres em um pod estão em execução, a condição do pod é considerada “Pronta”. No entanto, o aplicativo ainda pode não conseguir processar as solicitações do cliente. Por exemplo, o aplicativo pode precisar extrair alguns dados ou configurações de um recurso externo para poder processar solicitações. Nesse estado, você não gostaria de encerrar o aplicativo nem encaminhar nenhuma solicitação para ele. A sonda de prontidão permite garantir que o pod não seja considerado “pronto”, o que significa que ele não será adicionado ao EndpointSlice objeto até que o resultado da sondagem seja obtido. `success` Por outro lado, se a sonda falhar mais abaixo na linha, o Pod será removido do EndpointSlice objeto. Você pode configurar um teste de prontidão no manifesto do Pod para cada contêiner. `kubelet`O processo em cada nó executa a sondagem de prontidão nos contêineres desse nó.

Utilize portas de prontidão do Pod

Um aspecto da sonda de prontidão é o fato de que não há nenhum mecanismo externo de feedback/influência nela. O processo kubelet no nó executa a sonda e define o estado da sonda. Isso não tem nenhum impacto nas solicitações entre os próprios microsserviços na camada Kubernetes (tráfego leste-oeste), pois o EndpointSlice Controller mantém a lista de endpoints (pods) sempre atualizada. Por que e quando você precisaria de um mecanismo externo então?

Quando você expõe seus aplicativos usando o tipo de Load Balancer ou Kubernetes Ingress (para tráfego norte-sul), a lista de Pod do respectivo Kubernetes Service deve ser propagada IPs para o balanceador de carga da infraestrutura externa para que o balanceador de carga também tenha uma lista de destinos atualizada. O AWS Load Balancer Controller preenche essa lacuna aqui. Quando você usa o AWS Load Balancer Controller e aproveita`target group: IP`, assim como o `kube-proxy` AWS Load Balancer Controller, também recebe uma atualização (`watch`via) e, em seguida, ele se comunica com a API ELB para configurar e começar a registrar o IP do pod como destino no ELB.

Quando você executa uma atualização contínua de uma implantação, novos pods são criados e, assim que a condição de um novo pod estiver “pronta”, um old/existing pod será encerrado. Durante esse processo, o EndpointSlice objeto Kubernetes é atualizado mais rápido do que o tempo que o ELB leva para registrar os novos pods como destinos, consulte registro de destinos. Por um breve período, você pode ter uma incompatibilidade de estado entre a camada do Kubernetes e a camada

de infraestrutura, na qual as solicitações do cliente podem ser descartadas. Durante esse período, na camada do Kubernetes, novos pods estariam prontos para processar solicitações, mas, do ponto de vista do ELB, não estão.

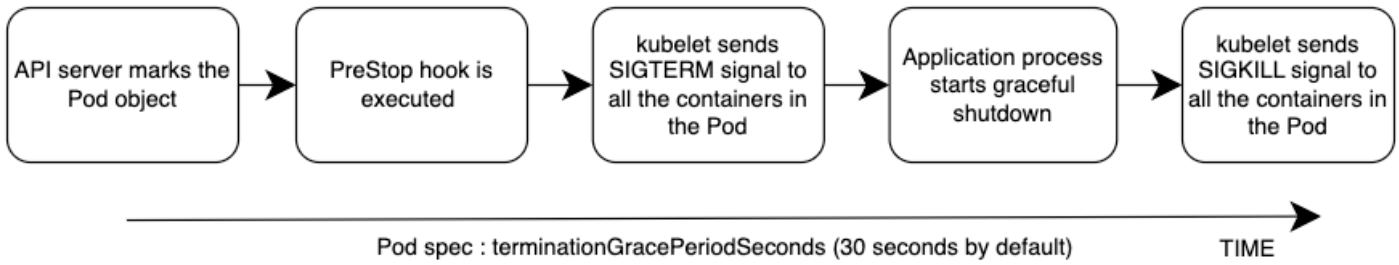
O Pod Readiness Gates permite que você defina requisitos adicionais que devem ser atendidos antes que a condição do Pod seja considerada “Pronta”. No caso do AWS ELB, o AWS Load Balancer Controller monitora o status do destino (o pod) no AWS ELB e, quando o registro do alvo é concluído e seu status fica “Saudável”, o controlador atualiza a condição do pod para “Pronto”. Com essa abordagem, você influencia a condição do pod com base no estado da rede externa, que é o status de destino no AWS ELB. O Pod Readiness Gates é crucial em cenários de atualização contínua, pois permite evitar que a atualização contínua de uma implantação encerre pods antigos até que o status de destino dos pods recém-criados se torne “Saudável” no AWS ELB.

Encerre os aplicativos normalmente

Seu aplicativo deve responder a um sinal SIGTERM iniciando seu desligamento normal para que os clientes não tenham nenhum tempo de inatividade. Isso significa que seu aplicativo deve executar procedimentos de limpeza, como salvar dados, fechar descritores de arquivos, fechar conexões de banco de dados, concluir solicitações em andamento normalmente e sair em tempo hábil para atender à solicitação de encerramento do pod. Você deve definir o período de carência como longo o suficiente para que a limpeza possa ser concluída. Para aprender como responder ao sinal SIGTERM, você pode consultar os recursos da respectiva linguagem de programação que você usa para seu aplicativo.

Se seu aplicativo não conseguir desligar normalmente após o recebimento de um sinal SIGTERM ou se ele ignorar/não receber o sinal, você poderá usar o PreStop hook para iniciar um desligamento normal do aplicativo. O gancho Prestop é executado imediatamente antes do envio do sinal SIGTERM e pode realizar operações arbitrárias sem precisar implementar essas operações no próprio código do aplicativo.

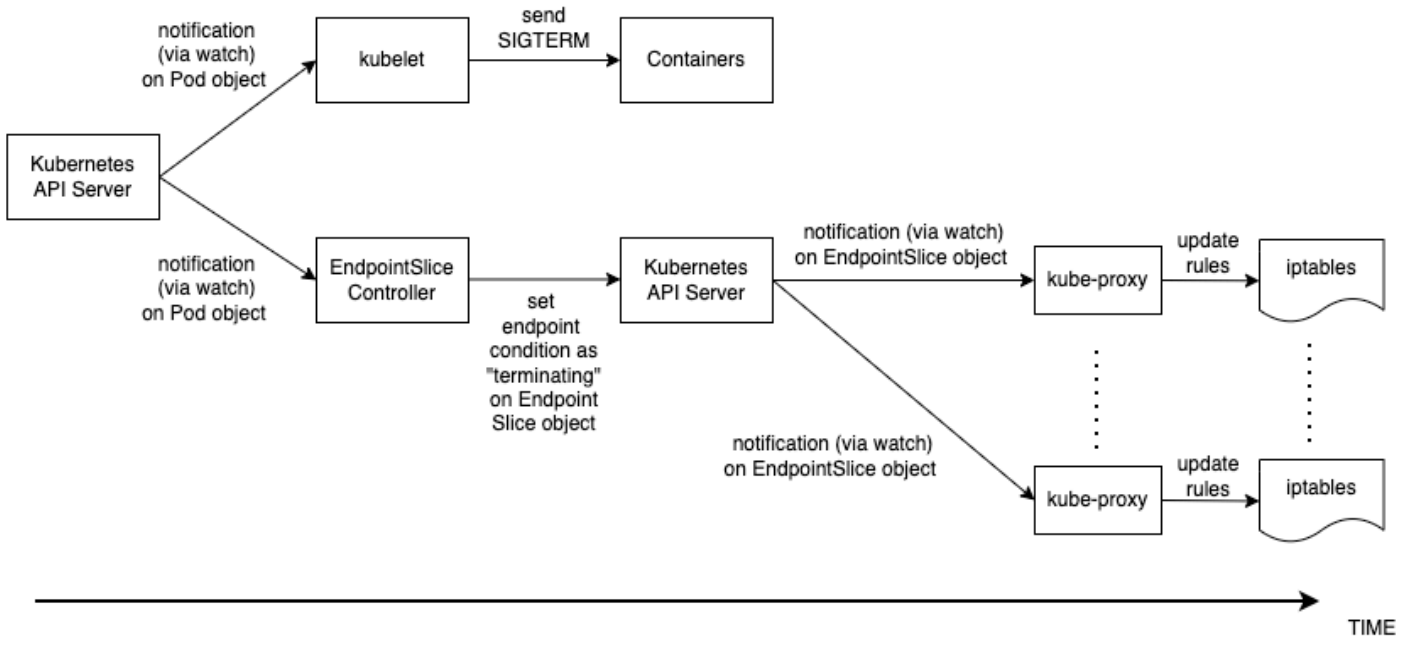
A sequência geral de eventos é mostrada no diagrama abaixo. Nota: independentemente do resultado de um procedimento de desligamento normal do aplicativo ou do resultado do PreStop gancho, os contêineres do aplicativo acabam sendo encerrados no final do período de carência via SIGKILL.



Consulte a seção Exclusão do pod na seção Apêndice abaixo para rever a sequência de eventos no processo de exclusão do pod.

Lide com elegância com as solicitações do cliente

A sequência de eventos na exclusão do pod é diferente da criação do pod. Quando um pod é criado, `kubelet` atualiza o IP do pod na API Kubernetes e somente então o EndpointSlice objeto é atualizado. Por outro lado, quando um pod está sendo encerrado, a API Kubernetes notifica o kubelet e o controlador ao mesmo tempo. EndpointSlice Inspecione cuidadosamente o diagrama a seguir, que mostra a sequência de eventos.



A forma como o estado se propaga desde o servidor da API até as regras de iptables nos nós explicados acima cria uma condição de corrida interessante. Como há uma grande chance de o contêiner receber o sinal SIGKILL muito antes que o kube-proxy em cada nó atualize as regras locais de iptables. Nesse caso, dois cenários que vale a pena mencionar são:

- Se seu aplicativo cancelar imediatamente e sem rodeios as solicitações e conexões em voo após o recebimento do SIGTERM, isso significa que os clientes veriam 50 vezes mais erros em todo o lugar.
- Mesmo que seu aplicativo garanta que todas as solicitações e conexões em andamento sejam processadas completamente após o recebimento do SIGTERM, durante o período de carência, as novas solicitações do cliente ainda serão enviadas ao contêiner do aplicativo porque as regras do iptables ainda podem não ter sido atualizadas. Até que o procedimento de limpeza feche o soquete do servidor no contêiner, essas novas solicitações resultarão em novas conexões. Quando o período de carência termina, essas conexões, que são estabelecidas após o SIGTERM, nesse momento são interrompidas incondicionalmente, pois o SIGKILL é enviado.

Definir o período de carência na especificação do Pod por tempo suficiente pode resolver esse desafio, mas, dependendo do atraso na propagação e do número real de solicitações do cliente, é difícil prever o tempo necessário para o aplicativo fechar as conexões normalmente. Portanto, a abordagem não tão perfeita, mas mais viável, aqui é usar um PreStop gancho para atrasar o sinal SIGTERM até que as regras do iptables sejam atualizadas para garantir que nenhuma nova solicitação do cliente seja enviada ao aplicativo, em vez disso, apenas as conexões existentes continuem. PreStop hook pode ser um manipulador Exec simples, como. `sleep 10`

O comportamento e a recomendação mencionados acima seriam igualmente aplicáveis quando você expõe seus aplicativos usando o tipo de Load Balancer do Kubernetes Service ou o Kubernetes Ingress (para tráfego norte a sul) usando o AWS Load Balancer Controller e alavancagem. `target group: IP` Porque, assim como `kube-proxy` o AWS Load Balancer Controller, também recebe uma atualização (via watch) do EndpointSlice objeto e depois se comunica com a API do ELB para começar a cancelar o registro do IP do pod no ELB. No entanto, dependendo da carga na API Kubernetes ou na API ELB, isso também pode levar tempo e o SIGTERM pode já ter sido enviado ao aplicativo há muito tempo. Quando o ELB começa a cancelar o registro do destino, ele para de enviar solicitações para esse destino, para que o aplicativo não receba nenhuma nova solicitação e o ELB também inicia um atraso de cancelamento de registro, que é de 300 segundos por padrão. Durante o processo de cancelamento do registro, o alvo é `draining` onde basicamente o ELB espera que as requests/existing conexões em voo com esse alvo sejam drenadas. Quando o atraso de cancelamento de registro expira, o alvo não é utilizado e todas as solicitações em voo para esse alvo são retiradas à força.

Use o orçamento de interrupção do Pod

Configure um Pod Disruption Budget (PDB) para seus aplicativos. PDBlimits o número de pods de um aplicativo replicado que estão inativos simultaneamente devido a interrupções voluntárias. Isso garante que um número mínimo ou porcentagem de pods permaneça disponível em uma StatefulSet implantação. Por exemplo, um aplicativo baseado em quórum precisa garantir que o número de réplicas em execução nunca fique abaixo do número necessário para um quórum. Ou um front-end da web pode garantir que o número de réplicas que atendem à carga nunca caia abaixo de uma certa porcentagem do total. O PDB protegerá o aplicativo contra ações como a drenagem de nós ou a implantação de novas versões de implantações. Lembre-se de que os PDBs não protegerão o aplicativo contra interrupções involuntárias, como falha do sistema operacional do nó ou perda de conectividade de rede. Para obter mais informações, consulte a documentação Especificando um orçamento de interrupção para seu aplicativo no Kubernetes.

Referências

- KubeCon Sessão Europa 2019 - Pronto? Um mergulho profundo nos Pod Readiness Gates for Service Health
- Livro - Kubernetes em ação
- Blog da AWS — Como escalar rapidamente seu aplicativo com o ALB no EKS (sem perder tráfego)

Apêndice

Criação de pods

É fundamental entender qual é a sequência de eventos em um cenário em que um pod é implantado e depois recebe e healthy/ready processa as solicitações do cliente. Vamos falar sobre a sequência de eventos.

1. Um pod é criado no plano de controle do Kubernetes (ou seja, por meio de um comando kubectl, atualização de implantação ou ação de escalonamento).
2. `kube-scheduler`atribui o pod a um nó no cluster.
3. O processo kubelet executado no nó atribuído recebe a atualização (via`watch`) e se comunica com o tempo de execução do contêiner para iniciar os contêineres definidos na especificação do Pod.

4. Quando os contêineres começam a ser executados, o kubelet atualiza a condição do Pod como `Ready` no objeto Pod na API Kubernetes.
5. O EndpointSlicecontrolador recebe a atualização da condição do pod (via`watch`) e adiciona o pod IP/Port como um novo endpoint ao EndpointSliceobjeto (lista de pods IPs) do respectivo serviço Kubernetes.
6. O processo kube-proxy em cada nó recebe a atualização (via`watch`) no EndpointSlice objeto e, em seguida, atualiza as regras do iptables em cada nó, com o novo IP/porta do pod.

Exclusão do pod

Assim como na criação do pod, é fundamental entender qual é a sequência de eventos durante a exclusão do pod. Vamos falar sobre a sequência de eventos.

1. Uma solicitação de exclusão do pod é enviada ao servidor da API Kubernetes (ou seja, por um `kubectl` comando, atualização de implantação ou ação de escalonamento).
2. O servidor da API Kubernetes inicia um período de carência, que é de 30 segundos por padrão, definindo o campo deletionTimestamp no objeto Pod. (O período de carência pode ser configurado na especificação do Pod até`terminationGracePeriodSeconds`)
3. O `kubelet` processo em execução no nó recebe a atualização (via watch) no objeto Pod e envia um sinal SIGTERM para o identificador de processo 1 (PID 1) dentro de cada contêiner desse pod. Em seguida, ele assiste `terminationGracePeriodSeconds` o.
4. O EndpointSliceController também recebe a atualização (via`watch`) da Etapa 2 e define a condição do endpoint como “encerramento” no EndpointSliceobjeto (lista de Pod IPs) do respectivo serviço Kubernetes.
5. O processo kube-proxy em cada nó recebe a atualização (via`watch`) no EndpointSlice objeto e as regras do iptables em cada nó são atualizadas pelo kube-proxy para parar de encaminhar as solicitações dos clientes para o pod.
6. Quando o `terminationGracePeriodSeconds` expira, `kubelet` ele envia o sinal SIGKILL para o processo pai de cada contêiner no Pod e o encerra à força.
7. TheEndpointSliceO controlador remove o endpoint do EndpointSliceobjeto.
8. O servidor da API exclui o objeto Pod.

Monitoramento de cargas de trabalho do EKS para problemas de desempenho de rede

Monitorando o tráfego do CoreDNS para problemas de limitação de DNS

Às vezes, a execução de cargas de trabalho intensivas de DNS pode apresentar falhas intermitentes do CoreDNS devido à limitação do DNS, e isso pode afetar os aplicativos nos quais você pode encontrar erros ocasionais. UnknownHostException

O Deployment for CoreDNS tem uma política de antiafinidade que instrui o programador do Kubernetes a executar instâncias do CoreDNS em nós de trabalho separados no cluster, ou seja, ele deve evitar a colocação de réplicas no mesmo nó de trabalho. Isso reduz efetivamente o número de consultas de DNS por interface de rede porque o tráfego de cada réplica é roteado por meio de uma ENI diferente. Se você perceber que as consultas de DNS estão sendo limitadas devido ao limite de 1024 pacotes por segundo, você pode 1) tentar aumentar o número de réplicas de CoreDNS ou 2) implementar. NodeLocal DNSCache Consulte Monitorar métricas do CoreDNS para obter mais informações.

Desafio

- A queda de pacotes ocorre em segundos e pode ser difícil monitorar adequadamente esses padrões para determinar se a limitação do DNS está realmente acontecendo.
- As consultas de DNS são limitadas no nível da interface de elastic network. Portanto, as consultas limitadas não aparecem no registro de consultas.
- Os logs de fluxo não capturam todo o tráfego de IP. Por exemplo: O tráfego gerado por instâncias quando elas entram em contato com o servidor de DNS da Amazon. Se você usa seu próprio servidor DNS, todo o tráfego para esse servidor DNS é registrado

Solução

Uma maneira fácil de identificar os problemas de limitação do DNS nos nós de trabalho é capturando métricas. `linklocal_allowance_exceeded` O linklocal_allowance_exceeded é o número de pacotes descartados porque o PPS do tráfego para os serviços de proxy locais excedeu o máximo da interface de rede. Isso afeta o tráfego para o serviço de DNS, o Instance Metadata Service e o Amazon Time Sync Service. Em vez de rastrear esse evento em tempo real, também podemos transmitir essa métrica para o Amazon Managed Service for Prometheus e visualizá-la no Amazon Managed Grafana

Monitorando atrasos nas consultas de DNS usando métricas do Conntrack

Outra métrica que pode ajudar a monitorar o atraso de limitação/consulta do CoreDNS é e. `conntrack_allowance_available` `conntrack_allowance_exceeded` Falhas de conectividade causadas pela superação das permissões de conexões rastreadas podem ter um impacto maior do que aquelas resultantes da superação de outras permissões. Ao confiar no TCP para transferir dados, os pacotes que são enfileirados ou descartados devido ao excesso das permissões de rede da EC2 instância, como largura de banda, PPS etc., geralmente são tratados normalmente graças aos recursos de controle de congestionamento do TCP. Os fluxos afetados serão reduzidos e os pacotes perdidos serão retransmitidos. No entanto, quando uma instância excede sua permissão de conexões rastreadas, nenhuma nova conexão pode ser estabelecida até que algumas das existentes sejam fechadas para abrir espaço para novas conexões.

`conntrack_allowance_availablee` `conntrack_allowance_exceeded` ajuda os clientes a monitorar a franquia de conexões rastreadas, que varia em cada instância. Essas métricas de desempenho de rede dão aos clientes visibilidade sobre o número de pacotes enfileirados ou descartados quando as permissões de rede de uma instância, como largura de banda de rede Packets-Per-Second (PPS), conexões rastreadas e acesso ao serviço Link-local (Amazon DNS, Instance Meta Data Service, Amazon Time Sync) são excedidas

`conntrack_allowance_available`é o número de conexões rastreadas que podem ser estabelecidas pela instância antes de atingir a permissão de conexões rastreadas desse tipo de instância (compatível somente com instâncias baseadas em nitro). `conntrack_allowance_exceeded`é o número de pacotes descartados porque o rastreamento de conexão excedeu o máximo da instância e não foi possível estabelecer novas conexões.

Outras métricas importantes de desempenho da rede

Outras métricas importantes de desempenho da rede incluem:

`bw_in_allowance_exceeded`(o valor ideal da métrica deve ser zero) é o número de pacotes em fila and/or descartados porque a largura de banda agregada de entrada excedeu o máximo da instância

`bw_out_allowance_exceeded`(o valor ideal da métrica deve ser zero) é o número de pacotes em fila and/or descartados porque a largura de banda agregada de saída excedeu o máximo da instância

`pps_allowance_exceeded`(o valor ideal da métrica deve ser zero) é o número de pacotes and/or descartados na fila porque o PPS bidirecional excedeu o máximo da instância

Capturando as métricas para monitorar cargas de trabalho em busca de problemas de desempenho de rede

O driver do Elastic Network Adapter (ENA) publica as métricas de desempenho da rede discutidas acima a partir das instâncias em que elas estão habilitadas. Todas as métricas de desempenho da rede podem ser publicadas CloudWatch usando o CloudWatch agente. Consulte o blog para obter mais informações.

Agora, vamos capturar as métricas discutidas acima, armazená-las no Amazon Managed Service for Prometheus e visualizá-las usando o Amazon Managed Grafana

Pré-requisitos

- ethtool - Certifique-se de que os nós de trabalho tenham o ethtool instalado
- Um espaço de trabalho AMP configurado em sua conta da AWS. Para obter instruções, consulte Criar um espaço de trabalho no Guia do usuário do AMP.
- Espaço de trabalho gerenciado pela Amazon em Grafana

Implantando o Prometheus Node Exporter

Para coletar as métricas do ethtool para os nós de trabalho, precisamos implantar um coletor que reúna as métricas e as exponha no formato Prometheus. Os comandos a seguir adicionarão o repositório `prometheus-community` Helm e instalarão o `prometheus-node-exporter` gráfico com o `ethtool` coletor ativado.

```
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'"
```

Implante o coletor ADOT para extrair as métricas do ethtool e armazená-las no espaço de trabalho do Amazon Managed Service for Prometheus.

Cada cluster no qual você instala o AWS Distro OpenTelemetry (ADOT) deve ter essa função para conceder à sua conta de serviço da AWS permissões para armazenar métricas no Amazon Managed Service for Prometheus. Siga estas etapas para criar e associar seu perfil do IAM à sua conta de serviço do Amazon EKS utilizando o 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
```

Vamos implantar o coletor ADOT para extrair as métricas do exportador prometheus ethtool e armazená-las no Amazon Managed Service for Prometheus

O procedimento a seguir usa um exemplo de arquivo YAML com a implantação como o valor do modo. Este é o modo padrão e implanta o Coletor ADOT de forma semelhante a uma aplicação autônoma. Essa configuração recebe métricas de OTLP do aplicativo de amostra e métricas do Amazon Managed Service for Prometheus extraídas de pods no cluster.

```
curl -o collector-config-amp.yaml https://raw.githubusercontent.com/aws-observability/
aws-otel-community/master/sample-configs/operator/collector-config-amp.yaml
```

Em collector-config-amp .yaml, substitua o seguinte pelos seus próprios valores:

- modo: implantação
- Conta de serviço: coletor de pontos
- ponto final: <YOUR_REMOTE_WRITE_ENDPOINT>
- região: <YOUR_ >AWS_REGION
- nome: adot-collector

```
kubectl apply -f collector-config-amp.yaml
```

Depois que o coletor de adot for implantado, as métricas serão armazenadas com sucesso no Amazon Prometheus

Configure o gerenciador de alertas no Amazon Managed Service para que o Prometheus envie notificações

Você pode usar o gerenciador de alertas no Amazon Managed Service for Prometheus para configurar regras de alerta para alertas críticos e, em seguida, enviar notificações para um tópico do Amazon SNS. Vamos configurar regras de gravação e regras de alerta para verificar as métricas discutidas até agora.

Usaremos o ACK Controller for Amazon Managed Service for Prometheus para provisionar as regras de alerta e gravação.

Vamos implantar o controlador ACL para o serviço 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
```

Execute o comando e depois de alguns instantes você verá a seguinte mensagem:

```
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.
```

Agora, vamos criar um arquivo yaml para provisionar a definição do gerenciador de alertas e os grupos de regras. Salve o arquivo abaixo 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"
```

Substitua seu ID DO ESPAÇO DE TRABALHO pelo ID do espaço de trabalho que você está usando.

Agora vamos configurar a definição do gerenciador de alertas. Salve o arquivo abaixo como `alertmanager.yaml`

```
apiVersion: prometheusservice.services.k8s.aws/v1alpha1
kind: AlertManagerDefinition
```

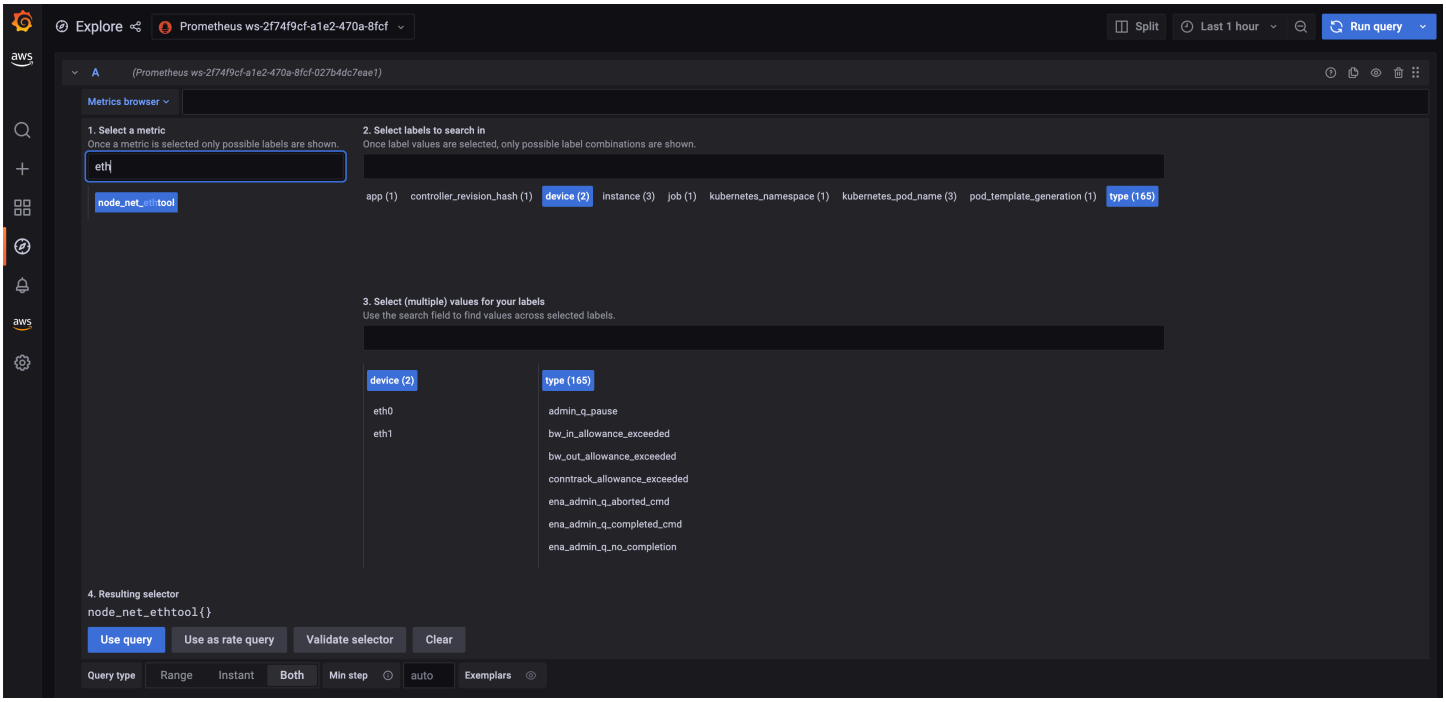
```
metadata:
  name: alert-manager
spec:
  workspaceID: <Your WORKSPACE-ID >
  configuration: |
    alertmanager_config: |
      route:
         receiver: default_receiver
       receivers:
       - name: default_receiver
          sns_configs:
          - topic_arn: TOPIC-ARN
            sigv4:
              region: REGION
            message: |
              alert_type: {{ .CommonLabels.alertname }}
              event_type: {{ .CommonLabels.event_type }}
```

Substitua You WORKSPACE-ID pelo Workspace ID do novo espaço de trabalho, TOPIC-ARN pelo ARN de um tópico do Amazon Simple Notification Service para o qual você deseja enviar os alertas e REGION pela região atual da carga de trabalho. Certifique-se de que seu espaço de trabalho tenha permissões para enviar mensagens para o Amazon SNS.

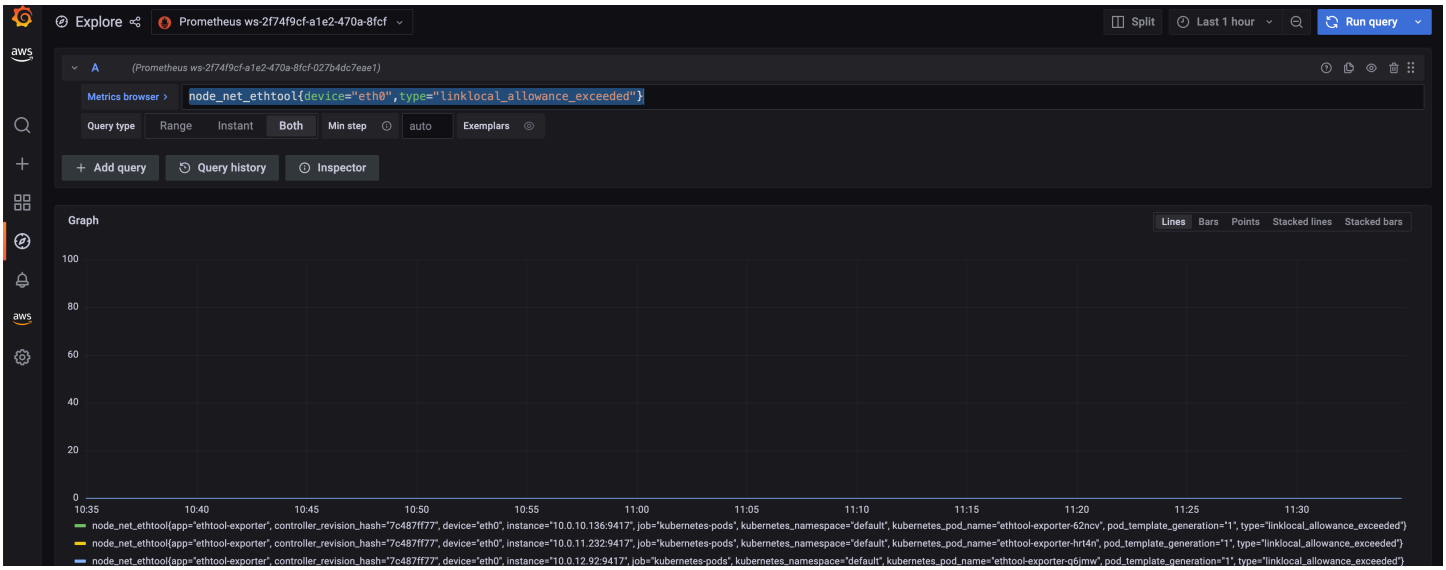
Visualize as métricas do ethtool no Amazon Managed Grafana

Vamos visualizar as métricas na Amazon Managed Grafana e criar um painel. Configure o Amazon Managed Service para Prometheus como uma fonte de dados dentro do console Amazon Managed Grafana. Para obter instruções, consulte Adicionar o Amazon Prometheus como fonte de dados

Vamos explorar as métricas no Amazon Managed Grafana agora: clique no botão explorar e pesquise por ethtool:



Vamos criar um painel para a métrica linklocal_allowance_exceeded usando a consulta. `rate(node_net_ethtool{device="eth0",type="linklocal_allowance_exceeded"}[30s])` Isso resultará no painel abaixo.

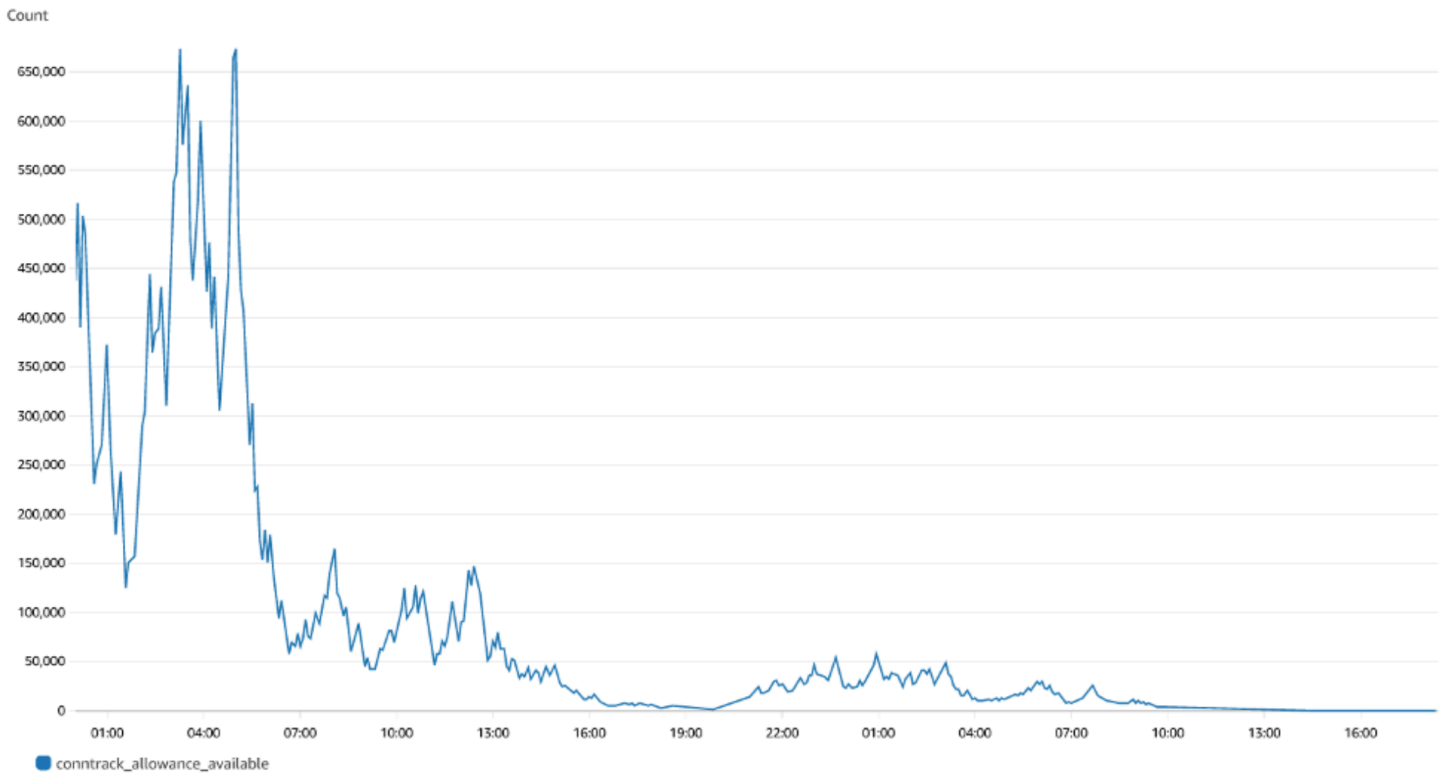


Podemos ver claramente que nenhum pacote foi descartado, pois o valor é zero.

Vamos criar um painel para a métrica conntrack_allowance_exceeded usando a consulta. `rate(node_net_ethtool{device="eth0",type="conntrack_allowance_exceeded"}[30s])` Isso resultará no painel abaixo.



A métrica `conntrack_allowance_exceeded` pode ser visualizada em CloudWatch, desde que você execute um agente do cloudwatch conforme descrito aqui. O painel resultante CloudWatch terá a seguinte aparência:



Podemos ver claramente que nenhum pacote foi descartado, pois o valor é zero. Se você estiver usando instâncias baseadas em Nitro, poderá criar um painel semelhante `conntrack_allowance_available` e monitorar proativamente as conexões em sua EC2

instância. Você pode ampliar ainda mais isso configurando alertas no Amazon Managed Grafana para enviar notificações para Slack, SNS, Pagerduty etc.

Executando o kube-proxy no modo IPVS

O EKS no modo IP Virtual Server (IPVS) resolve o problema de latência de rede frequentemente visto ao executar grandes clusters com mais de 1.000 serviços com o kube-proxy em execução no modo iptables legado. Esse problema de desempenho é o resultado do processamento sequencial das regras de filtragem de pacotes iptables para cada pacote. Esse problema de latência foi resolvido no nftables, o sucessor do iptables. No entanto, no momento em que este artigo foi escrito, o kube-proxy ainda estava em desenvolvimento para fazer uso de nftables. Para contornar esse problema, você pode configurar seu cluster para ser executado `kube-proxy` no modo IPVS.

Visão geral do

O IPVS, que é GA desde a versão 1.11 do Kubernetes, usa tabelas de hash em vez de pesquisa linear para processar pacotes, fornecendo eficiência para clusters com milhares de nós e serviços. O IPVS foi projetado para balanceamento de carga, tornando-o uma solução adequada para problemas de desempenho de rede do Kubernetes.

O IPVS oferece várias opções para distribuir tráfego para pods de back-end. Informações detalhadas de cada opção podem ser encontradas na documentação oficial do Kubernetes, mas uma lista simples é mostrada abaixo. Round Robin e Least Connections estão entre as opções mais populares de balanceamento de carga IPVS no 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)
```

Implementação

São necessárias apenas algumas etapas para habilitar o IPVS em seu cluster EKS. A primeira coisa que você precisa fazer é garantir que as imagens do nó de trabalho do EKS tenham o `ipvsadm`

pacote de administração do Linux Virtual Server instalado. Para instalar esse pacote em uma imagem baseada no Fedora, como Amazon Linux 2023, você pode executar o seguinte comando na instância do node de trabalho.

```
sudo dnf install -y ipvsadm
```

Em uma imagem baseada no Debian, como o Ubuntu, o comando de instalação ficaria assim.

```
sudo apt-get install ipvsadm
```

Em seguida, você precisa carregar os módulos do kernel para as opções de configuração do IPVS listadas acima. Recomendamos gravar esses módulos em um arquivo dentro do `/etc/modules-load.d/` diretório para que sobrevivam à reinicialização.

```
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'
```

Você pode executar o comando a seguir para carregar esses módulos em uma máquina que já esteja em execução.

```
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**
>
> É altamente recomendável executar essas etapas do nó de trabalho como parte do processo de inicialização do nó de trabalho por meio do script de dados do usuário ou em qualquer script de construção executado para criar uma AMI personalizada do nó de trabalho.

Em seguida, você configurará o cluster `kube-proxy` DaemonSet para ser executado no modo IPVS. Isso é feito definindo `kube-proxy` `mode` para `ipvs` e `ipvs scheduler` para uma das opções de balanceamento de carga listadas acima, por exemplo: `rr` para Round Robin.

> **Warning**
>
> Essa é uma mudança disruptiva e deve ser realizada fora do horário comercial. Recomendamos fazer essas alterações durante a criação inicial do cluster EKS para minimizar os impactos.

Você pode emitir um comando da AWS CLI para habilitar o IPVS atualizando o `kube-proxy` complemento EKS.

```
aws eks update-addon --cluster-name $CLUSTER_NAME --addon-name kube-proxy \
  --configuration-values '{"ipvs": {"scheduler": "rr"}, "mode": "ipvs"}' \
  --resolve-conflicts OVERWRITE
```

Ou você pode fazer isso modificando o `kube-proxy-config` ConfigMap em seu cluster.

```
kubectl -n kube-system edit cm kube-proxy-config
```

Encontre a `scheduler` configuração abaixo `ipvs` e defina o valor para uma das opções de balanceamento de carga do ipvs listadas acima, por exemplo: `rr` para Round Robin. Encontre a `mode` configuração, cujo padrão é`iptables`, e altere o valor para. `ipvs` O resultado de qualquer uma das opções deve ser semelhante à configuração abaixo.

```
  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
```

Se seus nós de trabalho foram unidos ao seu cluster antes de fazer essas alterações, você precisará reiniciar o DaemonSet kube-proxy.

```
kubectl -n kube-system rollout restart ds kube-proxy
```

Validação

Você pode validar se o cluster e os nós de trabalho estão sendo executados no modo IPVS emitindo o seguinte comando em um dos seus nós de trabalho.

```
sudo ipvsadm -L
```

No mínimo, você deve ver um resultado semelhante ao abaixo, mostrando entradas para o serviço Kubernetes API Server em e `10.100.0.1` o serviço CoreDNS em. `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**
>
> Este exemplo de saída vem de um cluster EKS com um intervalo de endereços IP de serviço de`10.100.0.0/16`.

Melhores práticas de escalabilidade do EKS

Este guia fornece conselhos para escalar clusters EKS. O objetivo de escalar um cluster EKS é maximizar a quantidade de trabalho que um único cluster pode realizar. Usar um único e grande cluster EKS pode reduzir a carga operacional em comparação com o uso de vários clusters, mas tem vantagens e desvantagens para coisas como implantações em várias regiões, isolamento de inquilinos e atualizações de clusters. Neste documento, vamos nos concentrar em como alcançar a máxima escalabilidade com um único cluster.

Como usar este guia

Este guia é destinado a desenvolvedores e administradores responsáveis pela criação e gerenciamento de clusters EKS na AWS. Ele se concentra em algumas práticas genéricas de escalabilidade do Kubernetes, mas não tem especificações para clusters Kubernetes autogerenciados ou clusters executados fora de uma região da AWS com o EKS Anywhere.

Cada tópico tem uma breve visão geral, seguida por recomendações e melhores práticas para operar clusters EKS em grande escala. Os tópicos não precisam ser lidos em uma ordem específica e as recomendações não devem ser aplicadas sem testar e verificar se funcionam em seus clusters.

Entendendo as dimensões de escalabilidade

A escalabilidade é diferente de desempenho e confiabilidade, e todas as três devem ser consideradas ao planejar suas necessidades de cluster e carga de trabalho. À medida que os clusters se expandem, eles precisam ser monitorados, mas este guia não abordará as melhores práticas de monitoramento. O EKS pode ser escalado para tamanhos grandes, mas você precisará planejar como escalar um cluster além de 300 nós ou 5.000 pods. Esses não são números absolutos, mas resultam da colaboração deste guia com vários usuários, engenheiros e profissionais de suporte.

O escalonamento no Kubernetes é multidimensional e não há configurações ou recomendações específicas que funcionem em todas as situações. As principais áreas em que podemos fornecer orientação para escalabilidade incluem:

O plano de controle do Kubernetes em um cluster EKS inclui todos os serviços que a AWS executa e escala automaticamente para você (por exemplo, servidor de API Kubernetes). Escalar o plano

de controle é responsabilidade da AWS, mas usar o plano de controle com responsabilidade é sua responsabilidade.

A escalabilidade do Kubernetes Data Plane lida com recursos da AWS que são necessários para seu cluster e cargas de trabalho, mas estão fora do plano de controle do EKS. Todos os recursos, incluindo EC2 instâncias, kubelet e armazenamento, precisam ser escalados à medida que seu cluster é escalado.

Os serviços de cluster são controladores e aplicativos Kubernetes que são executados dentro do cluster e fornecem funcionalidade para seu cluster e suas cargas de trabalho. Podem ser complementos do EKS e também outros serviços ou gráficos do Helm que você instala para conformidade e integrações. Esses serviços geralmente dependem das cargas de trabalho e, à medida que suas cargas de trabalho são escaladas, seus serviços de cluster precisarão ser escalados com elas.

As cargas de trabalho são o motivo pelo qual você tem um cluster e devem ser escaladas horizontalmente com o cluster. Existem integrações e configurações que as cargas de trabalho têm no Kubernetes que podem ajudar a escalar o cluster. Também há considerações arquitetônicas com abstrações do Kubernetes, como namespaces e serviços.

Dimensionamento extra grande

Se você estiver escalando um único cluster além de 1.000 nós ou 50.000 pods, adoraríamos falar com você. Recomendamos entrar em contato com sua equipe de suporte ou gerente técnico de contas para entrar em contato com especialistas que podem ajudá-lo a planejar e expandir além das informações fornecidas neste guia. O Amazon EKS pode suportar até 100.000 nós em um único cluster se você for selecionado para integração.

Plano de controle do Kubernetes

O plano de controle do Kubernetes consiste no Kubernetes API Server, no Kubernetes Controller Manager, no Scheduler e em outros componentes necessários para o funcionamento do Kubernetes. Os limites de escalabilidade desses componentes são diferentes dependendo do que você está executando no cluster, mas as áreas com maior impacto na escalabilidade incluem a versão, a utilização e o escalonamento individual do Node do Kubernetes.

Limite a carga de trabalho e a explosão de nós

> **Important**
>
> Para evitar atingir os limites da API no plano de controle, você deve limitar os picos de escala que aumentam o tamanho do cluster em porcentagens de dois dígitos por vez (por exemplo, 1000 nós a 1100 nós ou 4000 a 4500 pods de uma vez).

O plano de controle do EKS será escalado automaticamente à medida que seu cluster cresce, mas há limites para a rapidez com que ele será escalado. Quando você cria um cluster EKS pela primeira vez, o plano de controle não poderá ser escalado imediatamente para centenas de nós ou milhares de pods. Para ler mais sobre como o EKS fez melhorias de escalabilidade, consulte esta postagem no blog.

O escalonamento de aplicativos grandes exige que a infraestrutura se adapte para ficar totalmente pronta (por exemplo, aquecendo balanceadores de carga). Para controlar a velocidade do escalonamento, certifique-se de escalar com base nas métricas certas para seu aplicativo. O escalonamento de CPU e memória pode não prever com precisão as restrições do seu aplicativo, e usar métricas personalizadas (por exemplo, solicitações por segundo) no Kubernetes Horizontal Pod Autoscaler (HPA) pode ser uma melhor opção de escalabilidade.

Para usar uma métrica personalizada, consulte os exemplos na documentação do Kubernetes. Se você tiver necessidades de escalabilidade mais avançadas ou precisar escalar com base em fontes externas (por exemplo, fila AWS SQS), use o KEDA para escalonamento de carga de trabalho baseado em eventos.

Reduza a escala de nós e pods com segurança

Substitua instâncias de longa execução

A substituição regular de nós mantém seu cluster íntegro, evitando desvios de configuração e problemas que só ocorrem após um tempo de atividade prolongado (por exemplo, vazamentos lentos de memória). A substituição automatizada fornecerá bons processos e práticas para atualizações de nós e correções de segurança. Se cada nó do seu cluster for substituído regularmente, haverá menos trabalho necessário para manter processos separados para manutenção contínua.

Use as configurações de tempo de vida (TTL) do Karpenter para substituir as instâncias depois que elas estiverem em execução por um determinado período de tempo. Grupos de nós autogerenciados podem usar a `max-instance-lifetime` configuração para alternar nós automaticamente. Atualmente, os grupos de nós gerenciados não têm esse recurso, mas você pode rastrear a solicitação aqui GitHub.

Remova nós subutilizados

Você pode remover os nós quando eles não têm cargas de trabalho em execução usando o limite de redução no autoescalador de cluster do Kubernetes com o `--scale-down-utilization-threshold`ou, no Karpenter, você pode usar a configuração do provisionador. `ttlSecondsAfterEmpty`

Use orçamentos de interrupção de pods e desligamento seguro de nós

A remoção de pods e nós de um cluster Kubernetes exige que os controladores façam atualizações em vários recursos (por exemplo). EndpointSlices Fazer isso com frequência ou com muita rapidez pode causar limitação do servidor de API e interrupções no aplicativo à medida que as alterações se propagam para os controladores. Os orçamentos de interrupção do pod são uma prática recomendada para diminuir a rotatividade e proteger a disponibilidade da carga de trabalho à medida que os nós são removidos ou reprogramados em um cluster.

Use o cache do lado do cliente ao executar o Kubectl

Usar o comando kubectl de forma ineficiente pode adicionar carga adicional ao servidor da API Kubernetes. Você deve evitar executar scripts ou automações que usem kubectl repetidamente (por exemplo, em um loop for) ou executar comandos sem um cache local.

`kubectl`tem um cache do lado do cliente que armazena em cache as informações de descoberta do cluster para reduzir a quantidade de chamadas de API necessárias. O cache é ativado por padrão e é atualizado a cada 10 minutos.

Se você executar o kubectl a partir de um contêiner ou sem um cache do lado do cliente, poderá ter problemas de limitação de API. É recomendável manter o cache do cluster montando o `--cache-dir` para evitar fazer chamadas de API desnecessárias.

Desativar a compactação kubectl

Desativar a compactação kubectl em seu arquivo kubeconfig pode reduzir o uso da API e da CPU do cliente. Por padrão, o servidor comprime os dados enviados ao cliente para otimizar a largura

de banda da rede. Isso adiciona carga de CPU no cliente e no servidor para cada solicitação e a desativação da compactação pode reduzir a sobrecarga e a latência se você tiver largura de banda adequada. Para desativar a compactação, você pode usar o `--disable-compression=true` sinalizador ou definido `disable-compression: true` em seu arquivo kubeconfig.

```
apiVersion: v1
clusters:
- cluster:
    server: serverURL
    disable-compression: true
  name: cluster
```

Autoescalador de clusters de fragmentos

O Kubernetes Cluster Autoscaler foi testado para escalar até 1000 nós. Em um cluster grande com mais de 1000 nós, é recomendável executar várias instâncias do autoescalador de cluster no modo fragmentado. Cada instância do Cluster Autoscaler é configurada para escalar um conjunto de grupos de nós. O exemplo a seguir mostra duas configurações de escalonamento automático de cluster que são configuradas para cada grupo de 4 nós de escala.

ClusterAutoscaler-1

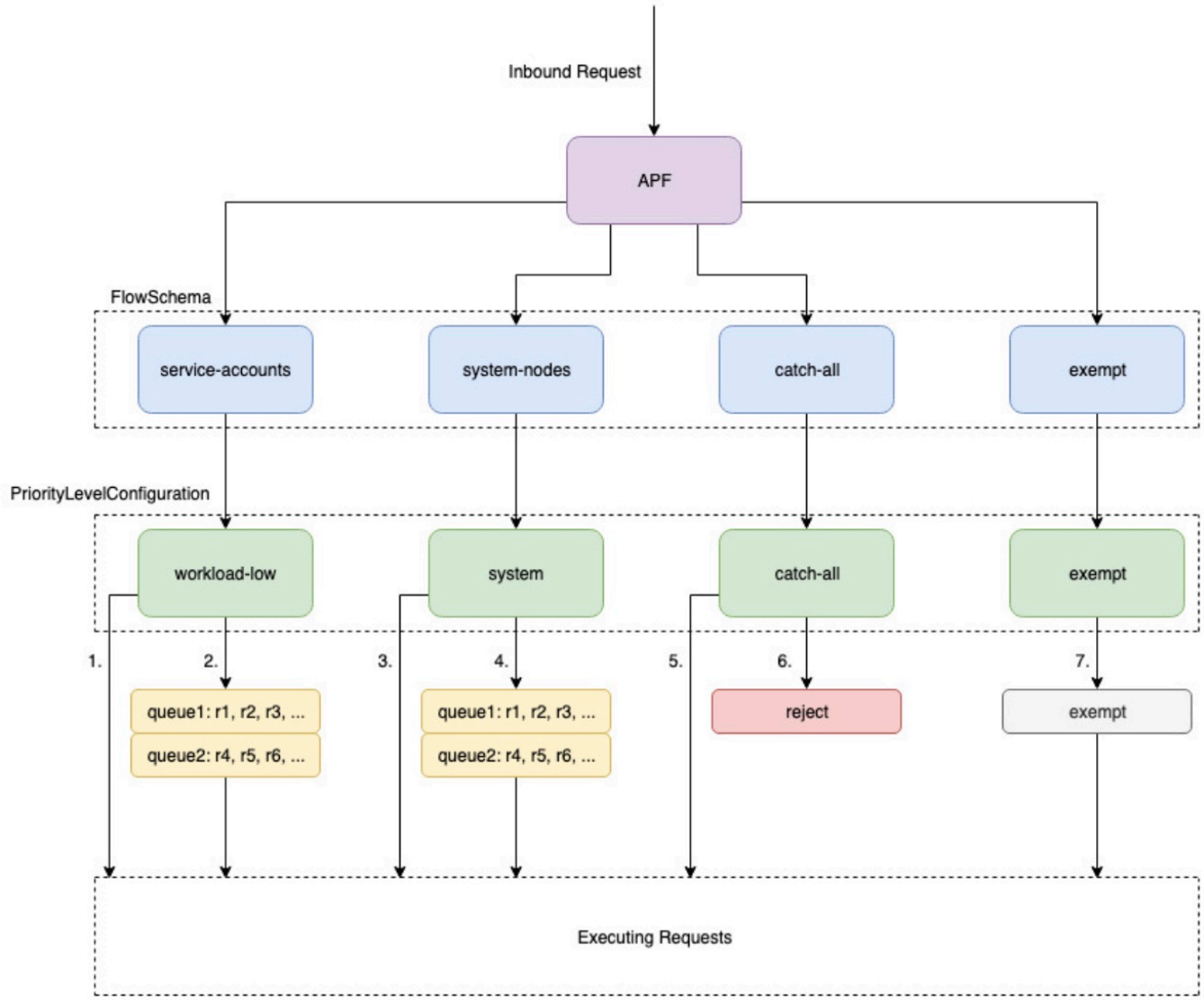
```
autoscalingGroups:
- name: eks-core-node-grp-20220823190924690000000011-80c1660e-030d-476d-cb0d-
d04d585a8fcb
  maxSize: 50
  minSize: 2
- name: eks-data_m1-20220824130553925600000011-5ec167fa-ca93-8ca4-53a5-003e1ed8d306
  maxSize: 450
  minSize: 2
- name: eks-data_m2-20220824130733258600000015-aac167fb-8bf7-429d-d032-e195af4e25f5
  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
```

Prioridade e imparcialidade da 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.

Visão geral do

Para se proteger da sobrecarga durante períodos de aumento de solicitações, o servidor de API limita o número de solicitações de bordo que ele pode ter pendentes em um determinado momento.

Quando esse limite for excedido, o servidor da API começará a rejeitar solicitações e retornará aos clientes um código de resposta HTTP 429 para “Muitas solicitações”. O servidor descartar solicitações e fazer com que os clientes tentem novamente mais tarde é preferível a não ter limites no número de solicitações do lado do servidor e sobrecarregar o plano de controle, o que pode resultar em desempenho degradado ou indisponibilidade.

O mecanismo usado pelo Kubernetes para configurar como essas solicitações de entrada são divididas entre os diferentes tipos de solicitação é chamado de API Priority and Fairness. O servidor da API configura o número total de solicitações em voo que ele pode aceitar ao somar os valores especificados pelos sinalizadores e. `--max-requests-inflight` `--max-mutating-requests-inflight` O EKS usa os valores padrão de 400 e 200 solicitações para esses sinalizadores, permitindo que um total de 600 solicitações sejam enviadas em um determinado momento. No entanto, ao escalar o plano de controle para tamanhos maiores em resposta ao aumento da utilização e da rotatividade da carga de trabalho, aumenta correspondentemente a cota de solicitações em voo até 2000 (sujeita a alterações). O APF especifica como essas cotas de solicitações de bordo são subdivididas entre os diferentes tipos de solicitação. Observe que os planos de controle EKS estão altamente disponíveis com pelo menos 2 servidores de API registrados em cada cluster. Isso significa que o número total de solicitações em voo que seu cluster pode processar é o dobro (ou mais se for ampliado horizontalmente) da cota de bordo definida por kube-apiserver. Isso equivale a vários milhares de requests/second nos maiores clusters EKS.

Dois tipos de objetos do Kubernetes, chamados PriorityLevelConfigurations e FlowSchemas, configuram como o número total de solicitações é dividido entre os diferentes tipos de solicitação. Esses objetos são mantidos automaticamente pelo servidor da API e o EKS usa a configuração padrão desses objetos para a versão secundária do Kubernetes em questão. PriorityLevelConfigurations representam uma fração do número total de solicitações permitidas. Por exemplo, a carga de trabalho máxima PriorityLevelConfiguration é alocada em 98 do total de 600 solicitações. A soma das solicitações alocadas para todos PriorityLevelConfigurations será igual a 600 (ou um pouco acima de 600, porque o servidor de API arredondará para cima se um determinado nível receber uma fração de uma solicitação). Para verificar o PriorityLevelConfigurations em seu cluster e o número de solicitações alocadas para cada um, você pode executar o comando a seguir. Estes são os padrões no 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
```

O segundo tipo de objeto é FlowSchemas. As solicitações do API Server com um determinado conjunto de propriedades são classificadas sob o mesmo FlowSchema. Essas propriedades incluem o usuário autenticado ou os atributos da solicitação, como o grupo de API, o namespace ou o recurso. A FlowSchema também especifica para qual PriorityLevelConfiguration tipo de solicitação deve ser mapeado. Os dois objetos juntos dizem: “Quero que esse tipo de solicitação conte para essa parcela de solicitações de bordo”. Quando uma solicitação chega ao servidor da API, ela verifica cada uma delas FlowSchemas até encontrar uma que corresponda a todas as propriedades necessárias. Se várias FlowSchemas corresponderem a uma solicitação, o servidor da API escolherá aquela FlowSchema com a menor precedência correspondente, especificada como uma propriedade no objeto.

O mapeamento de FlowSchemas to PriorityLevelConfigurations pode ser visualizado usando 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 pode ter um tipo de fila, rejeição ou isenção. Para os tipos Queue e Reject, um limite é imposto ao número máximo de solicitações em voo para esse nível de prioridade, no entanto, o comportamento é diferente quando esse limite é atingido. Por exemplo, o workload-high PriorityLevelConfiguration usa o tipo Queue e tem 98 solicitações disponíveis para uso pelo controller-manager, endpoint-controller, scheduler, controladores relacionados ao eks e de pods em execução no namespace kube-system. Como o tipo Queue é usado, o servidor da API tentará manter as solicitações na memória e espera que o número de solicitações em voo caia abaixo de 98 antes que essas solicitações expirem. Se uma determinada solicitação atingir o tempo limite na fila ou se muitas solicitações já estiverem na fila, o servidor de API não terá escolha a não ser descartar a solicitação e devolver ao cliente um 429. Observe que o enfileiramento pode impedir que uma solicitação receba um 429, mas tem a desvantagem de aumentar end-to-end a latência na solicitação.

Agora, considere o catch-all FlowSchema que é mapeado para o catch-all com o tipo Reject PriorityLevelConfiguration . Se os clientes atingirem o limite de 13 solicitações em voo, o servidor da API não fará filas e eliminará as solicitações instantaneamente com um código de resposta 429. Por fim, as solicitações mapeadas para um PriorityLevelConfiguration com o tipo Isento nunca receberão um 429 e sempre serão enviadas imediatamente. Isso é usado para solicitações de alta prioridade, como solicitações healthz ou solicitações provenientes do grupo system:masters.

Monitorando APF e solicitações descartadas

Para confirmar se alguma solicitação está sendo descartada devido ao APF, as métricas do servidor de API `apiserver_flowcontrol_rejected_requests_total` podem ser monitoradas para verificar o impacto e. FlowSchemas PriorityLevelConfigurations Por exemplo, essa métrica mostra que 100 solicitações das contas de serviço FlowSchema foram descartadas devido ao tempo limite das solicitações em filas de baixa carga de trabalho:

```
% 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 verificar o quão perto um determinado PriorityLevelConfiguration está de receber 429 segundos ou experimentar um aumento na latência devido ao enfileiramento, você pode comparar a diferença entre o limite de simultaneidade e a simultaneidade em uso. Neste exemplo, temos um buffer de 100 solicitações.

```
% 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 verificar se um determinado PriorityLevelConfiguration está passando por filas, mas não necessariamente por solicitações canceladas, a métrica para `apiserver_flowcontrol_current_inqueue_requests` pode ser referenciada:

```
% 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
```

Outras métricas úteis do Prometheus incluem:

- apiserver_flowcontrol_dispatched_requests_total
- apiserver_flowcontrol_request_execution_seconds
- apiserver_flowcontrol_request_wait_duration_seconds

Consulte a documentação inicial para obter uma lista completa das métricas do APF.

Evitando solicitações descartadas

Evite 429 alterando sua carga de trabalho

Quando o APF está descartando solicitações devido ao fato de uma determinada PriorityLevelConfiguration exceder o número máximo permitido de solicitações em voo, os clientes

afetados FlowSchemas podem diminuir o número de solicitações em execução em um determinado momento. Isso pode ser feito reduzindo o número total de solicitações feitas durante o período em que 429 estão ocorrendo. Observe que solicitações de longa duração, como chamadas caras de listas, são especialmente problemáticas porque contam como uma solicitação de bordo durante todo o período em que estão sendo executadas. Reduzir o número dessas solicitações caras ou otimizar a latência dessas chamadas de lista (por exemplo, reduzindo o número de objetos buscados por solicitação ou passando a usar uma solicitação de observação) pode ajudar a reduzir a simultaneidade total exigida por determinada carga de trabalho.

Evite 429s alterando suas configurações de APF

> ⚠ Warning
>
> Só altere as configurações padrão do APF se você souber o que está fazendo. Configurações incorretas do APF podem resultar na perda de solicitações do API Server e em interrupções significativas na carga de trabalho.

Outra abordagem para evitar solicitações perdidas é alterar o padrão FlowSchemas ou PriorityLevelConfigurations instalar em clusters EKS. O EKS instala as configurações padrão do upstream para FlowSchemas e PriorityLevelConfigurations para a versão secundária do Kubernetes em questão. O servidor da API reconciliará automaticamente esses objetos com seus padrões se modificados, a menos que a seguinte anotação nos objetos seja definida como falsa:

```
metadata:
  annotations:
    apf.kubernetes.io/autoupdate-spec: "false"
```

Em um alto nível, as configurações do APF podem ser modificadas para:

- Aloque mais capacidade de bordo para as solicitações que lhe interessam.
- Isole solicitações não essenciais ou caras que podem reduzir a capacidade de outros tipos de solicitação.

Isso pode ser feito alterando o padrão FlowSchemas PriorityLevelConfigurations e/ou criando novos objetos desses tipos. Os operadores podem aumentar os valores dos PriorityLevelConfigurations objetos relevantes assuredConcurrencyShares para aumentar a fração de solicitações de voo que lhes são alocadas. Além disso, o número de solicitações que podem ser enfileiradas em um

determinado momento também pode ser aumentado se o aplicativo puder lidar com a latência adicional causada pelas solicitações sendo enfileiradas antes de serem despachadas.

Como alternativa, é possível criar PriorityLevelConfigurations objetos novos FlowSchema e específicos para a carga de trabalho do cliente. Lembre-se de que alocar mais assuredConcurrencyShares para solicitações existentes PriorityLevelConfigurations ou novas PriorityLevelConfigurations fará com que o número de solicitações que podem ser processadas por outros buckets seja reduzido, pois o limite geral permanecerá em 600 em voo por servidor de API.

Ao fazer alterações nos padrões do APF, essas métricas devem ser monitoradas em um cluster de não produção para garantir que a alteração das configurações não cause 429s indesejados:

1. A métrica de `apiserver_flowcontrol_rejected_requests_total` deve ser monitorada para todos, FlowSchemas a fim de garantir que nenhum bucket comece a descartar solicitações.
2. Os valores de `apiserver_flowcontrol_nominal_limit_seats` e `apiserver_flowcontrol_current_executing_seats` devem ser comparados para garantir que a simultaneidade em uso não corra o risco de violar o limite desse nível de prioridade.

Um caso de uso comum para definir um novo FlowSchema e PriorityLevelConfiguration é o isolamento. Suponha que queiramos isolar chamadas de eventos de lista de longa duração dos pods para sua própria parcela de solicitações. Isso evitará que solicitações importantes de pods usando as contas de serviço existentes FlowSchema recebam 429 e fiquem sem capacidade de solicitação. Lembre-se de que o número total de solicitações em voo é finito. No entanto, este exemplo mostra que as configurações do APF podem ser modificadas para dividir melhor a capacidade de solicitação para uma determinada carga de trabalho:

Exemplo de FlowSchema objeto para isolar solicitações de eventos da 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
```

- Isso FlowSchema captura todas as chamadas de eventos da lista feitas por contas de serviço no namespace padrão.
- A precedência correspondente 8000 é menor do que o valor de 9000 usado pelas contas de serviço existentes, FlowSchema portanto, essas chamadas de eventos da lista list-events-default-service corresponderão a -accounts em vez de service-accounts.
- Estamos usando o catch-all PriorityLevelConfiguration para isolar essas solicitações. Esse bucket permite que apenas 13 solicitações de bordo sejam usadas por essas chamadas de eventos de lista de longa duração. Os pods começarão a receber 429 assim que tentarem emitir mais de 13 dessas solicitações simultaneamente.

Recuperação de recursos no servidor da API

Obter informações do servidor da API é um comportamento esperado para clusters de qualquer tamanho. À medida que você escala o número de recursos no cluster, a frequência das solicitações e o volume de dados podem rapidamente se tornar um gargalo para o plano de controle e levar à latência e lentidão da API. Dependendo da gravidade da latência, isso causa um tempo de inatividade inesperado se você não tomar cuidado.

Estar ciente do que você está solicitando e com que frequência são os primeiros passos para evitar esses tipos de problemas. Aqui está uma orientação para limitar o volume de consultas com base nas melhores práticas de escalabilidade. As sugestões nesta seção são fornecidas para começar com as opções que são conhecidas por serem as melhores escaláveis.

Use informantes compartilhados

Ao criar controladores e automação que se integram à API do Kubernetes, você geralmente precisará obter informações dos recursos do Kubernetes. Se você pesquisar esses recursos regularmente, isso pode causar uma carga significativa no servidor da API.

Usar um informante da biblioteca client-go proporcionará a você a vantagem de observar as alterações nos recursos com base em eventos, em vez de pesquisar mudanças. Os informantes reduzem ainda mais a carga usando cache compartilhado para os eventos e alterações, de forma que vários controladores observando os mesmos recursos não adicionem carga adicional.

Os controladores devem evitar pesquisar recursos de todo o cluster sem rótulos e seletores de campo, especialmente em clusters grandes. Cada pesquisa não filtrada exige que muitos dados desnecessários sejam enviados do etcd por meio do servidor da API para serem filtrados pelo cliente. Ao filtrar com base em rótulos e namespaces, você pode reduzir a quantidade de trabalho que o servidor da API precisa realizar para atender à solicitação e aos dados enviados ao cliente.

Otimize o uso da API Kubernetes

Ao chamar a API Kubernetes com controladores personalizados ou automação, é importante limitar as chamadas somente aos recursos necessários. Sem limites, você pode causar uma carga desnecessária no servidor da API e no etcd.

É recomendável usar o argumento watch sempre que possível. Sem argumentos, o comportamento padrão é listar objetos. Para usar watch em vez de list, você pode `?watch=true` anexar ao final da sua solicitação de API. Por exemplo, para colocar todos os pods no namespace padrão com um relógio, use:

```
/api/v1/namespaces/default/pods?watch=true
```

Se você estiver listando objetos, você deve limitar o escopo do que está listando e a quantidade de dados retornados. Você pode limitar os dados retornados adicionando `limit=500` argumentos às solicitações. O `fieldSelector` argumento e o `/namespace/` caminho podem ser úteis para garantir que suas listas tenham um escopo tão restrito quanto necessário. Por exemplo, para listar somente pods em execução no namespace padrão, use o caminho e os argumentos da API a seguir.

```
/api/v1/namespaces/default/pods?fieldSelector=status.phase=Running&limit=500
```

Ou liste todos os pods que estão sendo executados com:

```
/api/v1/pods?fieldSelector=status.phase=Running&limit=500
```

Outra opção para limitar as chamadas de observação ou os objetos listados é usar a `resourceVersions`qual você pode ler na documentação do Kubernetes. Sem uma `resourceVersion` discussão, você receberá a versão mais recente disponível, que requer uma leitura de quorum do etcd, que é a leitura mais cara e mais lenta do banco de dados. A ResourceVersion depende dos recursos que você está tentando consultar e pode ser encontrada no `metadata.resourseVersion` campo. Isso também é recomendado no caso de usar chamadas de observação e não apenas de listas de chamadas.

Há um especial `resourceVersion=0` disponível que retornará os resultados do cache do servidor da API. Isso pode reduzir a carga do etcd, mas não suporta paginação.

```
/api/v1/namespaces/default/pods?resourceVersion=0
```

É recomendável usar watch com uma resourceVersion definida como o valor conhecido mais recente recebido da lista ou observação anterior. Isso é tratado automaticamente no client-go. Mas é recomendável verificar novamente se você estiver usando um cliente k8s em outros idiomas.

```
/api/v1/namespaces/default/pods?watch=true&resourceVersion=362812295
```

Se você chamar a API sem nenhum argumento, ela consumirá mais recursos para o servidor da API e etcd. Essa chamada obterá todos os pods em todos os namespaces sem paginação ou limitação do escopo e exigirá um quorum de leitura do etcd.

```
/api/v1/pods
```

Evite DaemonSet rebanhos trovejantes

A DaemonSet garante que todos (ou alguns) nós executem uma cópia de um pod. À medida que os nós se juntam ao cluster, o daemonset-controller cria pods para esses nós. Quando os nós saem do cluster, esses pods são coletados como lixo. Excluir um DaemonSet limpará os pods criados por ele.

Alguns usos típicos de a DaemonSet são:

- Executando um daemon de armazenamento em cluster em cada nó
- Executando um daemon de coleta de registros em cada nó
- Executando um daemon de monitoramento de nós em cada nó

Em clusters com milhares de nós, criar um novo DaemonSet, atualizar um DaemonSet ou aumentar o número de nós pode resultar em uma alta carga colocada no plano de controle. Se DaemonSet os pods emitirem solicitações caras do servidor de API na inicialização do pod, eles podem causar um alto uso de recursos no plano de controle devido a um grande número de solicitações simultâneas.

Em operação normal, você pode usar a `RollingUpdate` para garantir o lançamento gradual de novos DaemonSet pods. Com uma estratégia de `RollingUpdate` atualização, depois de atualizar um DaemonSet modelo, o controlador mata DaemonSet pods antigos e cria novos DaemonSet pods automaticamente de forma controlada. No máximo, um pod do DaemonSet será executado em cada nó durante todo o processo de atualização. Você pode realizar uma distribuição gradual definindo `maxUnavailable` como 1, `maxSurge` 0 e `minReadySeconds` 60. Se você não especificar uma estratégia de atualização, o Kubernetes usará como padrão a criação de um `RollingUpdate` com `maxUnavailable` como 1, `maxSurge` como 0 e `minReadySeconds` como 0.

```
minReadySeconds: 60
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 0
    maxUnavailable: 1
```

A `RollingUpdate` garante a implantação gradual de novos DaemonSet pods se eles já tiverem sido criados e tiverem o DaemonSet número esperado de `Ready` pods em todos os nós. Problemas com o rebanho trovejante podem ocorrer sob certas condições que não são cobertas pelas estratégias. `RollingUpdate`

Evite rebanhos trovejantes na criação DaemonSet

Por padrão, independentemente da `RollingUpdate` configuração, o daemonset-controller no kube-controller-manager criará pods para todos os nós correspondentes simultaneamente quando você criar um novo. DaemonSet Para forçar o lançamento gradual dos pods depois de criar um DaemonSet, você pode usar um ou. `NodeSelector` `NodeAffinity` Isso criará um DaemonSet que corresponda a zero nós e, em seguida, você poderá atualizar gradualmente os nós para torná-los elegíveis para executar um pod a partir do a uma DaemonSet taxa controlada. Você pode seguir essa abordagem:

- Adicione um rótulo a todos os nós do`run-daemonset=false`.

```
kubectl label nodes --all run-daemonset=false
```

- Crie o seu DaemonSet com uma `NodeAffinity` configuração que corresponda a qualquer nó sem um `run-daemonset=false` rótulo. Inicialmente, isso fará com que você não DaemonSet tenha cápsulas correspondentes.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: run-daemonset
          operator: NotIn
          values:
          - "false"
```

- Remova a `run-daemonset=false` etiqueta dos seus nós em uma taxa controlada. Você pode usar esse script bash como exemplo:

```
#!/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
```

- Opcionalmente, remova a `NodeAffinity` configuração do seu DaemonSet objeto. Observe que isso também acionará a `RollingUpdate` e substituirá gradualmente todos os DaemonSet pods existentes porque o DaemonSet modelo foi alterado.

Evite rebanhos trovejantes em escalabilidade horizontal de nós

Da mesma forma que a DaemonSet criação, a criação rápida de novos nós pode resultar em um grande número de DaemonSet pods iniciando simultaneamente. Você deve criar novos nós em

uma taxa controlada para que o controlador crie DaemonSet pods na mesma taxa. Se isso não for possível, você pode tornar os novos nós inicialmente inelegíveis para os existentes DaemonSet usando`NodeAffinity`. Em seguida, você pode adicionar um rótulo aos novos nós gradualmente para que o daemonset-controller crie pods em uma taxa controlada. Você pode seguir essa abordagem:

- Adicione um rótulo a todos os nós existentes para `run-daemonset=true`

```
kubectl label nodes --all run-daemonset=true
```

- Atualize seu DaemonSet com uma `NodeAffinity` configuração para combinar com qualquer nó com um `run-daemonset=true` rótulo. Observe que isso também acionará a `RollingUpdate` e substituirá gradualmente todos os DaemonSet pods existentes porque o DaemonSet modelo foi alterado. Você deve esperar que o `RollingUpdate` seja concluído antes de avançar para a próxima etapa.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: run-daemonset
          operator: In
          values:
          - "true"
```

- Crie novos nós em seu cluster. Observe que esses nós não terão o `run-daemonset=true` rótulo, portanto, não DaemonSet corresponderão a esses nós.
- Adicione o `run-daemonset=true` rótulo aos seus novos nós (que atualmente não têm o `run-daemonset` rótulo) em uma taxa controlada. Você pode usar esse script bash como exemplo:

```
#!/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
```

- Opcionalmente, remova a `NodeAffinity` configuração do seu DaemonSet objeto e remova o `run-daemonset` rótulo de todos os nós.

Evite rebanhos trovejantes nas atualizações DaemonSet

Uma `RollingUpdate` política respeitará apenas a `maxUnavailable` configuração de DaemonSet pods que sejam`Ready`. Se a DaemonSet tiver apenas `NotReady` pods ou uma grande porcentagem de `NotReady` pods e você atualizar seu modelo, o daemonset-controller criará novos pods simultaneamente para qualquer pod. `NotReady` Isso pode resultar em problemas de rebanho trovejante se houver um número significativo de `NotReady` pods, por exemplo, se os pods falharem continuamente ou não conseguirem extrair imagens.

Para forçar o lançamento gradual de pods ao atualizar um DaemonSet e houver `NotReady` pods, você pode alterar temporariamente a estratégia de atualização no de para. DaemonSet `RollingUpdate` `OnDelete` Com`OnDelete`, depois de atualizar um DaemonSet modelo, o controlador cria novos pods depois que você exclui manualmente os antigos para que você possa controlar a implantação de novos pods. Você pode seguir essa abordagem:

- Verifique se você tem alguma `NotReady` cápsula no seu DaemonSet.
- Se não, você pode atualizar o DaemonSet modelo com segurança e a `RollingUpdate` estratégia garantirá uma implementação gradual.
- Se sim, você deve primeiro atualizá-lo DaemonSet para usar a `OnDelete` estratégia.

```
updateStrategy:
  type: OnDelete
```

- Em seguida, atualize seu DaemonSet modelo com as alterações necessárias.
- Após essa atualização, você pode excluir os DaemonSet pods antigos emitindo solicitações de exclusão de pods em uma taxa controlada. Você pode usar esse script bash como exemplo em que o DaemonSet nome é fluentd-elasticsearch no namespace kube-system:

```
#!/bin/bash

daemonset_pods=$(kubectl get --raw "/api/v1/namespaces/kube-system/pods?
labelSelector=name%3Dfluentd-elasticsearch" | jq -r '.items | .[].metadata.name')

for pod in ${daemonset_pods[@]}; do
   echo "Deleting pod $pod"
   kubectl delete pod $pod -n kube-system
   sleep 5
done
```

- Finalmente, você pode atualizar suas DaemonSet costas para a `RollingUpdate` estratégia anterior.

Plano de dados do Kubernetes

Selecionar tipos de EC2 instância é possivelmente uma das decisões mais difíceis que os clientes enfrentam em clusters com várias cargas de trabalho. Não existe one-size-fits uma solução completa. Aqui estão algumas dicas para ajudar você a evitar armadilhas comuns com a escalabilidade da computação.

Escalonamento automático de nós

Recomendamos que você use o escalonamento automático de nós, que reduz o trabalho e se integra profundamente ao Kubernetes. Os grupos de nós gerenciados e o Karpenter são recomendados para clusters de grande escala.

Os grupos de nós gerenciados oferecem a flexibilidade dos grupos do Amazon EC2 Auto Scaling, com benefícios adicionais para atualizações e configurações gerenciadas. Ele pode ser escalado com o Kubernetes Cluster Autoscaler e é uma opção comum para clusters que têm uma variedade de necessidades de computação.

O Karpenter é um autoescalador de nós de código aberto e nativo de carga de trabalho criado pela AWS. Ele escala os nós em um cluster com base nos requisitos de carga de trabalho para recursos (por exemplo, GPU) e manchas e tolerâncias (por exemplo, distribuição de zonas) sem gerenciar grupos de nós. Os nós são criados diretamente, EC2 o que evita cotas padrão de grupos de nós — 450 nós por grupo — e fornece maior flexibilidade de seleção de instâncias com menos sobrecarga operacional. Recomendamos que os clientes usem o Karpenter sempre que possível.

Use vários tipos de EC2 instância diferentes

Cada região da AWS tem um número limitado de instâncias disponíveis por tipo de instância. Se você criar um cluster que usa apenas um tipo de instância e escalar o número de nós além da capacidade da região, receberá um erro informando que nenhuma instância está disponível. Para evitar esse problema, você não deve limitar arbitrariamente o tipo de instâncias que podem ser usadas em seu cluster.

O Karpenter usará um amplo conjunto de tipos de instância compatíveis por padrão e escolherá uma instância no momento do provisionamento com base nos requisitos de carga de trabalho pendentes, na disponibilidade e no custo. Você pode ampliar a lista de tipos de instância usados na `karpenter.k8s.aws/instance-category` chave de NodePools.

O escalonador automático de clusters do Kubernetes exige que os grupos de nós tenham tamanhos semelhantes para que possam ser escalados de forma consistente. Você deve criar vários grupos com base no tamanho da CPU e da memória e escalá-los de forma independente. Use o ec2-instance-selector para identificar instâncias de tamanho semelhante para seus grupos de nós.

```
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
```

Prefira nós maiores para reduzir a carga do servidor de API

Ao decidir quais tipos de instância usar, menos nós grandes colocarão menos carga no plano de controle do Kubernetes porque haverá menos kubelets em execução. DaemonSets No entanto, nós grandes podem não ser totalmente utilizados como nós menores. Os tamanhos dos nós devem ser avaliados com base na disponibilidade da carga de trabalho e nos requisitos de escala.

Um cluster com três instâncias u-24tb1.metal (24 TB de memória e 448 núcleos) tem 3 kubelets e, por padrão, estaria limitado a 110 pods por nó. Se seus pods usam 4 núcleos cada, isso pode ser esperado (4 núcleos x 110 = 440 núcleos/nó). Com um cluster de 3 nós, sua capacidade de lidar com um incidente de instância seria baixa porque a interrupção de 1 instância poderia afetar 1/3 do cluster. Você deve especificar os requisitos de nós e a distribuição de pods em suas cargas de trabalho para que o programador do Kubernetes possa posicionar as cargas de trabalho adequadamente.

As cargas de trabalho devem definir os recursos de que precisam e a disponibilidade exigida por meio de imperfeições, tolerâncias e. PodTopologySpread Eles devem preferir os nós maiores que possam ser totalmente utilizados e que atendam às metas de disponibilidade para reduzir a carga do plano de controle, diminuir as operações e reduzir os custos.

O Kubernetes Scheduler tentará distribuir automaticamente as cargas de trabalho entre zonas de disponibilidade e hosts se houver recursos disponíveis. Se nenhuma capacidade estiver disponível, o Kubernetes Cluster Autoscaler tentará adicionar nós em cada zona de disponibilidade de maneira uniforme. O Karpenter tentará adicionar nós da forma mais rápida e barata possível, a menos que a carga de trabalho especifique outros requisitos.

Para forçar a distribuição das cargas de trabalho com o agendador e a criação de novos nós nas zonas de disponibilidade, você deve usar: 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
```

Use tamanhos de nós semelhantes para um desempenho consistente da carga de trabalho

As cargas de trabalho devem definir em quais nós de tamanho elas precisam ser executadas para permitir desempenho consistente e escalabilidade previsível. Uma carga de trabalho que solicita 500 m de CPU terá um desempenho diferente em uma instância com 4 núcleos versus uma com 16 núcleos. Evite tipos de instância que usam capacidade de intermitência, CPUs como instâncias da série T.

Para garantir que suas cargas de trabalho tenham um desempenho consistente, uma carga de trabalho pode usar os rótulos compatíveis do Karpenter para atingir tamanhos de instâncias específicos.

```
kind: deployment
...
spec:
  template:
    spec:
    containers:
    nodeSelector:
      karpenter.k8s.aws/instance-size: 8xlarge
```

As cargas de trabalho programadas em um cluster com o autoescalador de cluster do Kubernetes devem corresponder um seletor de nós aos grupos de nós com base na correspondência de rótulos.

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/nodegroup
            operator: In
            values:
            - 8-core-node-group    # match your node group name
```

Use os recursos computacionais de forma eficiente

Os recursos computacionais incluem EC2 instâncias e zonas de disponibilidade. Usar recursos computacionais de forma eficaz aumentará sua escalabilidade, disponibilidade, desempenho

e reduzirá seu custo total. O uso eficiente de recursos é extremamente difícil de prever em um ambiente de escalonamento automático com vários aplicativos. O Karpenter foi criado para provisionar instâncias sob demanda com base nas necessidades da carga de trabalho para maximizar a utilização e a flexibilidade.

O Karpenter permite que as cargas de trabalho declarem o tipo de recursos computacionais de que precisam sem primeiro criar grupos de nós ou configurar marcadores para nós específicos. Consulte as melhores práticas da Karpenter para obter mais informações. Considere habilitar a consolidação em seu provisionador Karpenter para substituir os nós que estão subutilizados.

Automatize as atualizações do Amazon Machine Image (AMI)

Manter os componentes do node de trabalho atualizados garantirá que você tenha os patches de segurança mais recentes e os recursos compatíveis com a API Kubernetes. A atualização do kubelet é o componente mais importante para a funcionalidade do Kubernetes, mas automatizar os patches do sistema operacional, do kernel e dos aplicativos instalados localmente reduzirá a manutenção à medida que você expande.

É recomendável usar o Amazon Linux 2 otimizado para Amazon EKS mais recente ou o Bottlerocket AMI otimizado para Amazon EKS para a imagem do seu nó. O Karpenter usará automaticamente a AMI mais recente disponível para provisionar novos nós no cluster. Os grupos de nós gerenciados atualizarão a AMI durante uma atualização do grupo de nós, mas não atualizarão a ID da AMI no momento do provisionamento do nó.

Para grupos de nós gerenciados, você precisa atualizar o modelo de lançamento do Auto Scaling Group (ASG) com a nova AMI IDs quando estiverem disponíveis para lançamentos de patches. Versões secundárias da AMI (por exemplo, 1.23.5 a 1.24.3) estarão disponíveis no console e na API do EKS como atualizações para o grupo de nós. As versões de lançamento de patches (por exemplo, 1.23.5 a 1.23.6) não serão apresentadas como atualizações para os grupos de nós. Se quiser manter seu grupo de nós atualizado com as versões de patch da AMI, você precisa criar uma nova versão do modelo de execução e permitir que o grupo de nós substitua as instâncias pela nova versão da AMI.

Você pode encontrar a AMI mais recente disponível nesta página ou usar a 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
```

Use vários volumes do EBS para contêineres

Os volumes do EBS têm uma cota input/output (E/S) com base no tipo de volume (por exemplo, gp3) e no tamanho do disco. Se seus aplicativos compartilharem um único volume raiz do EBS com o host, isso pode esgotar a cota de disco de todo o host e fazer com que outros aplicativos esperem pela capacidade disponível. Os aplicativos gravam em disco se gravarem arquivos em sua partição de sobreposição, montarem um volume local a partir do host e também quando fizerem login na saída padrão (STDOUT), dependendo do agente de registro usado.

Para evitar o I/O esgotamento do disco, você deve montar um segundo volume na pasta de estado do contêiner (por exemplo, /run/containerd), usar volumes EBS separados para armazenamento da carga de trabalho e desativar o registro local desnecessário.

Para montar um segundo volume em suas EC2 instâncias usando eksctl, você pode usar um grupo de nós com esta configuração:

```
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"
```

Se você estiver usando o terraform para provisionar seus grupos de nós, veja exemplos em EKS Blueprints for terraform. Se você estiver usando o Karpenter para provisionar nós, poderá usar `blockDeviceMappings`com os dados do usuário do nó para adicionar volumes adicionais.

Para montar um volume do EBS diretamente no seu pod, você deve usar o driver CSI do AWS EBS e consumir um volume com uma classe de armazenamento.

```
---
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 instâncias com limites baixos de conexão do EBS se as cargas de trabalho usarem volumes do EBS

O EBS é uma das maneiras mais fáceis de as cargas de trabalho terem armazenamento persistente, mas também vem com limitações de escalabilidade. Cada tipo de instância tem um número máximo de volumes do EBS que podem ser anexados. As cargas de trabalho precisam declarar em quais tipos de instância elas devem ser executadas e limitar o número de réplicas em uma única instância com contaminações do Kubernetes.

Desative o registro desnecessário no disco

Evite o registro local desnecessário ao não executar seus aplicativos com o registro de depuração na produção e desabilitar o registro que lê e grava no disco com frequência. O Journald é o serviço de registro local que mantém um buffer de log na memória e é liberado para o disco periodicamente. O Journald é preferível ao syslog, que registra todas as linhas imediatamente no disco. A desativação do syslog também reduz a quantidade total de armazenamento necessária e evita a necessidade de regras complicadas de rotação de registros. Para desativar o syslog, você pode adicionar o seguinte trecho à sua configuração do cloud-init:

```
runcmd:
  - [ systemctl, disable, --now, syslog.service ]
```

Instâncias de patch em vigor quando a velocidade de atualização do sistema operacional é uma necessidade

> **Important**
>
> A correção de instâncias em vigor só deve ser feita quando necessário. A Amazon recomenda tratar a infraestrutura como imutável e testar minuciosamente as atualizações que são promovidas em ambientes inferiores da mesma forma que os aplicativos. Esta seção se aplica quando isso não é possível.

A instalação de um pacote em um host Linux existente leva alguns segundos sem interromper as cargas de trabalho em contêineres. O pacote pode ser instalado e validado sem isolar, drenar ou substituir a instância.

Para substituir uma instância, primeiro você precisa criar, validar e distribuir uma nova AMIs. A instância precisa ter uma substituição criada, e a instância antiga precisa ser isolada e drenada. Em seguida, as cargas de trabalho precisam ser criadas na nova instância, verificadas e repetidas para todas as instâncias que precisam ser corrigidas. São necessárias horas, dias ou semanas para substituir as instâncias com segurança sem interromper as cargas de trabalho.

A Amazon recomenda o uso de uma infraestrutura imutável criada, testada e promovida a partir de um sistema automatizado e declarativo, mas se você precisar corrigir sistemas rapidamente, precisará corrigir os sistemas existentes e substituí-los à medida que novos AMIs forem disponibilizados. Devido ao grande diferencial de tempo entre a aplicação de patches e a

substituição de sistemas, recomendamos o uso do AWS Systems Manager Patch Manager para automatizar a aplicação de patches nos nós quando necessário.

A aplicação de patches permitirá que você implemente rapidamente atualizações de segurança e substitua as instâncias regularmente após a atualização da AMI. Se você estiver usando um sistema operacional com um sistema de arquivos raiz somente para leitura, como o Flatcar Container Linux ou o Bottlerocket OS, recomendamos usar os operadores de atualização que funcionam com esses sistemas operacionais. O operador de atualização do Flatcar Linux e o operador de atualização do Bottlerocket reiniciarão as instâncias para manter os nós atualizados automaticamente.

Serviços de cluster

Os serviços de cluster são executados dentro de um cluster EKS, mas não são cargas de trabalho do usuário. Se você tem um servidor Linux, geralmente precisa executar serviços como NTP, syslog e um tempo de execução de contêiner para suportar suas cargas de trabalho. Os serviços de cluster são serviços de suporte semelhantes que ajudam você a automatizar e operar seu cluster. No Kubernetes, eles geralmente são executados no namespace kube-system e alguns são executados como. DaemonSets

Espera-se que os serviços de cluster tenham um alto tempo de atividade e geralmente sejam essenciais durante interrupções e para solução de problemas. Se um serviço de cluster principal não estiver disponível, você poderá perder o acesso aos dados que podem ajudar a recuperar ou evitar uma interrupção (por exemplo, alta utilização do disco). Eles devem ser executados em instâncias de computação dedicadas, como um grupo de nós separado ou o AWS Fargate. Isso garantirá que os serviços de cluster não sejam afetados nas instâncias compartilhadas por cargas de trabalho que possam estar aumentando ou usando mais recursos.

CoreDNS de escala

O escalonamento do CoreDNS tem dois mecanismos principais. Reduzindo o número de chamadas para o serviço CoreDNS e aumentando o número de réplicas.

Reduza as consultas externas diminuindo os ndots

A configuração ndots especifica quantos pontos (também conhecidos como “pontos”) em um nome de domínio são considerados suficientes para evitar a consulta ao DNS. Se seu aplicativo tiver uma configuração de ndots de 5 (padrão) e você solicitar recursos de um domínio externo, como api.example.com (2 pontos), o CoreDNS será consultado para cada domínio de pesquisa definido

em /etc/resolv.conf para um domínio mais específico. Por padrão, os seguintes domínios serão pesquisados antes de fazer uma solicitação externa.

```
api.example.<namespace>.svc.cluster.local
api.example.svc.cluster.local
api.example.cluster.local
api.example.<region>.compute.internal
```

Os `region` valores `namespace` e serão substituídos pelo namespace das cargas de trabalho e pela região de computação. Você pode ter domínios de pesquisa adicionais com base nas configurações do seu cluster.

Você pode reduzir o número de solicitações para o CoreDNS diminuindo a opção ndots da sua carga de trabalho ou qualificando totalmente suas solicitações de domínio incluindo um final. (por exemplo`api.example.com.`). Se sua carga de trabalho se conectar a serviços externos via DNS, recomendamos definir ndots como 2 para que as cargas de trabalho não façam consultas desnecessárias de DNS de cluster dentro do cluster. Você pode definir um servidor DNS e um domínio de pesquisa diferentes se a carga de trabalho não exigir acesso aos serviços dentro do cluster.

```
spec:
  dnsPolicy: "None"
  dnsConfig:
    options:
      - name: ndots
        value: "2"
      - name: edns0
```

Se você reduzir ndots para um valor muito baixo ou se os domínios aos quais você está se conectando não incluírem especificidade suficiente (inclusive no final), é possível que as pesquisas de DNS falhem. Certifique-se de testar como essa configuração afetará suas cargas de trabalho.

Dimensione o CoreDNS horizontalmente

As instâncias do CoreDNS podem ser escaladas adicionando réplicas adicionais à implantação. É recomendável usar o NodeLocal DNS ou o autoescalador proporcional do cluster para escalar o CoreDNS.

NodeLocal O DNS exigirá a execução de uma instância por nó, como uma, o DaemonSet que exigirá mais recursos computacionais no cluster, mas evitará falhas nas solicitações de DNS e diminuirá

o tempo de resposta das consultas de DNS no cluster. O autoescalador proporcional do cluster escalará o CoreDNS com base no número de nós ou núcleos no cluster. Essa não é uma correlação direta com consultas de solicitação, mas pode ser útil dependendo das cargas de trabalho e do tamanho do cluster. A escala proporcional padrão é adicionar uma réplica adicional para cada 256 núcleos ou 16 nós no cluster, o que ocorrer primeiro.

Se estiver usando o complemento CoreDNS EKS, considere ativar a opção de escalonamento automático. O autoescalador CoreDNS ajusta dinamicamente o número de réplicas do CoreDNS monitorando a contagem de nós e os núcleos da CPU, usando uma fórmula que usa o máximo de (nódos/16) ou (núcleos de CPU/256), aumentando imediatamente quando necessário e diminuindo gradualmente para manter a estabilidade.

Dimensione o servidor de métricas do Kubernetes verticalmente

O Kubernetes Metrics Server oferece suporte ao escalonamento horizontal e vertical. Ao escalar horizontalmente o Metrics Server, ele estará altamente disponível, mas não será escalado horizontalmente para lidar com mais métricas de cluster. Você precisará escalar verticalmente o Metrics Server com base em suas recomendações à medida que os nós e as métricas coletadas forem adicionados ao cluster.

O Metrics Server mantém os dados que coleta, agrega e veicula na memória. Conforme o cluster cresce, a quantidade de dados que o Metrics Server armazena aumenta. Em grandes clusters, o Metrics Server exigirá mais recursos computacionais do que a reserva de memória e CPU especificada na instalação padrão. Você pode usar o Vertical Pod Autoscaler (VPA) ou o Addon Resizer para escalar o Metrics Server. O Addon Resizer é dimensionado verticalmente em proporção aos nós de trabalho e o VPA é dimensionado com base no uso da CPU e da memória.

Duração do CoreDNS lamduck

Os pods usam o `kube-dns` Serviço para resolução de nomes. O Kubernetes usa NAT de destino (DNAT) para redirecionar o tráfego dos nós para os pods de back-end `kube-dns` do CoreDNS. Conforme você escala a implantação do CoreDNS`kube-proxy`, atualiza as regras e cadeias de iptables nos nós para redirecionar o tráfego de DNS para os pods do CoreDNS. A propagação de novos endpoints ao aumentar a escala e a exclusão de regras ao reduzir o CoreDNS pode levar de 1 a 10 segundos, dependendo do tamanho do cluster.

Esse atraso na propagação pode causar falhas na pesquisa de DNS quando um pod CoreDNS é encerrado, mas as regras de iptables do nó não foram atualizadas. Nesse cenário, o nó pode continuar enviando consultas de DNS para um pod CoreDNS encerrado.

Você pode reduzir as falhas de pesquisa de DNS definindo uma duração lenta em seus pods do CoreDNS. Enquanto estiver no modo idiota, o CoreDNS continuará respondendo às solicitações em voo. Definir uma duração inadequada atrasará o processo de desligamento do CoreDNS, dando aos nós o tempo necessário para atualizar suas regras e cadeias de iptables.

Recomendamos definir a duração do CoreDNS lameduck para 30 segundos.

Sonda de prontidão para CoreDNS

Recomendamos usar `/ready` em vez da sonda `/health` de prontidão do CoreDNS.

De acordo com a recomendação anterior de definir a duração do lameduck em 30 segundos, fornecendo tempo suficiente para que as regras de iptables do nó sejam atualizadas antes do término do pod, empregar, em vez da sonda de prontidão `/ready` do CoreDNS, garante que `/health` o pod CoreDNS esteja totalmente preparado na inicialização para responder prontamente às solicitações de DNS.

```
readinessProbe:
  httpGet:
    path: /ready
    port: 8181
    scheme: HTTP
```

Para obter mais informações sobre o plug-in CoreDNS Ready, consulte https://coredns. io/plugins/ready/

Agentes de registro e monitoramento

Agentes de registro e monitoramento podem adicionar uma carga significativa ao seu plano de controle de cluster porque os agentes consultam o servidor da API para enriquecer registros e métricas com metadados de carga de trabalho. O agente em um nó só tem acesso aos recursos do nó local para ver coisas como nome do contêiner e do processo. Ao consultar o servidor da API, ele pode adicionar mais detalhes, como nome e rótulos de implantação do Kubernetes. Isso pode ser extremamente útil para solucionar problemas, mas prejudicial ao dimensionamento.

Como há muitas opções diferentes de registro e monitoramento, não podemos mostrar exemplos para cada provedor. Com o fluentbit, recomendamos habilitar o Use_Kubelet para buscar metadados do kubelet local em vez do Kubernetes API Server e definir um número que reduza as chamadas repetidas quando os dados podem `Kube_Meta_Cache_TTL` ser armazenados em cache (por exemplo, 60).

O monitoramento e o registro em escala têm duas opções gerais:

- Desativar integrações
- Amostragem e filtragem

Desabilitar as integrações geralmente não é uma opção porque você perde os metadados do log. Isso elimina o problema de escalonamento da API, mas introduzirá outros problemas ao não ter os metadados necessários quando necessário.

A amostragem e a filtragem reduzem o número de métricas e registros coletados. Isso reduzirá a quantidade de solicitações para a API Kubernetes e reduzirá a quantidade de armazenamento necessária para as métricas e os registros coletados. Reduzir os custos de armazenamento reduzirá o custo geral do sistema.

A capacidade de configurar a amostragem depende do software do agente e pode ser implementada em diferentes pontos de ingestão. É importante adicionar a amostragem o mais próximo possível do agente, pois provavelmente é aí que as chamadas do servidor da API acontecem. Entre em contato com seu provedor para saber mais sobre o suporte à amostragem.

Se você estiver usando CloudWatch um CloudWatch Logs, poderá adicionar a filtragem de agentes usando os padrões descritos na documentação.

Para evitar a perda de registros e métricas, você deve enviar seus dados para um sistema que possa armazenar dados em buffer em caso de interrupção no terminal receptor. Com o fluentbit, você pode usar o Amazon Kinesis Data Firehose para manter temporariamente os dados, o que pode reduzir a chance de sobrecarregar seu local final de armazenamento de dados.

Workloads

As cargas de trabalho têm um impacto no tamanho que seu cluster pode escalar. As cargas de trabalho que usam APIs muito o Kubernetes limitarão a quantidade total de cargas de trabalho que você pode ter em um único cluster, mas há alguns padrões que você pode alterar para ajudar a reduzir a carga.

As cargas de trabalho em um cluster Kubernetes têm acesso a recursos que se integram à API do Kubernetes (por exemplo, segredos e ServiceAccounts), mas esses recursos nem sempre são necessários e devem ser desativados se não estiverem sendo usados. Limitar o acesso à carga de trabalho e a dependência do plano de controle do Kubernetes aumentará o número de cargas de trabalho que você pode executar no cluster e melhorará a segurança de seus clusters removendo o

acesso desnecessário às cargas de trabalho e implementando práticas de privilégios mínimos. Leia as melhores práticas de segurança para obter mais informações.

Uso IPv6 para rede de pods

Você não pode fazer a transição de uma VPC de IPv4 para, IPv6 portanto, habilitar IPv6 antes de provisionar um cluster é importante. Se você habilitar IPv6 em uma VPC, isso não significa que você precisa usá-la. Se seus pods e serviços usarem, IPv6 você ainda poderá rotear o tráfego de e para endereços. IPv4 Consulte as melhores práticas de rede EKS para obter mais informações.

O uso IPv6 em seu cluster evita alguns dos limites mais comuns de escalabilidade de clusters e cargas de trabalho. IPv6 evita o esgotamento do endereço IP quando pods e nós não podem ser criados porque nenhum endereço IP está disponível. Ele também tem melhorias no desempenho por nó porque os pods recebem endereços IP mais rapidamente ao reduzir o número de anexos ENI por nó. Você pode obter um desempenho de nó semelhante usando o modo de IPv4 prefixo na VPC CNI, mas ainda precisa se certificar de que tem endereços IP suficientes disponíveis na VPC.

Limitar o número de serviços por namespace

O número máximo de serviços em um namespace é 5.000 e o número máximo de serviços em um cluster é 10.000. Para ajudar a organizar cargas de trabalho e serviços, aumentar o desempenho e evitar o impacto em cascata dos recursos com escopo de namespace, recomendamos limitar o número de serviços por namespace a 500.

O número de regras de tabelas IP criadas por nó com o kube-proxy cresce com o número total de serviços no cluster. A geração de milhares de regras de tabelas IP e o roteamento de pacotes por meio dessas regras têm um impacto no desempenho dos nós e aumentam a latência da rede.

Crie namespaces Kubernetes que englobem um único ambiente de aplicativo, desde que o número de serviços por namespace seja inferior a 500. Isso manterá a descoberta de serviços pequena o suficiente para evitar limites de descoberta de serviços e também pode ajudá-lo a evitar colisões de nomenclatura de serviços. Ambientes de aplicativos (por exemplo, dev, test, prod) devem usar clusters EKS separados em vez de namespaces.

Entenda as cotas do Elastic Load Balancer

Ao criar seus serviços, considere o tipo de balanceamento de carga que você usará (por exemplo, Network Load Balancer (NLB) ou Application Load Balancer (ALB)). Cada tipo de balanceador de carga fornece funcionalidades diferentes e tem cotas diferentes. Algumas das cotas padrão podem

ser ajustadas, mas há algumas cotas máximas que não podem ser alteradas. Para ver as cotas e o uso da sua conta, acesse o painel Service Quotas no console da AWS.

Por exemplo, os alvos padrão do ALB são 1000. Se você tiver um serviço com mais de 1.000 endpoints, precisará aumentar a cota, dividir o serviço em vários ALBs ou usar o Kubernetes Ingress. Os alvos padrão do NLB são 3000, mas estão limitados a 500 alvos por AZ. Se seu cluster executar mais de 500 pods para um serviço NLB, você precisará usar vários AZs ou solicitar um aumento do limite de cota.

Uma alternativa ao uso de um balanceador de carga acoplado a um serviço é usar um controlador de entrada. O controlador do AWS Load Balancer pode criar recursos ALBs de entrada, mas você pode considerar a execução de um controlador dedicado em seu cluster. Um controlador de entrada no cluster permite que você exponha vários serviços do Kubernetes a partir de um único balanceador de carga executando um proxy reverso dentro do seu cluster. Os controladores têm recursos diferentes, como suporte para a API do Gateway, que pode ter benefícios dependendo de quantas e do tamanho de suas cargas de trabalho.

Use o Route 53, o Global Accelerator ou CloudFront

Para disponibilizar um serviço usando vários balanceadores de carga como um único endpoint, você precisa usar o Amazon CloudFront, o AWS Global Accelerator ou o Amazon Route 53 para expor todos os balanceadores de carga como um único endpoint voltado para o cliente. Cada opção tem benefícios diferentes e pode ser usada separadamente ou em conjunto, dependendo de suas necessidades.

O Route 53 pode expor vários balanceadores de carga com um nome comum e pode enviar tráfego para cada um deles com base no peso atribuído. Você pode ler mais sobre os pesos do DNS na documentação e ler como implementá-los com o controlador DNS externo do Kubernetes na documentação do AWS Load Balancer Controller.

O Global Accelerator pode rotear cargas de trabalho para a região mais próxima com base no endereço IP solicitado. Isso pode ser útil para cargas de trabalho implantadas em várias regiões, mas não melhora o roteamento para um único cluster em uma única região. Usar o Route 53 em combinação com o Global Accelerator tem benefícios adicionais, como verificação de integridade e failover automático se um AZ não estiver disponível. Você pode ver um exemplo do uso do Global Accelerator com o Route 53 nesta postagem do blog.

CloudFront pode ser usado com o Route 53 e o Global Accelerator ou sozinho para rotear o tráfego para vários destinos. CloudFront armazena em cache os ativos que estão sendo servidos pelas

fontes de origem, o que pode reduzir os requisitos de largura de banda, dependendo do que você está servindo.

Use EndpointSlices em vez de endpoints

Ao descobrir pods que correspondem a uma etiqueta de serviço, você deve usar EndpointSlicesem vez de endpoints. Os endpoints eram uma maneira simples de expor serviços em pequenas escalas, mas serviços grandes que escalam automaticamente ou têm atualizações causam muito tráfego no plano de controle do Kubernetes. EndpointSlices têm agrupamento automático que permite coisas como dicas com reconhecimento de topologia.

Nem todos os controladores usam EndpointSlices por padrão. Você deve verificar as configurações do controle e ativá-lo, se necessário. Para o AWS Load Balancer Controller, você deve habilitar o uso da bandeira `--enable-endpoint-slices` opcional. EndpointSlices

Use segredos imutáveis e externos, se possível

O kubelet mantém um cache das chaves e valores atuais dos segredos que são usados em volumes para pods nesse nó. O kubelet monitora os segredos para detectar alterações. À medida que o cluster se expande, o número crescente de relógios pode afetar negativamente o desempenho do servidor da API.

Existem duas estratégias para reduzir o número de relógios em Secrets:

- Para aplicativos que não precisam de acesso aos recursos do Kubernetes, você pode desativar a montagem automática de segredos da conta de serviço definindo Token: false automountServiceAccount
- Se os segredos do seu aplicativo forem estáticos e não forem modificados no futuro, marque-os como imutáveis. O kubelet não mantém um monitor de API para segredos imutáveis.

Para desativar a montagem automática de uma conta de serviço em pods, você pode usar a seguinte configuração em sua carga de trabalho. Você pode substituir essas configurações se cargas de trabalho específicas precisarem de uma conta de serviço.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app
automountServiceAccountToken: true
```

Monitore o número de segredos no cluster antes que ele exceda o limite de 10.000. Você pode ver uma contagem total de segredos em um cluster com o comando a seguir. Você deve monitorar esse limite por meio de suas ferramentas de monitoramento de cluster.

```
kubectl get secrets -A | wc -l
```

Você deve configurar o monitoramento para alertar o administrador do cluster antes que esse limite seja atingido. Considere usar opções externas de gerenciamento de segredos, como o AWS Key Management Service (AWS KMS) ou o Hashicorp Vault com o driver CSI da Secrets Store.

Limitar o histórico de implantação

Os pods podem ser lentos ao criar, atualizar ou excluir porque objetos antigos ainda são rastreados no cluster. Você pode reduzir o número `revisionHistoryLimit` de implantações para limpar mais antigas, ReplicaSets o que reduzirá a quantidade total de objetos rastreados pelo Kubernetes Controller Manager. O limite de histórico padrão para implantações em 10.

Se o cluster criar muitos objetos de trabalho por meio CronJobs de outros mecanismos, use a `ttlSecondsAfterFinished`configuração para limpar automaticamente os pods antigos no cluster. Isso removerá os trabalhos executados com sucesso do histórico de trabalhos após um período de tempo especificado.

Desativar enableServiceLinks por padrão

Quando um pod é executado em um Node, o kubelet adiciona um conjunto de variáveis de ambiente para cada serviço ativo. Os processos Linux têm um tamanho máximo para seu ambiente, que pode ser alcançado se você tiver muitos serviços em seu namespace. O número de serviços por namespace não deve exceder 5.000. Depois disso, o número de variáveis do ambiente de serviço ultrapassa os limites do shell, fazendo com que os pods falhem na inicialização.

Há outros motivos pelos quais os pods não devem usar variáveis de ambiente de serviço para descoberta de serviços. Conflitos de nomes de variáveis de ambiente, nomes de serviços vazados e tamanho total do ambiente são alguns exemplos. Você deve usar o CoreDNS para descobrir endpoints de serviço.

Limite os webhooks de admissão dinâmica por recurso

Os webhooks de admissão dinâmica incluem webhooks de admissão e webhooks mutantes. Eles são endpoints de API que não fazem parte do plano de controle do Kubernetes e são chamados em

sequência quando um recurso é enviado para a API do Kubernetes. Cada webhook tem um tempo limite padrão de 10 segundos e pode aumentar a quantidade de tempo que uma solicitação de API leva se você tiver vários webhooks ou se algum deles atingir o tempo limite.

Certifique-se de que seus webhooks estejam altamente disponíveis, especialmente durante um incidente de AZ, e que a FailurePolicy esteja configurada corretamente para rejeitar o recurso ou ignorar a falha. Não chame webhooks quando não for necessário, permitindo que os comandos --dry-run kubectl ignorem o webhook.

```
apiVersion: admission.k8s.io/v1
kind: AdmissionReview
request:
  dryRun: False
```

Webhooks mutantes podem modificar recursos em sucessão frequente. Se você tiver 5 webhooks mutantes e implantar 50 recursos, o etcd armazenará todas as versões de cada recurso até que a compactação seja executada — a cada 5 minutos — para remover as versões antigas dos recursos modificados. Nesse cenário, quando o etcd remover recursos substituídos, haverá 200 versões de recursos removidas do etcd e, dependendo do tamanho dos recursos, poderão usar um espaço considerável no host etcd até que a desfragmentação seja executada a cada 15 minutos.

Essa desfragmentação pode causar pausas no etcd, o que pode ter outros efeitos na API e nos controladores do Kubernetes. Você deve evitar a modificação frequente de grandes recursos ou a modificação de centenas de recursos em rápida sucessão.

Compare cargas de trabalho em vários clusters

Se você tiver dois clusters que deveriam ter desempenho semelhante, mas não têm, tente comparar as métricas para identificar o motivo.

Por exemplo, comparar a latência do cluster é um problema comum. Isso geralmente é causado pela diferença no volume de solicitações de API. Você pode executar a CloudWatch LogInsight consulta a seguir para entender a diferença.

```
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
```

Você pode adicionar filtros adicionais para restringi-lo. por exemplo, focando em todas as solicitações de lista de`foo`.

```
filter @logStream like "kube-apiserver-audit"
| filter verb = "list"
| filter user.username like "foo"
| stats count(*) as cnt by objectRef.apiGroup, objectRef.apiVersion,
 objectRef.resource, responseStatus.code
| sort cnt desc
| limit 1000
```

Teoria de escalabilidade do Kubernetes

Nodes versus taxa de rotatividade

Muitas vezes, quando discutimos a escalabilidade do Kubernetes, fazemos isso em termos de quantos nós existem em um único cluster. Curiosamente, essa raramente é a métrica mais útil para entender a escalabilidade. Por exemplo, um cluster de 5.000 nós com um número grande, mas fixo, de pods não colocaria muito estresse no plano de controle após a configuração inicial. No entanto, se pegássemos um cluster de 1.000 nós e tentássemos criar 10.000 trabalhos de curta duração em menos de um minuto, isso colocaria uma grande pressão contínua no plano de controle.

O simples uso do número de nós para entender o dimensionamento pode ser enganoso. É melhor pensar em termos da taxa de mudança que ocorre em um período de tempo específico (vamos usar um intervalo de 5 minutos para essa discussão, pois é isso que as consultas do Prometheus normalmente usam por padrão). Vamos explorar por que enquadrar o problema em termos da taxa de mudança pode nos dar uma ideia melhor do que ajustar para alcançar a escala desejada.

Pensando em consultas por segundo

O Kubernetes tem vários mecanismos de proteção para cada componente — o Kubelet, o Scheduler, o Kube Controller Manager e o servidor de API — para evitar sobrecarregar o próximo elo na cadeia do Kubernetes. Por exemplo, o Kubelet tem um sinalizador para acelerar as chamadas para o servidor da API em uma determinada taxa. Esses mecanismos de proteção geralmente, mas nem sempre, são expressos em termos de consultas permitidas por segundo ou QPS.

É preciso ter muito cuidado ao alterar essas configurações de QPS. A remoção de um gargalo, como as consultas por segundo em um Kubelet, terá um impacto em outros componentes posteriores.

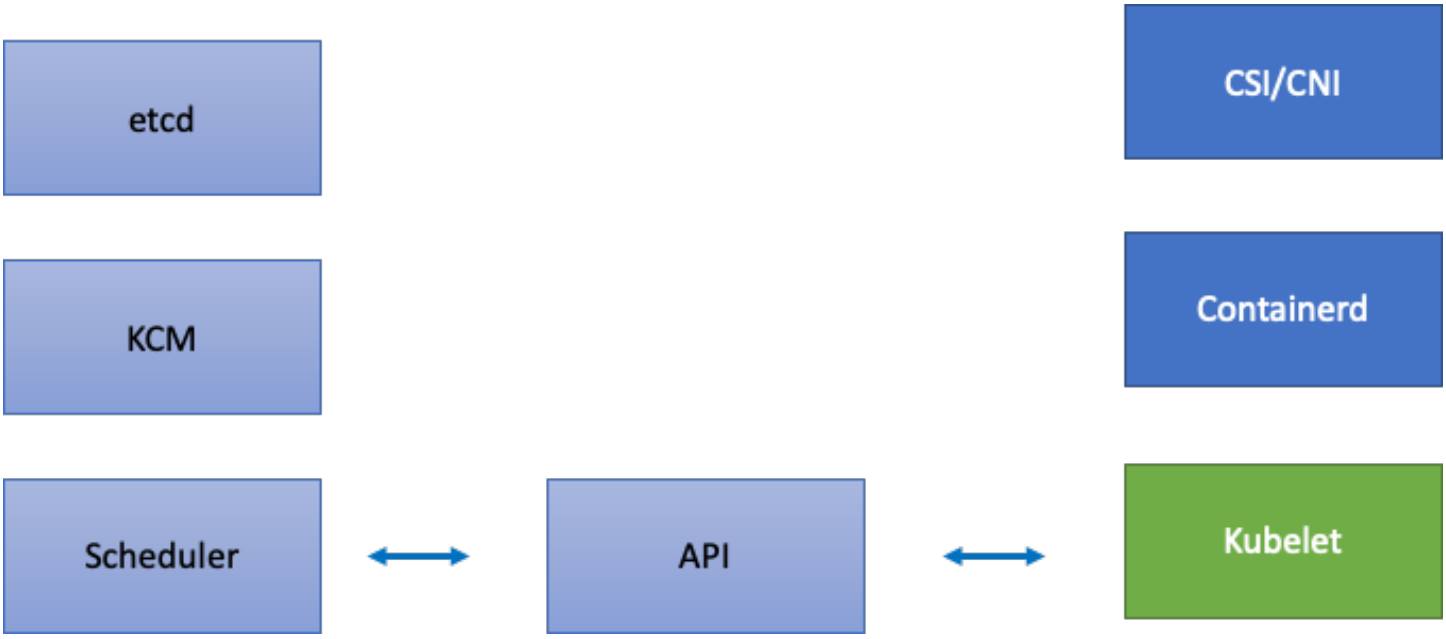
Isso pode e sobrecarregará o sistema acima de uma certa taxa, portanto, entender e monitorar cada parte da cadeia de serviços é fundamental para escalar com sucesso as cargas de trabalho no Kubernetes.

> **Note**
>
> O servidor da API tem um sistema mais complexo com a introdução da API Priority and Fairness, que discutiremos separadamente.

> **Note**
>
> Cuidado, algumas métricas parecem adequadas, mas na verdade estão medindo outra coisa. Por exemplo, `kubelet_http_inflight_requests` está relacionado apenas ao servidor de métricas no Kubelet, não ao número de solicitações do Kubelet para solicitações do apiserver. Isso pode fazer com que configuremos incorretamente o sinalizador QPS no Kubelet. Uma consulta nos registros de auditoria de um Kubelet específico seria uma maneira mais confiável de verificar as métricas.

Dimensionamento de componentes distribuídos

Como o EKS é um serviço gerenciado, vamos dividir os componentes do Kubernetes em duas categorias: componentes gerenciados pela AWS, que incluem etcd, Kube Controller Manager e Scheduler (na parte esquerda do diagrama), e componentes configuráveis pelo cliente, como o Kubelet, o Container Runtime e os vários operadores que chamam a AWS, APIs como os drivers de rede e armazenamento (na parte direita do diagrama). Deixamos o servidor da API no meio, mesmo que ele seja gerenciado pela AWS, pois as configurações de prioridade e justiça da API podem ser definidas pelos clientes.



Gargalos a montante e a jusante

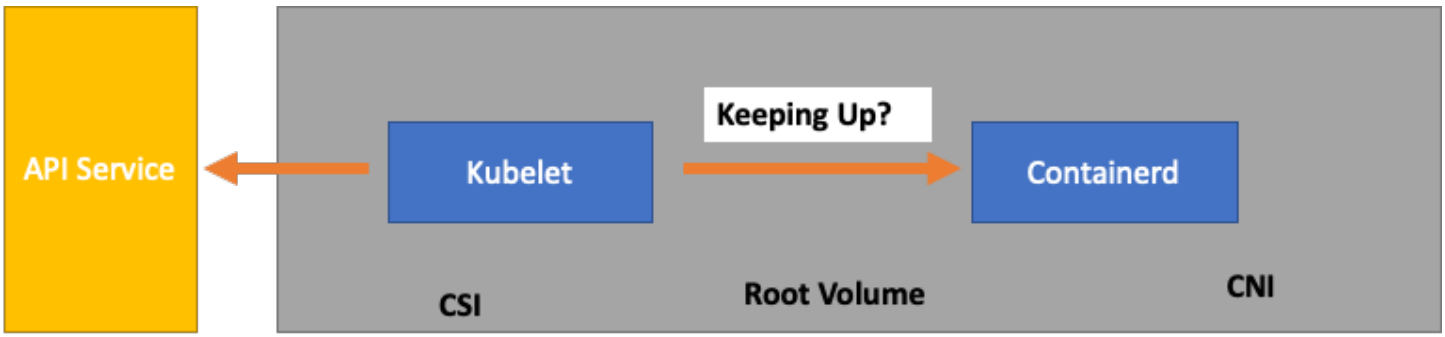
À medida que monitoramos cada serviço, é importante analisar as métricas em ambas as direções para identificar gargalos. Vamos aprender como fazer isso usando o Kubelet como exemplo. O Kubelet fala tanto com o servidor da API quanto com o tempo de execução do contêiner; como e o que precisamos monitorar para detectar se algum componente está enfrentando algum problema?

Quantos pods por Node

Quando analisamos os números de escala, como quantos pods podem ser executados em um nó, podemos considerar os 110 pods por nó que o upstream suporta pelo valor nominal.

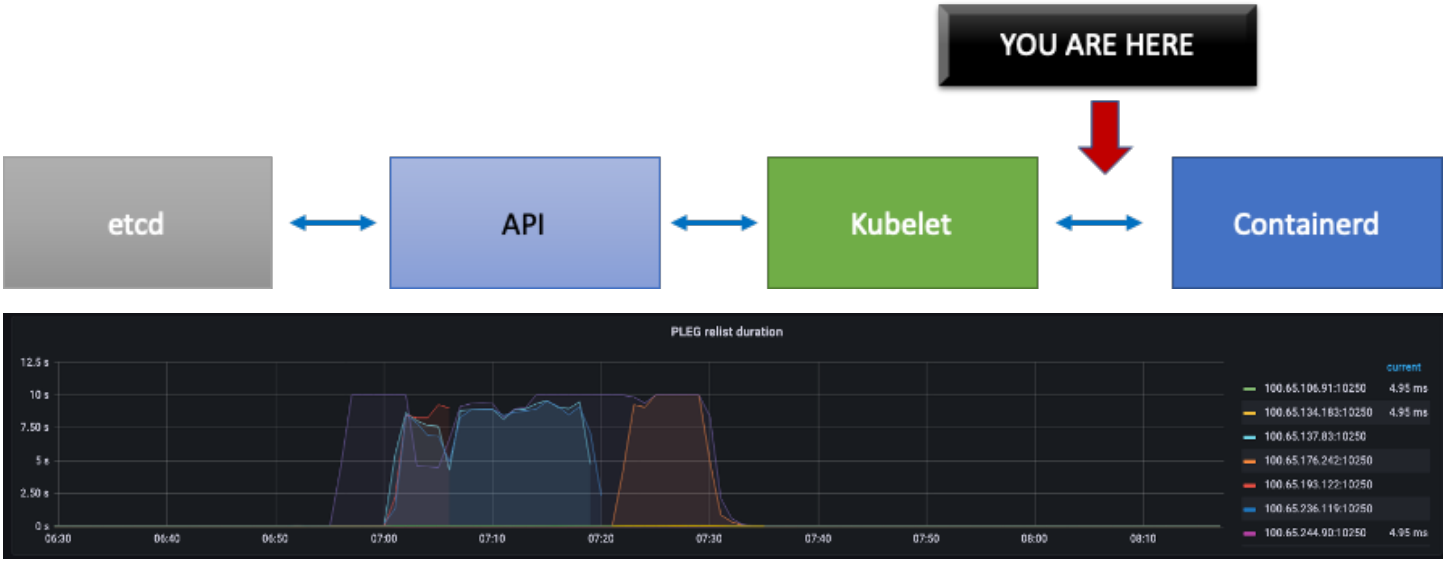
> **Note**
>
> https://kubernetes. io/docs/setup/best-practices/cluster-grande/

No entanto, sua carga de trabalho provavelmente é mais complexa do que a que foi testada em um teste de escalabilidade no Upstream. Para garantir que possamos atender ao número de pods que queremos executar na produção, vamos garantir que o Kubelet esteja “acompanhando” o tempo de execução do Containerd.



Para simplificar demais, o Kubelet está obtendo o status dos pods a partir do tempo de execução do contêiner (no nosso caso, Containerd). E se tivéssemos muitos pods mudando de status muito rapidamente? Se a taxa de alteração for muito alta, as solicitações [para o tempo de execução do contêiner] podem expirar.

> **Note**
>
> O Kubernetes está em constante evolução. Atualmente, esse subsistema está passando por mudanças. https://github.com/kubernetes/aprimoramentos/problemas/3386



No gráfico acima, vemos uma linha plana indicando que acabamos de atingir o valor de tempo limite da métrica de duração da geração de eventos do ciclo de vida do pod. Se você quiser ver isso em seu próprio cluster, você pode usar a seguinte sintaxe do PromQL.

```
increase(kubelet_pleg_relist_duration_seconds_bucket{instance="$instance"}
[$__rate_interval])
```

Se testemunharmos esse comportamento de tempo limite, sabemos que empurramos o nó para além do limite de que ele era capaz. Precisamos corrigir a causa do tempo limite antes de prosseguir. Isso pode ser conseguido reduzindo o número de pods por nó ou procurando erros que possam estar causando um alto volume de novas tentativas (afetando assim a taxa de rotatividade). A conclusão importante é que as métricas são a melhor maneira de entender se um nó é capaz de lidar com a taxa de rotatividade dos pods atribuídos versus usar um número fixo.

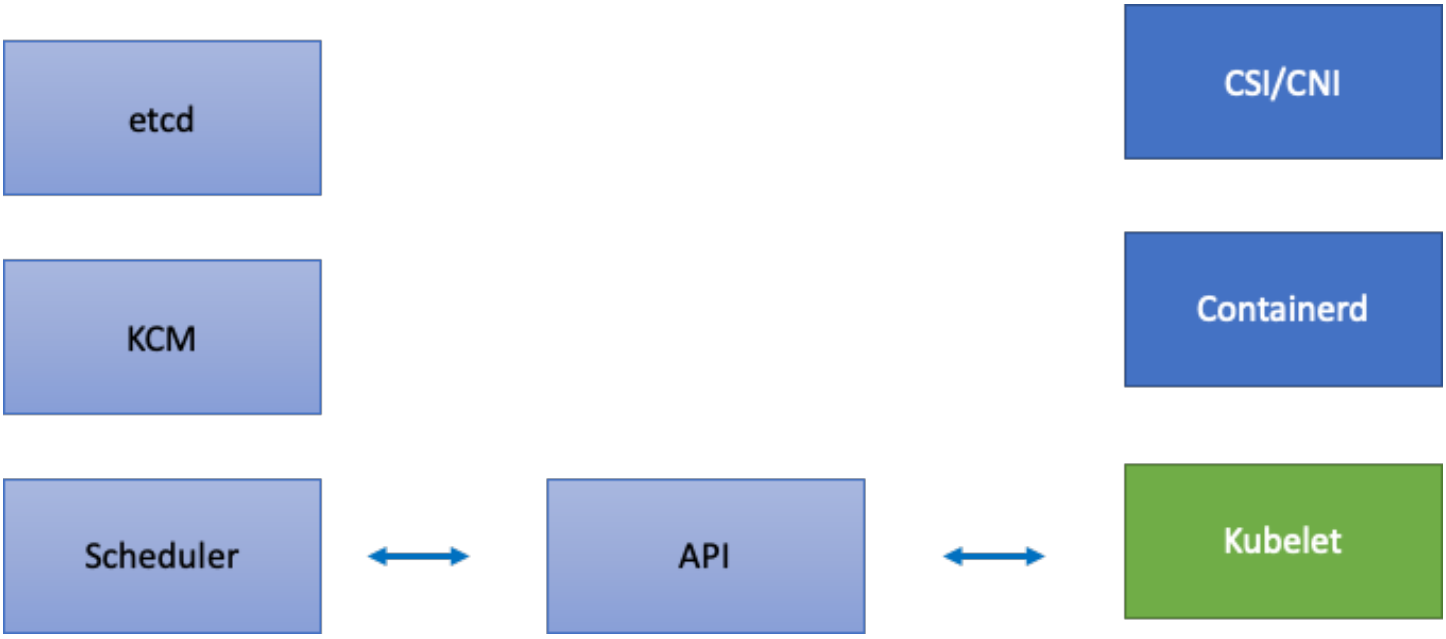
Dimensionar por métricas

Embora o conceito de usar métricas para otimizar sistemas seja antigo, ele geralmente é esquecido quando as pessoas iniciam sua jornada no Kubernetes. Em vez de focar em números específicos (ou seja, 110 pods por nó), concentramos nossos esforços em encontrar as métricas que nos ajudam a encontrar gargalos em nosso sistema. Compreender os limites corretos para essas métricas pode nos dar um alto grau de confiança de que nosso sistema está configurado de maneira ideal.

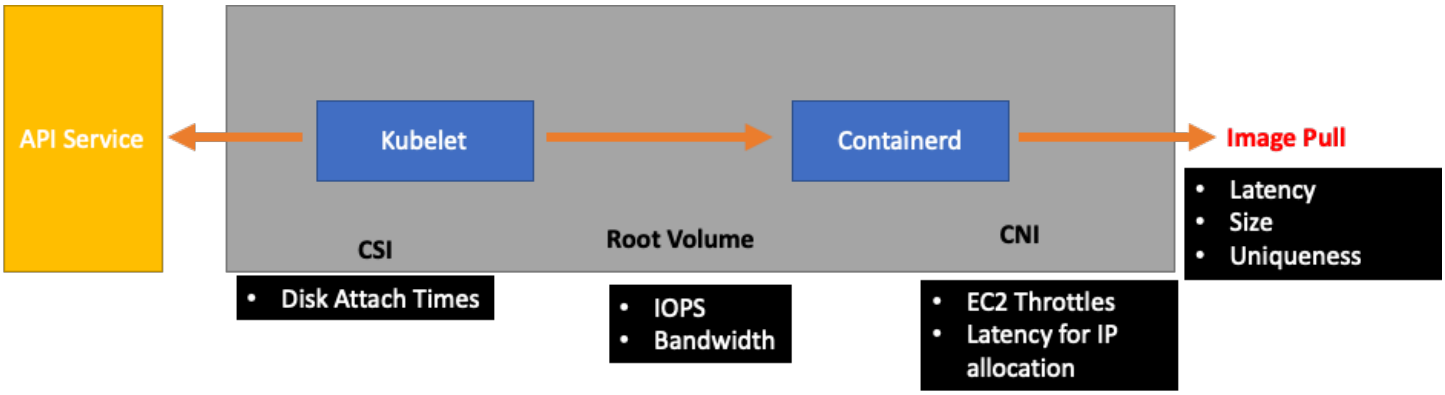
O impacto das mudanças

Um padrão comum que pode nos causar problemas é focar na primeira métrica ou erro de registro que pareça suspeito. Quando vimos que o Kubelet estava atingindo o tempo limite mais cedo, pudemos tentar coisas aleatórias, como aumentar a taxa por segundo que o Kubelet tem permissão para enviar, etc. No entanto, é aconselhável analisar o quadro geral de tudo o que está relacionado ao erro que encontramos primeiro. Faça cada alteração com um propósito e apoiada por dados.

A jusante do Kubelet estaria o tempo de execução do Containerd (erros de pod), DaemonSets como o driver de armazenamento (CSI) e o driver de rede (CNI) que se comunicam com a API, etc. EC2



Vamos continuar nosso exemplo anterior de que o Kubelet não acompanha o tempo de execução. Há vários pontos em que podemos empacotar um nó com tanta densidade que ele desencadeie erros.



Ao projetar o tamanho certo do nó para nossas cargas de trabalho, esses são easy-to-overlook sinais que podem estar colocando uma pressão desnecessária no sistema, limitando nossa escala e desempenho.

O custo de erros desnecessários

Os controladores Kubernetes são excelentes em tentar novamente quando surgem condições de erro, mas isso tem um custo. Essas novas tentativas podem aumentar a pressão sobre componentes como o Kube Controller Manager. É um importante fator do teste de escala monitorar esses erros.

Quando menos erros estão ocorrendo, é mais fácil identificar problemas no sistema. Ao garantir periodicamente que nossos clusters estejam livres de erros antes de grandes operações (como atualizações), podemos simplificar os registros de solução de problemas quando eventos imprevistos acontecem.

Expandindo nossa visão

Em clusters de grande escala com milhares de nós, não queremos procurar gargalos individualmente. No PromQL, podemos encontrar os valores mais altos em um conjunto de dados usando uma função chamada topk; sendo K uma variável, colocamos o número de itens que queremos. Aqui, usamos três nós para ter uma ideia se todos os Kubelets no cluster estão saturados. Estamos analisando a latência até agora, agora vamos ver se o Kubelet está descartando eventos.

```
topk(3, increase(kubelet_pleg_discard_events{}[$__rate_interval]))
```

Detalhando essa declaração.

- Usamos a variável Grafana `$__rate_interval` para garantir que ela receba as quatro amostras necessárias. Isso ignora um tópico complexo no monitoramento com uma variável simples.
- `topk`nos dê apenas os melhores resultados e o número 3 limita esses resultados a três. Essa é uma função útil para métricas em todo o cluster.
- `{}`diga-nos que não há filtros; normalmente, você colocaria o nome do trabalho de qualquer regra de raspagem; no entanto, como esses nomes variam, deixaremos em branco.

Dividindo o problema pela metade

Para resolver um gargalo no sistema, usaremos a abordagem de encontrar uma métrica que mostre que há um problema a montante ou a jusante, pois isso nos permite dividir o problema pela metade. Também será um princípio fundamental de como exibimos nossos dados de métricas.

Um bom lugar para começar com esse processo é o servidor da API, pois ele nos permite ver se há algum problema com um aplicativo cliente ou com o plano de controle.

Monitoramento do plano de controle

Servidor da API

Ao analisar nosso servidor de API, é importante lembrar que uma de suas funções é acelerar as solicitações de entrada para evitar sobrecarregar o plano de controle. O que pode parecer um gargalo no nível do servidor da API pode, na verdade, protegê-lo de problemas mais sérios. Precisamos considerar os prós e os contras de aumentar o volume de solicitações que circulam pelo sistema. Para determinar se os valores do servidor de API devem ser aumentados, aqui está uma pequena amostra do que precisamos considerar:

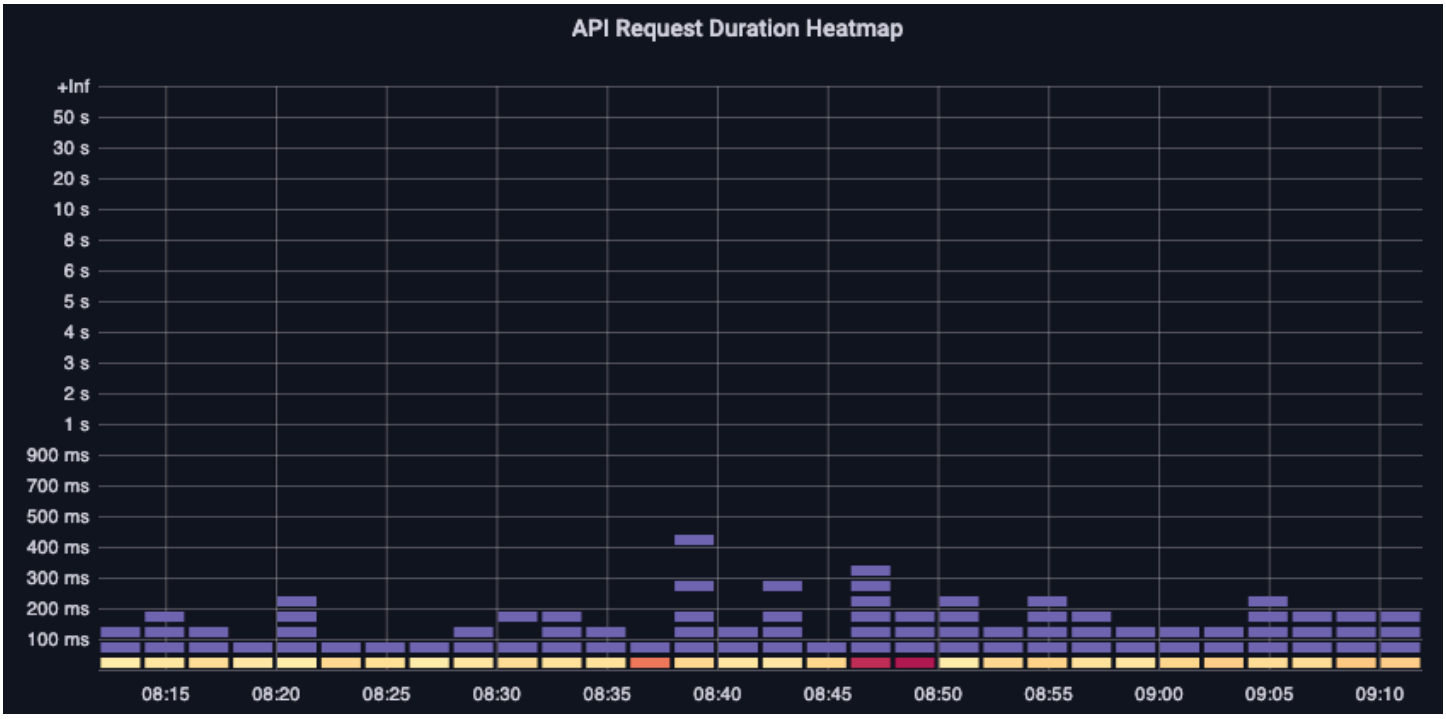
1. Qual é a latência das solicitações que passam pelo sistema?
2. Essa latência é o próprio servidor da API ou algo “downstream”, como etcd?
3. A profundidade da fila do servidor de API é um fator nessa latência?
4. As filas de API Priority and Fairness (APF) estão configuradas corretamente para os padrões de chamada de API que queremos?

Onde está o problema?

Para começar, podemos usar a métrica de latência da API para nos dar uma ideia de quanto tempo o servidor da API leva para atender às solicitações. Vamos usar o mapa de calor PromQL e Grafana abaixo para exibir esses dados.

```
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 obter uma descrição detalhada sobre como monitorar o servidor da API com o painel da API usado neste artigo, consulte o blog a seguir



Essas solicitações estão todas abaixo da marca de um segundo, o que é uma boa indicação de que o plano de controle está tratando as solicitações em tempo hábil. Mas e se esse não fosse o caso?

O formato que estamos usando na duração da solicitação de API acima é um mapa de calor. O que é bom sobre o formato do mapa de calor é que ele nos informa o valor de tempo limite da API por padrão (60 segundos). No entanto, o que realmente precisamos saber é em qual limite esse valor deve ser motivo de preocupação antes de atingirmos o limite de tempo limite. Para uma orientação aproximada de quais são os limites aceitáveis, podemos usar o SLO upstream do Kubernetes, que pode ser encontrado aqui

> **Note**
>
> Percebeu a função max nessa declaração? Ao usar métricas que agregam vários servidores (por padrão, dois servidores de API no EKS), é importante não calcular a média desses servidores juntos.

Padrões de tráfego assimétricos

E se um servidor de API [pod] estivesse levemente carregado e o outro muito carregado? Se calculássemos a média desses dois números juntos, poderíamos interpretar mal o que estava acontecendo. Por exemplo, aqui temos três servidores de API, mas toda a carga está em um desses

servidores de API. Como regra, qualquer coisa que tenha vários servidores, como servidores etcd e API, deve ser dividida ao investir em problemas de escala e desempenho.



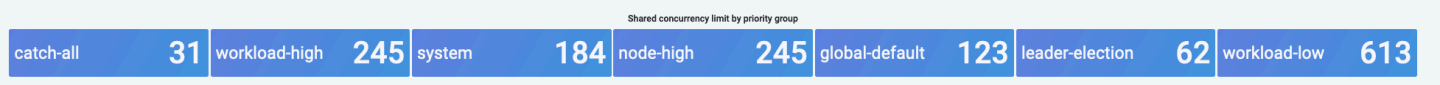
Com a mudança para a API Priority and Fairness, o número total de solicitações no sistema é apenas um fator a ser verificado para ver se o servidor da API está com excesso de assinaturas. Como o sistema agora funciona com uma série de filas, precisamos verificar se alguma dessas filas está cheia e se o tráfego dessa fila está diminuindo.

Vamos analisar essas filas com a seguinte consulta:

```
max without(instance)(apiserver_flowcontrol_nominal_limit_seats{})
```

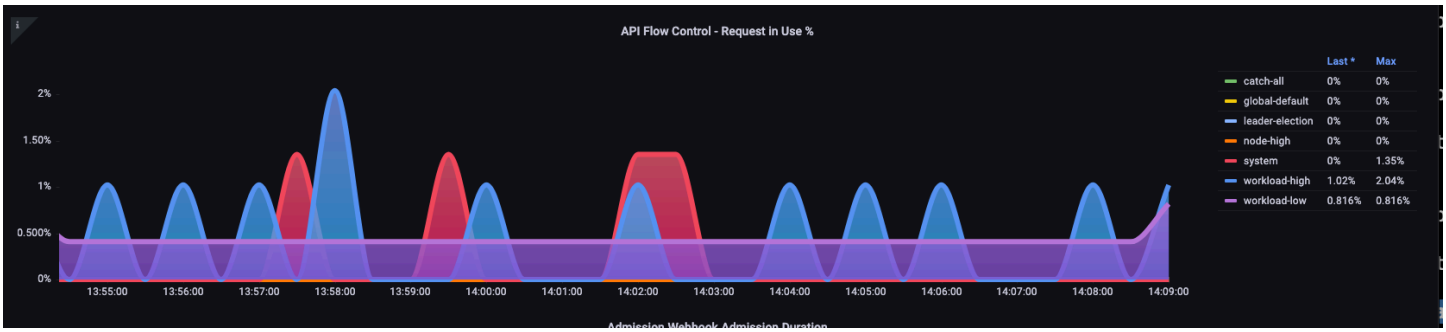
> **Note**
>
> Para obter mais informações sobre como a API A&F funciona, consulte o seguinte guia de melhores práticas

Aqui, vemos os sete grupos de prioridades diferentes que vêm por padrão no cluster.



Em seguida, queremos ver qual porcentagem desse grupo de prioridade está sendo usada, para que possamos entender se um determinado nível de prioridade está sendo saturado. Limitar as solicitações no nível de carga de trabalho baixa pode ser desejável, mas não seria desejável reduzir o nível de eleição de um líder.

O sistema API Priority and Fairness (APF) tem várias opções complexas. Algumas dessas opções podem ter consequências inesperadas. Um problema comum que vemos no campo é aumentar a profundidade da fila até o ponto em que ela começa a adicionar latência desnecessária. Podemos monitorar esse problema usando a `apiserver_flowcontrol_current_inqueue_request` métrica. Podemos verificar se há quedas usando `apiserver_flowcontrol_rejected_requests_total` o. Essas métricas terão um valor diferente de zero se algum bucket exceder sua simultaneidade.



Aumentar a profundidade da fila pode tornar o servidor de API uma fonte significativa de latência e deve ser feito com cuidado. Recomendamos ser criterioso com o número de filas criadas. Por exemplo, o número de compartilhamentos em um sistema EKS é 600. Se criarmos muitas filas, isso pode reduzir os compartilhamentos em filas importantes que precisam da taxa de transferência, como a fila de eleição do líder ou a fila do sistema. Criar muitas filas extras pode dificultar o tamanho correto dessas filas.

Para focar em uma mudança simples e impactante que você pode fazer no APF, simplesmente retiramos ações de compartimentos subutilizados e aumentamos o tamanho dos compartimentos que estão em seu uso máximo. Ao redistribuir de forma inteligente as ações entre esses compartimentos, você pode reduzir a probabilidade de quedas.

Para obter mais informações, visite as configurações de prioridade e imparcialidade da API no Guia de melhores práticas do EKS.

Latência de API versus etcd

Como podemos usar o metrics/logs do servidor de API para determinar se há um problema com o servidor de API, um problema upstream/downstream do servidor de API ou uma combinação de ambos. Para entender isso melhor, vamos ver como o API Server e o etcd podem ser relacionados e como é fácil solucionar problemas no sistema errado.

No gráfico abaixo, vemos a latência do servidor da API, mas também vemos que grande parte dessa latência está correlacionada ao servidor etcd devido às barras no gráfico que mostram a maior parte

da latência no nível etcd. Se houver 15 segundos de latência do etcd ao mesmo tempo em que houver 20 segundos de latência do servidor da API, a maior parte da latência estará, na verdade, no nível do etcd.

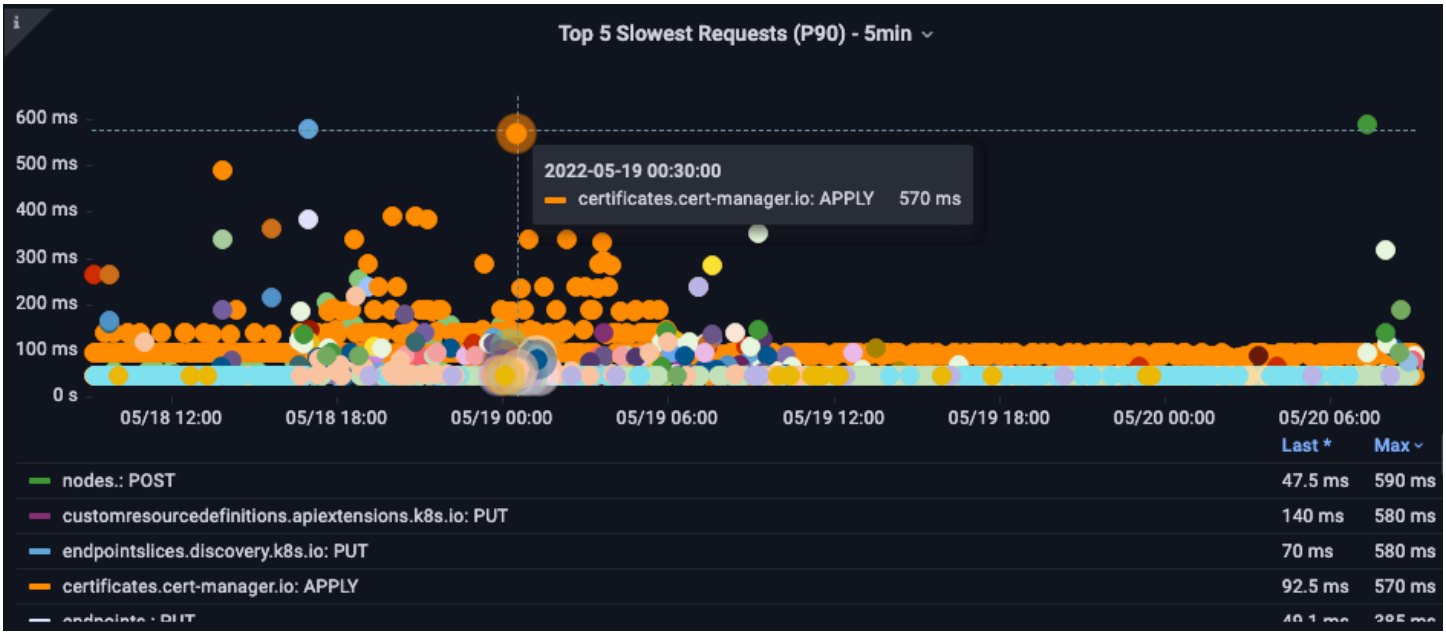
Ao analisar todo o fluxo, vemos que é aconselhável não focar apenas no servidor de API, mas também procurar sinais que indiquem que o etcd está sob pressão (ou seja, o aumento lento dos contadores de aplicação). Ser capaz de se mover rapidamente para a área problemática certa com apenas um olhar é o que torna um painel poderoso.

> **Note**
>
> O painel na seção pode ser encontrado em https://github.com/RiskyAdventure/Troubleshooting- Dashboards/blob/main/api -troubleshooter.json



Plano de controle versus problemas do lado do cliente

Neste gráfico, procuramos as chamadas de API que levaram mais tempo para serem concluídas nesse período. Nesse caso, vemos que um recurso personalizado (CRD) está chamando uma função APPLY que é a chamada mais latente durante o período de 05:40.



Com esses dados, podemos usar uma consulta Ad-Hoc PromQL ou CloudWatch Insights para extrair solicitações LIST do registro de auditoria durante esse período para ver qual aplicativo pode ser esse.

Encontrando a fonte com CloudWatch

As métricas são melhor usadas para encontrar a área problemática que queremos examinar e restringir o período de tempo e os parâmetros de pesquisa do problema. Assim que tivermos esses dados, queremos fazer a transição para registros para obter horários e erros mais detalhados. Para fazer isso, transformaremos nossos registros em métricas usando o CloudWatch Logs Insights.

Por exemplo, para investigar o problema acima, usaremos a seguinte consulta do CloudWatch Logs Insights para extrair o UserAgent e o RequestURI para que possamos identificar qual aplicativo está causando essa latência.

> **Note**
>
> Uma contagem apropriada precisa ser usada para evitar o List/Resync comportamento normal em um relógio.

```
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
```

Usando essa consulta, encontramos dois agentes diferentes executando um grande número de operações de lista de alta latência. Splunk e CloudWatch agente. Com os dados, podemos tomar a decisão de remover, atualizar ou substituir esse controlador por outro projeto.

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 obter mais detalhes sobre esse assunto, consulte o seguinte blog

Agendador

Como as instâncias do plano de controle do EKS são executadas em uma conta separada da AWS, não poderemos coletar esses componentes para obter métricas (o servidor da API é a exceção). No entanto, como temos acesso aos registros de auditoria desses componentes, podemos transformá-los em métricas para ver se algum dos subsistemas está causando um gargalo de escalabilidade. Vamos usar o CloudWatch Logs Insights para ver quantos pods não programados estão na fila do agendador.

Pods não programados no registro do agendador

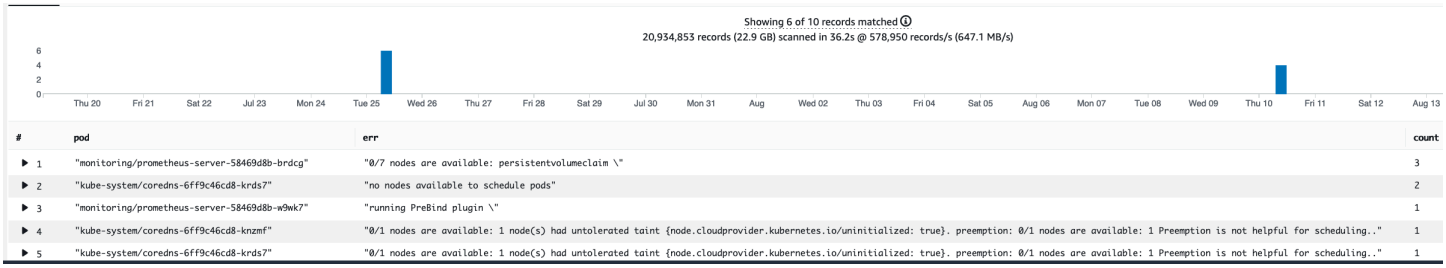
Se tivéssemos acesso para coletar as métricas do agendador diretamente em um Kubernetes autogerenciado (como o Kops), usaríamos o seguinte PromQL para entender o backlog do agendador.

```
max without(instance)(scheduler_pending_pods)
```

Como não temos acesso à métrica acima no EKS, usaremos a consulta CloudWatch Logs Insights abaixo para ver a lista de pendências verificando quantos pods não puderam ser cancelados durante um determinado período de tempo. Então, poderíamos nos aprofundar nas mensagens no horário de pico para entender a natureza do gargalo. Por exemplo, os nós que não estão girando rápido o suficiente ou o limitador de taxa no próprio 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
```

Aqui vemos os erros do programador dizendo que o pod não foi implantado porque o PVC de armazenamento não estava disponível.



#	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

O registro de auditoria deve ser ativado no plano de controle para ativar essa função. Também é uma prática recomendada limitar a retenção de registros para não aumentar os custos ao longo do tempo desnecessariamente. Um exemplo para ativar todas as funções de registro usando a ferramenta EKSCTL abaixo.

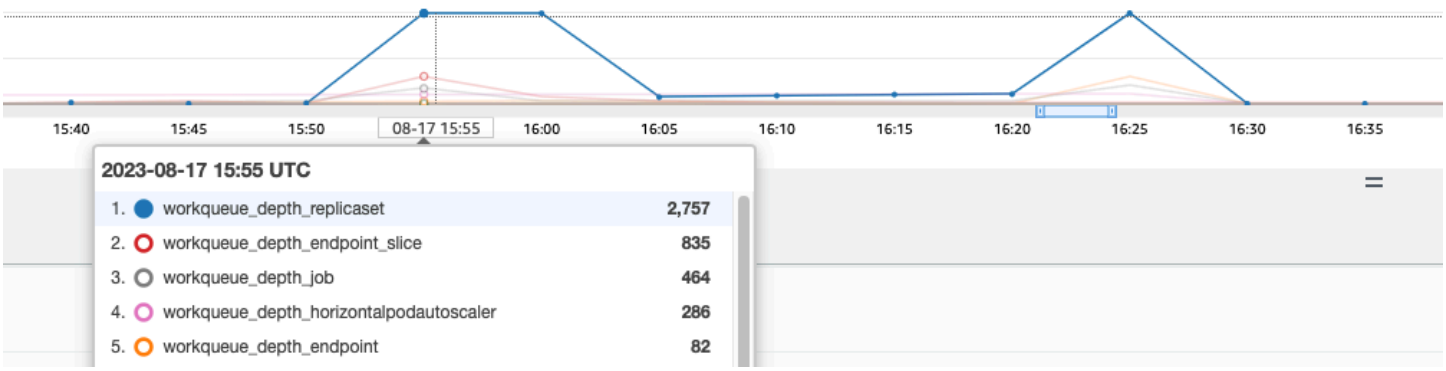
```
cloudWatch:
  clusterLogging:
    enableTypes: ["*"]
    logRetentionInDays: 10
```

Gerenciador de controladores Kube

O Kube Controller Manager, como todos os outros controladores, tem limites de quantas operações ele pode realizar ao mesmo tempo. Vamos analisar quais são alguns desses sinalizadores examinando uma configuração de KOPS na qual podemos definir esses parâmetros.

```
kubeControllerManager:
  concurrentEndpointSyncs: 5
  concurrentReplicasetSyncs: 5
  concurrentNamespaceSyncs: 10
  concurrentServiceaccountTokenSyncs: 5
  concurrentServiceSyncs: 5
  concurrentResourceQuotaSyncs: 5
  concurrentGcSyncs: 20
  kubeAPIBurst: 30
  kubeAPIQPS: "20"
```

Esses controladores têm filas que se enchem durante períodos de alta rotatividade em um cluster. Nesse caso, vemos que o controlador do conjunto replicaset tem uma grande lista de pendências em sua fila.



Temos duas maneiras diferentes de lidar com essa situação. Se executássemos o autogerenciamento, poderíamos simplesmente aumentar as goroutines simultâneas, no entanto, isso teria um impacto no etcd ao processar mais dados no KCM. A outra opção seria reduzir o número de objetos de replicaset usados `.spec.revisionHistoryLimit` na implantação para reduzir o número de objetos de replicaset que podemos reverter, reduzindo assim a pressão sobre esse controlador.

```
spec:
```

```
  revisionHistoryLimit: 2
```

Outros recursos do Kubernetes podem ser ajustados ou desativados para reduzir a pressão em sistemas com alta taxa de rotatividade. Por exemplo, se o aplicativo em nossos pods não precisar falar diretamente com a API k8s, desligar o segredo projetado nesses pods diminuiria a carga. ServiceaccountTokenSyncs Essa é a maneira mais desejável de resolver esses problemas, se possível.

```
kind: Pod
spec:
  automountServiceAccountToken: false
```

Em sistemas em que não temos acesso às métricas, podemos examinar novamente os registros para detectar contenção. Se quiséssemos ver o número de solicitações sendo processadas em um nível por controlador ou agregado, usaríamos a seguinte consulta do CloudWatch Logs Insights.

Volume total processado pelo 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
```

A principal conclusão aqui é, ao analisar os problemas de escalabilidade, examinar cada etapa do caminho (API, agendador, KCM, etc.) antes de passar para a fase detalhada de solução de problemas. Muitas vezes, na produção, você descobrirá que são necessários ajustes em mais de uma parte do Kubernetes para permitir que o sistema funcione com o máximo desempenho. É fácil solucionar inadvertidamente o que é apenas um sintoma (como o tempo limite do nó) de um gargalo muito maior.

ETC.

O etcd usa um arquivo mapeado na memória para armazenar pares de valores-chave de forma eficiente. Há um mecanismo de proteção para definir o tamanho desse espaço de memória disponível, geralmente definido nos limites de 2, 4 e 8 GB. Menos objetos no banco de dados significa menos limpeza que o etcd precisa fazer quando os objetos são atualizados e as versões mais antigas precisam ser limpas. Esse processo de limpeza de versões antigas de um objeto é chamado de compactação. Após várias operações de compactação, há um processo subsequente que recupera o espaço útil chamado desfragmentação, que ocorre acima de um determinado limite ou em um cronograma fixo.

Há alguns itens relacionados ao usuário que podemos fazer para limitar o número de objetos no Kubernetes e, assim, reduzir o impacto do processo de compactação e desfragmentação. Por exemplo, Helm mantém uma alta`revisionHistoryLimit`. Isso faz com que objetos mais antigos, como ReplicaSets no sistema, possam fazer reversões. Ao definir os limites do histórico em 2, podemos reduzir o número de objetos (como ReplicaSets) de dez para dois, o que, por sua vez, colocaria menos carga no sistema.

```
apiVersion: apps/v1
kind: Deployment
spec:
  revisionHistoryLimit: 2
```

Do ponto de vista do monitoramento, se os picos de latência do sistema ocorrerem em um padrão definido separado por horas, verificar se esse processo de desfragmentação é a origem pode ser útil. Podemos ver isso usando o CloudWatch Logs.

Se você quiser ver os start/end horários de desfragmentação, use a seguinte 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

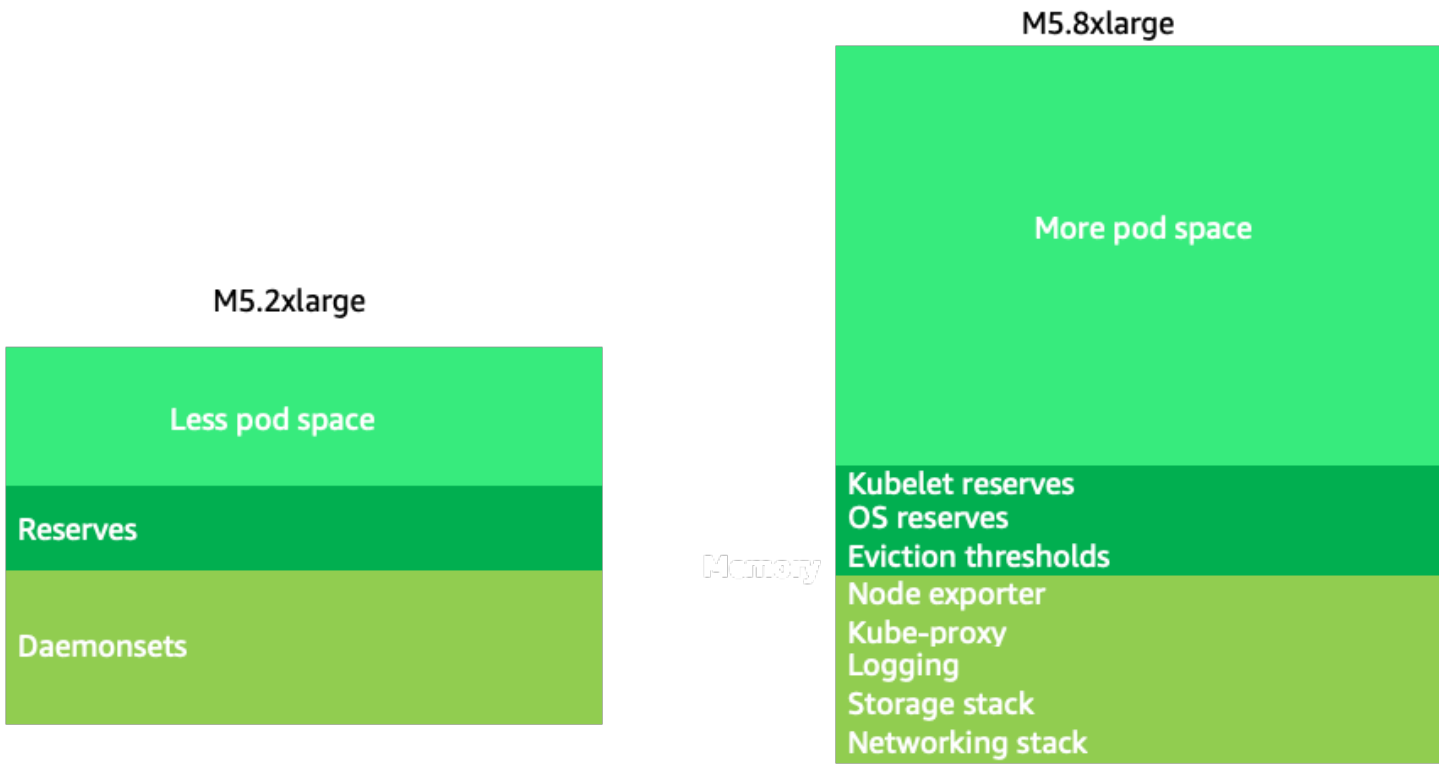
Eficiência do nó e da carga de trabalho

Ser eficiente com nossas cargas de trabalho e nós reduz complexity/cost ao mesmo tempo que aumenta o desempenho e a escala. Há muitos fatores a serem considerados ao planejar essa eficiência, e é mais fácil pensar em termos de compensações versus uma configuração de melhores práticas para cada recurso. Vamos explorar essas compensações em profundidade na seção a seguir.

Seleção de nós

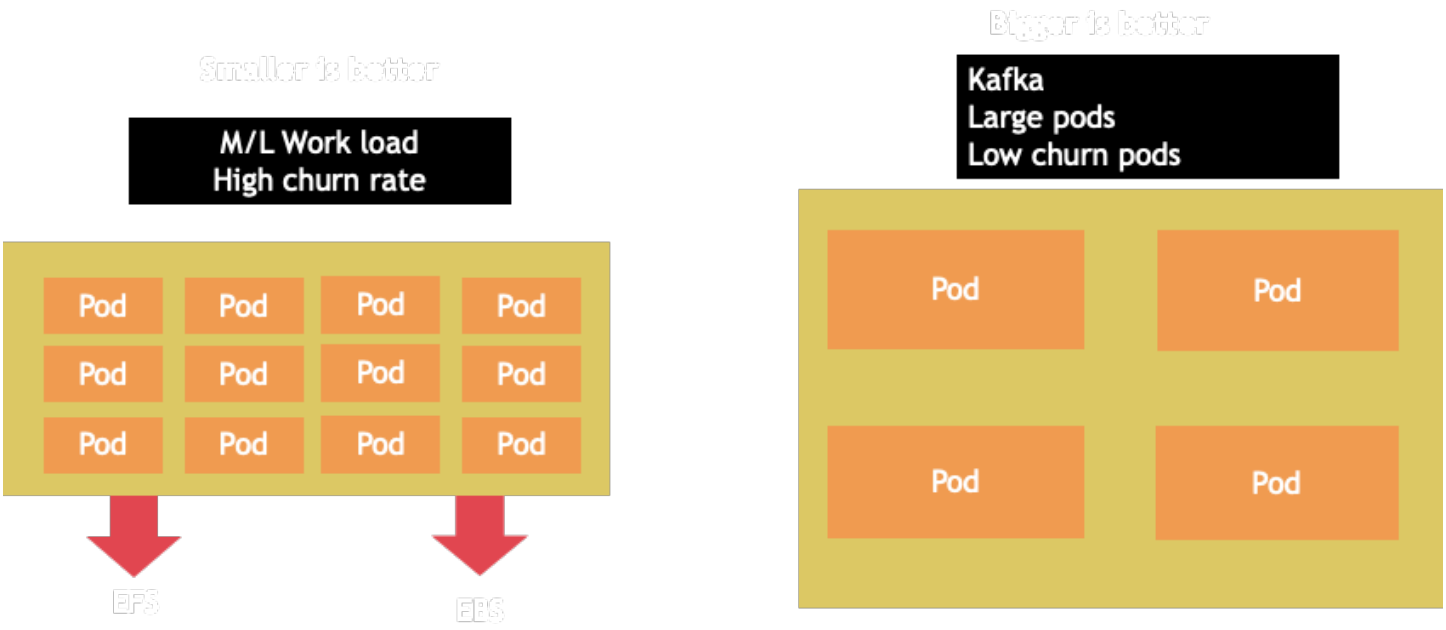
Usar tamanhos de nós um pouco maiores (4-12xlarge) aumenta o espaço disponível que temos para executar pods, pois reduz a porcentagem do nó usado para “sobrecarga”, como DaemonSetsreservas para componentes do sistema. No diagrama abaixo, vemos a diferença entre o espaço útil em um sistema 2xlarge versus 8xlarge com apenas um número moderado de. DaemonSets

> **Note**
>
> Como o k8s é dimensionado horizontalmente como regra geral, para a maioria dos aplicativos, não faz sentido considerar o impacto no desempenho dos nós de tamanhos NUMA, daí a recomendação de uma faixa abaixo desse tamanho de nó.



Tamanhos grandes de nós nos permitem ter uma porcentagem maior de espaço útil por nó. No entanto, esse modelo pode ser levado ao extremo ao empacotar o nó com tantos pods que causa erros ou satura o nó. O monitoramento da saturação dos nós é fundamental para o uso bem-sucedido de nós de tamanhos maiores.

A seleção de nós raramente é uma one-size-fits-all proposta. Geralmente, é melhor dividir as cargas de trabalho com taxas de rotatividade dramaticamente diferentes em grupos de nós diferentes. Cargas de trabalho em lotes pequenos com uma alta taxa de rotatividade seriam melhor atendidas pela família de instâncias de 4 vezes maior, enquanto um aplicativo de grande escala, como o Kafka, que usa 8 vCPUs e tem uma baixa taxa de rotatividade, seria melhor atendido pela família de 12 vezes maior.



> **Note**
>
> Outro fator a ser considerado com tamanhos de nós muito grandes é que os CGROUPS não ocultam o número total de vCPUs do aplicativo em contêineres. Os tempos de execução dinâmicos geralmente geram um número não intencional de threads de sistema operacional, criando uma latência difícil de solucionar. Para esses aplicativos, é recomendável fixar a CPU. Para uma exploração mais profunda do tópico, consulte o seguinte https:// www.youtube.com vídeo/assista? v = _ NqtfDy KAqg

Embalagem Node Bin

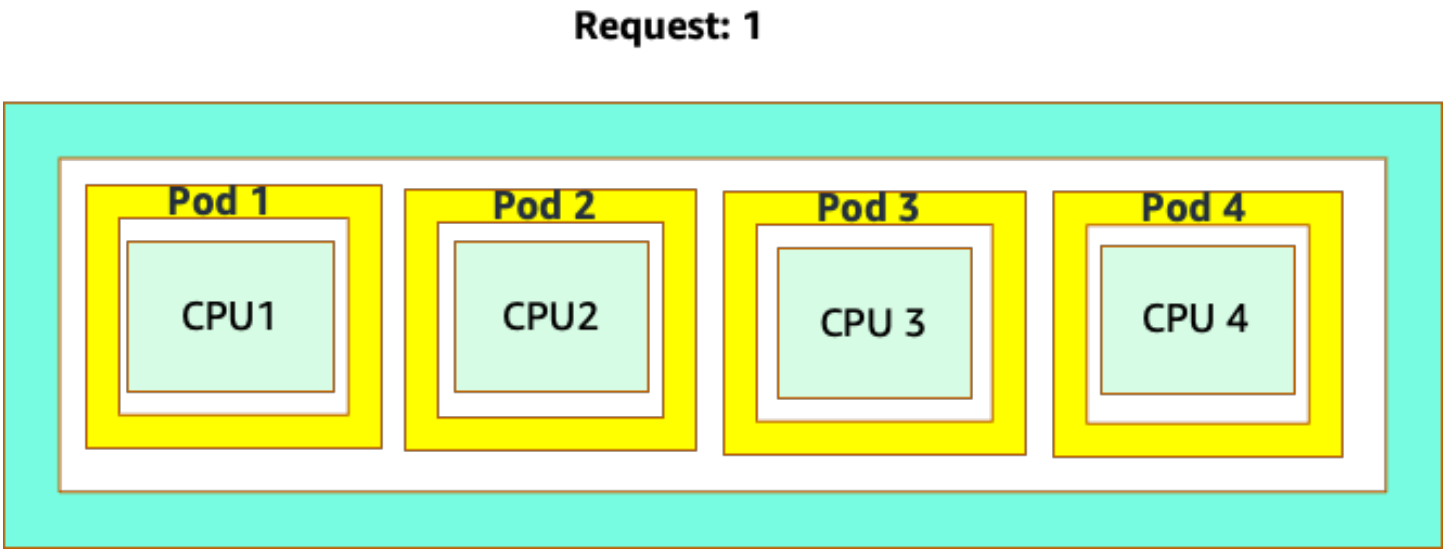
Regras do Kubernetes versus Linux

Há dois conjuntos de regras que precisamos considerar ao lidar com cargas de trabalho no Kubernetes. As regras do Kubernetes Scheduler, que usa o valor da solicitação para programar pods em um nó e, em seguida, o que acontece depois que o pod é agendado, que é o reino do Linux, não do Kubernetes.

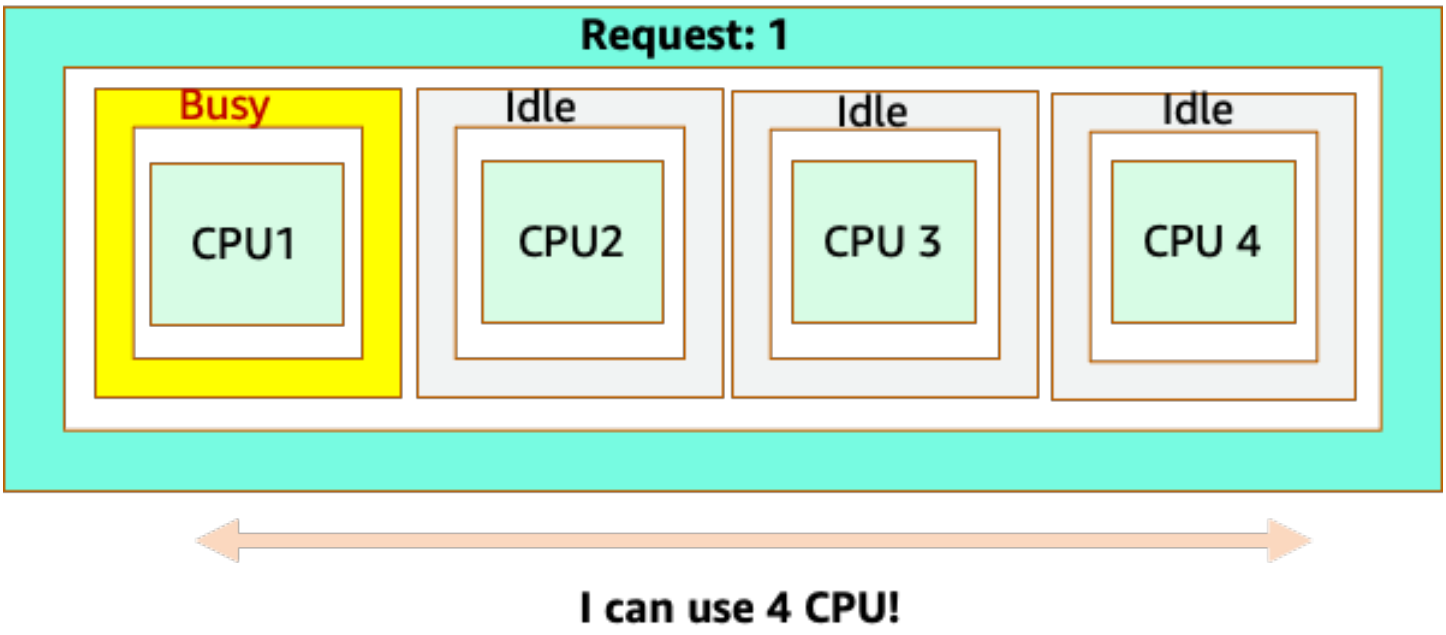
Depois que o agendador do Kubernetes é concluído, um novo conjunto de regras assume o controle, o Linux Completely Fair Scheduler (CFS). A principal conclusão é que o Linux CFS não tem o conceito de núcleo. Discutiremos por que pensar em núcleos pode causar grandes problemas na otimização de cargas de trabalho para escalabilidade.

Pensando em núcleos

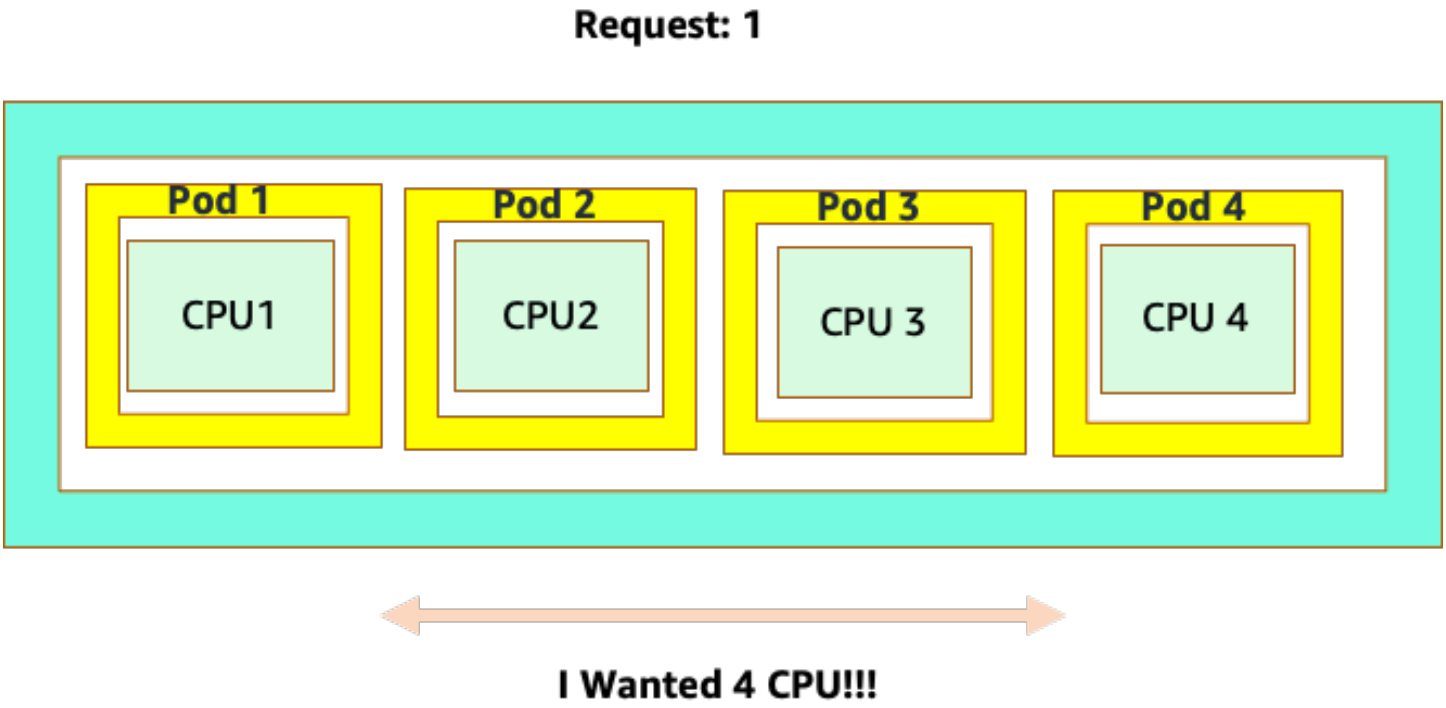
A confusão começa porque o programador do Kubernetes tem o conceito de núcleos. Do ponto de vista do programador do Kubernetes, se analisássemos um nó com 4 pods NGINX, cada um com uma solicitação de um conjunto principal, o nó ficaria assim.



No entanto, vamos fazer um experimento mental sobre como isso parece diferente do ponto de vista do CFS do Linux. A coisa mais importante a lembrar ao usar o sistema Linux CFS é: contêineres ocupados (CGROUPS) são os únicos contêineres que contam para o sistema de compartilhamento. Nesse caso, somente o primeiro contêiner está ocupado, portanto, é permitido usar todos os 4 núcleos no nó.



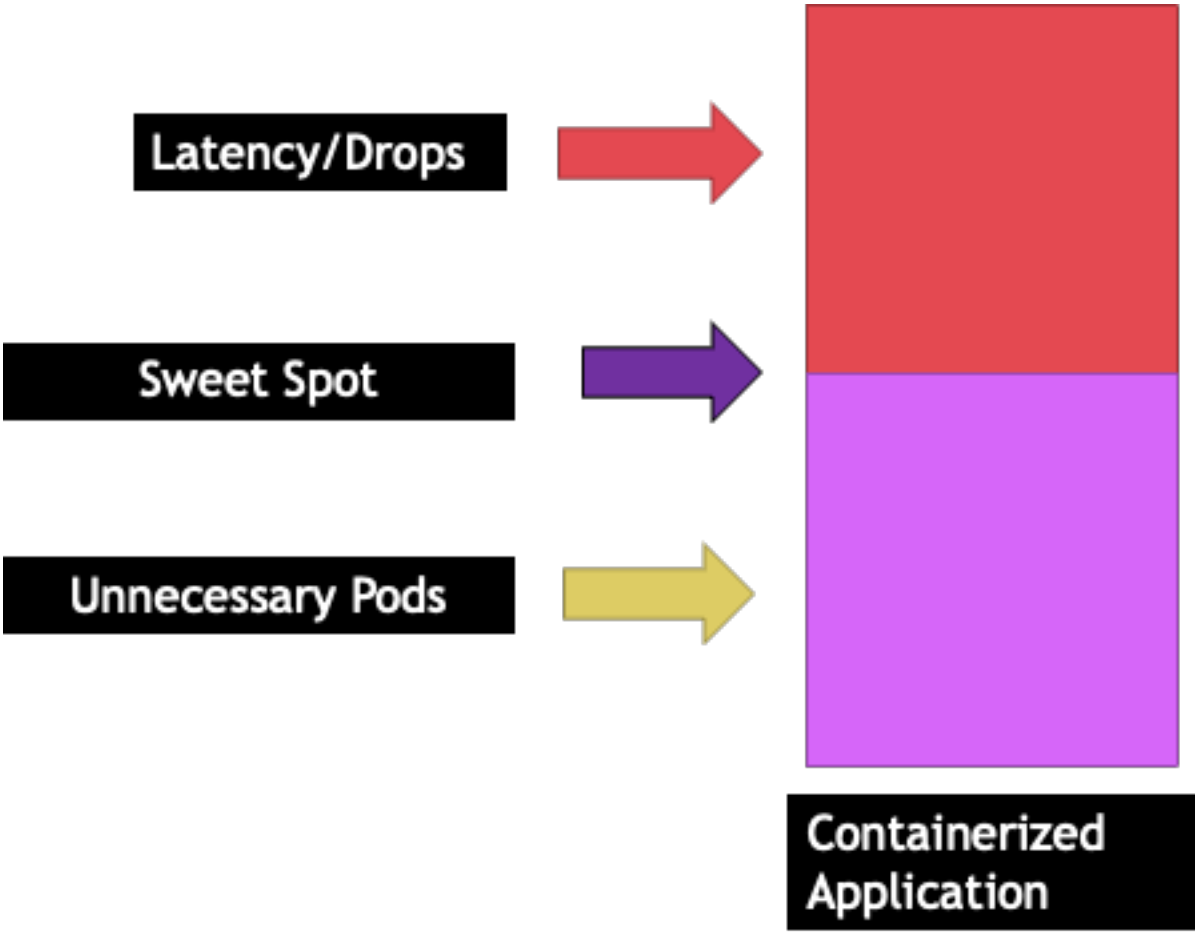
Por que isso importa? Digamos que executamos nosso teste de desempenho em um cluster de desenvolvimento em que um aplicativo NGINX era o único contêiner ocupado nesse nó. Quando movemos o aplicativo para produção, acontece o seguinte: o aplicativo NGINX quer 4 vCPUs de recursos, no entanto, como todos os outros pods no nó estão ocupados, o desempenho do nosso aplicativo é limitado.



Essa situação nos levaria a adicionar mais contêineres desnecessariamente porque não estávamos permitindo que nossos aplicativos fossem escalados até seu “``ponto ideal"`. Vamos explorar esse importante conceito de a com `"sweet spot"` um pouco mais de detalhes.

Tamanho correto da aplicação

Cada aplicativo tem um certo ponto em que não pode receber mais tráfego. Ir acima desse ponto pode aumentar os tempos de processamento e até mesmo reduzir o tráfego quando ultrapassado esse ponto. Isso é conhecido como ponto de saturação do aplicativo. Para evitar problemas de escalabilidade, devemos tentar escalar o aplicativo antes que ele atinja seu ponto de saturação. Vamos chamar esse ponto de ponto ideal.



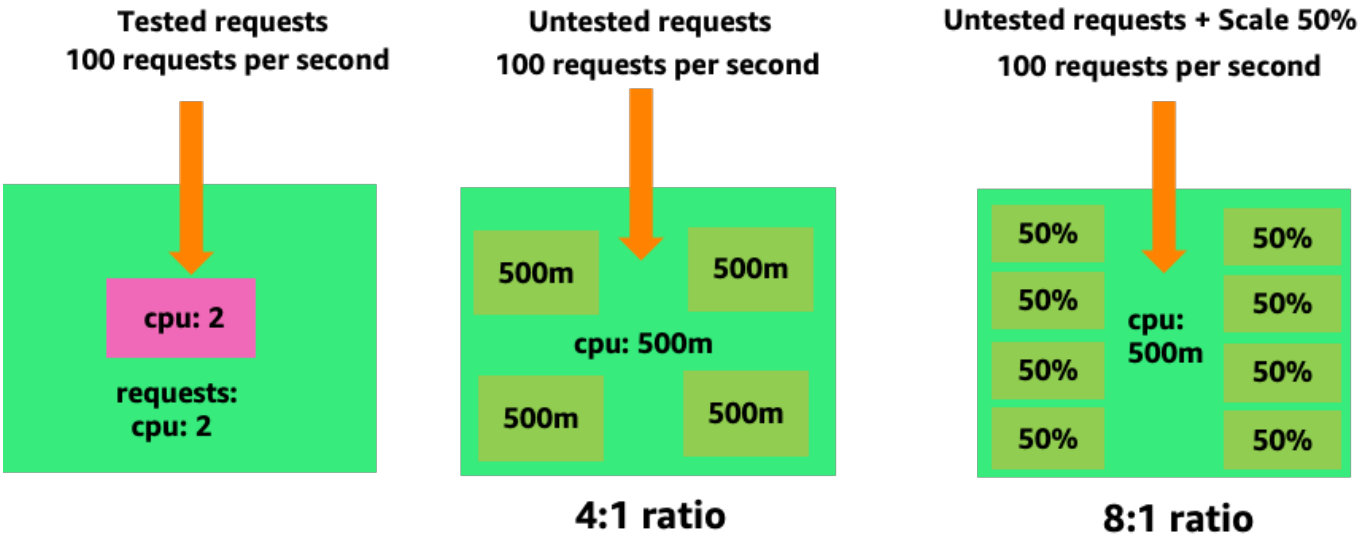
Precisamos testar cada um de nossos aplicativos para entender seu ponto ideal. Não haverá orientação universal aqui, pois cada aplicativo é diferente. Durante esse teste, estamos tentando entender a melhor métrica que mostra o ponto de saturação de nossos aplicativos. Muitas vezes, as métricas de utilização são usadas para indicar que um aplicativo está saturado, mas isso pode levar rapidamente a problemas de escalabilidade (exploraremos esse tópico em detalhes em uma seção posterior). Quando tivermos esse “`ponto ideal"`, podemos usá-lo para escalar com eficiência nossas cargas de trabalho.

Por outro lado, o que aconteceria se aumentássemos a escala bem antes do ponto ideal e criássemos frutos desnecessários? Vamos explorar isso na próxima seção.

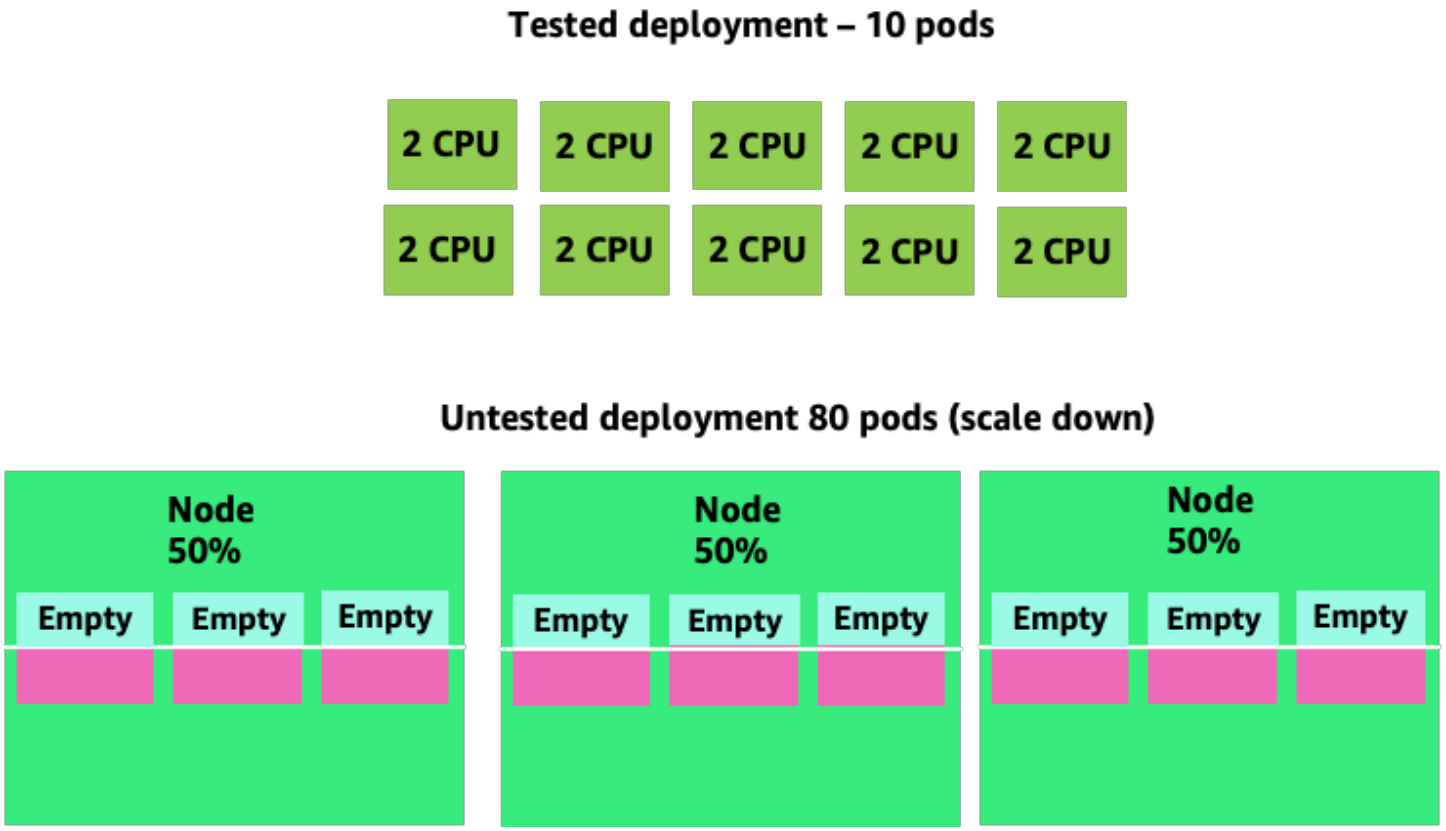
Expansão de cápsulas

Para ver como a criação de cápsulas desnecessárias pode sair do controle rapidamente, vamos dar uma olhada no primeiro exemplo à esquerda. A escala vertical correta desse contêiner ocupa cerca de dois v CPUs de utilização ao lidar com 100 solicitações por segundo. No entanto, se subprovisionássemos o valor das solicitações definindo as solicitações para meio núcleo, agora

precisaríamos de 4 pods para cada um dos pods realmente necessários. Para agravar ainda mais esse problema, se nosso HPA fosse definido como padrão de 50% da CPU, esses pods ficariam meio vazios, criando uma proporção de 8:1.

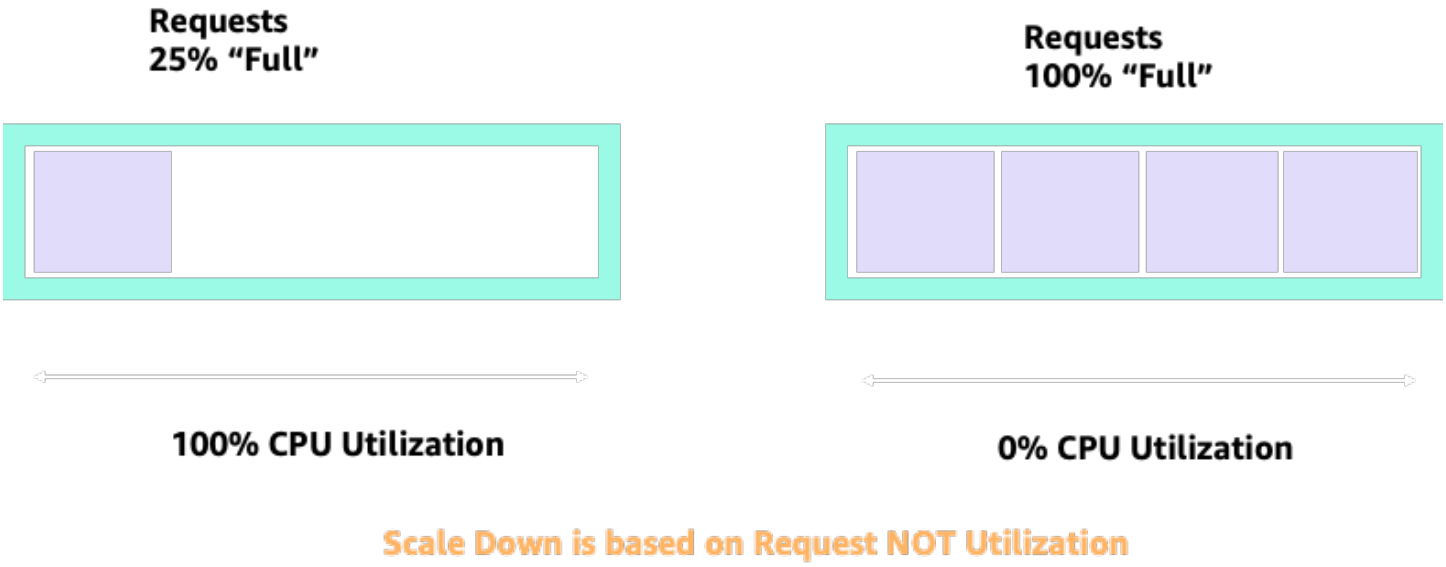


Aumentando esse problema, podemos ver rapidamente como isso pode sair do controle. Uma implantação de dez pods cujo ponto ideal foi definido incorretamente poderia rapidamente chegar a 80 pods e a infraestrutura adicional necessária para operá-los.



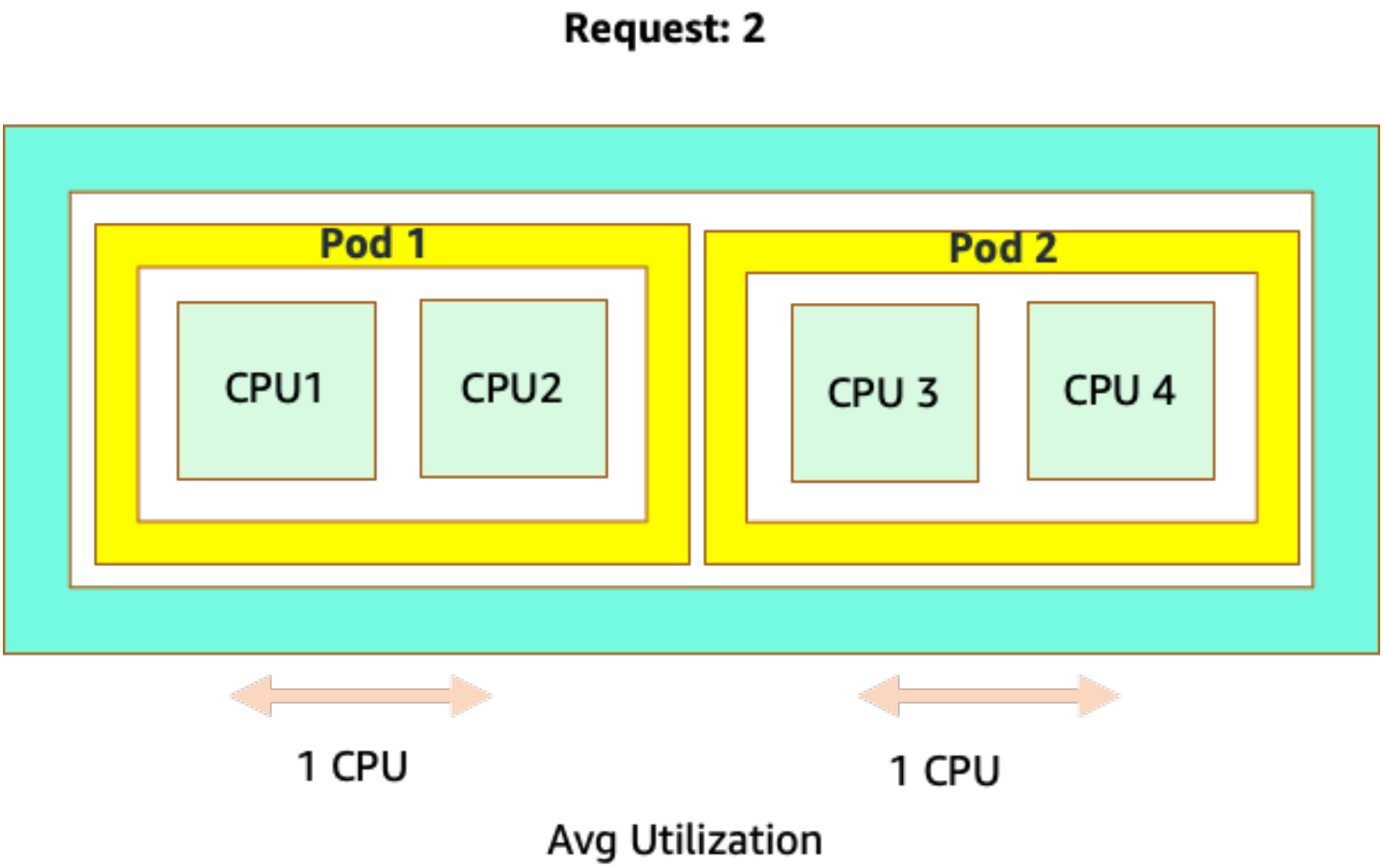
Agora que entendemos o impacto de não permitir que os aplicativos operem em seu ponto ideal, vamos voltar ao nível do nó e perguntar por que essa diferença entre o programador do Kubernetes e o CFS do Linux é tão importante?

Ao aumentar e diminuir a escala com o HPA, podemos ter um cenário em que temos muito espaço para alocar mais pods. Essa seria uma decisão ruim porque o nó representado à esquerda já está com 100% de utilização da CPU. Em um cenário irreal, mas teoricamente possível, poderíamos ter o outro extremo em que nosso nó está completamente cheio, mas nossa utilização de CPU é zero.

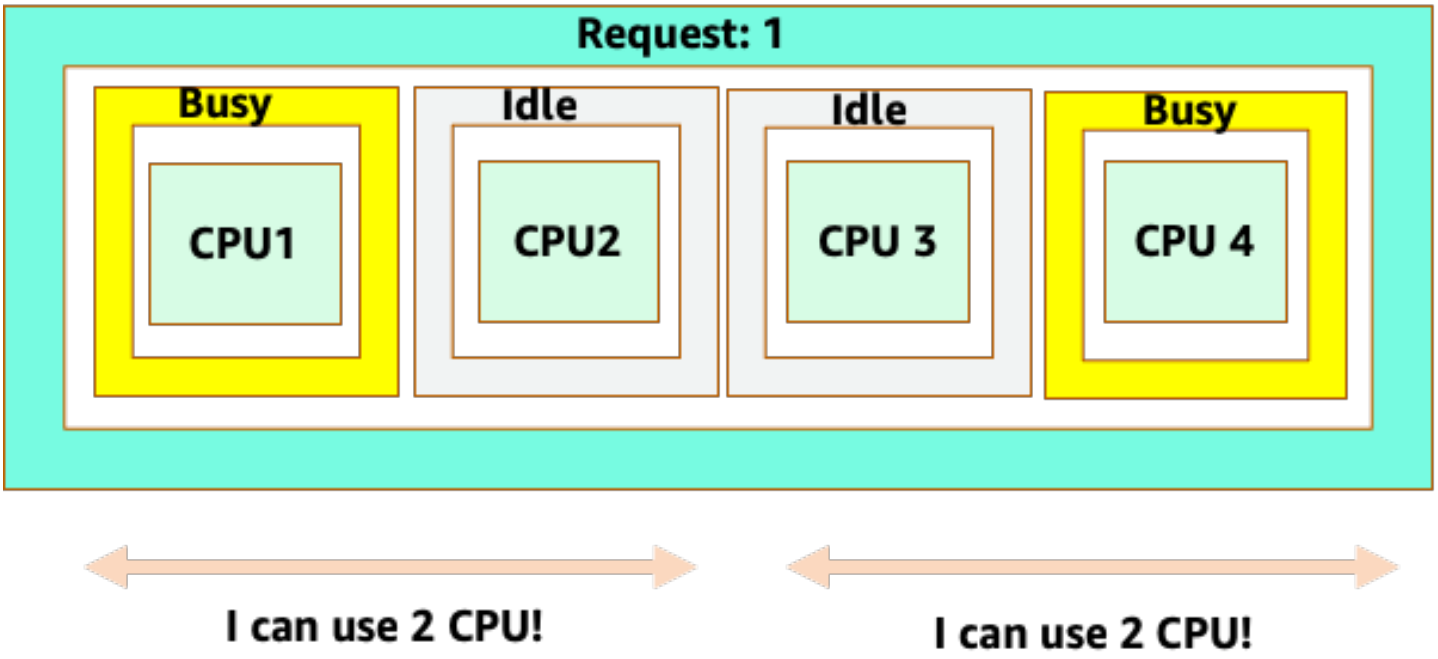


Solicitações de configuração

Seria tentador definir a solicitação no valor “ideal” desse aplicativo, mas isso causaria ineficiências, conforme ilustrado no diagrama abaixo. Aqui, definimos o valor da solicitação como 2 vCPUs, no entanto, a utilização média desses pods executa apenas 1 CPU na maioria das vezes. Essa configuração faria com que desperdiçássemos 50% de nossos ciclos de CPU, o que seria inaceitável.



Isso nos leva à resposta complexa para o problema. A utilização do contêiner não pode ser considerada isoladamente; é preciso levar em consideração os outros aplicativos em execução no nó. No exemplo a seguir, contêineres que são intermitentes por natureza são misturados com dois contêineres de baixa utilização da CPU que podem ter restrições de memória. Dessa forma, permitimos que os contêineres atinjam seu ponto ideal sem sobrecarregar o nó.



O conceito importante a ser extraído de tudo isso é que usar o conceito de núcleos do programador Kubernetes para entender o desempenho do contêiner Linux pode levar a uma tomada de decisão inadequada, pois eles não estão relacionados.

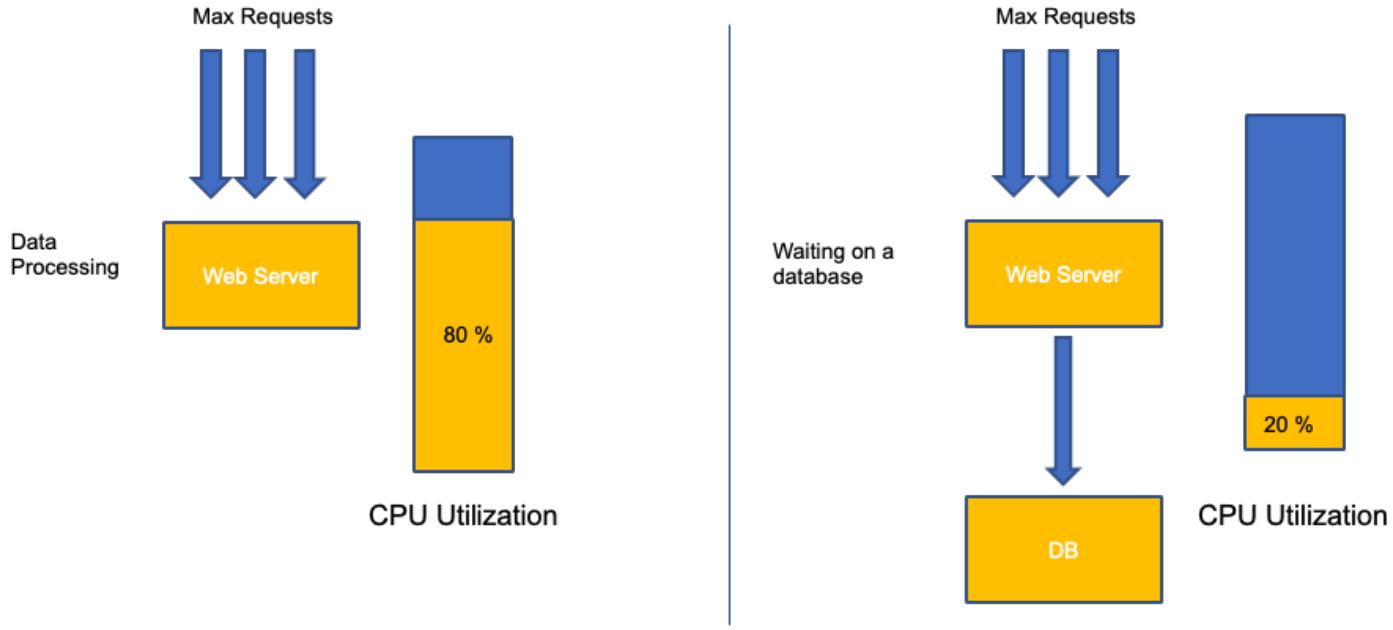
> **Note**
>
> O Linux CFS tem seus pontos fortes. Isso é especialmente verdadeiro para cargas de trabalho I/O baseadas. No entanto, se seu aplicativo usa núcleos completos sem cartões auxiliares e não tem I/O requisitos, a fixação da CPU pode remover uma grande complexidade desse processo e é incentivada com essas ressalvas.

Utilização versus saturação

Um erro comum no escalonamento de aplicativos é usar apenas a utilização da CPU para sua métrica de escalabilidade. Em aplicativos complexos, isso quase sempre é um indicador fraco de que um aplicativo está realmente saturado de solicitações. No exemplo à esquerda, vemos que todas as nossas solicitações estão realmente chegando ao servidor web, então a utilização da CPU está acompanhando bem a saturação.

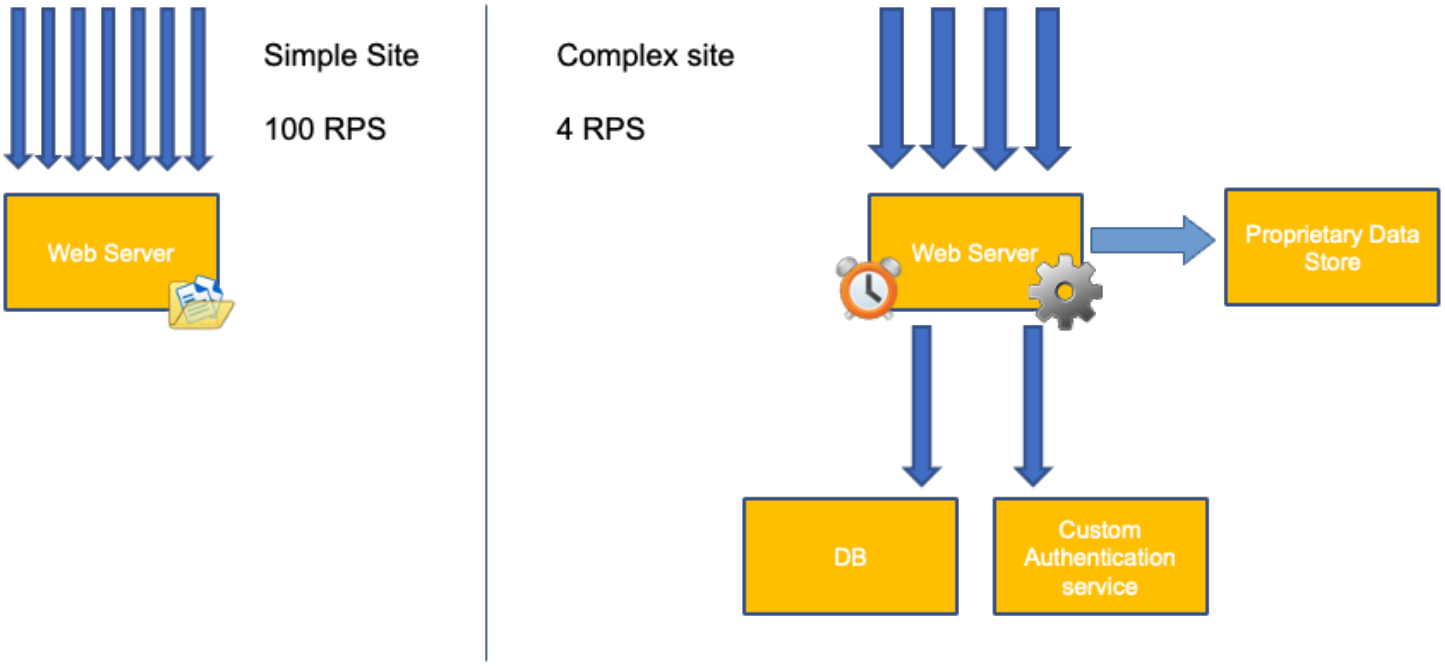
Em aplicativos do mundo real, é provável que algumas dessas solicitações sejam atendidas por uma camada de banco de dados ou uma camada de autenticação, etc. Nesse caso mais comum, observe

que a CPU não está rastreando com saturação, pois a solicitação está sendo atendida por outras entidades. Nesse caso, a CPU é um indicador muito ruim de saturação.



Usar a métrica errada no desempenho do aplicativo é o principal motivo para o escalonamento desnecessário e imprevisível no Kubernetes. É preciso ter muito cuidado ao escolher a métrica de saturação correta para o tipo de aplicativo que você está usando. É importante observar que não existe uma recomendação única que possa ser dada. Dependendo da linguagem usada e do tipo de aplicativo em questão, há um conjunto diversificado de métricas de saturação.

Podemos pensar que esse problema ocorre apenas com a utilização da CPU, no entanto, outras métricas comuns, como solicitação por segundo, também podem se enquadrar exatamente no mesmo problema discutido acima. Observe que a solicitação também pode ir para camadas de banco de dados, camadas de autenticação, não sendo atendida diretamente pelo nosso servidor web, portanto, é uma métrica ruim para a verdadeira saturação do próprio servidor web.



Infelizmente, não há respostas fáceis quando se trata de escolher a métrica de saturação correta. Aqui estão algumas diretrizes a serem levadas em consideração:

- Entenda o tempo de execução da sua linguagem - linguagens com vários encadeamentos de sistema operacional reagirão de forma diferente dos aplicativos de um único encadeamento, afetando o nó de forma diferente.
- Entenda a escala vertical correta: quanto buffer você quer na escala vertical de seus aplicativos antes de escalar um novo pod?
- Quais métricas realmente refletem a saturação do seu aplicativo - A métrica de saturação para um produtor Kafka seria bem diferente de um aplicativo web complexo.
- Como todos os outros aplicativos no nó afetam uns aos outros? O desempenho do aplicativo não é feito no vácuo; as outras cargas de trabalho no nó têm um grande impacto.

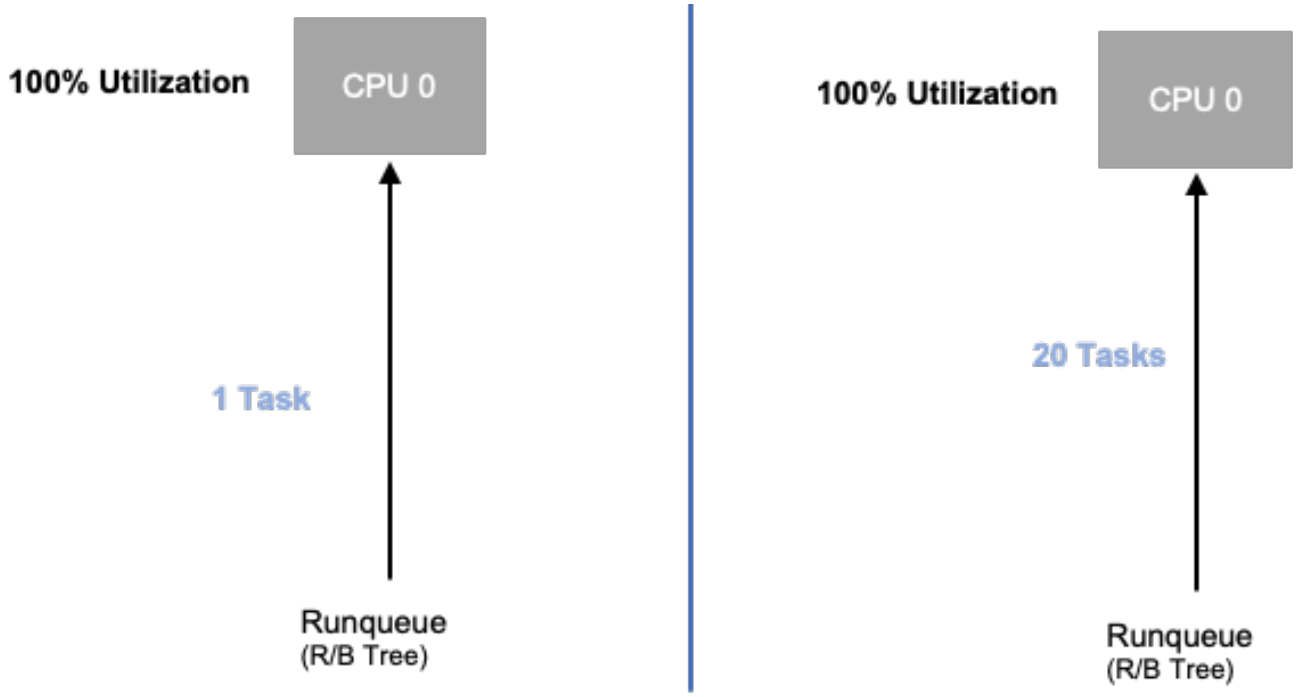
Para encerrar esta seção, seria fácil descartar o que foi dito acima como excessivamente complexo e desnecessário. Muitas vezes, podemos estar enfrentando um problema, mas não temos conhecimento da verdadeira natureza do problema porque estamos analisando as métricas erradas. Na próxima seção, veremos como isso pode acontecer.

Saturação do nó

Agora que exploramos a saturação de aplicativos, vamos analisar esse mesmo conceito do ponto de vista de um nó. Vamos pegar dois CPUs que são 100% utilizados para ver a diferença entre utilização e saturação.

A vCPU à esquerda é 100% utilizada, no entanto, nenhuma outra tarefa está esperando para ser executada nessa vCPU, portanto, em um sentido puramente teórico, isso é bastante eficiente. Enquanto isso, temos 20 aplicativos de thread único aguardando para serem processados por uma vCPU no segundo exemplo. Agora, todos os 20 aplicativos experimentarão algum tipo de latência enquanto aguardam sua vez de serem processados pela vCPU. Em outras palavras, a vCPU à direita está saturada.

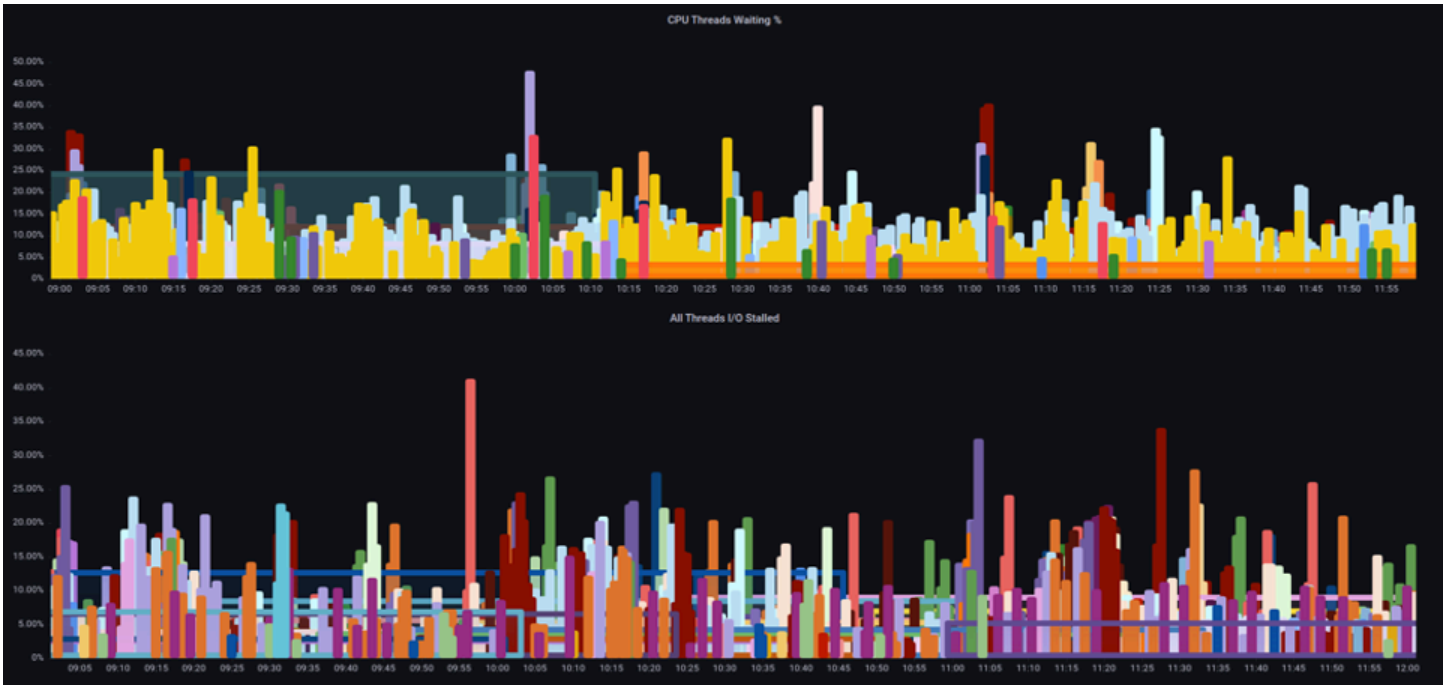
Não só não veríamos esse problema se estivéssemos apenas analisando a utilização, mas poderíamos atribuir essa latência a algo não relacionado, como uma rede, que nos levaria ao caminho errado.



É importante visualizar as métricas de saturação, não apenas as métricas de utilização, ao aumentar o número total de pods em execução em um nó a qualquer momento, pois podemos facilmente ignorar o fato de termos saturado demais um nó. Para essa tarefa, podemos usar métricas de informações de parada de pressão, conforme mostrado no gráfico abaixo.

PromQL - E/S paralisada

```
topk(3, ((irate(node_pressure_io_stalled_seconds_total[1m])) * 100))
```



> **Note**
>
> Para obter mais informações sobre as métricas de paralisação de pressão, consulte https:// facebookmicrosites.github. io/psi/docs/overview*

Com essas métricas, podemos dizer se os encadeamentos estão aguardando na CPU ou mesmo se todos os encadeamentos na caixa estão paralisados aguardando recursos, como memória, ou I/O. For example, we could see what percentage every thread on the instance was stalled waiting on I/O por um período de 1 min.

```
topk(3, ((irate(node_pressure_io_stalled_seconds_total[1m])) * 100))
```

Usando essa métrica, podemos ver no gráfico acima que cada thread na caixa ficou paralisado em 45% do tempo de espera I/O no limite máximo, o que significa que estávamos jogando fora todos esses ciclos de CPU naquele minuto. Entender que isso está acontecendo pode nos ajudar a recuperar uma quantidade significativa de tempo de vCPU, tornando o escalonamento mais eficiente.

HPA V2

É recomendável usar a versão autoscaling/v2 da API HPA. As versões mais antigas da API HPA podem travar o escalonamento em alguns casos extremos. Também foi limitado à duplicação de pods durante cada etapa de escalonamento, o que criou problemas para pequenas implantações que precisavam ser escaladas rapidamente.

O escalonamento automático/V2 nos permite mais flexibilidade para incluir vários critérios para escalar e nos permite uma grande flexibilidade ao usar métricas personalizadas e externas (não métricas K8s).

Como exemplo, podemos escalar o maior dos três valores (veja abaixo). Nós escalamos se a utilização média de todos os pods for superior a 50%, se as métricas personalizadas mostrarem os pacotes por segundo de entrada excederem a média de 1.000 ou se o objeto de entrada exceder 10.000 solicitações por segundo.

> **Note**
>
> Isso é apenas para mostrar a flexibilidade da API de auto-scaling, que não recomendamos regras excessivamente complexas que podem ser difíceis de solucionar na produção.

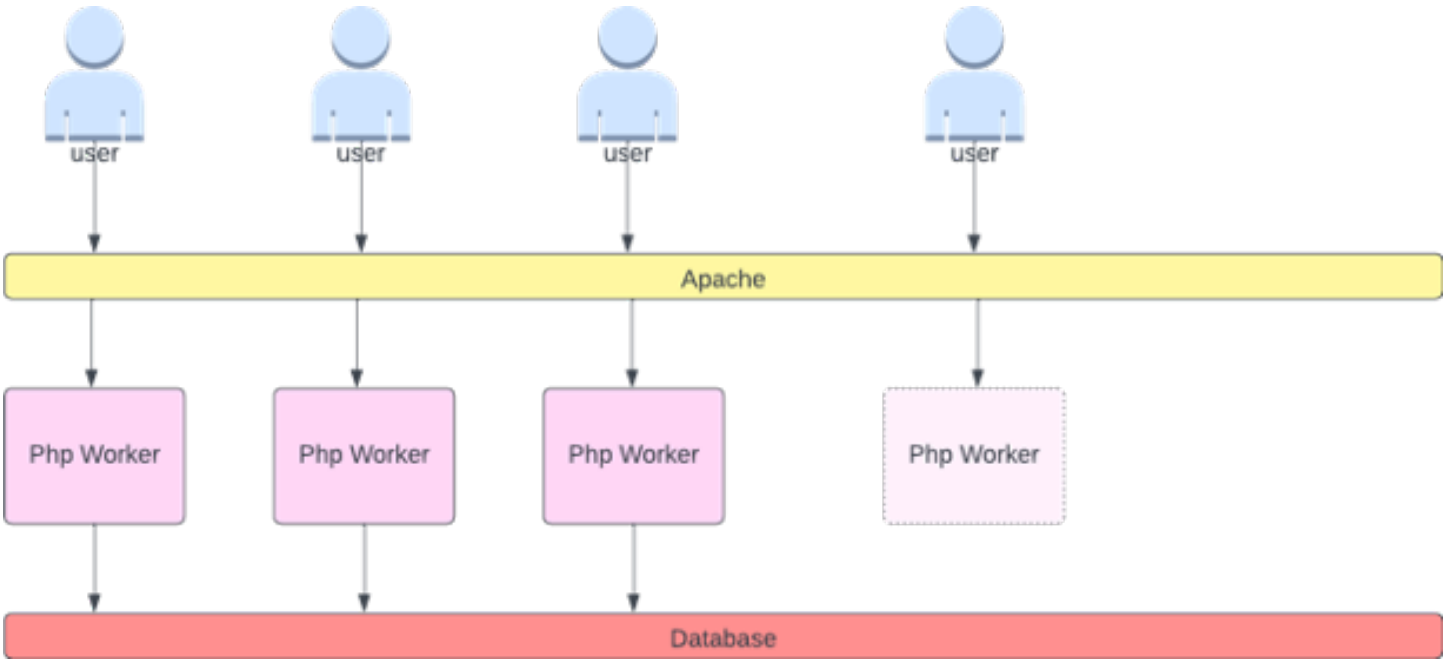
```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: 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
```

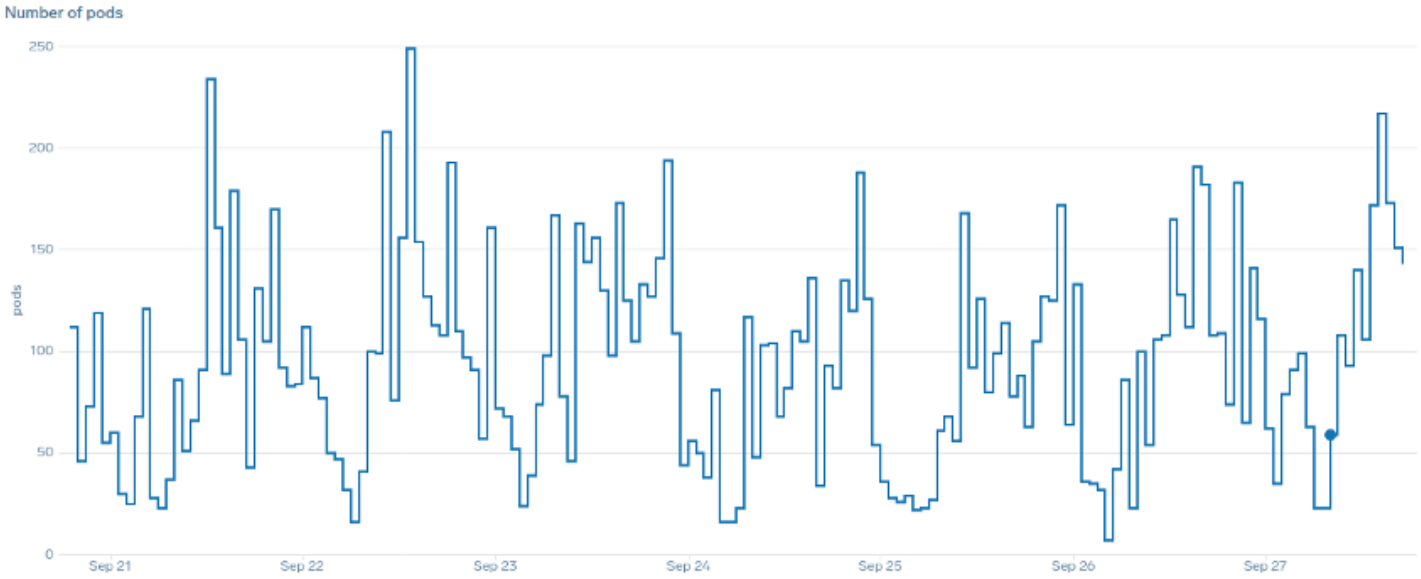
No entanto, aprendemos o perigo de usar essas métricas para aplicativos web complexos. Nesse caso, seria melhor usar métricas personalizadas ou externas que refletissem com precisão a saturação de nosso aplicativo versus a utilização. HPAv2 permite isso ao ter a capacidade de escalar de acordo com qualquer métrica, no entanto, ainda precisamos encontrar e exportar essa métrica para o Kubernetes para uso.

Por exemplo, podemos ver a contagem ativa de filas de threads no Apache. Isso geralmente cria um perfil de escala “mais suave” (falaremos mais sobre esse termo em breve). Se um encadeamento estiver ativo, não importa se esse encadeamento está aguardando em uma camada de banco de dados ou atendendo a uma solicitação localmente; se todos os encadeamentos do aplicativo estiverem sendo usados, é uma ótima indicação de que o aplicativo está saturado.

Podemos usar esse esgotamento de threads como um sinal para criar um novo pod com um pool de threads totalmente disponível. Isso também nos dá controle sobre o tamanho do buffer que queremos que o aplicativo absorva durante períodos de tráfego intenso. Por exemplo, se tivéssemos um pool total de 10 threads, escalar em 4 threads usados versus 8 threads usados teria um grande impacto no buffer que temos disponível ao escalar o aplicativo. Uma configuração de 4 faria sentido para um aplicativo que precisa ser escalado rapidamente sob carga pesada, enquanto uma configuração de 8 seria mais eficiente com nossos recursos se tivéssemos tempo suficiente para escalar devido ao número de solicitações aumentando lentamente em vez de acentuadamente com o tempo.

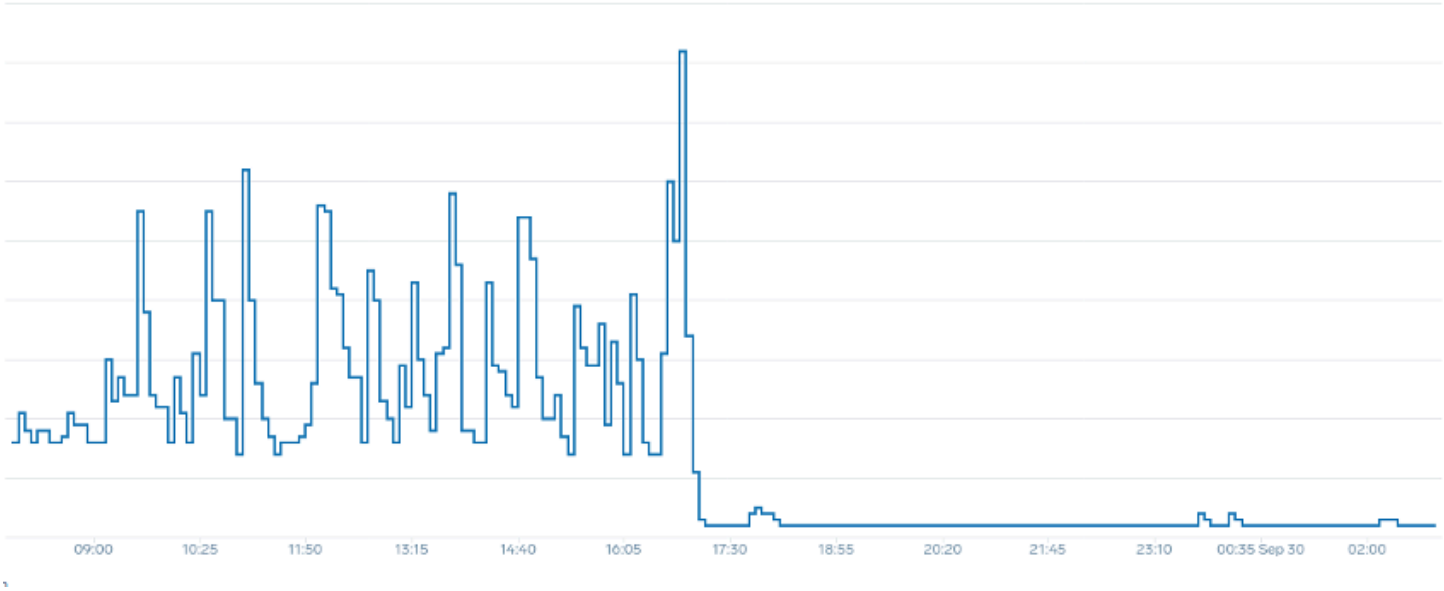


O que queremos dizer com o termo “suave” quando se trata de escalabilidade? Observe o gráfico abaixo em que estamos usando a CPU como métrica. Os pods nesta implantação estão aumentando em um curto período, de 50 pods até 250 pods, apenas para serem reduzidos imediatamente novamente. Isso é altamente ineficiente. O escalonamento é a principal causa de rotatividade em clusters.



Observe que, depois de mudarmos para uma métrica que reflete o ponto ideal correto de nosso aplicativo (parte central do gráfico), podemos escalar sem problemas. Nosso escalonamento agora é eficiente, e nossos pods podem ser totalmente escalados com o espaço disponível que fornecemos ajustando as configurações de solicitações. Agora, um grupo menor de cápsulas está fazendo o

trabalho que centenas de cápsulas faziam antes. Dados do mundo real mostram que esse é o fator número um na escalabilidade dos clusters Kubernetes.

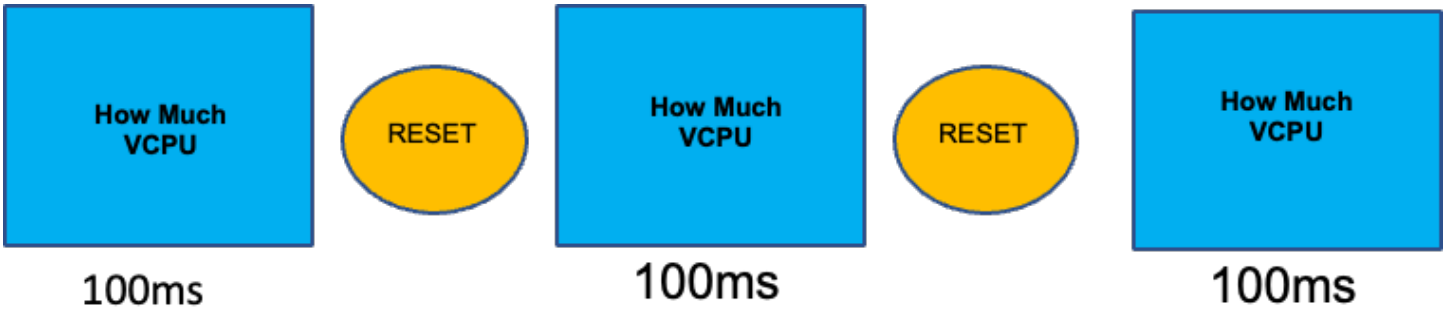


A principal conclusão é que a utilização da CPU é apenas uma dimensão do desempenho do aplicativo e do nó. Usar a utilização da CPU como um único indicador de integridade para nossos nós e aplicativos cria problemas de escalabilidade, desempenho e custo, todos conceitos estreitamente vinculados. Quanto maior o desempenho do aplicativo e dos nós, menos você precisa escalar, o que, por sua vez, reduz seus custos.

Encontrar e usar as métricas de saturação corretas para escalar seu aplicativo específico também permite monitorar e alertar sobre os verdadeiros gargalos desse aplicativo. Se essa etapa crítica for ignorada, os relatórios de problemas de desempenho serão difíceis, se não impossíveis, de entender.

Definindo limites de CPU

Para completar esta seção sobre tópicos incompreendidos, abordaremos os limites de CPU. Resumindo, os limites são metadados associados ao contêiner que tem um contador que é reiniciado a cada 100 ms. Isso ajuda o Linux a controlar quantos recursos da CPU são usados em todo o nó por um contêiner específico em um período de 100 ms.



Um erro comum ao definir limites é presumir que o aplicativo é de thread único e está sendo executado somente em sua vCPU “`atribuída"`. Na seção acima, aprendemos que o CFS não atribui núcleos e, na realidade, um contêiner executando grandes pools de threads será programado em todas as vCPUs disponíveis na caixa.

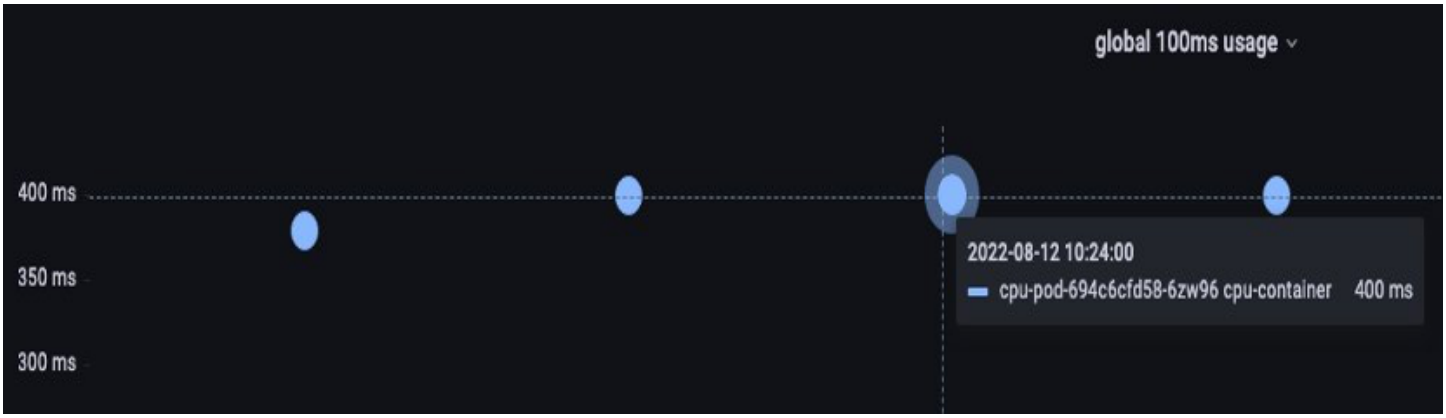
Se 64 threads de sistema operacional estiverem sendo executados em 64 núcleos disponíveis (do ponto de vista do nó Linux), tornaremos a conta total de tempo de CPU usado em um período de 100 ms muito grande após a soma do tempo de execução de todos esses 64 núcleos. Como isso pode ocorrer apenas durante um processo de coleta de lixo, pode ser muito fácil perder algo assim. É por isso que é necessário usar métricas para garantir que tenhamos o uso correto ao longo do tempo antes de tentar definir um limite.

Felizmente, temos uma maneira de ver exatamente quanta vCPU está sendo usada por todos os threads em um aplicativo. Usaremos a métrica `container_cpu_usage_seconds_total` para essa finalidade.

Como a lógica de limitação ocorre a cada 100 ms e essa métrica é uma métrica por segundo, faremos o ProMQL para corresponder a esse período de 100 ms. Se você quiser se aprofundar nesse trabalho de declaração do PromQL, consulte o blog a seguir.

Consulta PromQL:

```
topk(3, max by (pod, container)(rate(container_cpu_usage_seconds_total{image!="",
 instance="$instance"}[$__rate_interval]))) / 10
```



Quando sentirmos que temos o valor certo, podemos limitar a produção. Em seguida, torna-se necessário verificar se nosso aplicativo está sendo estrangulado devido a algo inesperado. Podemos fazer isso olhando para `container_cpu_throttled_seconds_total`

```
topk(3, max by (pod, container)(rate(container_cpu_cfs_throttled_seconds_total{image!
=``""``, instance=``"$instance"``}[$__rate_interval]))) / 10
```



Memória

A alocação de memória é outro exemplo em que é fácil confundir o comportamento de agendamento do Kubernetes com o comportamento do Linux. CGroup Este é um tópico com mais nuances, pois houve grandes mudanças na forma como a CGroup v2 lida com a memória no Linux e o Kubernetes mudou sua sintaxe para refletir isso; leia este blog para obter mais detalhes.

Diferentemente das solicitações de CPU, as solicitações de memória não são usadas após a conclusão do processo de agendamento. Isso ocorre porque não podemos compactar a memória na CGroup v1 da mesma forma que podemos com a CPU. Isso nos deixa apenas com limites de memória, que são projetados para funcionar como uma proteção contra vazamentos de memória

ao encerrar completamente o pod. Esta é uma proposta de estilo tudo ou nada, no entanto, agora recebemos novas maneiras de resolver esse problema.

Primeiro, é importante entender que configurar a quantidade certa de memória para contêineres não é tão simples quanto parece. O sistema de arquivos no Linux usará memória como cache para melhorar o desempenho. Esse cache crescerá com o tempo, e pode ser difícil saber quanta memória é bom ter para o cache, mas pode ser recuperada sem um impacto significativo no desempenho do aplicativo. Isso geralmente resulta na interpretação incorreta do uso da memória.

Ter a capacidade de “compactar” a memória foi um dos principais fatores por trás da CGroup v2. Para saber mais sobre por que a CGroup V2 foi necessária, consulte a apresentação de Chris Down, na LISA21 qual ele explica por que não conseguir definir a memória mínima corretamente foi um dos motivos que o levaram a criar métricas de CGroup v2 e de paralisação de pressão.

Felizmente, o Kubernetes agora tem o conceito de `memory.min` e abaixo. `memory.high` `requests.memory` Isso nos dá a opção de liberar agressivamente essa memória em cache para uso de outros contêineres. Quando o contêiner atinge o limite máximo de memória, o kernel pode recuperar agressivamente a memória desse contêiner até o valor definido em. `memory.min` Isso nos dá mais flexibilidade quando um nó fica sob pressão de memória.

A questão chave é: qual valor `memory.min` definir? É aqui que as métricas de paralisação da pressão de memória entram em ação. Podemos usar essas métricas para detectar “perda” de memória no nível do contêiner. Então, podemos usar controladores como fbtax para detectar os valores corretos procurando `memory.min` por essa perda de memória e definir dinamicamente o valor para essa configuração. `memory.min`

Resumo

Para resumir a seção, é fácil combinar os seguintes conceitos:

- Utilização e saturação
- Regras de desempenho do Linux com a lógica do Kubernetes Scheduler

É preciso ter muito cuidado para manter esses conceitos separados. Desempenho e escala estão interligados em um nível profundo. O escalonamento desnecessário cria problemas de desempenho, o que, por sua vez, cria problemas de escalabilidade.

Kubernetes Upstream SLOs

O Amazon EKS executa o mesmo código das versões upstream do Kubernetes e garante que os clusters EKS operem dentro do SLOs definido pela comunidade do Kubernetes. O Kubernetes Scalability Special Interest Group (SIG) define as metas de escalabilidade e investiga gargalos no desempenho por meio de e. SLIs SLOs

SLIs são como medimos um sistema, como métricas ou medidas que podem ser usadas para determinar o quão “bem” o sistema está funcionando, por exemplo, latência ou contagem de solicitações. SLOs defina os valores que são esperados para quando o sistema está funcionando “bem”, por exemplo, a latência da solicitação permanece em menos de 3 segundos. O Kubernetes SLOs e o Kubernetes SLIs se concentram no desempenho dos componentes do Kubernetes e são completamente independentes do Amazon EKS Service, SLAs que se concentra na disponibilidade do endpoint do cluster EKS.

O Kubernetes tem vários recursos que permitem aos usuários ampliar o sistema com complementos ou drivers personalizados, como drivers CSI, webhooks de admissão e escalonadores automáticos. Essas extensões podem impactar drasticamente o desempenho de um cluster Kubernetes de maneiras diferentes, ou seja, um webhook de admissão `failurePolicy=Ignore` pode adicionar latência às solicitações de API K8s se o destino do webhook não estiver disponível. O Kubernetes Scalability SIG define escalabilidade usando uma estrutura do tipo “você promete, nós prometemos”:

> Se você prometer: - configurar corretamente seu cluster - usar os recursos de extensibilidade “razoavelmente” - manter a carga no cluster dentro dos limites recomendados
>
> então prometemos que seu cluster se expanda, ou seja: - todos SLOs estão satisfeitos.

Kubernetes SLOs

O Kubernetes SLOs não leva em conta todos os plug-ins e limitações externas que podem afetar um cluster, como escalabilidade de nós de trabalho ou webhooks de admissão. Eles SLOs se concentram nos componentes do Kubernetes e garantem que as ações e os recursos do Kubernetes estejam operando dentro das expectativas. Isso SLOs ajuda os desenvolvedores do Kubernetes a garantir que as alterações no código do Kubernetes não prejudiquem o desempenho de todo o sistema.

O Kubernetes Scalability SIG define o seguinte SLO/ oficial. SLIs A equipe do Amazon EKS executa regularmente testes de escalabilidade em clusters EKS SLOs/SLIs para monitorar a degradação do desempenho à medida que mudanças são feitas e novas versões são lançadas.

Objetivo	Definição	SLO
Latência da solicitação de API (mutação)	Latência de processamento de chamadas de API mutantes para objetos únicos para cada par (recurso, verbo), medida como 99º percentil nos últimos 5 minutos	Na instalação padrão do Kubernetes, para cada par (recurso, verbo), excluindo recursos virtuais e agregados e definições de recursos personalizadas, 99º percentil por dia de cluster <= 1s
Latência da solicitação de API (somente leitura)	Latência de processamento de chamadas de API sem streaming (somente leitura) para cada par (recurso, escopo), medida como 99º percentil nos últimos 5 minutos	Na instalação padrão do Kubernetes, para cada par (recurso, escopo), excluindo recursos virtuais e agregados e definições de recursos personalizadas, 99º percentil por dia de cluster: (a) <= 1s se (b) <= 30s caso contrário (se ou) `scope=resource` `scope=namespace` `scope=cluster`
Latência de inicialização do pod	Latência de inicialização de pods sem estado programáv eis, excluindo o tempo para extrair imagens e executar contêineres de inicialização, medido desde o registro de data e hora de criação do pod até o momento em que todos os contêineres são relatados como iniciados e observado s por meio do watch, medido	Na instalação padrão do Kubernetes, 99º percentil por dia de cluster <= 5s

Objetivo	Definição	SLO
	como 99º percentil nos últimos 5 minutos	

Latência da solicitação de API

O `kube-apiserver` foi `--request-timeout` definido como `1m0s` por padrão, o que significa que uma solicitação pode ser executada por até um minuto (60 segundos) antes de atingir o tempo limite e ser cancelada. Os SLOs definidos para latência são divididos pelo tipo de solicitação que está sendo feita, que pode ser mutante ou somente para leitura:

Mutante

Solicitações mutantes no Kubernetes fazem alterações em um recurso, como criações, exclusões ou atualizações. Essas solicitações são caras porque essas alterações devem ser gravadas no back-end do etcd antes que o objeto atualizado seja retornado. O Etcd é um armazenamento distribuído de valores-chave usado para todos os dados do cluster Kubernetes.

Essa latência é medida como o 99º percentil em 5 minutos para pares (recurso, verbo) de recursos do Kubernetes. Por exemplo, isso mediria a latência das solicitações Create Pod e Update Node. A latência da solicitação deve ser <= 1 segundo para satisfazer o SLO.

Somente leitura

Solicitações somente para leitura recuperam um único recurso (como Get Pod X) ou uma coleção (como “Get all Pods from Namespace X”). O `kube-apiserver` mantém um cache de objetos, portanto, os recursos solicitados podem ser retornados do cache ou talvez precisem ser recuperados primeiro do etcd. Essas latências também são medidas pelo 99º percentil em 5 minutos, no entanto, solicitações somente para leitura podem ter escopos separados. O SLO define dois objetivos diferentes:

- Para solicitações feitas para um único recurso (ou seja,`kubectl get pod -n mynamespace my-controller-xxx`), a latência da solicitação deve permanecer <= 1 segundo.
- Para solicitações feitas para vários recursos em um namespace ou cluster (por exemplo,`kubectl get pods -A`), a latência deve permanecer <= 30 segundos

O SLO tem valores-alvo diferentes para diferentes escopos de solicitação porque as solicitações feitas para uma lista de recursos do Kubernetes esperam que os detalhes de todos os objetos na solicitação sejam retornados dentro do SLO. Em grandes clusters ou grandes coleções de recursos, isso pode resultar em grandes tamanhos de resposta que podem levar algum tempo para retornar. Por exemplo, em um cluster executando dezenas de milhares de pods, com cada pod tendo aproximadamente 1 KiB quando codificado em JSON, o retorno de todos os pods no cluster consistiria em 10 MB ou mais. Os clientes do Kubernetes podem ajudar a reduzir esse tamanho de resposta usando o APIList Chunking para recuperar grandes coleções de recursos.

Latência de inicialização do pod

Esse SLO se preocupa principalmente com o tempo que leva desde a criação do pod até o momento em que os contêineres desse pod realmente começam a ser executados. Para medir isso, calcula-se a diferença entre o timestamp de criação registrado no pod e quando um WATCH nesse pod relata que os contêineres foram iniciados (excluindo o tempo de extração da imagem do contêiner e da execução inicial do contêiner). Para satisfazer o SLO, o 99º percentil por dia de cluster dessa latência de inicialização do pod deve permanecer <=5 segundos.

Observe que esse SLO pressupõe que os nós de trabalho já existam nesse cluster em um estado pronto para o agendamento do pod. Esse SLO não leva em conta a extração de imagens ou a execução de contêineres de inicialização e também limita o teste a “pods sem estado” que não utilizam plug-ins de armazenamento persistente.

Métricas de SLI do Kubernetes

O Kubernetes também está melhorando a observabilidade ao SLIs adicionar métricas do Prometheus aos componentes do Kubernetes que as monitoram ao longo do tempo. SLIs Usando a Prometheus Query Language (PromQL), podemos criar consultas que exibem o desempenho do SLI ao longo do tempo em ferramentas como os painéis Prometheus ou Grafana. Abaixo estão alguns exemplos acima. SLOs

Latência da solicitação do servidor de API

Métrica	Definição
apiserver_request_sli_duration_seconds	Distribuição da latência de resposta (sem contar a duração do webhook e os tempos de espera da fila de prioridade e imparcialidade)

Métrica	Definição
	em segundos para cada verbo, grupo, versão, recurso, sub-recurso, escopo e componente.
apiserver_request_duration_seconds	Distribuição da latência de resposta em segundos para cada verbo, valor de simulação , grupo, versão, recurso, sub-recurso, escopo e componente.

> **Note**
>
> A `apiserver_request_sli_duration_seconds` métrica está disponível a partir do Kubernetes 1.27.

Você pode usar essas métricas para investigar os tempos de resposta do API Server e se há gargalos nos componentes do Kubernetes ou em outros plugins/componentes. As consultas abaixo são baseadas no painel do SLO da comunidade.

Latência de solicitação de API SLI (mutante) - dessa vez, não inclui a execução do webhook ou o tempo de espera na fila. `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`

Total da latência da solicitação de API (mutante) - esse é o tempo total que a solicitação levou no servidor da API. Esse tempo pode ser maior que o tempo de SLI, pois inclui a execução do webhook e os tempos de espera de prioridade e justiça da 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`

Nessas consultas, estamos excluindo as solicitações da API de streaming que não retornam imediatamente, como `kubectl port-forward` or `kubectl exec` requests (`subresource!~"proxy|attach|log|exec|portforward"`), e estamos filtrando apenas os verbos do Kubernetes que modificam objects (). `verb=~"CREATE|DELETE|PATCH|POST|PUT"` Em seguida, calculamos o 99º percentil dessa latência nos últimos 5 minutos.

Podemos usar uma consulta semelhante para as solicitações de API somente para leitura. Basta modificar os verbos que estamos filtrando para incluir as ações `LIST` somente leitura e. `GET` Também há limites de SLO diferentes, dependendo do escopo da solicitação, ou seja, obter um único recurso ou listar vários recursos.

Latência de solicitação de API SLI (somente leitura) - dessa vez não inclui a execução do webhook ou o tempo de espera na fila. Para um único recurso (escopo=recurso, limite=1s) `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"GET", scope=~"resource"}[5m])) by (resource, subresource, verb, scope, le))`

Para uma coleção de recursos no mesmo namespace (scope=namespace, threshold=5s) `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"LIST", scope=~"namespace"}[5m])) by (resource, subresource, verb, scope, le))`

Para uma coleção de recursos em todo o cluster (escope=cluster, threshold=30s) `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"LIST", scope=~"cluster"}[5m])) by (resource, subresource, verb, scope, le))`

Latência total da solicitação de API (somente leitura) - esse é o tempo total que a solicitação levou no servidor da API. Esse tempo pode ser maior que o tempo de SLI porque inclui a execução do webhook e os tempos de espera. Para um único recurso (escopo=recurso, limite=1s) `histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_bucket{verb=~"GET", scope=~"resource"}[5m])) by (resource, subresource, verb, scope, le))`

Para uma coleção de recursos no mesmo namespace (scope=namespace, threshold=5s) `histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_bucket{verb=~"LIST", scope=~"namespace"}[5m])) by (resource, subresource, verb, scope, le))`

Para uma coleção de recursos em todo o cluster (escope=cluster, threshold=30s) `histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_bucket{verb=~"LIST", scope=~"cluster"}[5m])) by (resource, subresource, verb, scope, le))`

As métricas do SLI fornecem informações sobre o desempenho dos componentes do Kubernetes, excluindo o tempo que as solicitações passam esperando nas filas de prioridade e justiça da API,

trabalhando por meio de webhooks de admissão ou outras extensões do Kubernetes. As métricas totais fornecem uma visão mais holística, pois refletem o tempo que seus aplicativos esperariam por uma resposta do servidor da API. A comparação dessas métricas pode fornecer informações sobre onde os atrasos no processamento da solicitação estão sendo introduzidos.

Latência de inicialização do pod

Métrica	Definição
kubelet_pod_start_sli_duration_seconds	Duração em segundos para iniciar um pod, excluindo o tempo para extrair imagens e executar contêineres de inicialização, medido desde o registro de data e hora de criação do pod até o momento em que todos os contêiner es são relatados como iniciados e observados por meio do watch
kubelet_pod_start_duration_seconds	Duração em segundos, desde o kubelet ver um pod pela primeira vez até o pod começar a funcionar. Isso não inclui o tempo para programar o pod ou expandir a capacidade do nó de trabalho.

> **Note**
>
> `kubelet_pod_start_sli_duration_seconds`está disponível a partir do Kubernetes 1.27.

Semelhante às consultas acima, você pode usar essas métricas para obter informações sobre por quanto tempo o escalonamento de nós, as extraições de imagens e os contêineres de inicialização estão atrasando o lançamento do pod em comparação com as ações do Kubelet.

Latência de inicialização do pod SLI - esse é o tempo desde a criação do pod até o momento em que os contêineres do aplicativo são relatados como em execução. Isso inclui o tempo necessário para que a capacidade do nó de trabalho esteja disponível e o pod seja programado, mas não inclui o tempo necessário para extrair imagens ou para

que os contêineres de inicialização sejam executados. `histogram_quantile(0.99, sum(rate(kubelet_pod_start_sli_duration_seconds_bucket[5m])) by (le))`

Latência total de inicialização do pod - esse é o tempo que o kubelet leva para iniciar o pod pela primeira vez. Isso é medido a partir do momento em que o kubelet recebe o pod via WATCH, o que não inclui o tempo de escalonamento ou agendamento do nó de trabalho. Isso inclui o tempo para extrair imagens e iniciar contêineres para execução. `histogram_quantile(0.99, sum(rate(kubelet_pod_start_duration_seconds_bucket[5m])) by (le))`

SLOs em seu cluster

Se você estiver coletando as métricas do Prometheus dos recursos do Kubernetes em seu cluster EKS, poderá obter informações mais detalhadas sobre o desempenho dos componentes do plano de controle do Kubernetes.

O repositório perf-tests inclui painéis Grafana que exibem as latências e as métricas críticas de desempenho do cluster durante os testes. A configuração de testes de desempenho aproveita o kube-prometheus-stack, um projeto de código aberto que vem configurado para coletar métricas do Kubernetes, mas você também pode usar o Amazon Managed Prometheus e o Amazon Managed Grafana.

Se você estiver usando a solução Prometheus `kube-prometheus-stack` ou uma solução similar, poderá instalar o mesmo painel para observar SLOs o seu cluster em tempo real.

1. Primeiro, você precisará instalar as Regras do Prometheus que são usadas nos painéis com. `kubectl apply -f prometheus-rules.yaml` Você pode baixar uma cópia das regras aqui: https://github.com/kubernetes/ perf- tests/blob/master/clusterloader2/pkg/prometheus/manifests/ prometheus -rules.yaml
   a. Certifique-se de verificar se o namespace no arquivo corresponde ao seu ambiente
   b. Verifique se os rótulos correspondem ao valor do `prometheus.prometheusSpec.ruleSelector` leme se você estiver usando `kube-prometheus-stack`
2. Em seguida, você pode instalar os painéis no Grafana. Os painéis json e os scripts python para gerá-los estão disponíveis aqui: perf- https://github.com/kubernetes/ tests/tree/master/ clusterloader2/pkg/prometheus/manifests/dashboards
   a. o `slo.json` painel exibe o desempenho do cluster em relação ao Kubernetes SLOs

Considere que SLOs eles estão focados no desempenho dos componentes do Kubernetes em seus clusters, mas há métricas adicionais que você pode analisar que fornecem diferentes perspectivas ou insights sobre seu cluster. Projetos comunitários do Kubernetes, como o K, ube-state-metrics podem ajudar você a analisar rapidamente as tendências em seu cluster. Os plug-ins e drivers mais comuns da comunidade Kubernetes também emitem métricas do Prometheus, permitindo que você investigue coisas como escaladores automáticos ou agendadores personalizados.

O Guia de Melhores Práticas de Observabilidade tem exemplos de outras métricas do Kubernetes que você pode usar para obter mais informações.

Limites e Quotas de Serviço Conhecidos

O Amazon EKS pode ser usado para uma variedade de cargas de trabalho e pode interagir com uma ampla variedade de serviços da AWS, e vimos cargas de trabalho de clientes encontrarem uma variedade semelhante de cotas de serviços da AWS e outros problemas que prejudicam a escalabilidade.

Sua conta da AWS tem cotas padrão (um limite máximo no número de cada recurso da AWS que sua equipe pode solicitar). Cada serviço da AWS define sua própria cota, e as cotas geralmente são específicas da região. Você pode solicitar aumentos para algumas cotas (limites flexíveis) e outras cotas não podem ser aumentadas (limites rígidos). Você deve considerar esses valores ao arquitetar seus aplicativos. Considere revisar esses limites de serviço periodicamente e incorporá-los durante o design do aplicativo.

Você pode analisar o uso em sua conta e abrir uma solicitação de aumento de cota no console de Quotas de Serviços da AWS ou usando a AWS CLI. Consulte a documentação da AWS do respectivo serviço da AWS para obter mais detalhes sobre as Cotas de Serviço e quaisquer outras restrições ou avisos sobre seu aumento.

> **Note**
>
> O Amazon EKS Service Quotas lista as cotas de serviço e tem links para solicitar aumentos quando disponíveis.

Outras Cotas de Serviços da AWS

Vimos clientes da EKS serem afetados pelas cotas listadas abaixo para outros serviços da AWS. Alguns deles podem se aplicar apenas a casos de uso ou configurações específicos, no entanto,

você pode considerar se sua solução encontrará algum deles à medida que for escalando. As cotas são organizadas por serviço e cada cota tem uma ID no formato L-XXXXXXXX que você pode usar para pesquisá-la no console de cotas de serviços da AWS

Serviço	Cota (L-xxxxx)	Impacto	Identificação (L-xxxxx)	padrão
IAM	Funções por conta	Pode limitar o número de clusters ou funções do IRSA em uma conta.	FE177L-D64	1.000
IAM	OpenId conectar provedores por conta	Pode limitar o número de clusters por conta, o OpenID Connect é usado pelo IRSA	L-858F3967	100
IAM	Duração da política de confiança da função	Pode limitar o número de clusters aos quais uma função do IAM está associada para o IRSA	L-C07B4B0D	2.048
VPC	Grupos de segurança por interface de rede	Pode limitar o controle ou a conectividade da rede para seu cluster	L-2 AFB9258	5
VPC	IPv4 Blocos CIDR por VPC	Pode limitar o número de EKS Worker Nodes	L-83CA0A9D	5

Serviço	Cota (L-xxxxx)	Impacto	Identificação (L-xxxxx)	padrão
VPC	Rotas por tabela de rotas	Pode limitar o controle ou a conectividade da rede para seu cluster	L-93826ACB	50
VPC	Conexões emparelhadas de VPC ativas por VPC	Pode limitar o controle ou a conectividade da rede para seu cluster	L-7E9ECCDB	50
VPC	Regras de entrada ou saída por grupo de segurança.	Pode limitar o controle ou a conectividade da rede para seu cluster, alguns controladores no EKS criam novas regras	L-0 095F EA8	50
VPC	VPCs por região	Pode limitar o número de clusters por conta ou o controle ou a conectividade da rede para seu cluster	L-F678F1CE	5

Serviço	Cota (L-xxxxx)	Impacto	Identificação (L-xxxxx)	padrão
VPC	Gateways da Internet por região	Pode limitar o número de clusters por conta ou o controle ou a conectividade da rede para seu cluster	L-A4707A72	5
VPC	Interfaces de rede por região	Pode limitar o número de nós do EKS Worker ou impactar as scaling/update atividades do plano de controle do EKS.	DF5L-E4 CA3	5.000
VPC	Uso de endereço de rede	Pode limitar o número de clusters por conta ou o controle ou a conectividade da rede para seu cluster	BB24L-F6E5	64.000

Serviço	Cota (L-xxxxx)	Impacto	Identificação (L-xxxxx)	padrão
VPC	Uso de endereços de rede emparelhados	Pode limitar o número de clusters por conta ou o controle ou a conectividade da rede para seu cluster	L- CD17 FD4 B	128.000
ELB	Ouvintes por Network Load Balancer	Pode limitar o controle da entrada de tráfego no cluster.	L-57A373D6	50
ELB	Grupos de destino por região	Pode limitar o controle da entrada de tráfego no cluster.	L-B2855CB	3.000
ELB	Metas por Application Load Balancer	Pode limitar o controle da entrada de tráfego no cluster.	L-7E6692B2	1.000
ELB	Destinos por Network Load Balancer	Pode limitar o controle da entrada de tráfego no cluster.	EEF1L-AD04	3.000

Serviço	Cota (L-xxxxx)	Impacto	Identificação (L-xxxxx)	padrão
ELB	Destinos por zona de disponibilidade por Network Load Balancer	Pode limitar o controle da entrada de tráfego no cluster.	L-B211E961	500
ELB	Metas por grupo-alvo por região	Pode limitar o controle da entrada de tráfego no cluster.	L-A0D0B863	1.000
ELB	Application Load Balancers por região	Pode limitar o controle da entrada de tráfego no cluster.	L-53 DA6 B97	50
ELB	Classic Load Balancers por região	Pode limitar o controle da entrada de tráfego no cluster.	L-E9E9831D	20
ELB	Network Load Balancers por região	Pode limitar o controle da entrada de tráfego no cluster.	L-69A177A2	50

Serviço	Cota (L-xxxxx)	Impacto	Identificação (L-xxxxx)	padrão
EC2	Executando instâncias padrão sob demanda (A, C, D, H, I, M, R, T, Z) (como uma contagem máxima de vCPU)	Pode limitar o número de EKS Worker Nodes	L-1216C47A	5
EC2	Todas as solicitações de instância spot padrão (A, C, D, H, I, M, R, T, Z) (como contagem máxima de vCPU)	Pode limitar o número de EKS Worker Nodes	L-34B43A08	5
EC2	EC2-PVC elástico IPs	Pode limitar o número de NAT GWs (e, portanto, VPCs), o que pode limitar o número de clusters em uma região	L-0263D0A3	5
EBS	Snapshots por região	Pode limitar a estratégia de backup para cargas de trabalho com estado	L-309 BACF6	100.000

Serviço	Cota (L-xxxxx)	Impacto	Identificação (L-xxxxx)	padrão
EBS	Armazenamento para volumes SSD de uso geral (gp3), em TiB	Pode limitar o número de EKS Worker Nodes ou Persisten tVolume armazenamento	L-7A658B76	50
EBS	Armazenamento para volumes SSD de uso geral (gp2), em TiB	Pode limitar o número de EKS Worker Nodes ou Persisten tVolume armazenamento	LD-18D FCD1	50
ECR	Repositórios registrados	Pode limitar o número de cargas de trabalho em seus clusters	CFEB8L-E8D	100.000
ECR	Imagem por repositório	Pode limitar o número de cargas de trabalho em seus clusters	L-03A36 CE1	20.000
SecretsManager	Segredos por região	Pode limitar o número de cargas de trabalho em seus clusters	L-2F66C23C	500.000

Limitação de solicitações da AWS

Os serviços da AWS também implementam a limitação de solicitações para garantir que continuem funcionando e estejam disponíveis para todos os clientes. Semelhante às Cotas de Serviço, cada serviço da AWS mantém seus próprios limites de limitação de solicitações. Considere revisar a respectiva documentação do AWS Service se suas cargas de trabalho precisarem emitir rapidamente um grande número de chamadas de API ou se você notar erros de limitação de solicitações em seu aplicativo.

EC2 As solicitações de API relacionadas ao provisionamento de interfaces de EC2 rede ou endereços IP podem encontrar limitação de solicitações em grandes clusters ou quando os clusters aumentam drasticamente. A tabela abaixo mostra algumas das ações de API das quais vimos clientes encontrarem limitação de solicitações. Você pode revisar os padrões do limite EC2 de taxa e as etapas para solicitar um aumento do limite de taxa na EC2 documentação sobre Rate Throttling.

Ações mutantes	Ações somente para leitura
AssignPrivateIpAddresses	DescribeDhcpOptions
AttachNetworkInterface	DescribeInstances
CreateNetworkInterface	DescribeNetworkInterfaces
DeleteNetworkInterface	DescribeSecurityGroups
DeleteTags	DescribeTags
DetachNetworkInterface	DescribeVpcs
ModifyNetworkInterfaceAttribute	DescribeVolumes
UnassignPrivateIpAddresses	

Outros limites conhecidos

- O Route 53 também tem um limite de taxa bastante baixo de 5 solicitações por segundo para a API do Route 53. Se você tiver um grande número de domínios para atualizar com um projeto como o DNS externo, poderá observar limitação de taxa e atrasos na atualização de domínios.

- Alguns tipos de instância Nitro têm um limite de anexação de volume de 28 que é compartilhado entre volumes, interfaces de rede e volumes de armazenamento de NVMe instâncias do Amazon EBS. Se suas cargas de trabalho estiverem montando vários volumes do EBS, você poderá encontrar limites na densidade de pods que você pode obter com esses tipos de instância.
- Há um número máximo de conexões que podem ser rastreadas por instância do Ec2. Se suas cargas de trabalho estiverem lidando com um grande número de conexões, você poderá ver falhas ou erros de comunicação porque esse máximo foi atingido. Você pode usar as métricas de desempenho da `conntrack_allowance_exceeded` rede `conntrack_allowance_available` e para monitorar o número de conexões rastreadas nos nós de trabalho do EKS.
- No ambiente EKS, o limite de armazenamento do etcd é de 8 GiB de acordo com a orientação upstream. Monitore `apiserver_storage_size_bytes` a métrica para rastrear o tamanho do banco de dados etcd. Você pode consultar as regras de alerta `etcdBackendQuotaLowSpace` e `etcdExcessiveDatabaseGrowth` configurar esse monitoramento.

Práticas recomendadas para upgrades de clusters

Este guia mostra aos administradores de cluster como planejar e executar sua estratégia de upgrade do Amazon EKS. Também descreve como atualizar nós autogerenciados, grupos de nós gerenciados, nós Karpenter e nós Fargate. Ela não inclui orientação sobre EKS Anywhere, Kubernetes autogerenciados, AWS Outposts ou AWS Local Zones.

Visão geral do

Uma versão do Kubernetes engloba tanto o plano de controle quanto o plano de dados. Para garantir uma operação tranquila, tanto o plano de controle quanto o plano de dados devem executar a mesma versão secundária do Kubernetes, como a 1.24. Embora a AWS gerencie e atualize o plano de controle, atualizar os nós de trabalho no plano de dados é de sua responsabilidade.

- Plano de controle — A versão do plano de controle é determinada pelo servidor da API Kubernetes. Nos clusters do Amazon EKS, a AWS se encarrega de gerenciar esse componente. As atualizações do plano de controle podem ser iniciadas por meio da API da AWS.
- Plano de dados — A versão do plano de dados está associada às versões do Kubelet em execução em seus nós individuais. É possível ter nós no mesmo cluster executando versões diferentes. Você pode verificar as versões de todos os nós executando`kubectl get nodes`.

Antes da atualização

Se você planeja atualizar sua versão do Kubernetes no Amazon EKS, existem algumas políticas, ferramentas e procedimentos importantes que você deve implementar antes de iniciar uma atualização.

- Entenda as políticas de suspensão de uso — Obtenha uma compreensão profunda de como a política de suspensão de uso do Kubernetes funciona. Esteja ciente de quaisquer mudanças futuras que possam afetar seus aplicativos existentes. As versões mais recentes do Kubernetes geralmente eliminam gradualmente determinados APIs recursos, o que pode causar problemas na execução de aplicativos.
- Analise o registro de alterações do Kubernetes — Analise minuciosamente o log de alterações do Kubernetes junto com as versões do Amazon EKS Kubernetes para entender qualquer possível impacto em seu cluster, como alterações significativas que possam afetar suas cargas de trabalho.

- Avalie a compatibilidade dos complementos do cluster — O Amazon EKS não atualiza automaticamente um complemento quando novas versões são lançadas ou depois que você atualiza seu cluster para uma nova versão secundária do Kubernetes. Analise a atualização de um complemento para entender a compatibilidade de qualquer complemento de cluster existente com a versão do cluster para a qual você pretende fazer o upgrade.
- Ativar registro do plano de controle — Ative o registro do plano de controle para capturar registros, erros ou problemas que possam surgir durante o processo de atualização. Considere revisar esses registros para detectar quaisquer anomalias. Teste as atualizações do cluster em um ambiente que não seja de produção ou integre testes automatizados em seu fluxo de trabalho de integração contínua para avaliar a compatibilidade de versões com seus aplicativos, controladores e integrações personalizadas.
- Explore o eksctl para gerenciamento de clusters — Considere usar o eksctl para gerenciar seu cluster EKS. Ele fornece a capacidade de atualizar o plano de controle, gerenciar complementos e lidar com atualizações out-of-the-box do nó de trabalho.
- Opte pelo modo automático ou grupos de nós gerenciados — Simplifique e automatize as atualizações dos nós de trabalho usando o Modo Automático ou os grupos de nós gerenciados pelo EKS. Essas opções simplificam o processo e reduzem a intervenção manual.
- Utilize o plug-in kubectl Convert — Aproveite o plug-in kubectl convert para facilitar a conversão dos arquivos de manifesto do Kubernetes entre diferentes versões da API. Isso pode ajudar a garantir que suas configurações permaneçam compatíveis com a nova versão do Kubernetes.

Mantenha seu cluster up-to-date

Manter-se atualizado com as atualizações do Kubernetes é fundamental para um ambiente EKS seguro e eficiente, refletindo o modelo de responsabilidade compartilhada no Amazon EKS. Ao integrar essas estratégias ao seu fluxo de trabalho operacional, você está se posicionando para manter up-to-date clusters seguros que aproveitam ao máximo os recursos e melhorias mais recentes. Táticas:

- Política de versão compatível — Alinhado com a comunidade Kubernetes, o Amazon EKS normalmente fornece três versões ativas do Kubernetes. Uma versão secundária do Kubernetes está sob suporte padrão no Amazon EKS nos primeiros 14 meses após seu lançamento. Depois que uma versão excede a data de término do suporte padrão, ela entra no suporte estendido pelos 12 meses seguintes. Os avisos de suspensão de uso são emitidos pelo menos 60 dias antes de uma versão atingir a data de fim do suporte padrão. Para obter mais detalhes, consulte a documentação do ciclo de vida da versão do EKS.

- Política de atualização automática — É altamente recomendável ficar em sincronia com as atualizações do Kubernetes em seu cluster EKS. Os clusters executados em uma versão do Kubernetes que completou seu ciclo de vida de 26 meses (14 meses de suporte padrão, mais 12 meses de suporte estendido) serão atualizados automaticamente para a próxima versão. Observe que você pode desativar o suporte estendido. A falha na atualização proativa antes que uma versão end-of-life acione uma atualização automática, o que pode interromper suas cargas de trabalho e sistemas. Para obter informações adicionais, consulte a versão EKS FAQs.
- Crie runbooks de atualização — Estabeleça um processo bem documentado para gerenciar atualizações. Como parte de sua abordagem proativa, desenvolva runbooks e ferramentas especializadas adaptadas ao seu processo de atualização. Isso não apenas melhora sua preparação, mas também simplifica transições complexas. Faça com que seja uma prática padrão atualizar seus clusters pelo menos uma vez por ano. Essa prática alinha você aos avanços tecnológicos contínuos, aumentando assim a eficiência e a segurança do seu ambiente.

Revise o calendário de lançamentos do EKS

Consulte o calendário de lançamentos do EKS Kubernetes para saber quando novas versões chegarão e quando o suporte para versões específicas terminará. Geralmente, o EKS lança três versões secundárias do Kubernetes anualmente, e cada versão secundária tem suporte por cerca de 14 meses.

Além disso, revise as informações de lançamento upstream do Kubernetes.

Entenda como o modelo de responsabilidade compartilhada se aplica às atualizações de clusters

Você é responsável por iniciar a atualização do plano de controle do cluster e do plano de dados. Saiba como iniciar um upgrade. Quando você inicia uma atualização de cluster, a AWS gerencia a atualização do plano de controle do cluster. Você é responsável por atualizar o plano de dados, incluindo os pods e complementos do Fargate. Você deve validar e planejar atualizações para cargas de trabalho em execução em seu cluster para garantir que sua disponibilidade e operações não sejam afetadas após a atualização do cluster.

Atualize clusters no local

O EKS oferece suporte a uma estratégia de upgrade de cluster no local. Isso mantém os recursos do cluster e mantém a configuração do cluster consistente (por exemplo, endpoint de API, OIDC e balanceadores de carga). ENIs Isso causa menos interrupções para os usuários do cluster e usará as cargas de trabalho e os recursos existentes no cluster sem exigir que você reimplante cargas de trabalho ou migre recursos externos (por exemplo, DNS, armazenamento).

Ao realizar uma atualização de cluster no local, é importante observar que somente uma atualização de versão secundária pode ser executada por vez (por exemplo, de 1,24 para 1,25).

Isso significa que, se você precisar atualizar várias versões, será necessária uma série de atualizações sequenciais. Planejar atualizações sequenciais é mais complicado e tem um risco maior de tempo de inatividade. Nessa situação, vejathe section called “Avalie Blue/Green os clusters como uma alternativa aos upgrades de clusters no local”.

Atualize seu plano de controle e plano de dados em sequência

Para atualizar um cluster, você precisará realizar as seguintes ações:

1. Analise as notas de lançamento do Kubernetes e do EKS.
2. Faça um backup do cluster. (opcional)
3. Identifique e corrija o uso obsoleto e removido da API em suas cargas de trabalho.
4. Certifique-se de que os grupos de nós gerenciados, se usados, estejam na mesma versão do Kubernetes do plano de controle. Os grupos de nós gerenciados pelo EKS e os nós criados pelos perfis EKS Fargate oferecem suporte a duas variações menores de versão entre o plano de controle e o plano de dados para o Kubernetes versão 1.27 e versões anteriores. A partir da versão 1.28 e versões posteriores, os grupos de nós gerenciados pelo EKS e os nós criados pelos Perfis Fargate do EKS suportam 3 variações menores de versão entre o plano de controle e o plano de dados. Por exemplo, se a versão do seu plano de controle EKS for 1.28, você poderá usar com segurança versões do kubelet tão antigas quanto 1.25. Se sua versão do EKS for 1.27, a versão mais antiga do kubelet que você pode usar é 1.25.
5. Atualize o plano de controle do cluster usando o console ou o CLI da AWS.
6. Analise a compatibilidade do complemento. Atualize seus complementos e controladores personalizados do Kubernetes, conforme necessário.
7. Atualize o kubectl.

8. Atualize o plano de dados do cluster. Atualize seus nós para a mesma versão secundária do Kubernetes do seu cluster atualizado.

> **Tip**
>
> Se seu cluster foi criado usando o EKS Auto Mode, você não precisa atualizar seu plano de dados de cluster. Depois de atualizar seu plano de controle, o EKS Auto Mode começará a atualizar incrementalmente os nós gerenciados, respeitando todos os orçamentos de interrupção do pod. Certifique-se de monitorar essas atualizações para verificar a conformidade com seus requisitos operacionais.

Use a documentação do EKS para criar uma lista de verificação de atualização

A documentação da versão do EKS Kubernetes inclui uma lista detalhada das alterações de cada versão. Crie uma lista de verificação para cada atualização.

Para obter orientações específicas sobre a atualização da versão do EKS, consulte a documentação para ver as alterações e considerações notáveis de cada versão.

- Analise as notas de lançamento das versões do Kubernetes no suporte padrão
- Analise as notas de lançamento das versões do Kubernetes sobre suporte estendido

Atualize complementos e componentes usando a API Kubernetes

Antes de atualizar um cluster, você deve entender quais versões dos componentes do Kubernetes você está usando. Faça o inventário dos componentes do cluster e identifique os componentes que usam a API Kubernetes diretamente. Isso inclui componentes essenciais do cluster, como agentes de monitoramento e registro, autoescaladores de cluster, drivers de armazenamento de contêineres (por exemplo, EBS CSI, EFS CSI), controladores de entrada e quaisquer outras cargas de trabalho ou complementos que dependam diretamente da API do Kubernetes.

> **Tip**
>
> Os componentes críticos do cluster geralmente são instalados em um `*-system` namespace

```
kubectl get ns | grep '-system'
```

Depois de identificar os componentes que dependem da API Kubernetes, verifique a documentação deles para verificar a compatibilidade de versões e os requisitos de atualização. Por exemplo, consulte a documentação do AWS Load Balancer Controller para ver a compatibilidade das versões. Alguns componentes podem precisar ser atualizados ou a configuração alterada antes de prosseguir com a atualização do cluster. Alguns componentes essenciais a serem verificados incluem CoreDNS, kube-proxy, VPC CNI e drivers de armazenamento.

Os clusters geralmente contêm muitas cargas de trabalho que usam a API Kubernetes e são necessárias para a funcionalidade da carga de trabalho, como controladores de entrada, sistemas de entrega contínua e ferramentas de monitoramento. Ao atualizar um cluster EKS, você também deve atualizar seus complementos e ferramentas de terceiros para garantir que sejam compatíveis.

Veja os seguintes exemplos de complementos comuns e sua documentação de atualização relevante:

- Amazon VPC CNI: para obter a versão recomendada do complemento Amazon VPC CNI para cada versão de cluster, consulte Atualização do plug-in CNI da Amazon VPC para o complemento autogerenciado do Kubernetes. Quando instalado como um complemento do Amazon EKS, ele só pode ser atualizado em uma versão secundária por vez.
- kube-proxy: consulte Atualização do complemento autogerenciado kube-proxy do Kubernetes.
- CoreDNS: consulte Atualização do complemento autogerenciado CoreDNS.
- Controlador do AWS Load Balancer: O controlador do AWS Load Balancer precisa ser compatível com a versão do EKS que você implantou. Consulte o guia de instalação para obter mais informações.
- Driver Amazon Elastic Block Store (Amazon EBS) Container Storage Interface (CSI): para obter informações de instalação e atualização, consulte Gerenciamento do driver CSI do Amazon EBS como um complemento do Amazon EKS.
- Driver da interface de armazenamento de contêiner (CSI) do Amazon Elastic File System (Amazon EFS): para obter informações de instalação e atualização, consulte o driver CSI do Amazon EFS.
- Kubernetes Metrics Server: para obter mais informações, consulte metrics-server em. GitHub
- Autoescalador de cluster do Kubernetes: para atualizar a versão do escalador automático do cluster do Kubernetes, altere a versão da imagem na implantação. O escalador automático de cluster é fortemente acoplado ao agendador do Kubernetes. Você sempre precisará atualizá-lo ao

atualizar o cluster. Analise as GitHub versões para encontrar o endereço da versão mais recente correspondente à sua versão secundária do Kubernetes.

- Karpenter: Para obter informações sobre instalação e atualização, consulte a documentação do Karpenter.

> **Tip**
>
> Você não precisa atualizar manualmente nenhum dos recursos do Amazon EKS Auto Mode, incluindo os recursos de escalonamento automático de computação, armazenamento em blocos e balanceamento de carga.

Verifique os requisitos básicos do EKS antes da atualização

A AWS exige certos recursos em sua conta para concluir o processo de atualização. Se esses recursos não estiverem presentes, o cluster não poderá ser atualizado. Uma atualização do plano de controle requer os seguintes recursos:

1. Endereços IP disponíveis: o Amazon EKS exige até cinco endereços IP disponíveis das sub-redes que você especificou ao criar o cluster para atualizar o cluster. Caso contrário, atualize sua configuração de cluster para incluir novas sub-redes de cluster antes de realizar a atualização da versão.
2. Função do EKS IAM: a função IAM do plano de controle ainda está presente na conta com as permissões necessárias.
3. Se o seu cluster tiver a criptografia secreta habilitada, certifique-se de que a função do IAM do cluster tenha permissão para usar a chave do AWS Key Management Service (AWS KMS).

Verifique os endereços IP disponíveis

Para atualizar o cluster, o Amazon EKS requer até cinco endereços IP disponíveis das sub-redes que você especificou ao criar o cluster.

Para verificar se suas sub-redes têm endereços IP suficientes para atualizar o cluster, você pode executar o seguinte 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 |
+--------------------------+--------------+-------+
```

O VPC CNI Metrics Helper pode ser usado para criar um painel CloudWatch para métricas de VPC. O Amazon EKS recomenda atualizar as sub-redes do cluster usando a API "UpdateClusterConfiguration" antes de iniciar a atualização da versão do Kubernetes se você estiver ficando sem endereços IP nas sub-redes inicialmente especificadas durante a criação do cluster. Verifique se as novas sub-redes que você receberá:

- pertencem ao mesmo conjunto dos AZs selecionados durante a criação do cluster.
- pertencem à mesma VPC fornecida durante a criação do cluster

Considere associar blocos CIDR adicionais se os endereços IP no bloco CIDR da VPC existente se esgotarem. A AWS permite a associação de blocos CIDR adicionais com seu cluster VPC existente, expandindo efetivamente seu pool de endereços IP. Essa expansão pode ser realizada com a introdução de intervalos IP privados adicionais (RFC 1918) ou, se necessário, intervalos de IP públicos (não RFC 1918). Você deve adicionar novos blocos CIDR da VPC e permitir que a atualização da VPC seja concluída antes que o Amazon EKS possa usar o novo CIDR. Depois disso, você pode atualizar as sub-redes com base nos blocos CIDR recém-configurados para a VPC.

Verifique a função do EKS IAM

Para verificar se a função do IAM está disponível e tem a política correta de assumir função em sua conta, você pode executar os seguintes 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"
        }
    ]
}
```

Migrar para complementos do EKS

O Amazon EKS instala automaticamente complementos como o plug-in CNI do Amazon VPC para Kubernetes e o CoreDNS para cada cluster. `kube-proxy` Os complementos podem ser autogerenciados ou instalados como complementos do Amazon EKS. Os complementos do Amazon EKS são uma forma alternativa de gerenciar complementos usando a API EKS.

Você pode usar os complementos do Amazon EKS para atualizar versões com um único comando. Por exemplo:

```
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
```

Verifique se você tem algum complemento do EKS com:

```
aws eks list-addons --cluster-name <cluster name>
```

> **Warning**
>
> Os complementos do EKS não são atualizados automaticamente durante uma atualização do plano de controle. Você deve iniciar as atualizações do complemento EKS e selecionar a versão desejada.

- Você é responsável por selecionar uma versão compatível de todas as versões disponíveis. Revise as orientações sobre compatibilidade de versões complementares.
- Os complementos do Amazon EKS só podem ser atualizados em uma versão secundária por vez.

Saiba mais sobre quais componentes estão disponíveis como complementos do EKS e como começar.

Saiba como fornecer uma configuração personalizada para um complemento EKS.

Identifique e corrija o uso removido da API antes de atualizar o plano de controle

Você deve identificar o uso da API removido APIs antes de atualizar seu plano de controle EKS. Para fazer isso, recomendamos o uso de ferramentas que possam verificar um cluster em execução ou arquivos de manifesto estáticos renderizados do Kubernetes.

A execução da verificação em arquivos de manifesto estáticos geralmente é mais precisa. Se executadas em clusters ativos, essas ferramentas podem retornar falsos positivos.

Uma API Kubernetes obsoleta não significa que a API tenha sido removida. Você deve verificar a Política de suspensão de uso do Kubernetes para entender como a remoção da API afeta suas cargas de trabalho.

Informações sobre clusters

O Cluster Insights é um recurso que fornece descobertas sobre problemas que podem afetar a capacidade de atualizar um cluster EKS para versões mais recentes do Kubernetes. Essas descobertas são organizadas e gerenciadas pelo Amazon EKS e oferecem recomendações sobre como corrigi-las. Ao aproveitar o Cluster Insights, você pode minimizar o esforço gasto para atualizar para versões mais recentes do Kubernetes.

Para ver insights de um cluster EKS, você pode executar o 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 obter uma saída mais descritiva sobre o insight recebido, você pode executar o comando:

```
aws eks describe-insight --region <region-code> --id <insight-id> --cluster-name <my-
cluster>
```

Você também tem a opção de visualizar insights no console do Amazon EKS. Depois de selecionar seu cluster na lista de clusters, as descobertas do insight estão localizadas abaixo da `Upgrade Insights` guia.

Se você encontrar um cluster insight com`"status": ERROR`, deverá resolver o problema antes de realizar a atualização do cluster. Execute o `aws eks describe-insight` comando que compartilhará o seguinte conselho de remediação:

Recursos afetados:

```
"resources": [
      {
        "insightStatus": {
          "status": "ERROR"
        },
        "kubernetesResourceUri": "/apis/policy/v1beta1/podsecuritypolicies/null"
      }
]
```

APIs obsoleto:

```
"deprecationDetails": [
      {
        "usage": "/apis/flowcontrol.apiserver.k8s.io/v1beta2/flowschemas",
        "replacedWith": "/apis/flowcontrol.apiserver.k8s.io/v1beta3/flowschemas",
        "stopServingVersion": "1.29",
        "clientStats": [],
        "startServingReplacementVersion": "1.26"
      }
]
```

Ação recomendada a ser tomada:

```
"recommendation": "Update manifests and API clients to use newer Kubernetes APIs if
 applicable before upgrading to Kubernetes v1.26."
```

A utilização de insights de cluster por meio do console EKS ou da CLI ajuda a acelerar o processo de atualização bem-sucedida das versões do cluster EKS. Saiba mais com os seguintes recursos: * Documentação oficial do EKS * Blog de lançamento do Cluster Insights.

K ube-no-trouble

K ube-no-trouble é um utilitário de linha de comando de código aberto com o comando`kubent`. Quando você executa `kubent` sem nenhum argumento, ele usa seu KubeConfig contexto atual, verifica o cluster e imprime um relatório com o que APIs será descontinuado e removido.

```
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)
```

Ele também pode ser usado para verificar arquivos de manifesto estáticos e pacotes helm. É recomendável executá-lo `kubent` como parte de um processo de integração contínua (CI) para identificar problemas antes que os manifestos sejam implantados. O escaneamento de manifestos também é mais preciso do que o escaneamento de clusters ativos.

Kube-no-trouble fornece um exemplo de conta e função de serviço com as permissões apropriadas para escanear o cluster.

Plutão

Outra opção é o pluto, que é semelhante `kubent` porque suporta a digitalização de um cluster ativo, arquivos de manifesto, gráficos de leme e tem uma GitHub ação que você pode incluir em seu processo 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 verificar se seu cluster não usa obsoleto APIs antes da atualização, você deve monitorar:

- métrica `apiserver_requested_deprecated_apis` desde o 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 nos registros de auditoria com `k8s.io/deprecated` definido como`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
```

O que produzirá linhas se as obsoletas APIs estiverem em uso:

```
{
    "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\"}}"
            },
[...]
```

Atualize as cargas de trabalho do Kubernetes. Use kubectl-convert para atualizar manifestos

Depois de identificar quais cargas de trabalho e manifestos precisam ser atualizados, talvez seja necessário alterar o tipo de recurso em seus arquivos de manifesto (por exemplo,

PodSecurityPolicies para PodSecurityStandards). Isso exigirá a atualização da especificação do recurso e pesquisas adicionais, dependendo do recurso que está sendo substituído.

Se o tipo de recurso permanecer o mesmo, mas a versão da API precisar ser atualizada, você poderá usar o `kubectl-convert` comando para converter automaticamente seus arquivos de manifesto. Por exemplo, para converter uma implantação antiga em`apps/v1`. Para obter mais informações, consulte Instalar o plug-in kubectl convert no site do Kubernetes.

```
kubectl-convert -f <file> --output-version <group>/<version>
```

Configure PodDisruptionBudgets e garanta topologySpreadConstraints a disponibilidade de suas cargas de trabalho enquanto o plano de dados é atualizado

Garanta que suas cargas de trabalho estejam adequadas PodDisruptionBudgetse garanta topologySpreadConstraintsa disponibilidade de suas cargas de trabalho enquanto o plano de dados é atualizado. Nem toda carga de trabalho exige o mesmo nível de disponibilidade, então você precisa validar a escala e os requisitos de sua carga de trabalho.

Certifique-se de que as cargas de trabalho estejam distribuídas em várias zonas de disponibilidade e em vários hosts com distribuições de topologia proporcionará um maior nível de confiança de que as cargas de trabalho migrarão para o novo plano de dados automaticamente sem incidentes.

Aqui está um exemplo de carga de trabalho que sempre terá 80% das réplicas disponíveis e distribuirá as réplicas entre zonas e 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
```

O AWS Resilience Hub adicionou o Amazon Elastic Kubernetes Service (Amazon EKS) como um recurso compatível. O Resilience Hub fornece um único local para definir, validar e rastrear a resiliência de seus aplicativos para que você possa evitar o tempo de inatividade desnecessário causado por interrupções de software, infraestrutura ou operações.

Use grupos de nós gerenciados ou Karpenter para simplificar as atualizações do plano de dados

Tanto o Managed Node Groups quanto o Karpenter simplificam as atualizações de nós, mas adotam abordagens diferentes.

Os grupos de nós gerenciados automatizam o provisionamento e o gerenciamento do ciclo de vida dos nós. Isso significa que você pode criar, atualizar ou encerrar nós automaticamente com uma única operação.

Na configuração padrão, o Karpenter cria automaticamente novos nós usando a AMI otimizada EKS mais recente compatível. À medida que o EKS lança o EKS Optimized atualizado AMIs ou o cluster é atualizado, o Karpenter começará automaticamente a usar essas imagens. O Karpenter também implementa o Node Expiry para atualizar os nós.

O Karpenter pode ser configurado para ser personalizado. AMIs Se você usa custom AMIs com o Karpenter, você é responsável pela versão do kubelet.

Confirme a compatibilidade da versão com os nós existentes e o plano de controle

Antes de prosseguir com uma atualização do Kubernetes no Amazon EKS, é vital garantir a compatibilidade entre seus grupos de nós gerenciados, nós autogerenciados e o plano de controle. A compatibilidade é determinada pela versão do Kubernetes que você está usando e varia de acordo com diferentes cenários. Táticas:

- Kubernetes v1.28+ — * * A partir da versão 1.28 do Kubernetes em diante, há uma política de versão mais branda para os componentes principais. Especificamente, a distorção suportada entre o servidor da API Kubernetes e o kubelet foi ampliada em uma versão secundária, passando de n-2 para n-3. Por exemplo, se a versão do seu plano de controle EKS for 1.28, você poderá usar com segurança versões do kubelet tão antigas quanto 1.25. Essa distorção de versão é compatível com o AWS Fargate, grupos de nós gerenciados e nós autogerenciados. É altamente recomendável manter suas versões do Amazon Machine Image (AMI) up-to-date por motivos de segurança. Versões mais antigas do kubelet podem representar riscos de segurança devido a possíveis vulnerabilidades e exposições comuns (CVEs), que podem superar os benefícios de usar versões mais antigas do kubelet.
- Kubernetes < v1.28 — Se você estiver usando uma versão anterior à v1.28, a distorção suportada entre o servidor da API e o kubelet é n-2. Por exemplo, se sua versão do EKS for 1.27, a versão mais antiga do kubelet que você pode usar é 1.25. Essa distorção de versão é aplicável em todo o AWS Fargate, grupos de nós gerenciados e nós autogerenciados.

Habilitar a expiração de nós para nós gerenciados pelo Karpenter

Uma maneira pela qual o Karpenter implementa atualizações de nós é usando o conceito de expiração de nós. Isso reduz o planejamento necessário para atualizações de nós. Quando você define um valor para ttlSecondsUntilExpirado em seu provisionador, isso ativa a expiração do nó. Depois que os nós atingem a idade definida em segundos, eles são drenados e excluídos com segurança. Isso é verdade mesmo se eles estiverem em uso, permitindo que você substitua os nós por instâncias atualizadas recém-provisionadas. Quando um nó é substituído, o Karpenter usa a versão mais recente otimizada para EKS. AMIs Para obter mais informações, consulte Desprovisionamento no site da Karpenter.

O Karpenter não adiciona automaticamente instabilidade a esse valor. Para evitar interrupções excessivas na carga de trabalho, defina um orçamento de interrupção do pod, conforme mostrado na documentação do Kubernetes.

Se você configurar ttlSecondsUntilExpirado em um provisionador, isso se aplicará aos nós existentes associados ao provisionador.

Use o recurso Drift para nós gerenciados pelo Karpenter

O recurso Drift do Karpenter pode atualizar automaticamente os nós provisionados pelo Karpenter para permanecerem sincronizados com o plano de controle do EKS. Atualmente, o Karpenter Drift precisa ser ativado usando um portão de recursos. A configuração padrão do Karpenter usa a AMI otimizada para EKS mais recente para a mesma versão principal e secundária do plano de controle do cluster EKS.

Depois que uma atualização do EKS Cluster for concluída, o recurso Drift do Karpenter detectará que os nós provisionados pelo Karpenter estão usando o EKS otimizado AMIs para a versão anterior do cluster e automaticamente isolará, drenará e substituirá esses nós. Para apoiar a migração de pods para novos nós, siga as melhores práticas do Kubernetes definindo cotas de recursos apropriadas e usando orçamentos de interrupção de pods (PDB). O desprovisionamento do Karpenter pré-ativará os nós de substituição com base nas solicitações de recursos do pod e os respeitará ao desprovisionar os nós. PDBs

Use eksctl para automatizar atualizações para grupos de nós autogerenciados

Grupos de nós autogerenciados são EC2 instâncias que foram implantadas em sua conta e anexadas ao cluster fora do serviço EKS. Eles geralmente são implantados e gerenciados por alguma forma de ferramenta de automação. Para atualizar grupos de nós autogerenciados, você deve consultar a documentação de suas ferramentas.

Por exemplo, o eksctl suporta a exclusão e drenagem de nós autogerenciados.

Algumas ferramentas comuns incluem:

- eksctl
- KOPs
- Projetos do EKS

Faça backup do cluster antes de fazer o upgrade

Novas versões do Kubernetes introduzem mudanças significativas em seu cluster Amazon EKS. Depois de atualizar um cluster, você não pode fazer o downgrade dele.

O Velero é uma ferramenta de código aberto suportada pela comunidade que pode ser usada para fazer backups de clusters existentes e aplicar os backups em um novo cluster.

Observe que você só pode criar novos clusters para as versões do Kubernetes atualmente compatíveis com o EKS. Se a versão que seu cluster está executando atualmente ainda for suportada e uma atualização falhar, você poderá criar um novo cluster com a versão original e restaurar o plano de dados. Observe que os recursos da AWS, incluindo o IAM, não estão incluídos no backup do Velero. Esses recursos precisariam ser recriados.

Reinicie as implantações do Fargate após atualizar o plano de controle

Para atualizar os nós do plano de dados do Fargate, você precisa reimplantar as cargas de trabalho. Você pode identificar quais cargas de trabalho estão sendo executadas nos nós do Fargate listando todos os pods com a opção. `-o wide` Qualquer nome de nó que comece com `fargate-` precisará ser reimplantado no cluster.

Avalie Blue/Green os clusters como uma alternativa aos upgrades de clusters no local

Alguns clientes preferem fazer uma estratégia de blue/green upgrade. Isso pode trazer benefícios, mas também inclui desvantagens que devem ser consideradas.

Os benefícios são:

- Possível alterar várias versões do EKS de uma só vez (por exemplo, 1,23 a 1,25)
- Capaz de voltar para o cluster antigo
- Cria um novo cluster que pode ser gerenciado com sistemas mais novos (por exemplo, terraform)
- As cargas de trabalho podem ser migradas individualmente

Algumas desvantagens incluem:

- Alteração no endpoint da API e no OIDC que requer a atualização dos consumidores (por exemplo, kubectl e CI/CD)
- Requer que dois clusters sejam executados paralelamente durante a migração, o que pode ser caro e limitar a capacidade da região
- É necessária mais coordenação se as cargas de trabalho dependerem umas das outras para serem migradas em conjunto
- Os balanceadores de carga e o DNS externo não podem facilmente abranger vários clusters

Embora essa estratégia seja possível, ela é mais cara do que uma atualização local e requer mais tempo para coordenação e migrações de carga de trabalho. Pode ser necessário em algumas situações e deve ser planejado com cuidado.

Com altos graus de automação e sistemas declarativos GitOps, isso pode ser mais fácil de fazer. Você precisará tomar precauções adicionais para cargas de trabalho com estado, para que os dados sejam copiados e migrados para novos clusters.

Revise essas postagens do blog para obter mais informações:

- Atualização do cluster Kubernetes: a estratégia de implantação azul-verde
- Migração de clusters Amazon EKS azul/verde ou canário para cargas de trabalho ArgoCD sem estado

Acompanhe as principais mudanças planejadas no projeto Kubernetes — Pense no futuro

Não veja apenas a próxima versão. Analise as novas versões do Kubernetes à medida que elas são lançadas e identifique as principais mudanças. Por exemplo, alguns aplicativos usaram diretamente a API docker, e o suporte para Container Runtime Interface (CRI) para Docker (também conhecido como Dockershim) foi removido no Kubernetes. `1.24` Esse tipo de mudança requer mais tempo para se preparar.

Analise todas as alterações documentadas da versão para a qual você está atualizando e anote todas as etapas de atualização necessárias. Além disso, observe todos os requisitos ou procedimentos específicos dos clusters gerenciados do Amazon EKS.

- Registro de alterações do Kubernetes

Orientação específica sobre remoções de recursos

Remoção do Dockershim na versão 1.25 - Use o detector para Docker Socket (DDS)

A AMI otimizada para EKS para 1.25 não inclui mais suporte para Dockershim. Se você tem uma dependência do Dockershim, por exemplo, está montando o soquete Docker, precisará remover essas dependências antes de atualizar seus nós de trabalho para 1,25.

Encontre instâncias em que você dependa do soquete Docker antes de atualizar para a versão 1.25. Recomendamos usar o Detector for Docker Socket (DDS), um plug-in kubectl. .

Remoção da PodSecurityPolicy versão 1.25 - Migre para os padrões de segurança do Pod ou uma solução policy-as-code

`PodSecurityPolicy`foi descontinuado no Kubernetes 1.21 e foi removido no Kubernetes 1.25. Se você estiver usando PodSecurityPolicy em seu cluster, deverá migrar para os padrões de segurança (PSS) integrados do Kubernetes Pod ou para uma policy-as-code solução antes de atualizar seu cluster para a versão 1.25 para evitar interrupções nas cargas de trabalho.

A AWS publicou um FAQ detalhado na documentação do EKS.

Analise as melhores práticas dos Padrões de Segurança do Pod (PSS) e do Pod Security Admission (PSA).

Leia a postagem do blog sobre PodSecurityPolicy Depreciação no site do Kubernetes.

Depreciação do driver de armazenamento em árvore na versão 1.23 - Migração para drivers de interface de armazenamento de contêiner (CSI)

A Container Storage Interface (CSI) foi projetada para ajudar o Kubernetes a substituir seus mecanismos existentes de driver de armazenamento em árvore. O recurso de migração Container Storage Interface (CSI) do Amazon EBS é habilitado por padrão em clusters `1.23` e posteriores do Amazon EKS. Se você tiver pods em execução em uma versão `1.22` ou em um cluster anterior, deverá instalar o driver CSI do Amazon EBS antes de atualizar seu cluster para a versão `1.23` para evitar a interrupção do serviço.

Analise as perguntas frequentes sobre a migração do Amazon EBS CSI.

Recursos adicionais

ClowdHaus Orientação de atualização do EKS

ClowdHaus O EKS Upgrade Guidance é uma CLI para ajudar na atualização dos clusters do Amazon EKS. Ele pode analisar um cluster em busca de possíveis problemas a serem corrigidos antes da atualização.

GoNoGo

GoNoGoé uma ferramenta em estágio alfa para determinar a confiança na atualização de seus complementos de cluster.

Melhores práticas para otimização de custos

Otimização de custos é alcançar seus resultados comerciais com o menor preço. Ao seguir a documentação deste guia, você otimizará suas cargas de trabalho do Amazon EKS.

Diretrizes gerais

Na nuvem, há várias diretrizes gerais que podem ajudar você a otimizar os custos de seus microsserviços:

- Garanta que as cargas de trabalho executadas no Amazon EKS sejam independentes dos tipos de infraestrutura específicos para executar seus contêineres. Isso proporcionará maior flexibilidade em relação à execução deles nos tipos de infraestrutura mais baratos. Ao usar o Amazon EKS com EC2, pode haver exceções quando temos cargas de trabalho que exigem tipos específicos de tipos de EC2 instância, como exigir uma GPU ou outros tipos de instância, devido à natureza da carga de trabalho.
- Selecione instâncias de contêiner com perfil ideal — crie um perfil de seus ambientes de produção ou pré-produção e monitore métricas críticas, como CPU e memória, usando serviços como o Amazon CloudWatch Container Insights para Amazon EKS ou ferramentas de terceiros que estão disponíveis no ecossistema Kubernetes. Isso garantirá que possamos alocar a quantidade certa de recursos e evitar o desperdício de recursos.
- Aproveite as diferentes opções de compra disponíveis na AWS para executar o EKS com EC2, por exemplo, On-Demand, Spot e Savings Plan.

Melhores práticas de otimização de custos do EKS

Há três áreas gerais de melhores práticas para otimização de custos na nuvem:

- Recursos econômicos (Auto Scaling, Down Scaling, políticas e opções de compra)
- Conscientização de despesas (usando a AWS e ferramentas de terceiros)
- Otimizando ao longo do tempo (tamanho certo)

Como acontece com qualquer orientação, existem vantagens e desvantagens. Certifique-se de trabalhar com sua organização para entender as prioridades dessa carga de trabalho e quais práticas recomendadas são mais importantes.

Como usar este guia

Este guia é destinado às equipes de DevOps responsáveis por implementar e gerenciar os clusters EKS e as cargas de trabalho que eles suportam. O guia está organizado em diferentes áreas de melhores práticas para facilitar o consumo. Cada tópico tem uma lista de recomendações, ferramentas a serem usadas e melhores práticas para otimização de custos de seus clusters EKS. Os tópicos não precisam ser lidos em uma ordem específica.

Principais recursos dos serviços da AWS e do Kubernetes

A otimização de custos é suportada pelos seguintes serviços e recursos da AWS:

- EC2 Tipos de instância, Savings Plan (e Instâncias Reservadas) e Instâncias Spot, a preços diferentes.
- Auto Scaling junto com as políticas nativas de Auto Scaling do Kubernetes. Considere o Savings Plan (instâncias previamente reservadas) para cargas de trabalho previsíveis. Use armazenamentos de dados gerenciados, como EBS e EFS, para obter elasticidade e durabilidade dos dados do aplicativo.
- O painel do console Billing and Cost Management, juntamente com o AWS Cost Explorer, fornece uma visão geral do seu uso da AWS. Use o AWS Organizations para obter detalhes granulares de faturamento. Detalhes de várias ferramentas de terceiros também foram compartilhados.
- O Amazon CloudWatch Container Metrics fornece métricas sobre o uso de recursos pelo cluster EKS. Além do painel do Kubernetes, há várias ferramentas no ecossistema Kubernetes que podem ser usadas para reduzir o desperdício.

Este guia inclui um conjunto de recomendações que você pode usar para melhorar a otimização de custos do seu cluster Amazon EKS.

Feedback

Este guia está sendo lançado GitHub para coletar feedback direto e sugestões da comunidade mais ampla do EKS/Kubernetes. Se você tem uma prática recomendada que acha que devemos incluir no guia, registre um problema ou envie um PR no GitHub repositório. Nossa intenção é atualizar o guia periodicamente à medida que novos recursos são adicionados ao serviço ou quando uma nova prática recomendada evolui.

Estrutura de otimização de custos

A AWS Cloud Economics é uma disciplina que ajuda os clientes a aumentar a eficiência e reduzir seus custos por meio da adoção de tecnologias computacionais modernas, como o Amazon EKS. A disciplina recomenda seguir uma metodologia chamada “estrutura de gerenciamento financeiro em nuvem (CFM)”, que consiste em 4 pilares:



O pilar See: mensuração e responsabilidade

O pilar See é um conjunto básico de atividades e tecnologias que definem como medir, monitorar e criar responsabilidade pelos gastos com a nuvem. Geralmente é chamado de “Observabilidade”, “Instrumentação” ou “Telemetria”. As capacidades e limitações da infraestrutura de “Observabilidade” determinam o que pode ser otimizado. Obter uma visão clara de seus custos é um primeiro passo fundamental na otimização de custos, pois você precisa saber de onde está começando. Esse tipo de visibilidade também orientará os tipos de atividades que você precisará realizar para otimizar ainda mais seu ambiente.

Aqui está uma breve visão geral de nossas melhores práticas para o pilar See:

- Defina e mantenha uma estratégia de marcação para suas cargas de trabalho.
  - Use a marcação de instâncias, a marcação de clusters EKS permite que você veja os custos individuais do cluster e os aloque em seus relatórios de custo e uso.
- Estabeleça relatórios e monitoramento do uso do EKS usando tecnologias como o Kubecost.

  - Ative os painéis de inteligência em nuvem. Com os recursos devidamente marcados e usando visualizações, você pode medir e estimar os custos.
- Aloque os custos da nuvem para aplicativos, linhas de negócios (LoBs) e fluxos de receita.
- Defina, meça e divulgue efficiency/value KPIs com as partes interessadas do negócio. Por exemplo, crie um KPI de “métrica unitária” que meça o custo por transação. Por exemplo, um serviço de compartilhamento de caronas pode ter um KPI para “custo por viagem”.

Para obter mais detalhes sobre as tecnologias e atividades recomendadas associadas a esse pilar, consulte a seção Otimização de custos - Observabilidade deste guia.

O pilar Economizar: Otimização de custos

Esse pilar é baseado nas tecnologias e capacidades desenvolvidas no pilar “Ver”. As seguintes atividades normalmente se enquadram nesse pilar:

- Identifique e elimine o desperdício em seu ambiente.
- Arquiteto e design para eficiência de custos.
- Escolha a melhor opção de compra, por exemplo, instâncias sob demanda versus instâncias spot.
- Adapte-se à medida que os serviços evoluem: à medida que os serviços da AWS evoluem, a forma de usar esses serviços com eficiência pode mudar. Esteja disposto a se adaptar para levar em conta essas mudanças.

Como essas atividades são operacionais, elas dependem muito das características do seu ambiente. Pergunte a si mesmo: quais são os principais fatores de custos? Que valor comercial seus diferentes ambientes oferecem? Quais opções de compra e opções de infraestrutura, por exemplo, tipos de família de instâncias, são mais adequadas para cada ambiente?

Abaixo está uma lista priorizada dos fatores de custo mais comuns para clusters EKS:

1. Custos de computação: combinar vários tipos de famílias de instâncias, opções de compra e equilibrar a escalabilidade com a disponibilidade exigem uma análise cuidadosa. Para obter mais informações, consulte as recomendações na seção Otimização de custos - Computação deste guia.
2. Custos de rede: usar 3 AZs para clusters EKS pode aumentar potencialmente os custos de tráfego entre AZ. Para nossas recomendações sobre como equilibrar os requisitos de HA com a redução dos custos de tráfego de rede, consulte a seção Otimização de custos - Rede deste guia.

3. Custos de armazenamento: dependendo da stateful/stateless natureza das cargas de trabalho nos clusters EKS e de como os diferentes tipos de armazenamento são usados, o armazenamento pode ser considerado parte da carga de trabalho. Para considerações relacionadas aos custos de armazenamento do EKS, consulte a seção Otimização de custos - Armazenamento deste guia.

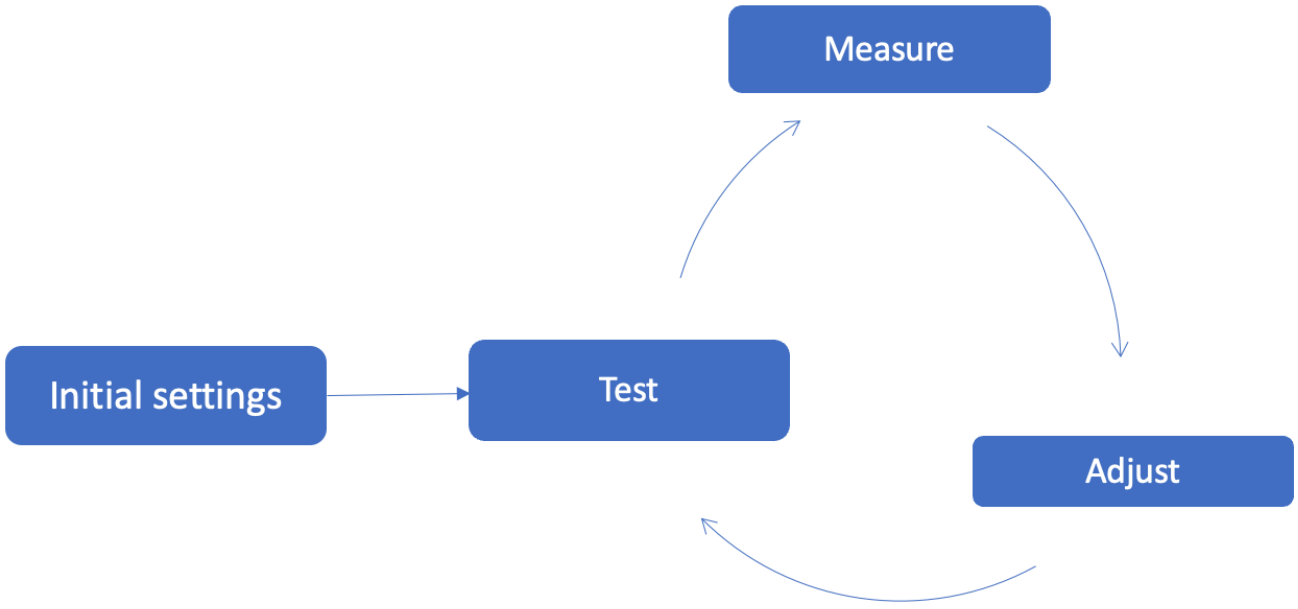
O pilar do Plano: planejamento e previsão

Depois que as recomendações do pilar See são implementadas, os clusters são otimizados continuamente. À medida que a experiência é adquirida na operação eficiente de clusters, as atividades de planejamento e previsão podem se concentrar em:

- Orçamento e previsão de custos de nuvem de forma dinâmica.
- Quantificando o valor comercial fornecido pelos serviços de contêineres da EKS.
- Integrando o gerenciamento de custos do cluster EKS com o planejamento de gerenciamento financeiro de TI.

O pilar Run

A otimização de custos é um processo contínuo e envolve uma série de melhorias incrementais:



Garantir o patrocínio executivo para esses tipos de atividades é crucial para integrar a otimização do cluster EKS aos esforços "" da organização. FinOps Ele permite o alinhamento das partes interessadas por meio de uma compreensão compartilhada dos custos do cluster EKS, da implementação das grades de proteção de custos do cluster EKS e da garantia de que as ferramentas, a automação e as atividades evoluam de acordo com as necessidades da organização.

Referências

- Economia da nuvem da AWS, gerenciamento financeiro da nuvem

Conscientização sobre despesas

Conscientizar as despesas é entender quem, onde e o que está causando despesas em seu cluster EKS. Obter uma imagem precisa desses dados ajudará a aumentar a conscientização sobre seus gastos e destacar áreas a serem corrigidas.

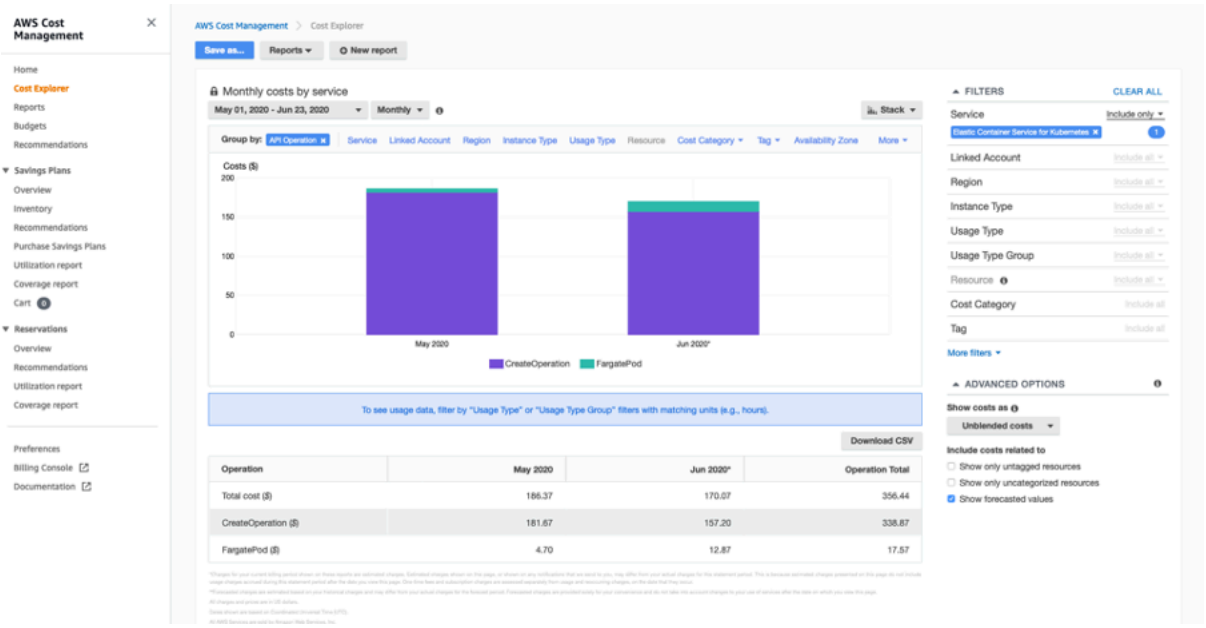
Recomendações

Use o Cost Explorer

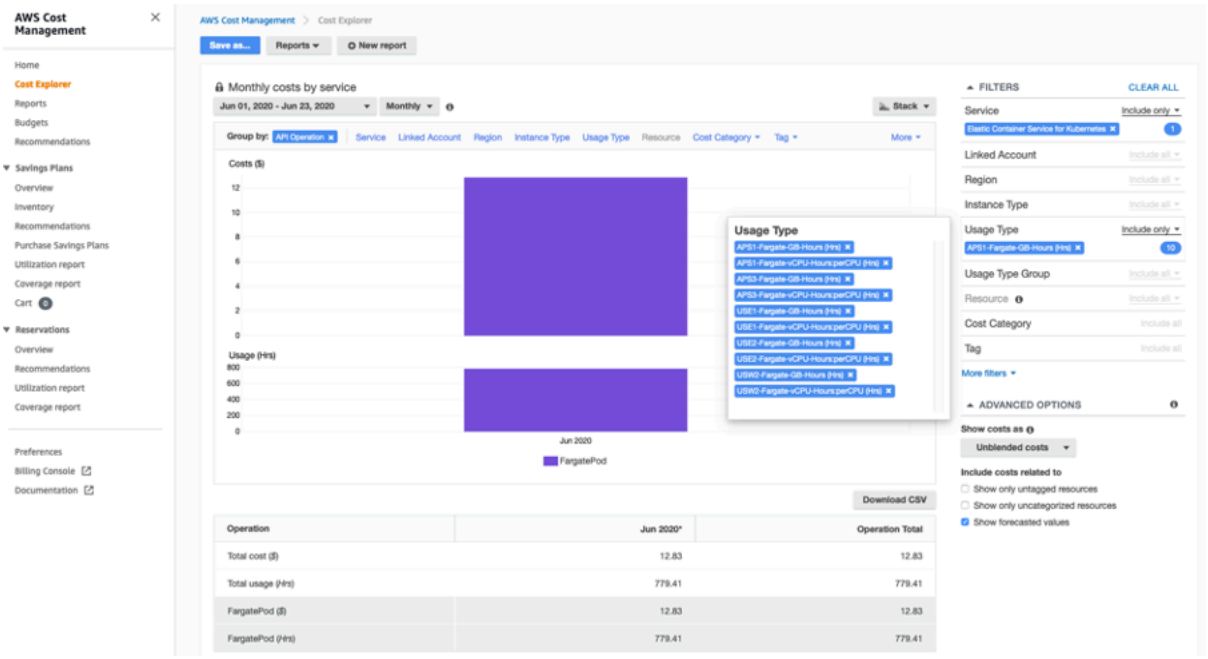
O AWS Cost Explorer tem uma easy-to-use interface que permite visualizar, entender e gerenciar seus custos e uso da AWS ao longo do tempo. Você pode analisar dados de custo e uso, em vários níveis, usando os filtros disponíveis no Cost Explorer.

Custos do plano de controle EKS e do EKS Fargate

Usando os filtros, podemos consultar os custos incorridos com os custos do EKS no Plano de Controle e no Fargate Pod, conforme mostrado no diagrama abaixo:



Usando os filtros, podemos consultar os custos agregados dos Fargate Pods em todas as regiões do EKS, o que inclui horas de vCPU por CPU e GB por hora, conforme mostrado no diagrama abaixo:



Marcação de recursos

O Amazon EKS oferece suporte à adição de tags da AWS aos seus clusters do Amazon EKS. Isso facilita o controle do acesso à API EKS para gerenciar seus clusters. As tags adicionadas a um cluster do EKS são específicas do recurso de cluster do AWS EKS e não se propagam para outros recursos da AWS usados pelo cluster, como EC2 instâncias ou balanceadores de carga. Atualmente,

a marcação de clusters é compatível com todos os clusters EKS novos e existentes por meio da API, console e. SDKs

O AWS Fargate é uma tecnologia que fornece capacidade computacional sob demanda e do tamanho certo para contêineres. Antes de agendar pods no Fargate no cluster, é necessário definir pelo menos um perfil do Fargate que especifique quais pods devem usá-lo quando forem executados.

Adicionar e listar tags a um cluster EKS:

```
$ aws eks tag-resource --resource-arn arn:aws:eks:us-west-2:xxx:cluster/ekscluster1 --
tags team=devops,env=staging,bu=cio,costcenter=1234
$ aws eks list-tags-for-resource --resource-arn arn:aws:eks:us-west-2:xxx:cluster/
ekscluster1
{
    "tags": {
        "bu": "cio",
        "env": "staging",
        "costcenter": "1234",
        "team": "devops"
    }
}
```

Depois de ativar as tags de alocação de custos no AWS Cost Explorer, a AWS usa as tags de alocação de custos para organizar seus custos de recursos em seu relatório de alocação de custos, para facilitar a categorização e o controle de seus custos da AWS.

As etiquetas não têm significado semântico no Amazon EKS e são interpretadas estritamente como uma sequência de caracteres. Por exemplo, você pode definir um conjunto de etiquetas para os clusters do Amazon EKS para ajudar a monitorar o proprietário e o nível da pilha de cada cluster.

Use o AWS Trusted Advisor

O AWS Trusted Advisor oferece um rico conjunto de verificações e recomendações de melhores práticas em cinco categorias: otimização de custos; segurança; tolerância a falhas; desempenho; e limites de serviço.

Para otimização de custos, o Trusted Advisor ajuda a eliminar recursos não utilizados e ociosos e recomenda assumir compromissos com a capacidade reservada. Os principais itens de ação que ajudarão o Amazon EKS envolverão EC2 instâncias pouco utilizadas, endereços IP elásticos não associados, balanceadores de carga ociosos, volumes subutilizados do EBS, entre outras coisas. A

lista completa de verificações é fornecida em https://aws.amazon.com/premiumsupport/ technology/ trusted-advisor/best -practice-checklist/.

O Trusted Advisor também fornece recomendações de Savings Plans e Reserved EC2 Instances para instâncias e Fargate, o que permite que você se comprometa com um valor de uso consistente em troca de tarifas com desconto.

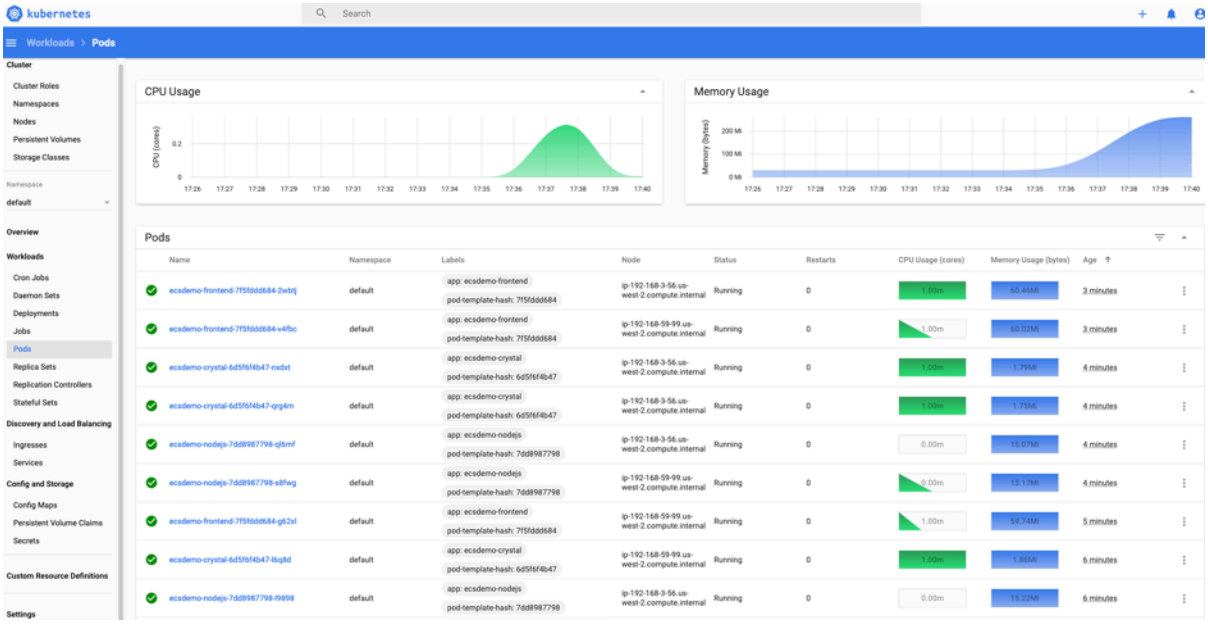
> **Note**
>
> As recomendações do Trusted Advisor são recomendações genéricas e não específicas do EKS.

Use o painel do Kubernetes

Painel do Kubernetes

O Kubernetes Dashboard é uma interface de usuário baseada na web de uso geral para clusters Kubernetes, que fornece informações sobre o cluster Kubernetes, incluindo o uso de recursos em nível de cluster, nó e pod. A implantação do painel do Kubernetes em um cluster do Amazon EKS é descrita na documentação do Amazon EKS.

O painel fornece detalhamentos do uso de recursos para cada nó e pod, bem como metadados detalhados sobre pods, serviços, implantações e outros objetos do Kubernetes. Essas informações consolidadas fornecem visibilidade do seu ambiente Kubernetes.



comandos kubectl top e describe

Visualizando métricas de uso de recursos com os comandos kubectl top e kubectl describe. O kubectl top mostrará o uso atual da CPU e da memória para os pods ou nós em todo o cluster ou para um pod ou nó específico. O comando kubectl describe fornecerá informações mais detalhadas sobre um nó ou pod específico.

```
$ kubectl top pods
$ kubectl top nodes
$ kubectl top pod pod-name --namespace mynamespace --containers
```

Usando o comando top, a saída exibirá a quantidade total de CPU (em núcleos) e memória (em MiB) que o nó está usando e as porcentagens da capacidade alocável do nó que esses números representam. Em seguida, você pode detalhar até o próximo nível, o nível do contêiner dentro dos pods, adicionando uma sinalização --containers.

```
$ kubectl describe node <node>
$ kubectl describe pod <pod>
```

kubectl describe retorna a porcentagem da capacidade total disponível que cada solicitação ou limite de recurso representa.

kubectl top e descreva, acompanhe a utilização e a disponibilidade de recursos essenciais, como CPU, memória e armazenamento em pods, nós e contêineres do Kubernetes. Essa conscientização ajudará a entender o uso de recursos e a controlar os custos.

Use o CloudWatch Container Insights

Use o CloudWatch Container Insights para coletar, agregar e resumir métricas e registros de seus aplicativos e microsserviços em contêineres. O Container Insights está disponível para o Amazon Elastic Kubernetes Service EC2 no Amazon e para as plataformas Kubernetes na Amazon. EC2 As métricas incluem a utilização de recursos, como CPU, memória, disco e rede.

A instalação dos insights é fornecida na documentação.

CloudWatch cria métricas agregadas no nível de cluster, nó, pod, tarefa e serviço como CloudWatch métricas.

A consulta a seguir mostra uma lista de nós, ordenada pela utilização média da CPU do nó

```
STATS avg(node_cpu_utilization) as avg_node_cpu_utilization by NodeName
| SORT avg_node_cpu_utilization DESC
```

Uso da CPU por nome do contêiner

```
stats pct(container_cpu_usage_total, 50) as CPUPercMedian by kubernetes.container_name
| filter Type="Container"
```

Uso do disco por nome do contêiner

```
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
```

Mais exemplos de consultas são fornecidos na documentação do Container Insights

Essa conscientização ajudará a entender o uso de recursos e a controlar os custos.

Usando o Kubecost para conscientização e orientação sobre despesas

Ferramentas de terceiros, como o kubecost, também podem ser implantadas no Amazon EKS para obter visibilidade do custo de execução do seu cluster Kubernetes. Consulte este blog da AWS para monitorar os custos usando o Kubecost

Implantando o kubecost usando o 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"
```

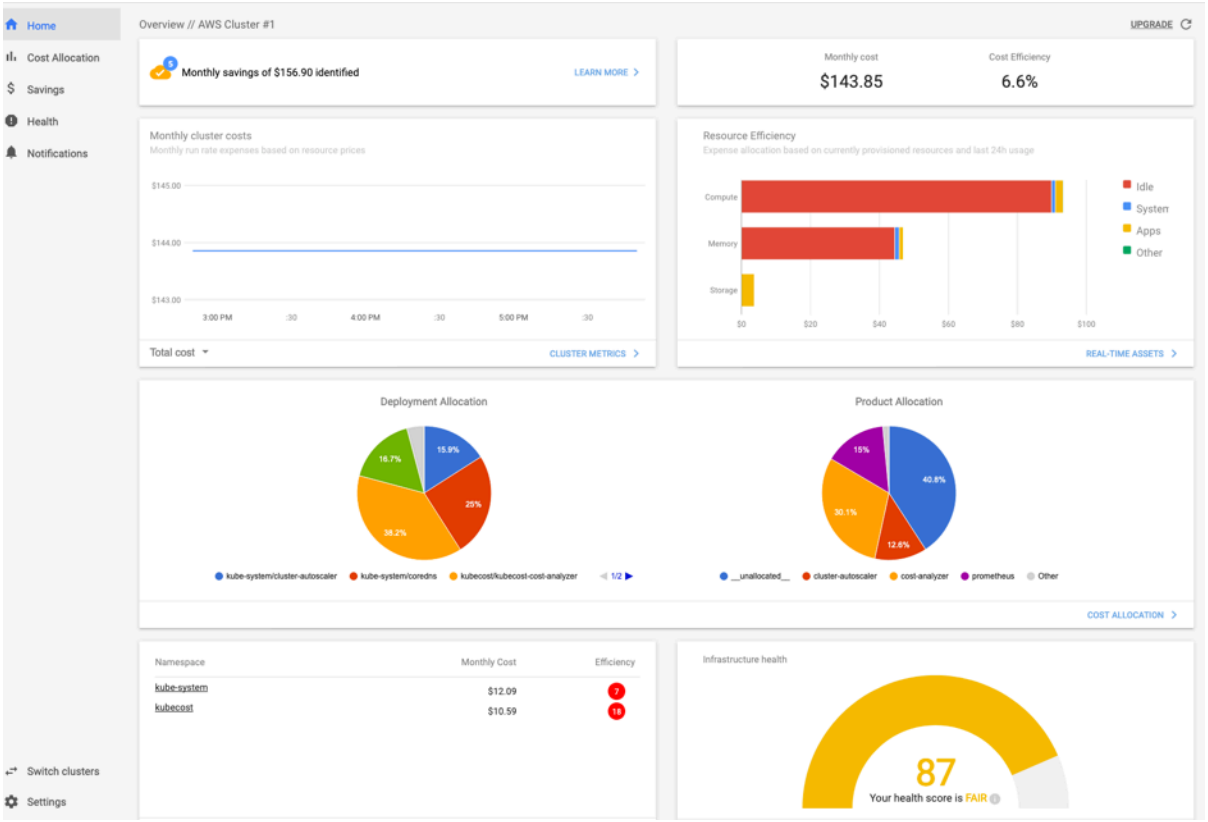
```
NAME: kubecost
LAST DEPLOYED: Mon May 18 08:49:05 2020
NAMESPACE: kubecost
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
----------------------------------------------------Kubecost has been successfully
 installed. When pods are Ready, you can enable port-forwarding with the following
 command:

    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

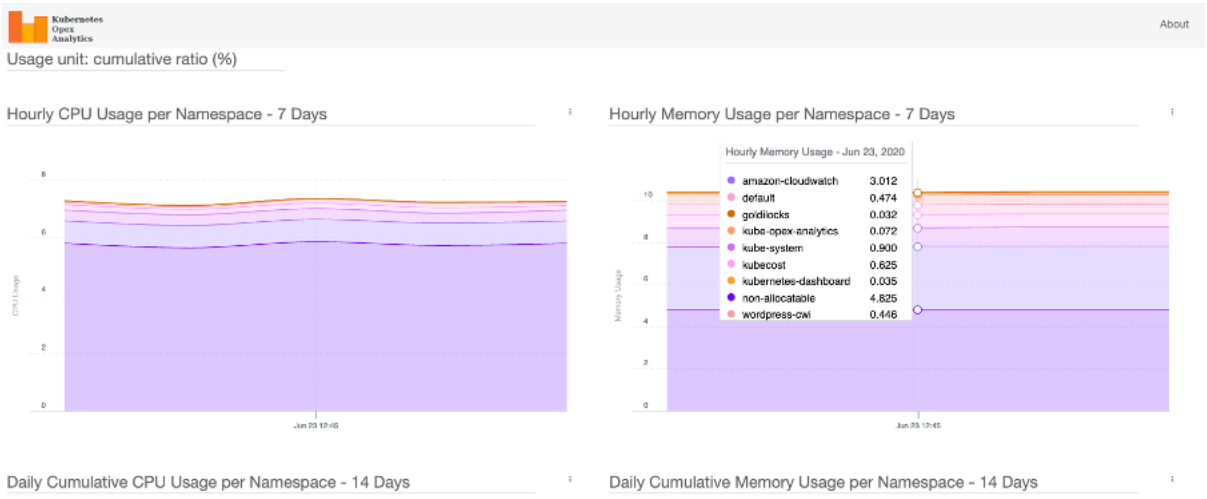
NOTE: If you are using Cloud 9 or have a need to forward it to a different port like
 8080, issue the following command
$ kubectl port-forward --namespace kubecost deployment/kubecost-cost-analyzer 8080:9090
```

Painel do Kubecost -



Use a ferramenta de análise de alocação de custos e planejamento de capacidade do Kubernetes

O Kubernetes Opex Analytics é uma ferramenta para ajudar as organizações a rastrear os recursos consumidos por seus clusters Kubernetes para evitar pagamentos excessivos. Para fazer isso, ele gera relatórios de uso de curto (7 dias), médio (14 dias) e longo prazo (12 meses) mostrando informações relevantes sobre a quantidade de recursos que cada projeto está gastando ao longo do tempo.



Escala de Yota

O Yotascale ajuda a alocar com precisão os custos do Kubernetes. O recurso de alocação de custos do Kubernetes do Yotascale utiliza dados de custo reais, que incluem descontos de instâncias reservadas e preços de instâncias spot, em vez de estimativas genéricas de taxa de mercado, para informar o custo total do Kubernetes

Mais detalhes podem ser encontrados em seu site.

Conselheiro Alcide

Alcide é parceiro de tecnologia avançada da AWS Partner Network (APN). O Alcide Advisor ajuda a garantir que a configuração do cluster, dos nós e dos pods do Amazon EKS seja ajustada para ser executada de acordo com as melhores práticas de segurança e as diretrizes internas. O Alcide Advisor é um serviço sem agente para auditoria e conformidade do Kubernetes, criado para garantir um DevSecOps fluxo seguro e sem atrito, fortalecendo o estágio de desenvolvimento antes de passar para a produção.

Mais detalhes podem ser encontrados nesta postagem do blog.

Outras ferramentas

Coleta de lixo do Kubernetes

A função do coletor de lixo do Kubernetes é excluir certos objetos que já tiveram um proprietário, mas não têm mais um proprietário.

Condado de Fargate

O Fargatecount é uma ferramenta útil que permite aos clientes da AWS rastrear, com uma CloudWatch métrica personalizada, o número total de pods EKS que foram implantados no Fargate em uma região específica de uma conta específica. Isso ajuda a acompanhar todos os pods do Fargate em execução em um cluster EKS.

Popeye - Um desinfetante de cluster Kubernetes

Popeye - A Kubernetes Cluster Sanitizer é um utilitário que escaneia o cluster Kubernetes ativo e relata possíveis problemas com os recursos e configurações implantados. Ele limpa seu cluster com base no que está implantado e não no que está no disco. Ao escanear seu cluster, ele detecta configurações incorretas e ajuda você a garantir que as melhores práticas estejam em vigor

Recursos

Consulte os recursos a seguir para saber mais sobre as melhores práticas para otimização de custos.

Documentação e blogs

- O Amazon EKS oferece suporte à marcação

Ferramentas

- O que é o AWS Billing and Cost Management?
- Amazon CloudWatch Container Insights
- Como rastrear custos em clusters Amazon EKS multilocatários usando o Kubecost
- Cubecost

- Kube Opsview
- Análise Opex do Kubernetes

Computação e escalonamento automático

Como desenvolvedor, você fará estimativas sobre os requisitos de recursos do seu aplicativo, por exemplo, CPU e memória, mas se você não os estiver ajustando continuamente, eles podem ficar desatualizados, o que pode aumentar seus custos e piorar o desempenho e a confiabilidade. Ajustar continuamente os requisitos de recursos de um aplicativo é mais importante do que acertá-los na primeira vez.

As melhores práticas mencionadas abaixo ajudarão você a criar e operar cargas de trabalho com reconhecimento de custos que alcancem resultados comerciais, minimizando os custos e permitindo que sua organização maximize o retorno sobre o investimento. Uma ordem de importância de alto nível para otimizar os custos de computação do cluster é:

1. Cargas de trabalho do tamanho certo
2. Reduza a capacidade não utilizada
3. Otimize os tipos de capacidade computacional (por exemplo, Spot) e aceleradores (por exemplo) GPUs

Dimensione corretamente suas cargas de trabalho

Na maioria dos clusters EKS, a maior parte do custo vem das EC2 instâncias usadas para executar suas cargas de trabalho em contêineres. Você não conseguirá dimensionar corretamente seus recursos computacionais sem entender seus requisitos de cargas de trabalho. É por isso que é essencial que você use as solicitações e os limites apropriados e faça ajustes nessas configurações conforme necessário. Além disso, dependências, como tamanho da instância e seleção de armazenamento, podem afetar o desempenho da carga de trabalho, o que pode ter várias consequências não intencionais nos custos e na confiabilidade.

As solicitações devem estar alinhadas com a utilização real. Se as solicitações de um contêiner forem muito altas, haverá capacidade não utilizada, o que é um grande fator nos custos totais do cluster. Cada contêiner em um pod, por exemplo, aplicativo e sidecars, deve ter suas próprias solicitações e limites definidos para garantir que os limites agregados do pod sejam os mais precisos possíveis.

Utilize ferramentas como Goldilocks, KRR e Kubecost, que estimam as solicitações e os limites de recursos para seus contêineres. Dependendo da natureza dos aplicativos, dos performance/cost requisitos e da complexidade, você precisa avaliar quais métricas são melhores para escalar, em que ponto o desempenho do aplicativo se degrada (ponto de saturação) e como ajustar as solicitações e os limites adequadamente. Consulte o Dimensionamento correto do aplicativo para obter mais orientações sobre esse tópico.

Recomendamos usar o Horizontal Pod Autoscaler (HPA) para controlar quantas réplicas do seu aplicativo devem ser executadas, o Vertical Pod Autoscaler (VPA) para ajustar quantas solicitações e limites seu aplicativo precisa por réplica e um autoescalador de nós, como Karpenter ou Cluster Autoscaler, para ajustar continuamente o número total de nós em seu cluster. As técnicas de otimização de custos usando o Karpenter e o Cluster Autoscaler estão documentadas em uma seção posterior deste documento.

O Vertical Pod Autoscaler pode ajustar as solicitações e os limites atribuídos aos contêineres para que as cargas de trabalho funcionem de forma otimizada. Você deve executar o VPA no modo de auditoria para que ele não faça alterações automaticamente e reinicie seus pods. Ele sugerirá mudanças com base nas métricas observadas. Com quaisquer alterações que afetem as cargas de trabalho de produção, você deve analisar e testar essas alterações primeiro em um ambiente que não seja de produção, pois elas podem ter impacto na confiabilidade e no desempenho do seu aplicativo.

Reduzir o consumo

A melhor maneira de economizar dinheiro é provisionar menos recursos. Uma forma de fazer isso é ajustar as cargas de trabalho com base nos requisitos atuais. Você deve iniciar qualquer esforço de otimização de custos certificando-se de que suas cargas de trabalho definam seus requisitos e sejam escalonadas dinamicamente. Isso exigirá a obtenção de métricas de seus aplicativos e a definição de configurações, como `PodDisruptionBudgets`Pod Readiness Gates, para garantir que seu aplicativo possa ser ampliado e reduzido de forma dinâmica com segurança. É importante considerar que a restrição PodDisruptionBudgets pode impedir que o Autoescalador de Cluster e o Karpenter reduzam os nós, já que tanto o Autoescalador de Cluster quanto o Karpenter respeitam. PodDisruptionBudgets O valor 'minAvailable' no PodDisruptionBudget deve sempre ser menor do que o número de pods na implantação e você deve manter um bom buffer entre os dois. Por exemplo, em uma implantação de 6 pods em que você deseja um mínimo de 4 pods em execução o tempo todo, defina o 'minAvailable' em seu como 4. PodDisruptionBidget Isso permitirá que o Cluster Autoscaler e o Karpenter drenem e expulsem com segurança os pods dos nós subutilizados

durante um evento de redução do Node. Consulte o documento de perguntas frequentes do Cluster Autoscaler.

O Horizontal Pod Autoscaler é um autoescalador flexível de carga de trabalho que pode ajustar quantas réplicas são necessárias para atender aos requisitos de desempenho e confiabilidade do seu aplicativo. Ele tem um modelo flexível para definir quando aumentar e diminuir a escala com base em várias métricas, como CPU, memória ou métricas personalizadas, por exemplo, profundidade da fila, número de conexões com um pod etc.

O Kubernetes Metrics Server permite o escalonamento em resposta a métricas integradas, como uso de CPU e memória, mas se você quiser escalar com base em outras métricas, como a profundidade da fila da Amazon CloudWatch ou do SQS, considere projetos de escalonamento automático orientados por eventos, como o KEDA. Consulte esta postagem do blog sobre como usar o KEDA com CloudWatch métricas. Se você não tiver certeza de quais métricas monitorar e escalar com base, confira as melhores práticas de monitoramento de métricas que importam.

A redução do consumo da carga de trabalho cria excesso de capacidade em um cluster e, com a configuração adequada de escalonamento automático, permite que você reduza os nós automaticamente e reduza seu gasto total. Recomendamos que você não tente otimizar a capacidade computacional manualmente. O programador Kubernetes e os autoescaladores de nós foram projetados para lidar com esse processo para você.

Reduza a capacidade não utilizada

Depois de determinar o tamanho correto dos aplicativos, reduzindo o excesso de solicitações, você pode começar a reduzir a capacidade computacional provisionada. Você deve ser capaz de fazer isso dinamicamente se tiver reservado um tempo para dimensionar corretamente suas cargas de trabalho nas seções acima. Há dois autoescaladores de nós principais usados com o Kubernetes na AWS.

Karpenter e Cluster Autoscaler

Tanto o Karpenter quanto o Kubernetes Cluster Autoscaler escalarão o número de nós em seu cluster à medida que os pods forem criados ou removidos e os requisitos de computação mudarem. O objetivo principal de ambos é o mesmo, mas a Karpenter adota uma abordagem diferente para gerenciamento, provisionamento e desprovisionamento de nós, o que pode ajudar a reduzir custos e otimizar o uso em todo o cluster.

À medida que os clusters aumentam de tamanho e a variedade de cargas de trabalho aumenta, fica mais difícil pré-configurar grupos de nós e instâncias. Assim como nas solicitações de carga

de trabalho, é importante definir uma linha de base inicial e ajustá-la continuamente conforme necessário.

Se você estiver usando o Cluster Autoscaler, ele respeitará os valores “mínimo” e “máximo” de cada grupo de Auto Scaling (ASG) e ajustará somente o valor “desejado”. É importante prestar atenção ao definir esses valores para o ASG subjacente, pois o Cluster Autoscaler não conseguirá reduzir um ASG além de sua contagem “mínima”. Defina a contagem “desejada” como o número de nós necessários durante o horário comercial normal e “mínima” como o número de nós necessários fora do horário comercial. Consulte o documento de perguntas frequentes do Cluster Autoscaler.

Expansor prioritário do escalador automático de cluster

O Kubernetes Cluster Autoscaler funciona escalando grupos de nós — chamados de grupo de nós — para cima e para baixo à medida que os aplicativos aumentam e diminuem a escala. Se você não estiver escalando dinamicamente as cargas de trabalho, o autoescalador de cluster não ajudará você a economizar dinheiro. O escalador automático de cluster exige que um administrador de cluster crie grupos de nós com antecedência para que as cargas de trabalho sejam consumidas. Os grupos de nós precisam ser configurados para usar instâncias que tenham o mesmo “perfil”, ou seja, aproximadamente a mesma quantidade de CPU e memória.

Você pode ter vários grupos de nós e o autoescalador de cluster pode ser configurado para definir níveis de escalabilidade prioritária, e cada grupo de nós pode conter nós de tamanhos diferentes. Os grupos de nós podem ter diferentes tipos de capacidade e o expansor prioritário pode ser usado primeiro para escalar grupos mais baratos.

Abaixo está um exemplo de um trecho de configuração de cluster que usa a `ConfigMap` para priorizar a capacidade reservada antes de usar instâncias sob demanda. Você pode usar a mesma técnica para priorizar instâncias Graviton ou Spot em relação a outros 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.*
```

O uso de grupos de nós pode ajudar os recursos computacionais subjacentes a fazer o esperado por padrão, por exemplo, distribuir nós entre si AZs, mas nem todas as cargas de trabalho têm os mesmos requisitos ou expectativas, e é melhor deixar que os aplicativos declarem seus requisitos explicitamente. Para obter mais informações sobre o Cluster Autoscaler, consulte a seção de melhores práticas.

Desagendador

O escalador automático de cluster pode adicionar e remover a capacidade de nós de um cluster com base na necessidade de programar novos pods ou na subutilização dos nós. Ele não tem uma visão holística do posicionamento do pod depois de ter sido programado para um nó. Se você estiver usando o autoescalador de cluster, você também deve consultar o desagendador do Kubernetes para evitar o desperdício de capacidade em seu cluster.

Se você tem 10 nós em um cluster e cada nó é 60% utilizado, você não está usando 40% da capacidade provisionada no cluster. Com o autoescalador de cluster, você pode definir o limite de utilização por nó para 60%, mas isso só tentaria reduzir um único nó depois que a utilização caísse abaixo de 60%.

Com o desagendador, ele pode analisar a capacidade e a utilização do cluster após os pods terem sido programados ou os nós terem sido adicionados ao cluster. Ele tenta manter a capacidade total do cluster acima de um limite especificado. Ele também pode remover pods com base em contaminações de nós ou em novos nós que se juntam ao cluster para garantir que os pods sejam executados em seu ambiente computacional ideal. Observe que o desagendador não agenda a substituição dos pods despejados, mas depende do agendador padrão para isso.

Consolidação Karpenter

O Karpenter adota uma abordagem “sem grupos” para o gerenciamento de nós. Essa abordagem é mais flexível para diferentes tipos de carga de trabalho e requer menos configuração inicial para administradores de cluster. Em vez de predefinir grupos e escalar cada grupo conforme a necessidade das cargas de trabalho, o Karpenter usa provisionadores e modelos de nós para definir amplamente quais tipos de EC2 instâncias podem ser criadas e configurações sobre as instâncias à medida que são criadas.

O empacotamento em compartimentos é a prática de utilizar mais recursos da instância ao empacotar mais cargas de trabalho em menos instâncias de tamanho ideal. Embora isso ajude a reduzir seus custos de computação ao provisionar apenas os recursos que suas cargas de trabalho usam, isso tem uma desvantagem. Pode levar mais tempo para iniciar novas cargas de trabalho porque a capacidade precisa ser adicionada ao cluster, especialmente durante grandes eventos de escalabilidade. Considere o equilíbrio entre otimização de custos, desempenho e disponibilidade ao configurar o empacotamento de lixo.

O Karpenter pode monitorar e agrupar continuamente para melhorar a utilização dos recursos da instância e reduzir seus custos de computação. O Karpenter também pode selecionar um nó de trabalho mais econômico para sua carga de trabalho. Isso pode ser feito ativando o sinalizador de “consolidação” como verdadeiro no provisionador (exemplo de trecho de código abaixo). O exemplo abaixo mostra um exemplo de provisionador que permite a consolidação. No momento em que escrevo este guia, o Karpenter não substituirá uma instância Spot em execução por uma instância Spot mais barata. Para obter mais detalhes sobre a consolidação do Karpenter, consulte este blog.

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
  name: enable-binpacking
spec:
  consolidation:
    enabled: true
```

Para cargas de trabalho que podem não ser interrompidas, por exemplo, trabalhos em lotes de longa execução sem ponto de verificação, considere anotar pods com a anotação. `do-not-evict` Ao desativar o despejo dos pods, você está dizendo ao Karpenter que ele não deve remover voluntariamente os nós que contêm esse pod. No entanto, se um `do-not-evict` pod for adicionado a um nó enquanto o nó estiver sendo drenado, os pods restantes ainda serão despejados, mas esse pod bloqueará o encerramento até que seja removido. Em ambos os casos, o nó será isolado para

evitar que trabalhos adicionais sejam agendados no nó. Abaixo está um exemplo mostrando como definir a anotação:

```
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
```

Remova nós subutilizados ajustando os parâmetros do autoescalador de cluster

A utilização do nó é definida como a soma dos recursos solicitados dividida pela capacidade. Por padrão, `scale-down-utilization-threshold` está definido como 50%. Esse parâmetro pode ser usado junto com `escale-down-unneeded-time`, que determina por quanto tempo um nó deve ser desnecessário antes de ser elegível para redução de escala — o padrão é 10 minutos. Os pods ainda em execução em um nó que foi reduzido serão programados em outros nós pelo kube-scheduler. O ajuste dessas configurações pode ajudar a remover nós que estão subutilizados, mas é importante testar esses valores primeiro para não forçar o cluster a ser reduzido prematuramente.

Você pode evitar que a redução ocorra garantindo que os pods que são caros de remover sejam protegidos por um rótulo reconhecido pelo escalador automático de cluster. Para fazer isso, certifique-se de que os pods que são caros de despejar tenham a anotação. `cluster-autoscaler.kubernetes.io/safe-to-evict=false` Abaixo está um exemplo de yaml para definir a anotação:

```
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
```

Marque nós com o Cluster Autoscaler e o Karpenter

As tags de recursos da AWS são usadas para organizar seus recursos e monitorar seus custos da AWS em um nível detalhado. Eles não se correlacionam diretamente com os rótulos do Kubernetes para controle de custos. É recomendável começar com a rotulagem de recursos do Kubernetes e utilizar ferramentas como o Kubecost para obter relatórios de custos de infraestrutura com base nos rótulos do Kubernetes em pods, namespaces etc.

Os nós de trabalho precisam ter tags para mostrar informações de faturamento no AWS Cost Explorer. Com o escalador automático de cluster, marque seus nós de trabalho dentro de um grupo de nós gerenciados usando o modelo de lançamento. Para grupos de nós autogerenciados, marque suas instâncias usando o grupo de escalonamento EC2 automático. Para instâncias provisionadas pelo Karpenter, marque-as usando spec.tags no modelo de nó.

Clusters multilocatários

Ao trabalhar em clusters compartilhados por equipes diferentes, você pode não ter visibilidade de outras cargas de trabalho em execução no mesmo nó. Embora as solicitações de recursos possam ajudar a isolar algumas preocupações do “vizinho barulhento”, como o compartilhamento de CPU, elas podem não isolar todos os limites dos recursos, como a exaustão do disco. I/O Nem todo recurso consumível de uma carga de trabalho pode ser isolado ou limitado. As cargas de trabalho que consomem recursos compartilhados em taxas mais altas do que outras cargas de trabalho devem ser isoladas por meio de contaminações e tolerâncias nos nós. Outra técnica avançada para essa carga de trabalho é a fixação da CPU, que garante uma CPU exclusiva em vez de uma CPU compartilhada para o contêiner.

Isolar cargas de trabalho em um nível de nó pode ser mais caro, mas pode ser possível agendar BestEfforttrabalhos ou aproveitar economias adicionais usando instâncias reservadas, processadores Graviton ou Spot.

Os clusters compartilhados também podem ter restrições de recursos no nível do cluster, como exaustão de IP, limites de serviço do Kubernetes ou solicitações de escalonamento de API. Você deve revisar o guia de melhores práticas de escalabilidade para garantir que seus clusters evitem esses limites.

Você pode isolar recursos em um namespace ou no nível do provisionador Karpenter. As cotas de recursos fornecem uma forma de definir limites sobre quantos recursos as cargas de trabalho em um namespace podem consumir. Essa pode ser uma boa barreira inicial, mas deve ser avaliada continuamente para garantir que não restrinja artificialmente o escalonamento das cargas de trabalho.

Os provisionadores do Karpenter podem definir limites para alguns dos recursos consumíveis em um cluster (por exemplo, CPU, GPU), mas você precisará configurar os aplicativos de locatário para usar o provisionador apropriado. Isso pode impedir que um único provisionador crie muitos nós em um cluster, mas deve ser avaliado continuamente para garantir que o limite não seja muito baixo e, por sua vez, evitar que as cargas de trabalho aumentem.

Escalonamento automático programado

Talvez você precise reduzir seus clusters nos fins de semana e fora do horário de expediente. Isso é particularmente relevante para clusters de teste e de não produção, nos quais você deseja reduzir para zero quando não estão em uso. Soluções como o cluster-turndown podem reduzir as réplicas para zero com base em um cronograma cron. Você também pode conseguir isso com o Karpenter, descrito no seguinte blog da AWS.

Otimize os tipos de capacidade computacional

Depois de otimizar a capacidade total de computação em seu cluster e utilizar o compartimento de armazenamento, você deve verificar o tipo de computação que você provisionou em seus clusters e como você paga por esses recursos. A AWS tem planos de economia de computação que podem reduzir o custo de sua computação, que categorizaremos nos seguintes tipos de capacidade:

- Spot
- Savings Plans
- Sob demanda
- Fargate

Cada tipo de capacidade tem diferentes vantagens e desvantagens em termos de despesas gerais de gerenciamento, disponibilidade e compromissos de longo prazo, e você precisará decidir qual é a melhor opção para seu ambiente. Nenhum ambiente deve depender de um único tipo de capacidade e você pode combinar vários tipos de execução em um único cluster para otimizar os requisitos e custos específicos da carga de trabalho.

Spot

O tipo de capacidade spot provisiona EC2 instâncias da capacidade ociosa em uma zona de disponibilidade. O Spot oferece os maiores descontos — até 90% — mas essas instâncias podem ser interrompidas quando necessárias em outro lugar. Além disso, nem sempre há capacidade para provisionar novas instâncias spot e as instâncias spot existentes podem ser recuperadas com um aviso de interrupção de 2 minutos. Se seu aplicativo tiver um longo processo de inicialização ou desligamento, as instâncias spot podem não ser a melhor opção.

A computação spot deve usar uma grande variedade de tipos de instância para reduzir a probabilidade de não ter capacidade spot disponível. As interrupções da instância precisam ser tratadas para desligar os nós com segurança. Os nós provisionados com o Karpenter ou parte de um grupo de nós gerenciados oferecem suporte automático às notificações de interrupção da instância. Se você estiver usando nós autogerenciados, precisará executar o manipulador de terminação de nós separadamente para encerrar instâncias spot normalmente.

É possível equilibrar instâncias spot e sob demanda em um único cluster. Com o Karpenter, você pode criar provisionadores ponderados para alcançar um equilíbrio entre diferentes tipos de capacidade. Com o Cluster Autoscaler, você pode criar grupos de nós mistos com instâncias spot e sob demanda ou reservadas.

Aqui está um exemplo do uso do Karpenter para priorizar as instâncias spot antes das instâncias sob demanda. Ao criar um provisionador, você pode especificar Spot, On-Demand ou ambos (conforme mostrado abaixo). Quando você especifica ambos, e se o pod não especifica explicitamente se precisa usar Spot ou On-Demand, o Karpenter prioriza o Spot ao provisionar um nó com estratégia de alocação. 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 e AWS EDP

Você pode reduzir seus gastos com computação usando um plano de economia de computação. Os planos de economia oferecem preços reduzidos para um compromisso de 1 ou 3 anos de uso da computação. O uso pode ser aplicado a EC2 instâncias em um cluster EKS, mas também se aplica a qualquer uso de computação, como Lambda e Fargate. Com os planos de economia, você pode reduzir custos e ainda escolher qualquer tipo de EC2 instância durante o período de compromisso.

O plano de economia de computação pode reduzir seu EC2 custo em até 66% sem exigir compromissos sobre quais tipos de instância, famílias ou regiões você deseja usar. As economias são aplicadas automaticamente às instâncias à medida que você as usa.

EC2 Os Instance Savings Plans oferecem até 72% de economia em computação com um compromisso de uso em uma região e EC2 família específicas, por exemplo, instâncias da família C. Você pode mudar o uso para qualquer AZ na região, usar qualquer geração da família de instâncias, por exemplo, c5 ou c6, e usar qualquer tamanho de instância dentro da família. O desconto será aplicado automaticamente a qualquer instância em sua conta que corresponda aos critérios do plano de poupança.

As instâncias reservadas são semelhantes ao EC2 Instance Savings Plan, mas também garantem capacidade em uma zona ou região de disponibilidade e reduzem o custo — até 72% — em relação às instâncias sob demanda. Depois de calcular a capacidade reservada necessária, você pode selecionar por quanto tempo gostaria de reservá-la (1 ano ou 3 anos). Os descontos serão aplicados automaticamente à medida que você executa essas EC2 instâncias em sua conta.

Os clientes também têm a opção de se inscrever em um contrato empresarial com a AWS. Os contratos empresariais oferecem aos clientes a opção de personalizar os contratos que melhor atendem às suas necessidades. Os clientes podem desfrutar de descontos nos preços com base no AWS EDP (Enterprise Discount Program). Para obter informações adicionais sobre contratos corporativos, entre em contato com seu representante de vendas da AWS.

Sob demanda

EC2 As instâncias sob demanda têm o benefício da disponibilidade sem interrupções — em comparação com a spot — e sem compromissos de longo prazo — em comparação com os planos

de economia. Se você deseja reduzir custos em um cluster, deve reduzir o uso de EC2 instâncias sob demanda.

Depois de otimizar seus requisitos de carga de trabalho, você deve ser capaz de calcular a capacidade mínima e máxima para seus clusters. Esse número pode mudar com o tempo, mas raramente diminui. Considere usar um Savings Plan para tudo que estiver abaixo do mínimo e determine a capacidade que não afetará a disponibilidade do seu aplicativo. Qualquer outra coisa que não possa ser usada continuamente ou seja necessária para disponibilidade pode ser usada sob demanda.

Conforme mencionado nesta seção, a melhor maneira de reduzir seu uso é consumir menos recursos e utilizar os recursos que você provisiona ao máximo possível. Com o autoescalador de cluster, você pode remover nós subutilizados com a configuração. `scale-down-utilization-threshold` Com o Karpenter, é recomendável ativar a consolidação.

Para identificar manualmente os tipos de EC2 instância que podem ser usados com suas cargas de trabalho, você deve usar o ec2-instance-selector, que pode mostrar as instâncias disponíveis em cada região, bem como as instâncias compatíveis com o EKS. Exemplo de uso para instâncias com arquitetura de processo x86, 4 Gb de memória, 2 v CPUs e disponíveis na região 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 ambientes que não sejam de produção, você pode reduzir automaticamente os clusters durante horas não utilizadas, como noites e fins de semana. O projeto kubecost cluster-turndown é um exemplo de controlador que pode reduzir automaticamente seu cluster com base em um cronograma definido.

Computação Fargate

A computação Fargate é uma opção de computação totalmente gerenciada para clusters EKS. Ele fornece isolamento de pods ao programar um pod por nó em um cluster Kubernetes. Ele permite que

você dimensione seus nós de computação de acordo com os requisitos de CPU e RAM de sua carga de trabalho para controlar rigorosamente o uso da carga de trabalho em um cluster.

O Fargate pode escalar cargas de trabalho tão pequenas quanto 0,25 vCPU com 0,5 GB de memória e até 16 vCPU com 120 GB de memória. Há limites para quantas variações de tamanho de pod estão disponíveis e você precisará entender como sua carga de trabalho se encaixa melhor em uma configuração do Fargate. Por exemplo, se sua carga de trabalho exigir 1 vCPU com 0,5 GB de memória, o menor pod Fargate será 1 vCPU com 2 GB de memória.

Embora o Fargate tenha muitos benefícios, como o gerenciamento de nenhuma EC2 instância ou sistema operacional, ele pode exigir mais capacidade computacional do que as EC2 instâncias tradicionais, devido ao fato de que cada pod implantado é isolado como um nó separado no cluster. Isso requer mais duplicação para coisas como o Kubelet, agentes de registro e qualquer coisa DaemonSets que você normalmente implantaria em um nó. DaemonSets não são compatíveis com o Fargate e precisarão ser convertidos em pod “`sidecars"` e executados junto com o aplicativo.

O Fargate não pode se beneficiar do empacotamento de compartimentos ou do provisionamento excessivo de CPU porque o limite da carga de trabalho é um nó que não pode ser interrompido ou compartilhado entre cargas de trabalho. O Fargate economizará tempo de gerenciamento de EC2 instâncias, o que por si só tem um custo, mas os custos de CPU e memória podem ser mais caros do que outros tipos de EC2 capacidade. Os pods Fargate podem aproveitar o plano de economia de computação para reduzir o custo sob demanda.

Otimize o uso da computação

Outra forma de economizar dinheiro em sua infraestrutura computacional é usar uma computação mais eficiente para a carga de trabalho. Isso pode vir de uma computação de uso geral com maior desempenho, como os processadores Graviton, que são até 20% mais baratos e 60% mais eficientes em termos de energia do que os processadores x86, ou aceleradores específicos para cargas de trabalho, como e. GPUs FPGAs Você precisará criar contêineres que possam ser executados na arquitetura arm e configurar nós com os aceleradores certos para suas cargas de trabalho.

O EKS tem a capacidade de executar clusters com arquitetura mista (por exemplo, amd64 e arm64) e, se seus contêineres forem compilados para várias arquiteturas, você poderá aproveitar os processadores Graviton com o Karpenter permitindo ambas as arquiteturas em seu provisionador. Para manter um desempenho consistente, no entanto, é recomendável manter cada carga de trabalho em uma única arquitetura de computação e usar arquiteturas diferentes somente se não houver capacidade adicional disponível.

Os provisionadores podem ser configurados com várias arquiteturas e as cargas de trabalho também podem solicitar arquiteturas específicas em suas especificações de carga de trabalho.

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
  name: default
spec:
  requirements:
  - key: "kubernetes.io/arch"
    operator: In
    values: ["arm64", "amd64"]
```

Com o Cluster Autoscaler, você precisará criar um grupo de nós para instâncias do Graviton e definir tolerâncias de nós em sua carga de trabalho para utilizar a nova capacidade.

GPUs e FPGAs pode aumentar consideravelmente o desempenho de sua carga de trabalho, mas a carga de trabalho precisará ser otimizada para usar o acelerador. Muitos tipos de carga de trabalho para aprendizado de máquina e inteligência artificial podem ser usados GPUs para computação e instâncias podem ser adicionados a um cluster e montados em uma carga de trabalho usando solicitações de recursos.

```
spec:
  template:
    spec:
    - containers:
      ...
      resources:
          limits:
            nvidia.com/gpu: "1"
```

Alguns hardwares de GPU podem ser compartilhados entre várias cargas de trabalho para que uma única GPU possa ser provisionada e usada. Para ver como configurar o compartilhamento da GPU da carga de trabalho, consulte o plug-in do dispositivo de GPU virtual para obter mais informações. Você também pode consultar os seguintes blogs:

- Compartilhamento de GPU no Amazon EKS com divisão de tempo e instâncias aceleradas da NVIDIA EC2
- Maximização da utilização de GPU com a GPU de múltiplas instâncias (MIG) da NVIDIA no Amazon EKS: execução de mais pods por GPU para melhorar a performance

Otimização de custos - Rede

A arquitetura de sistemas para alta disponibilidade (HA) é uma prática recomendada para obter resiliência e tolerância a falhas. Na prática, isso significa distribuir suas cargas de trabalho e a infraestrutura subjacente em várias zonas de disponibilidade (AZs) em uma determinada região da AWS. Garantir que essas características estejam em vigor em seu ambiente Amazon EKS aumentará a confiabilidade geral do seu sistema. Em conjunto com isso, seus ambientes EKS provavelmente também serão compostos por uma variedade de construções (ou seja, VPCs), componentes (ou seja,) e integrações (ou seja, ECR e outros registros de contêineres ELBs).

A combinação de sistemas altamente disponíveis e outros componentes específicos do caso de uso pode desempenhar um papel significativo na forma como os dados são transferidos e processados. Isso, por sua vez, terá um impacto nos custos incorridos devido à transferência e processamento de dados.

As práticas detalhadas abaixo ajudarão você a projetar e otimizar seus ambientes EKS para obter economia em diferentes domínios e casos de uso.

Comunicação de pod a pod

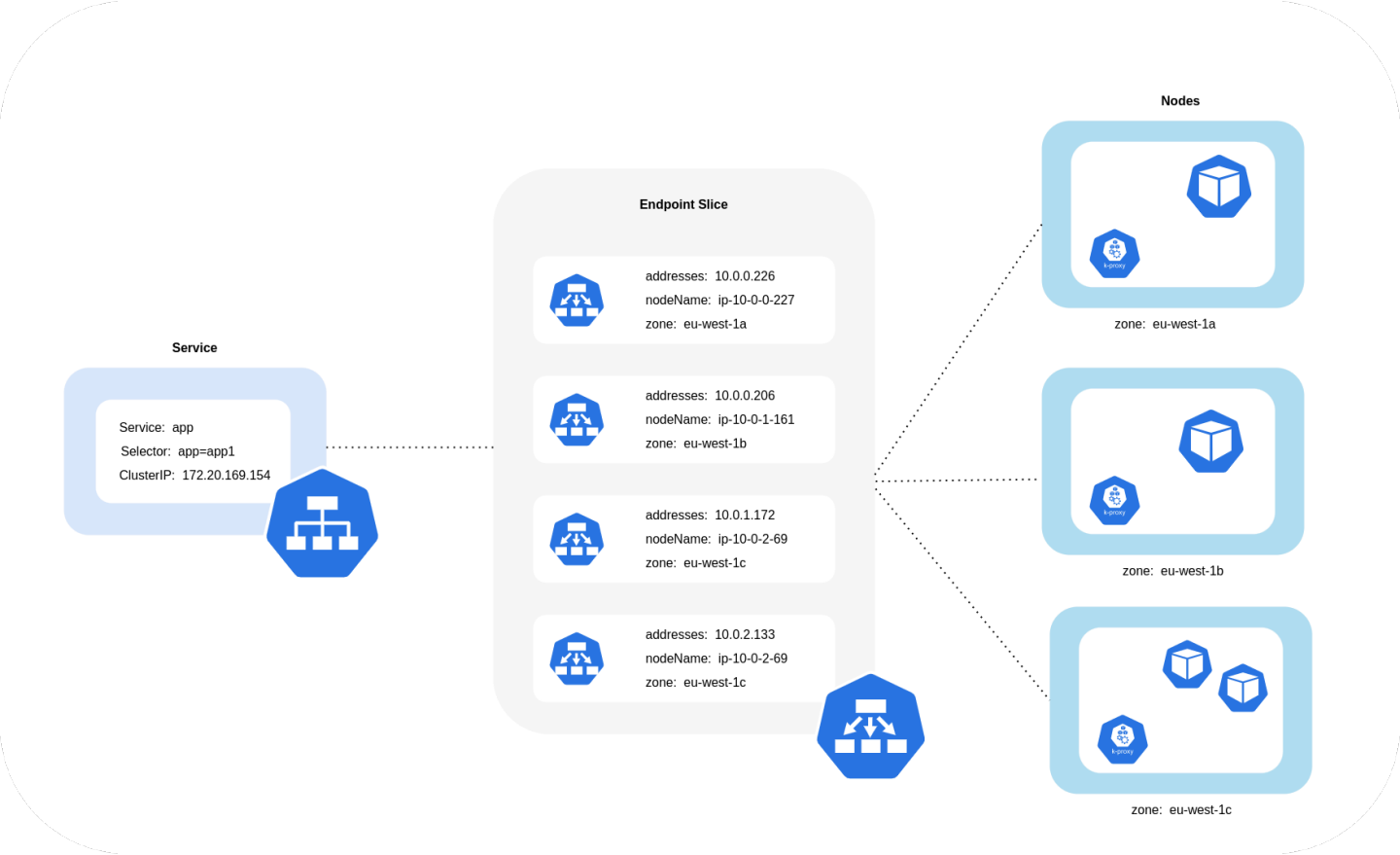
Dependendo da sua configuração, a comunicação de rede e a transferência de dados entre pods podem ter um impacto significativo no custo geral da execução das cargas de trabalho do Amazon EKS. Esta seção abordará diferentes conceitos e abordagens para mitigar os custos associados à comunicação entre pods, considerando arquiteturas de alta disponibilidade (HA), desempenho e resiliência de aplicativos.

Restringindo o tráfego a uma zona de disponibilidade

Logo no início, o projeto Kubernetes começou a desenvolver construções com reconhecimento de topologia, incluindo rótulos como kubernetes. io/hostname, topology.kubernetes.io/region, and topology.kubernetes.io/zoneatribuído aos nós para permitir recursos como distribuição de carga de trabalho em domínios de falha e provisionadores de volume com reconhecimento de topologia. Depois de se graduar no Kubernetes 1.17, os rótulos também foram utilizados para permitir recursos de roteamento com reconhecimento de topologia para comunicação de pod a pod.

Abaixo estão algumas estratégias sobre como controlar a quantidade de tráfego entre AZ entre pods em seu cluster EKS para reduzir custos e minimizar a latência.

Se você quiser visibilidade granular da quantidade de tráfego cross-AZ entre os pods em seu cluster (como a quantidade de dados transferidos em bytes), consulte esta postagem.



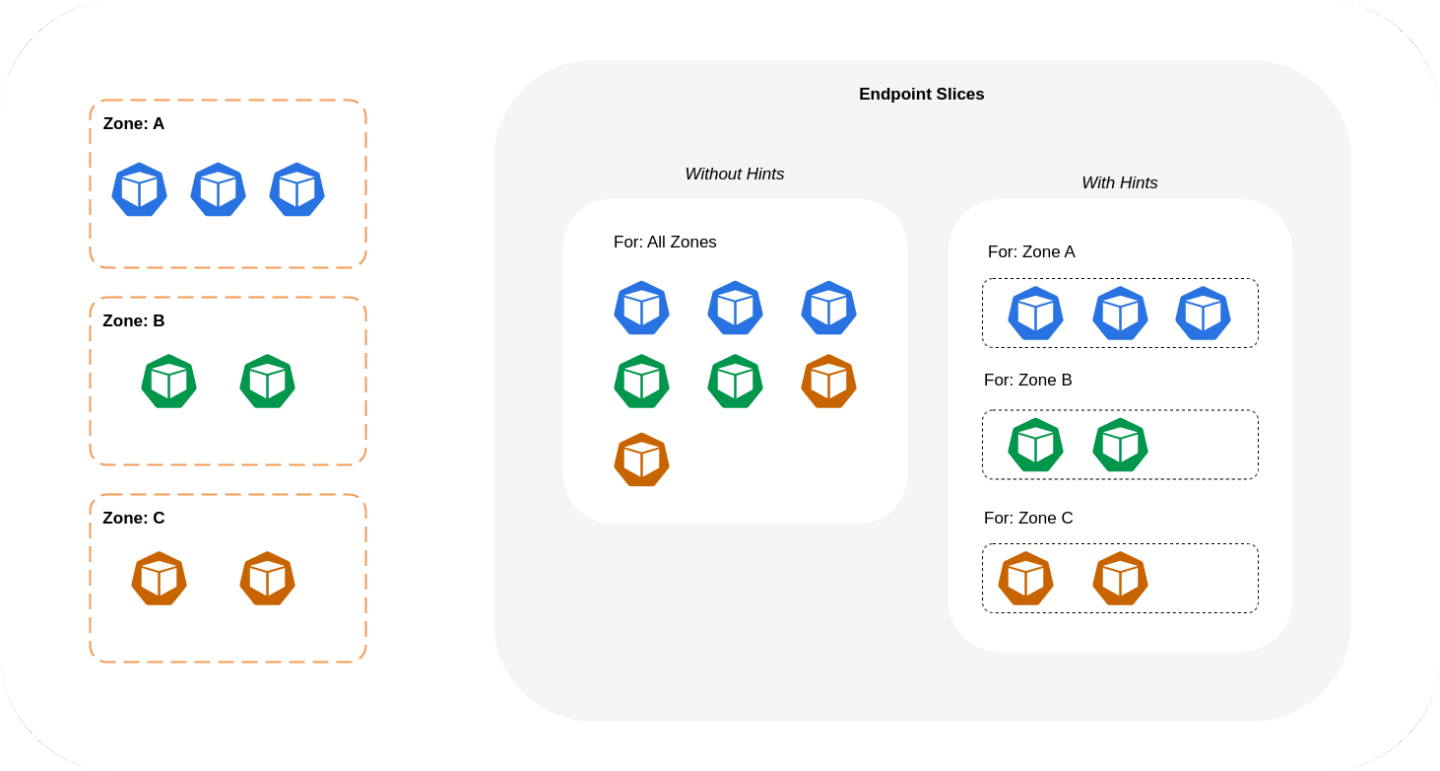
Como mostra o diagrama acima, os serviços são a camada de abstração de rede estável que recebe o tráfego destinado aos seus pods. Quando um serviço é criado, vários EndpointSlices são criados. Cada um EndpointSlice tem uma lista de endpoints contendo um subconjunto de endereços de pod junto com os nós em que estão sendo executados e qualquer informação adicional de topologia. Ao usar a CNI da Amazon VPC, o kube-proxy, um daemonset executado em cada nó, mantém as regras de rede para permitir a comunicação do pod e a descoberta de serviços (uma alternativa baseada em EBPF CNIs pode não usar o kube-proxy, mas fornecer um comportamento equivalente). Ele cumpre o papel de roteamento interno, mas o faz com base no que consome do criado. EndpointSlices

No EKS, o kube-proxy usa principalmente regras NAT iptables (ou IPVS, NFTablescomo alternativas) para distribuição de tráfego em todos os pods no cluster, independentemente do nó ou do posicionamento do AZ. Essa distribuição padrão pode levar ao roteamento de tráfego entre AZ, potencialmente causando maior latência para aplicativos confidenciais e cobranças de transferência de dados entre AZ em grandes implantações.

Usando o roteamento com reconhecimento de topologia (anteriormente conhecido como dicas com reconhecimento de topologia)

Quando o roteamento com reconhecimento de topologia é ativado e implementado em um serviço Kubernetes, o EndpointSlice controlador alocará proporcionalmente os endpoints para as diferentes zonas pelas quais seu cluster está distribuído. Para cada um desses endpoints, o EndpointSlice controlador também definirá uma dica para a zona. As dicas descrevem para qual zona um endpoint deve fornecer tráfego. `kube-proxy`em seguida, roteará o tráfego de uma zona para um endpoint com base nas dicas aplicadas.

O diagrama abaixo mostra como EndpointSlices as dicas são organizadas de forma que `kube-proxy` possam saber para qual destino elas devem ir com base em seu ponto de origem zonal. Sem dicas, essa alocação ou organização não existe e o tráfego será enviado por proxy para diferentes destinos zonais, independentemente de sua origem.



Em alguns casos, o EndpointSlice controlador pode aplicar uma dica para uma zona diferente, o que significa que o endpoint pode acabar servindo tráfego proveniente de uma zona diferente. A razão para isso é tentar manter uma distribuição uniforme do tráfego entre endpoints em diferentes zonas.

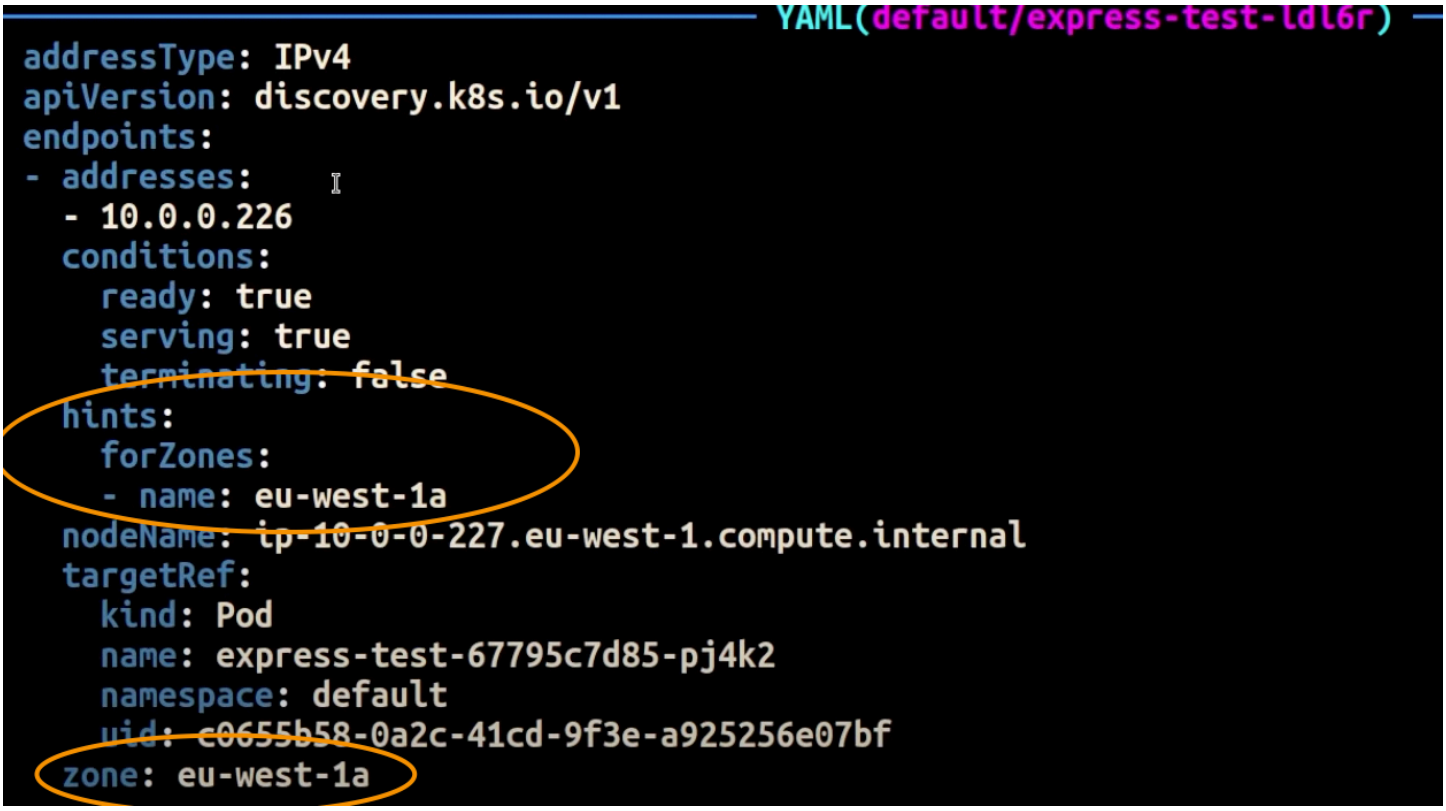
Abaixo está um trecho de código sobre como habilitar o roteamento com reconhecimento de topologia para um serviço.

```
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
```

A captura de tela abaixo mostra o resultado da aplicação bem-sucedida pela EndpointSlice controladora de uma dica em um endpoint para uma réplica de pod em execução no AZ. `eu-west-1a`



> **Note**
>
> É importante observar que o roteamento com reconhecimento de topologia ainda está na versão beta. Esse recurso funciona de forma mais previsível com cargas de trabalho

distribuídas uniformemente na topologia do cluster, pois o controlador aloca endpoints proporcionalmente entre as zonas, mas pode ignorar as atribuições de dicas quando os recursos do nó em uma zona estão muito desequilibrados para evitar sobrecarga excessiva. Portanto, é altamente recomendável usá-lo em conjunto com restrições de agendamento que aumentem a disponibilidade de um aplicativo, como restrições de dispersão da topologia do pod. Observe que as dicas também podem não ser atribuídas quando a capacidade flutua entre zonas, como ao usar Amazon EC2 Spot Instances, pois interrupções ou substituições não são detectadas em tempo real ao calcular a distribuição proporcional.

Usando a distribuição de tráfego

Introduzida no Kubernetes 1.30 e disponibilizada em geral na versão 1.33, a distribuição de tráfego oferece uma alternativa mais simples ao roteamento com reconhecimento de topologia para a preferência de tráfego na mesma zona. Embora o roteamento com reconhecimento de topologia tente usar uma abordagem inteligente ao roteamento de tráfego para evitar sobrecarregar os endpoints, isso resultou em um comportamento imprevisível. Em vez disso, a distribuição de tráfego prioriza a previsibilidade. A PreferClose opção direciona o kube-proxy a criar regras que roteiam o tráfego para os endpoints da mesma zona primeiro com base na dica zonal definida pelo Controller. EndpointSlice Quando nenhum endpoint da mesma zona está disponível, ele volta a distribuir o tráfego em qualquer endpoint de cluster para o Serviço. Esse recurso foi projetado para cargas de trabalho que aceitam a desvantagem de otimizar por proximidade, em vez da tentativa de distribuição uniforme da carga fornecida pelo Roteamento com Reconhecimento de Topologia.

Abaixo está um trecho de código sobre como habilitar a distribuição de tráfego para um serviço.

```
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
```

Ao habilitar a distribuição de tráfego, surge um desafio comum: os endpoints em uma única AZ podem ficar sobrecarregados se a maior parte do tráfego for originada dessa mesma zona. Essa sobrecarga pode criar problemas significativos:

- Um único escalador automático horizontal de pods (HPA) gerenciando uma implantação Multi-AZ pode responder escalando pods em diferentes. AZs No entanto, essa ação falha em abordar efetivamente o aumento da carga na zona afetada.
- Essa situação, por sua vez, pode levar à ineficiência de recursos. Quando autoescaladores de cluster, como o Karpenter, detectam a expansão horizontal do pod em diferentes AZs, eles podem provisionar nós adicionais nos não afetados, resultando em alocação AZs desnecessária de recursos.

Para superar esse desafio:

- Crie implantações separadas por zona, que teriam suas próprias HPAs escalas independentes umas das outras.
- Aproveite as restrições de dispersão de topologia para garantir a distribuição da carga de trabalho em todo o cluster, o que ajuda a evitar sobrecargas de endpoints em zonas de alto tráfego.

Usando autoescaladores: provisionar nós para uma AZ específica

É altamente recomendável executar suas cargas de trabalho em ambientes altamente disponíveis em vários AZs ambientes. Isso melhora a confiabilidade de seus aplicativos, especialmente quando há um incidente de um problema com uma AZ. Caso esteja disposto a sacrificar a confiabilidade para reduzir os custos relacionados à rede, você pode restringir seus nós a uma única AZ.

Para executar todos os seus pods na mesma AZ, provisione os nós de trabalho na mesma AZ ou programe os pods nos nós de trabalho em execução na mesma AZ. Para provisionar nós em uma única AZ, defina um grupo de nós com sub-redes pertencentes à mesma AZ com o Cluster Autoscaler (CA). Para o Karpenter, use `topology.kubernetes.io/zone` e especifique a AZ em que você gostaria de criar os nós de trabalho. Por exemplo, o trecho do provisionador Karpenter abaixo provisiona os nós na AZ 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"]
```

Autoescalador de cluster (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
...
```

Usando Pod Assignment e Node Affinity

Como alternativa, se você tiver nós de trabalho em execução em vários AZs, cada nó terá o rótulo topology.kubernetes.io/zone com o valor de sua AZ (como us-west-2a ou us-west-2b). Você pode utilizar `nodeSelector` ou `nodeAffinity` programar pods para os nós em um único AZ. Por exemplo, o arquivo de manifesto a seguir agendará o pod dentro de um nó em execução no 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
```

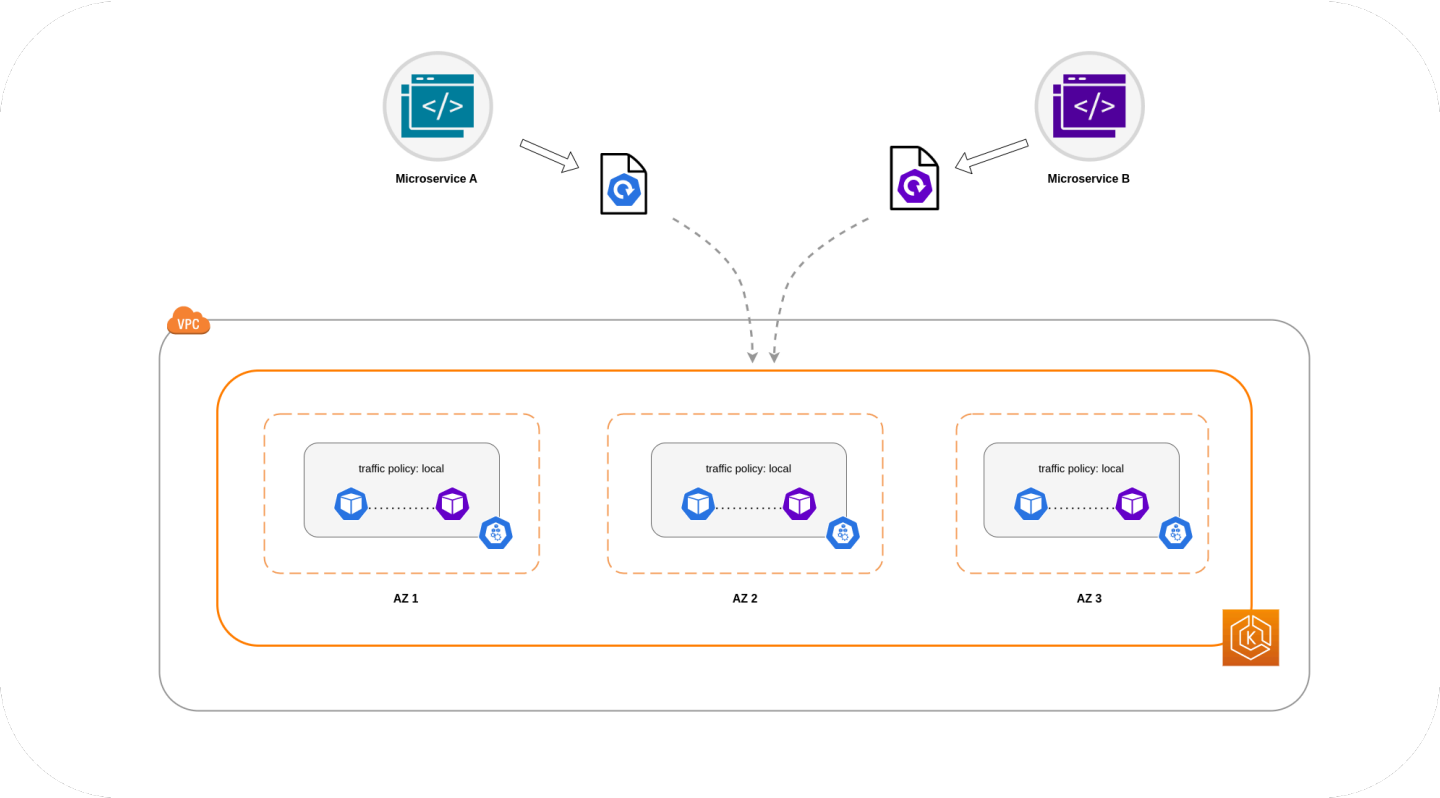
Restringindo o tráfego a um nó

Há casos em que restringir o tráfego em um nível zonal não é suficiente. Além de reduzir custos, você pode ter a necessidade adicional de reduzir a latência da rede entre determinados aplicativos que têm intercomunicação frequente. Para obter o desempenho ideal da rede e reduzir custos, você precisa de uma forma de restringir o tráfego a um nó específico. Por exemplo, o microsserviço A deve sempre se comunicar com o microsserviço B no nó 1, mesmo em configurações de alta disponibilidade (HA). Ter o Microsserviço A no Nó 1 conversando com o Microsserviço B no Nó 2 pode ter um impacto negativo no desempenho desejado para aplicativos dessa natureza, especialmente se o Nó 2 estiver em uma AZ totalmente separada.

Usando a Política de Tráfego Interno do Serviço

Para restringir o tráfego de rede do Pod a um nó, você pode usar a política de tráfego interno do Serviço. Por padrão, o tráfego enviado para o serviço de uma carga de trabalho será distribuído aleatoriamente entre os diferentes endpoints gerados. Então, em uma arquitetura de HA, isso significa que o tráfego do microsserviço A pode ir para qualquer réplica do microsserviço B em qualquer nó em diferentes. AZs No entanto, com a política de tráfego interna do Serviço definida como`Local`, o tráfego será restrito aos endpoints no nó de onde o tráfego se originou. Essa política determina o uso exclusivo de endpoints locais de nós. Por implicação, seus custos relacionados ao tráfego de rede para essa carga de trabalho serão menores do que se a distribuição fosse em todo o cluster. Além disso, a latência será menor, tornando seu aplicativo mais eficiente.

> **Note**
>
> É importante observar que esse recurso não pode ser combinado com o roteamento com reconhecimento de topologia no Kubernetes.



Abaixo está um trecho de código sobre como definir a política de tráfego interno para um serviço.

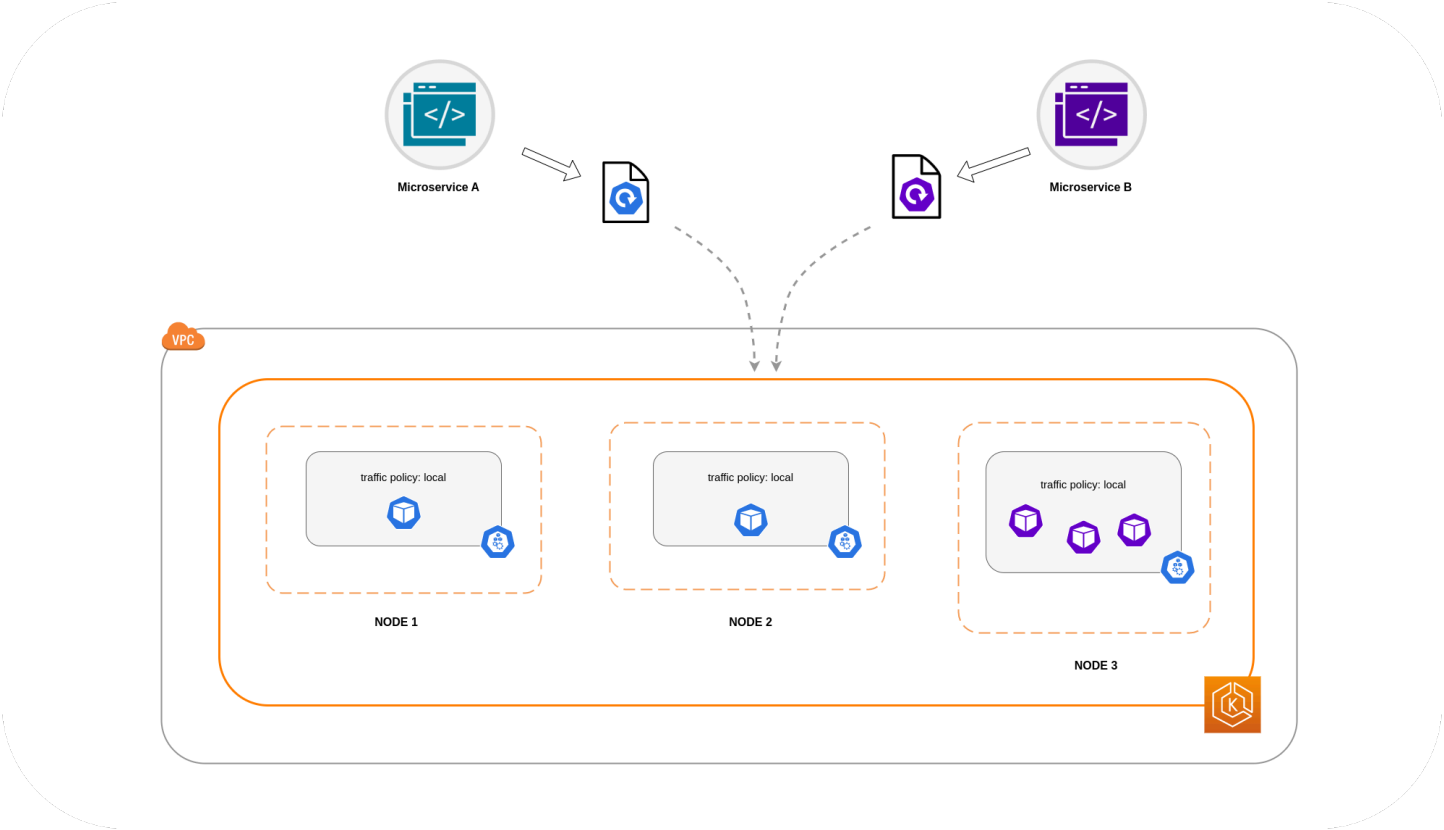
```
apiVersion: v1
kind: Service
metadata:
  name: orders-service
  namespace: ecommerce
spec:
  selector:
    app: orders
  type: ClusterIP
  ports:

* protocol: TCP
port: 3003
targetPort: 3003
  internalTrafficPolicy: Local
```

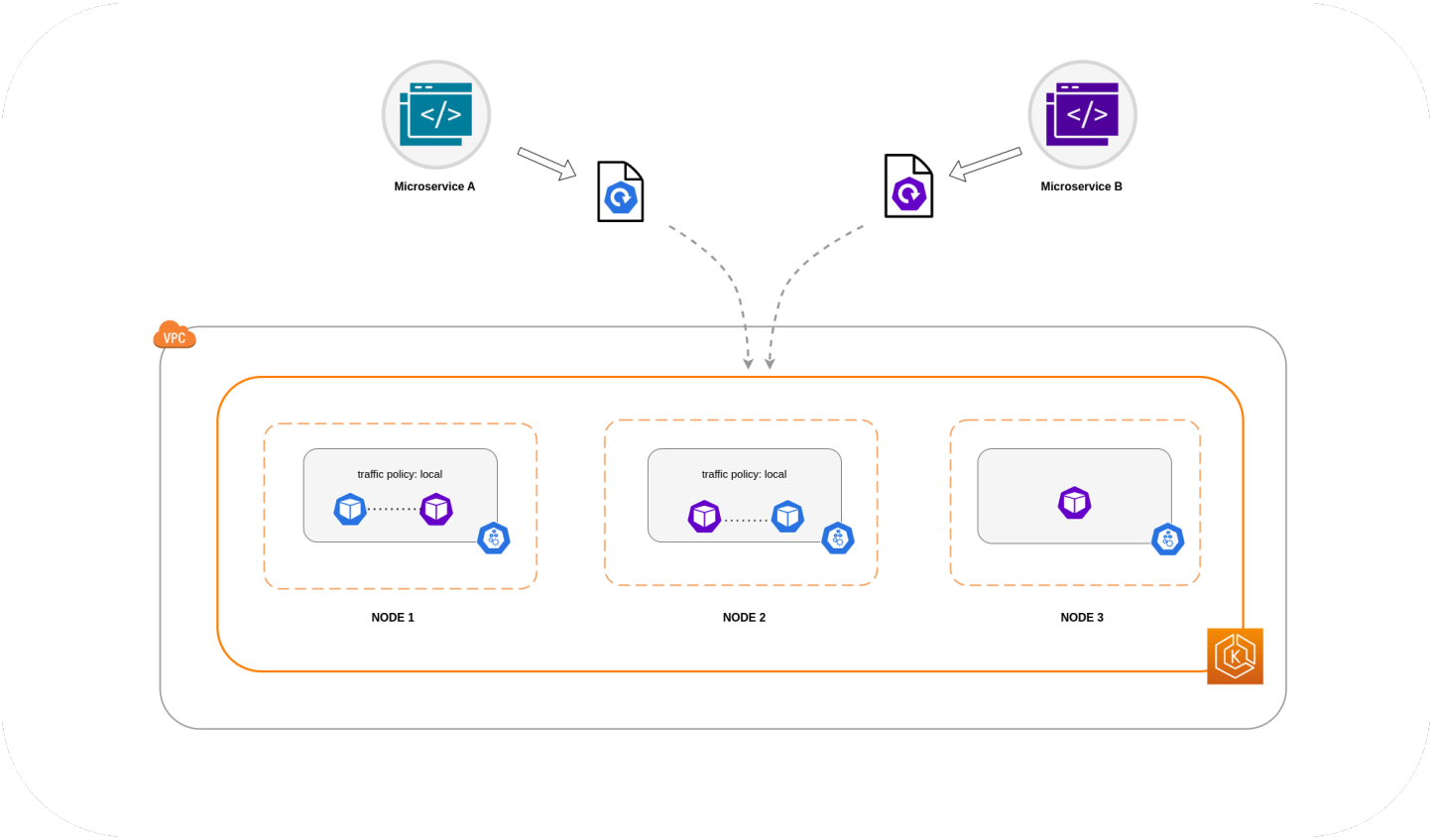
Para evitar um comportamento inesperado do seu aplicativo devido a quedas de tráfego, você deve considerar as seguintes abordagens:

- Execute réplicas suficientes para cada um dos pods comunicantes
- Tenha uma distribuição relativamente uniforme de pods usando restrições de distribuição de topologia
- Faça uso das regras de afinidade de pods para co-localização de pods comunicantes

Neste exemplo, você tem 2 réplicas do Microsserviço A e 3 réplicas do Microsserviço B. Se o Microsserviço A tiver suas réplicas espalhadas entre os nós 1 e 2, e o Microsserviço B tiver todas as suas 3 réplicas no Nó 3, eles não conseguirão se comunicar por causa da política de tráfego interna. `Local` Quando não há endpoints locais de nós disponíveis, o tráfego é interrompido.



Se o microsserviço B tiver 2 de suas 3 réplicas nos nós 1 e 2, haverá comunicação entre os aplicativos de mesmo nível. Mas você ainda teria uma réplica isolada do Microservice B sem nenhuma réplica de mesmo nível com a qual se comunicar.



> **Note**
>
> Em alguns cenários, uma réplica isolada como a mostrada no diagrama acima pode não ser motivo de preocupação se ainda servir a um propósito (como atender solicitações de tráfego externo de entrada).

Usando a Política de Tráfego Interno do Serviço com Restrições de Distribuição de Topologia

Usar a política de tráfego interno em conjunto com restrições de dispersão de topologia pode ser útil para garantir que você tenha o número certo de réplicas para comunicação de microsserviços em nós diferentes.

```
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
```

Usando a política de tráfego interno do serviço com as regras de afinidade do Pod

Outra abordagem é usar as regras de afinidade do Pod ao usar a política de tráfego interno do Service. Com a afinidade de Pod, você pode influenciar o agendador a co-localizar determinados Pods por causa de sua comunicação frequente. Ao aplicar restrições estritas de agendamento (`requiredDuringSchedulingIgnoredDuringExecution`) em determinados pods, você obterá melhores resultados para a co-localização de pods quando o programador estiver colocando pods em nós.

```
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"
```

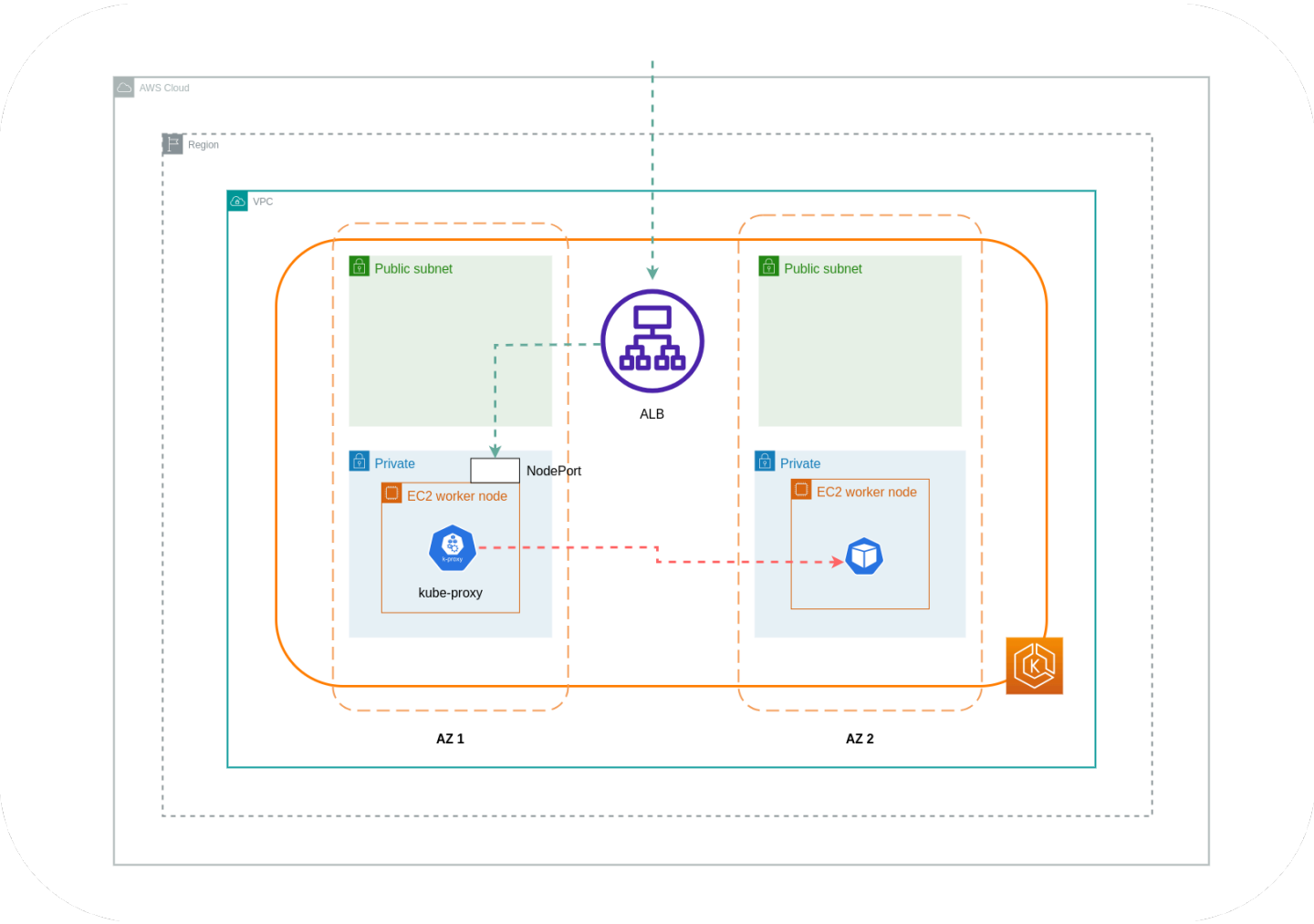
Comunicação entre o Load Balancer e o pod

As cargas de trabalho do EKS geralmente são gerenciadas por um balanceador de carga que distribui o tráfego para os pods relevantes em seu cluster EKS. Sua arquitetura pode incluir balanceadores de carga internos and/or externos. Dependendo da sua arquitetura e das configurações de tráfego de rede, a comunicação entre balanceadores de carga e pods pode contribuir significativamente para as cobranças de transferência de dados.

Você pode usar o AWS Load Balancer Controller para gerenciar automaticamente a criação de recursos do ELB (ALB e NLB). As cobranças de transferência de dados incorridas nessas configurações dependerão do caminho percorrido pelo tráfego da rede. O AWS Load Balancer Controller oferece suporte a dois modos de tráfego de rede, modo de instância e modo IP.

Ao usar o modo de instância, um NodePort será aberto em cada nó em seu cluster EKS. O balanceador de carga então distribuirá o tráfego uniformemente entre os nós. Se um nó tiver o pod de destino em execução, não haverá custos de transferência de dados. No entanto, se o pod de destino estiver em um nó separado e em um AZ diferente do NodePort receptor do tráfego, haverá um salto de rede extra do kube-proxy para o pod de destino. Nesse cenário, haverá cobranças de transferência de dados entre AZ. Devido à distribuição uniforme do tráfego entre os nós, é altamente provável que haja cobranças adicionais de transferência de dados associadas aos saltos de tráfego de rede entre zonas dos proxies kube para os pods de destino relevantes.

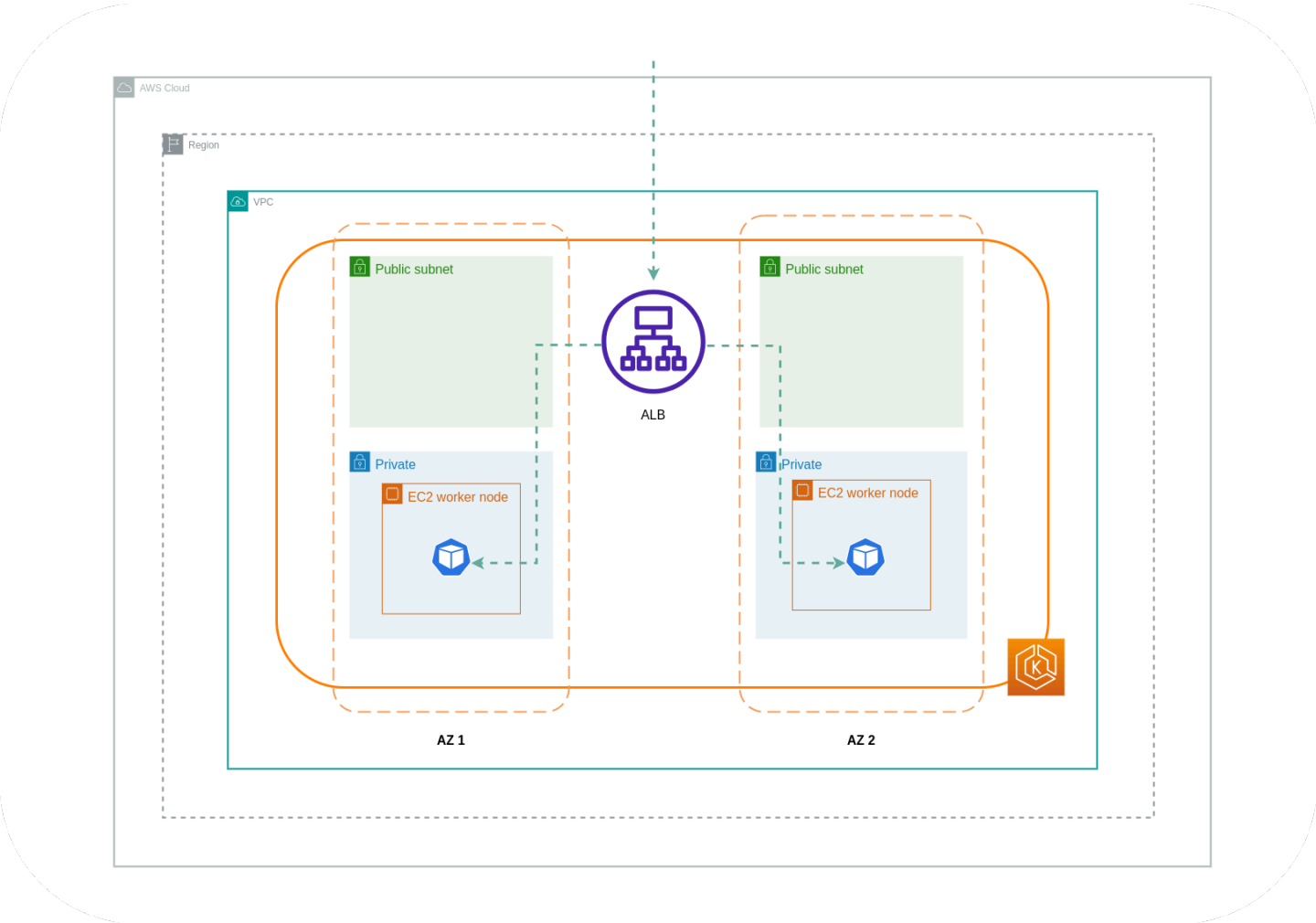
O diagrama abaixo mostra um caminho de rede para o tráfego que flui do balanceador de carga para o e NodePort, posteriormente, do `kube-proxy` pod de destino em um nó separado em uma AZ diferente. Esse é um exemplo da configuração do modo de instância.



Ao usar o modo IP, o tráfego de rede é enviado por proxy do balanceador de carga diretamente para o pod de destino. Como resultado, não há cobranças de transferência de dados envolvidas nessa abordagem.

> **Note**
>
> É recomendável que você defina seu balanceador de carga para o modo de tráfego IP para reduzir as cobranças de transferência de dados. Para essa configuração, também é importante garantir que seu balanceador de carga seja implantado em todas as sub-redes em sua VPC.

O diagrama abaixo mostra os caminhos de rede para o tráfego que flui do balanceador de carga para os pods no modo IP da rede.



Transferência de dados do Container Registry

Amazon ECR

A transferência de dados para o registro privado do Amazon ECR é gratuita. A transferência de dados na região não tem custo, mas a transferência de dados para a Internet e entre regiões será cobrada de acordo com as taxas de transferência de dados pela Internet em ambos os lados da transferência.

Você deve utilizar o recurso ECRs integrado de replicação de imagens para replicar as imagens relevantes do contêiner na mesma região das suas cargas de trabalho. Dessa forma, a replicação seria carregada uma vez e todas as capturas de imagem da mesma região (intrarregião) seriam gratuitas.

Você pode reduzir ainda mais os custos de transferência de dados associados à extração de imagens do ECR (transferência de dados para fora) usando o Interface VPC Endpoints para se

conectar aos repositórios ECR da região. A abordagem alternativa de se conectar ao endpoint público da AWS do ECR (por meio de um NAT Gateway e um Internet Gateway) acarretará maiores custos de processamento e transferência de dados. A próxima seção abordará a redução dos custos de transferência de dados entre suas cargas de trabalho e os serviços da AWS com mais detalhes.

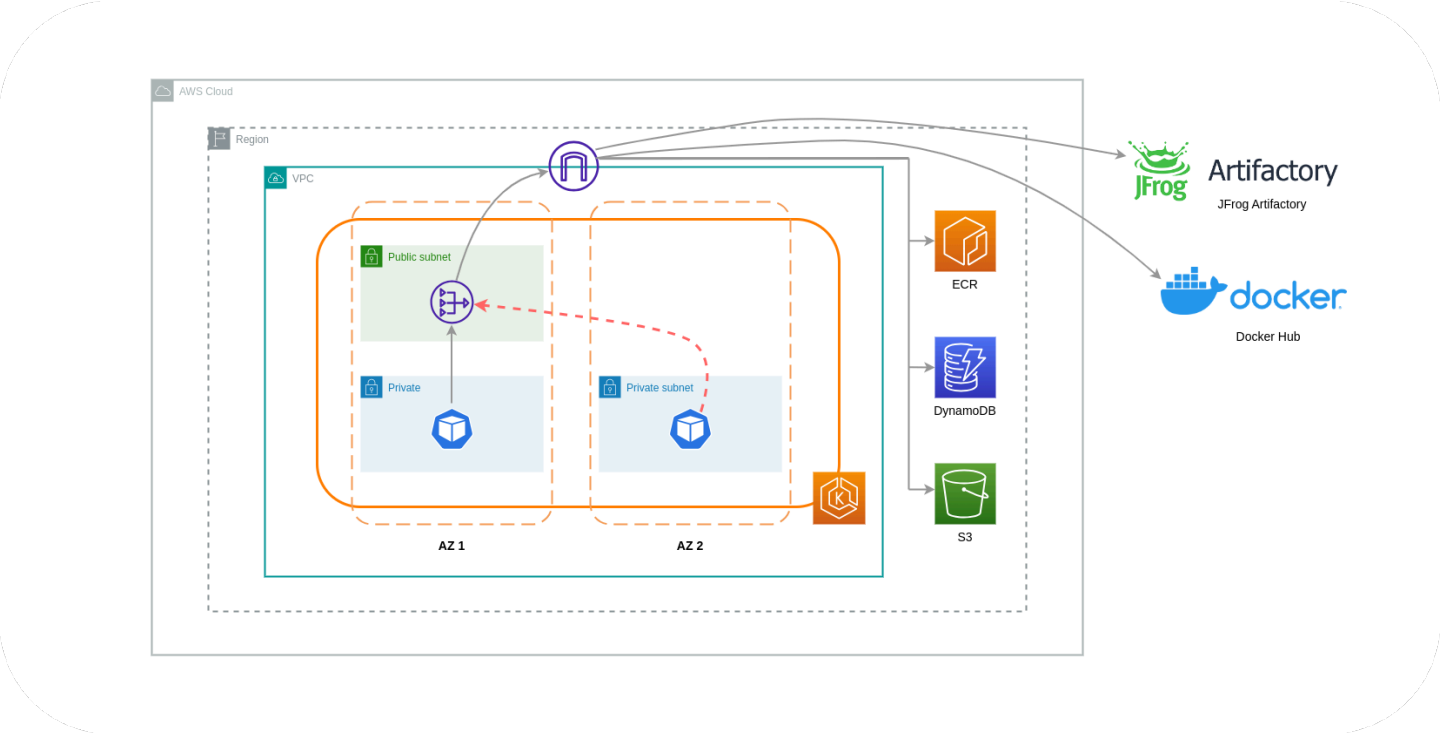
Se você estiver executando cargas de trabalho com imagens especialmente grandes, você pode criar suas próprias Amazon Machine Images (AMIs) personalizadas com imagens de contêiner pré-armazenadas em cache. Isso pode reduzir o tempo inicial de extração da imagem e os possíveis custos de transferência de dados de um registro de contêiner para os nós de trabalho do EKS.

Transferência de dados para a Internet e serviços da AWS

É uma prática comum integrar cargas de trabalho do Kubernetes com outros serviços da AWS ou ferramentas e plataformas de terceiros pela Internet. A infraestrutura de rede subjacente usada para rotear o tráfego de e para o destino relevante pode afetar os custos incorridos no processo de transferência de dados.

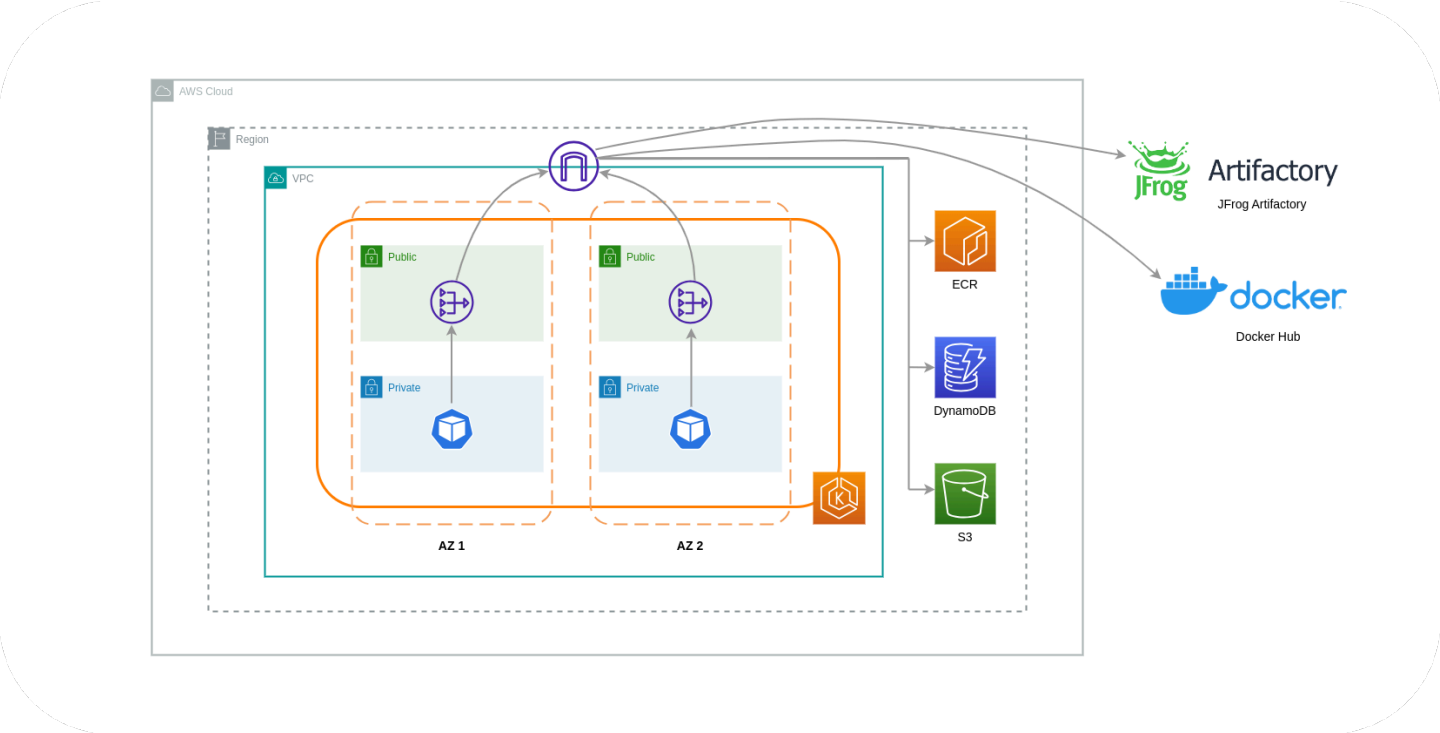
Usando gateways NAT

Os gateways NAT são componentes de rede que realizam a tradução de endereços de rede (NAT). O diagrama abaixo mostra pods em um cluster EKS se comunicando com outros serviços da AWS (Amazon ECR, DynamoDB e S3) e plataformas de terceiros. Neste exemplo, os pods estão sendo executados em sub-redes privadas separadas. AZs Para enviar e receber tráfego da Internet, um NAT Gateway é implantado na sub-rede pública de uma AZ, permitindo que qualquer recurso com endereços IP privados compartilhe um único endereço IP público para acessar a Internet. Esse NAT Gateway, por sua vez, se comunica com o componente Internet Gateway, permitindo que os pacotes sejam enviados ao seu destino final.



Ao usar gateways NAT para esses casos de uso, você pode minimizar os custos de transferência de dados implantando um gateway NAT em cada AZ. Dessa forma, o tráfego roteado para a Internet passará pelo NAT Gateway na mesma AZ, evitando a transferência de dados entre AZ. No entanto, mesmo que você economize no custo da transferência de dados Inter-AZ, a implicação dessa configuração é que você incorrerá no custo de um NAT Gateway adicional em sua arquitetura.

Essa abordagem recomendada é mostrada no diagrama abaixo.



Uso de endpoints da VPC

Para reduzir ainda mais os custos nessas arquiteturas, você deve usar VPC Endpoints para estabelecer conectividade entre suas cargas de trabalho e os serviços da AWS. Os VPC Endpoints permitem que você acesse os serviços da AWS de dentro de uma VPC sem data/network que pacotes atravessem a Internet. Todo o tráfego é interno e permanece dentro da rede da AWS. Há dois tipos de VPC Endpoints: VPC Endpoints de interface (compatíveis com vários serviços da AWS) e VPC Endpoints de gateway (compatíveis somente com S3 e DynamoDB).

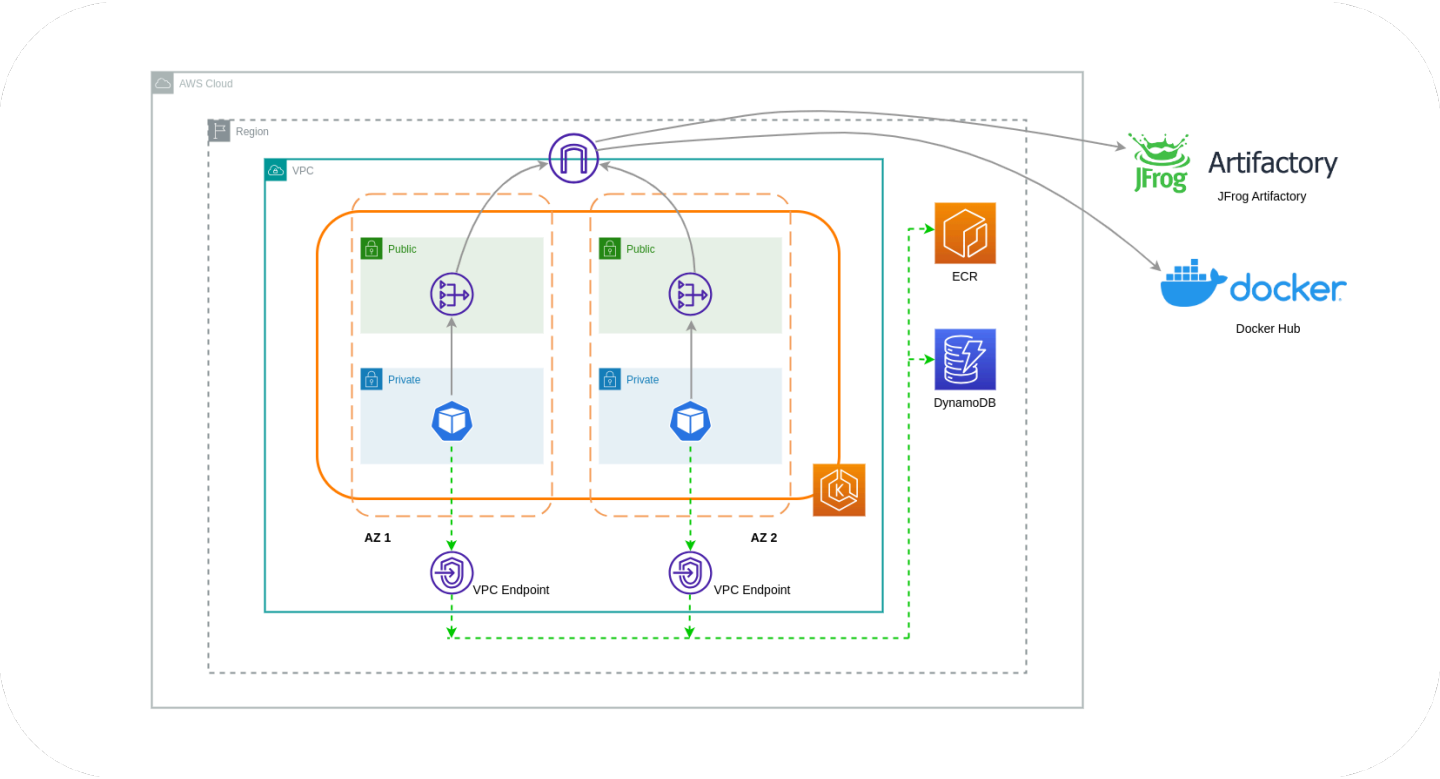
Endpoints VPC de gateway

Não há custos horários ou de transferência de dados associados aos Gateway VPC Endpoints. Ao usar os VPC Endpoints do Gateway, é importante observar que eles não podem ser estendidos além dos limites da VPC. Eles não podem ser usados em emparelhamento de VPC, redes VPN ou via Direct Connect.

Interface VPC Endpoints

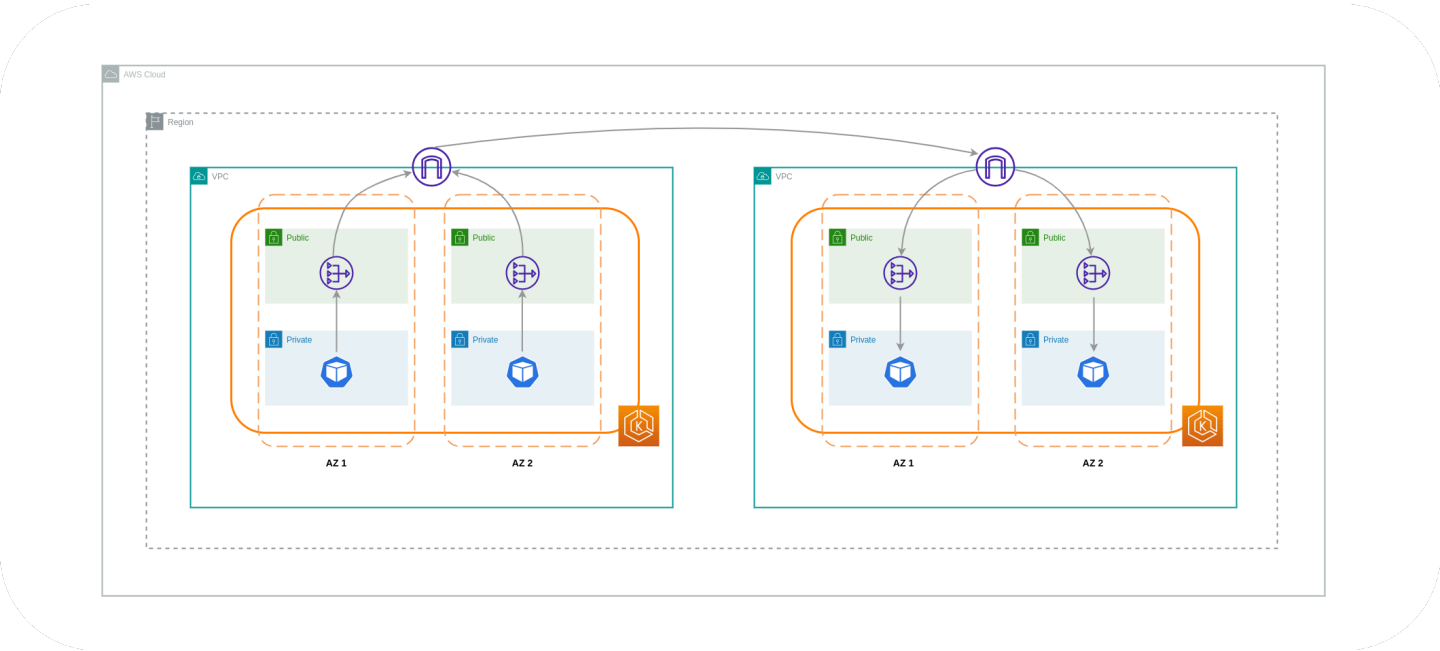
Os VPC Endpoints têm uma cobrança por hora e uma cobrança adicional associada ao processamento de dados por meio da ENI subjacente. Observe que a transferência de dados Inter-AZ [não é cobrada] (https://aws.amazon.com/about-aws/whats-new/2022/04/ - -/aws-data-transfer-price). reduction-privatelink-transit-gateway client-vpn-services

O diagrama abaixo mostra os pods se comunicando com os serviços da AWS por meio de VPC Endpoints.



Transferência de dados entre VPCs

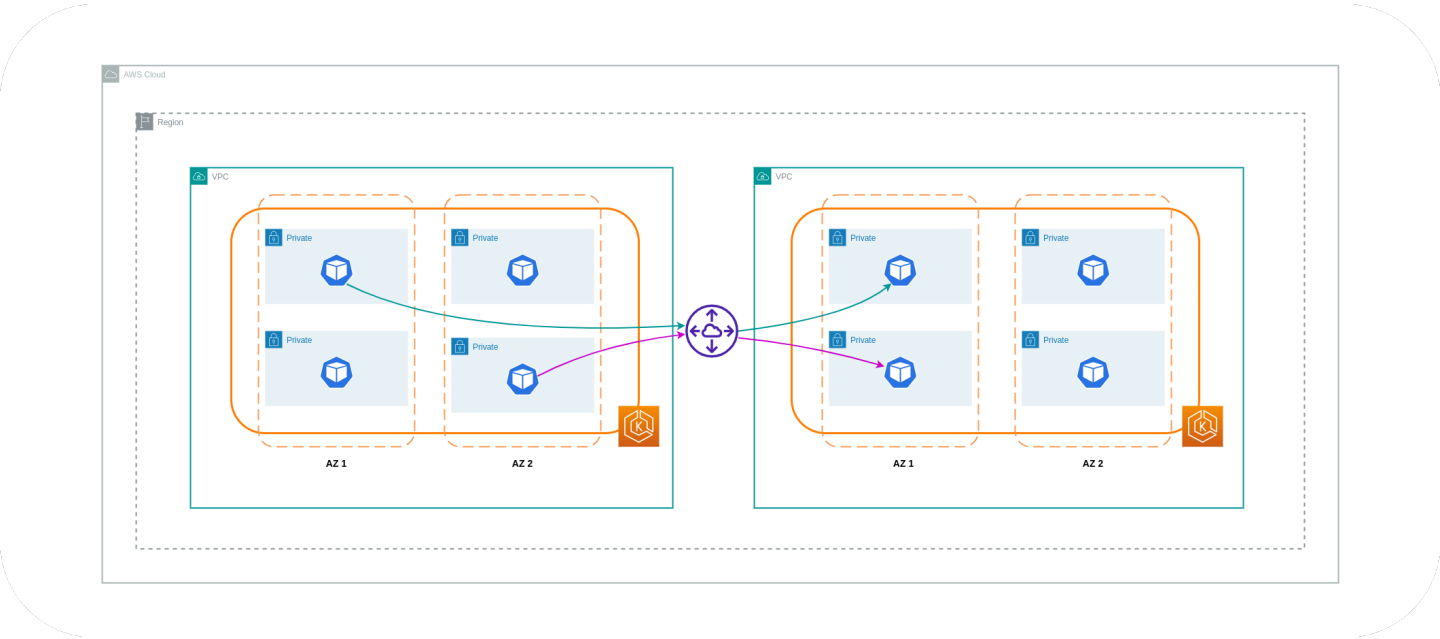
Em alguns casos, você pode ter cargas de trabalho distintas VPCs (dentro da mesma região da AWS) que precisam se comunicar entre si. Isso pode ser feito permitindo que o tráfego atravesse a Internet pública por meio de gateways de Internet conectados aos respectivos. VPCs Essa comunicação pode ser habilitada pela implantação de componentes de infraestrutura como EC2 instâncias, gateways NAT ou instâncias NAT em sub-redes públicas. No entanto, uma configuração incluindo esses componentes incorrerá em cobranças pelos processing/transferring dados que entram e saem do VPCs. Se o tráfego de e para o separado VPCs estiver se movendo AZs, haverá uma cobrança adicional na transferência de dados. O diagrama abaixo mostra uma configuração que usa gateways NAT e gateways de Internet para estabelecer comunicação entre cargas de trabalho em diferentes. VPCs



Conexões de emparelhamento de VPC

Para reduzir os custos desses casos de uso, você pode usar o VPC Peering. Com uma conexão de emparelhamento VPC, não há cobranças de transferência de dados para o tráfego de rede que permanece dentro da mesma AZ. Se o tráfego cruzar AZs, haverá um custo. No entanto, a abordagem de emparelhamento de VPC é recomendada para uma comunicação econômica entre cargas de trabalho separadas dentro da mesma região da AWS. VPCs No entanto, é importante observar que o emparelhamento de VPC é eficaz principalmente para conectividade VPC 1:1 porque não permite redes transitivas.

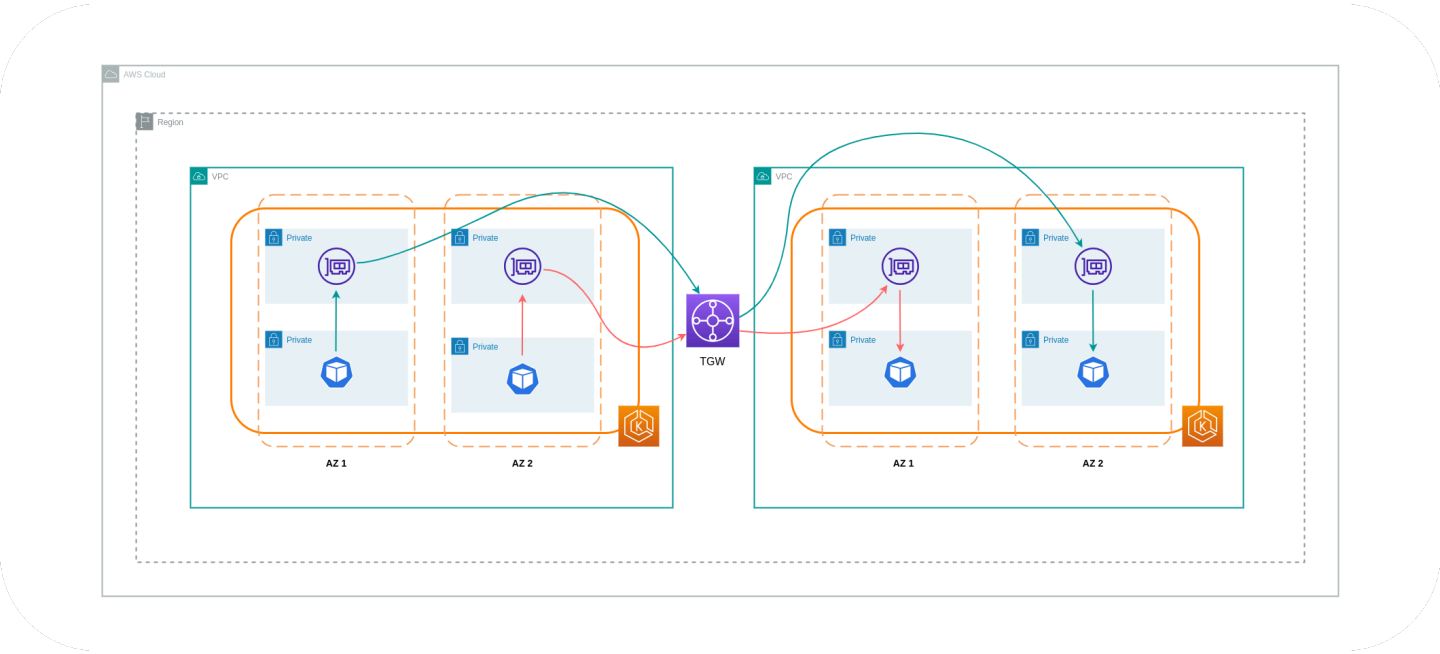
O diagrama abaixo é uma representação de alto nível da comunicação de cargas de trabalho por meio de uma conexão de emparelhamento de VPC.



Conexões de rede transitivas

Conforme apontado na seção anterior, as conexões de emparelhamento de VPC não permitem conectividade de rede transitiva. Se você quiser conectar 3 ou mais VPCs com requisitos de rede transitiva, use um Transit Gateway (TGW). Isso permitirá que você supere os limites do emparelhamento de VPC ou qualquer sobrecarga operacional associada a ter várias conexões de emparelhamento de VPC entre várias. VPCs Você é cobrado por hora e pelos dados enviados ao TGW. Não há custo de destino associado ao tráfego Inter-AZ que flui pelo TGW.

O diagrama abaixo mostra o tráfego Inter-AZ fluindo por meio de um TGW entre cargas de trabalho diferentes, VPCs mas dentro da mesma região da AWS.



Usando um Service Mesh

Os service meshes oferecem recursos de rede poderosos que podem ser usados para reduzir os custos relacionados à rede em seus ambientes de cluster EKS. No entanto, você deve considerar cuidadosamente as tarefas operacionais e a complexidade que uma malha de serviços introduzirá em seu ambiente se você adotar uma.

Restringindo o tráfego às zonas de disponibilidade

Usando a distribuição ponderada por localidade do Istio

O Istio permite que você aplique políticas de rede ao tráfego após a ocorrência do roteamento. Isso é feito usando regras de destino, como distribuição ponderada por localidade. Usando esse recurso, você pode controlar o peso (expresso em porcentagem) do tráfego que pode ir para um determinado destino com base em sua origem. A origem desse tráfego pode ser de um balanceador de carga externo (ou público) ou de um pod dentro do próprio cluster. Quando todos os endpoints do Pod estiverem disponíveis, a localidade será selecionada com base em um algoritmo ponderado de balanceamento de carga round-robin. Caso determinados endpoints não estejam íntegros ou estejam indisponíveis, o peso da localidade será ajustado automaticamente para refletir essa alteração nos endpoints disponíveis.

> **Note**
>
> Antes de implementar a distribuição ponderada por localidade, você deve começar entendendo seus padrões de tráfego de rede e as implicações que a política de regra de destino pode ter no comportamento do seu aplicativo. Dessa forma, é importante ter mecanismos de rastreamento distribuídos implementados com ferramentas como AWS X-Ray ou Jaeger.

As regras de destino do Istio detalhadas acima também podem ser aplicadas para gerenciar o tráfego de um balanceador de carga para pods em seu cluster EKS. As regras de distribuição ponderada por localidade podem ser aplicadas a um serviço que recebe tráfego de um balanceador de carga altamente disponível (especificamente o Ingress Gateway). Essas regras permitem controlar a quantidade de tráfego que vai para onde, com base em sua origem zonal - o balanceador de carga nesse caso. Se configurado corretamente, ocorrerá menos tráfego de saída entre zonas em comparação com um balanceador de carga que distribui o tráfego de maneira uniforme ou aleatória para réplicas de pod em diferentes áreas. AZs

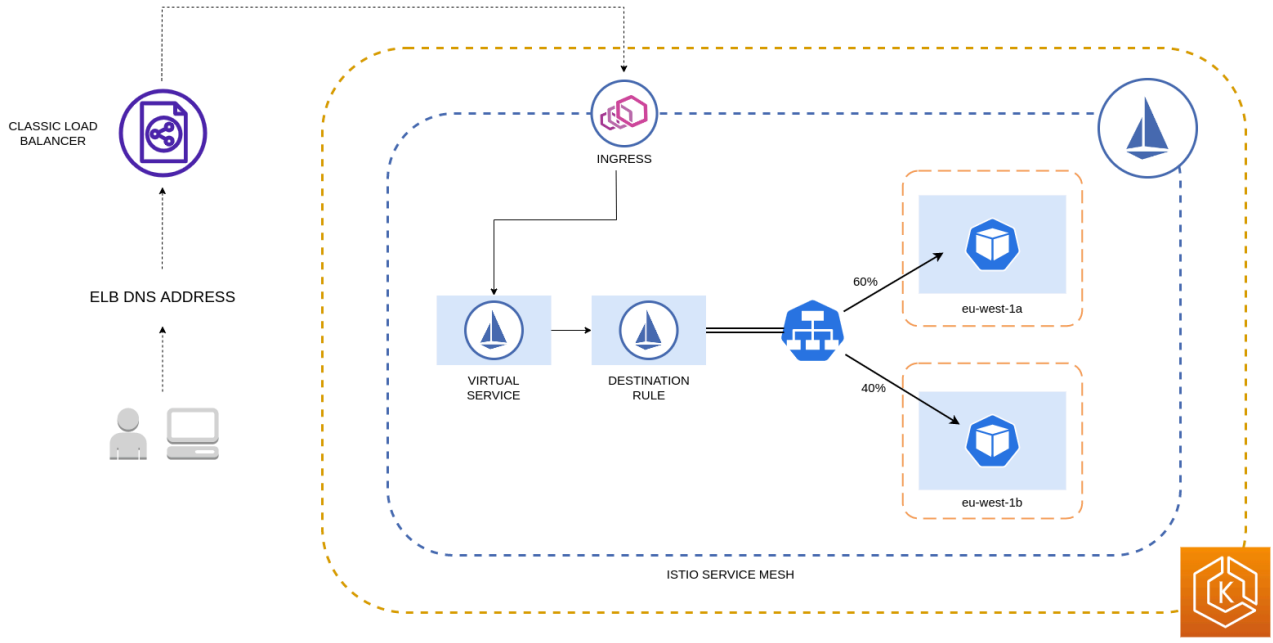
Abaixo está um exemplo de bloco de código de um recurso de regra de destino no Istio. Como pode ser visto abaixo, esse recurso especifica configurações ponderadas para tráfego de entrada de 3 diferentes AZs na região. `eu-west-1` Essas configurações declaram que a maioria do tráfego de entrada (70% nesse caso) de uma determinada AZ deve ser encaminhada por proxy para um destino na mesma AZ de onde se origina.

```
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**
>
> O peso mínimo que pode ser distribuído no destino é 1%. O motivo para isso é manter regiões e zonas de failover no caso de os endpoints no destino principal ficarem insalubres ou indisponíveis.

O diagrama abaixo mostra um cenário em que há um balanceador de carga altamente disponível na região eu-west-1 e a distribuição ponderada por localidade é aplicada. A política de regra de destino desse diagrama está configurada para enviar 60% do tráfego proveniente de eu-west-1a para pods na mesma AZ, enquanto 40% do tráfego de eu-west-1a deve ir para pods em eu-west-1b.



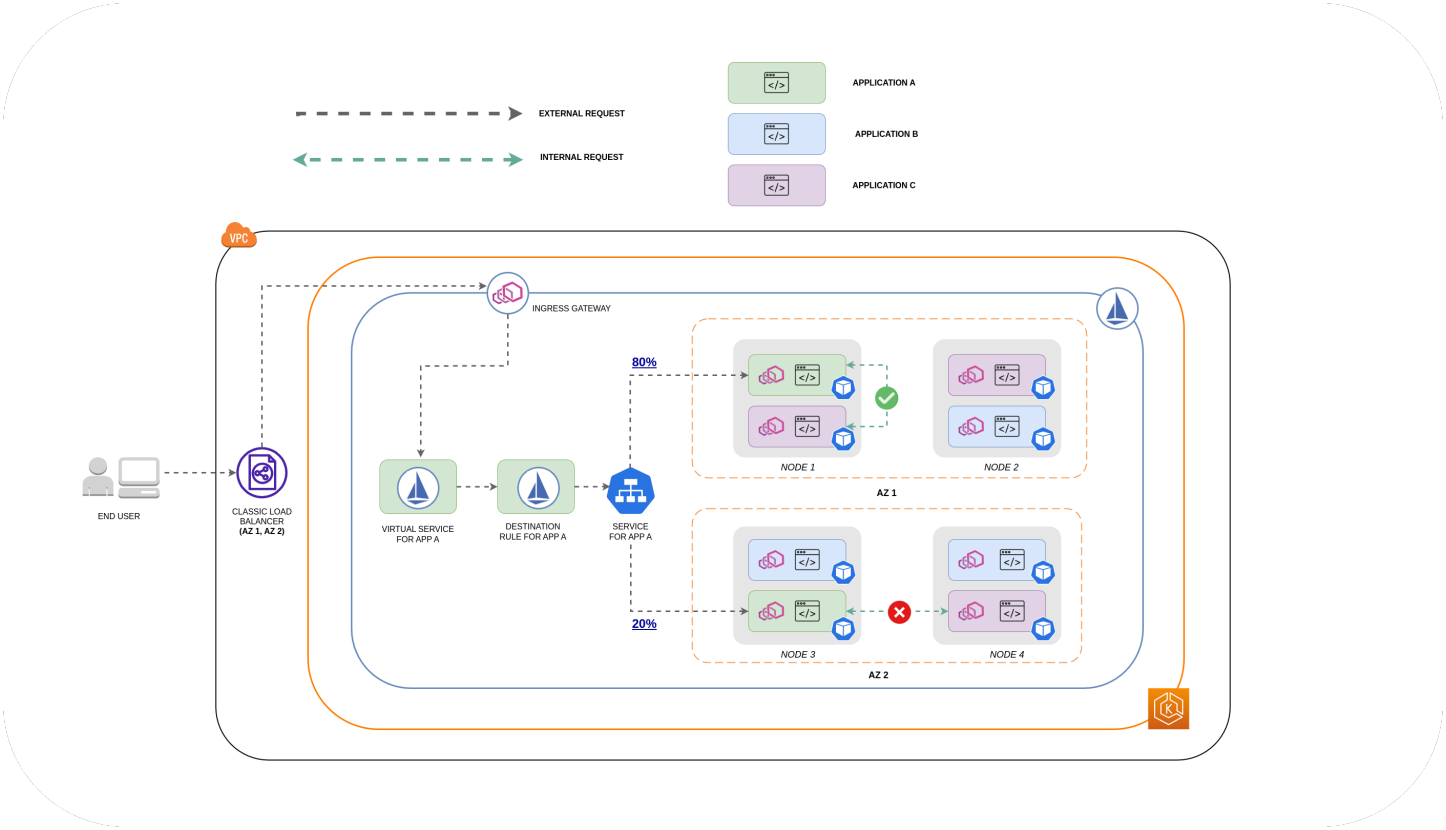
Restringindo o tráfego para zonas e nós de disponibilidade

Usando a política de tráfego interno do serviço com o Istio

Para reduzir os custos de rede associados ao tráfego externo de entrada e ao tráfego interno entre os pods, você pode combinar as regras de destino do Istio e a política de tráfego interno do Kubernetes Service. A maneira de combinar as regras de destino do Istio com a política de tráfego interno do serviço dependerá em grande parte de três coisas:

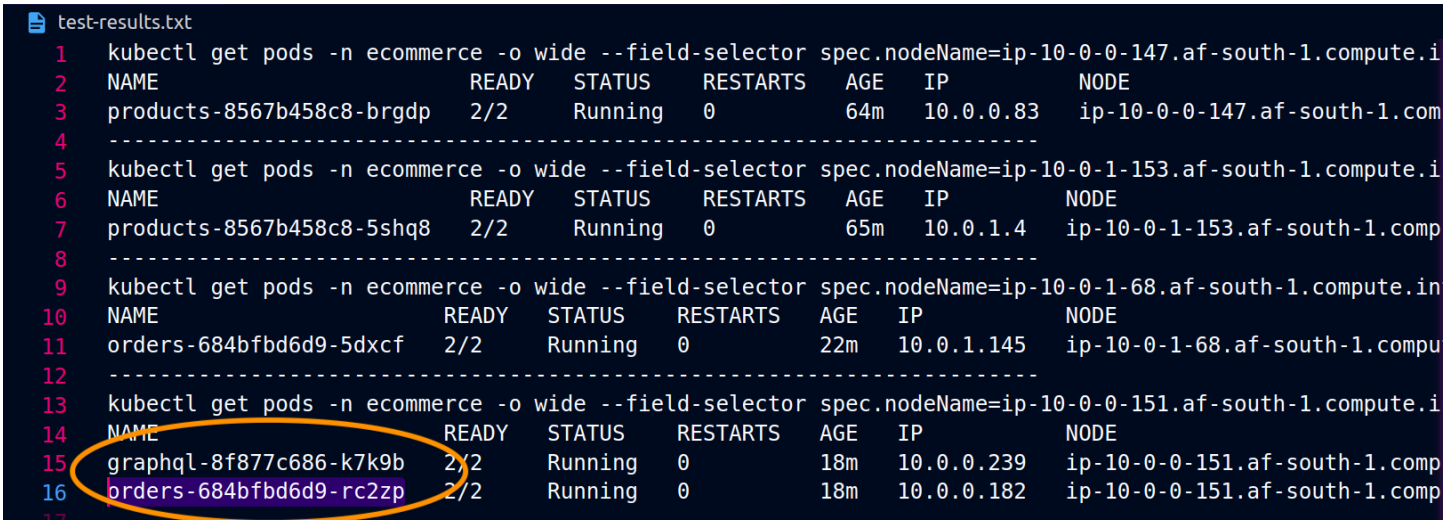
- O papel dos microserviços
- Padrões de tráfego de rede nos microsserviços
- Como os microsserviços devem ser implantados em toda a topologia de cluster do Kubernetes

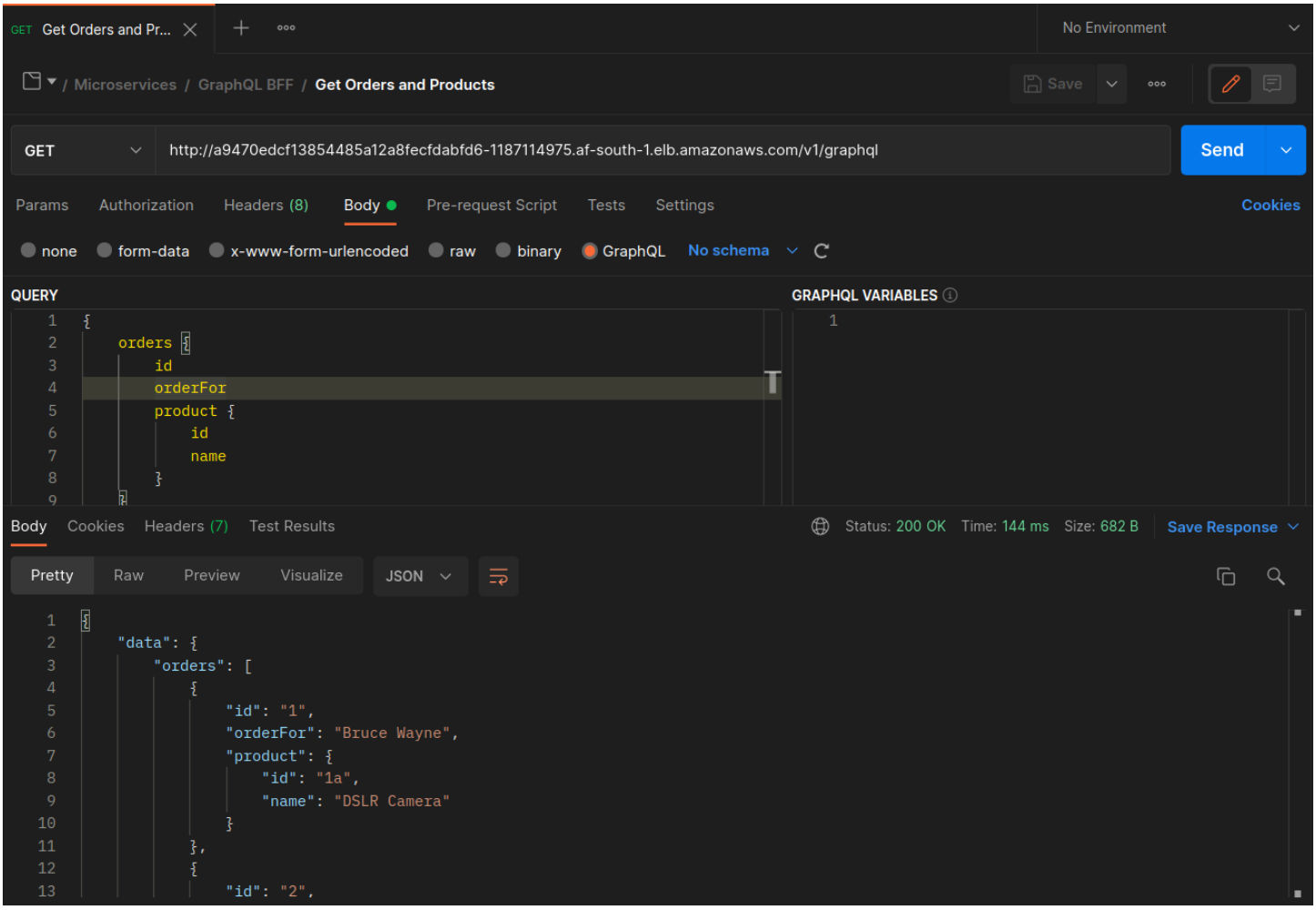
O diagrama abaixo mostra como seria o fluxo de rede no caso de uma solicitação aninhada e como as políticas mencionadas acima controlariam o tráfego.



1. O usuário final faz uma solicitação ao APP A, que por sua vez faz uma solicitação aninhada ao APP C. Essa solicitação é enviada primeiro para um balanceador de carga altamente disponível, que tem instâncias em AZ 1 e AZ 2, conforme mostra o diagrama acima.
2. A solicitação externa de entrada é então encaminhada para o destino correto pelo Istio Virtual Service.
3. Depois que a solicitação é roteada, a regra de destino do Istio controla quanto tráfego vai para o respectivo AZs com base na origem (AZ 1 ou AZ 2).
4. Em seguida, o tráfego vai para o serviço do APP A e, em seguida, é enviado por proxy para os respectivos endpoints do pod. Conforme mostrado no diagrama, 80% do tráfego de entrada é enviado para endpoints do Pod no AZ 1 e 20% do tráfego de entrada é enviado para o AZ 2.
5. O APP A então faz uma solicitação interna ao APP C. O serviço do APP C tem uma política de tráfego interna habilitada (`internalTrafficPolicy`: Local`).
6. A solicitação interna do APP A (no NODE 1) para o APP C é bem-sucedida devido ao endpoint local do nó disponível para o APP C.
7. A solicitação interna do APP A (no NODE 3) para o APP C falha porque não há endpoints locais de nó disponíveis para o APP C. Como mostra o diagrama, o APP C não tem réplicas no NODE 3. * *

As capturas de tela abaixo foram capturadas de um exemplo ao vivo dessa abordagem. O primeiro conjunto de capturas de tela demonstra uma solicitação externa bem-sucedida para um `graphql` e uma solicitação aninhada bem-sucedida do `graphql` para uma `orders` réplica co-localizada no nó. `ip-10-0-0-151.af-south-1.compute.internal`





Com o Istio, você pode verificar e exportar as estatísticas de qualquer [cluster upstream] (https://www.envoyproxy. io/docs/envoy/latest/intro/arch_overview/intro/terminology) e endpoints que seus proxies conhecem. Isso pode ajudar a fornecer uma imagem do fluxo da rede, bem como da participação da distribuição entre os serviços de uma carga de trabalho. Continuando com o mesmo exemplo, os `orders` endpoints que o `graphql` proxy conhece podem ser obtidos usando o seguinte 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**
...
```

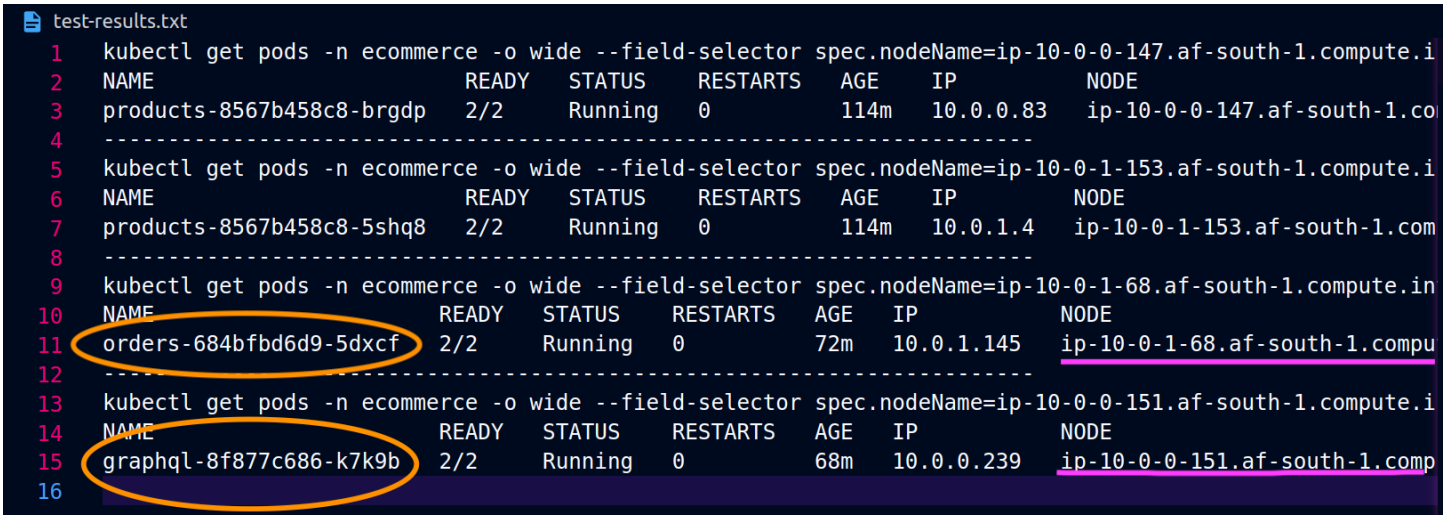
Nesse caso, o `graphql` proxy só conhece o `orders` endpoint da réplica com a qual compartilha um nó. Se você remover a `internalTrafficPolicy: Local` configuração do serviço de pedidos e executar novamente um comando como o descrito acima, os resultados retornarão todos os endpoints das réplicas espalhados pelos diferentes nós. Além disso, ao examinar os `rq_total` respectivos endpoints, você notará uma participação relativamente uniforme na distribuição da rede. Consequentemente, se os endpoints estiverem associados a serviços upstream executados de forma diferente AZs, essa distribuição de rede entre zonas resultará em custos mais altos.

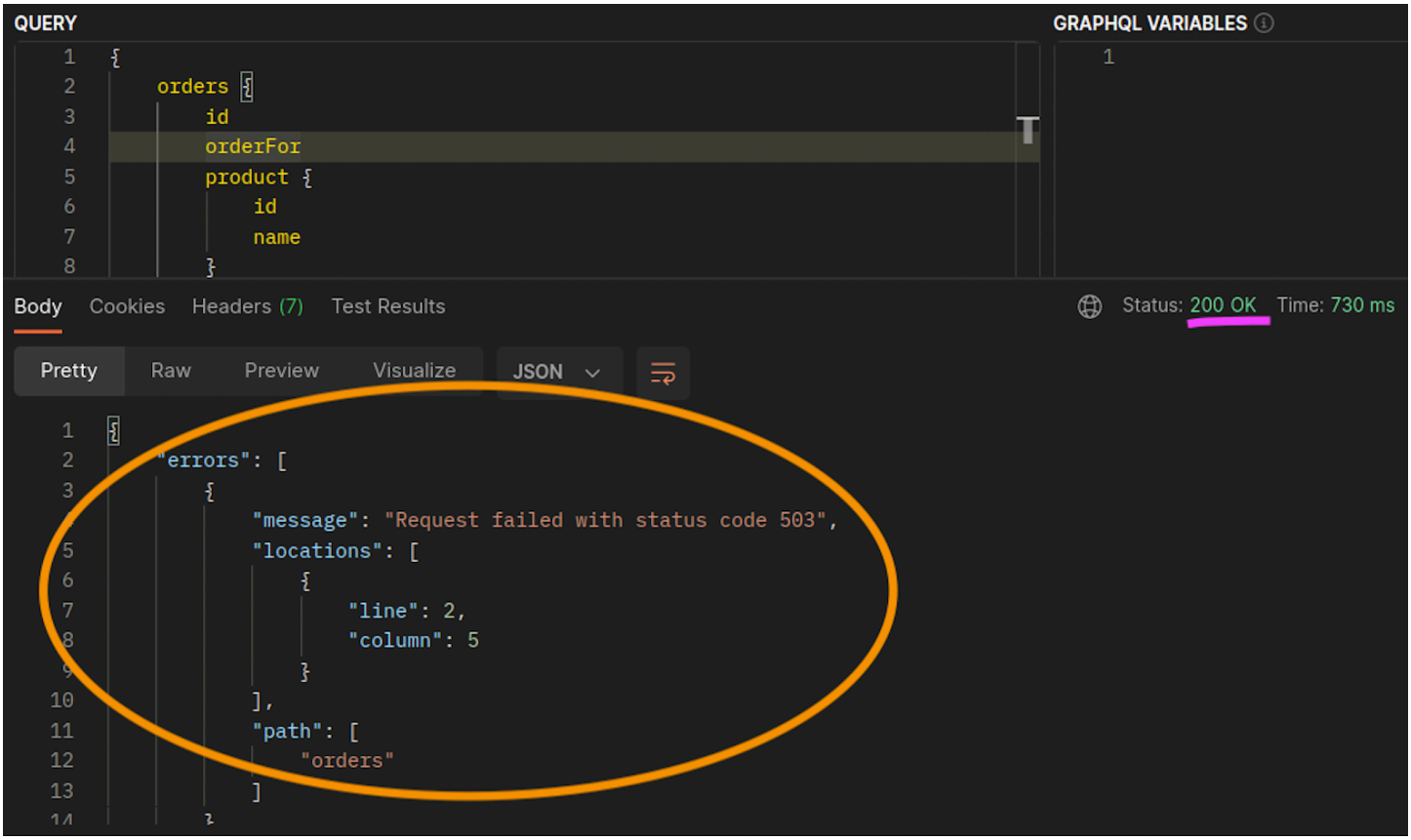
Conforme mencionado na seção anterior acima, você pode co-localizar pods que se comunicam com frequência usando a afinidade de pods.

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

Quando as `orders` réplicas `graphql` e não coexistem no mesmo node (`ip-10-0-0-151.af-south-1.compute.internal`), a primeira solicitação para `graphql` é bem-sucedida, conforme observado `200 response code` na captura de tela do Postman abaixo, enquanto a segunda solicitação aninhada de `graphql` para falha com a. `orders` `503 response code`





Recursos adicionais

- Abordando os custos de latência e transferência de dados no EKS usando o Istio
- Explorando o efeito das dicas com reconhecimento de topologia no tráfego de rede no Amazon Elastic Kubernetes Service
- Obtendo visibilidade de seus bytes de rede de pod para pod do Amazon EKS Cross-AZ

- Otimize o tráfego AZ com o Istio
- Otimize o tráfego AZ com roteamento com reconhecimento de topologia
- Otimize o custo e o desempenho do Kubernetes com a política de tráfego interno de serviços
- Otimize o custo e o desempenho do Kubernetes com a política de tráfego interno do Istio e do Service
- Visão geral dos custos de transferência de dados para arquiteturas comuns
- Entendendo os custos de transferência de dados para serviços de contêineres da AWS

Armazenamento

Visão geral

Há cenários em que talvez você queira executar aplicativos que precisam preservar os dados por um curto ou longo prazo. Para esses casos de uso, os volumes podem ser definidos e montados por pods para que seus contêineres possam acessar diferentes mecanismos de armazenamento. O Kubernetes oferece suporte a diferentes tipos de volumes para armazenamento temporário e persistente. A escolha do armazenamento depende muito dos requisitos do aplicativo. Para cada abordagem, há implicações de custo e as práticas detalhadas abaixo que ajudarão você a obter eficiência de custos para cargas de trabalho que precisam de alguma forma de armazenamento em seus ambientes EKS.

Volumes efêmeros

Os volumes temporários são para aplicativos que exigem volumes locais transitórios, mas não exigem que os dados sejam persistidos após a reinicialização. Exemplos disso incluem requisitos de espaço temporário, armazenamento em cache e dados de entrada somente para leitura, como dados de configuração e segredos. Você pode encontrar mais detalhes sobre os volumes efêmeros do Kubernetes aqui. A maioria dos volumes efêmeros (por exemplo, emptyDir, ConfigMap, DownwardAPI, secret, hostpath) é suportada por dispositivos graváveis conectados localmente (geralmente o disco raiz) ou RAM, por isso é importante escolher o volume de host mais econômico e com melhor desempenho.

Usando volumes do EBS

Recomendamos começar com gp3 como volume raiz do host. É o último volume SSD de uso geral oferecido pelo Amazon EBS e também oferece um preço mais baixo (até 20%) por GB em comparação com os volumes gp2.

Usando lojas de EC2 instâncias da Amazon

As lojas de EC2 instâncias da Amazon fornecem armazenamento temporário em nível de bloco para suas EC2 instâncias. O armazenamento fornecido pelos armazenamentos de EC2 instâncias pode ser acessado por meio de discos fisicamente conectados aos hosts. Ao contrário do Amazon EBS, você só pode anexar volumes de armazenamento de instâncias quando a instância é iniciada, e esses volumes só existem durante a vida útil da instância. Eles não podem ser desanexados e reconectados a outras instâncias. Você pode aprender mais sobre as lojas de EC2 instâncias da Amazon aqui. Não há taxas adicionais associadas a um volume de armazenamento de instâncias. Isso as torna (volumes de armazenamento de instâncias) mais econômicas do que as EC2 instâncias gerais com grandes volumes do EBS.

Para usar volumes de armazenamento local no Kubernetes, você deve particionar, configurar e formatar os discos usando os EC2 dados do usuário da Amazon para que os volumes possam ser montados conforme a especificação do pod. HostPath Como alternativa, você pode aproveitar o Provisionador Estático de Volume Persistente Local para simplificar o gerenciamento do armazenamento local. O provisionador estático Local Persistent Volume permite que você acesse volumes de armazenamento de instâncias locais por meio da interface padrão do Kubernetes PersistentVolumeClaim (PVC). Além disso, ele provisionará PersistentVolumes (PVs) que contém informações de afinidade de nós para programar pods para os nós corretos. Embora use o Kubernetes PersistentVolumes, os volumes de armazenamento de EC2 instâncias são efêmeros por natureza. Os dados gravados em discos temporários só estão disponíveis durante a vida útil da instância. Quando a instância é encerrada, os dados também são encerrados. Consulte este blog para obter mais detalhes.

Lembre-se de que, ao usar volumes de armazenamento de EC2 instâncias da Amazon, o limite total de IOPS é compartilhado com o host e vincula os pods a um host específico. Você deve analisar minuciosamente seus requisitos de carga de trabalho antes de adotar os volumes de armazenamento de EC2 instâncias da Amazon.

Volumes persistentes

O Kubernetes geralmente está associado à execução de aplicativos sem estado. No entanto, há cenários em que talvez você queira executar microsserviços que precisem preservar dados ou informações persistentes de uma solicitação para a outra. Os bancos de dados são um exemplo comum desses casos de uso. No entanto, os pods e os contêineres ou processos dentro deles são de natureza efêmera. Para manter os dados além da vida útil de um pod, você pode usar PVs para definir o acesso ao armazenamento em um local específico que seja independente do pod. Os

custos associados ao armazenamento PVs são altamente dependentes do tipo de armazenamento que está sendo usado e de como os aplicativos o estão consumindo.

Há diferentes tipos de opções de armazenamento compatíveis com o Kubernetes no PVs Amazon EKS listados aqui. As opções de armazenamento abordadas abaixo são Amazon EBS, Amazon EFS, Amazon FSx for Lustre, Amazon FSx for NetApp ONTAP.

Volumes do Amazon Elastic Block Store (EBS)

Os volumes do Amazon EBS podem ser consumidos como Kubernetes PVs para fornecer volumes de armazenamento em nível de bloco. Eles são adequados para bancos de dados que dependem de leituras e gravações aleatórias e aplicativos de alto rendimento que realizam leituras e gravações longas e contínuas. O driver Amazon Elastic Block Store Container Storage Interface (CSI) permite que os clusters do Amazon EKS gerenciem o ciclo de vida dos volumes do Amazon EBS para volumes persistentes. A Container Storage Interface permite e facilita a interação entre o Kubernetes e um sistema de armazenamento. Quando um driver CSI é implantado em seu cluster EKS, você pode acessar seus recursos por meio dos recursos de armazenamento nativos do Kubernetes, como Volumes Persistentes (PVs), Declarações de Volume Persistentes () e Classes de Armazenamento (PVCs). SCs Este link fornece exemplos práticos de como interagir com volumes do Amazon EBS com o driver CSI do Amazon EBS.

Escolhendo o volume certo

Recomendamos usar a última geração de armazenamento em bloco (gp3), pois ela fornece o equilíbrio certo entre preço e desempenho. Ele também permite que você escale o volume de IOPS e a taxa de transferência independentemente do tamanho do volume, sem precisar provisionar capacidade adicional de armazenamento em blocos. Se você estiver usando volumes gp2 no momento, é altamente recomendável migrar para volumes gp3. A postagem do blog Como migrar clusters do Amazon EKS de volumes do EBS gp2 para gp3 explica como migrar de gp2 para gp3 em clusters Amazon EKS com backup e restauração usando o recurso CSI Volume Snapshots, que exige tempo de inatividade do aplicativo.

O Amazon EBS permite alterar as características do volume, como tamanho do volume, IOPS e taxa de transferência on-line. Utilizando esse recurso, é possível migrar de gp2 para gp3 sem tempo de inatividade do aplicativo usando anotações de PVC, conforme descrito neste blog, que requer o driver EBS CSI v1.19.0+, ou começando com o Amazon EKS v1.31 e o driver EBS CSI 1.35 usando a API conforme descrito aqui. VolumeAttributesClass

Quando você tem aplicativos que exigem maior desempenho e precisam de volumes maiores do que o que um único volume gp3 pode suportar, considere usar o io2 block express. Esse tipo de

armazenamento é ideal para sua implantação maior, mais I/O intensiva e de missão crítica, como SAP HANA ou outros bancos de dados grandes com requisitos de baixa latência. Lembre-se de que o desempenho do EBS de uma instância é limitado pelos limites de desempenho da instância, portanto, nem todas as instâncias oferecem suporte a volumes expressos de blocos io2. Você pode verificar os tipos de instância compatíveis e outras considerações neste documento.

Um único volume gp3 pode suportar até 16.000 IOPS no máximo, 1.000 taxa de transferência máxima, no MiB/s máximo 16 TiB. A última geração de volume SSD de IOPS provisionadas que fornece até 256.000 IOPS, 4.000 MiB/s, taxa de transferência e 64 TiB.

Entre essas opções, você deve adaptar melhor o desempenho e o custo do armazenamento às necessidades de seus aplicativos.

Monitore e otimize ao longo do tempo

É importante entender o desempenho básico do seu aplicativo e monitorá-lo em relação aos volumes selecionados para verificar se ele está atendendo ao seu requirements/expectations ou se está superprovisionado (por exemplo, um cenário em que as IOPS provisionadas não estão sendo totalmente utilizadas).

Em vez de alocar um grande volume desde o início, você pode aumentar gradualmente o tamanho do volume à medida que acumula dados. Você pode redimensionar volumes dinamicamente usando o recurso de redimensionamento de volume no driver CSI do Amazon Elastic Block Store (). aws-ebs-csi-driver Lembre-se de que você só pode aumentar o tamanho do volume do EBS.

Para identificar e remover quaisquer volumes pendentes do EBS, você pode usar a categoria de otimização de custos do AWS Trusted Advisor. Esse recurso ajuda a identificar volumes não conectados ou com atividade de gravação muito baixa por um período de tempo. Há uma ferramenta nativa da nuvem, de código aberto e somente para leitura, chamada Popeye, que escaneia clusters Kubernetes ativos e relata possíveis problemas com recursos e configurações implantados. Por exemplo, ele pode verificar se não estão sendo usados PVs PVCs e verificar se eles estão vinculados ou se há algum erro na montagem do volume.

Para se aprofundar no monitoramento, consulte o guia de observabilidade da otimização de custos do EKS.

Outra opção que você pode considerar são as recomendações de volume do AWS Compute Optimizer Amazon EBS. Essa ferramenta identifica automaticamente a configuração ideal do volume e o nível correto de desempenho necessário. Por exemplo, ele pode ser usado para configurações

ideais relacionadas a IOPS provisionadas, tamanhos de volume e tipos de volumes do EBS com base na utilização máxima durante os últimos 14 dias. Também quantifica a possível economia mensal de custos derivada de suas recomendações. Você pode revisar este blog para obter mais detalhes.

Backup retention policy (Política de retenção de backup)

Você pode fazer backup dos dados em seus volumes do Amazon EBS tirando point-in-time snapshots. O driver CSI do Amazon EBS oferece suporte a snapshots de volume. Você pode aprender como criar um snapshot e restaurar um EBS PV usando as etapas descritas aqui.

Os instantâneos subsequentes são backups incrementais, o que significa que somente os blocos no dispositivo que foram alterados após o instantâneo mais recente são salvos. Isso minimiza o tempo necessário para criar o snapshot e economiza em custos de armazenamento ao não duplicar os dados. No entanto, aumentar o número de snapshots antigos do EBS sem uma política de retenção adequada pode causar custos inesperados ao operar em grande escala. Se você estiver fazendo backup direto dos volumes do Amazon EBS por meio da API da AWS, você pode aproveitar o Amazon Data Lifecycle Manager (DLM), que fornece uma solução de gerenciamento de ciclo de vida automatizada e baseada em políticas para snapshots do Amazon Elastic Block Store (EBS) e Amazon Machine Images () apoiado pelo EBS. AMIs O console facilita a automatização da criação, retenção e exclusão de snapshots do EBS e. AMIs

> **Note**
>
> Atualmente, não há como usar o Amazon DLM por meio do driver CSI do Amazon EBS.

Em um ambiente Kubernetes, você pode aproveitar uma ferramenta de código aberto chamada Velero para fazer backup de seus volumes persistentes do EBS. Você pode definir um sinalizador TTL ao programar a tarefa para expirar os backups. Aqui está um guia da Velero como exemplo.

Amazon Elastic File System (EFS)

O Amazon Elastic File System (EFS) é um sistema de arquivos totalmente elástico e sem servidor que permite compartilhar dados de arquivos usando a interface padrão do sistema de arquivos e a semântica do sistema de arquivos para um amplo espectro de cargas de trabalho e aplicativos. Exemplos de cargas de trabalho e aplicativos incluem Wordpress e Drupal, ferramentas para desenvolvedores, como JIRA e Git, e sistemas de notebooks compartilhados, como o Jupyter, além de diretórios pessoais.

Um dos principais benefícios do Amazon EFS é que ele pode ser montado por vários contêineres espalhados por vários nós e várias zonas de disponibilidade. Outro benefício é que você paga apenas pelo armazenamento que usa. Os sistemas de arquivos EFS aumentarão e diminuirão automaticamente à medida que você adiciona e remove arquivos, o que elimina a necessidade de planejamento de capacidade.

Para usar o Amazon EFS no Kubernetes, você precisa usar o driver Amazon Elastic File System Container Storage Interface (CSI),. aws-efs-csi-driver Atualmente, o motorista pode criar pontos de acesso dinamicamente. No entanto, o sistema de arquivos do Amazon EFS precisa ser provisionado primeiro e fornecido como uma entrada para o parâmetro da classe de armazenamento Kubernetes.

Escolhendo a classe de armazenamento EFS certa

O Amazon EFS oferece quatro classes de armazenamento.

Duas classes de armazenamento padrão:

- Padrão Amazon EFS
- Amazon EFS Standard-Infrequent Access (EFS Standard-IA)

Duas classes de armazenamento de uma zona:

- Zona única do Amazon EFS
- Amazon EFS One Zone-Infrequent Access (EFS One Zone-IA)

As classes de armazenamento de acesso infrequente (IA) são econômicas para arquivos que não são acessados todos os dias. Com o gerenciamento do ciclo de vida do Amazon EFS, você pode mover arquivos que não foram acessados durante a política de ciclo de vida (7, 14, 30, 60 ou 90 dias) para as classes de armazenamento IA, o que pode reduzir o custo de armazenamento em até 92% em comparação com as classes de armazenamento EFS Standard e EFS One Zone, respectivamente.

Com o EFS Intelligent-Tiering, o gerenciamento do ciclo de vida monitora os padrões de acesso do seu sistema de arquivos e move automaticamente os arquivos para a classe de armazenamento mais adequada.

> **Note**
>
> aws-efs-csi-driver atualmente não tem controle sobre mudanças nas classes de armazenamento, no gerenciamento do ciclo de vida ou no Intelligent-Tiering. Eles devem ser configurados manualmente no console da AWS ou por meio do EFS APIs.

> **Note**
>
> aws-efs-csi-driver não é compatível com imagens de contêiner baseadas em Windows.

> **Note**
>
> Há um problema de memória conhecido quando vol-metrics-opt-in(para emitir métricas de volume) é ativado devido à DiskUsagefunção que consome uma quantidade de memória proporcional ao tamanho do seu sistema de arquivos. Atualmente, recomendamos desativar a opção `-- vol-metrics-opt-in `em sistemas de arquivos grandes para evitar o consumo excessivo de memória. Aqui está um link de problema do github para obter mais detalhes.

Amazon FSx para Lustre

O Lustre é um sistema de arquivos paralelo de alto desempenho comumente usado em cargas de trabalho que exigem uma taxa de transferência de até centenas de GB/s latências inferiores a milissegundos por operação. Ele é usado para cenários como treinamento de aprendizado de máquina, modelagem financeira, HPC e processamento de vídeo. O Amazon FSx for Lustre fornece um armazenamento compartilhado totalmente gerenciado com escalabilidade e desempenho, perfeitamente integrado ao Amazon S3.

Você pode usar volumes de armazenamento persistente do Kubernetes apoiados pelo FSx for Lustre usando o driver CSI FSx for Lustre do Amazon EKS ou seu cluster Kubernetes autogerenciado na AWS. Consulte a documentação do Amazon EKS para obter mais detalhes e exemplos.

Link para o Amazon S3

É recomendável vincular um repositório de dados de longo prazo altamente durável que reside no Amazon S3 ao FSx seu sistema de arquivos for Lustre. Uma vez vinculados, grandes conjuntos

de dados são carregados lentamente, conforme necessário, do Amazon S3 para FSx os sistemas de arquivos Lustre. Você também pode executar suas análises e resultados no S3 e, em seguida, excluir seu sistema de arquivos Lustre.

Escolhendo as opções certas de implantação e armazenamento

FSx for Lustre oferece diferentes opções de implantação. A primeira opção é chamada de scratch e não replica dados, enquanto a segunda opção é chamada de persistente, que, como o nome indica, persiste os dados.

A primeira opção (scratch) pode ser usada para reduzir o custo do processamento temporário de dados de curto prazo. A opção de implantação persistente foi projetada para armazenamento de longo prazo que replica automaticamente os dados dentro de uma zona de disponibilidade da AWS. Ele também suporta armazenamento SSD e HDD.

Você pode configurar o tipo de implantação desejado nos parâmetros do Kubernetes do sistema de arquivos FSx for lustre. StorageClass Aqui está um link que fornece exemplos de modelos.

> **Note**
>
> Para cargas de trabalho sensíveis à latência ou cargas de trabalho que exigem os mais altos níveis de IOPs/taxa de transferência, você deve escolher o armazenamento SSD. Para cargas de trabalho focadas na taxa de transferência que não são sensíveis à latência, você deve escolher o armazenamento em HDD.

Ativar compactação de dados

Você também pode ativar a compactação de dados em seu sistema de arquivos especificando "LZ4" como o Tipo de compactação de dados. Depois de habilitado, todos os arquivos recém-gravados serão automaticamente compactados FSx para o Lustre antes de serem gravados no disco e descompactados quando forem lidos. LZ4 o algoritmo de compressão de dados não tem perdas, então os dados originais podem ser totalmente reconstruídos a partir dos dados compactados.

Você pode configurar o tipo de compactação de dados LZ4 conforme os parâmetros do Kubernetes do sistema de arquivos FSx for lustre. StorageClass A compactação é desativada quando o valor é definido como NONE, que é o padrão. Esse link fornece exemplos de modelos.

> **Note**
>
> O Amazon FSx for Lustre não é compatível com imagens de contêiner baseadas em Windows.

Amazon FSx para NetApp ONTAP

O Amazon FSx for NetApp ONTAP é um armazenamento compartilhado totalmente gerenciado baseado no sistema NetApp de arquivos ONTAP. FSx for ONTAP fornece armazenamento de arquivos compartilhado rico em recursos, rápido e flexível que é amplamente acessível a partir de instâncias computacionais Linux, Windows e macOS executadas na AWS ou localmente.

O Amazon FSx for NetApp ONTAP oferece suporte a dois níveis de armazenamento: 1/nível primário e 2/nível de pool de capacidade.

A camada primária é uma camada provisionada e baseada em SSD de alto desempenho para dados ativos e sensíveis à latência. A camada de pool de capacidade totalmente elástica é otimizada em termos de custo para dados acessados com pouca frequência, é escalada automaticamente à medida que os dados são hierarquizados e oferece petabytes de capacidade praticamente ilimitados. Você pode ativar a compactação e a desduplicação de dados no armazenamento do pool de capacidade e reduzir ainda mais a quantidade de capacidade de armazenamento que seus dados consomem. NetAppO FabricPool recurso nativo baseado em políticas da monitora continuamente os padrões de acesso aos dados, transferindo automaticamente os dados bidirecionalmente entre os níveis de armazenamento para otimizar o desempenho e o custo.

NetAppO Astra Trident da fornece orquestração dinâmica de armazenamento usando um driver CSI que permite que os clusters do Amazon EKS gerenciem o ciclo de vida de volumes persistentes apoiados PVs pela Amazon para sistemas de arquivos ONTAP. FSx NetApp Para começar, consulte Usar o Astra Trident com a Amazon FSx para NetApp ONTAP na documentação do Astra Trident.

Outras considerações

Minimize o tamanho da imagem do contêiner

Depois que os contêineres são implantados, as imagens dos contêineres são armazenadas em cache no host como várias camadas. Ao reduzir o tamanho das imagens, a quantidade de armazenamento necessária no host pode ser reduzida.

Ao usar imagens básicas reduzidas, como imagens de rascunho ou imagens de contêiner sem distribuição (que contêm somente seu aplicativo e suas dependências de tempo de execução) desde o início, você pode reduzir o custo de armazenamento, além de outros benefícios auxiliares, como reduzir a área da superfície de ataque e reduzir os tempos de extração da imagem.

Você também deve considerar o uso de ferramentas de código aberto, como o Slim.ai, que fornece uma maneira fácil e segura de criar imagens mínimas.

Várias camadas de pacotes, ferramentas, dependências de aplicativos e bibliotecas podem facilmente aumentar o tamanho da imagem do contêiner. Ao usar construções de vários estágios, você pode copiar seletivamente artefatos de um estágio para outro, excluindo tudo o que não é necessário da imagem final. Você pode conferir mais práticas recomendadas de criação de imagens aqui.

Outra coisa a considerar é por quanto tempo as imagens em cache devem ser mantidas. Talvez você queira limpar as imagens obsoletas do cache de imagens quando uma certa quantidade de disco for utilizada. Isso ajudará a garantir que você tenha espaço suficiente para a operação do host. Por padrão, o kubelet realiza a coleta de lixo em imagens não utilizadas a cada cinco minutos e em contêineres não utilizados a cada minuto.

Para configurar opções para coleta de lixo de contêineres e imagens não utilizados, ajuste o kubelet usando um arquivo de configuração e altere os parâmetros relacionados à coleta de lixo usando o tipo de recurso. *`KubeletConfiguration`*

Você pode aprender mais sobre isso na documentação do Kubernetes.

Observabilidade

Introdução

As ferramentas de observabilidade ajudam você a detectar, corrigir e investigar com eficiência suas cargas de trabalho. O custo dos dados de telemetria aumenta naturalmente à medida que o uso do EKS aumenta. Às vezes, pode ser difícil equilibrar suas necessidades operacionais e medir o que é importante para sua empresa e manter os custos de observabilidade sob controle. Este guia se concentra em estratégias de otimização de custos para os três pilares da observabilidade: registros, métricas e rastreamentos. Cada uma dessas melhores práticas pode ser aplicada de forma independente para se adequar às metas de otimização da sua organização.

Registro em log

O registro desempenha um papel vital no monitoramento e na solução de problemas dos aplicativos em seu cluster. Há várias estratégias que podem ser empregadas para otimizar os custos de registro. As estratégias de melhores práticas listadas abaixo incluem examinar suas políticas de retenção de registros para implementar controles granulares sobre por quanto tempo os dados de registro são mantidos, enviar dados de registro para diferentes opções de armazenamento com base na importância e utilizar a filtragem de registros para restringir os tipos de mensagens de registro que são armazenadas. O gerenciamento eficiente da telemetria de registros pode levar à economia de custos em seus ambientes.

Plano de controle EKS

Otimize seus registros do plano de controle

O plano de controle do Kubernetes é um conjunto de componentes que gerenciam os clusters e esses componentes enviam diferentes tipos de informações como fluxos de log para um grupo de logs na Amazon. CloudWatch Embora haja benefícios em habilitar todos os tipos de registro do plano de controle, você deve estar ciente das informações em cada registro e dos custos associados ao armazenamento de toda a telemetria do registro. Você é cobrado pelos custos padrão CloudWatch de ingestão e armazenamento de dados do Logs para registros enviados para o Amazon CloudWatch Logs a partir de seus clusters. Antes de habilitá-los, avalie se cada fluxo de log é necessário.

Por exemplo, em clusters que não são de produção, habilite seletivamente tipos de log específicos, como os registros do servidor de API, somente para análise e desative-os posteriormente. Mas para clusters de produção, nos quais talvez você não consiga reproduzir eventos e a resolução de problemas exija mais informações de registro, você pode habilitar todos os tipos de registro. Mais detalhes sobre a implementação da otimização de custos do plano de controle estão nesta postagem do blog.

Transmita registros para o S3

Outra prática recomendada de otimização de custos é transmitir os registros do plano de controle para o S3 por meio de assinaturas de CloudWatch registros. O aproveitamento das assinaturas de CloudWatch registros permite que você encaminhe seletivamente os registros para o S3, o que fornece um armazenamento de longo prazo mais econômico em comparação com a retenção de registros indefinidamente. CloudWatch Por exemplo, para clusters de produção, você pode criar

um grupo crítico de registros e aproveitar as assinaturas para transmitir esses registros para o S3 após 15 dias. Isso garantirá que você tenha acesso rápido aos registros para análise, mas também economizará custos ao mover os registros para um armazenamento mais econômico.

> **Important**
>
> A partir de 05/09/2023, os registros do EKS são classificados como registros vendidos na Amazon Logs. CloudWatch Os Vended Logs são registros de serviços específicos da AWS publicados nativamente pelos serviços da AWS em nome do cliente e disponíveis com preços de desconto por volume. Visite a página de CloudWatch preços da Amazon para saber mais sobre os preços da Vended Logs.

Plano de dados EKS

Retenção de log

A política CloudWatch de retenção padrão da Amazon é manter os registros indefinidamente e nunca expirar, incorrendo em custos de armazenamento aplicáveis à sua região da AWS. Para reduzir os custos de armazenamento, você pode personalizar a política de retenção para cada grupo de registros com base em seus requisitos de carga de trabalho.

Em um ambiente de desenvolvimento, um longo período de retenção pode não ser necessário. Mas em um ambiente de produção, você pode definir uma política de retenção mais longa para atender aos requisitos de solução de problemas, conformidade e planejamento de capacidade. Por exemplo, se você estiver executando um aplicativo de comércio eletrônico durante a alta temporada de festas, o sistema está sob carga mais pesada e podem surgir problemas que podem não ser imediatamente perceptíveis. Convém definir uma retenção de registros mais longa para solução de problemas detalhada e análise pós-evento.

Você pode configurar seus períodos de retenção no CloudWatch console da AWS ou na API da AWS com a duração de 1 dia a 10 anos com base em cada grupo de registros. Ter um período de retenção flexível pode economizar custos de armazenamento de registros e, ao mesmo tempo, manter registros críticos.

Opções de armazenamento de registros

O armazenamento é um grande fator de custos de observabilidade, portanto, é crucial otimizar sua estratégia de armazenamento de registros. Suas estratégias devem se alinhar aos requisitos de suas

cargas de trabalho e, ao mesmo tempo, manter o desempenho e a escalabilidade. Uma estratégia para reduzir os custos de armazenamento de registros é aproveitar os buckets do AWS S3 e seus diferentes níveis de armazenamento.

Encaminhe os registros diretamente para o S3

Considere encaminhar registros menos críticos, como ambientes de desenvolvimento, diretamente para o S3 em vez do Cloudwatch. Isso pode ter um impacto imediato nos custos de armazenamento de registros. Uma opção é encaminhar os registros diretamente para o S3 usando o Fluentbit. Você define isso na `[OUTPUT]` seção, o destino onde FluentBit transmite os registros do contêiner para retenção. Revise os parâmetros de configurações adicionais aqui.

```
[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
```

Encaminhar registros CloudWatch somente para análise de curto prazo

Para registros mais críticos, como ambientes de produção em que talvez seja necessário realizar análises imediatas dos dados, considere encaminhá-los para CloudWatch. Você define isso na `[OUTPUT]` seção, o destino onde FluentBit transmite os registros do contêiner para retenção. Revise os parâmetros de configurações adicionais aqui.

```
[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
```

No entanto, isso não afetará instantaneamente sua economia de custos. Para economizar ainda mais, você precisará exportar esses registros para o Amazon S3.

Exportar para o Amazon S3 a partir de CloudWatch

Para armazenar CloudWatch registros da Amazon a longo prazo, recomendamos exportar seus CloudWatch registros do Amazon EKS para o Amazon Simple Storage Service (Amazon S3). Você pode encaminhar os registros para o bucket do Amazon S3 criando uma tarefa de exportação por meio do console ou da API. Depois de fazer isso, o Amazon S3 apresenta muitas opções para reduzir ainda mais os custos. Você pode definir suas próprias regras de ciclo de vida do Amazon S3 para mover seus registros para uma classe de armazenamento que atenda às suas necessidades ou aproveitar a classe de armazenamento Amazon S3 Intelligent-Tiering para que a AWS mova automaticamente os dados para armazenamento de longo prazo com base no seu padrão de uso. Consulte este blog para obter mais detalhes. Por exemplo, para seu ambiente de produção, os registros permanecem CloudWatch por mais de 30 dias e depois são exportados para o bucket do Amazon S3. Em seguida, você pode usar o Amazon Athena para consultar os dados no bucket do Amazon S3 se precisar consultar os registros posteriormente.

Reduza os níveis de registro

Pratique o registro seletivo para seu aplicativo. Tanto seus aplicativos quanto os nós geram registros por padrão. Para os registros do seu aplicativo, ajuste os níveis de registro para se alinharem à criticidade da carga de trabalho e do ambiente. Por exemplo, o aplicativo java abaixo está gerando `INFO` registros, que é a configuração padrão típica do aplicativo e, dependendo do código, pode resultar em um alto volume de dados de log.

```
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!");    } }
```

Em um ambiente de desenvolvimento, altere seu nível de registro para`DEBUG`, pois isso pode ajudá-lo a depurar problemas ou detectar possíveis problemas antes que eles entrem em produção.

```
      log.setLevel(Level.DEBUG);
```

Em um ambiente de produção, considere modificar seu nível de registro para `ERROR` ou`FATAL`. Isso produzirá o log somente quando o aplicativo tiver erros, reduzindo a saída do log e ajudando você a se concentrar em dados importantes sobre o status do aplicativo.

```
      log.setLevel(Level.ERROR);
```

Você pode ajustar vários níveis de registro de componentes do Kubernetes. Por exemplo, se você estiver usando o Bottlerocket como seu sistema operacional EKS Node, há configurações que permitem ajustar o nível de registro do processo kubelet. Um trecho dessa configuração está abaixo. Observe o nível de registro padrão de 2, que ajusta a verbosidade de registro do processo. `kubelet`

```
[settings.kubernetes]
log-level = "2"
image-gc-high-threshold-percent = "85"
image-gc-low-threshold-percent = "80"
```

Para um ambiente de desenvolvimento, você pode definir o nível de log maior que 2 para visualizar eventos adicionais. Isso é bom para depuração. Para um ambiente de produção, você pode definir o nível como 0 para visualizar somente eventos críticos.

Aproveite os filtros

Ao usar uma configuração padrão do EKS Fluentbit para enviar registros de contêiner para o Cloudwatch, FluentBit captura e envia TODOS os registros de contêineres de aplicativos enriquecidos com metadados do Kubernetes para o Cloudwatch, conforme mostrado no bloco de configuração abaixo. `[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}
```

A `[INPUT]` seção acima está ingerindo todos os registros do contêiner. Isso pode gerar uma grande quantidade de dados que talvez não sejam necessários. A filtragem desses dados pode reduzir a quantidade de dados de log enviados e CloudWatch , portanto, reduzir seus custos. Você pode aplicar um filtro aos seus registros antes que ele saia para CloudWatch. O Fluentbit define isso na `[FILTER]` seção. Por exemplo, impedir que os metadados do Kubernetes sejam anexados aos eventos de registro pode reduzir seu volume de registros.

```
    [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

As métricas fornecem informações valiosas sobre o desempenho do seu sistema. Ao consolidar todas as métricas de recursos disponíveis ou relacionadas ao sistema em um local centralizado,

você ganha a capacidade de comparar e analisar dados de desempenho. Essa abordagem centralizada permite que você tome decisões estratégicas mais informadas, como aumentar ou reduzir os recursos. Além disso, as métricas desempenham um papel crucial na avaliação da integridade dos recursos, permitindo que você tome medidas proativas quando necessário. Geralmente, os custos de observabilidade aumentam com a coleta e retenção de dados de telemetria. Abaixo estão algumas estratégias que você pode implementar para reduzir o custo da telemetria métrica: coletar somente métricas que importam, reduzir a cardinalidade de seus dados de telemetria e ajustar a granularidade de sua coleta de dados de telemetria.

Monitore o que importa e colete somente o que você precisa

A primeira estratégia de redução de custos é reduzir o número de métricas que você está coletando e, por sua vez, reduzir os custos de retenção.

1. Comece analisando seus requisitos e/ou os de suas partes interessadas para determinar as métricas mais importantes. As métricas de sucesso são diferentes para cada pessoa! Saiba qual é a aparência bonita e meça para isso.
2. Considere mergulhar profundamente nas cargas de trabalho que você está suportando e identificar seus principais indicadores de desempenho (KPIs), também conhecidos como “Sinais Dourados”. Eles devem estar alinhados aos requisitos dos negócios e das partes interessadas. Calcular SLIs e SLAs usar a Amazon CloudWatch e o Metric Math são cruciais para gerenciar a confiabilidade do serviço. SLOs Siga as melhores práticas descritas neste guia para monitorar e manter com eficácia o desempenho do seu ambiente EKS.
3. Em seguida, continue percorrendo as diferentes camadas da infraestrutura para conectar e correlacionar o cluster, o nó e as métricas de infraestrutura adicionais do EKS à sua carga de trabalho KPIs. Armazene suas métricas de negócios e métricas operacionais em um sistema onde você possa correlacioná-las e tirar conclusões com base nos impactos observados em ambas.
4. O EKS expõe métricas do plano de controle, cluster kube-state-metrics, pods e nós. A relevância de todas essas métricas depende de suas necessidades, mas é provável que você não precise de todas as métricas nas diferentes camadas. Você pode usar este guia de métricas essenciais do EKS como base para monitorar a integridade geral de um cluster EKS e suas cargas de trabalho.

Aqui está um exemplo de configuração do prometheus scrape em que estamos usando o `relabel_config` para manter apenas as métricas do kubelet e eliminar todas as métricas do contêiner. `metric_relabel_config`

```
  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
```

Reduza a cardinalidade quando aplicável

Cardinalidade se refere à exclusividade dos valores dos dados em combinação com suas dimensões (por exemplo, rótulos prometheus) para um conjunto de métricas específico. Métricas de alta cardinalidade têm muitas dimensões e cada combinação métrica de dimensão tem maior exclusividade. Uma maior cardinalidade resulta em maior tamanho de dados de telemetria métrica e necessidades de armazenamento, o que aumenta o custo.

No exemplo de alta cardinalidade abaixo, vemos que a métrica, a latência, tem dimensões, requestid, customerID e serviço, e cada dimensão tem muitos valores exclusivos. Cardinalidade é a medida da combinação do número de valores possíveis por Dimensão. No Prometheus, cada conjunto de dimensões/rótulos exclusivos é considerado como uma nova métrica, portanto, alta cardinalidade significa mais métricas.

Em ambientes EKS com muitas métricas e dimensões/rótulos por métrica (cluster, namespace, serviço, pod, contêiner etc.), a cardinalidade tende a crescer. Para otimizar o custo, considere cuidadosamente a cardinalidade das métricas que você está coletando. Por exemplo, se você estiver agregando uma métrica específica para visualização no nível do cluster, poderá eliminar rótulos adicionais que estejam em uma camada inferior, como o rótulo do namespace.

Para identificar métricas de alta cardinalidade no prometheus, você pode executar a seguinte consulta PROMQL para determinar quais alvos de scrape têm o maior número de métricas (cardinalidade):

```
topk_max(5, max_over_time(scrape_samples_scraped[1h]))
```

e a consulta PROMQL a seguir pode ajudá-lo a determinar quais alvos de scrape têm as maiores taxas de rotatividade de métricas (quantas novas séries de métricas foram criadas em um determinado scrape):

```
topk_max(5, max_over_time(scrape_series_added[1h]))
```

Se você estiver usando o grafana, poderá usar o Mimirtool do Grafana Lab para analisar seus painéis do grafana e as regras do prometheus para identificar métricas de alta cardinalidade não utilizadas. Siga este guia sobre como usar os `mimirtool analyze prometheus` comandos `mimirtool analyze` e para identificar métricas ativas que não são referenciadas em seus painéis.

Considere a granularidade métrica

Coletar métricas com maior granularidade, como a cada segundo versus cada minuto, pode ter um grande impacto na quantidade de telemetria coletada e armazenada, o que aumenta o custo. Determine intervalos sensatos ou de coleta de métricas que se equilibrem entre granularidade suficiente para detectar problemas transitórios e baixos o suficiente para serem econômicos. Diminua a granularidade das métricas usadas para planejamento de capacidade e análise de maior janela de tempo.

Abaixo está um trecho da configuração padrão do AWS Distro for Opentelemetry (ADOT) EKS Addon Collector.

> **Important**
>
> o intervalo global de raspagem do prometheus é definido como 15s. Esse intervalo de raspagem pode ser aumentado, resultando em uma diminuição na quantidade de dados métricos coletados no 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
```

Rastreamento

O principal custo associado ao rastreamento decorre da geração de armazenamento de traços. Com o rastreamento, o objetivo é reunir dados suficientes para diagnosticar e entender os aspectos de desempenho. No entanto, como os custos dos rastreamentos do X-Ray são baseados nos dados encaminhados para o X-Ray, apagar os traços após o encaminhamento não reduzirá seus custos. Vamos analisar formas de reduzir seus custos de rastreamento e, ao mesmo tempo, manter os dados para que você realize a análise adequada.

Aplicar regras de amostragem

A taxa de amostragem do X-Ray é conservadora por padrão. Defina regras de amostragem nas quais você possa controlar a quantidade de dados coletados. Isso melhorará a eficiência do desempenho e reduzirá os custos. Ao diminuir a taxa de amostragem, você pode coletar rastreamentos da solicitação somente do que suas cargas de trabalho precisam, mantendo uma estrutura de custos mais baixa.

Por exemplo, você tem um aplicativo java que deseja depurar os rastreamentos de todas as solicitações de uma rota problemática.

Configure por meio do SDK para carregar regras de amostragem de um 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
  }
}
```

Por meio do console

Aplique o Tail Sampling com o AWS Distro for OpenTelemetry (ADOT)

O ADOT Tail Sampling permite que você controle o volume de traços ingeridos no serviço. No entanto, o Tail Sampling permite que você defina as políticas de amostragem após a conclusão de todos os períodos da solicitação, em vez de no início. Isso limita ainda mais a quantidade de dados brutos transferidos CloudWatch, reduzindo assim os custos.

Por exemplo, se você estiver coletando amostras de 1% do tráfego para uma página de destino e 10% das solicitações para uma página de pagamento, isso pode deixar 300 rastros por um período de 30 minutos. Com uma regra ADOT Tail Sampling que filtra erros específicos, você pode ficar com 200 traços, o que diminui o número de traços armazenados.

```
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]
```

Aproveite as opções de armazenamento do Amazon S3

Você deve aproveitar o bucket AWS S3 e suas diferentes classes de armazenamento para armazenar os rastreamentos. Exporte rastreamentos para o S3 antes que o período de retenção expire. Use as regras de ciclo de vida do Amazon S3 para mover os dados de rastreamento para a classe de armazenamento que atenda aos seus requisitos.

Por exemplo, se você tiver rastros com 90 dias, o Amazon S3 Intelligent-Tiering pode mover automaticamente os dados para armazenamento de longo prazo com base no seu padrão de uso. Você pode usar o Amazon Athena para consultar os dados no Amazon S3 se precisar consultar os rastreamentos posteriormente. Isso pode reduzir ainda mais o custo do rastreamento distribuído.

Recursos adicionais:

- Guia de melhores práticas de observabilidade
- Coleção de métricas de melhores práticas
- AWS re:Invent 2022 — Melhores práticas de observabilidade na Amazon () COP343
- AWS re:Invent 2022 — Observabilidade: melhores práticas para aplicativos modernos () COP344

Práticas recomendadas para Windows

Este guia fornece conselhos sobre como executar contêineres e nós do Windows.

Tópicos

- Gerenciamento otimizado de AMI do Windows para Amazon EKS
- Configurar o gMSA para pods e contêineres do Windows
- Fortalecimento dos nós de trabalho do Windows
- Digitalização de imagens de containers
- Versão e licença do Windows Server
- Registro em log
- Monitoramento
- Rede Windows
- Evitando erros de OOM
- Aplicação de patches em servidores e contêineres Windows
- Executando cargas de trabalho heterogêneas
- Contextos de segurança do pod
- Opções de armazenamento persistente
- Fortalecimento de imagens de contêineres do Windows

Gerenciamento otimizado de AMI do Windows para Amazon EKS

O Windows Amazon EKS otimizado AMIs foi desenvolvido com base no Windows Server 2019 e no Windows Server 2022. Elas são configuradas para servirem como imagem base para nós do Amazon EKS. Por padrão, AMIs incluem os seguintes componentes:

- kubelet
- kube-proxy
- AWS IAM Authenticator para Kubernetes
- csi-proxy
- containerd

Você pode recuperar programaticamente o ID da Amazon Machine Image (AMI) para o Amazon EKS otimizado AMIs consultando a API AWS Systems Manager Parameter Store. Esse parâmetro elimina a necessidade de você pesquisar manualmente a AMI otimizada do Amazon EKS IDs. Para obter mais informações sobre a API Systems Manager Parameter Store, consulte GetParameter. Sua conta de usuário deve ter a permissão ssm: GetParameter IAM para recuperar os metadados otimizados da AMI do Amazon EKS.

O exemplo a seguir recupera o ID da AMI da AMI otimizada mais recente do Amazon EKS para Windows Server 2019 LTSC Core. O número da versão listado no nome da AMI está relacionado à compilação correspondente do Kubernetes para a qual ela 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
```

Resultado do exemplo:

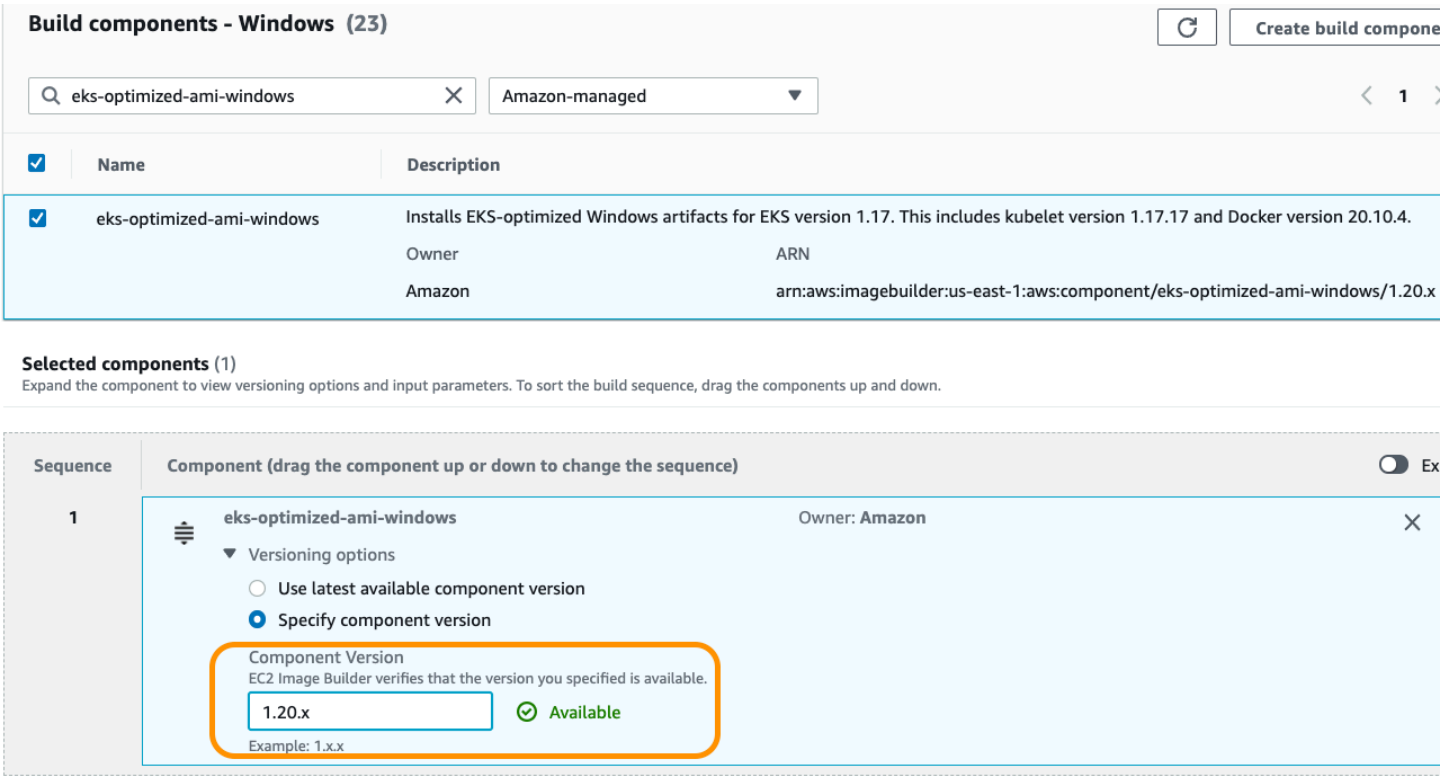
```
ami-09770b3eec4552d4e
```

Gerenciando sua própria AMI Windows otimizada para Amazon EKS

Uma etapa essencial em direção aos ambientes de produção é manter a mesma AMI do Windows otimizada para Amazon EKS e a mesma versão kubelet em todo o cluster do Amazon EKS.

Usar a mesma versão em todo o cluster Amazon EKS reduz o tempo durante a solução de problemas e aumenta a consistência do cluster. O Amazon EC2 Image Builder ajuda a criar e manter janelas personalizadas otimizadas para o Amazon EKS AMIs para serem usadas em um cluster do Amazon EKS.

Use o Amazon EC2 Image Builder para selecionar entre as versões do Windows Server, as datas de lançamento da AMI do AWS Windows Server e a versão de compilação do and/or sistema operacional. A etapa de criação de componentes permite que você selecione entre os artefatos do Windows otimizados para EKS existentes e as versões do kubelet. Para obter mais informações: https://docs.aws.amazon.com/eks/ latest/userguide/eks - custom-ami-windows .html



OBSERVAÇÃO: Antes de selecionar uma imagem base, consulte a seção Versão e Licença do Windows Server para obter detalhes importantes sobre as atualizações do canal de lançamento.

Configurando um lançamento mais rápido para EKS personalizado otimizado AMIs

Ao usar uma AMI personalizada otimizada para Windows Amazon EKS, os nós de trabalho do Windows podem ser iniciados até 65% mais rápido ao ativar o recurso Fast Launch. Esse recurso mantém um conjunto de instantâneos pré-provisionados que têm a especialização Sysprep, as etapas do Windows Out of Box Experience (OOBE) e as reinicializações necessárias já concluídas. Esses instantâneos são então usados em lançamentos subsequentes, reduzindo o tempo de expansão ou substituição de nós. O Fast Launch só pode ser habilitado para AMIs você por meio do EC2 console ou da AWS CLI, e o número de snapshots mantidos é configurável.

OBSERVAÇÃO: O Fast Launch não é compatível com a AMI otimizada para EKS padrão fornecida pela Amazon. Crie uma AMI personalizada conforme descrito acima antes de tentar ativá-la.

Para obter mais informações: AWS Windows AMIs — Configure sua AMI para acelerar o lançamento

Armazenando camadas base do Windows em cache de forma personalizada AMIs

As imagens de contêiner do Windows são maiores do que as do Linux. Se você estiver executando qualquer aplicativo em contêineres baseado em o.NET Framework, o tamanho médio da imagem é de cerca de 8,24 GB. Durante o agendamento do pod, a imagem do contêiner deve ser totalmente extraída e extraída no disco antes que o pod alcance o status Running.

Durante esse processo, o tempo de execução do contêiner (containerd) extrai e extrai toda a imagem do contêiner no disco. A operação de extração é um processo paralelo, o que significa que o tempo de execução do contêiner puxa as camadas da imagem do contêiner em paralelo. Por outro lado, a operação de extração ocorre em um processo sequencial e é I/O intensiva. Por isso, a imagem do contêiner pode levar mais de 8 minutos para ser totalmente extraída e pronta para ser usada pelo tempo de execução do contêiner (containerd) e, como resultado, o tempo de inicialização do pod pode levar vários minutos.

Conforme mencionado no tópico Patching Windows Server and Container, há uma opção para criar uma AMI personalizada com o EKS. Durante a preparação da AMI, você pode adicionar um componente adicional do EC2 Image Builder para extrair localmente todas as imagens de contêiner do Windows necessárias e gerar a AMI. Essa estratégia reduzirá drasticamente o tempo em que um pod atinge o status Executando.

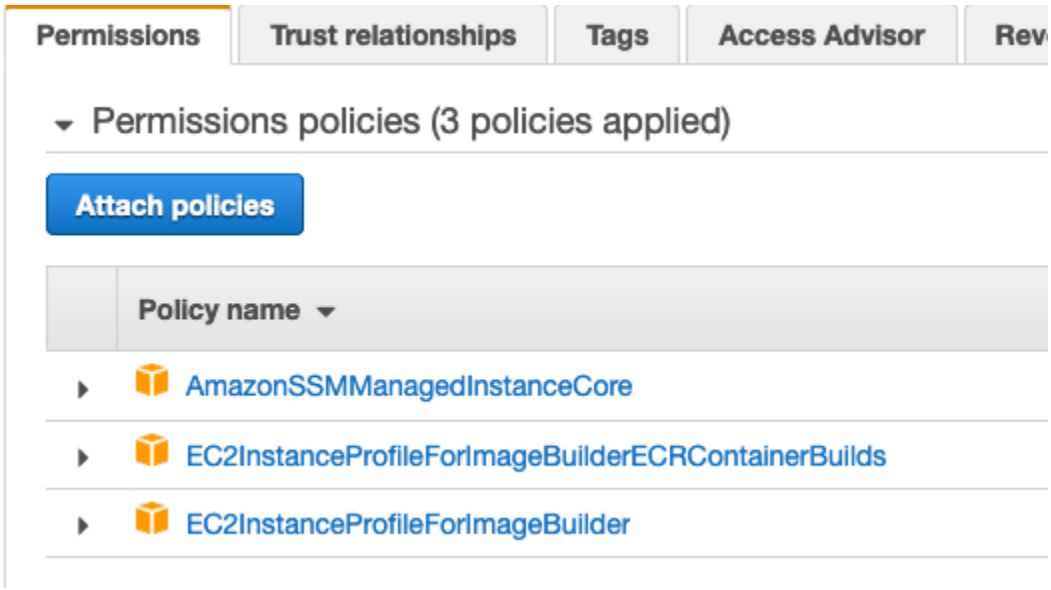
No Amazon EC2 Image Builder, crie um componente para baixar as imagens necessárias e anexá-las à receita da imagem. O exemplo a seguir extrai uma imagem específica de um repositório ECR.

```
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 garantir que o componente a seguir funcione conforme o esperado, verifique se a função do IAM usada pelo EC2 Image builder (EC2InstanceProfileForImageBuilder) tem as políticas anexadas:



Publicação no blog

Na postagem do blog a seguir, você encontrará um passo a passo sobre como implementar a estratégia de armazenamento em cache para o Amazon EKS Windows AMIs personalizado:

Acelerando os tempos de lançamento de contêineres do Windows com o criador de EC2 imagens e a estratégia de cache de imagens

Configurar o gMSA para pods e contêineres do Windows

O que é uma conta gMSA

Aplicativos baseados em Windows, como aplicativos.NET, geralmente usam o Active Directory como provedor de identidade, fornecendo o uso do protocolo NTLM ou authorization/authentication Kerberos.

Um servidor de aplicativos para trocar tíquetes Kerberos com o Active Directory precisa ser associado a um domínio. Os contêineres do Windows não oferecem suporte a uniões de domínio e não fariam muito sentido, pois os contêineres são recursos efêmeros, sobrecarregando o pool RID do Active Directory.

No entanto, os administradores podem aproveitar as contas gMSA do Active Directory para negociar uma autenticação do Windows para recursos como contêineres do Windows, NLB e farms de servidores.

Caso de uso do contêiner Windows e gMSA

Os aplicativos que utilizam a autenticação do Windows e são executados como contêineres do Windows se beneficiam do gMSA porque o Windows Node é usado para trocar o ticket Kerberos em nome do contêiner. Há duas opções disponíveis para configurar o nó de trabalho do Windows para oferecer suporte à integração do gMSA:

1 - Nós de trabalho do Windows unidos ao domínio

Nessa configuração, o nó de trabalho do Windows é associado ao domínio do Active Directory, e a conta AD Computer dos nós de trabalho do Windows é usada para se autenticar no Active Directory e recuperar a identidade gMSA a ser usada com o pod.

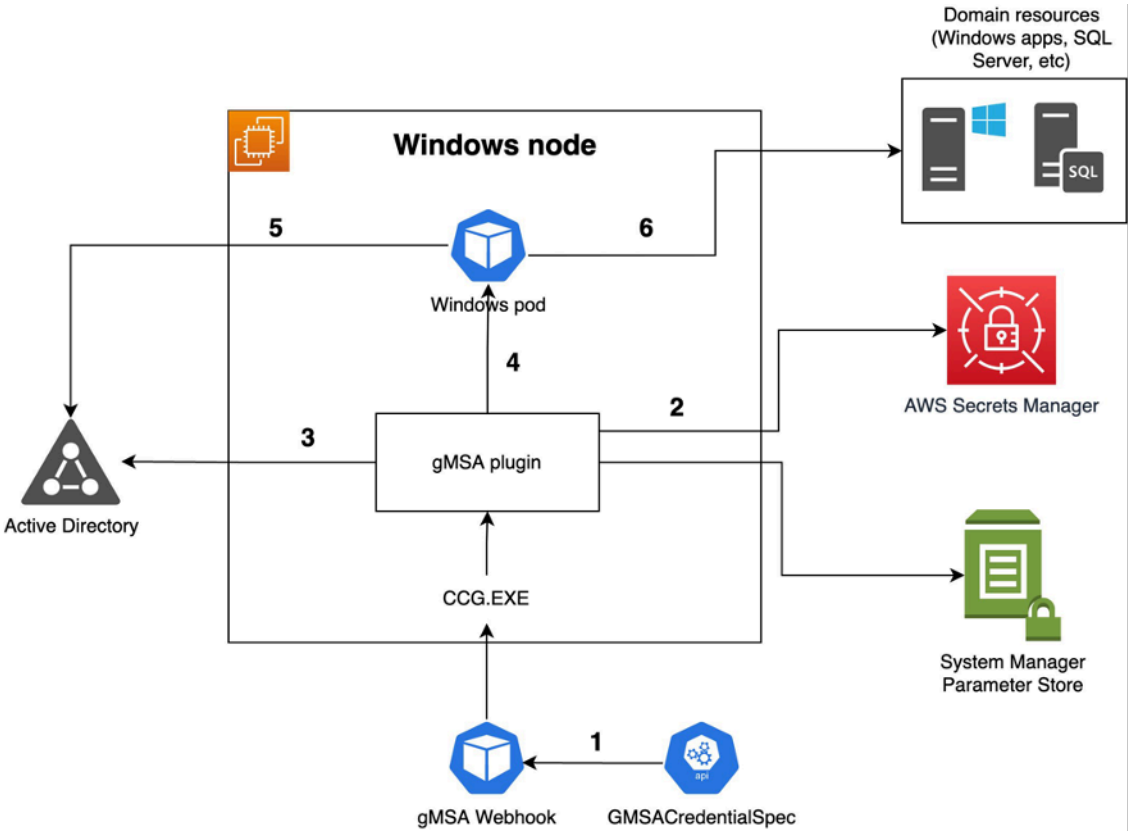
Na abordagem unida por domínio, você pode gerenciar e fortalecer facilmente seus nós de trabalho do Windows usando o Active Directory existente GPOs; no entanto, isso gera sobrecarga operacional e atrasos adicionais durante a união do nó de trabalho do Windows no cluster Kubernetes, pois requer reinicializações adicionais durante a inicialização do nó e a limpeza da garagem do Active Directory após o cluster Kubernetes encerrar os nós.

Na postagem do blog a seguir, você encontrará detalhes step-by-step sobre como implementar a abordagem de nó de trabalho do Windows associado ao domínio:

Autenticação do Windows em pods Windows do Amazon EKS

2 - Nós de trabalho do Windows sem domínio

Nessa configuração, o nó de trabalho do Windows não está associado ao domínio do Active Directory e uma identidade “portátil” (usuário/senha) é usada para se autenticar no Active Directory e recuperar a identidade gMSA a ser usada com o pod.



A identidade portátil é de um usuário do Active Directory; a identidade (usuário/senha) é armazenada no AWS Secrets Manager ou no AWS System Manager Parameter Store, e um plug-in desenvolvido pela AWS chamado ccg_plugin será usado para recuperar essa identidade do AWS Secrets Manager ou do AWS System Manager Parameter Store e passá-la ao containerd para recuperar a identidade gMSA e disponibilizá-la para o pod.

Nessa abordagem sem domínio, você pode se beneficiar de não ter nenhuma interação com o Active Directory durante a inicialização do nó de trabalho do Windows ao usar o gMSA e reduzir a sobrecarga operacional dos administradores do Active Directory.

Na postagem do blog a seguir, você encontrará detalhes step-by-step sobre como implementar a abordagem de nó de trabalho do Windows sem domínio:

Autenticação do Windows sem domínio para pods Windows do Amazon EKS

Observação importante

Apesar de o pod poder usar uma conta gMSA, também é necessário configurar o aplicativo ou serviço adequadamente para oferecer suporte à autenticação do Windows, por exemplo, para

configurar o Microsoft IIS para oferecer suporte à autenticação do Windows, você deve prepará-lo via dockerfile:

```
RUN Install-WindowsFeature -Name Web-Windows-Auth -IncludeAllSubFeature
RUN Import-Module WebAdministration; Set-ItemProperty 'IIS:\AppPools\SiteName' -name
 processModel.identityType -value 2
RUN Import-Module WebAdministration; Set-WebConfigurationProperty -Filter '/
system.webServer/security/authentication/anonymousAuthentication' -Name Enabled -Value
 False -PSPath 'IIS:\' -Location 'SiteName'
RUN Import-Module WebAdministration; Set-WebConfigurationProperty -Filter '/
system.webServer/security/authentication/windowsAuthentication' -Name Enabled -Value
 True -PSPath 'IIS:\' -Location 'SiteName'
```

Fortalecimento dos nós de trabalho do Windows

O fortalecimento do sistema operacional é uma combinação de configuração do sistema operacional, correção e remoção de pacotes de software desnecessários, que visam bloquear um sistema e reduzir a superfície de ataque. É uma prática recomendada preparar sua própria AMI otimizada para Windows para EKS com as configurações de fortalecimento exigidas pela sua empresa.

A AWS fornece uma nova AMI otimizada para Windows para EKS todo mês contendo os patches de segurança mais recentes do Windows Server. No entanto, ainda é responsabilidade do usuário fortalecer sua AMI aplicando as configurações de sistema operacional necessárias, independentemente de usar grupos de nós autogerenciados ou gerenciados.

A Microsoft oferece uma variedade de ferramentas, como o Microsoft Security Compliance Toolkit e o Security Baselines, que ajudam você a obter o fortalecimento com base nas necessidades de suas políticas de segurança. Os benchmarks do CIS também estão disponíveis e devem ser implementados em cima de uma AMI Windows otimizada para Amazon EKS para ambientes de produção.

Reduzindo a superfície de ataque com o Windows Server Core

O Windows Server Core é uma opção de instalação mínima que está disponível como parte da AMI otimizada para Windows para EKS. A implantação do Windows Server Core tem alguns benefícios. Primeiro, ele ocupa um espaço de disco relativamente pequeno, sendo 6 GB no Server Core contra 10 GB no Windows Server com experiência de desktop. Em segundo lugar, ele tem uma superfície de ataque menor por causa de sua base de código menor e disponível APIs.

A AWS fornece aos clientes novos Windows otimizados para Amazon EKS AMIs todos os meses, contendo os patches de segurança mais recentes da Microsoft, independentemente da versão compatível com o Amazon EKS. Como prática recomendada, os nós de trabalho do Windows devem ser substituídos por novos com base na mais recente AMI otimizada para Amazon EKS. Qualquer nó em execução por mais de 45 dias sem uma atualização ou substituição de nó carece das melhores práticas de segurança.

Evitando conexões RDP

O Remote Desktop Protocol (RDP) é um protocolo de conexão desenvolvido pela Microsoft para fornecer aos usuários uma interface gráfica para se conectar a outro computador Windows em uma rede.

Como prática recomendada, você deve tratar seus nós de trabalho do Windows como se fossem hosts efêmeros. Isso significa que não há conexões de gerenciamento, atualizações e solução de problemas. Qualquer modificação e atualização deve ser implementada como uma nova AMI personalizada e substituída pela atualização de um grupo de Auto Scaling. Consulte Aplicação de patches em servidores e contêineres do Windows e gerenciamento otimizado de AMI do Windows pelo Amazon EKS.

Desative as conexões RDP nos nós do Windows durante a implantação passando o valor false na propriedade ssh, conforme o exemplo abaixo:

```
nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
```

Se for necessário acessar o nó do Windows, use o AWS System Manager Session Manager para estabelecer uma PowerShell sessão segura por meio do console da AWS e do agente SSM. Para ver como implementar a solução, assista Acesse com segurança instâncias do Windows usando o AWS Systems Manager Session Manager

Para usar o System Manager Session Manager, uma política adicional do IAM deve ser aplicada à função do IAM usada para iniciar o nó de trabalho do Windows. Abaixo está um exemplo em que a Amazon SSMManaged InstanceCore é especificada no manifesto do `eksctl` cluster:

```
 nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
  iam:
    attachPolicyARNs:
      - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
      - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
      - arn:aws:iam::aws:policy/ElasticLoadBalancingFullAccess
      - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
      - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
```

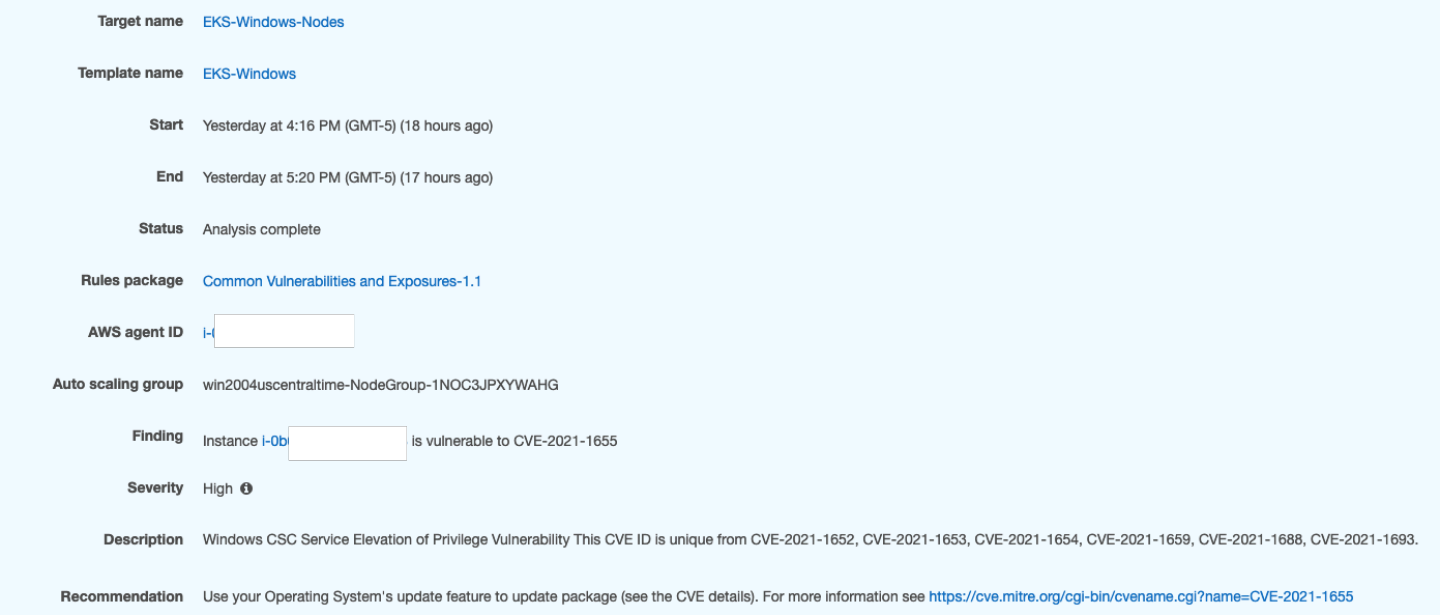
Amazon Inspector

> O Amazon Inspector é um serviço automatizado de avaliação de segurança que ajuda a melhorar a segurança e a conformidade dos aplicativos implantados na AWS. O Amazon Inspector avalia automaticamente os aplicativos quanto à exposição, vulnerabilidades e desvios das melhores práticas. Depois de executar uma avaliação, o Amazon Inspector fornece uma lista detalhada das descobertas de segurança, priorizadas de acordo com seu nível de severidade. Essas descobertas podem ser analisadas diretamente ou como parte de relatórios de avaliação detalhados que estão disponíveis por meio do console ou da API do Amazon Inspector.

O Amazon Inspector pode ser usado para executar a avaliação do CIS Benchmark no nó de trabalho do Windows e pode ser instalado em um Windows Server Core executando as seguintes tarefas:

1. Baixe o seguinte arquivo.exe: https://inspector-agent.amazonaws.com/windows/ installer/latest/ AWSAgentInstall .exe
2. Transfira o agente para o nó de trabalho do Windows.
3. Execute o seguinte comando PowerShell para instalar o agente do Amazon Inspector: `. \AWSAgentInstall.exe /install`

Abaixo está a saída após a primeira execução. Como você pode ver, ele gerou descobertas com base no banco de dados CVE. Você pode usar isso para fortalecer seus nós de trabalho ou criar uma AMI com base nas configurações reforçadas.



Para obter mais informações sobre o Amazon Inspector, incluindo como instalar agentes do Amazon Inspector, configurar a avaliação do CIS Benchmark e gerar relatórios, assista ao vídeo Melhorando a segurança e a conformidade das cargas de trabalho do Windows com o Amazon Inspector.

Amazon GuardDuty

> GuardDutyA Amazon é um serviço de detecção de ameaças que monitora continuamente atividades maliciosas e comportamentos não autorizados para proteger suas contas, cargas de trabalho e dados da AWS armazenados no Amazon S3. Com a nuvem, a coleta e a agregação das atividades da conta e da rede são simplificadas, mas pode ser demorado para as equipes de segurança analisarem continuamente os dados do registro de eventos em busca de possíveis ameaças.

Ao usar a Amazon, GuardDuty você tem visibilidade sobre atividades maliciosas contra nós de trabalho do Windows, como ataques de força bruta RDP e Port Probe.

Assista ao GuardDuty vídeo Detecção de ameaças para cargas de trabalho do Windows usando a Amazon para saber como implementar e executar benchmarks do CIS na AMI otimizada do EKS para Windows

Segurança na Amazon EC2 para Windows

Leia sobre as melhores práticas de segurança para instâncias EC2 do Amazon Windows para implementar controles de segurança em todas as camadas.

Digitalização de imagens de containers

A digitalização de imagens é um recurso automatizado de avaliação de vulnerabilidades que ajuda a melhorar a segurança das imagens de contêiner do seu aplicativo, examinando-as em busca de uma ampla variedade de vulnerabilidades do sistema operacional.

Atualmente, o Amazon Elastic Container Registry (ECR) só pode escanear a imagem do contêiner Linux em busca de vulnerabilidades. No entanto, existem ferramentas de terceiros que podem ser integradas a um CI/CD pipeline existente para digitalização de imagens de contêineres do Windows.

- Âncora
- PaloAlto Nuvem Prisma
- Trend Micro - Verificação inteligente de segurança profunda

Para saber mais sobre como integrar essas soluções com o Amazon Elastic Container Repository (ECR), confira:

- Anchore, digitalizando imagens no Amazon Elastic Container Registry (ECR)
- PaloAlto, digitalizando imagens no Amazon Elastic Container Registry (ECR)
- TrendMicro, digitalizando imagens no Amazon Elastic Container Registry (ECR)

Versão e licença do Windows Server

Versão Windows Server

Uma AMI otimizada para Windows do Amazon EKS é baseada nas edições Windows Server 2019 e 2022 Datacenter no Long-Term Servicing Channel (LTSC). A versão do Datacenter não tem uma limitação no número de contêineres em execução em um nó de trabalho. Para obter mais informações: https://docs.microsoft.com/en-us/ virtualization/windowscontainers/about/faq

Canal de manutenção de longo prazo (LTSC)

Anteriormente chamado de “Long-Term Servicing Branch”, esse é o modelo de lançamento com o qual você já está familiarizado, em que uma nova versão principal do Windows Server é lançada a cada 2-3 anos. Os usuários têm direito a 5 anos de suporte principal e 5 anos de suporte estendido.

Licenciamento

Ao lançar uma EC2 instância da Amazon com uma AMI baseada em Windows Server, a Amazon cobre os custos de licenciamento e a conformidade da licença para você.

Registro em log

Os aplicativos em contêineres geralmente direcionam os registros do aplicativo para o STDOUT. O tempo de execução do contêiner captura esses registros e faz algo com eles. Normalmente, grava em um arquivo. O local onde esses arquivos são armazenados depende do tempo de execução e da configuração do contêiner.

Uma diferença fundamental com os pods do Windows é que eles não geram STDOUT. Você pode executar LogMonitorpara recuperar o ETW (Event Tracing for Windows), os registros de eventos do Windows e outros registros específicos do aplicativo a partir da execução de contêineres do Windows e canalizar a saída de log formatada para STDOUT. Esses registros podem então ser transmitidos usando fluent-bit ou fluentd para o destino desejado, como a Amazon. CloudWatch

O mecanismo de coleta de STDOUT/STDERR registros recupera registros dos pods do Kubernetes. A DaemonSeté uma forma comum de coletar registros de contêineres. Ele oferece a capacidade de gerenciar o log routing/filtering/enrichment independentemente do aplicativo. Um fluentd DaemonSet pode ser usado para transmitir esses registros e quaisquer outros registros gerados pelo aplicativo para um agregador de registros desejado.

Informações mais detalhadas sobre o streaming de logs das cargas de trabalho do Windows para CloudWatch são explicadas aqui.

Recomendações de registro

As melhores práticas gerais de registro não são diferentes ao operar cargas de trabalho do Windows no Kubernetes.

- Sempre registre entradas de registro estruturadas (JSON/SYSLOG), o que facilita o manuseio de entradas de registro, pois há muitos analisadores pré-escritos para esses formatos estruturados.
- Centralize registros - contêineres de registro dedicados podem ser usados especificamente para coletar e encaminhar mensagens de registro de todos os contêineres para um destino
- Mantenha a verbosidade do registro baixa, exceto durante a depuração. A verbosidade coloca muito estresse na infraestrutura de registro e eventos significativos podem ser perdidos no ruído.

- Sempre registre as informações do aplicativo junto com o ID da transação/solicitação para rastreabilidade. Os objetos do Kubernetes não carregam o nome do aplicativo, então, por exemplo, um nome de pod `windows-twryrqyw` pode não ter nenhum significado ao depurar registros. Isso ajuda na rastreabilidade e na solução de problemas de aplicativos com seus registros agregados.

  A forma como você gera esses transaction/correlation IDs depende da construção de programação. Mas um padrão muito comum é usar um Aspect/Interceptor de registro, que pode usar o MDC (contexto de diagnóstico mapeado) para injetar um transaction/correlation ID exclusivo em cada solicitação recebida, da seguinte forma:

```
import org.slf4j.MDC;
import java.util.UUID;
Class LoggingAspect { //interceptor

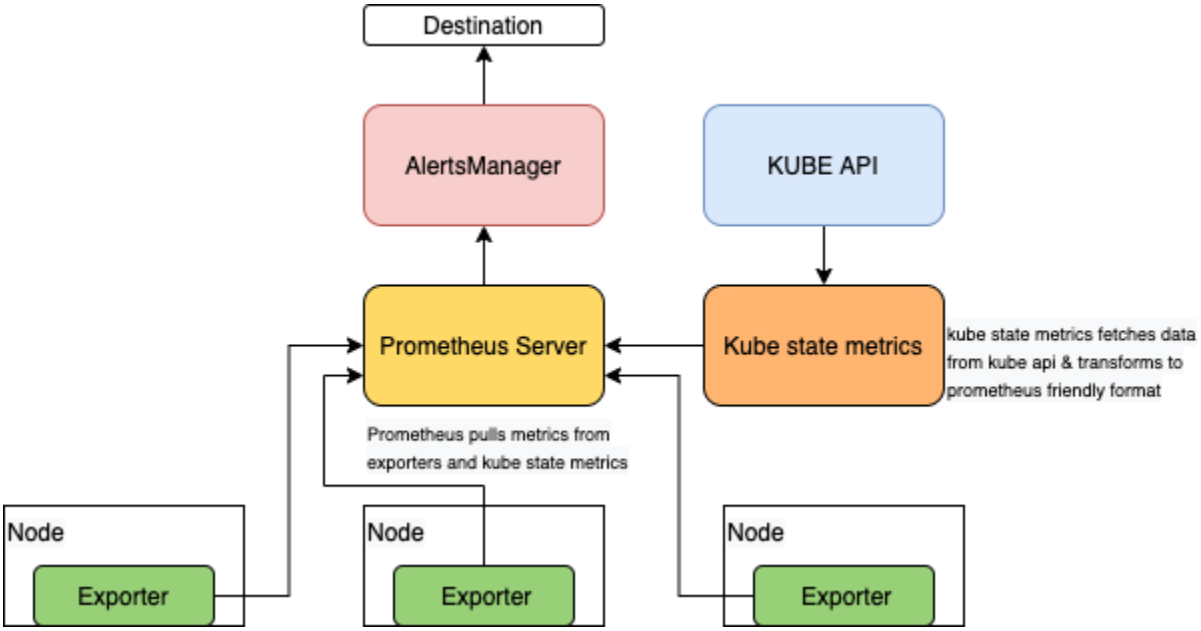
    @Before(value = "execution(* *.*(..))")
    func before(...) {
        transactionId = generateTransactionId();
        MDC.put(CORRELATION_ID, transactionId);
    }

    func generateTransactionId() {
        return UUID.randomUUID().toString();
    }
}
```

Monitoramento

O Prometheus, um projeto graduado da CNCF, é de longe o sistema de monitoramento mais popular com integração nativa ao Kubernetes. O Prometheus coleta métricas em torno de contêineres, pods, nós e clusters. Além disso, o Prometheus AlertsManager aproveita o que permite programar alertas para avisá-lo se algo estiver errado em seu cluster. O Prometheus armazena os dados métricos como dados de série temporal identificados pelo nome e pelos pares da métrica. key/value O Prometheus inclui uma maneira de consultar usando uma linguagem chamada PromQL, abreviação de Prometheus Query Language.

A arquitetura de alto nível da coleção de métricas do Prometheus é mostrada abaixo:



O Prometheus usa um mecanismo de extração e extrai métricas de alvos usando exportadores e da API Kubernetes usando as métricas de estado do kube. Isso significa que aplicativos e serviços devem expor um endpoint HTTP (S) contendo métricas formatadas pelo Prometheus. O Prometheus então, de acordo com sua configuração, extrairá periodicamente métricas desses endpoints HTTP (S).

Um exportador permite que você consuma métricas de terceiros como métricas formatadas pelo Prometheus. Normalmente, um exportador Prometheus é implantado em cada nó. Para obter uma lista completa dos exportadores, consulte os exportadores do Prometheus. Embora o exportador de nós seja adequado para exportar métricas de hardware e sistema operacional do host para nós Linux, ele não funcionará para nós do Windows.

Em um cluster EKS de nós mistos com nós do Windows, ao usar o gráfico de comando estável do Prometheus, você verá pods com falha nos nós do Windows, pois esse exportador não se destina ao Windows. Você precisará tratar o pool de trabalho do Windows separadamente e, em vez disso, instalar o exportador do Windows no grupo de nós de trabalho do Windows.

Para configurar o monitoramento do Prometheus para nós do Windows, você precisa baixar e instalar o exportador WMI no próprio servidor Windows e, em seguida, configurar os destinos dentro da configuração de rascunho do arquivo de configuração do Prometheus. A página de lançamentos fornece todos os instaladores.msi disponíveis, com os respectivos conjuntos de recursos e correções de erros. O instalador configurará o windows_exporter como um serviço do Windows, além de criar uma exceção no firewall do Windows. Se o instalador for executado sem nenhum parâmetro, o exportador será executado com as configurações padrão para coletores, portas, etc.

Você pode conferir a seção de melhores práticas de agendamento deste guia, que sugere o uso taints/tolerations ou RuntimeClass a implantação seletiva do exportador de nós somente em nós Linux, enquanto o exportador do Windows é instalado nos nós do Windows à medida que você inicializa o nó ou usa uma ferramenta de gerenciamento de configuração de sua escolha (por exemplo, chef, Ansible, SSM etc.).

Observe que, diferentemente dos nós Linux em que o exportador de nós é instalado como um daemonset, nos nós do Windows, o exportador WMI é instalado no próprio host. O exportador exportará métricas como o uso da CPU, a memória e o uso do disco I/O e também poderá ser usado para monitorar sites e aplicativos do IIS, interfaces e serviços de rede.

O windows_exporter exibirá todas as métricas dos coletores habilitados por padrão. Essa é a forma recomendada de coletar métricas para evitar erros. No entanto, para uso avançado, o windows_exporter pode receber uma lista opcional de coletores para filtrar métricas. O parâmetro collect [], na configuração do Prometheus, permite que você faça isso.

As etapas de instalação padrão para Windows incluem baixar e iniciar o exportador como um serviço durante o processo de inicialização com argumentos, como os coletores que você deseja 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"
```

Por padrão, as métricas podem ser extraídas no endpoint /metrics na porta 9182. Nesse ponto, o Prometheus pode consumir as métricas adicionando o seguinte scrape_config à configuração do 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', ...]
```

A configuração do Prometheus é recarregada usando

```
> ps aux | grep prometheus
> kill HUP <PID>
```

Uma maneira melhor e recomendada de adicionar alvos é usar uma definição de recurso personalizada chamada ServiceMonitor, que vem como parte do operador Prometheus], que fornece a definição de ServiceMonitor um objeto e um controlador que ativará o que definimos e criará automaticamente ServiceMonitors a configuração necessária do Prometheus.

O ServiceMonitor, que especifica declarativamente como grupos de serviços do Kubernetes devem ser monitorados, é usado para definir um aplicativo do qual você deseja extrair métricas do Kubernetes. Dentro do, ServiceMonitor especificamos os rótulos do Kubernetes que o operador pode usar para identificar o Serviço Kubernetes que, por sua vez, identifica os pods que desejamos monitorar.

Para aproveitar o ServiceMonitor, crie um objeto Endpoint apontando para destinos específicos do Windows, um serviço headless e um ServiceMontor para os nós do Windows.

```
apiVersion: v1
kind: Endpoints
metadata:
  labels:
    k8s-app: wmiexporter
  name: wmiexporter
  namespace: kube-system
subsets:
- addresses:
  - ip: NODE-ONE-IP
    targetRef:
      kind: Node
      name: NODE-ONE-NAME
  - ip: NODE-TWO-IP
    targetRef:
      kind: Node
      name: NODE-TWO-NAME
  - ip: NODE-THREE-IP
    targetRef:
      kind: Node
      name: NODE-THREE-NAME
  ports:
  - name: http-metrics
    port: 9182
```

```
    protocol: TCP

---
apiVersion: v1
kind: Service ##Headless Service
metadata:
  labels:
    k8s-app: wmiexporter
  name: wmiexporter
  namespace: kube-system
spec:
  clusterIP: None
  ports:
  - name: http-metrics
    port: 9182
    protocol: TCP
    targetPort: 9182
  sessionAffinity: None
  type: ClusterIP

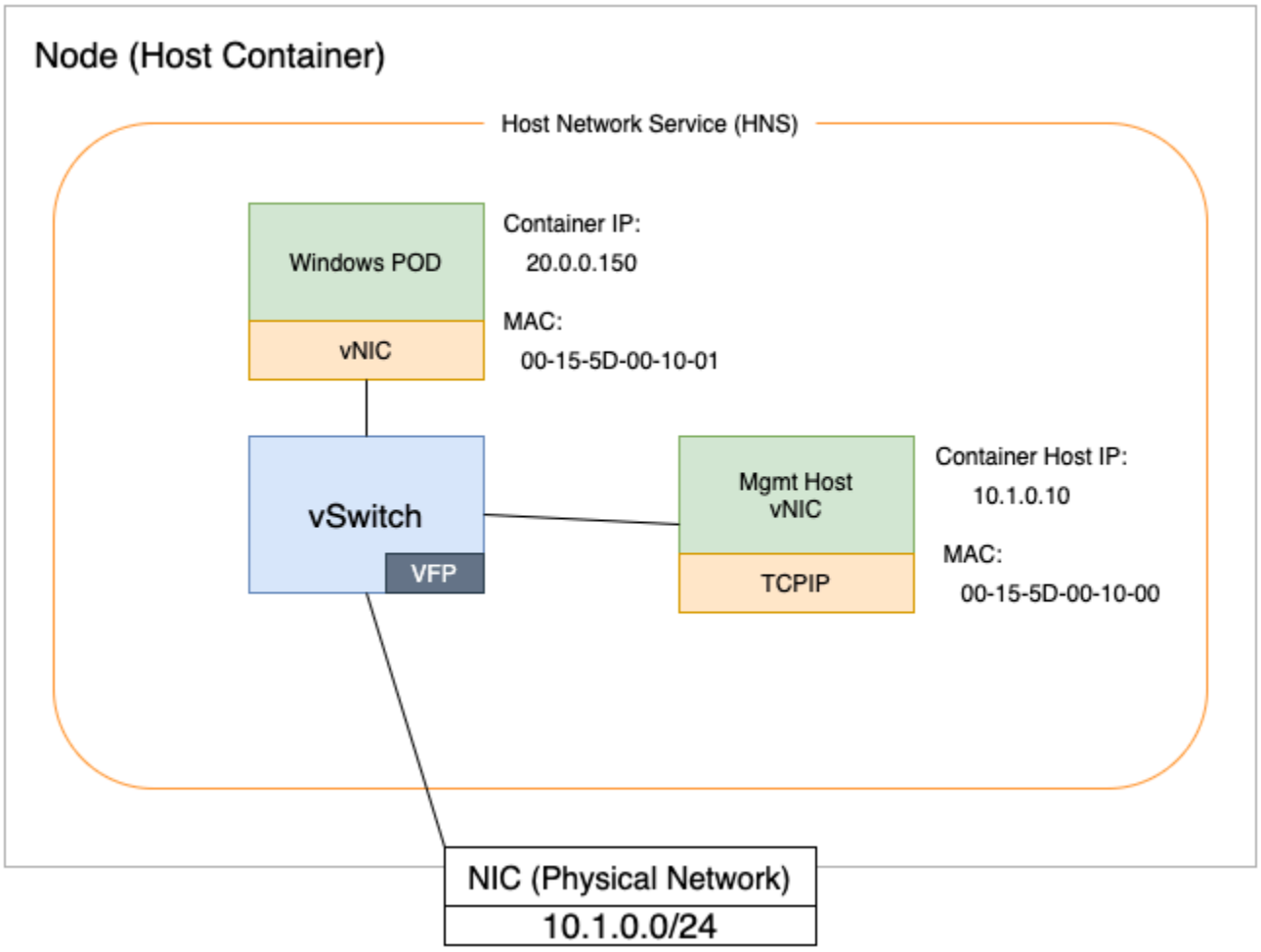
---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor ##Custom ServiceMonitor Object
metadata:
  labels:
    k8s-app: wmiexporter
  name: wmiexporter
  namespace: monitoring
spec:
  endpoints:
  - interval: 30s
    port: http-metrics
  jobLabel: k8s-app
  namespaceSelector:
    matchNames:
    - kube-system
  selector:
    matchLabels:
      k8s-app: wmiexporter
```

Para obter mais detalhes sobre o operador e o uso do ServiceMonitor, confira a documentação oficial do operador. Observe que o Prometheus oferece suporte à descoberta dinâmica de destinos usando muitas opções de descoberta de serviços.

Rede Windows

Visão geral do Windows Container Networking

Os contêineres do Windows são fundamentalmente diferentes dos contêineres do Linux. Os contêineres Linux usam construções Linux, como namespaces, o sistema de arquivos de união e cgroups. No Windows, essas construções são abstraídas dos containerd pelo Host Compute Service (HCS). O HCS atua como uma camada de API que fica acima da implementação do contêiner no Windows. Os contêineres do Windows também utilizam o Host Network Service (HNS), que define a topologia da rede em um nó.



Do ponto de vista da rede, o HCS e o HNS fazem com que os contêineres do Windows funcionem como máquinas virtuais. Por exemplo, cada contêiner tem um adaptador de rede virtual (vNIC) conectado a um switch virtual Hyper-V (vSwitch), conforme mostrado no diagrama acima.

Gerenciamento de endereços IP

Um nó no Amazon EKS usa sua interface de rede elástica (ENI) para se conectar a uma rede VPC da AWS. Atualmente, há suporte para apenas um único ENI por nó de trabalho do Windows. O gerenciamento de endereços IP para nós do Windows é executado pelo VPC Resource Controller, que é executado no plano de controle. Mais detalhes sobre o fluxo de trabalho para gerenciamento de endereços IP dos nós do Windows podem ser encontrados aqui.

O número de pods que um nó de trabalho do Windows pode suportar é determinado pelo tamanho do nó e pelo número de endereços disponíveis IPv4 . Você pode calcular o IPv4 endereço disponível no nó conforme abaixo:

- Por padrão, somente IPv4 endereços secundários são atribuídos à ENI. Nesse caso:

```
Total IPv4 addresses available for Pods = Number of supported IPv4 addresses in the
 primary interface - 1
```

  Subtraímos um da contagem total, pois um IPv4 endereço será usado como endereço principal do ENI e, portanto, não poderá ser alocado aos pods.
- Se o cluster foi configurado para alta densidade de pods ativando o recurso de delegação de prefixo, então-

```
Total IPv4 addresses available for Pods = (Number of supported IPv4 addresses in the
 primary interface - 1) * 16
```

  Aqui, em vez de alocar IPv4 endereços secundários, o VPC Resource Controller alocará `/28` `prefixes` e, portanto, o número total de IPv4 endereços disponíveis será aumentado 16 vezes.

Usando a fórmula acima, podemos calcular o máximo de pods para um operador do Windows com base em uma instância m5.large, conforme abaixo:

- Por padrão, quando executado no modo IP secundário-

```
10 secondary IPv4 addresses per ENI - 1 = 9 available IPv4 addresses
```

- Ao usar `prefix delegation` -

```
(10 secondary IPv4 addresses per ENI - 1) * 16 = 144 available IPv4 addresses
```

Para obter mais informações sobre quantos endereços IP um tipo de instância pode suportar, consulte Endereços IP por interface de rede por tipo de instância.

Outra consideração importante é o fluxo do tráfego da rede. Com o Windows, existe o risco de esgotamento de portas em nós com mais de 100 serviços. Quando essa condição surgir, os nós começarão a gerar erros com a seguinte mensagem:

“Falha na criação da política: o hcnCreateLoad balanceador falhou no Win32: a porta especificada já existe.”

Para resolver esse problema, usamos o Direct Server Return (DSR). O DSR é uma implementação de distribuição de carga de rede assimétrica. Em outras palavras, o tráfego de solicitação e resposta usa caminhos de rede diferentes. Esse recurso acelera a comunicação entre os pods e reduz o risco de esgotamento da porta. Portanto, recomendamos habilitar o DSR nos nós do Windows.

O DSR é habilitado por padrão no Windows Server SAC EKS Optimized. AMIs Para o Windows Server 2019 LTSC EKS Optimized AMIs, você precisará habilitá-lo durante o provisionamento da instância usando o script abaixo e usando o Windows Server 2019 Full ou Core como o AmiFamily no NodeGroup. `eksctl` Consulte a AMI personalizada eksctl para obter informações adicionais.

```
nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
```

Para utilizar o DSR no Windows Server 2019 e versões posteriores, você precisará especificar os seguintes sinalizadores kube-proxy durante a inicialização da instância. Você pode fazer isso ajustando o script de dados do usuário associado ao Launch Template dos grupos de nós autogerenciados.

```
<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>
```

A habilitação do DSR pode ser verificada seguindo as instruções no blog Microsoft Networking e nos contêineres do Windows no AWS Lab.

```
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}
```

Se preservar os IPv4 endereços disponíveis e minimizar o desperdício for crucial para sua sub-rede, geralmente é recomendável evitar o uso do modo de delegação de prefixo, conforme mencionado em Modo de prefixo para Windows - Quando evitar. Se o uso da delegação de prefixo ainda for desejado, você poderá tomar medidas para otimizar a utilização do IPv4 endereço em sua sub-rede. Consulte Configurando parâmetros para delegação de prefixo para obter instruções detalhadas sobre como ajustar a solicitação de IPv4 endereço e o processo de alocação. O ajuste dessas configurações pode ajudá-lo a encontrar um equilíbrio entre a conservação de IPv4 endereços e os benefícios da densidade de pods da delegação de prefixos.

Ao usar a configuração padrão de atribuição de IPv4 endereços secundários, atualmente não há configurações compatíveis para manipular como o VPC Resource Controller solicita e aloca endereços. IPv4 Mais especificamente, `minimum-ip-target` e `warm-ip-target` são compatíveis apenas com o modo de delegação de prefixo. Observe também que, no modo IP secundário, dependendo dos endereços IP disponíveis na interface, o VPC Resource Controller normalmente aloca 3 IPv4 endereços não utilizados no nó em seu nome IPs para mantê-lo aquecido e acelerar a inicialização do pod. Se você quiser minimizar o desperdício de IP de endereços IP quentes não utilizados, você pode programar mais pods em um determinado nó do Windows de forma que você use o máximo possível de capacidade de endereço IP do ENI. Mais explicitamente, você pode evitar que o warm não IPs seja usado se todos os endereços IP na ENI já estiverem sendo usados pelo nó e pelos pods em execução. Outra solução alternativa para ajudá-lo a resolver as restrições com a disponibilidade de endereços IP em suas sub-redes pode ser explorar o aumento do tamanho da sub-rede ou a separação dos nós do Windows em suas próprias sub-redes dedicadas.

Além disso, é importante observar que não IPv6 há suporte nos nós do Windows no momento.

Opções de interface de rede de contêiner (CNI)

O AWSVPC CNI é o plug-in CNI de fato para nós de trabalho do Windows e do Linux. Embora a AWSVPC CNI satisfaça as necessidades de muitos clientes, ainda pode haver momentos em que você precise considerar alternativas, como uma rede de sobreposição, para evitar o esgotamento do IP. Nesses casos, o Calico CNI pode ser usado no lugar do AWSVPC CNI. O Project Calico é um software de código aberto desenvolvido pela Tigera. Esse software inclui uma CNI que funciona com o EKS. As instruções para instalar o Calico CNI no EKS podem ser encontradas na página de instalação do Projeto Calico EKS.

Políticas de rede

É considerado uma prática recomendada mudar do modo padrão de comunicação aberta entre pods em seu cluster Kubernetes para limitar o acesso com base nas políticas de rede. O Project Calico de código aberto tem um forte suporte para políticas de rede que funcionam com nós Linux e Windows. Esse recurso é separado e não depende do uso do Calico CNI. Portanto, recomendamos instalar o Calico e usá-lo para gerenciamento de políticas de rede.

As instruções para instalar o Calico no EKS podem ser encontradas na página Instalando o Calico no Amazon EKS.

Além disso, as recomendações fornecidas no Guia de Melhores Práticas de Segurança do Amazon EKS — Seção Rede se aplicam igualmente aos clusters EKS com nós de trabalho do Windows. No

entanto, alguns recursos como “Grupos de segurança para pods” não são suportados pelo Windows no momento.

Evitando erros de OOM

O Windows não tem um assassino de out-of-memory processos como o Linux. O Windows sempre trata todas as alocações de memória do modo de usuário como virtuais, e os arquivos de paginação são obrigatórios. O efeito final é que o Windows não alcançará as condições de falta de memória da mesma forma que o Linux. Os processos serão enviados para o disco em vez de ficarem sujeitos ao encerramento de falta de memória (OOM). Se a memória estiver superprovisionada e toda a memória física estiver esgotada, a paginação poderá diminuir o desempenho.

Sistema de reserva e memória kubelet

Diferente do Linux, onde `--kubelet-reserve` captura a reserva de recursos para daemons do sistema kubernetes, como kubelet, tempo de execução do contêiner, etc; e `--system-reserve` captura a reserva de recursos para daemons do sistema operacional, como sshd, udev e etc. No Windows, esses sinalizadores não capturam e definem limites de memória no kubelet ou nos processos em execução no nó.

No entanto, você pode combinar esses sinalizadores para conseguir reduzir NodeAllocatablea capacidade no nó com o limite de recursos de memória do manifesto do pod para controlar a alocação de memória por pod. Usando essa estratégia, você tem um melhor controle da alocação de memória, bem como um mecanismo para minimizar out-of-memory (OOM) nos nós do Windows.

Nos nós do Windows, uma prática recomendada é reservar pelo menos 2 GB de memória para o sistema operacional e o processo. Use `--kubelet-reserve` and/or `--system-reserve` para reduzir NodeAllocatable.

Seguindo a documentação dos nós do Windows autogerenciados do Amazon EKS, use o CloudFormation modelo para iniciar um novo grupo de nós do Windows com personalizações na configuração do kubelet. O CloudFormation tem um elemento chamado `BootstrapArguments` que é o mesmo que`KubeletExtraArgs`. Use com as seguintes bandeiras e 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%"
```

Se eksctl for a ferramenta de implantação, verifique a documentação a seguir para personalizar a configuração do kubelet https://eksctl. io/usage/customizing-o-kubelet/

Requisitos de memória de contêiner do Windows

De acordo com a documentação da Microsoft, uma imagem base do Windows Server para NANO requer pelo menos 30 MB, enquanto o Server Core requer 45 MB. Esses números aumentam à medida que você adiciona componentes do Windows, como o.NET Framework, serviços Web como IIS e aplicativos.

É essencial que você saiba a quantidade mínima de memória exigida pela imagem do contêiner do Windows, ou seja, a imagem base mais as camadas do aplicativo, e a defina como a do contêiner resources/requests na especificação do pod. Você também deve definir um limite para evitar que os pods consumam toda a memória de nós disponível no caso de um problema no aplicativo.

No exemplo abaixo, quando o programador do Kubernetes tenta colocar um pod em um nó, as solicitações do pod são usadas para determinar qual nó tem recursos suficientes disponíveis para agendamento.

```
spec:
 - name: iis
   image: mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019
   resources:
     limits:
       cpu: 1
       memory: 800Mi
     requests:
       cpu: .1
       memory: 128Mi
```

Conclusão

O uso dessa abordagem minimiza os riscos de esgotamento da memória, mas não impede que isso aconteça. Usando o Amazon CloudWatch Metrics, você pode configurar alertas e remediações em caso de esgotamento da memória.

Aplicação de patches em servidores e contêineres Windows

A correção do Windows Server é uma tarefa de gerenciamento padrão para administradores do Windows. Isso pode ser feito usando ferramentas diferentes, como Amazon System Manager - Patch

Manager, WSUS, System Center Configuration Manager e muitas outras. No entanto, os nós do Windows em um cluster do Amazon EKS não devem ser tratados como servidores Windows comuns. Eles devem ser tratados como um servidor imutável. Simplificando, evite atualizar um nó existente, basta iniciar um novo com base em uma nova AMI atualizada.

Usando o EC2 Image Builder, você pode automatizar a AMIs construção, criando receitas e adicionando componentes.

O exemplo a seguir mostra componentes, que podem ser componentes preexistentes criados pela AWS (gerenciados pela Amazon), bem como os componentes que você cria (de minha propriedade). Preste muita atenção ao componente gerenciado pela Amazon chamado update-windows, que atualiza o Windows Server antes de gerar a AMI por meio do pipeline do Image Builder EC2 .

Associated components

Search components

1

Component name	Version	Description	Date created	Platform	Owner	ARN
eks-optimized-ami-windows	1.20.0	Installs EKS-optimized Windows artifacts for EKS version 1.20. This includes kubelet version 1.20.4 and Docker version 20.10.6.	Aug 18, 2021 6:33 PM	Windows	Amazon	arn:aws:imagebuilder:us-east-1:aws:component/eks-optimized-ami-windows/1.20.0/1
update-windows	1.0.1	Updates Windows with the latest security updates.	Aug 18, 2021 6:33 PM	Windows	Amazon	arn:aws:imagebuilder:us-east-1:aws:component/update-windows/1.0.1/1
Docker Pull	1.0.1	-	Dec 22, 2021 12:08 PM	Windows	[redacted]	arn:aws:imagebuilder:us-east-[redacted]component/docker-pull/1.0.1/1
simple-boot-test-windows	1.0.0	Executes a simple boot test.	Nov 30, 2019 11:39 PM	Windows	Amazon	arn:aws:imagebuilder:us-east-1:aws:component/simple-boot-test-windows/1.0.0/1

EC2 O Image Builder permite que você crie AMIs com base no Amazon Managed Public AMIs e as personalize para atender às suas necessidades comerciais. Em seguida, você pode AMIs associá-los aos modelos de inicialização, o que permite vincular uma nova AMI ao grupo de Auto Scaling criado pelo EKS Nodegroup. Depois que isso for concluído, você poderá começar a encerrar os nós do Windows existentes e novos serão lançados com base na nova AMI atualizada.

Empurrando e puxando imagens do Windows

A Amazon publica EKS otimizado AMIs que inclui duas imagens de contêiner do Windows em cache.

```
mcr.microsoft.com/windows/servercore
mcr.microsoft.com/windows/nanoserver
```

```
REPOSITORY                                                      TAG                      IMAGE ID       CREATED          SIZE
mcr.microsoft.com/windows/servercore                            ltsc2019                 617dd5ead5b8   6 weeks ago      5.74GB
mcr.microsoft.com/windows/nanoserver                            1809                     8334af2fa7ba   6 weeks ago      257MB
[redacted].dkr.ecr.us-east-1.amazonaws.com/fluentd-windows-coreltsc   latest             721afca2c725   7 weeks ago      6.96GB
fluent/fluentd                                                  v1.14-windows-ltsc2019-1 721afca2c725   7 weeks ago      6.96GB
amazonaws.com/eks/pause-windows                                 latest                   6392f69ae6e7   10 months ago    255MB
```

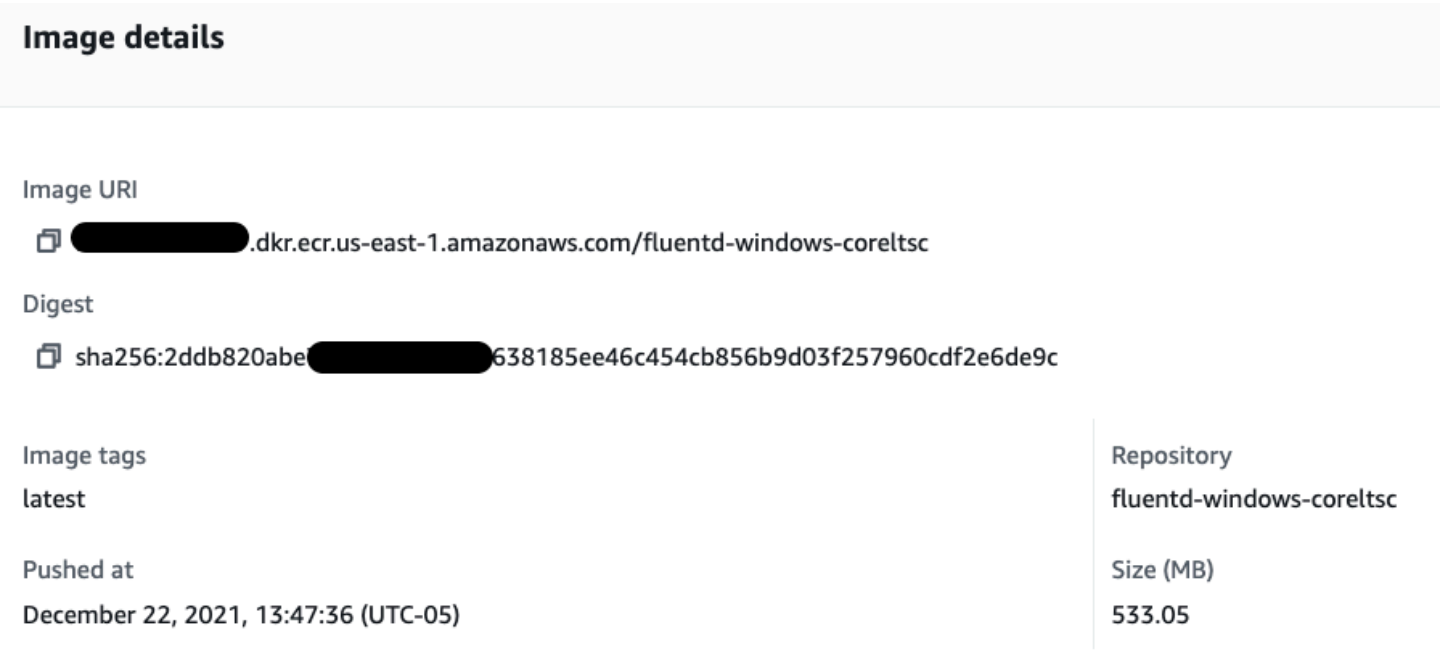
As imagens em cache são atualizadas após as atualizações no sistema operacional principal. Quando a Microsoft lançar uma nova atualização do Windows que afeta diretamente a imagem base

do contêiner do Windows, a atualização será lançada como uma atualização comum do Windows no sistema operacional principal. Manter o ambiente up-to-date oferece um ambiente mais seguro no nível do Node e do Container.

O tamanho de uma imagem de contêiner do Windows influencia push/pull as operações que podem levar a tempos de inicialização lentos do contêiner. O armazenamento em cache de imagens de contêiner do Windows permite que I/O operações caras (extração de arquivos) ocorram na criação da compilação da AMI, em vez de na inicialização do contêiner. Como resultado, todas as camadas de imagem necessárias serão extraídas na AMI e estarão prontas para serem usadas, acelerando o tempo em que um contêiner do Windows é iniciado e pode começar a aceitar tráfego. Durante uma operação push, somente as camadas que compõem sua imagem são carregadas no repositório.

O exemplo a seguir mostra que, no Amazon ECR, as imagens de fluentd-windows-sac2004 têm apenas 390,18 MB. Essa é a quantidade de upload que aconteceu durante a operação push.

O exemplo a seguir mostra uma imagem ltsc fluente do Windows enviada para um repositório Amazon ECR. O tamanho da camada armazenada no ECR é 533,05 MB.



A saída abaixo de`docker image ls`, o tamanho do fluentd v1.14-windows-ltsc2019-1 é de 6,96 GB no disco, mas isso não significa que ele baixou e extraiu essa quantidade de dados.

Na prática, durante a operação de extração, somente os 533,05 MB compactados serão baixados e extraídos.

```
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
```

A coluna de tamanho mostra o tamanho geral da imagem, 6,96 GB. Detalhando:

- Imagem básica do Windows Server Core 2019 LTSC = 5,74 GB
- Imagem base fluente não comprimida = 6,96 GB
- Diferença no disco = 1,2 GB
- ECR de imagem final comprimida fluente = 533,05 MB

A imagem base já existe no disco local, resultando na quantidade total em disco de 1,2 GB a mais. Na próxima vez que você ver a quantidade GBs na coluna de tamanho, não se preocupe muito, provavelmente mais de 70% já estão no disco como uma imagem de contêiner em cache.

Referência

Acelerando os tempos de lançamento de contêineres do Windows com o criador de EC2 imagens e a estratégia de cache de imagens

Executando cargas de trabalho heterogêneas

O Kubernetes tem suporte para clusters heterogêneos, nos quais você pode ter uma mistura de nós Linux e Windows no mesmo cluster. Dentro desse cluster, você pode ter uma mistura de pods que rodam no Linux e pods que rodam no Windows. Você pode até mesmo executar várias versões do Windows no mesmo cluster. No entanto, há vários fatores (conforme mencionado abaixo) que precisarão ser considerados ao tomar essa decisão.

Atribuição PODs às melhores práticas de atribuição aos nós

Para manter as cargas de trabalho do Linux e do Windows em seus respectivos nós específicos do sistema operacional, você precisa usar alguma combinação de seletores de nós e contaminações/

tolerâncias. O principal objetivo de programar cargas de trabalho em um ambiente heterogêneo é evitar a quebra da compatibilidade das cargas de trabalho Linux existentes.

Garantindo que cargas de trabalho específicas do sistema operacional cheguem ao host de contêiner apropriado

Os usuários podem garantir que os contêineres do Windows possam ser programados no host apropriado usando nodeSelectors. Atualmente, todos os nós do Kubernetes têm os seguintes rótulos padrão:

```
kubernetes.io/os = [windows|linux]
kubernetes.io/arch = [amd64|arm64|...]
```

Se uma especificação do Pod não incluir um NodeSelector`"kubernetes.io/os": windows`, o Pod poderá ser programado em qualquer host, Windows ou Linux. Isso pode ser problemático, pois um contêiner do Windows só pode ser executado no Windows e um contêiner do Linux só pode ser executado no Linux.

Em ambientes corporativos, não é incomum ter um grande número de implantações pré-existentes para contêineres Linux, bem como um ecossistema de off-the-shelf configurações, como os gráficos do Helm. Nessas situações, você pode hesitar em fazer alterações nos nodeSelectors de uma implantação. A alternativa é usar Taints.

Por exemplo: `--register-with-taints='os=windows:NoSchedule'`

Se você estiver usando o EKS, o eksctl oferece maneiras de aplicar manchas por meio do ClusterConfig:

```
NodeGroups:
  - name: windows-ng
    amiFamily: WindowsServer2022FullContainer
    ...
    labels:
      nodeclass: windows2022
    taints:
      os: "windows:NoSchedule"
```

Adicionando uma contaminação a todos os nós do Windows, o agendador não agendará pods nesses nós, a menos que eles tolerem a contaminação. Exemplo de manifesto de pod:

```
nodeSelector:
    kubernetes.io/os: windows
tolerations:
    - key: "os"
      operator: "Equal"
      value: "windows"
      effect: "NoSchedule"
```

Manipulando várias compilações do Windows no mesmo cluster

A imagem base do contêiner do Windows usada por cada pod deve corresponder à mesma versão de compilação do kernel do nó. Se você quiser usar várias compilações do Windows Server no mesmo cluster, defina rótulos de nós adicionais, nodeSelectors ou utilize um rótulo chamado windows-build.

O Kubernetes 1.17 adiciona automaticamente um novo rótulo node.kubernetes.io/windows-build para simplificar o gerenciamento de várias compilações do Windows no mesmo cluster. Se você estiver executando uma versão mais antiga, é recomendável adicionar esse rótulo manualmente aos nós do Windows.

Esse rótulo reflete o número principal, secundário e de compilação do Windows que precisam corresponder para fins de compatibilidade. Abaixo estão os valores usados atualmente para cada versão do Windows Server.

É importante observar que o Windows Server está migrando para o Long-Term Servicing Channel (LTSC) como o principal canal de lançamento. O Windows Server Semi-Annual Channel (SAC) foi descontinuado em 9 de agosto de 2022. Não haverá versões futuras do SAC do Windows Server.

Nome do produto	Número (s) de compilação
Servidor LTSC 2022 completo	10.0.20348
Núcleo do servidor 2019 LTSC	10.0.17763

É possível verificar a versão de compilação do sistema operacional por meio do seguinte comando:

```
kubectl get nodes -o wide
```

A saída KERNEL-VERSION corresponde à versão de compilação do sistema operacional 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
```

O exemplo abaixo aplica um NodeSelector adicional ao manifesto do pod para corresponder à versão correta do Windows build ao executar diferentes versões do sistema operacional de grupos de nós do Windows.

```
nodeSelector:
    kubernetes.io/os: windows
    node.kubernetes.io/windows-build: '10.0.20348'
tolerations:
    - key: "os"
    operator: "Equal"
    value: "windows"
    effect: "NoSchedule"
```

Simplificação NodeSelector e tolerância em manifestos de Pod usando RuntimeClass

Você também pode usá-lo RuntimeClass para simplificar o processo de uso de manchas e tolerâncias. Isso pode ser feito criando um RuntimeClass objeto que é usado para encapsular essas manchas e tolerâncias.

Crie um RuntimeClass executando o seguinte manifesto:

```
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"
```

Depois que a classe Runtime for criada, atribua-a usando como especificação no manifesto do 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
```

Managed Node Group Support

Para ajudar os clientes a executar seus aplicativos Windows de forma mais simplificada, a AWS lançou o suporte para o suporte do Amazon EKS Managed Node Group (MNG) para contêineres Windows em 15 de dezembro de 2022. Para ajudar a alinhar as equipes de operações, o Windows MNGs é habilitado usando os mesmos fluxos de trabalho e ferramentas do Linux. MNGs Há suporte para as versões completas e principais da família AMI (Amazon Machine Image) do Windows Server 2019 e 2022.

As seguintes famílias de AMI são compatíveis com grupos de nós gerenciados (MNG).

Família AMI
Windows_Core_2019_x86_64
Windows_Full_2019_x86_64
Windows_Core_2022_x86_64
Windows_Full_2022_x86_64

Documentações adicionais

Documentação oficial da AWS: https://docs.aws.amazon.com/eks/ latest/userguide/windows - support.html

Para entender melhor como o Pod Networking (CNI) funciona, confira o seguinte link: - networking.html https://docs.aws.amazon.com/eks/ latest/userguide/pod

Blog da AWS sobre a implantação do Managed Node Group para Windows no EKS: https:// aws.amazon.com/blogs/ containers/ deploying-amazon-eks-windows -/managed-node-groups

Contextos de segurança do pod

As Políticas de Segurança do Pod (PSP) e os Padrões de Segurança do Pod (PSS) são duas formas principais de reforçar a segurança no Kubernetes. Observe que PodSecurityPolicy está obsoleto a partir do Kubernetes v1.21 e será removido na v1.25. O Pod Security Standard (PSS) é a abordagem recomendada pelo Kubernetes para reforçar a segurança no futuro.

A Política de Segurança do Pod (PSP) é uma solução nativa no Kubernetes para implementar políticas de segurança. O PSP é um recurso em nível de cluster que controla aspectos sensíveis à segurança da especificação do Pod. Usando a Política de Segurança do Pod, você pode definir um conjunto de condições que os pods devem atender para serem aceitos pelo cluster. O recurso PSP está disponível desde os primórdios do Kubernetes e foi projetado para impedir que pods mal configurados sejam criados em um determinado cluster.

Para obter mais informações sobre as políticas de segurança do Pod, consulte a documentação do Kubernetes. De acordo com a política de descontinuação do Kubernetes, as versões mais antigas deixarão de receber suporte nove meses após a descontinuação do recurso.

Por outro lado, os Padrões de Segurança do Pod (PSS), que são a abordagem de segurança recomendada e normalmente implementados usando contextos de segurança, são definidos como parte das especificações do pod e do contêiner no manifesto do pod. O PSS é o padrão oficial que a equipe do projeto Kubernetes definiu para abordar as melhores práticas relacionadas à segurança para pods. Ele define políticas como linha de base (minimamente restritiva, padrão), privilegiada (irrestritiva) e restrita (mais restritiva).

Recomendamos começar com o perfil básico. O perfil básico do PSS fornece um equilíbrio sólido entre segurança e potencial atrito, exigindo uma lista mínima de exceções e serve como um bom ponto de partida para a segurança da carga de trabalho. Se você estiver usando atualmente, PSPs recomendamos mudar para o PSS. Mais detalhes sobre as políticas do PSS podem ser encontrados na documentação do Kubernetes. Essas políticas podem ser aplicadas com várias ferramentas, incluindo as da OPA e da Kyverno. Por exemplo, o Kyverno fornece a coleção completa de políticas de PSS aqui.

As configurações de contexto de segurança permitem conceder privilégios para selecionar processos, usar perfis de programas para restringir recursos a programas individuais, permitir escalonamento de privilégios, filtrar chamadas do sistema, entre outras coisas.

Os pods do Windows no Kubernetes têm algumas limitações e diferenciais das cargas de trabalho padrão baseadas em Linux quando se trata de contextos de segurança.

O Windows usa um objeto Job por contêiner com um filtro de namespace do sistema para conter todos os processos em um contêiner e fornecer isolamento lógico do host. Não há como executar um contêiner do Windows sem a filtragem de namespace em vigor. Isso significa que os privilégios do sistema não podem ser declarados no contexto do host e, portanto, os contêineres privilegiados não estão disponíveis no Windows.

A seguir `windowsOptions` estão as únicas opções documentadas do Contexto de Segurança do Windows, enquanto as demais são opções gerais do Contexto de Segurança.

Para obter uma lista dos atributos de contexto de segurança que são suportados no Windows versus Linux, consulte a documentação oficial aqui.

As configurações específicas do pod são aplicadas a todos os contêineres. Se não for especificado, as opções do PodSecurityContext serão usadas. Se definido em SecurityContext e PodSecurityContext, o valor especificado em SecurityContext tem precedência.

Por exemplo, a configuração de runAsUser nome para pods e contêineres, que é uma opção do Windows, é um equivalente aproximado da runAsUser configuração específica do Linux e, no manifesto a seguir, o contexto de segurança específico do pod é aplicado a todos os contêineres.

```
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
```

Já a seguir, o contexto de segurança em nível de contêiner substitui o contexto de segurança em nível 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
```

Exemplos de valores aceitáveis para o campo runAsUser Nome: ContainerAdministrator, ContainerUser, NT AUTHORITY\ NETWORK SERVICE, NT AUTHORITY\ LOCAL SERVICE

Geralmente, é uma boa ideia executar seus contêineres com ContainerUser pods do Windows. Os usuários não são compartilhados entre o contêiner e ContainerAdministrator o host, mas têm privilégios adicionais dentro do contêiner. Observe que há limitações de nome de usuário que você deve conhecer.

Um bom exemplo de quando usar ContainerAdministrator é definir PATH. Você pode usar a diretiva USER para fazer isso, assim:

```
USER ContainerAdministrator
RUN setx /M PATH "%PATH%;C:/your/path"
USER ContainerUser
```

Observe também que os segredos são escritos em texto não criptografado no volume do nó (em comparação com tmpfs/in-memory no Linux). Isso significa que você precisa fazer duas coisas

- Use o arquivo ACLs para proteger a localização do arquivo secreto
- Use criptografia em nível de volume usando BitLocker

Opções de armazenamento persistente

O que é um plug-in em árvore versus um plug-in out-of-tree de volume?

Antes da introdução da Container Storage Interface (CSI), todos os plug-ins de volume estavam em árvore, o que significa que eram criados, vinculados, compilados e enviados com os principais binários do Kubernetes e ampliavam a API principal do Kubernetes. Isso significava que adicionar um novo sistema de armazenamento ao Kubernetes (um plug-in de volume) exigia a verificação do código no repositório de código principal do Kubernetes.

Out-of-tree os plug-ins de volume são desenvolvidos independentemente da base de código do Kubernetes e são implantados (instalados) nos clusters do Kubernetes como extensões. Isso dá aos fornecedores a capacidade de atualizar os drivers out-of-band, ou seja, separadamente do ciclo de lançamento do Kubernetes. Isso é amplamente possível porque o Kubernetes criou uma interface de armazenamento ou CSI que fornece aos fornecedores uma maneira padrão de interagir com o k8s.

Você pode conferir mais sobre as classes de armazenamento do Amazon Elastic Kubernetes Services (EKS) e os drivers CSI em .html https://docs.aws.amazon.com/eks/ latest/userguide/storage

Plug-in de volume em árvore para Windows

Os volumes do Kubernetes permitem que aplicativos, com requisitos de persistência de dados, sejam implantados no Kubernetes. O gerenciamento de volumes persistentes consiste em contêineres provisioning/de-provisioning/resizing of volumes, attaching/detaching a volume to/from a Kubernetes node, and mounting/dismounting a volume to/from individuais em um pod. O código para implementar essas ações de gerenciamento de volume para um back-end ou protocolo de armazenamento específico é enviado na forma de um plug-in de volume do Kubernetes (plug-ins de volume em árvore). No Amazon Elastic Kubernetes Services (EKS), a seguinte classe de plug-ins de volume do Kubernetes é compatível com o Windows:

Plug-in de volume na árvore: Store awsElasticBlock

Para usar o plug-in de volume In-tree nos nós do Windows, é necessário criar um adicional StorageClass para usar o NTFS como o FSType. No EKS, o padrão StorageClass usa ext4 como o FSType padrão.

StorageClass A fornece uma forma de os administradores descreverem as “classes” de armazenamento que oferecem. Classes diferentes podem ser mapeadas para quality-of-service níveis, políticas de backup ou políticas arbitrárias determinadas pelos administradores do cluster. O Kubernetes não tem opinião sobre o que as classes representam. Esse conceito às vezes é chamado de “perfis” em outros sistemas de armazenamento.

Você pode verificar isso executando o seguinte comando:

```
kubectl describe storageclass gp2
```

Saída:

```
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 criar o novo StorageClass para oferecer suporte ao NTFS, use o seguinte manifesto:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gp2-windows
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  fsType: ntfs
volumeBindingMode: WaitForFirstConsumer
```

Crie o StorageClass executando o seguinte comando:

```
kubectl apply -f NTFSStorageClass.yaml
```

A próxima etapa é criar uma Declaração de Volume Persistente (PVC).

A PersistentVolume (PV) é uma parte do armazenamento no cluster que foi provisionada por um administrador ou provisionada dinamicamente usando PVC. É um recurso no cluster, assim como um nó é um recurso do cluster. Esse objeto de API captura os detalhes da implementação do armazenamento, seja NFS, iSCSI ou um sistema de armazenamento. cloud-provider-specific

A PersistentVolumeClaim (PVC) é uma solicitação de armazenamento feita por um usuário. As solicitações podem solicitar tamanhos e modos de acesso específicos (por exemplo, elas podem ser montadas ReadWriteOnce ReadOnlyMany ou ReadWriteMany).

Os usuários precisam PersistentVolumes de atributos diferentes, como desempenho, para diferentes casos de uso. Os administradores de cluster precisam ser capazes de oferecer uma variedade PersistentVolumes que difere em mais formas do que apenas tamanho e modos de acesso, sem expor os usuários aos detalhes de como esses volumes são implementados. Para essas necessidades, existe o StorageClass recurso.

No exemplo abaixo, o PVC foi criado nas janelas do namespace.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ebs-windows-pv-claim
  namespace: windows
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gp2-windows
  resources:
    requests:
      storage: 1Gi
```

Crie o PVC executando o seguinte comando:

```
kubectl apply -f persistent-volume-claim.yaml
```

O manifesto a seguir cria um Windows Pod, configura o VolumeMount as `C:\Data` e usa o PVC como armazenamento conectado`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'
```

Teste os resultados acessando o pod do Windows por meio de PowerShell:

```
kubectl exec -it podname powershell -n windows
```

Dentro do Windows Pod, execute: `ls`

Saída:

```
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
```

O diretório de dados é fornecido pelo volume do EBS.

Out-of-tree para Windows

O código associado aos plug-ins CSI é fornecido como out-of-tree scripts e binários que normalmente são distribuídos como imagens de contêiner e implantados usando construções padrão do Kubernetes, como e. DaemonSets StatefulSets Os plug-ins CSI lidam com uma ampla variedade de ações de gerenciamento de volume no Kubernetes. Os plug-ins CSI geralmente consistem em plug-ins de nó (que são executados em cada nó como um DaemonSet) e plug-ins de controlador.

Os plug-ins de nós CSI (especialmente aqueles associados a volumes persistentes expostos como dispositivos de bloco ou em um sistema de arquivos compartilhado) precisam realizar várias operações privilegiadas, como varredura de dispositivos de disco, montagem de sistemas de arquivos etc. Essas operações são diferentes para cada sistema operacional host. Para nós de trabalho Linux, os plug-ins de nós CSI em contêineres são normalmente implantados como contêineres privilegiados. Para nós de trabalho do Windows, as operações privilegiadas para plug-ins de nós CSI em contêineres são suportadas usando o csi-proxy, um binário autônomo e gerenciado pela comunidade que precisa ser pré-instalado em cada nó do Windows.

A AMI otimizada para Windows do Amazon EKS inclui proxy CSI a partir de abril de 2022. Os clientes podem usar o driver SMB CSI nos nós do Windows para acessar o Amazon for Windows File Server, o Amazon FSx FSx for NetApp ONTAP SMB Shares, o and/or AWS Storage Gateway — File Gateway.

O blog a seguir tem detalhes de implementação sobre como configurar o SMB CSI Driver para usar o Amazon FSx for Windows File Server como um armazenamento persistente para Windows Pods.

Servidor FSx de arquivos Amazon para Windows

Uma opção é usar o Amazon FSx for Windows File Server por meio de um recurso SMB chamado SMB Global Mapping, que possibilita montar um compartilhamento SMB no host e, em seguida, passar os diretórios desse compartilhamento para um contêiner. O contêiner não precisa ser configurado com um servidor, compartilhamento, nome de usuário ou senha específicos. Em vez disso, tudo isso é tratado no host. O contêiner funcionará da mesma forma como se tivesse armazenamento local.

> O mapeamento global de pequenas e médias empresas é transparente para o orquestrador e é montado por meio do HostPath qual pode implicar preocupações seguras.

No exemplo abaixo, o caminho `G:\Directory\app-state` é um compartilhamento SMB no Windows Node.

```
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
```

O blog a seguir tem detalhes de implementação sobre como configurar o Amazon FSx para Windows File Server como um armazenamento persistente para Windows Pods.

Fortalecimento de imagens de contêineres do Windows

Você está fortalecendo suas imagens de contêiner do Windows? Ao longo dos anos, trabalhei com clientes em todo o mundo para ajudá-los a migrar cargas de trabalho legadas para contêineres, especialmente cargas de trabalho do Windows. Com mais de 20 anos de experiência, vi organizações dedicarem esforços e recursos substanciais para fortalecer seus servidores Windows, implementando tudo, desde benchmarks do CIS até proteção antivírus em tempo de execução para proteger dados confidenciais.

No entanto, surgiu uma tendência preocupante. À medida que essas máquinas virtuais altamente seguras são modernizadas em contêineres, muitas práticas críticas de fortalecimento estão sendo negligenciadas. As melhores práticas de segurança do Windows, da imagem base (SO) aos serviços da Web, como o IIS, geralmente são negligenciadas, com a maior parte do foco colocada exclusivamente na proteção do host do contêiner. É vital reconhecer que, embora os contêineres operem em namespaces isolados, eles ainda compartilham as primitivas do kernel com o host. Normalmente, os atacantes estão mais interessados no movimento lateral do que em atacar diretamente o hospedeiro do contêiner, permitindo que explorem configurações fracas de segurança do contêiner e acessem dados confidenciais.

O objetivo das documentações é destacar algumas configurações de segurança essenciais que você deve implementar especificamente para contêineres do Windows que hospedam `ASP.NET` sites no IIS. Vamos nos concentrar em quatro áreas principais:

- Políticas de segurança da conta
- Políticas de auditoria
- Melhores práticas de segurança do IIS
- Princípio do privilégio mínimo

Começaremos investigando por que cada uma dessas configurações de segurança é vital para proteger seus contêineres do Windows, examinando os riscos específicos que elas mitigam e os benefícios de segurança que oferecem. A seguir, examinaremos um trecho de código que demonstra como implementar essas configurações corretamente em seu Dockerfile, garantindo que seu contêiner seja protegido contra possíveis ameaças. Por fim, detalharemos cada configuração, oferecendo uma explicação abrangente de sua função, impacto na segurança de contêineres e como ela contribui para proteger seus aplicativos. Essa abordagem não apenas mostrará como aplicar essas melhores práticas, mas também fornecerá a visão para entender por que elas são essenciais para manter uma postura de segurança robusta em ambientes em contêineres.

1. Configurar políticas de conta (senha ou bloqueio) usando políticas de segurança e registro locais

O Windows Server Core é uma opção de instalação mínima que está disponível como parte da [AMI otimizada para Windows para EKS] (https://docs.aws.amazon.com/eks/latest/userguide/eks- optimized-windows-ami .html). A configuração de políticas de conta (senha ou bloqueio) usando políticas de segurança locais e o registro fortalece a segurança do sistema ao impor regras

robustas de senha e bloqueio. Essas políticas exigem que os usuários criem senhas fortes com um comprimento e complexidade mínimos definidos, protegendo contra ataques comuns relacionados a senhas.

Ao definir uma idade máxima para a senha, os usuários são solicitados a atualizá-las regularmente, reduzindo a probabilidade de comprometimento das credenciais. As políticas de bloqueio adicionam uma camada extra de proteção ao bloquear temporariamente as contas após um número específico de tentativas malsucedidas de login, ajudando a evitar ataques de força bruta. Definir essas configurações por meio do Registro do Windows permite que os administradores apliquem essas medidas de segurança no nível do sistema, garantindo uniformidade e conformidade em toda a organização. A aplicação dessas políticas de conta em um contêiner do Windows é essencial para manter a consistência da segurança, embora os contêineres geralmente sejam efêmeros e se destinem a cargas de trabalho isoladas:

Consistência de segurança

- Conformidade: aplicar políticas de senha e regras de bloqueio consistentes em contêineres ajuda a manter a conformidade de segurança, especialmente em ambientes que exigem controles de acesso rígidos (por exemplo, conformidade regulatória, como HIPAA, PCI-DSS).
- Contêineres reforçados: a aplicação dessas configurações garante que seu contêiner do Windows seja protegido contra acesso não autorizado ou ataques baseados em senha, alinhando a postura de segurança do seu contêiner às políticas de segurança mais amplas do sistema.

Proteção contra ataques de força bruta

- Bloqueio de conta: essas configurações ajudam a se defender contra tentativas de login por força bruta bloqueando contas após um número específico de tentativas de login malsucedidas. Isso evita que os invasores tentem um número ilimitado de senhas.
- Complexidade da senha: exigir senhas complexas com tamanho suficiente reduz a probabilidade de senhas fracas serem exploradas, mesmo em ambientes isolados em contêineres.

Cenários de vários usuários

- Se seu aplicativo em contêineres foi projetado para lidar com vários usuários ou exigir autenticação de usuário, a aplicação de políticas de senha garante que as contas de usuário dentro do contêiner sigam regras de segurança rígidas, limitando o acesso somente a usuários autorizados.

Contêineres Windows persistentes

- Embora os contêineres sejam geralmente considerados efêmeros, alguns contêineres do Windows podem executar serviços de longo prazo ou lidar com o gerenciamento de usuários, o que torna importante aplicar políticas de segurança adequadas, semelhantes às de um servidor Windows normal.

Consistência em ambientes híbridos

- Se você estiver executando máquinas virtuais e contêineres em sua infraestrutura, a aplicação das mesmas políticas de segurança (por exemplo, password/lockout políticas) em todos os ambientes garante padrões de segurança uniformes, simplificando a governança e o gerenciamento.

Em resumo, a aplicação dessas políticas de conta em contêineres do Windows garante que seus contêineres não sejam um ponto fraco em sua estratégia de segurança, protegendo contra ataques de senha e reforçando a consistência em todo o ambiente.

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
```

Explicação:

Esta seção define políticas de conta para configurações de senha e bloqueio por meio do Registro do Windows. Essas políticas ajudam a reforçar a segurança controlando os requisitos de senha e os limites de bloqueio da conta.

1. MinimumPasswordLength (MINPWLEN) = 14 Essa configuração define o número mínimo de caracteres para uma senha. O intervalo é de 0 a 14 caracteres; o padrão é seis caracteres.
2. MaximumPasswordAge (MAXPWAGE) = 60 Essa configuração define o número máximo de dias em que uma senha é válida. Nenhum limite é especificado usando UNLIMITED. /MAXPWAGE não pode ser menor que /MINPWAGE. O intervalo é de 1 a 999; o padrão é 90 dias
3. Limite de bloqueio (LOCKOUTTHRESHOLD) = 5 Essa configuração define o limite para tentativas de login malsucedidas. Após 5 tentativas incorretas, a conta será bloqueada.

Essas configurações ajudam a melhorar a segurança das senhas e evitar ataques de força bruta, aplicando políticas de senha fortes e bloqueando contas após um certo número de tentativas malsucedidas de login.

2. Políticas de auditoria

As políticas de auditoria são importantes para os contêineres do Windows porque fornecem visibilidade crítica de eventos de segurança, como tentativas de login e uso de privilégios, ajudando a detectar acesso não autorizado, monitorar a atividade do usuário e garantir a conformidade com os padrões regulatórios. Mesmo na natureza efêmera dos contêineres, os registros de auditoria são essenciais para a investigação de incidentes, a detecção proativa de ameaças e a manutenção de uma postura de segurança consistente em ambientes conteinerizados.

Monitoramento e conformidade de segurança:

- Rastreie as atividades do usuário: as políticas de auditoria permitem que os administradores monitorem as atividades do usuário, como tentativas de login e uso de privilégios, dentro do contêiner. Isso é fundamental para detectar acesso não autorizado ou comportamento suspeito.
- Conformidade regulatória: muitas organizações precisam registrar eventos de segurança para fins de conformidade com regulamentações como HIPAA, PCI-DSS e GDPR. A ativação de políticas de auditoria em contêineres garante que você atenda a esses requisitos, mesmo em ambientes em contêineres.

Investigação de incidentes:

- Análise e análise forense: se um aplicativo ou serviço em contêineres for comprometido, os registros de auditoria podem fornecer informações valiosas para a análise pós-incidente. Eles ajudam as equipes de segurança a rastrear as ações tomadas pelos invasores ou identificar como uma violação ocorreu.
- Detecção em tempo real: os registros de auditoria permitem que os administradores configurem alertas em tempo real para eventos críticos (por exemplo, tentativas fracassadas de login, escalonamento de privilégios). Esse monitoramento proativo ajuda a detectar ataques precocemente e permite tempos de resposta mais rápidos.

Consistência em todos os ambientes:

- Postura de segurança uniforme: ao aplicar políticas de auditoria em contêineres por meio do registro, você garante práticas de segurança consistentes em ambientes conteinerizados e não conteinerizados. Isso evita que os contêineres se tornem um ponto cego para o monitoramento de segurança.
- Visibilidade em ambientes híbridos: para organizações que executam servidores e contêineres Windows tradicionais, as políticas de auditoria fornecem visibilidade e controle semelhantes em todas as plataformas, tornando o gerenciamento mais fácil e eficaz.

Rastreando operações privilegiadas:

- Auditoria de uso de privilégios: em ambientes de contêineres em que os aplicativos são executados com privilégios elevados ou onde tarefas administrativas são executadas, a auditoria de operações privilegiadas garante a responsabilidade. Você pode registrar quem acessou recursos confidenciais ou executou tarefas críticas dentro do contêiner.
- Evite o abuso de privilégios: ao monitorar o uso de privilégios, você pode detectar quando usuários não autorizados tentam elevar seus privilégios ou acessar áreas restritas dentro do contêiner, o que ajuda a evitar ataques internos ou externos.

Detectando tentativas de acesso não autorizado:

- Tentativas de login malsucedidas: habilitar políticas de auditoria para tentativas de login malsucedidas ajuda a identificar ataques de força bruta ou tentativas não autorizadas de acessar aplicativos em contêineres. Isso fornece visibilidade sobre quem está tentando obter acesso ao sistema e com que frequência.
- Monitoramento de bloqueio de conta: a auditoria de eventos de bloqueio de conta permite que os administradores detectem e investiguem possíveis bloqueios causados por atividades suspeitas ou maliciosas.

Segurança persistente mesmo em ambientes efêmeros:

- Efêmeros e seguros: embora os contêineres sejam efêmeros, o que significa que podem ser excluídos e recriados com frequência, a auditoria ainda desempenha um papel fundamental para garantir que os eventos de segurança sejam capturados enquanto o contêiner está em execução. Isso garante que os eventos críticos de segurança sejam registrados durante o ciclo de vida do contêiner.

Registro centralizado:

- Encaminhamento de registros para sistemas centralizados: os contêineres podem ser integrados a sistemas de registro centralizados (por exemplo, ELK stack, AWS CloudWatch) para capturar registros de auditoria de várias instâncias de contêineres. Isso permite uma melhor análise e correlação dos eventos de segurança em sua infraestrutura.

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
```

Explicação:

Esta seção configura as políticas de auditoria usando modificações no registro. As políticas de auditoria controlam quais eventos de segurança são registrados pelo Windows, o que ajuda a monitorar e detectar tentativas de acesso não autorizado.

1. SCENoApplyLegacyAuditPolicy = 0 Isso desativa o formato antigo da política de auditoria, permitindo políticas de auditoria mais granulares introduzidas em versões posteriores do Windows. Isso é importante para configurações de auditoria modernas.
2. Subcategoria Auditpol: “Logon” Essa configuração permite a auditoria de eventos de login bem-sucedidos e fracassados. O valor 3 significa que o Windows registrará as tentativas de login bem-sucedidas e malsucedidas. Isso ajuda a monitorar quem está acessando o sistema e detectando tentativas de login malsucedidas.

Essas políticas de auditoria são essenciais para o monitoramento e a conformidade da segurança, pois fornecem registros detalhados de eventos de segurança importantes, como tentativas de login e uso de operações privilegiadas.

3. Melhores práticas de segurança do IIS para contêineres do Windows

A implementação das melhores práticas do IIS em contêineres do Windows é importante por vários motivos, garantindo que seus aplicativos sejam seguros, de alto desempenho e escaláveis. Embora os contêineres forneçam isolamento e um ambiente leve, eles ainda precisam de uma configuração adequada para evitar vulnerabilidades e problemas operacionais. Veja por que seguir as melhores práticas para o IIS em contêineres do Windows é crucial:

Segurança

- Prevenção de vulnerabilidades comuns: o IIS geralmente é alvo de ataques como cross-site scripting (XSS), clickjacking e divulgação de informações. A implementação de cabeçalhos de segurança (por exemplo X-Content-Type-Options X-Frame-Options,, e Strict-Transport-Security) ajuda a proteger seu aplicativo contra essas ameaças.
- O isolamento não é suficiente: os contêineres são isolados, mas uma instância do IIS mal configurada pode expor informações confidenciais, como detalhes da versão do servidor, listagens de diretórios ou comunicações não criptografadas. Ao desativar recursos como navegação em diretórios e remover o cabeçalho da versão do IIS, você minimiza a superfície de ataque.
- Criptografia e HTTPS: as melhores práticas, como impor conexões somente HTTPS, garantem que os dados em trânsito sejam criptografados, protegendo informações confidenciais de serem interceptadas.

Desempenho

- Uso eficiente de recursos: as melhores práticas do IIS, como habilitar a compactação dinâmica e estática, reduzem o uso da largura de banda e melhoram os tempos de carregamento. Essas otimizações são especialmente importantes em ambientes em contêineres, nos quais os recursos são compartilhados entre os contêineres e o sistema host.
- Registro otimizado: a configuração adequada do registro (por exemplo, incluindo o X-Forwarded-For cabeçalho) garante que você possa rastrear a atividade do cliente e, ao mesmo tempo, minimizar a sobrecarga de registro desnecessária. Isso ajuda você a coletar dados relevantes para a solução de problemas sem prejudicar o desempenho.

Escalabilidade e capacidade de manutenção

- Consistência entre ambientes: ao seguir as melhores práticas, você garante que a configuração do IIS seja consistente em várias instâncias de contêiner. Isso simplifica o escalonamento e garante

que, quando novos contêineres forem implantados, eles sigam as mesmas diretrizes de segurança e desempenho.

- Configurações automatizadas: as melhores práticas em Dockerfiles, como definir permissões de pastas e desativar recursos desnecessários, garantem que cada novo contêiner seja configurado automaticamente corretamente. Isso reduz a intervenção manual e diminui o risco de erro humano.

Conformidade

- Atendimento aos requisitos regulatórios: muitos setores têm requisitos regulatórios rígidos (por exemplo, PCI-DSS, HIPAA) que exigem medidas de segurança específicas, como comunicações criptografadas (HTTPS) e registro de solicitações de clientes. Seguir as melhores práticas do IIS em contêineres ajuda a garantir a conformidade com esses padrões.
- Auditabilidade: a implementação de políticas de auditoria e o registro seguro permitem a rastreabilidade de eventos, o que é fundamental nas auditorias. Por exemplo, registrar o X-Forwarded-For cabeçalho garante que os endereços IP do cliente sejam registrados corretamente em arquiteturas baseadas em proxy.

Minimizando o risco em ambientes compartilhados

- Evitando configurações incorretas: os contêineres compartilham o kernel do host e, embora estejam isolados uns dos outros, uma instância do IIS mal configurada pode expor vulnerabilidades ou criar gargalos de desempenho. As melhores práticas garantem que cada instância do IIS seja executada de maneira ideal, reduzindo o risco de problemas entre contêineres.
- Acesso com privilégios mínimos: definir as permissões adequadas para pastas e arquivos dentro do contêiner (por exemplo, usar Set-Acl in PowerShell) garante que os usuários e processos dentro do contêiner tenham apenas o acesso necessário, reduzindo o risco de escalonamento de privilégios ou adulteração de dados.

Resiliência em ambientes efêmeros

- Natureza efêmera dos contêineres: os contêineres geralmente têm vida curta e são reconstruídos com frequência. A aplicação das melhores práticas do IIS garante que cada contêiner seja configurado de forma segura e consistente, independentemente de quantas vezes ele seja reimplantado. Isso evita que configurações incorretas sejam introduzidas ao longo do tempo.

- Mitigação de possíveis configurações incorretas: ao aplicar automaticamente as melhores práticas (por exemplo, desabilitar protocolos ou cabeçalhos fracos), o risco de uma configuração incorreta durante a reinicialização ou atualização do contêiner é minimizado.

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"
```

Explicação:

Esse comando configura o IIS para registrar o X-Forwarded-For cabeçalho, que geralmente é usado para capturar o endereço IP original do cliente quando uma solicitação passa por um proxy ou balanceador de carga. Por padrão, o IIS registra somente o endereço IP do balanceador de carga ou

do proxy reverso, portanto, adicionar esse campo de log personalizado ajuda a rastrear o verdadeiro IP do cliente para auditoria, análise e solução de problemas de segurança.

1. X-Forwarded-For cabeçalho que é comumente usado para capturar o endereço IP original do cliente quando uma solicitação passa por um proxy ou balanceador de carga. Por padrão, o IIS registra somente o endereço IP do balanceador de carga ou do proxy reverso, portanto, adicionar esse campo de log personalizado ajuda a rastrear o verdadeiro IP do cliente para auditoria, análise e solução de problemas de segurança.
2. Strict-Transport-Security (HSTS) Garante que os navegadores se comuniquem somente por HTTPS. O max-age=31536000 especifica que essa política é aplicada por 1 ano e aplica a política a todos os subdomínios. includeSubDomains
3. X-Content-Type-Options impede que os navegadores “detectem MIME” uma resposta diferente do tipo de conteúdo declarado. Isso ajuda a evitar alguns tipos de ataques.
4. X-XSS-Protection Permite a proteção de Cross-Site Scripting (XSS) em navegadores.
5. X-Frame-Options Impede que a página seja incorporada em iframes, protegendo contra ataques de clickjacking.
6. Desabilitar a divulgação da versão do IIS Esse comando desativa o cabeçalho do servidor nas respostas HTTP, o que, por padrão, revela a versão do IIS que está sendo usada. Ocultar essas informações ajuda a reduzir o risco de invasores identificarem e atacarem vulnerabilidades específicas da versão do IIS.
7. Ativar conexões somente HTTPS Esta seção (comentada) impõe conexões HTTPS e desativa o HTTP. Se não for comentado, o Dockerfile configurará o IIS para escutar somente na porta 443 (HTTPS) e removerá a associação HTTP padrão na porta 80. Isso é útil ao encerrar o SSL dentro do contêiner e garante que todo o tráfego seja criptografado.
8. Desativar a navegação no diretório Impede que o IIS mostre uma listagem de diretórios quando nenhum documento padrão está presente. Isso evita expor a estrutura interna do arquivo aos usuários.
9. Passar por páginas de erro personalizadas Garante que, se o aplicativo tiver seu próprio tratamento de erros, o IIS permitirá que as páginas de erro do aplicativo passem em vez de mostrar as páginas de erro padrão do IIS.
10. Modo de erro detalhado configura o IIS para exibir mensagens de erro detalhadas somente para solicitações locais, ajudando os desenvolvedores a diagnosticar problemas sem expor informações confidenciais a usuários externos.
11. Garanta as permissões de pasta adequadas Esse bloco configura as permissões de pasta para a raiz da web do IIS (C:\inetpub\wwwroot). Ele define as permissões de leitura e execução para os

grupos IIS_IUSRS e Usuários, garantindo que esses usuários possam acessar a pasta, mas não modificar arquivos. Definir as permissões corretas minimiza o risco de acesso não autorizado ou adulteração dos arquivos hospedados pelo servidor web.

Seguir as melhores práticas do IIS em contêineres do Windows garante que seus aplicativos em contêineres sejam seguros, de alto desempenho e escaláveis. Essas práticas ajudam a evitar vulnerabilidades, otimizar o uso de recursos, garantir a conformidade e manter a consistência em todas as instâncias do contêiner. Embora os contêineres sejam projetados para serem isolados, a configuração adequada é necessária para minimizar os riscos e garantir a confiabilidade de sua aplicação em ambientes de produção.

4. Princípio do privilégio mínimo

O Princípio do Privilégio Mínimo (PoLP) é crucial para contêineres Windows por vários motivos importantes, principalmente para aumentar a segurança e minimizar os riscos em ambientes em contêineres. Esse princípio determina que um sistema ou aplicativo deve operar com o nível mínimo de permissões necessárias para funcionar adequadamente. Veja por que isso é importante nos contêineres do Windows:

Minimizando a superfície de ataque

- Os contêineres geralmente executam aplicativos que interagem com vários componentes do sistema e, quanto mais privilégios um aplicativo tiver, mais amplo será seu acesso a esses componentes. Ao limitar as permissões do contêiner apenas ao necessário, o PolP reduz significativamente a superfície de ataque, tornando mais difícil para um invasor explorar o contêiner se ele for comprometido.

Limitando o impacto de contêineres comprometidos

- Se um contêiner do Windows for comprometido, a execução de aplicativos com privilégios excessivos (por exemplo, acesso de administrador ou de nível raiz) pode permitir que um invasor tenha controle sobre arquivos críticos do sistema ou aumente os privilégios em todo o host do contêiner. Ao aplicar o PolP, mesmo que um contêiner seja violado, o invasor fica limitado no que pode fazer, impedindo uma maior escalação e acesso a recursos confidenciais ou outros contêineres.

Proteção em ambientes multilocatários

- Em ambientes corporativos ou em nuvem, vários contêineres podem ser executados na mesma infraestrutura física ou virtual. O PoIP garante que um contêiner comprometido não tenha a capacidade de acessar recursos ou dados pertencentes a outros locatários. Esse isolamento é crucial para manter a segurança em ambientes compartilhados e multilocatários, protegendo contra movimentos laterais entre contêineres.

Mitigando a escalação de privilégios

- Os contêineres que funcionam com altos privilégios podem ser usados por invasores para aumentar os privilégios no sistema. O PoIP mitiga esse risco restringindo o acesso do contêiner aos recursos do sistema, evitando ações não autorizadas ou escalonamentos de privilégios além do ambiente do contêiner.

Conformidade e auditoria

- Muitos padrões regulatórios e estruturas de segurança (por exemplo, PCI DSS, HIPAA, GDPR) exigem que os sistemas sigam o PoIP para limitar o acesso a dados confidenciais. A execução de contêineres do Windows com privilégios restritos ajuda as organizações a cumprir essas regulamentações e garante que os aplicativos tenham acesso somente aos recursos de que precisam especificamente.

Reduzindo o risco de configuração incorreta

- Quando os contêineres são executados com privilégios desnecessários, até mesmo uma pequena configuração incorreta pode levar a graves vulnerabilidades de segurança. Por exemplo, se um contêiner executado como administrador for acidentalmente exposto à Internet, um invasor poderá obter o controle do sistema. O PoIP ajuda a evitar esses riscos ao adotar privilégios limitados, tornando as configurações incorretas menos perigosas.

Postura aprimorada de segurança de contêineres

- Seguindo o PoIP, os contêineres ficam melhor isolados do sistema hospedeiro subjacente e uns dos outros. Isso garante que o aplicativo em contêiner tenha menos probabilidade de acessar ou modificar arquivos ou processos do sistema fora do escopo definido, preservando a integridade do sistema operacional do host e de outras cargas de trabalho.

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
```

Explicação:

Nesta seção, o ContainerUser comando USER especifica que o aplicativo dentro do contêiner do Windows deve ser executado sob a ContainerUser conta em vez da conta padrão do administrador.

Veja por que isso é importante, especialmente em um ambiente multilocatário:

1. Princípio do menor privilégio: a ContainerUser conta é um usuário não administrativo com privilégios limitados. A execução do aplicativo nessa conta segue o princípio do menor privilégio, o que ajuda a minimizar o risco de exploração. Se um invasor comprometesse o aplicativo, ele teria acesso limitado ao sistema, reduzindo os possíveis danos.
2. Segurança aprimorada: em ambientes multilocatários, os contêineres podem compartilhar a mesma infraestrutura subjacente. Executar como ContainerUser garante que, mesmo que um contêiner seja comprometido, ele não tenha privilégios administrativos para acessar ou modificar arquivos críticos do sistema ou outros contêineres. Isso reduz significativamente a superfície de ataque.
3. Evitar o acesso root: por padrão, os contêineres podem ser executados com permissões elevadas (semelhantes ao acesso root em contêineres Linux), o que pode ser perigoso se explorado. O uso ContainerUser garante que o aplicativo não seja executado com direitos administrativos desnecessários, dificultando que os invasores aumentem os privilégios.
4. Prática recomendada para ambientes multilocatários: em ambientes em que vários usuários ou organizações compartilham a mesma infraestrutura (como na nuvem), a segurança é fundamental. A execução de aplicativos com permissões restritas impede que o aplicativo de um inquilino afete outros, protegendo dados e recursos confidenciais em toda a plataforma.

O ContainerUser comando USER garante que o aplicativo seja executado com privilégios mínimos, aumentando a segurança em ambientes multilocatários ao limitar os danos que poderiam ser causados se o contêiner fosse comprometido. Essa é uma prática recomendada para evitar acesso não autorizado ou escalonamento de privilégios em um ambiente em contêineres.

O Princípio do Privilégio Mínimo é essencial para contêineres do Windows porque limita o impacto potencial das violações de segurança, reduz a superfície de ataque e impede o acesso não

autorizado a componentes críticos do sistema. Ao executar aplicativos em contêineres apenas com as permissões necessárias, as organizações podem melhorar significativamente a segurança e a estabilidade de seus ambientes de contêineres, especialmente em infraestruturas multilocatárias e compartilhadas.

Considerações finais: Por que proteger seus contêineres do Windows é essencial no cenário atual de ameaças

No mundo digital em rápida evolução de hoje, onde as ameaças estão se tornando mais sofisticadas e abundantes, proteger seus contêineres do Windows não é apenas uma recomendação, é uma necessidade absoluta. Os contêineres fornecem uma maneira leve e flexível de empacotar e implantar aplicativos, mas não estão imunes às vulnerabilidades de segurança. À medida que mais empresas adotam contêineres para otimizar sua infraestrutura, elas também se tornam um alvo potencial para ataques cibernéticos, se não estiverem devidamente protegidas.

A Internet está repleta de várias ameaças, desde agentes mal-intencionados que visam vulnerabilidades não corrigidas até bots automatizados que verificam configurações incorretas. Sem as medidas de segurança corretas, os contêineres podem ser explorados para expor dados confidenciais, aumentar privilégios ou servir como pontos de entrada para ataques que podem comprometer sua infraestrutura mais ampla. Isso torna a segurança de contêineres tão crítica quanto proteger qualquer outra parte do seu ambiente.

Ao usar contêineres do Windows, muitas práticas recomendadas de segurança tradicionais ainda se aplicam. Implementar políticas de conta robustas, proteger as configurações do IIS, aplicar HTTPS, usar regras rígidas de firewall e aplicar o menor privilégio de acesso a arquivos essenciais são medidas fundamentais para garantir que o contêiner permaneça resiliente contra ataques. Além disso, a auditoria e o registro regulares fornecem visibilidade do que está acontecendo dentro do contêiner, permitindo que você capture atividades suspeitas antes que elas se transformem em um incidente completo.

A proteção de contêineres do Windows também está alinhada aos requisitos regulatórios que exigem a proteção de dados confidenciais e a garantia da integridade do aplicativo. À medida que as arquiteturas nativas em nuvem e em contêineres se tornam mais predominantes, garantir a segurança em todas as camadas, da imagem base ao contêiner em execução, ajudará a proteger suas operações e manter a confiança do cliente.

Em resumo, o aumento de aplicativos em contêineres, juntamente com o crescente número de ameaças cibernéticas, torna a segurança de contêineres um aspecto inegociável do gerenciamento

moderno da infraestrutura. Ao aderir às melhores práticas e monitorar continuamente as vulnerabilidades, as empresas podem aproveitar a agilidade e a eficiência dos contêineres Windows sem comprometer a segurança. Nesse ambiente cheio de ameaças, proteger seus contêineres do Windows não é apenas uma opção, é essencial.

Melhores práticas para implantações híbridas

Este guia fornece orientação sobre como executar implantações em ambientes locais ou periféricos com EKS Hybrid Nodes ou EKS Anywhere.

Atualmente, publicamos guias para os seguintes tópicos:

- Melhores práticas para nós híbridos EKS e desconexões de rede

Nodos híbridos EKS e desconexões de rede

A arquitetura EKS Hybrid Nodes pode ser nova para clientes que estão acostumados a executar clusters Kubernetes locais inteiramente em seus próprios data centers ou locais periféricos. Com os EKS Hybrid Nodes, o plano de controle do Kubernetes é executado em uma região da AWS e somente os nós são executados localmente, resultando em uma arquitetura de cluster Kubernetes “ampliada” ou “estendida”.

Isso leva a uma pergunta comum: “O que acontece se meus nós forem desconectados do plano de controle do Kubernetes?”

Neste guia, respondemos a essa pergunta por meio de uma análise dos tópicos a seguir. É recomendável validar a estabilidade e a confiabilidade de seus aplicativos por meio de desconexões de rede, pois cada aplicativo pode se comportar de forma diferente com base em suas dependências, configuração e ambiente. Consulte o eks-hybrid-examples GitHub aws-samples/ repo para ver a configuração, os procedimentos e os resultados dos testes que você pode consultar para testar desconexões de rede com os EKS Hybrid Nodes e seus próprios aplicativos. O GitHub repositório também contém detalhes adicionais dos testes usados para validar o comportamento explicado neste guia.

- Melhores práticas para estabilidade por meio de desconexões de rede
- Comportamento de failover do pod Kubernetes por meio de desconexões de rede
- Tráfego de rede de aplicativos por meio de desconexões de rede
- Credenciais do host por meio de desconexões de rede

Melhores práticas para estabilidade por meio de desconexões de rede

Rede de alta disponibilidade

A melhor abordagem para evitar desconexões de rede entre os nós híbridos e o plano de controle do Kubernetes é usar conexões redundantes e resilientes do seu ambiente local de e para a AWS. Consulte o kit de ferramentas de resiliência do AWS Direct Connect e a documentação do AWS Site-to-Site VPN para obter mais informações sobre a arquitetura de redes híbridas altamente disponíveis com essas soluções.

Aplicativos altamente disponíveis

Ao arquitetar aplicativos, considere seus domínios de falha e os efeitos dos diferentes tipos de interrupções. O Kubernetes fornece mecanismos integrados para implantar e manter réplicas de aplicativos em domínios de nós, zonas e regiões. O uso desses mecanismos depende da arquitetura do aplicativo, dos ambientes e dos requisitos de disponibilidade. Por exemplo, aplicativos sem estado geralmente podem ser implantados com várias réplicas e podem se mover entre hosts arbitrários e capacidade de infraestrutura, e você pode usar seletores de nós e restrições de distribuição de topologia para executar instâncias do aplicativo em diferentes domínios. Para obter detalhes sobre técnicas em nível de aplicativo para criar aplicativos resilientes no Kubernetes, consulte o Guia de melhores práticas do EKS.

O Kubernetes avalia as informações zonais dos nós que estão desconectados do plano de controle do Kubernetes ao determinar se os pods devem ser movidos para outros nós. Se todos os nós em uma zona estiverem inacessíveis, o Kubernetes cancelará os despejos de pod dos nós dessa zona. Como prática recomendada, se você tiver uma implantação com nós em execução em vários data centers ou locais físicos, atribua uma zona a cada nó com base em seu data center ou localização física. Quando você executa o EKS com nós na nuvem, esse rótulo de zona é aplicado automaticamente pela AWS cloud-controller-manager. No entanto, a não cloud-controller-manager é usado com nós híbridos, então você pode passar essas informações por meio da configuração do kubelet. Um exemplo de como configurar uma zona em sua configuração de nó para nós híbridos é mostrado abaixo. A configuração é passada quando você conecta seus nós híbridos ao seu cluster com os nós híbridos CLI ()`nodeadm`. Para obter mais informações sobre o `topology.kubernetes.io/zone` rótulo, consulte a documentação do Kubernetes. Para obter mais informações sobre a CLI de nós híbridos, consulte a referência nodeadm de nós 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:
    ...
```

Monitoramento de rede

Se você usa o AWS Direct Connect ou o AWS Site-to-Site VPN para sua conectividade híbrida, você pode aproveitar CloudWatch os alarmes, registros e métricas para observar o estado da sua conexão híbrida e diagnosticar problemas. Para obter mais informações, consulte Monitoramento de recursos do AWS Direct Connect e Monitorar uma conexão Site-to-Site VPN da AWS.

É recomendável criar alarmes para `NodeNotReady` eventos relatados pela node-lifecycle-controller execução no plano de controle do EKS, o que indica que um nó híbrido pode estar passando por uma desconexão de rede. Você pode criar esse alarme ativando o registro do plano de controle EKS para o Controller Manager e criando um filtro métrico CloudWatch para a mensagem “Gravando mensagem de evento de alteração de status para nó” com o status = “NodeNotReady”. Depois de criar um filtro métrico, você pode criar um alarme para esse filtro com base nos limites desejados. Para obter mais informações, consulte Alarmes para registros na CloudWatch documentação.

Você pode usar as métricas integradas do Transit Gateway (TGW) e do Virtual Private Gateway (VGW) para observar o tráfego de rede que entra e sai do seu TGW ou VGW. Você pode criar alarmes para essas métricas para detectar cenários em que o tráfego de rede cai abaixo dos níveis normais, indicando um possível problema de rede entre os nós híbridos e o plano de controle do EKS. As métricas TGW e VGW estão descritas na tabela a seguir.

Gateway	Métrica	Description
Gateway de trânsito	BytesIn	Os bytes recebidos pelo TGW do anexo (plano de controle EKS para nós híbridos)
Gateway de trânsito	BytesOut	Os bytes enviados do TGW para o anexo (nós híbridos para o plano de controle EKS)

Gateway	Métrica	Description
Gateway privado virtual	TunnelDataIn	Os bytes enviados do lado AWS da conexão através do túnel VPN até o gateway do cliente (plano de controle EKS para nós híbridos)
Gateway privado virtual	TunnelDataOut	Os bytes recebidos no lado AWS da conexão por meio do túnel VPN do gateway do cliente (nós híbridos para o plano de controle EKS)

Você também pode usar o CloudWatch Network Monitor para obter uma visão mais profunda de suas conexões híbridas para reduzir o tempo médio de recuperação e determinar se os problemas de rede se originam na AWS ou no seu ambiente. CloudWatch O Network Monitor pode ser usado para visualizar a perda e a latência de pacotes em suas conexões de rede híbrida, definir alertas e limites e, em seguida, tomar medidas para melhorar o desempenho da rede. Para obter mais informações, consulte Usando o Amazon CloudWatch Network Monitor.

O EKS oferece várias opções para monitorar a integridade de seus clusters e aplicativos. Para a integridade do cluster, você pode usar o painel de observabilidade no console do EKS para detectar, solucionar problemas e remediar problemas rapidamente. Você também pode usar o Amazon Managed Service para Prometheus, o AWS Distro for Open Telemetry (ADOT) e para monitoramento de clusters CloudWatch , aplicativos e infraestrutura. Para obter mais informações sobre as opções de observabilidade do EKS, consulte Monitore o desempenho do seu cluster e visualize os registros.

Solução de problemas local

Para se preparar para desconexões de rede entre os nós híbridos e o plano de controle do EKS, você pode configurar back-ends secundários de monitoramento e registro para manter a observabilidade dos aplicativos quando os serviços regionais da AWS não estiverem acessíveis. Por exemplo, você pode configurar o coletor AWS Distro for Open Telemetry (ADOT) para enviar métricas e registros para vários back-ends. Você também pode usar ferramentas locais, como a `crictl` CLI, para interagir localmente com pods e contêineres como substituto `kubectl` ou outros clientes compatíveis com a API Kubernetes que normalmente consultam o endpoint do servidor da API Kubernetes. Para obter mais informações`crictl`, consulte a `crictl`documentação nas ferramentas GitHub cri-. Alguns `crictl` comandos úteis estão listados abaixo.

Liste os pods em execução no host:

```
crictl pods
```

Liste os contêineres em execução no host:

```
crictl ps
```

Listar imagens em execução no host:

```
crictl images
```

Obtenha registros de um contêiner em execução no host:

```
crictl logs CONTAINER_NAME
```

Obtenha estatísticas dos pods em execução no host:

```
crictl statsp
```

Tráfego de rede de aplicativos

Ao usar nós híbridos, é importante considerar e compreender os fluxos de rede do tráfego do seu aplicativo e as tecnologias que você usa para expor seus aplicativos externamente ao seu cluster. Diferentes tecnologias para balanceamento de carga e entrada de aplicativos se comportam de forma diferente durante as desconexões da rede. Por exemplo, se você estiver usando o recurso BGP Control Plane do Cilium para balanceamento de carga de aplicativos, a sessão BGP de seus pods e serviços poderá ficar inativa durante as desconexões da rede. Isso acontece porque a funcionalidade do alto-falante BGP está integrada ao agente Cilium, e o agente Cilium será reiniciado continuamente quando desconectado do plano de controle do Kubernetes. O motivo da reinicialização é devido à falha na verificação de integridade do Cilium, pois sua integridade está associada ao acesso ao plano de controle do Kubernetes (consulte CFP: #31702 com uma melhoria opcional no Cilium v1.17). Da mesma forma, se você estiver usando Application Load Balancers (ALB) ou Network Load Balancers (NLB) para tráfego de aplicativos originado na região da AWS, esse tráfego poderá ficar temporariamente inativo se seu ambiente local perder a conectividade com a região da AWS. É recomendável validar se as tecnologias que você usa para balanceamento de carga e entrada permanecem estáveis durante as desconexões da rede antes da implantação na produção. O exemplo no eks-hybrid-examples GitHub aws-samples/repo usa o MetalLB para balanceamento de carga no modo L2, que permanece estável durante as desconexões de rede entre os nós híbridos e o plano de controle EKS.

Analise as dependências em serviços remotos da AWS

Ao usar nós híbridos, esteja ciente das dependências que você assume nos serviços regionais da AWS que são externos ao seu ambiente local ou de borda. Os exemplos incluem acessar o Amazon S3 ou o Amazon RDS para obter dados de aplicativos, usar o Amazon Managed Service para Prometheus ou para métricas e logs, usar balanceadores de carga de aplicativos e de rede CloudWatch para tráfego originado na região e retirar contêineres do Amazon Elastic Container Registry. Esses serviços não estarão acessíveis durante as desconexões de rede entre seu ambiente local e a AWS. Se seu ambiente local estiver propenso a desconexões de rede com a AWS, analise seu uso dos serviços da AWS e garanta que a perda de uma conexão com esses serviços não comprometa a estabilidade estática de seus aplicativos.

Ajuste o comportamento de failover do pod Kubernetes

Há opções para ajustar o comportamento de failover de pod durante desconexões de rede para aplicativos que não são portáteis entre hosts ou para ambientes com recursos limitados que não têm capacidade disponível para failover de pod. Geralmente, é importante considerar os requisitos de recursos de seus aplicativos e ter capacidade suficiente para que uma ou mais instâncias do aplicativo passem para um host diferente se um nó falhar.

- Opção 1 - Uso DaemonSets: essa opção se aplica a aplicativos que podem e devem ser executados em todos os nós do cluster. DaemonSets são configurados automaticamente para tolerar a contaminação inacessível, que mantém os DaemonSet pods vinculados a seus nós por meio de desconexões de rede.
- Opção 2 - Ajuste `tolerationSeconds` para manchas inacessíveis: você pode ajustar a quantidade de tempo que seus pods permanecem vinculados aos nós durante as desconexões da rede. Faça isso configurando os pods de aplicativos para tolerar a contaminação inacessível com o `NoExecute` efeito por um período especificado (`tolerationSeconds`na especificação do aplicativo). Com essa opção, quando há desconexões de rede, seus pods de aplicativos permanecem vinculados aos nós até `tolerationSeconds` expirarem. Considere isso com cuidado, `tolerationSeconds` pois aumentar a contaminação inacessível `NoExecute` significa que os grupos que funcionam em hospedeiros inacessíveis podem levar mais tempo para serem transferidos para outros hospedeiros saudáveis e acessíveis.
- Opção 3: Controlador personalizado: você pode criar e executar um controlador personalizado (ou outro software) que monitore o Kubernetes em busca da contaminação inacessível causada pelo efeito. `NoExecute` Quando essa contaminação é detectada, o controlador personalizado pode verificar as métricas específicas do aplicativo para avaliar a integridade do aplicativo. Se

o aplicativo estiver íntegro, o controlador personalizado poderá remover a mancha inacessível, impedindo o despejo de pods dos nós durante as desconexões da rede.

Um exemplo de como configurar uma implantação com `tolerationSeconds` for the unreachable taint é mostrado abaixo. No exemplo, `tolerationSeconds` está definido como `1800` (30 minutos), o que significa que os pods executados em nós inacessíveis só serão removidos se a desconexão da rede durar mais de 30 minutos.

```
apiVersion: apps/v1
kind: Deployment
metadata:
...
spec:
...
      tolerations:
      - key: "node.kubernetes.io/unreachable"
        operator: "Exists"
        effect: "NoExecute"
        tolerationSeconds: 1800
```

Failover do pod Kubernetes por meio de desconexões de rede

Começamos com uma revisão dos principais conceitos, componentes e configurações que influenciam o comportamento do Kubernetes durante as desconexões de rede entre os nós e o plano de controle do Kubernetes. O EKS está em conformidade com o Kubernetes upstream, então todos os conceitos, componentes e configurações do Kubernetes descritos aqui se aplicam às implantações de EKS e EKS Hybrid Nodes.

Foram feitas melhorias no EKS especificamente para melhorar o comportamento de failover do pod durante desconexões de rede. Para obter mais informações, consulte os GitHub problemas #131294 e #131481 no repositório upstream do Kubernetes.

Conceitos

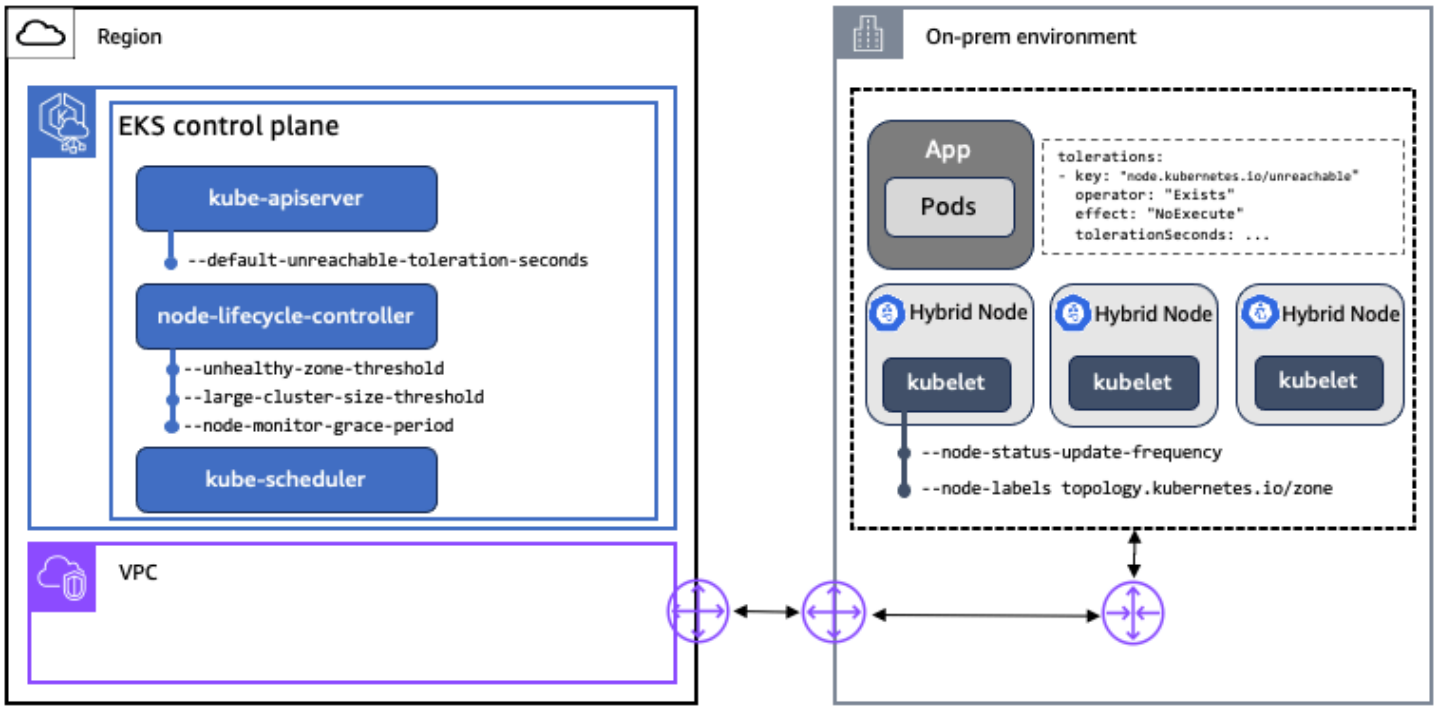
Contaminações e tolerações: manchas e tolerações são usadas no Kubernetes para controlar o agendamento de pods em nós. As manchas são definidas pelo node-lifecycle-controller para indicar que os nós não são elegíveis para agendamento ou que os pods desses nós devem ser despejados. Quando os nós estão inacessíveis devido a uma desconexão de rede, aplica-se o 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/unreachableA mancha corresponde ao NodeCondition Pronto ser Desconhecido. Os usuários podem especificar tolerâncias para manchas no nível da aplicação no. PodSpec

- NoSchedule: Não há novos pods programados no nó contaminado, a menos que tenham uma tolerância equivalente. Os pods que já estão em execução no node não são despejados.
- NoExecute: Os frutos que não toleram a contaminação são despejados imediatamente. Os pods que toleram a mancha (sem especificar tolerationSeconds) permanecem vinculados para sempre. Os pods que toleram a contaminação com uma tolerância especificada. Os segundos permanecem limitados pelo tempo especificado. Depois de decorrido esse tempo, o controlador do ciclo de vida do nó expulsa os pods do nó.

Locações de nós: o Kubernetes usa a API Lease para comunicar os heartbeats do kubelet node ao servidor da API Kubernetes. Para cada nó, há um objeto Lease com um nome correspondente. Internamente, cada pulsação do kubelet atualiza o campo spec.renewTime do objeto Lease. O plano de controle do Kubernetes usa o timestamp desse campo para determinar a disponibilidade do nó. Se os nós estiverem desconectados do plano de controle do Kubernetes, eles não poderão atualizar spec.renewTime para sua locação, e o plano de controle interpreta isso como Pronto sendo desconhecido. NodeCondition

Componentes



Componente	Subcomponente	Description
Plano de controle do Kubernetes	kube-api-server	O servidor da API é um componente principal do plano de controle do Kubernetes que expõe a API do Kubernetes.
Plano de controle do Kubernetes	node-lifecycle-controller	Um dos controladores que ele kube-controller-manager executa. Ele é responsável por detectar e responder aos problemas dos nós.
Plano de controle do Kubernetes	programador de cubos	Um componente do plano de controle que observa os pods recém-criados sem nenhum nó atribuído e seleciona um nó no qual eles serão executados.
Nó do Kubernetes	kubelet	Um agente que é executado em cada nó do cluster. O kubelet observa PodSpecs e garante que os contêineres descritos neles PodSpecs estejam funcionando e saudáveis.

Definições de configuração

Componente	Configuração	Description	Padrão do K8s	Padrão EKS	Configurável no EKS
kube-api-server	default-unreachable-toleration-seconds	Indica se a tolerância `unreachable:NoExecute` que é adicionada por padrão a cada cápsula que ainda não tem essa tolerância. `tolerationSeconds`	300	300	Não
node-lifecycle-controller	node-monitor-grace-period	A quantidade de tempo que um nó pode ficar sem responder antes de ser marcado como não íntegro. Deve ser N vezes maior que o do kubelet `nodeStatusUpdateFrequency` , onde N é o número de novas	40	40	Não

Compor e	Configuração	Description	Padrão do K8s	Padrão EKS	Configurá vel no EKS
		tentativas permitidas para que o kubelet publique o status do nó.			
node-life cycle-con troller	large-cluster-size-threshold	O número de nós nos quais o cluster é node-lifecycle-controller tratado como grande para a lógica de despejo. `--secondary-node-eviction-rate` é substituído por 0 para clusters desse tamanho ou menores.	50	100.000	Não
node-life cycle-con troller	unhealthy-zone-threshold	A porcentagem de nós em uma zona que não deve estar pronta para que essa zona seja tratada como não íntegra.	55%	55%	Não
kubelet	node-stat us-update-frequency	Com que frequência o kubelet publica o status do nó no plano de controle. Deve ser compatível com `nodeMonit orGracePeriod` in node-lifecycle-con troller.	10	10	Sim
kubelet	rótulos de nós	Rótulos a serem adicionados ao registrar o nó no cluster. O rótulo `topology. kubernetes.io/zone` pode ser especificado com nós híbridos para agrupar nós em zonas.	Nenhum	Nenhum	Sim

Failover do pod Kubernetes por meio de desconexões de rede

O comportamento descrito aqui pressupõe que os pods estejam sendo executados como implantações do Kubernetes com configurações padrão e que o EKS seja usado como o provedor do Kubernetes. O comportamento real pode ser diferente com base no ambiente, no tipo de desconexão da rede, nos aplicativos, nas dependências e na configuração do cluster. O conteúdo deste guia foi

validado usando um aplicativo específico, uma configuração de cluster e um subconjunto de plug-ins. É altamente recomendável testar o comportamento em seu próprio ambiente e com seus próprios aplicativos antes de passar para a produção.

Quando há desconexões de rede entre os nós e o plano de controle do Kubernetes, o kubelet em cada nó desconectado não pode se comunicar com o plano de controle do Kubernetes. Consequentemente, o kubelet não pode despejar pods nesses nós até que a conexão seja restaurada. Isso significa que os pods executados nesses nós antes da desconexão da rede continuam funcionando durante a desconexão, supondo que nenhuma outra falha faça com que eles sejam desligados. Em resumo, você pode obter estabilidade estática durante as desconexões de rede entre os nós e o plano de controle do Kubernetes, mas não pode realizar operações de mutação em seus nós ou cargas de trabalho até que a conexão seja restaurada.

Há cinco cenários principais que produzem diferentes comportamentos de failover do pod com base na natureza da desconexão da rede. Em todos os cenários, o cluster fica saudável novamente sem a intervenção do operador quando os nós se reconectam ao plano de controle do Kubernetes. Os cenários abaixo descrevem os resultados esperados com base em nossas observações, mas esses resultados podem não se aplicar a todas as configurações possíveis de aplicativos e clusters.

Cenário 1: interrupção total do cluster

Resultado esperado: os pods em nós inacessíveis não são despejados e continuam funcionando nesses nós.

Uma interrupção total do cluster significa que todos os nós do cluster estão desconectados do plano de controle do Kubernetes. Nesse cenário, o node-lifecycle-controller no plano de controle detecta que todos os nós do cluster estão inacessíveis e cancela qualquer despejo de pod.

Os administradores do cluster verão todos os nós com status `Not Ready` durante a desconexão. O status do pod não muda e nenhum novo pod é programado em nenhum nó durante a desconexão e a reconexão subsequente.

Cenário 2: interrupção total da zona

Resultado esperado: os pods em nós inacessíveis não são despejados e continuam funcionando nesses nós.

Uma interrupção completa da zona significa que todos os nós na zona estão desconectados do plano de controle do Kubernetes. Nesse cenário, o node-lifecycle-controller no plano de controle detecta que todos os nós na zona estão inacessíveis e cancela qualquer despejo do pod.

Os administradores do cluster verão todos os nós com status `Not Ready` durante a desconexão. O status do pod não muda e nenhum novo pod é programado em nenhum nó durante a desconexão e a reconexão subsequente.

Cenário 3: interrupção da zona majoritária

Resultado esperado: os pods em nós inacessíveis não são despejados e continuam funcionando nesses nós.

Uma interrupção de zona majoritária significa que a maioria dos nós em uma determinada zona está desconectada do plano de controle do Kubernetes. As zonas no Kubernetes são definidas por nós com o mesmo rótulo. `topology.kubernetes.io/zone` Se nenhuma zona for definida no cluster, uma interrupção majoritária significa que a maioria dos nós em todo o cluster está desconectada. Por padrão, a maioria é definida pelo node-lifecycle-controller sun`healthy-zone-threshold`, que é definido como 55% no Kubernetes e no EKS. Como `large-cluster-size-threshold` está definido como 100.000 no EKS, se 55% ou mais dos nós em uma zona estiverem inacessíveis, os despejos de pod serão cancelados (já que a maioria dos clusters é muito menor do que 100.000 nós).

Os administradores de cluster verão a maioria dos nós na zona com status `Not Ready` durante a desconexão, mas o status dos pods não mudará e eles não serão reprogramados em outros nós.

Observe que o comportamento acima se aplica somente a clusters maiores que três nós. Em grupos de três nós ou menos, os pods em nós inacessíveis são programados para serem despejados e os novos pods são programados em nós saudáveis.

Durante os testes, observamos ocasionalmente que os pods foram removidos de exatamente um nó inacessível durante as desconexões da rede, mesmo quando a maioria dos nós da zona estava inacessível. Ainda estamos investigando uma possível condição de corrida no Kubernetes node-lifecycle-controller como a causa desse comportamento.

Cenário 4: Interrupção da zona minoritária

Resultado esperado: os pods são removidos de nós inacessíveis e novos pods são programados nos nós disponíveis e elegíveis.

Uma interrupção minoritária significa que uma porcentagem menor de nós em uma zona está desconectada do plano de controle do Kubernetes. Se nenhuma zona for definida no cluster, uma interrupção minoritária significa que a minoria de nós em todo o cluster está desconectada. Conforme declarado, a minoria é definida pela un`healthy-zone-threshold` configuração de node-lifecycle-

controller, que é 55% por padrão. Nesse cenário, se a desconexão da rede durar mais do que `default-unreachable-toleration-seconds` (5 minutos) e `node-monitor-grace-period` (40 segundos) e menos de 55% dos nós em uma zona estiverem inacessíveis, novos pods serão programados em nós saudáveis, enquanto os pods em nós inacessíveis serão marcados para remoção.

Os administradores de cluster verão novos pods criados em nós saudáveis, e os pods em nós desconectados serão exibidos como. `Terminating` Lembre-se de que, embora os pods nos nós desconectados tenham um `Terminating` status, eles não são totalmente removidos até que o nó se reconecte ao plano de controle do Kubernetes.

Cenário 5: reinicialização do nó durante interrupção da rede

Resultado esperado: os pods em nós inacessíveis não são iniciados até que os nós se reconectem ao plano de controle do Kubernetes. O failover do pod segue a lógica descrita nos Cenários 1—3, dependendo do número de nós inacessíveis.

A reinicialização do nó durante a interrupção da rede significa que outra falha (como ciclo de alimentação, out-of-memory evento ou outro problema) ocorreu em um nó ao mesmo tempo que a desconexão da rede. Os pods que estavam em execução nesse nó quando a desconexão da rede começou não são reiniciados automaticamente durante a desconexão se o kubelet também tiver sido reiniciado. O kubelet consulta o servidor da API Kubernetes durante a inicialização para saber quais pods ele deve executar. Se o kubelet não conseguir acessar o servidor da API devido a uma desconexão de rede, ele não poderá recuperar as informações necessárias para iniciar os pods.

Nesse cenário, ferramentas locais de solução de problemas, como a `crictl` CLI, não podem ser usadas para iniciar os pods manualmente como uma medida de “quebra de vidro”. O Kubernetes normalmente remove pods com falha e cria novos em vez de reiniciar os pods existentes (consulte #10213 no repositório containerd para obter detalhes). GitHub Os pods estáticos são o único objeto de carga de trabalho do Kubernetes que é controlado pelo kubelet e pode ser reiniciado durante esses cenários. No entanto, geralmente não é recomendável usar pods estáticos para implantações de aplicativos. Em vez disso, implante várias réplicas em diferentes hosts para garantir a disponibilidade do aplicativo no caso de várias falhas simultâneas, como uma falha de nó mais uma desconexão de rede entre seus nós e o plano de controle do Kubernetes.

Tráfego de rede de aplicativos por meio de desconexões de rede

Os tópicos desta página estão relacionados à rede de cluster do Kubernetes e ao tráfego do aplicativo durante as desconexões de rede entre os nós e o plano de controle do Kubernetes.

Cilium

O Cilium tem vários modos para gerenciamento de endereços IP (IPAM), encapsulamento, balanceamento de carga e roteamento de clusters. Os modos validados neste guia usaram Cluster Scope IPAM, VXLAN overlay, balanceamento de carga BGP e kube-proxy. O Cilium também foi usado sem balanceamento de carga BGP, substituindo-o pelo balanceamento de carga MetalLB L2.

A base da instalação do Cilium consiste no operador Cilium e nos agentes Cilium. O operador Cilium é executado como uma implantação e registra as definições de recursos personalizados do Cilium (CRDs), gerencia o IPAM e sincroniza objetos de cluster com o servidor da API Kubernetes, entre outros recursos. Os agentes Cilium são executados em cada nó como um DaemonSet e gerenciam os programas eBPF para controlar as regras de rede para cargas de trabalho em execução no cluster.

Geralmente, o roteamento no cluster configurado pelo Cilium permanece disponível e no local durante as desconexões da rede, o que pode ser confirmado observando os fluxos de tráfego no cluster e as regras da tabela IP (iptables) para a rede do pod.

```
ip route show table all | grep cilium
```

```
10.86.2.0/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16 mtu 1450
10.86.2.64/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16 mtu 1450
10.86.2.128/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16 mtu 1450
10.86.2.192/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16 mtu 1450
10.86.3.0/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16
10.86.3.16 dev cilium_host proto kernel scope link
...
```

No entanto, durante as desconexões da rede, o operador do Cilium e os agentes do Cilium são reiniciados devido ao acoplamento de suas verificações de saúde com a integridade da conexão com o servidor da API Kubernetes. Espera-se ver o seguinte nos registros do operador Cilium e dos agentes do Cilium durante as desconexões da rede. Durante as desconexões da rede, você pode usar ferramentas como a `crictl` CLI para observar as reinicializações desses componentes, incluindo seus 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
```

Se você estiver usando o recurso BGP Control Plane do Cilium para balanceamento de carga de aplicativos, a sessão BGP de seus pods e serviços poderá ficar inativa durante as desconexões da rede porque a funcionalidade do alto-falante BGP está integrada ao agente Cilium, e o agente Cilium será reiniciado continuamente quando desconectado do plano de controle do Kubernetes. Para obter mais informações, consulte o Guia de operação do plano de controle Cilium BGP na documentação do Cilium. Além disso, se você tiver uma falha simultânea durante uma desconexão de rede, como um ciclo de energia ou reinicialização da máquina, as rotas do Cilium não serão preservadas por meio dessas ações, embora as rotas sejam recriadas quando o nó se reconectar ao plano de controle do Kubernetes e o Cilium for reiniciado.

Calico

Em breve

Metal B

O MetalLB tem dois modos para balanceamento de carga: modo L2 e modo BGP. Consulte a documentação do MetalLB para obter detalhes sobre como esses modos de balanceamento de carga funcionam e suas limitações. A validação deste guia usou o MetalLB no modo L2, em que uma máquina no cluster assume a propriedade do Kubernetes Service e usa o ARP IPv4 para tornar os endereços IP do balanceador de carga acessíveis na rede local. Ao executar o MetalLB, há um controlador responsável pela atribuição de IP e alto-falantes executados em cada nó, responsáveis por serviços de publicidade com endereços IP atribuídos. O controlador MetalLB funciona como um Deployment e os alto-falantes MetalLB funcionam como um. DaemonSet Durante as desconexões da rede, o controlador e os alto-falantes MetalLB não conseguem monitorar o servidor da API Kubernetes em busca de recursos de cluster, mas continuam em execução. Mais importante ainda,

os Serviços que estão usando o MetalLB para conectividade externa permanecem disponíveis e acessíveis durante as desconexões da rede.

kube-proxy

Nos clusters EKS, o kube-proxy é executado como um DaemonSet em cada nó e é responsável por gerenciar as regras de rede para permitir a comunicação entre serviços e pods, traduzindo os endereços IP do serviço para os endereços IP dos pods subjacentes. As regras de tabelas IP (iptables) configuradas pelo kube-proxy são mantidas durante as desconexões da rede, o roteamento no cluster continua funcionando e os pods kube-proxy continuam em execução.

Você pode observar as regras do kube-proxy com os seguintes comandos iptables. O primeiro comando mostra que os pacotes que passam pela `PREROUTING` cadeia são direcionados para a `KUBE-SERVICES` cadeia.

```
iptables -t nat -L PREROUTING
```

```
Chain PREROUTING (policy ACCEPT)
target         prot opt source      destination
KUBE-SERVICES  all  --  anywhere    anywhere      /* kubernetes service portals */
```

Inspecionando a `KUBE-SERVICES` cadeia, podemos ver as regras para os vários serviços de cluster.

```
Chain KUBE-SERVICES (2 references)
target                     prot opt source      destination
KUBE-SVL-NZTS37XDTDNXGCKJ  tcp  --  anywhere    172.16.189.136  /* kube-system/hubble-
peer:peer-service cluster IP /
KUBE-SVC-2BINP2AXJOTI3HJ5  tcp  --  anywhere    172.16.62.72    / default/metallb-
webhook-service cluster IP /
KUBE-SVC-LRNEBRA3Z5YGJ4QC  tcp  --  anywhere    172.16.145.111  / default/redis-leader
 cluster IP /
KUBE-SVC-I7SKRZYQ7PWYV5X7  tcp  --  anywhere    172.16.142.147  / kube-system/eks-
extension-metrics-api:metrics-api cluster IP /
KUBE-SVC-JD5MR3NA4I4DYORP  tcp  --  anywhere    172.16.0.10     / kube-system/kube-
dns:metrics cluster IP /
KUBE-SVC-TCOU7JCQXEZGVUNU  udp  --  anywhere    172.16.0.10     / kube-system/kube-
dns:dns cluster IP /
KUBE-SVC-ERIFXISQEP7F7OF4  tcp  --  anywhere    172.16.0.10     / kube-system/kube-
dns:dns-tcp cluster IP /
KUBE-SVC-ENODL3HWJ5BZY56Q  tcp  --  anywhere    172.16.7.26     / default/frontend
 cluster IP /
```

```
KUBE-EXT-ENODL3HWJ5BZY56Q  tcp  --  anywhere    <LB-IP>    / default/frontend
 loadbalancer IP /
KUBE-SVC-NPX46M4PTMTKRN6Y  tcp  --  anywhere    172.16.0.1      / default/
kubernetes:https cluster IP /
KUBE-SVC-YU5RV2YQWHLZ5XPR  tcp  --  anywhere    172.16.228.76   / default/redis-
follower cluster IP /
KUBE-NODEPORTS             all  --  anywhere    anywhere        / kubernetes service
 nodeports; NOTE: this must be the last rule in this chain */
```

Inspecionando a cadeia do serviço de front-end do aplicativo, podemos ver os endereços IP do pod que sustentam o serviço.

```
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 */
```

As seguintes mensagens de log do kube-proxy são esperadas durante as desconexões da rede, enquanto ele tenta observar o servidor da API Kubernetes em busca de atualizações nos recursos de nós e endpoints.

```
"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

Por padrão, os pods em clusters EKS usam o endereço IP do cluster CoreDNS como servidor de nomes para consultas de DNS no cluster. Em clusters EKS, o CoreDNS é executado como

uma implantação em nós. Com os nós híbridos, os pods podem continuar se comunicando com o CoreDNS durante as desconexões da rede quando há réplicas do CoreDNS em execução local nos nós híbridos. Se você tiver um cluster EKS com nós na nuvem e nós híbridos em seu ambiente local, é recomendável ter pelo menos uma réplica do CoreDNS em cada ambiente. O CoreDNS continua atendendo consultas de DNS para registros que foram criados antes da desconexão da rede e continua executando a reconexão da rede para estabilidade estática.

As seguintes mensagens de log do CoreDNS são esperadas durante as desconexões de rede, à medida que ele tenta listar objetos do servidor da API Kubernetes.

```
Failed to watch *v1.Namespace: failed to list *v1.Namespace: Get "https://<k8s-cluster-
ip>:443/api/v1/namespaces?resourceVersion=2263964": dial tcp <k8s-cluster-ip>:443: i/o
 timeout
Failed to watch *v1.Service: failed to list *v1.Service: Get "https://<k8s-cluster-
ip>:443/api/v1/services?resourceVersion=2263966": dial tcp <k8s-cluster-ip>:443: i/o
 timeout
Failed to watch *v1.EndpointSlice: failed to list *v1.EndpointSlice: Get "https://<k8s-
cluster-ip>:443/apis/discovery.k8s.io/v1/endpointslices?resourceVersion=2263896": dial
 tcp <k8s-cluster-ip>: i/o timeout
```

Credenciais do host por meio de desconexões de rede

O EKS Hybrid Nodes é integrado às ativações híbridas do AWS Systems Manager (SSM) e ao AWS IAM Roles Anywhere para credenciais temporárias do IAM que são usadas para autenticar o nó com o plano de controle do EKS. Tanto o SSM quanto o IAM Roles Anywhere atualizam automaticamente as credenciais temporárias que eles gerenciam nos hosts locais. É recomendável usar um único provedor de credenciais nos nós híbridos do seu cluster: ativações híbridas de SSM ou IAM Roles Anywhere, mas não ambas.

Ativações híbridas SSM

As credenciais temporárias fornecidas pelo SSM são válidas por uma hora. Você não pode alterar a duração da validade da credencial ao usar o SSM como seu provedor de credenciais. As credenciais temporárias são alternadas automaticamente pelo SSM antes de expirarem, e a rotação não afeta o status de seus nós ou aplicativos. No entanto, quando há desconexões de rede entre o agente SSM e o endpoint regional do SSM, o SSM não consegue atualizar as credenciais e elas podem expirar.

O SSM usa o backoff exponencial para novas tentativas de atualização de credenciais se não conseguir se conectar aos endpoints regionais do SSM. Na versão do agente SSM `3.3.808.0` e posterior (lançada em agosto de 2024), o recuo exponencial é limitado a 30 minutos. Dependendo

da duração da desconexão da rede, pode levar até 30 minutos para o SSM atualizar as credenciais, e os nós híbridos não se reconectarão ao plano de controle do EKS até que as credenciais sejam atualizadas. Nesse cenário, você pode reiniciar o agente SSM para forçar uma atualização de credencial. Como efeito colateral do comportamento atual de atualização de credenciais do SSM, os nós podem se reconectar em momentos diferentes, dependendo de quando o agente do SSM em cada nó consegue atualizar suas credenciais. Por causa disso, você pode ver o failover do pod de nós que ainda não estão reconectados a nós que já estão reconectados.

Obtenha a versão do agente SSM. Você também pode verificar a seção Fleet Manager do console SSM:

```
# AL2023, RHEL
yum info amazon-ssm-agent
# Ubuntu
snap list amazon-ssm-agent
```

Reinicie o agente SSM:

```
# AL2023, RHEL
systemctl restart amazon-ssm-agent
# Ubuntu
systemctl restart snap.amazon-ssm-agent.amazon-ssm-agent
```

Veja os registros do agente SSM:

```
tail -f /var/log/amazon/ssm/amazon-ssm-agent.log
```

Mensagens de log esperadas durante desconexões de rede:

```
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

As credenciais temporárias provisionadas pelo IAM Roles Anywhere são válidas por uma hora por padrão. Você pode configurar a duração da validade da credencial com o IAM Roles Anywhere por meio do `durationSeconds`campo em seu perfil do IAM Roles Anywhere. A duração máxima da validade da credencial é de 12 horas. A `MaxSessionDuration`configuração na função do IAM do Hybrid Nodes deve ser maior do que a `durationSeconds` configuração no perfil do IAM Roles Anywhere.

Ao usar o IAM Roles Anywhere como provedor de credenciais para seus nós híbridos, a reconexão com o plano de controle do EKS após as desconexões da rede normalmente ocorre alguns segundos após a restauração da rede, porque o kubelet liga `aws_signing_helper credential-process` para obter credenciais sob demanda. Embora não esteja diretamente relacionado a nós híbridos ou desconexões de rede, você pode configurar notificações e alertas para a expiração do certificado ao usar o IAM Roles Anywhere. Para obter mais informações, consulte Personalizar as configurações de notificação no IAM Roles Anywhere.

Práticas recomendadas para executar AI/ML cargas de trabalho

> **Tip**
>
> Visite Get Hands on with EKS para saber mais sobre os próximos AI/ML eventos e workshops do Amazon EKS.

A implementação das melhores práticas ao executar AI/ML cargas de trabalho no EKS pode garantir que essas cargas de trabalho tenham desempenho, sejam econômicas, resilientes e tenham recursos adequados. As melhores práticas estão divididas nas seguintes seções gerais: computação, rede, armazenamento, observabilidade e desempenho.

Feedback

Este guia está sendo lançado GitHub para coletar feedback direto e sugestões da EKS/Kubernetes comunidade em geral. Se você tem uma prática recomendada que acha que devemos incluir no guia, registre um problema ou envie um PR no GitHub repositório. Nossa intenção é atualizar o guia periodicamente à medida que novos recursos são adicionados ao serviço ou quando uma nova prática recomendada evolui.

Computação e escalonamento automático

Otimização de recursos de GPU e gerenciamento de custos

Agende cargas de trabalho com requisitos de GPU usando rótulos conhecidos

Para AI/ML cargas de trabalho sensíveis a diferentes características da GPU (por exemplo, GPU, memória da GPU), recomendamos especificar os requisitos da GPU usando rótulos de agendamento conhecidos suportados pelos tipos de nós usados com o Karpenter e grupos de nós gerenciados. Deixar de defini-los pode resultar no agendamento de pods em instâncias com recursos de GPU inadequados, causando falhas ou degradação do desempenho. Recomendamos usar o NodeSelector ou o Node afinity para especificar em qual nó um pod deve ser executado e definir os recursos de computação (CPU, memória GPUs etc.) na seção de recursos do pod.

Exemplo

Por exemplo, usando o seletor de nó de nome da GPU ao usar o 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
```

Use o Kubernetes Device Plugin para expor GPUs

Para expor GPUs em nós, o driver da GPU NVIDIA deve estar instalado no sistema operacional do nó e no tempo de execução do contêiner configurado para permitir que o programador do Kubernetes atribua pods aos nós com disponibilidade. GPUs O processo de configuração do NVIDIA Kubernetes Device Plugin depende da AMI EKS Accelerated que você está usando:

- AMI acelerada por Bottlerocket: essa AMI inclui o driver de GPU NVIDIA e o plug-in de dispositivo NVIDIA Kubernetes está pré-instalado e pronto para uso, permitindo o suporte à GPU pronto para uso. Nenhuma configuração adicional é necessária para expor GPUs ao agendador do Kubernetes.
- AL2023 AMI acelerada: essa AMI inclui o driver de GPU NVIDIA, mas o plug-in de dispositivo NVIDIA Kubernetes não está pré-instalado. Você deve instalar e configurar o plug-in do dispositivo separadamente, normalmente por meio de um DaemonSet. Observe que, se você usar eksctl para criar seu cluster e especificar um tipo de instância de GPU (por exemplo,`g5.xlarge`) no seu ClusterConfig, `eksctl` selecionará automaticamente a AMI apropriada e instalará o plug-in de dispositivo NVIDIA Kubernetes. Para saber mais, consulte Suporte à GPU na documentação do eksctl.

Se você decidir usar o EKS Accelerated AMIs e o operador de GPU NVIDIA para gerenciar componentes como o plug-in de dispositivo NVIDIA Kubernetes, observe a desativação do

gerenciamento do driver de GPU NVIDIA e do kit de ferramentas de contêiner NVIDIA de acordo com a documentação pré-instalados de drivers de GPU NVIDIA e do kit de ferramentas de contêiner NVIDIA da NVIDIA.

Para verificar se o NVIDIA Device Plugin está ativo e GPUs exposto corretamente, execute:

```
kubectl describe node | grep nvidia.com/gpu
```

Esse comando verifica se o `nvidia.com/gpu` recurso está dentro da capacidade e dos recursos alocáveis do nó. Por exemplo, um nó com uma GPU deve aparecer`nvidia.com/gpu: 1`. Consulte o Guia de agendamento de GPU do Kubernetes para obter mais informações.

Use vários tipos de EC2 instância diferentes

Usar o maior número possível de tipos de EC2 instância é uma prática recomendada importante para escalabilidade no Amazon EKS, conforme descrito na the section called “Plano de dados” seção. Essa recomendação também se aplica a instâncias com hardware acelerado (por exemplo, GPUs). Se você criar um cluster que usa apenas um tipo de instância e tentar escalar o número de nós além da capacidade da região, poderá receber um erro de capacidade insuficiente (ICE), indicando que nenhuma instância está disponível. É importante entender as características exclusivas de suas AI/ML cargas de trabalho antes de diversificá-las arbitrariamente. Analise os tipos de instância disponíveis usando a ferramenta EC2 Instance Type Explorer para gerar uma lista de tipos de instância que correspondam aos seus requisitos computacionais específicos e evite limitar arbitrariamente o tipo de instâncias que podem ser usadas em seu cluster.

As instâncias de computação acelerada são oferecidas em diferentes modelos de compra para atender cargas de trabalho de curto, médio prazo e de estado estável. Para cargas de trabalho de curto prazo, flexíveis e tolerantes a falhas, nas quais você gostaria de evitar fazer uma reserva, consulte as instâncias spot. Blocos de capacidade, instâncias sob demanda e planos de economia permitem que você provisione instâncias de computação aceleradas para cargas de trabalho de médio e longo prazo. Para aumentar as chances de acessar com sucesso a capacidade necessária em sua opção de compra preferida, é recomendável usar uma lista diversificada de tipos de instância e zonas de disponibilidade. Como alternativa, se você encontrar ICEs um modelo de compra específico, tente novamente usar um modelo diferente.

Exemplo O exemplo a seguir mostra como permitir que um Karpenter NodePool provisione instâncias G e P maiores que as gerações 3 (por exemplo, p3). Para saber mais, consulte a seção Escalabilidade.

```
- 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 obter detalhes sobre o uso de instâncias spot para GPUs, consulte “Considere o uso de instâncias EC2 spot da Amazon GPUs com Karpenter” abaixo.

Considere o uso de instâncias EC2 spot da Amazon GPUs com o Karpenter

As Amazon EC2 Spot Instances permitem que você aproveite a EC2 capacidade não utilizada na nuvem da AWS e estão disponíveis com até 90% de desconto em comparação com os preços sob demanda. As instâncias EC2 spot da Amazon podem ser interrompidas com um aviso prévio de dois minutos quando EC2 a capacidade for recuperada. Para obter mais informações, consulte Instâncias spot no Guia EC2 do usuário da Amazon. O Amazon EC2 Spot pode ser uma ótima opção para cargas de trabalho tolerantes a falhas, sem estado e flexíveis (horário e tipo de instância). Para saber mais sobre quando usar instâncias spot, consulte Melhores práticas de instâncias EC2 spot. Você também pode usar Instâncias Spot para AI/ML cargas de trabalho se elas forem compatíveis com Spot.

Casos de uso

As cargas de trabalho amigáveis ao local podem ser big data, cargas de trabalho em contêineres, CI/CD, servidores web sem estado, computação de alto desempenho (HPC) e cargas de trabalho de renderização. As instâncias spot não são adequadas para cargas de trabalho inflexíveis, com estado, intolerantes a falhas ou fortemente acopladas entre os nós da instância (por exemplo, cargas de trabalho com processos paralelos que dependem muito uns dos outros para computação, exigindo comunicação constante entre nós, como aplicativos de computação de alto desempenho baseados em MPI, como dinâmica de fluidos computacional ou bancos de dados distribuídos com interdependências complexas). Aqui estão os casos de uso específicos que recomendamos (em nenhuma ordem específica):

- Inferência on-line em tempo real: use instâncias spot para obter um escalonamento econômico otimizado para suas cargas de trabalho de inferência em tempo real, desde que suas cargas de trabalho sejam amigáveis aos pontos. Em outras palavras, o tempo de inferência é inferior a dois minutos, o aplicativo é tolerante a falhas a interrupções e pode ser executado em diferentes tipos de instância. Garanta a alta disponibilidade por meio da diversidade de instâncias (por exemplo,

em vários tipos de instância e zonas de disponibilidade) ou reservas, enquanto implementa a tolerância a falhas em nível de aplicativo para lidar com possíveis interrupções pontuais.

- Ajuste de hiperparâmetros: use instâncias spot para executar trabalhos de ajuste exploratório de forma oportunista, pois interrupções podem ser toleradas sem perdas significativas, especialmente para experimentos de curta duração.
- Aumento de dados: use instâncias spot para realizar tarefas de pré-processamento e aumento de dados que podem ser reiniciadas a partir dos pontos de verificação se forem interrompidas, tornando-as ideais para a disponibilidade variável do Spot.
- Modelos de ajuste fino: use instâncias spot para fazer ajustes com mecanismos robustos de checkpoint para retomar a partir do último estado salvo, minimizando o impacto das interrupções da instância.
- Inferência em lote: use instâncias Spot para processar grandes lotes de solicitações de inferência off-line de uma non-real-time forma em que os trabalhos possam ser pausados e retomados, oferecendo o melhor alinhamento com a economia de custos do Spot e lidando com possíveis interrupções por meio de novas tentativas ou diversificação.
- Subconjuntos de treinamento oportunistas: use instâncias Spot para cargas de trabalho de treinamento marginais ou experimentais (por exemplo, modelos menores abaixo de 10 milhões de parâmetros), onde interrupções são aceitáveis e otimizações de eficiência, como diversificação entre tipos de instância ou regiões, podem ser aplicadas, embora não sejam recomendadas para treinamento em escala de produção devido a possíveis interrupções.

Considerações

Para usar instâncias spot para cargas de trabalho aceleradas no Amazon EKS, há várias considerações importantes (em nenhuma ordem específica):

- Use o Karpenter para gerenciar instâncias spot com a consolidação avançada ativada. Ao especificar karpenter.sh/capacity-type como “spot” em seu Karpenter, o Karpenter provisionará instâncias spot por padrão sem nenhuma NodePool configuração adicional. No entanto, para habilitar a Spot-to-Spot consolidação avançada, que substitui nós Spot subutilizados por alternativas Spot com preços mais baixos, você precisa habilitar o SpotToSpotConsolidation feature gate definindo --feature-gates SpotToSpotConsolidation =true nos argumentos do controlador Karpenter ou por meio da variável de ambiente FEATURE_GATES. O Karpenter usa a estratégia de price-capacity-optimizedalocação para provisionar instâncias. EC2 Com base nos NodePool requisitos e nas restrições dos pods, o Karpenter agrupa pods não programáveis e envia um conjunto diversificado de tipos de instância para a API do Amazon Fleet. EC2 Você

pode usar a ferramenta EC2 Instance Type Explorer para gerar uma lista de tipos de instância que correspondam aos seus requisitos específicos de computação.

- Garanta que as cargas de trabalho sejam independentes, tolerantes a falhas e flexíveis. As cargas de trabalho devem ser independentes, tolerantes a falhas e flexíveis em termos de tamanho. instance/GPU Isso permite uma retomada perfeita após interrupções no Spot, e a flexibilidade da instância permite que você permaneça potencialmente no Spot por mais tempo. Habilite o tratamento de interrupções Spot no Karpenter configurando o valor Settings.interruptionQueue Helm com o nome da fila do AWS SQS para capturar eventos de interrupção Spot. Por exemplo, ao instalar via Helm, use --set “settings.interruptionqueue=$ {CLUSTER_NAME}”. Para ver um exemplo, consulte o guia Introdução ao Karpenter. Quando o Karpenter percebe um evento de interrupção do Spot, ele automaticamente isola, contamina, drena e encerra o (s) nó (s) antes do evento de interrupção para maximizar o período de carência de rescisão dos pods. Ao mesmo tempo, o Karpenter iniciará imediatamente um novo nó para que ele esteja pronto o mais rápido possível.
- Evite restringir excessivamente a seleção do tipo de instância. Você deve evitar restringir ao máximo os tipos de instância. Ao não restringir os tipos de instância, há uma chance maior de adquirir capacidade spot em grande escala com uma frequência menor de interrupções de instâncias spot a um custo menor. Por exemplo, evite limitar-se a tipos específicos (por exemplo, g5.xlarge). Considere especificar um conjunto diversificado de categorias e gerações de instâncias usando chaves 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/MLa carga de trabalho requer um número específico ou limitado de aceleradores, mas é flexível entre regiões. Você pode usar o Spot Placement Score para identificar dinamicamente a região ideal para implantar sua carga de trabalho antes do lançamento.
- Amplie NodePool os requisitos para incluir um número maior de famílias de EC2 instâncias semelhantes. Cada pool de instâncias spot consiste em uma capacidade de EC2 instância não utilizada para um tipo específico de instância em uma zona de disponibilidade (AZ) específica. Quando o Karpenter tenta provisionar um novo nó, ele seleciona um tipo de instância que corresponde aos requisitos do NodePool. Se nenhum tipo de instância compatível tiver capacidade spot em qualquer AZ, o provisionamento falhará. Para evitar esse problema, permita instâncias mais amplas da série G (geração 4 ou superior) da NVIDIA em todos os tamanhos e zonas de disponibilidade (AZs), considerando as necessidades de hardware, como memória de GPU ou Ray Tracing. Como as instâncias podem ser de tipos diferentes, você precisa garantir que sua carga de trabalho possa ser executada em cada tipo e que o desempenho obtido atenda às suas necessidades.

- Aproveite todas as zonas de disponibilidade em uma região. A capacidade disponível varia de acordo com a zona de disponibilidade (AZ). Um tipo específico de instância pode estar indisponível em uma AZ, mas abundante em outra. Cada combinação exclusiva de um tipo de instância e uma zona de disponibilidade constitui um pool de capacidade spot separado. Ao solicitar capacidade AZs em toda a região dentro de seus NodePool requisitos de Karpenter, você está efetivamente pesquisando mais piscinas ao mesmo tempo. Isso maximiza o número de pools de capacidade spot e, portanto, aumenta a probabilidade de adquirir capacidade spot. Para fazer isso, em sua NodePool configuração, omita totalmente a chave topology.kubernetes.io/zone para permitir que o Karpenter selecione entre todas as disponíveis AZs na região ou liste explicitamente AZs usando o operador: In e forneça os valores (por exemplo, us-west-2a).
- Considere usar o Spot Placement Score (SPS) para ter visibilidade da probabilidade de acessar com sucesso a capacidade necessária usando instâncias spot. O Spot Placement Score (SPS) é uma ferramenta que fornece uma pontuação para ajudar você a avaliar a probabilidade de uma solicitação spot ser bem-sucedida. Ao usar o SPS, primeiro você especifica seus requisitos de computação para suas instâncias spot e, em seguida, a Amazon EC2 retorna as 10 principais regiões ou zonas de disponibilidade (AZs) onde sua solicitação spot provavelmente será bem-sucedida. Regiões e zonas de disponibilidade são pontuadas em uma escala de 1 a 10. Uma pontuação de 10 indica que sua solicitação Spot tem alta probabilidade, mas não tem garantia de sucesso. Uma pontuação de 1 indica que sua solicitação de spot tem muito pouca probabilidade de ter êxito. A mesma pontuação pode ser retornada para diferentes regiões ou zonas de disponibilidade. Para saber mais, consulte Orientações para criar um painel Spot Placement Score Tracker na AWS. Como a capacidade spot flutua o tempo todo, o SPS ajudará você a identificar qual combinação de tipos de instância e regiões funciona melhor para suas restrições de carga de trabalho (ou seja, flexibilidade, desempenho, tamanho etc.). AZs Se sua AI/ML carga de trabalho exigir um número específico ou limitado de aceleradores, mas for flexível entre regiões, você poderá usar a pontuação de posicionamento do Spot para identificar dinamicamente a região ideal para implantar sua carga de trabalho antes do lançamento. Para ajudá-lo a descobrir automaticamente a probabilidade de adquirir capacidade Spot, fornecemos uma orientação para criar um painel de controle do SPS Tracker. Essa solução monitora as pontuações do SPS ao longo do tempo usando uma configuração YAML para configurações diversificadas (por exemplo, incluindo requisitos de instância GPUs), armazena métricas e fornece painéis para comparar configurações. CloudWatch Defina painéis por carga de trabalho para avaliar as necessidades de vCPU, memória e GPU, garantindo configurações ideais para clusters EKS, incluindo a consideração do uso de outras regiões da AWS. Para saber mais, consulte Como funciona a pontuação de posicionamento do Spot.

- Lide com as interrupções e os testes do Spot com elegância. Para um pod com um período de encerramento superior a dois minutos, o nó antigo será interrompido antes que esses pods sejam reprogramados, o que pode afetar a disponibilidade da carga de trabalho. Considere o aviso de interrupção Spot de dois minutos ao projetar seus aplicativos, implemente o ponto de verificação em aplicativos de longa execução (por exemplo, salvando o progresso em armazenamento persistente como o Amazon S3) para retomar após interrupções, estenda os terminationGracePeriod segundos (o padrão é 30 segundos) nas especificações do pod para permitir mais tempo para um desligamento normal e lidar com interrupções usando os sinais and/or SIGTERM do ciclo de vida PreStop em seu aplicativo para atividades de desligamento regulares, como limpeza, estado salvamento e fechamento da conexão. Para cargas de trabalho em tempo real, em que o tempo de escalabilidade é importante e as cargas de trabalho demoram mais de dois minutos para que o aplicativo esteja pronto para atender ao tráfego, considere otimizar a inicialização do contêiner e os tempos de carregamento do modelo de ML por meio da análise e das melhores práticas. the section called “Armazenamento” the section called “desempenho” Para testar um nó substituto, use o AWS Fault Injection Service (FIS) para simular interrupções pontuais.

Além dessas melhores práticas básicas do Spot, leve esses fatores em consideração ao gerenciar cargas de trabalho de GPU no Amazon EKS. Diferentemente das cargas de trabalho baseadas em CPU, as cargas de trabalho de GPU são particularmente sensíveis aos detalhes do hardware, como recursos da GPU e memória de GPU disponível. As cargas de trabalho da GPU podem ser restringidas pelos tipos de instância que podem usar, com menos opções disponíveis em comparação com. CPUs Como primeira etapa, avalie se sua carga de trabalho é flexível em termos de instância. Se você não sabe quantos tipos de instância sua carga de trabalho pode usar, teste-os individualmente para garantir a compatibilidade e a funcionalidade. Identifique o quão flexível você pode ser para diversificar o máximo possível, ao mesmo tempo em que confirma que a diversificação mantém a carga de trabalho funcionando e compreende quaisquer impactos no desempenho (por exemplo, na produtividade ou no tempo de conclusão). Como parte da diversificação de suas cargas de trabalho, considere o seguinte:

- Analise a compatibilidade do CUDA e da estrutura. Suas cargas de trabalho de GPU podem ser otimizadas para hardware e tipos de GPU específicos (por exemplo, V100 em p3 versus A100 em p4) ou escritas para versões específicas de CUDA para bibliotecas como, portanto TensorFlow, certifique-se de revisar a compatibilidade de suas cargas de trabalho. Essa compatibilidade é crucial para evitar erros de tempo de execução, falhas, falhas na aceleração da GPU (por exemplo, versões CUDA incompatíveis com estruturas como PyTorch ou TensorFlow podem

impedir a execução) ou a capacidade de aproveitar recursos de hardware como /precision. FP16 INT8

- Memória GPU. Certifique-se de avaliar os requisitos de memória dos seus modelos e definir o perfil do uso de memória do seu modelo durante o tempo de execução usando ferramentas como o DCGM Exporter e definir a memória mínima de GPU necessária para o tipo de instância em rótulos conhecidos como karpenter.k8s.aws/. instance-gpu-memory A VRAM da GPU varia entre os tipos de instância (por exemplo, NVIDIA T4 tem 16 GB, A10G tem 24 GB, V100 tem 16 a 32 GB), e os modelos de ML (por exemplo, modelos de linguagem grande) podem exceder a memória disponível, causando erros (OOM) ou falhas. out-of-memory Para instâncias spot no EKS, isso pode limitar a diversificação. Por exemplo, você não pode incluir tipos de VRAM mais baixos se seu modelo não se adequar, o que pode limitar o acesso aos pools de capacidade e aumentar o risco de interrupção. Observe que, para inferência de GPU única e nó único (por exemplo, vários pods programados no mesmo nó para utilizar seus recursos de GPU), isso pode limitar a diversificação, pois você só pode incluir tipos de instância com VRAM suficiente na configuração do Spot.
- Precisão e desempenho de ponto flutuante. Nem todas as arquiteturas de GPU da Nvidia têm a mesma precisão de ponto flutuante (por exemplo, FP16/). INT8 Avalie o desempenho dos tipos principais (CUDA/Tensor/RT) e a precisão de ponto flutuante necessários para suas cargas de trabalho. Usar uma GPU mais barata e com menor desempenho não significa que seja melhor, então considere avaliar o desempenho em termos de trabalho concluído dentro de um prazo específico para entender o impacto da diversificação.

Cenário: diversificação para cargas de trabalho de inferência em tempo real

Para uma carga de trabalho de inferência on-line em tempo real em instâncias spot, você pode configurar um Karpenter NodePool para diversificar entre famílias e gerações de instâncias de GPU compatíveis. Essa abordagem garante alta disponibilidade ao usar vários pools Spot e, ao mesmo tempo, manter o desempenho por meio de restrições nos recursos, memória e arquitetura da GPU. Ele suporta o uso de alternativas quando a capacidade da instância é restrita, minimizando interrupções e otimizando a latência de inferência. Este exemplo NodePool indica: use instâncias das séries g e p maiores que 3, que tenham mais de 20 GB de memória de GPU.

Exemplo

```
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 pontos de verificação para trabalhos de treinamento de longa duração

O checkpoint é uma técnica de tolerância a falhas que envolve salvar periodicamente o estado de um processo, permitindo que ele seja retomado a partir do último ponto salvo em caso de interrupções. No aprendizado de máquina, é comumente associado ao treinamento, em que trabalhos de longa

duração podem economizar pesos do modelo e estados do otimizador para retomar o treinamento após falhas, como problemas de hardware ou interrupções de instâncias spot.

Você usa pontos de verificação para salvar o estado dos modelos de aprendizado de máquina (ML) durante o treinamento. Os pontos de verificação são snapshots do modelo e podem ser configurados pelas funções de retorno de chamada dos frameworks de ML. Você pode usar pontos de verificação salvos para reiniciar um trabalho de treinamento a partir do ponto de verificação salvo pela última vez. Usando pontos de verificação, você salva os instantâneos do modelo em treinamento devido a uma interrupção inesperada no trabalho ou na instância de treinamento. Isso permite que você retome o treinamento do modelo futuramente a partir de um ponto de verificação. Além de implementar um sistema de resiliência de nós, recomendamos a implementação de pontos de verificação para mitigar o impacto das interrupções, incluindo aquelas causadas por falhas de hardware ou interrupções da Amazon EC2 Spot Instance.

Sem pontos de verificação, as interrupções podem resultar em perda de tempo de computação e perda de progresso, o que é caro para trabalhos de treinamento de longa duração. O ponto de verificação permite que as tarefas salvem seu estado periodicamente (por exemplo, pesos do modelo e estados do otimizador) e sejam retomadas a partir do último ponto de verificação (último lote processado) após uma interrupção. Para implementar o ponto de verificação, projete seu aplicativo para processar dados em grandes lotes e salvar resultados intermediários em armazenamento persistente, como um bucket do Amazon S3 por meio do Mountpoint for Amazon S3 CSI Driver enquanto o trabalho de treinamento progride.

Casos de uso

A verificação é particularmente benéfica em cenários específicos para equilibrar a tolerância a falhas com a sobrecarga de desempenho. Considere usar o ponto de verificação nos seguintes casos:

- A duração do trabalho excede algumas horas: para trabalhos de treinamento de longa duração (por exemplo, mais de 1 a 2 horas para modelos pequenos ou days/weeks para grandes modelos básicos com bilhões de parâmetros), em que a perda de progresso devido a interrupções é cara. Trabalhos mais curtos podem não justificar a I/O sobrecarga.
- Para instâncias spot ou falhas de hardware: em ambientes propensos a interrupções, como EC2 Spot (aviso prévio de 2 minutos) ou falhas de hardware (por exemplo, erros de memória de GPU), o ponto de verificação permite uma retomada rápida, tornando o Spot viável para redução de custos em cargas de trabalho tolerantes a falhas.
- Treinamento distribuído em grande escala: para configurações com hundreds/thousands aceleradores (por exemplo, >100 GPUs), onde o tempo médio entre falhas diminui linearmente

com a escala. Use para model/data paralelismo para lidar com o acesso simultâneo ao ponto de verificação e evitar reinicializações completas.

- Modelos de grande escala com alta demanda de recursos: no treinamento LLM em escala de petabytes, em que as falhas são inevitáveis devido ao tamanho do cluster; abordagens em camadas (locais rápidas a cada 5 a 30 minutos para transientes, duráveis por hora para falhas graves) otimizam o tempo de recuperação versus a eficiência.

Use blocos de capacidade de ML para garantir a capacidade das instâncias P e Trainium

Os blocos de capacidade para ML permitem que você reserve instâncias de GPU muito procuradas, especificamente instâncias P (por exemplo, p6-b200, p5, p5e, p5en, p4d, p4de) e instâncias Trainium (por exemplo, trn1, trn2), para começar quase imediatamente ou em uma data futura para suportar suas cargas de trabalho de aprendizado de máquina (ML) de curta duração. Essas reservas são ideais para garantir a capacidade para tarefas de computação intensiva, como treinamento e ajuste fino de modelos. EC2 O preço do Capacity Blocks consiste em uma taxa de reserva e uma taxa de sistema operacional. Para saber mais sobre os preços, consulte os preços EC2 do Capacity Blocks for ML.

GPUs Para reservar AI/ML cargas de trabalho no Amazon EKS para garantir a capacidade previsível, recomendamos usar os blocos de capacidade de ML para reservas de capacidade de curto prazo ou sob demanda (ODCRs) para garantia de capacidade de uso geral.

- ODCRs permitem que você reserve a capacidade da EC2 instância (por exemplo, instâncias de GPU como g5 ou p5) em uma zona de disponibilidade específica por um período, garantindo disponibilidade, mesmo durante alta demanda. ODCRs não tem compromisso de longo prazo, mas você paga a taxa sob demanda pela capacidade reservada, seja ela usada ou ociosa. No EKS, ODCRs são suportados por tipos de nós como Karpenter e grupos de nós gerenciados. Para priorizar ODCRs no Karpenter, configure o NodeClass para usar o campo. `capacityReservationSelectorTerms` Consulte a documentação do Karpenter NodePools .
- Os blocos de capacidade são um mecanismo de reserva especializado para instâncias de GPU (por exemplo, p5, p4d) ou Trainium (trn1, trn2), projetado para cargas de trabalho de ML de curto prazo, como treinamento de modelos, ajuste fino ou experimentação. Você reserva capacidade por um período definido (normalmente de 24 horas a 182 dias) a partir de uma data futura, pagando somente pelo tempo reservado. Eles são pré-pagos, exigem planejamento prévio para as necessidades de capacidade e não oferecem suporte ao escalonamento automático, mas são alocados para redes de baixa latência. EC2 UltraClusters Eles cobram apenas pelo período

reservado. Para saber mais, consulte Encontre e compre blocos de capacidade ou comece configurando grupos de nós gerenciados com blocos de capacidade usando as instruções em Criar um grupo de nós gerenciados com blocos de capacidade para ML.

Reserve capacidade por meio do AWS Management Console e configure seus nós para usar blocos de capacidade de ML. Planeje reservas com base em cronogramas de carga de trabalho e teste em um cluster de preparação. Consulte a documentação de blocos de capacidade para obter mais informações.

Considere reservas de capacidade sob demanda, Amazon EC2 Spot ou sob demanda (ODCRs) para instâncias G Amazon EC2

Para EC2 instâncias G Amazon, considere as diferentes opções de compra de instâncias sob demanda, instâncias EC2 spot da Amazon e reservas de capacidade sob demanda. ODCRspermitem que você reserve a capacidade da EC2 instância em uma zona de disponibilidade específica por um determinado período, garantindo a disponibilidade mesmo durante alta demanda. Ao contrário do ML Capacity Blocks, que só estão disponíveis para instâncias P e Trainium, ODCRs podem ser usados para uma variedade maior de tipos de instância, incluindo instâncias G, tornando-os adequados para cargas de trabalho que exigem diferentes recursos de GPU, como inferência ou gráficos. Ao usar as Instâncias EC2 Spot da Amazon, ser capaz de diversificar em diferentes tipos, tamanhos e zonas de disponibilidade de instâncias é fundamental para poder permanecer no Spot por mais tempo.

ODCRs não tem compromisso de longo prazo, mas você paga a taxa sob demanda pela capacidade reservada, seja ela usada ou ociosa. ODCRs podem ser criados para uso imediato ou programados para uma data futura, oferecendo flexibilidade no planejamento da capacidade. No Amazon EKS, ODCRs são compatíveis com tipos de nós como Karpenter e grupos de nós gerenciados. Para priorizar ODCRs no Karpenter, configure o NodeClass para usar o campo. `capacityReservationSelectorTerms` Consulte a documentação do Karpenter NodePools . Para obter mais informações sobre criação ODCRs, incluindo comandos de CLI, consulte Introdução à reserva de capacidade sob demanda.

Considere outros tipos e tamanhos de instâncias aceleradas

Selecionar a instância e o tamanho acelerados apropriados é essencial para otimizar o desempenho e o custo em suas cargas de trabalho de ML no Amazon EKS. Por exemplo, diferentes famílias de instâncias de GPU têm desempenho e recursos diferentes, como memória de GPU. Para ajudar você a escolher a opção mais econômica, analise as instâncias de GPU disponíveis na página Tipos

de EC2 instância em Computação acelerada. Avalie vários tipos e tamanhos de instâncias para encontrar a melhor opção para seus requisitos específicos de carga de trabalho. Considere fatores como o número GPUs, a memória e o desempenho da rede. Ao selecionar cuidadosamente o tipo e o tamanho corretos da instância de GPU, você pode obter melhor utilização de recursos e eficiência de custos em seus clusters EKS.

Se você usar uma instância de GPU em um nó EKS, ela terá o `nvidia-device-plugin-daemonset` pod no `kube-system` namespace por padrão. Para ter uma ideia rápida de se você está utilizando totalmente a (s) GPU (s) em sua instância, você pode usar nvidia-smi conforme mostrado aqui:

```
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
```

- Se `utilization.memory` estiver próximo de 100%, seus códigos provavelmente estão limitados à memória. Isso significa que a GPU (memória) está totalmente utilizada, mas pode sugerir que uma otimização adicional do desempenho deva ser investigada.
- Se `utilization.gpu` estiver próximo de 100%, isso não significa necessariamente que a GPU esteja totalmente utilizada. Uma métrica melhor a ser observada é a proporção de `power.draw` para`power.limit`. Se essa proporção for de 100% ou mais, seus códigos estão utilizando totalmente a capacidade computacional da GPU.
- A `-l 5` bandeira diz para gerar as métricas a cada 5 segundos. No caso de um único tipo de instância de GPU, o sinalizador de consulta de índice não é necessário.

Para saber mais, consulte instâncias de GPU na documentação da AWS.

Otimize a alocação de recursos de GPU com divisão de tempo, MIG e alocação fracionária de GPU

Limites estáticos de recursos no Kubernetes (por exemplo, CPU, memória, contagem de GPU) podem levar ao provisionamento excessivo ou à subutilização, especialmente para cargas de trabalho dinâmicas, como inferência. AI/ML Selecionar a GPU certa é importante. Para cargas de trabalho de baixo volume ou picos, o corte de tempo permite que várias cargas de trabalho

compartilhem uma única GPU compartilhando seus recursos de computação, potencialmente melhorando a eficiência e reduzindo o desperdício. O compartilhamento de GPU pode ser obtido por meio de diferentes opções:

- Aproveite os seletores de nós/afinidade de nós para influenciar o agendamento: garanta que os nós provisionados e os pods estejam programados de acordo com a carga de trabalho apropriada GPUs (por exemplo,) `karpenter.k8s.aws/instance-gpu-name: "a100"`
- Time-Slicing: programa cargas de trabalho para compartilhar os recursos computacionais de uma GPU ao longo do tempo, permitindo execução simultânea sem particionamento físico. Isso é ideal para cargas de trabalho com demandas de computação variáveis, mas pode não ter isolamento de memória.
- GPU de várias instâncias (MIG): A MIG permite que uma única GPU NVIDIA seja particionada em várias instâncias isoladas e é compatível com NVIDIA Ampere (por exemplo, GPU A100), NVIDIA Hopper (por exemplo, GPU H100) e NVIDIA Blackwell (por exemplo, Blackwell). GPUs GPUs Cada instância MIG recebe recursos dedicados de computação e memória, permitindo o compartilhamento de recursos em ambientes multilocatários ou cargas de trabalho que exigem garantias de recursos, o que permite otimizar a utilização dos recursos da GPU, incluindo cenários como atender a vários modelos com tamanhos de lote diferentes por meio de divisão de tempo.
- Alocação fracionária de GPU: usa agendamento baseado em software para alocar partes da computação ou da memória de uma GPU às cargas de trabalho, oferecendo flexibilidade para cargas de trabalho dinâmicas. O NVIDIA KAI Scheduler, parte da plataforma Run:AI, permite isso ao permitir que os pods solicitem recursos fracionários de GPU.

Para habilitar esses recursos no EKS, você pode implantar o NVIDIA Device Plugin, que expõe GPUs como recursos programáveis e oferece suporte ao corte de tempo e ao MIG. Para saber mais, consulte Time-Slicing GPUs no Kubernetes e Compartilhamento de GPU no Amazon EKS com o time-slicing e as instâncias aceleradas da NVIDIA. EC2

Exemplo

Por exemplo, para habilitar o corte de tempo com o NVIDIA Device Plugin:

```
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
```

Exemplo

Por exemplo, para usar o KAI Scheduler para alocação fracionária de GPU, implante-o junto com o operador de GPU NVIDIA e especifique recursos de GPU fracionários na especificação do pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: fractional-gpu-pod-example
  annotations:
    gpu-fraction: "0.5"  # Annotation for 50% GPU
  labels:
    runai/queue: "default"  # Required queue assignment
spec:
  containers:
  - name: ml-workload
    image: nvcr.io/nvidia/pytorch:25.04-py3
    resources:
      limits:
        nvidia.com/gpu: 1
  nodeSelector:
    nvidia.com/gpu: "true"
  schedulerName: kai-scheduler
```

Resiliência de nós e gerenciamento de tarefas de treinamento

Implemente verificações de integridade do Node com recuperação automatizada

Para trabalhos de treinamento distribuídos no Amazon EKS que exigem comunicação frequente entre nós, como treinamento em modelos de várias GPUs em vários nós, problemas de hardware, como falhas na GPU ou no EFA, podem causar interrupções nos trabalhos de treinamento. Essas interrupções podem levar à perda do progresso do treinamento e ao aumento dos custos, especialmente para AI/ML cargas de trabalho de longa duração que dependem de hardware estável.

Para ajudar a aumentar a resiliência contra falhas de hardware, como falhas de GPU em clusters EKS que executam cargas de trabalho de GPU, recomendamos usar o EKS Node Monitoring Agent with Auto Repair ou a Amazon. SageMaker HyperPod Enquanto o EKS Node Monitoring Agent with Auto Repair fornece recursos como monitoramento de integridade de nós e reparo automático usando mecanismos Kubernetes padrão, SageMaker HyperPod oferece resiliência direcionada e recursos adicionais projetados especificamente para treinamento de ML em grande escala, como verificações de integridade profundas e retomada automática de trabalhos.

- O EKS Node Monitoring Agent com Node Auto Repair monitora continuamente a integridade do nó lendo registros e aplicando NodeConditions, incluindo condições padrão `Ready` e condições específicas do hardware acelerado para identificar problemas como falhas na GPU ou na rede. Quando um nó é considerado insalubre, o Node Auto Repair o isola e o substitui por um novo nó. O reagendamento de pods e o reinício dos trabalhos dependem dos mecanismos padrão do Kubernetes e da política de reinicialização do trabalho.
- O agente de verificação SageMaker HyperPodprofunda de integridade e monitoramento de integridade monitora continuamente o status de integridade das instâncias baseadas em GPU e Trainium. Ele é personalizado para AI/ML cargas de trabalho, usando rótulos (por exemplo, node-health-status) para gerenciar a integridade do nó. Quando um nó é considerado não íntegro, HyperPod aciona a substituição automática do hardware defeituoso, como. GPUs Por padrão, ele detecta falhas relacionadas à rede no EFA por meio de suas verificações básicas de integridade e oferece suporte à retomada automática de trabalhos de treinamento interrompidos, permitindo que os trabalhos continuem a partir do último ponto de verificação, minimizando as interrupções em tarefas de ML em grande escala.

Tanto para o EKS Node Monitoring Agent com Auto Repair quanto para SageMaker HyperPod clusters usando EFA, para monitorar métricas específicas do EFA, como erros de acesso remoto direto à memória (RDMA) e quedas de pacotes, certifique-se de que o driver AWS EFA esteja instalado. Além disso, recomendamos implantar o complemento CloudWatch Observability ou usar ferramentas como o DCGM Exporter com Prometheus e Grafana para monitorar o EFA, a GPU e, para obter, métricas específicas relacionadas aos seus recursos. SageMaker HyperPod

Desative a consolidação do Karpenter para cargas de trabalho sensíveis à interrupção

Para cargas de trabalho sensíveis a interrupções, como processamento, tarefas de AI/ML previsão em grande escala ou treinamento, recomendamos ajustar as políticas de consolidação do Karpenter para evitar interrupções durante a execução do trabalho. O recurso de consolidação do Karpenter otimiza automaticamente os custos do cluster ao encerrar nós subutilizados ou substituí-los por

alternativas mais baratas. No entanto, mesmo quando uma carga de trabalho utiliza totalmente uma GPU, o Karpenter pode consolidar nós se identificar um tipo de instância com preço mais baixo e do tamanho certo que atenda aos requisitos do pod, causando interrupções no trabalho.

A política de `WhenEmptyOrUnderutilized` consolidação pode encerrar os nós prematuramente, levando a tempos de execução mais longos. Por exemplo, interrupções podem atrasar a retomada do trabalho devido ao reagendamento do pod e ao recarregamento de dados, o que pode ser caro para trabalhos de inferência em lote de longa duração. Para mitigar isso, você pode definir o `WhenEmpty` e `consolidationPolicy` configurar uma `consolidateAfter` duração, como 1 hora, para reter os nós durante picos de carga de trabalho. Por exemplo:

```
disruption:
  consolidationPolicy: WhenEmpty
  consolidateAfter: 60m
```

Essa abordagem melhora a latência de inicialização do pod para cargas de trabalho de inferência em lote com picos e outros trabalhos sensíveis à interrupção, como processamento de dados de inferência on-line em tempo real ou treinamento de modelos, em que o custo da interrupção supera a economia de custos de computação. O Karpenter NodePool Disruption Budgets é outro recurso para gerenciar interrupções no Karpenter. Com os orçamentos, você pode garantir que não mais do que um certo número de nós seja interrompido no NodePool momento escolhido. Você também pode usar orçamentos de interrupção para evitar que todos os nós sejam interrompidos em um determinado horário (por exemplo, horários de pico). Para saber mais, consulte a documentação do Karpenter Consolidation.

Use ttlSecondsAfter Finished para limpar automaticamente os trabalhos do Kubernetes

Recomendamos a configuração de `ttlSecondsAfterFinished` trabalhos do Kubernetes no Amazon EKS para excluir automaticamente os objetos de trabalho concluídos. Objetos de trabalho persistentes consomem recursos do cluster, como memória do servidor de API, e complicam o monitoramento ao desordenar painéis (por exemplo, Grafana, Amazon). CloudWatch Por exemplo, definir um TTL de 1 hora garante que os trabalhos sejam removidos logo após a conclusão, mantendo seu cluster organizado. Para obter mais detalhes, consulte Limpeza automática de trabalhos concluídos.

Configure a preempção de trabalhos de baixa prioridade para trabalhos e cargas de trabalho de maior prioridade

Para cargas de AI/ML trabalho de prioridade mista no Amazon EKS, você pode configurar a preempção de tarefas de baixa prioridade para garantir que tarefas de maior prioridade (por exemplo, inferência em tempo real) recebam recursos imediatamente. Sem preempção, cargas de trabalho de baixa prioridade, como processos em lote (por exemplo, inferência em lote, processamento de dados), serviços que não sejam em lote (por exemplo, tarefas em segundo plano, tarefas cron) ou tarefas com uso intensivo de CPU/memória (por exemplo, serviços web) podem atrasar pods críticos ocupando nós. A preempção permite que o Kubernetes expulse pods de baixa prioridade quando pods de alta prioridade precisam de recursos, garantindo a alocação eficiente de recursos em nós com, ou memória. GPUs CPUs Recomendamos usar o Kubernetes `PriorityClass` para atribuir prioridades e controlar o comportamento de `PodDisruptionBudget` despejo.

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: low-priority
value: 100
---
spec:
  priorityClassName: low-priority
```

Consulte a documentação de prioridade e preempção do Kubernetes para obter mais informações.

Dimensionamento e desempenho de aplicativos

Personalize a capacidade de computação para cargas de trabalho de ML com Karpenter ou Static Nodes

Para garantir uma capacidade computacional econômica e responsiva para fluxos de trabalho de aprendizado de máquina (ML) no Amazon EKS, recomendamos adaptar sua estratégia de provisionamento de nós às características da sua carga de trabalho e aos compromissos de custo. Abaixo estão duas abordagens a serem consideradas: just-in-time escalabilidade com Karpenter e grupos de nós estáticos para capacidade reservada.

- Just-in-time escaladores de plano de dados como o Karpenter: para fluxos de trabalho dinâmicos de ML com demandas de computação variáveis (por exemplo, inferência baseada em GPU

seguida por plotagem baseada em CPU), recomendamos usar escaladores de plano de dados como o Karpenter. just-in-time

- Use grupos de nós estáticos para cargas de trabalho previsíveis: Para cargas de trabalho de ML previsíveis e estáveis ou ao usar instâncias reservadas, os grupos de nós gerenciados pelo EKS podem ajudar a garantir que a capacidade reservada seja totalmente provisionada e utilizada, maximizando a economia. Essa abordagem é ideal para tipos de instância específicos comprometidos por meio de RIs ou ODCRs.

Exemplo

Este é um exemplo de um Karpenter diversificado NodePoolque permite o lançamento de instâncias da g Amazon em que a geração de EC2 instâncias é maior que três.

```
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
```

Exemplo

Exemplo de uso de grupos de nós estáticos para uma carga de trabalho de treinamento:

```
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
```

Use imperfeições e tolerâncias para evitar que cargas de trabalho não aceleradas sejam programadas em instâncias aceleradas

O agendamento de cargas de trabalho não aceleradas em recursos de GPU não é eficiente em termos de computação. Recomendamos o uso de manchas e tolerância para garantir que os pods de cargas de trabalho não aceleradas não sejam programados em nós inadequados. Consulte a documentação do Kubernetes para obter mais informações.

Escala com base no desempenho do modelo

Para cargas de trabalho de inferência, recomendamos usar o escalonamento automático orientado a eventos (KEDA) do Kubernetes para escalar com base nas métricas de desempenho do modelo, como solicitações de inferência ou taxa de transferência de tokens, com períodos de espera

apropriados. As políticas de escalabilidade estática podem provisionar recursos em excesso ou insuficiente, afetando o custo e a latência. Saiba mais na documentação do KEDA.

Alocação dinâmica de recursos para gerenciamento avançado de GPU

A alocação dinâmica de recursos (DRA) representa um avanço fundamental no gerenciamento de recursos da GPU Kubernetes. O DRA vai além das limitações tradicionais de plug-ins de dispositivos para permitir compartilhamento sofisticado de GPU, reconhecimento de topologia e coordenação de recursos entre nós. Disponível na versão 1.33 do Amazon EKS, o DRA aborda desafios críticos em AI/ML cargas de trabalho fornecendo o seguinte:

- Alocação refinada de GPU
- Mecanismos avançados de compartilhamento, como serviço multiprocesso (MPS) e GPU de múltiplas instâncias (MIG)
- Support para arquiteturas de hardware de próxima geração, incluindo NVIDIA 00 GB2 UltraServers

A alocação tradicional de GPU é tratada GPUs como recursos inteiros opacos, criando uma subutilização significativa (geralmente de 30 a 40% em clusters de produção). Isso ocorre porque as cargas de trabalho recebem acesso exclusivo à totalidade, GPUs mesmo quando exigem apenas recursos fracionários. O DRA transforma esse modelo introduzindo uma alocação estruturada e declarativa que fornece ao programador do Kubernetes visibilidade completa das características do hardware e dos requisitos de carga de trabalho. Isso permite decisões inteligentes de posicionamento e compartilhamento eficiente de recursos.

Vantagens de usar o DRA em vez do plug-in de dispositivo NVIDIA

O plug-in de dispositivo NVIDIA (a partir da versão`0.12.0`) oferece suporte a mecanismos de compartilhamento de GPU, incluindo corte de tempo, MPS e MIG. No entanto, existem limitações arquitetônicas que o DRA aborda.

Limitações do plug-in do dispositivo NVIDIA

- Configuração estática: as configurações de compartilhamento de GPU (réplicas de divisão de tempo e configurações de MPS) exigem pré-configuração em todo o cluster. `ConfigMaps` Isso dificulta o fornecimento de diferentes estratégias de compartilhamento para diferentes cargas de trabalho.
- Seleção granular limitada: embora o plug-in do dispositivo exponha as características da GPU por meio de rótulos de nós, as cargas de trabalho não podem solicitar dinamicamente configurações

específicas da GPU (tamanho da memória e recursos de computação) como parte da decisão de agendamento.

- Sem coordenação de recursos entre nós: não é possível gerenciar recursos de GPU distribuídos em vários nós ou expressar requisitos complexos de topologia, como NVLink domínios para sistemas como o NVIDIA 00. GB2
- Restrições do agendador: o programador do Kubernetes trata os recursos da GPU como números inteiros opacos, limitando sua capacidade de tomar decisões com base na topologia ou lidar com dependências complexas de recursos.
- Complexidade da configuração: a configuração de diferentes estratégias de compartilhamento requer uma rotulagem cuidadosa `ConfigMaps` e múltipla de nós, criando complexidade operacional.

Soluções com DRA

- Seleção dinâmica de recursos: o DRA permite que as cargas de trabalho especifiquem requisitos detalhados (memória da GPU, versões do driver e atributos específicos) no momento da solicitação. `resourceclaims` Isso permite uma correspondência mais flexível de recursos.
- Reconhecimento da topologia: por meio de parâmetros estruturados e seletores de dispositivos, o DRA lida com requisitos complexos, como comunicação entre nós de GPU e interconexões coerentes com a memória.
- Gerenciamento de recursos entre nós: `computeDomains` permita a coordenação de recursos de GPU distribuídos em vários nós, o que é essencial para sistemas como GB2 00 com canais IMEX.
- Configuração específica da carga de trabalho: cada uma `ResourceClaim` especifica diferentes estratégias e configurações de compartilhamento, permitindo um controle refinado por carga de trabalho, em vez de configurações em todo o cluster.
- Integração aprimorada do agendador: o DRA fornece ao programador informações detalhadas do dispositivo e permite decisões de posicionamento mais inteligentes com base na topologia do hardware e nas características dos recursos.

Importante: O DRA não substitui totalmente o plug-in do dispositivo NVIDIA. O driver NVIDIA DRA funciona junto com o plug-in do dispositivo para fornecer recursos aprimorados. O plug-in do dispositivo continua lidando com a descoberta e o gerenciamento básicos da GPU, enquanto o DRA adiciona recursos avançados de alocação e agendamento.

Instâncias suportadas pelo DRA e seus recursos

O suporte ao DRA varia de acordo com a família de EC2 instâncias da Amazon e a arquitetura da GPU, conforme mostrado na tabela a seguir.

Família de instâncias	Tipo de GPU	Corte no tempo	Suporte MIG	Suporte MPS	Suporte IMEX	Casos de uso
G5	NVIDIA A10G	Sim	Não	Sim	Não	Cargas de trabalho de inferência e gráficos
G6	NVIDIA L4	Sim	Não	Sim	Não	Inferência de IA e processamento de vídeo
G6e	NVIDIA L40S	Sim	Não	Sim	Não	Treinamento, inferência e gráficos
P4d/ P4de	NVIDIA A100	Sim	Sim	Sim	Não	Treinamento em grande escala e HPC
P5	NVIDIA H100	Sim	Sim	Sim	Não	Treinamento do modelo básico
P6	NVIDIA B200	Sim	Sim	Sim	Não	Modelos de bilhões ou trilhões de parâmetros, treinamento distribuído e inferência
P6e	NVIDIA 00 GB2	Sim	Sim	Sim	Sim	Modelos de bilhões ou trilhões de parâmetros, treinamento distribuído e inferência

A seguir estão as descrições de cada recurso na tabela:

- Divisão de tempo: permite que várias cargas de trabalho compartilhem recursos de computação da GPU ao longo do tempo.
- GPU de várias instâncias (MIG): particionamento em nível de hardware que cria instâncias de GPU isoladas.
- Serviço multiprocesso (MPS): permite a execução simultânea de vários processos CUDA em uma única GPU.
- Internode Memory Exchange (IMEX): comunicação coerente com a memória entre os nós para 00. GB2 UltraServers

Recursos adicionais do

Para obter mais informações sobre os drivers Kubernetes DRA e NVIDIA DRA, consulte os seguintes recursos em: GitHub

- Kubernetes dynamic-resource-allocation
- Proposta de aprimoramento do Kubernetes para DRA
- Driver NVIDIA DRA para GPUs
- Exemplos e início rápido do NVIDIA DRA

Configure a alocação dinâmica de recursos para gerenciamento avançado de GPU

O tópico a seguir mostra como configurar a alocação dinâmica de recursos (DRA) para gerenciamento avançado de GPU.

Pré-requisitos

Antes de implementar o DRA no Amazon EKS, certifique-se de que seu ambiente atenda aos seguintes requisitos.

Configuração do cluster

- Versão em execução do cluster Amazon EKS `1.33` ou posterior
- Grupos de nós gerenciados do Amazon EKS (atualmente, o DRA é suportado somente por grupos de nós gerenciados com AL2 023 e Bottlerocket otimizado AMIs para NVIDIA, não com Karpenter)
- Nós de trabalho habilitados para GPU NVIDIA com tipos de instância apropriados

Componentes necessários

- Versão do plug-in do dispositivo NVIDIA `0.17.1` ou posterior
- Versão do driver NVIDIA DRA `25.3.0` ou posterior

Etapa 1: Criar cluster com um grupo de nós habilitado para DRA usando eksctl

1. Crie um arquivo de configuração de cluster chamado`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. Crie o cluster:

```
eksctl create cluster -f dra-eks-cluster.yaml
```

Etapa 2: implantar o plug-in do dispositivo NVIDIA

Implante o plug-in de dispositivo NVIDIA para permitir a descoberta básica da GPU:

1. Adicione o repositório Helm do plug-in de dispositivo NVIDIA:

```
helm repo add nvidia https://nvidia.github.io/k8s-device-plugin
helm repo update
```

2. Crie valores personalizados para o plug-in do 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 o plug-in do dispositivo 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
```

Etapa 3: Implantar o gráfico Helm do driver NVIDIA DRA

1. Crie um arquivo de `dra-driver-values.yaml` valores para o driver 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. Adicione o repositório NVIDIA NGC Helm:

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
```

3. Instale o driver 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
```

Etapa 4: Verificar a instalação do DRA

1. Verifique se os recursos da API DRA estão disponíveis:

```
kubectl api-resources | grep resource.k8s.io/v1beta1
```

A seguir está a saída esperada:

```
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. Confira as classes de dispositivos disponíveis:

```
kubectl get deviceclasses
```

Veja a seguir um exemplo da saída esperada:

```
NAME                                        AGE
compute-domain-daemon.nvidia.com            4h39m
compute-domain-default-channel.nvidia.com   4h39m
gpu.nvidia.com                              4h39m
mig.nvidia.com                              4h39m
```

Quando uma instância de GPU G6 recém-criada se junta ao seu cluster Amazon EKS com o DRA ativado, as seguintes ações ocorrem:

- O driver NVIDIA DRA descobre automaticamente a GPU A10G e cria duas nesse nó. `resourceslices`
- A `gpu.nvidia.com` fatia registra o dispositivo físico da GPU A10G com suas especificações (memória, capacidade de computação e muito mais).
- Como o A10G não oferece suporte ao particionamento MIG, a `compute-domain.nvidia.com` fatia cria um único domínio computacional representando todo o contexto computacional da GPU.
- Em seguida, eles `resourceslices` são publicados no servidor da API Kubernetes, disponibilizando os recursos da GPU para agendamento. `resourceclaims`

  O programador DRA agora pode alocar de forma inteligente essa GPU para pods que solicitam recursos de GPU`resourceclaimtemplates`, fornecendo um gerenciamento de recursos mais flexível em comparação com as abordagens tradicionais de plug-ins de dispositivos. Isso acontece automaticamente sem intervenção manual. O nó simplesmente fica disponível para cargas de trabalho da GPU quando o driver DRA conclui o processo de descoberta e registro de recursos.

Quando você executa o seguinte comando:

```
kubectl get resourceslices
```

Veja a seguir um exemplo da saída esperada:

```
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
```

Avance para the section called “Programe uma carga de trabalho de GPU simples usando a alocação dinâmica de recursos”.

Programe uma carga de trabalho de GPU simples usando a alocação dinâmica de recursos

Para programar uma carga de trabalho simples da GPU usando a alocação dinâmica de recursos (DRA), siga as etapas a seguir. Antes de continuar, verifique se você seguiuthe section called “Configure a alocação dinâmica de recursos para gerenciamento avançado de GPU”.

1. Crie um arquivo básico `ResourceClaimTemplate` para alocação de GPU com um arquivo chamado: `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. Aplique o modelo:

```
kubectl apply -f basic-gpu-claim-template.yaml
```

3. Verifique o status:

```
kubectl get resourceclaimtemplates -n gpu-test1
```

   A seguir está um exemplo de saída:

```
NAME         AGE
single-gpu   9m16s
```

4. Crie um pod que use o `ResourceClaimTemplate` com um arquivo chamado`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. Aplique e monitore o Pod:

```
kubectl apply -f basic-gpu-pod.yaml
```

6. Verifique o status do pod:

```
kubectl get pod -n gpu-test1
```

   Veja a seguir um exemplo de saída esperada:

```
NAME      READY   STATUS    RESTARTS   AGE
gpu-pod   1/1     Running   0          13m
```

7. Verifique o `ResourceClaim` status:

```
kubectl get resourceclaims -n gpu-test1
```

   Veja a seguir um exemplo de saída esperada:

```
NAME                 STATE                AGE
gpu-pod-gpu0-l76cg   allocated,reserved   9m6s
```

8. Veja os registros do pod para ver as informações da GPU:

```
kubectl logs gpu-pod -n gpu-test1
```

   Veja a seguir um exemplo de saída esperada:

```
GPU 0: NVIDIA L4 (UUID: GPU-da7c24d7-c7e3-ed3b-418c-bcecc32af7c5)
```

Continue the section called “Técnicas de otimização de GPU com alocação dinâmica de recursos” para obter técnicas mais avançadas de otimização de GPU usando o DRA.

Técnicas de otimização de GPU com alocação dinâmica de recursos

As cargas de trabalho de GPU modernas exigem um gerenciamento sofisticado de recursos para alcançar a utilização ideal e a eficiência de custos. O DRA permite várias técnicas avançadas de otimização que abordam diferentes casos de uso e recursos de hardware:

- O corte de tempo permite que várias cargas de trabalho compartilhem recursos computacionais da GPU ao longo do tempo, tornando-a ideal para cargas de trabalho de inferência com uso esporádico da GPU. Para obter um exemplo, consulte the section called “Otimize as cargas de trabalho da GPU com divisão de tempo”.
- O serviço multiprocesso (MPS) permite a execução simultânea de vários processos CUDA em uma única GPU com melhor isolamento do que o corte de tempo. Para obter um exemplo, consulte the section called “Otimize as cargas de trabalho da GPU com o MPS”.
- A GPU de múltiplas instâncias (MIG) fornece particionamento em nível de hardware, criando instâncias de GPU isoladas com recursos dedicados de computação e memória. Para obter um exemplo, consulte the section called “Otimize as cargas de trabalho da GPU com a GPU de várias instâncias”.
- O Internode Memory Exchange (IMEX) permite a comunicação coerente com a memória entre os nós para treinamento distribuído em sistemas NVIDIA 00. GB2 Para obter um exemplo, consulte the section called “Otimize as cargas de trabalho da GPU com o IMEX usando GB2 100 instâncias P6e”.

Essas técnicas podem melhorar significativamente a utilização dos recursos. As organizações relatam que a utilização da GPU aumenta de 30 a 40% com a alocação tradicional para 80 a 90% com estratégias de compartilhamento otimizadas. A escolha da técnica depende das características da carga de trabalho, dos requisitos de isolamento e dos recursos do hardware.

Otimize as cargas de trabalho da GPU com divisão de tempo

O corte de tempo permite que várias cargas de trabalho compartilhem recursos computacionais da GPU, programando-as para serem executadas sequencialmente na mesma GPU física. É ideal para cargas de trabalho de inferência com uso esporádico de GPU.

Execute as etapas a seguir.

1. Defina um `ResourceClaimTemplate` para dividir o tempo com um arquivo chamado: `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 um pod usando o corte de tempo com um arquivo chamado: `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. Aplique o modelo e o Pod:

```
kubectl apply -f timeslicing-claim-template.yaml
kubectl apply -f timeslicing-pod.yaml
```

4. Monitore as reivindicações de recursos:

```
kubectl get resourceclaims -n timeslicing-gpu -w
```

   A seguir está um exemplo de saída:

```
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
```

Primeiro pod (`inference-pod-1`)

- Estado: `allocated,reserved`
- Significado: o DRA encontrou uma GPU disponível e a reservou para este Pod
- Status do pod: começa a ser executado imediatamente

Segundo pod (`training-pod-2`)

- Estado: `pending`
- Significado: Esperando que o DRA configure o corte de tempo na mesma GPU
- Status do pod: Esperando para ser agendado

- O estado passará de `pending` `allocated,reserved` para `running`

Otimize as cargas de trabalho da GPU com o MPS

O Multi-Process Service (MPS) permite a execução simultânea de vários contextos CUDA em uma única GPU com melhor isolamento do que o corte de tempo.

Execute as etapas a seguir.

1. Defina um `ResourceClaimTemplate` para MPS com um arquivo chamado`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 um pod usando MPS com um arquivo chamado`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. Aplique o modelo e crie vários pods MPS:

```
kubectl apply -f mps-claim-template.yaml
kubectl apply -f mps-pod.yaml
```

4. Monitore as reivindicações de recursos:

```
kubectl get resourceclaims -n mps-gpu -w
```

A seguir está um exemplo de saída:

```
NAME                                              STATE                AGE
mps-multi-container-pod-shared-gpu-claim-2p9kx   allocated,reserved   86s
```

Essa configuração demonstra o verdadeiro compartilhamento de GPU usando o NVIDIA Multi-Process Service (MPS) por meio da alocação dinâmica de recursos (DRA). Ao contrário do corte de tempo, em que as cargas de trabalho se revezam usando a GPU sequencialmente, o MPS permite que os dois contêineres sejam executados simultaneamente na mesma GPU física. O principal insight é que o compartilhamento do DRA MPS requer vários contêineres em um único pod, não vários pods separados. Quando implantado, o driver DRA aloca um `ResourceClaim` para o pod e configura automaticamente o MPS para permitir que os contêineres de inferência e treinamento sejam executados simultaneamente.

Cada contêiner tem seu próprio espaço de memória de GPU isolado e recursos computacionais, com o daemon MPS coordenando o acesso ao hardware subjacente. Você pode verificar se isso está funcionando fazendo o seguinte:

- Verificaçãonvidia-smi, que mostrará os dois contêineres como processos M+C (`MPS + Compute`) que compartilham o mesmo dispositivo de GPU.
- Monitorando os registros de ambos os contêineres, que exibirão registros de data e hora intercalados, comprovando a execução simultânea.

Essa abordagem maximiza a utilização da GPU, permitindo que cargas de trabalho complementares compartilhem eficientemente o caro hardware da GPU, em vez de deixá-lo subutilizado por um único processo.

Contêiner 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 |
+-----------------------------------------------------------------------------------------
+
```

Contêiner 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 |
+-----------------------------------------------------------------------------------------
+
```

Otimize as cargas de trabalho da GPU com a GPU de várias instâncias

A GPU de várias instâncias (MIG) fornece particionamento em nível de hardware, criando instâncias de GPU isoladas com recursos dedicados de computação e memória.

O uso do particionamento MIG dinâmico com vários perfis requer o operador de GPU NVIDIA. O operador de GPU NVIDIA usa o MIG Manager para criar perfis MIG e reinicializar as instâncias

da GPU, como P4D, P4De, P5, P6 e outras, para aplicar as alterações de configuração. O GPU Operator inclui recursos abrangentes de gerenciamento de MIG por meio do componente MIG Manager, que observa as alterações nos rótulos dos nós e aplica automaticamente a configuração MIG apropriada. Quando uma alteração do perfil MIG é solicitada, o operador desliga normalmente todos os clientes da GPU, aplica a nova geometria da partição e reinicia os serviços afetados. Esse processo requer uma reinicialização do nó para instâncias de GPU para garantir transições de estado de GPU limpas. É por isso que habilitar `WITH0REBOOT=true` a configuração do MIG Manager é essencial para implantações bem-sucedidas do MIG.

Você precisa do driver NVIDIA DRA e do operador de GPU NVIDIA para trabalhar com o MIG no Amazon EKS. Além disso, você não precisa do NVIDIA Device Plugin e do DCGM Exporter, pois eles fazem parte do NVIDIA GPU Operator. Como o EKS NVIDIA AMIs vem com os drivers NVIDIA pré-instalados, desativamos a implantação de drivers pelo operador da GPU para evitar conflitos e aproveitar os drivers otimizados já presentes nas instâncias. O driver NVIDIA DRA gerencia a alocação dinâmica de recursos para instâncias MIG, enquanto o operador da GPU gerencia todo o ciclo de vida da GPU. Isso inclui configuração MIG, funcionalidade de plug-in de dispositivo, monitoramento por meio de DCGM e descoberta de recursos de nós. Essa abordagem integrada fornece uma solução completa para gerenciamento de GPU corporativa, com isolamento em nível de hardware e recursos dinâmicos de alocação de recursos.

Etapa 1: implantar o operador de GPU NVIDIA

1. Adicione o repositório NVIDIA GPU Operator:

```
helm repo add nvidia https://nvidia.github.io/gpu-operator
helm repo update
```

2. Crie um `gpu-operator-values.yaml` arquivo:

```
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 o GPU Operator usando o `gpu-operator-values.yaml` arquivo:

```
helm install gpu-operator nvidia/gpu-operator \
  --namespace gpu-operator \
  --create-namespace \
  --version v25.3.1 \
```

```
  --values gpu-operator-values.yaml
```

Este gráfico do Helm implanta os seguintes componentes e vários perfis MIG:

- Plugin de dispositivo (agendamento de recursos de GPU)
- DCGM Exporter (métricas e monitoramento de GPU)
- Node Feature Discovery (NFD - rotulagem de hardware)
- Descoberta de recursos de GPU (GFD - rotulagem específica da GPU)
- MIG Manager (particionamento de GPU de várias instâncias)
- Kit de ferramentas de contêiner (tempo de execução de contêiner de GPU)
- Controlador do operador (gerenciamento do ciclo de vida)

4. Verifique os pods de implantação:

```
kubectl get pods -n gpu-operator
```

A seguir está um exemplo de saída:

```
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. Crie um cluster Amazon EKS com um grupo de nós gerenciados pelo P4de para testar os exemplos do 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
```

O operador de GPU NVIDIA usa o rótulo adicionado aos nós `nvidia.com/mig.config: "p4de-half-balanced"` e particiona a GPU com o perfil fornecido.

6. Faça login na `p4de` instância.
7. Execute o seguinte comando:

```
nvidia-smi -L
```

Você deve ver o seguinte exemplo de saída:

```
[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)
```

O NVIDIA GPU Operator aplicou com sucesso o perfil `p4de-half-balanced` MIG à sua instância P4DE, criando partições de GPU em nível de hardware conforme configurado. Veja como o particionamento funciona:

O operador da GPU aplicou essa configuração a partir do seu perfil MIG incorporado:

```
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
```

Com base na sua `nvidia-smi -L` saída, veja o que o operador da GPU criou:

- Habilitado para MIG GPUs (0-3): particionado por hardware

- GPU 0: NVIDIA A100- -80 GB SXM4
  - MIG 3g.40gb Device 0 — Grandes cargas de trabalho (40 GB de memória, 42) SMs
  - MIG 2g.20gb Dispositivo 1 — Cargas de trabalho médias (20 GB de memória, 28) SMs
  - MIG 1g.10gb Dispositivo 2 — Cargas de trabalho pequenas (10 GB de memória, 14) SMs
  - MIG 1g.10gb Dispositivo 3 — Cargas de trabalho pequenas (10 GB de memória, 14) SMs
- GPU 1: NVIDIA A100- -80 GB SXM4
  - MIG 3g.40gb Dispositivo 0 — Layout de partição idêntico
  - MIG 2g.20gb Dispositivo 1
  - MIG 1g.10gb Dispositivo 2
  - MIG 1g.10gb Dispositivo 3
- GPU 2 e GPU 3 — Mesmo padrão da GPU 0 e da GPU 1
- Completo GPUs (4-7): Sem particionamento 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

Depois que o operador de GPU NVIDIA cria as partições MIG, o driver NVIDIA DRA detecta automaticamente essas instâncias isoladas de hardware e as disponibiliza para alocação dinâmica de recursos no Kubernetes. O driver DRA descobre cada instância MIG com seu perfil específico (1g.10gb, 2g.20gb, 3g.40gb) e as expõe como recursos programáveis por meio da classe de dispositivo. `mig.nvidia.com`

O driver DRA monitora continuamente a topologia MIG e mantém um inventário das instâncias disponíveis em todas. GPUs Quando um pod solicita um perfil MIG específico por meio de um`ResourceClaimTemplate`, o driver DRA seleciona de forma inteligente uma instância MIG apropriada de qualquer GPU disponível, permitindo uma verdadeira multilocação em nível de hardware. Essa alocação dinâmica permite que várias cargas de trabalho isoladas sejam executadas simultaneamente na mesma GPU física, mantendo limites rígidos de recursos e garantias de desempenho.

Etapa 2: testar a alocação de recursos do MIG

Agora, vamos executar alguns exemplos para demonstrar como o DRA aloca dinamicamente instâncias MIG para diferentes cargas de trabalho. Implante `resourceclaimtemplates` e teste os pods para ver como o driver DRA coloca as cargas de trabalho nas partições MIG disponíveis, permitindo que vários contêineres compartilhem recursos de GPU com isolamento em nível de hardware.

1. Crie `mig-claim-template.yaml` para conter o 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 os três modelos:

```
kubectl apply -f mig-claim-template.yaml
```

3. Execute o seguinte comando:

```
kubectl get resourceclaimtemplates -n mig-gpu
```

A seguir está um exemplo de saída:

```
NAME                 AGE
mig-large-template   71m
mig-medium-template  71m
mig-small-template   71m
```

4. Crie `mig-pod.yaml` para agendar vários trabalhos para aproveitar issoresourceclaimtemplates:

```
---
# 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. Aplique essa especificação, que deve implantar três pods:

```
kubctl apply -f mig-pod.yaml
```

   Esses pods devem ser programados pelo driver do DRA.

6. Verifique os registros do Pod do driver DRA e você verá uma saída semelhante a esta:

```
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. Verifique o status do pod `resourceclaims` para ver o status:

```
kubectl get resourceclaims -n mig-gpu -w
```

   A seguir está um exemplo de saída:

```
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 você pode ver, todos os pods foram movidos de pendentes para `allocated,reserved` pelo driver do DRA.

8. Execute `nvidia-smi` a partir do nó. Você notará que três processadores Python estão em execução:

```
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 |**
+-----------------------------------------------------------------------------------------
+
```

Otimize as cargas de trabalho da GPU com o IMEX usando GB2 100 instâncias P6e

O IMEX (Internode Memory Exchange) permite a comunicação coerente com a memória entre os nós para treinamento distribuído na NVIDIA 00. GB2 UltraServers

Execute as etapas a seguir.

1. Defina um `ComputeDomain` para treinamento de vários nós com um arquivo chamado`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 um pod usando canais IMEX com um arquivo chamado`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**
>
> Isso requer instâncias P6e GB2 00.

3. Implante o IMEX aplicando os modelos `ComputeDomain` e:

```
kubectl apply -f imex-claim-template.yaml
```

```
kubectl apply -f imex-compute-domain.yaml
kubectl apply -f imex-pod.yaml
```

4. Verifique o `ComputeDomain` status.

```
kubectl get computedomain distributed-training-domain
```

5. Monitore a implantação do daemon IMEX.

```
kubectl get pods -n nvidia-dra-driver -l resource.nvidia.com/computeDomain
```

6. Verifique os canais IMEX no Pod:

```
kubectl exec imex-distributed-training -- ls -la /dev/nvidia-caps-imex-channels/
```

7. Veja os registros do pod:

```
kubectl logs imex-distributed-training
```

   Veja a seguir um exemplo da saída esperada:

```
=== 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 obter mais informações, consulte o exemplo da NVIDIA em. GitHub

Redes

Considere maior largura de banda de rede ou adaptador de malha elástica para aplicativos com alta comunicação entre nós

Para cargas de trabalho de treinamento distribuídas no Amazon EKS com altas demandas de comunicação entre nós, considere selecionar instâncias com maior largura de banda de rede ou Elastic Fabric Adapter (EFA). O desempenho insuficiente da rede pode dificultar a transferência de dados, retardando as tarefas de aprendizado de máquina, como o treinamento distribuído de várias

GPUs. Observe que as cargas de trabalho de inferência normalmente não têm alta comunicação entre nós.

Exemplo

Por exemplo, usando o 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
```

Garanta que ferramentas como MPI e NCCL estejam instaladas em sua imagem de contêiner para aproveitar o EFA em trabalhos de treinamento.

Aumente o número de endereços IP disponíveis para permitir tempos de inicialização mais rápidos do pod

No EKS, cada pod precisa de um endereço IP do bloco CIDR da VPC. À medida que seu cluster se expande com mais nós e pods, você corre o risco de esgotar o endereço IP ou diminuir o desempenho, mas habilitar a delegação de prefixos pode mitigar esses problemas ao pré-alocar intervalos de IP e reduzir as chamadas de EC2 API, resultando em tempos de inicialização mais rápidos do pod e melhor escalabilidade.

Ativar a delegação de prefixos após criar seu cluster permite que a VPC Container Network Interface (CNI) atribua prefixos IP (/28, cada um fornecendo 16 endereços IP) às interfaces de rede nas instâncias. EC2 Isso significa que cada nó pode suportar mais pods, reduzindo o risco de escassez de IP. Por exemplo, em uma `c5.4xlarge` instância, você pode oferecer suporte a até 110 pods com delegação de prefixo.

Embora a delegação de prefixos seja crucial para otimizar o uso de IP em ambientes com muitos pods pequenos, as AI/ML cargas de trabalho geralmente usam menos pods maiores (por exemplo, um pod por GPU). A ativação da delegação de prefixos permite que a CNI da VPC pré-aloque um prefixo para acelerar a inicialização do pod, mantendo um pool aquecido. Isso significa que os endereços IP estão prontamente disponíveis, reduzindo o tempo necessário para a inicialização do pod em comparação com a alocação sob demanda no modo sem prefixo. Nesses casos, a economia de IP ao habilitar a delegação de prefixos oferece benefícios de desempenho para AI/ML cargas de trabalho. Ao reduzir o número de chamadas de EC2 API necessárias para a configuração do endereço IP e pré-alocar intervalos de IP, o uso da delegação de prefixos permite tempos de inicialização mais rápidos do pod, o que é particularmente benéfico para escalar rapidamente as cargas de trabalho. AI/ML

Para habilitar a delegação de prefixos:

```
kubectl set env daemonset/aws-node -n kube-system ENABLE_PREFIX_DELEGATION=true
```

Garanta o planejamento adequado das sub-redes VPC para evitar o esgotamento do endereço IP, especialmente em grandes implantações, e gerencie os blocos CIDR para evitar sobreposições. VPCs Para saber mais, consulte Otimizar a utilização de endereços IP e atribuir mais endereços IP aos nós do Amazon EKS com prefixos.

Segurança

Segurança e conformidade

Considere o S3 com KMS para armazenamento compatível com criptografia

A menos que você especifique o contrário, todos os buckets do S3 usam o SSE-S3 por padrão para criptografar objetos em repouso. No entanto, você pode optar por configurar buckets para usar criptografia do lado do servidor com chaves do AWS Key Management Service (AWS KMS) (SSE-KMS) em vez disso. Os controles de segurança do AWS KMS podem ajudá-lo a satisfazer os requisitos de conformidade relacionados à criptografia. Você pode usar essas chaves do KMS para proteger seus dados em buckets do Amazon S3. Quando você usa a criptografia SSE-KMS com um bucket S3, as chaves do AWS KMS devem estar na mesma região do bucket.

Configure seus buckets de uso geral para usar chaves de bucket do S3 para SSE-KMS, a fim de reduzir seus custos de solicitação do AWS KMS em até 99%, diminuindo o tráfego de solicitações

do Amazon S3 para o AWS KMS. As chaves de bucket do S3 estão sempre habilitadas `GET` e `PUT` operam em um bucket de diretório e não podem ser desativadas.

Observe que o Amazon S3 Express One Zone usa um tipo específico de bucket chamado bucket de diretório S3. Os buckets de diretório são exclusivamente para a classe de armazenamento S3 Express One Zone e permitem acesso de alto desempenho e baixa latência. Para configurar a criptografia de bucket padrão em um bucket de diretório do S3, use a AWS CLI e especifique o ID da chave KMS ou o ARN, não o alias, como no exemplo a seguir:

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"}}]}'
```

Certifique-se de que a função IAM do seu pod EKS tenha permissões KMS (por exemplo,`kms:Decrypt`) para acessar objetos criptografados. Teste isso em um ambiente de teste fazendo o upload de um modelo de amostra para o bucket, montando-o em um pod (por exemplo, por meio do driver CSI Mountpoint S3) e verificando se o pod pode ler os dados criptografados sem erros. Registros de auditoria via AWS CloudTrail para confirmar a conformidade com os requisitos de criptografia. Consulte a documentação do KMS para obter detalhes de configuração e gerenciamento de chaves.

Armazenamento

Gerenciamento e armazenamento de dados

Implante modelos de IA em pods usando um driver CSI

As cargas de trabalho de IA/ML geralmente exigem acesso a grandes artefatos de modelos (por exemplo, pesos e configurações treinados), e os pods precisam de uma forma confiável e escalável de acessá-los sem incorporá-los às imagens do contêiner, o que pode aumentar o tamanho das imagens e os tempos de extração do Container Registry. Para reduzir a sobrecarga operacional do gerenciamento de montagens de volume, recomendamos implantar modelos de IA em pods montando serviços de armazenamento da Amazon (por exemplo, S3, FSx para Lustre, para FSx OpenZFS, EFS) como volumes persistentes () usando seus respectivos drivers CSI. PVs Para obter detalhes sobre a implementação, consulte os tópicos subsequentes nesta seção.

Otimize o armazenamento para caches de modelos de ML no EKS

Aproveitar uma solução de armazenamento ideal é fundamental para minimizar a latência de inicialização do pod e do aplicativo, reduzir o uso de memória, obter os níveis desejados de desempenho para acelerar as cargas de trabalho e garantir a escalabilidade das cargas de trabalho de ML. As cargas de trabalho de ML geralmente dependem de arquivos de modelo (pesos), que podem ser grandes e exigir acesso compartilhado aos dados entre pods ou nós. A seleção da solução de armazenamento ideal depende das características da sua carga de trabalho, como eficiência de um único nó, acesso a vários nós, requisitos de latência, restrições de custo e também requisitos de integração de dados (como com um repositório de dados do Amazon S3). Recomendamos comparar diferentes soluções de armazenamento com suas cargas de trabalho para entender qual delas atende às suas necessidades, e fornecemos as seguintes opções para ajudá-lo a avaliar com base em suas necessidades de carga de trabalho.

O driver EKS CSI oferece suporte aos seguintes serviços de armazenamento da AWS, cada um com seu próprio driver CSI e vem com seus próprios pontos fortes para fluxos de trabalho de IA e ML:

- Ponto de montagem para Amazon S3
- Amazon FSx para Lustre
- Amazon FSx para OpenZFS
- Amazon EFS
- Amazon EBS

A escolha do serviço de armazenamento da AWS depende da sua arquitetura de implantação, escala, requisitos de desempenho e estratégia de custo. Os drivers CSI de armazenamento precisam ser instalados em seu cluster EKS, o que permite que o driver CSI crie e gerencie volumes persistentes (PV) fora do ciclo de vida de um pod. Usando o driver CSI, você pode criar definições fotovoltaicas de serviços de armazenamento da AWS compatíveis como recursos de cluster do EKS. Os pods podem então acessar esses volumes de armazenamento para seus volumes de dados por meio da criação de uma reivindicação de volume persistente (PVC) para o PV. Dependendo do serviço de armazenamento da AWS e do seu cenário de implantação, um único PVC (e seu PV associado) pode ser conectado a vários pods para uma carga de trabalho. Por exemplo, para o treinamento de ML, os dados de treinamento compartilhados são armazenados em um PV e acessados por vários pods; para inferência on-line em tempo real, os modelos LLM são armazenados em cache em um PV e acessados por vários pods. Exemplos de arquivos YAML

PV e PVC para os serviços de armazenamento da AWS são fornecidos abaixo para ajudar você a começar.

Desempenho de monitoramento O baixo desempenho do disco pode atrasar a leitura da imagem do contêiner, aumentar a latência de inicialização do pod e diminuir a taxa de transferência de inferência ou treinamento. Use CloudWatch a Amazon para monitorar as métricas de desempenho dos seus serviços de armazenamento da AWS. Ao identificar gargalos de desempenho, modifique seus parâmetros de configuração de armazenamento para otimizar o desempenho.

Cenário: carga de trabalho de várias instâncias de GPU

Amazon FSx for Lustre: em cenários em que você tem um ambiente de várias instâncias de computação de EC2 GPU com cargas de trabalho dinâmicas sensíveis à latência e com alta taxa de transferência de largura de banda, como treinamento distribuído e fornecimento de modelos, e precisa de integração nativa com o repositório de dados Amazon S3, recomendamos o Amazon for Lustre. FSx FSx for Lustre fornece um sistema de arquivos paralelo de alto desempenho totalmente gerenciado, projetado para cargas de trabalho com uso intensivo de computação, como computação de alto desempenho (HPC) e Machine Learning.

Você pode instalar o driver CSI for Lustre FSx para montar FSx sistemas de arquivos no EKS como um volume persistente (PV) e, em seguida, implantar o sistema de arquivos Lustre como um cache autônomo de alto desempenho ou como um sistema de arquivos vinculado ao S3 FSx para atuar como um cache de alto desempenho para dados do S3, fornecendo alta I/O e rápida taxa de transferência para acesso aos dados em suas instâncias de computação da GPU. FSx for Lustre pode ser implantado com opções de armazenamento Scratch-SSD ou SSD persistente:

- Armazenamento SSD Scratch: recomendado para cargas de trabalho efêmeras ou de curta duração (horas), com capacidade de taxa de transferência fixa por TiB provisionada.
- Armazenamento SSD persistente: recomendado para cargas de trabalho de missão crítica e de longa duração que exigem o mais alto nível de disponibilidade, por exemplo, simulações de HPC, análise de big data ou treinamento em Machine Learning. Com o armazenamento SSD persistente, você pode configurar a capacidade de armazenamento e a capacidade de transferência (por TiB) necessárias.

Considerações de desempenho:

- Pod administrativo FSx para gerenciar o sistema de arquivos Lustre: configure um pod “administrativo” que tenha o cliente Lustre instalado e tenha o sistema de FSx arquivos montado.

Isso habilitará um ponto de acesso para permitir o ajuste fino do sistema de FSx arquivos e também em situações em que você precise pré-aquecer o sistema de FSx arquivos com seus dados de treinamento de ML ou modelos LLM antes de iniciar suas instâncias de computação de GPU. Isso é especialmente importante se sua arquitetura utiliza instâncias de GPU/computação da EC2 Amazon baseadas em Spot, nas quais você pode utilizar o pod administrativo para “aquecer” ou “pré-carregar” os dados desejados no FSx sistema de arquivos, para que os dados estejam prontos para serem processados quando você executar suas instâncias da Amazon baseadas em Spot. EC2

- Elastic Fabric Adapter (EFA): os tipos de implantação de armazenamento SSD persistente oferecem suporte ao Elastic Fabric Adapter (EFA), onde o uso do EFA é ideal para cargas de trabalho baseadas em GPU de alto desempenho e taxa de transferência. Observe que FSx o Lustre oferece suporte ao NVIDIA GPUDirect Storage (GDS), em que o GDS é uma tecnologia que cria um caminho de dados direto entre o armazenamento local ou remoto e a memória da GPU, para permitir um acesso mais rápido aos dados.
- Compressão: ative a compactação de dados no sistema de arquivos se você tiver tipos de arquivo que possam ser compactados. Isso pode ajudar a aumentar o desempenho, pois a compactação de dados reduz a quantidade de dados transferidos entre os servidores FSx de arquivos e o armazenamento Lustre.
- Configuração de distribuição do sistema de arquivos Lustre:
  - Distribuição de dados: permite que FSx o Luster distribua os dados de um arquivo em vários destinos de armazenamento de objetos (OSTs) em um sistema de arquivos Lustre, maximizando o acesso e a taxa de transferência paralelos, especialmente para trabalhos de treinamento de ML em grande escala.
  - Distribuição autônoma do sistema de arquivos: por padrão, uma configuração de distribuição do Lustre de 4 componentes é criada para você por meio do recurso de layouts de arquivo progressivos (PFL) do Lustre. FSx Na maioria dos cenários, você não precisa atualizar a contagem/tamanho padrão de faixas do PFL Lustre. Se você precisar ajustar a distribuição de dados do Lustre, poderá ajustar manualmente a distribuição do Lustre consultando os parâmetros de distribuição de um FSx sistema de arquivos do Lustre.
  - Sistema de arquivos vinculado ao S3: os arquivos importados para o FSx sistema de arquivos usando a integração nativa do Amazon S3 (Data Repository Association ou DRA) não usam o layout PFL padrão, mas usam o layout no parâmetro do sistema de arquivos. `ImportedFileChunkSize` Arquivos importados do S3 maiores que o `ImportedFileChunkSize` serão armazenados em vários OSTs com uma contagem de faixas

com base no valor `ImportedFileChunkSize` definido (padrão 1GiB). Se você tiver arquivos grandes, recomendamos ajustar esse parâmetro para um valor maior.

- Posicionamento: implante um sistema de arquivos FSx for Lustre na mesma zona de disponibilidade dos nós de computação ou GPU para permitir o acesso de menor latência aos dados, evitando padrões de acesso entre zonas de disponibilidade. Se você tiver vários nós de GPU localizados em diferentes zonas de disponibilidade, recomendamos implantar um sistema de FSx arquivos em cada zona de disponibilidade para acesso a dados de baixa latência.

Exemplo

Definição de volume persistente (PV) FSx para um sistema de arquivos for Lustre, usando provisionamento estático (onde a FSx instância já foi provisionada).

```
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]
```

Exemplo

Definição de reivindicação de volume persistente para PV chamada`fsx-pv`:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: fsx-claim
```

```
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 1200Gi
  volumeName: fsx-pv
```

Exemplo

Configure um pod para usar uma declaração de volume 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 obter exemplos completos, consulte os exemplos FSx de drivers do Lustre em GitHub. Monitore as métricas FSx de desempenho do Amazon for Lustre usando a Amazon CloudWatch. Quando forem identificados gargalos de desempenho, ajuste seus parâmetros de configuração conforme necessário.

Cenário: carga de trabalho de instância de GPU única

Mountpoint para Amazon S3 com driver CSI: você pode montar um bucket S3 como um volume em seus pods usando o driver Mountpoint for Amazon S3 CSI. Esse método permite um controle de acesso refinado sobre quais pods podem acessar buckets específicos do S3. Cada pod tem sua própria instância de ponto de montagem e cache local (5 a 10 GB), isolando o carregamento do modelo e o desempenho de leitura entre os pods. Essa configuração oferece suporte à

autenticação em nível de pod com funções do IAM para contas de serviço (IRSA) e controle de versão independente de modelos para diferentes modelos ou clientes. A desvantagem é o aumento do uso de memória e do tráfego da API, pois cada pod emite chamadas de API do S3 e mantém seu próprio cache.

Exemplo parcial de um YAML de implantação de pod com driver 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
```

Considerações de desempenho:

- Cache de dados: o Mountpoint for S3 pode armazenar conteúdo em cache para reduzir custos e melhorar o desempenho de leituras repetidas no mesmo arquivo. Consulte Configuração de cache para ver as opções e os parâmetros de armazenamento em cache.
- Tamanho parcial do objeto: ao armazenar e acessar arquivos com mais de 72 GB, consulte Configurando o desempenho do Mountpoint para entender como configurar os parâmetros e a `--write-part-size` linha de comando para atender aos requisitos de perfil de dados `--read-part-size` e carga de trabalho.

- O cache compartilhado foi projetado para objetos de até 1 MB de tamanho. Ele não suporta objetos grandes. Use a opção Cache local para armazenar objetos em cache NVMe ou volumes do EBS no nó EKS.
- Cobranças de solicitação de API: ao realizar um grande número de operações de arquivo com o Mountpoint for S3, as cobranças de solicitação de API podem se tornar uma parte dos custos de armazenamento. Para mitigar isso, se uma consistência forte não for necessária, sempre ative o cache de metadados e defina o `metadata-ttl` período para reduzir o número de operações de API para o S3.

Para obter mais detalhes, consulte o driver Mountpoint for Amazon S3 CSI na documentação oficial do Amazon EKS. Recomendamos monitorar as métricas de desempenho do Amazon S3 com as CloudWatch métricas da Amazon se ocorrerem gargalos e ajustar sua configuração quando necessário.

Amazon FSx for OpenZFS: armazenamento compartilhado persistente

Para cenários envolvendo várias instâncias de computação de EC2 GPU com cargas de trabalho sensíveis à latência que exigem alta disponibilidade, alto desempenho, economia e várias implantações de pods para diferentes aplicativos, recomendamos a Amazon para OpenZFS. FSx Alguns exemplos de carga de trabalho incluem inferência em tempo real, aprendizado por reforço e treinamento de redes adversárias generativas. FSx para OpenZFS é particularmente benéfico para cargas de trabalho que precisam de acesso de alto desempenho a uma estrutura de diretórios focada com arquivos pequenos usando pequenos padrões de acesso a dados de E/S. Além disso, FSx o OpenZFS oferece a flexibilidade de escalar o desempenho independentemente da capacidade de armazenamento, ajudando você a obter a melhor eficiência de custos ao adequar o tamanho do armazenamento às necessidades reais e, ao mesmo tempo, manter os níveis de desempenho exigidos.

O driver CSI nativo FSx do OpenZFS permite a criação de vários PVCs sistemas de arquivos em um único, criando vários volumes. Isso reduz a sobrecarga de gerenciamento e maximiza a utilização da taxa de transferência e do IOPS do sistema de arquivos por meio de implantações consolidadas de pods de aplicativos em um único sistema de arquivos. Além disso, inclui recursos corporativos, como instantâneos com cópia zero, clones com cópia zero e cotas de usuários e grupos, que podem ser provisionadas dinamicamente por meio do driver CSI.

FSx para OpenZFS suporta três tipos diferentes de implantação após a criação:

- Single-AZ: opção de menor custo com latências inferiores a um milissegundo, mas não oferece alta disponibilidade no nível do sistema de arquivos ou da zona de disponibilidade. Recomendado para cargas de trabalho de desenvolvimento e teste ou aquelas que têm alta disponibilidade na camada de aplicação.
- Single-AZ (HA): fornece alta disponibilidade no nível do sistema de arquivos com latências inferiores a um milissegundo. Recomendado para cargas de trabalho de alto desempenho que exigem alta disponibilidade.
- Multi-AZ: fornece alta disponibilidade no nível do sistema de arquivos, bem como em todas as zonas de disponibilidade. Recomendado para cargas de trabalho de alto desempenho que exigem disponibilidade adicional em todas as zonas de disponibilidade.

Considerações de desempenho:

- Tipo de implantação: se a disponibilidade adicional nas zonas de disponibilidade não for um requisito, considere usar o tipo de implantação Single-AZ (HA). Esse tipo de implantação fornece até 100% da taxa de transferência para gravações, mantém latências inferiores a um milissegundo e os sistemas de arquivos Gen2 têm um NVMe cache adicional para armazenar até terrabytes de dados acessados com frequência. Os sistemas de arquivos Multi-AZ fornecem até 75% da taxa de transferência para gravações com uma latência maior para acomodar o tráfego entre AZ.
- Taxa de transferência e IOPS: tanto a taxa de transferência quanto a IOPS configuradas para o sistema de arquivos podem ser aumentadas ou reduzidas após a implantação. Você pode provisionar até 10% GB/s of disk throughput providing up to 21GB/s do acesso aos dados em cache. O IOPS pode ser escalado até 400.000 a partir do disco e o cache pode fornecer mais de 1 milhão de IOPS. Observe que o escalonamento da taxa de transferência de um sistema de arquivos Single-AZ causa uma breve interrupção do sistema de arquivos, pois não existe alta disponibilidade. O escalonamento da taxa de transferência de um sistema de arquivos Single-AZ (HA) ou Multi-AZ pode ser feito sem interrupções. O SSD IOPS pode ser escalado uma vez a cada seis horas.
- Classe de armazenamento: FSx para OpenZFS suporta tanto a classe de armazenamento SSD quanto a classe de armazenamento Intelligent-Tiering. Para AI/ML cargas de trabalho, é recomendável usar a classe de armazenamento SSD que fornece desempenho consistente à carga de trabalho, mantendo as CPUs/GPUs o mais ocupadas possível.
- Compressão: ative o algoritmo de LZ4 compactação se você tiver uma carga de trabalho que possa ser compactada. Isso reduz a quantidade de dados que cada arquivo consome no cache,

permitindo que mais dados sejam fornecidos diretamente do cache como taxa de transferência de rede e IOPS, reduzindo a carga no disco SSD.

- Tamanho do registro: a maioria das AI/ML cargas de trabalho se beneficiará ao deixar o tamanho de registro padrão de 128 KiB. Esse valor só deve ser reduzido se o conjunto de dados consistir em arquivos grandes (acima de 10 GiB) com acesso consistente em pequenos blocos abaixo de 128 KiB do aplicativo.

Depois que o sistema de arquivos é criado, um volume raiz associado é criado automaticamente pelo serviço. É uma prática recomendada armazenar dados em volumes secundários do volume raiz no sistema de arquivos. Usando o driver CSI FSx para OpenZFS, você cria uma Declaração de Volume Persistente associada para criar dinamicamente o volume secundário.

Exemplos:

Uma definição de Classe de Armazenamento (SC) para um volume FSx para OpenZFS, usada para criar um volume secundário do volume raiz ($ROOT_VOL_ID) em um sistema de arquivos existente e exportar o volume para o VPC CIDR ($VPC_CIDR) usando o 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
```

Uma reivindicação de volume persistente (PVC) criada dinamicamente em relação à fsxz-vol-sc criada acima. Observe que a capacidade de armazenamento alocada é de 1Gi, isso é necessário FSx para volumes OpenZFS, conforme observado nas perguntas frequentes do driver CSI. O volume receberá a capacidade total provisionada ao sistema de arquivos com essa configuração. Se a capacidade do volume precisar ser restringida, você poderá fazer isso usando cotas de usuários ou grupos.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: dynamic-vol-pvc
  namespace: example
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: fsxz-vol-sc
  resources:
    requests:
      storage: 1Gi
```

Configure um pod para montar um volume usando a Declaração de Volume 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 caches de modelos compartilhados

Em cenários em que você tem um ambiente de várias instâncias de computação de EC2 GPU e tem cargas de trabalho dinâmicas que exigem acesso compartilhado ao modelo em vários nós e zonas de disponibilidade (por exemplo, inferência on-line em tempo real com o Karpenter) com necessidades moderadas de desempenho e escalabilidade, recomendamos usar um sistema de arquivos Amazon Elastic File System (EFS) como um volume persistente por meio do driver EFS CSI. O Amazon EFS é um sistema de arquivos NFS baseado em nuvem totalmente gerenciado, altamente disponível e escalável que habilita EC2 instâncias e contêineres com armazenamento compartilhado de arquivos, com desempenho consistente e onde não é necessário provisionamento antecipado de armazenamento. Use o EFS como o volume do modelo e monte o volume como um sistema de arquivos compartilhado por meio da definição de um volume persistente no cluster EKS. Cada reivindicação de volume persistente (PVC) apoiada por um sistema de arquivos EFS é criada como um ponto de acesso EFS ao sistema de arquivos EFS. O EFS permite que vários nós e pods acessem os mesmos arquivos de modelo, eliminando a necessidade de sincronizar dados com o sistema de arquivos de cada nó. Instale o driver EFS CSI para integrar o EFS com o EKS.

Você pode implantar um sistema de arquivos Amazon EFS com os seguintes modos de taxa de transferência:

- Taxa de transferência contínua: dimensiona a taxa de transferência de acordo com o tamanho do sistema de arquivos, adequada para cargas de trabalho variadas com picos ocasionais.
- Taxa de transferência provisionada: taxa de transferência dedicada, ideal para trabalhos consistentes de treinamento de ML com necessidades de desempenho previsíveis dentro de limites.
- Taxa de transferência elástica (recomendada para ML): escala automaticamente com base na carga de trabalho e na relação custo-benefício para cargas de trabalho de ML variáveis.

Para ver as especificações de desempenho, consulte as especificações de desempenho do Amazon EFS.

Considerações de desempenho:

- Use o Elastic Throughput para cargas de trabalho variadas.
- Use a classe de armazenamento padrão para cargas de trabalho de ML ativas.

Para obter exemplos completos do uso do sistema de arquivos Amazon EFS como um volume persistente em seu cluster e pods do EKS, consulte os exemplos de drivers CSI do EFS em. GitHub Monitore as métricas de desempenho do Amazon EFS usando a Amazon CloudWatch. Quando forem identificados gargalos de desempenho, ajuste seus parâmetros de configuração conforme necessário.

Use o S3 Express One Zone para fluxos de trabalho orientados a objetos e sensíveis à latência

Para AI/ML cargas de trabalho sensíveis à latência no Amazon EKS, como treinamento de modelos em grande escala, inferência ou análise de alto desempenho, recomendamos usar o S3 Express One Zone para armazenamento e recuperação de modelos de alto desempenho. O S3 Express One Zone oferece um namespace hierárquico, como um sistema de arquivos, no qual você simplesmente faz o upload para um bucket de diretórios, adequado para “colocar tudo dentro” e, ao mesmo tempo, manter a alta velocidade. Isso é particularmente útil se você estiver acostumado com fluxos de trabalho orientados a objetos. Como alternativa, se você estiver mais acostumado com sistemas de arquivos (por exemplo, compatíveis com POSIX), você pode preferir o Amazon FSx for Lustre ou o OpenZFS. O Amazon S3 Express One Zone armazena dados em uma única zona de disponibilidade (AZ) usando buckets de diretório e oferecendo menor latência do que os buckets S3 padrão, que distribuem dados entre vários. AZs Para obter melhores resultados, certifique-se de colocar sua computação EKS na mesma AZ do bucket Express One Zone. Para saber mais sobre as diferenças do S3 Express One Zone, consulte Diferenças para buckets de diretório.

Para acessar o S3 Express One Zone com a semântica do sistema de arquivos, recomendamos usar o driver Mountpoint S3 CSI, que monta buckets S3 (incluindo Express One Zone) como um sistema de arquivos local. Isso traduz operações de arquivo (por exemplo, abertura, leitura, gravação) em chamadas de API do S3, fornecendo acesso de alto rendimento otimizado para cargas de trabalho com muita leitura de vários clientes e gravações sequenciais em novos objetos. Para obter detalhes sobre as operações e limitações suportadas (por exemplo, sem conformidade total com POSIX, mas anexos e renomeações compatíveis com o Express One Zone), consulte a documentação semântica do Mountpoint.

Benefícios de desempenho

- Fornece acesso aos dados até 10 vezes mais rápido do que o S3 Standard, com latência consistente de um dígito em milissegundos e custos de solicitação até 80% menores.
- É escalável para lidar com centenas de milhares a milhões de solicitações por segundo por bucket de diretório, evitando limitações ou interrupções observadas no S3 padrão durante cargas

extremas (por exemplo, de clusters com dezenas a centenas de milhares de redes saturadas). GPUs/CPUs

- Usa um mecanismo de autenticação baseado em sessão. Autentique-se uma vez para obter um token de sessão e, em seguida, execute operações repetidas em alta velocidade sem sobrecarga de autenticação por solicitação. Isso é otimizado para cargas de trabalho, como pontos de verificação frequentes ou carregamento de dados.

Casos de uso recomendados

- Cache: Um dos principais casos de uso do driver Mountpoint S3 CSI com o S3 Express One Zone é o armazenamento em cache. A primeira instância lê dados do S3 Standard (uso geral), armazenando-os em cache na Express One Zone de baixa latência. As leituras subsequentes de outros clientes acessam os dados em cache mais rapidamente, o que é ideal para cenários de vários nós em que vários nós EKS leem os mesmos dados (por exemplo, conjuntos de dados de treinamento compartilhados). Isso pode melhorar o desempenho em até 7 vezes para acessos repetidos e reduzir os custos de computação. Para cargas de trabalho que exigem total conformidade com POSIX (por exemplo, bloqueio de arquivos e modificações no local), considere o Amazon FSx for Lustre ou o OpenZFS como alternativas.
- AI/ML Treinamento e inferência em grande escala: ideais para cargas de trabalho com centenas ou milhares de nós de computação (por exemplo, GPUs em clusters EKS) em que a limitação de uso geral do S3 pode causar atrasos, desperdiçando recursos computacionais caros. Por exemplo, pesquisadores de LLM ou organizações que executam o modelo diário tests/checkpoints se beneficiam de um acesso rápido e confiável sem quebrar o S3 regional. Para cargas de trabalho de menor escala (por exemplo, dezenas de nós), o S3 Standard ou outras classes de armazenamento podem ser suficientes.
- Pipelines de dados: Load/prepare modelos, arquive dados de treinamento ou pontos de verificação de fluxo. Se sua equipe prefere o armazenamento de objetos em vez dos sistemas de arquivos tradicionais (por exemplo, devido à familiaridade com o S3), use isso em vez de fazer alterações de engenharia para opções compatíveis com POSIX, como o Lustre. FSx

Considerações

- Resiliência: o design Single-AZ fornece 99,999999999% de durabilidade (a mesma do S3 padrão, por meio de redundância dentro do AZ), mas menor disponibilidade (99,95% projetado, 99,9% de SLA) em comparação com as classes Multi-AZ (disponibilidade de 99,99%). É menos resistente às

falhas do AZ. Use para dados recriáveis ou armazenados em cache. Considere a replicação ou os backups Multi-AZ para cargas de trabalho críticas.

- Suporte de API e recursos: suporta um subconjunto do S3 APIs (por exemplo, sem políticas de ciclo de vida ou replicação); pode exigir pequenas alterações no aplicativo para autenticação de sessão ou manipulação de objetos.
- Integração com o EKS: coloque seu EKS pods/nodes na mesma AZ do bucket de diretórios para minimizar a latência da rede. Use o Mountpoint para Amazon S3 ou drivers CSI para acesso nativo do Kubernetes.
- Teste: teste a latência de recuperação em um ambiente de não produção para validar os ganhos de desempenho. Monitore a aceleração em cenários S3 padrão (por exemplo, alta saturação de GPU) e compare.

A classe de armazenamento S3 Express One Zone está disponível em várias regiões e se integra ao EKS para cargas de trabalho que precisam de acesso a objetos sem esperar pelo armazenamento. Para saber mais, consulte Introdução ao S3 Express One Zone.

Observabilidade

Monitoramento e observabilidade

Explicação das métricas da GPU

A métrica de utilização da GPU mostra se a GPU executou algum trabalho durante a janela de amostra. Essa métrica captura a porcentagem de tempo em que a GPU executou pelo menos uma instrução, mas não revela a eficiência com que a GPU usou seu hardware. Uma GPU contém vários Multiprocessadores de Streaming (SMs), que são as unidades de processamento paralelo que executam instruções. Uma leitura de 100% de utilização pode significar que a GPU executou cargas de trabalho paralelas pesadas em todas as suas SMs, ou pode significar que uma única pequena instrução ativou a GPU durante o período da amostra. Para entender a utilização real, você precisa examinar as métricas da GPU em vários níveis da arquitetura de hardware. Cada multiprocessador de streaming é construído a partir de diferentes tipos de núcleo, e cada camada expõe características de desempenho diferentes. Métricas de alto nível (utilização da GPU, utilização da memória, potência da GPU e temperatura da GPU, visíveis por meio da nvidia-smi) mostram se o dispositivo está ativo. Métricas mais detalhadas (utilização do SM, atividade do SM e uso do tensor core) revelam a eficiência com que a GPU usa seus recursos.

Almeje o alto uso de energia da GPU

A subutilização GPUs desperdiça a capacidade computacional e aumenta os custos porque as cargas de trabalho não conseguem engajar todos os componentes da GPU simultaneamente. Para AI/ML cargas de trabalho no Amazon EKS, monitore o uso de energia da GPU como proxy para identificar a atividade real da GPU. A utilização da GPU relata a porcentagem de tempo em que a GPU executa qualquer kernel, mas não revela se os multiprocessadores de streaming, os controladores de memória e os núcleos tensores estão todos ativos ao mesmo tempo. O uso de energia expõe essa lacuna porque o hardware totalmente engajado consome significativamente mais energia do que o hardware que executa núcleos leves ou fica ocioso entre as tarefas. Compare o consumo de energia com a potência de design térmico (TDP) da GPU para detectar a subutilização e, em seguida, investigue se sua carga de trabalho está limitada pelo pré-processamento da CPU, E/S de rede ou tamanhos de lote ineficientes.

Configure o CloudWatch Container Insights no Amazon EKS para identificar pods, nós ou cargas de trabalho com baixo consumo de energia da GPU. Essa ferramenta se integra diretamente ao Amazon EKS e permite monitorar o consumo de energia da GPU e ajustar o agendamento de pods ou os tipos de instância quando o uso de energia estiver abaixo dos níveis desejados. Se você precisar de visualização avançada ou painéis personalizados, use o DCGM-Exporter da NVIDIA com Prometheus e Grafana para monitoramento nativo do Kubernetes. Ambas as abordagens revelam as principais métricas da NVIDIA, como `nvidia_smi_power_draw` (consumo de energia da GPU) e `nvidia_smi_temperature_gpu` (temperatura da GPU). Para ver uma lista de métricas, explore https://docs.aws.amazon.com/AmazonCloudWatch/ latest/monitoring/CloudWatch o -Agent-NVIDIA-GPU.htm. Procure padrões como baixo consumo consistente de energia durante horários específicos ou para trabalhos específicos. Essas tendências ajudam a identificar onde consolidar as cargas de trabalho ou ajustar a alocação de recursos.

Limites estáticos de recursos no Kubernetes (como contagem de CPU, memória e GPU) geralmente levam ao provisionamento excessivo ou à subutilização, especialmente para cargas de trabalho dinâmicas, como inferência, em que a demanda flutua. AI/ML Analise suas tendências de utilização e consolide cargas de trabalho em menos. GPUs Certifique-se de que cada GPU alcance a utilização total antes de alocar outras. Essa abordagem reduz o desperdício e reduz os custos. Para obter orientação detalhada sobre como otimizar as estratégias de agendamento e compartilhamento, consulte as melhores práticas de computação e escalonamento automático do EKS

Observabilidade e métricas

Usando ferramentas de monitoramento e observabilidade para suas cargas de trabalho AI/ML

Os AI/ML serviços modernos exigem coordenação entre infraestrutura, modelagem e lógica de aplicativos. Os engenheiros da plataforma gerenciam a infraestrutura e a pilha de observabilidade. Eles coletam, armazenam e visualizam métricas. AI/ML os engenheiros definem métricas específicas do modelo e monitoram o desempenho sob cargas e distribuição de dados variáveis. Os desenvolvedores de aplicativos consomem APIs, encaminham solicitações e rastreiam métricas de nível de serviço e interações de usuários. Sem práticas unificadas de observabilidade, essas equipes trabalham em silos e perdem sinais críticos sobre a integridade e o desempenho do sistema. Estabelecer visibilidade compartilhada em todos os ambientes garante que todas as partes interessadas possam detectar problemas com antecedência e manter um serviço confiável.

A otimização de clusters do Amazon EKS para AI/ML cargas de trabalho apresenta desafios de monitoramento exclusivos, especialmente em relação ao gerenciamento de memória de GPU. Sem o monitoramento adequado, as organizações enfrentam erros out-of-memory (OOM), ineficiências de recursos e custos desnecessários. O monitoramento eficaz garante melhor desempenho, resiliência e custos mais baixos para os clientes da EKS. Use uma abordagem holística que combine três camadas de monitoramento. Primeiro, monitore as métricas granulares da GPU usando o NVIDIA DCGM Exporter para monitorar o uso de energia da GPU, a temperatura da GPU, a atividade do SM, a ocupação do SM e os erros de XID. Em segundo lugar, monitore estruturas de serviços de inferência, como Ray e vLLM, para obter informações distribuídas sobre a carga de trabalho por meio de suas métricas nativas. Em terceiro lugar, colete insights em nível de aplicativo para rastrear métricas personalizadas específicas para sua carga de trabalho. Essa abordagem em camadas oferece visibilidade desde a utilização do hardware até o desempenho do aplicativo.

Ferramentas e estruturas

Várias ferramentas e estruturas fornecem out-of-the-box métricas nativas para monitorar AI/ML cargas de trabalho. Essas métricas integradas eliminam a necessidade de instrumentação personalizada e reduzem o tempo de configuração. As métricas se concentram em aspectos de desempenho, como latência, taxa de transferência e geração de tokens, que são essenciais para o fornecimento de inferências e o benchmarking. O uso de métricas nativas permite que você comece a monitorar imediatamente sem criar canais de coleta personalizados.

- vLLM: um mecanismo de atendimento de alto rendimento para modelos de linguagem grandes (LLMs) que fornece métricas nativas, como latência de solicitações e uso de memória.
- Ray: uma estrutura de computação distribuída que emite métricas para cargas de trabalho de IA escaláveis, incluindo tempos de execução de tarefas e utilização de recursos.
- Hugging Face Text Generation Inference (TGI): um kit de ferramentas para implantação e exibição LLMs, com métricas integradas para desempenho de inferência.
- NVIDIA genai-perf: uma ferramenta de linha de comando para comparar modelos generativos de IA, medir a taxa de transferência, a latência e as métricas específicas do LLM, como solicitações concluídas em intervalos de tempo específicos.

Métodos de observabilidade

Recomendamos implementar quaisquer mecanismos adicionais de observabilidade de uma das seguintes formas.

CloudWatch Container Insights Se sua organização prefere ferramentas nativas da AWS com configuração mínima, recomendamos CloudWatch o Container Insights. Ele se integra ao NVIDIA DCGM Exporter para coletar métricas de GPU e oferece painéis pré-construídos para insights rápidos. Habilitado pela instalação do complemento CloudWatch Observability em seu cluster, o Container Insights implanta e gerencia o ciclo de vida do NVIDIA DCGM Exporter, que coleta métricas de GPU dos drivers da Nvidia e as expõe a. CloudWatch

Depois de instalar o Container Insights, detecta CloudWatch automaticamente a NVIDIA GPUs em seu ambiente e coleta métricas críticas de integridade e desempenho. Essas métricas aparecem em out-of-the-box painéis selecionados. Você também pode integrar o Ray e o vLLM com o CloudWatch uso do Unified CloudWatch Agent para enviar suas métricas nativas. Essa abordagem unificada simplifica a observabilidade em ambientes EKS e permite que as equipes se concentrem no ajuste do desempenho e na otimização de custos, em vez de criar uma infraestrutura de monitoramento.

Para obter uma lista completa das métricas disponíveis, consulte as métricas do Amazon EKS e do Kubernetes Container Insights. Para step-by-step obter orientação sobre a implementação do monitoramento de GPU, consulte Obtenha insights operacionais para cargas de trabalho de GPU NVIDIA usando o Amazon CloudWatch Container Insights. Para exemplos práticos de otimização da latência de inferência, consulte Otimizando a capacidade de resposta da IA: um guia prático para inferência otimizada para latência do Amazon Bedrock.

Prometheus e Grafana gerenciados Se sua organização precisa de painéis personalizados e recursos avançados de visualização, implante o Prometheus com o NVIDIA DCGM-Exporter e o

Grafana para monitoramento nativo do Kubernetes. O Prometheus coleta e armazena métricas de GPU do DCGM-Exporter, enquanto o Grafana fornece recursos flexíveis de visualização e alerta. Essa abordagem oferece mais controle sobre o design do painel e a retenção de métricas em comparação com o CloudWatch Container Insights.

Você pode estender essa pilha de monitoramento integrando estruturas de código aberto como Ray e vLLM Ray e vLLM para exportar suas métricas nativas para o Prometheus. Você também pode conectar o Grafana a uma fonte de dados AWS X-Ray para visualizar traços distribuídos e identificar gargalos de desempenho em todo o seu pipeline de inferência. Essa combinação fornece end-to-end visibilidade de métricas em nível de GPU por meio de fluxos de solicitação em nível de aplicativo.

Para step-by-step obter orientação sobre a implantação dessa pilha de monitoramento, consulte Monitoramento de cargas de trabalho de GPU no Amazon EKS usando serviços de código aberto gerenciados pela AWS.

Considere monitorar o treinamento básico e as métricas de ajuste fino

Monitore as principais métricas de treinamento para monitorar a integridade e o desempenho do seu cluster Amazon EKS e das cargas de trabalho de aprendizado de máquina em execução nele. As cargas de trabalho de treinamento têm requisitos de monitoramento diferentes das cargas de trabalho de inferência porque são executadas por longos períodos, consomem recursos de forma diferente e exigem visibilidade da convergência do modelo e da eficiência do pipeline de dados. As métricas abaixo ajudam a identificar gargalos, otimizar a alocação de recursos e garantir que os trabalhos de treinamento sejam concluídos com êxito. Para step-by-step obter orientação sobre a implementação dessa abordagem de monitoramento, consulte Introdução à observação de cargas de trabalho de aprendizado de máquina no Amazon EKS.

Métricas de uso de recursos

Monitore as métricas de uso de recursos para validar se seus recursos estão sendo consumidos adequadamente. Essas métricas ajudam a identificar gargalos e a causa raiz dos problemas de desempenho.

- CPU, memória, rede, potência da GPU e temperatura da GPU — monitore essas métricas para garantir que os recursos alocados atendam às demandas da carga de trabalho e identifiquem oportunidades de otimização. Monitore métricas como gpu_memory_usage_bytes para identificar padrões de consumo de memória e detectar picos de uso. Calcule percentis como o 95º percentil (P95) para entender as maiores demandas de memória durante o treinamento. Essa análise ajuda você a otimizar modelos e infraestrutura para evitar erros de OOM e reduzir custos.

- Ocupação do SM, atividade e FPxx atividade do SM - monitore essas métricas para entender como o recurso subjacente na GPU está sendo usado. Como regra geral, a meta de 0,8 para SM Activity.
- Utilização de recursos de nós e pods — acompanhe o uso de recursos no nível do nó e do pod para identificar a contenção de recursos e possíveis gargalos. Monitore se os nós se aproximam dos limites de capacidade, o que pode atrasar o agendamento do pod e atrasar os trabalhos de treinamento.
- Utilização de recursos em comparação com solicitações e limites — Compare o uso real dos recursos com as solicitações e limites configurados para determinar se seu cluster pode lidar com cargas de trabalho atuais e acomodar cargas de trabalho futuras. Essa comparação revela se você precisa ajustar as alocações de recursos para evitar erros de OOM ou desperdício de recursos.
- Métricas internas de estruturas de ML - Capture métricas internas de treinamento e convergência de estruturas de ML, como e. TensorFlow PyTorch Essas métricas incluem curvas de perda, taxa de aprendizado, tempo de processamento em lote e duração da etapa de treinamento. Visualize essas métricas usando ferramentas semelhantes TensorBoard para rastrear a convergência de modelos e identificar ineficiências de treinamento.

Métricas de desempenho do modelo

Monitore as métricas de desempenho do modelo para validar se seu processo de treinamento produz modelos que atendam à precisão e aos requisitos de negócios. Essas métricas ajudam a determinar quando interromper o treinamento, comparar as versões do modelo e identificar a degradação do desempenho.

- Exatidão, precisão, recuperação e pontuação F1 — acompanhe essas métricas para entender o desempenho do seu modelo nos dados de validação. Calcule a pontuação F1 em um conjunto de validação após cada período de treinamento para avaliar se o modelo está melhorando e quando atinge níveis de desempenho aceitáveis.
- Métricas específicas do negócio e KPIs — Defina e acompanhe métricas que medem diretamente o valor comercial de suas AI/ML iniciativas. Para um sistema de recomendação, acompanhe métricas como taxa de cliques ou taxa de conversão para garantir que o modelo gere os resultados comerciais pretendidos.
- Desempenho ao longo do tempo — compare as métricas de desempenho entre as versões do modelo e as execuções de treinamento para identificar tendências e detectar a degradação. Monitore se as versões mais recentes do modelo mantêm ou melhoram o desempenho em

comparação com os modelos básicos. Essa comparação histórica ajuda você a decidir se deve implantar novos modelos ou investigar problemas de treinamento.

Qualidade de dados e métricas de desvio

Monitore a qualidade dos dados e altere as métricas para garantir que seus dados de treinamento permaneçam consistentes e representativos. O desvio de dados pode fazer com que o desempenho do modelo diminua com o tempo, enquanto os problemas de qualidade dos dados podem impedir que os modelos converjam ou produzam resultados não confiáveis.

- Propriedades estatísticas dos dados de entrada — Acompanhe as propriedades estatísticas, como média, desvio padrão e distribuição dos recursos de entrada ao longo do tempo, para detectar desvios ou anomalias nos dados. Monitore se as distribuições de recursos mudam significativamente em relação aos seus dados básicos de treinamento. Por exemplo, se a média de um recurso crítico mudar em mais de dois desvios padrão, investigue se seu pipeline de dados mudou ou se a fonte de dados subjacente mudou.
- Detecção e alertas de desvios de dados — implemente mecanismos automatizados para detectar e alertar sobre problemas de qualidade de dados antes que eles afetem o treinamento. Use testes estatísticos, como o teste de Kolmogorov-Smirnov ou o teste qui-quadrado, para comparar as distribuições de dados atuais com seus dados de treinamento originais. Configure alertas quando os testes detectarem desvios significativos para que você possa treinar novamente os modelos com dados atualizados ou investigar problemas no pipeline de dados.

Métricas de latência e taxa de transferência

Monitore as métricas de latência e produtividade para identificar gargalos em seu pipeline de treinamento e otimizar a utilização de recursos. Essas métricas ajudam você a entender onde o tempo é gasto durante o treinamento e onde concentrar os esforços de otimização.

- End-to-End Latência dos canais de treinamento de ML — meça o tempo total para que os dados fluam por todo o seu pipeline de treinamento, desde a ingestão de dados até a atualização do modelo. Acompanhe essa métrica em todas as sessões de treinamento para identificar se as mudanças no pipeline melhoram ou diminuem o desempenho. A alta latência geralmente indica gargalos no carregamento de dados, no pré-processamento ou na comunicação de rede entre os nós.
- Rendimento e taxa de processamento do treinamento — Acompanhe o volume de dados que seu pipeline de treinamento processa por unidade de tempo para garantir a utilização eficiente dos

recursos. Monitore métricas como amostras processadas por segundo ou lotes concluídos por minuto. A baixa taxa de transferência em relação à capacidade do hardware sugere ineficiências no carregamento de dados, no pré-processamento ou na computação do modelo que desperdiçam ciclos de GPU.

- Latência de salvamento e restauração do ponto de verificação — Monitore o tempo necessário para salvar os pontos de verificação do modelo no armazenamento (S3 FSx, EFS) e restaurá-los na GPU ou na memória da CPU ao retomar trabalhos ou se recuperar de falhas. As operações lentas do ponto de verificação prolongam o tempo de recuperação do trabalho e aumentam os custos. Monitore o tamanho do ponto de verificação, a duração do salvamento, a duração da restauração e a contagem de falhas para identificar oportunidades de otimização, como compactação ou níveis de armazenamento mais rápidos.
- Tempo de carregamento e pré-processamento de dados - meça o tempo gasto carregando dados do armazenamento e aplicando transformações de pré-processamento. Compare esse tempo com o tempo de computação do modelo para determinar se seu treinamento está vinculado a dados ou à computação. Se o carregamento de dados consumir mais de 20% do tempo total de treinamento, considere otimizar seu pipeline de dados com armazenamento em cache, pré-busca ou armazenamento mais rápido.

Taxas de erro e falhas

Monitore taxas de erro e falhas em todo o pipeline de treinamento para manter a confiabilidade e evitar o desperdício de recursos computacionais. Erros não detectados podem fazer com que os trabalhos de treinamento falhem silenciosamente, produzam modelos inválidos ou desperdicem horas de tempo de GPU antes que você perceba problemas.

- Monitoramento de erros do pipeline — acompanhe os erros em todos os estágios do seu pipeline de ML, incluindo pré-processamento de dados, treinamento de modelos e operações de ponto de verificação. Registre os tipos de erros, as frequências e os componentes afetados para identificar problemas rapidamente. Os erros comuns incluem incompatibilidades de formato de dados, out-of-memory falhas durante o pré-processamento e falhas de salvamento no ponto de verificação devido aos limites de armazenamento. Configure alertas quando as taxas de erro excederem os limites básicos para que você possa investigar antes que os erros ocorram em cascata.
- Análise de erros recorrentes — identifique e investigue padrões em erros recorrentes para evitar futuras falhas e melhorar a confiabilidade do pipeline. Analise os registros para descobrir se amostras de dados específicas, tamanhos de lotes ou configurações de treinamento causam falhas de forma consistente. Por exemplo, se determinados tipos de dados de entrada acionarem

erros de pré-processamento, adicione verificações de validação no início do pipeline ou atualize sua lógica de limpeza de dados. Acompanhe o tempo médio entre falhas (MTBF) para medir se a confiabilidade da tubulação melhora com o tempo. ”

Métricas específicas do Kubernetes e do EKS

Monitore as métricas do Kubernetes e do EKS para garantir que sua infraestrutura de cluster permaneça saudável e possa suportar suas cargas de trabalho de treinamento. Essas métricas ajudam você a detectar problemas de infraestrutura antes que eles causem falhas nas tarefas de treinamento ou degradação do desempenho.

- Métricas do estado do cluster do Kubernetes — monitore a integridade e o status dos objetos do Kubernetes, incluindo pods, nós, implantações e serviços. Acompanhe o status dos pods para identificar os pods presos em estados de loop pendente, com falha ou de falha. Monitore as condições dos nós para detectar problemas como pressão do disco, pressão da memória ou indisponibilidade da rede. Use o kubectl ou as ferramentas de monitoramento para verificar essas métricas continuamente e configurar alertas quando os pods falharem na inicialização ou se tornarem não programáveis.
- Métricas de execução do pipeline de treinamento — Acompanhe execuções bem-sucedidas e malsucedidas do pipeline, durações de trabalhos, tempos de conclusão de etapas e erros de orquestração. Monitore se os trabalhos de treinamento são concluídos dentro dos prazos esperados e se as taxas de falha aumentam com o tempo. Monitore métricas como taxa de sucesso no trabalho, duração média do trabalho e tempo até o fracasso. Essas métricas ajudam a identificar se problemas de infraestrutura, problemas de configuração ou problemas de qualidade de dados causam falhas no treinamento.
- Métricas de serviço da AWS — Acompanhe métricas para serviços da AWS que suportam sua infraestrutura EKS e cargas de trabalho de treinamento. Monitore as métricas do S3, como latência da solicitação, taxas de erro e taxa de transferência, para garantir que o desempenho do carregamento de dados permaneça consistente. Acompanhe as métricas de volume do EBS, incluindo IOPS, taxa de transferência e tamanho da fila para detectar gargalos de armazenamento. Monitore os registros de fluxo da VPC e as métricas de rede para identificar problemas de conectividade entre os nós ou com serviços externos.
- Métricas do plano de controle do Kubernetes — monitore o servidor da API, o agendador, o gerenciador do controlador e o banco de dados etcd para detectar problemas ou falhas de desempenho que afetam as operações do cluster. Acompanhe a latência, a taxa de solicitação e a taxa de erro da solicitação do servidor de API para garantir que o plano de controle responda

rapidamente às solicitações de agendamento. Monitore o tamanho do banco de dados etcd, a duração do commit e as mudanças de liderança para detectar problemas de estabilidade. A alta latência do servidor de API ou as frequentes mudanças no líder do etcd podem atrasar o agendamento do pod e estender os tempos de inicialização dos trabalhos de treinamento.

Registros de aplicativos e instâncias

Colete e analise registros de aplicativos e instâncias para diagnosticar problemas que as métricas sozinhas não conseguem explicar. Os registros fornecem um contexto detalhado sobre erros, mudanças de estado e eventos do sistema que ajudam você a entender por que os trabalhos de treinamento falham ou têm um desempenho ruim. A correlação de registros com métricas permite que você identifique as causas-raiz mais rapidamente.

- Registros de aplicativos — colete registros de aplicativos de seus trabalhos de treinamento, pipelines de dados e estruturas de ML para identificar gargalos e diagnosticar falhas. Esses registros capturam informações detalhadas sobre a execução do trabalho, incluindo erros de carregamento de dados, falhas na inicialização do modelo, erros de salvamento do ponto de verificação e avisos específicos da estrutura. Correlacione registros de data e hora com picos de métricas para entender o que causou degradação ou falhas no desempenho. Por exemplo, se a utilização da GPU cair repentinamente, verifique se há erros nos registros do aplicativo que indiquem paralisações no pipeline de dados ou falhas no pré-processamento. Use ferramentas de registro centralizadas, como CloudWatch Logs ou Fluent Bit, para agregar registros de todos os pods e torná-los pesquisáveis.
- Registros de instância - Colete registros no nível da instância, como registros do diário do sistema e saída dmesg, para detectar problemas de hardware e problemas no nível do kernel. Esses registros revelam problemas como erros de driver de GPU, falhas de alocação de memória, I/O erros de disco e problemas de interface de rede que podem não aparecer nos registros do aplicativo. Correlacione registros de instâncias com registros e métricas de aplicativos para determinar se as falhas de treinamento resultam de problemas de hardware ou problemas de aplicativos. Por exemplo, se um trabalho de treinamento falhar com um out-of-memory erro, verifique os registros do dmesg em busca de mensagens do kernel OOM killer que indiquem se o sistema ficou sem memória ou se o aplicativo excedeu seus limites de contêiner. Configure alertas para erros críticos de hardware, como erros de GPU XID ou falhas de disco, para que você possa substituir instâncias com falha antes que elas causem interrupções generalizadas no treinamento.

As seções a seguir mostram como coletar as métricas descritas acima usando duas abordagens recomendadas pela AWS: CloudWatch Container Insights e Amazon Managed Prometheus Amazon Managed Prometheus com Amazon Managed Grafana. Escolha o CloudWatch Container Insights se você preferir ferramentas nativas da AWS com configuração mínima e painéis pré-criados. Escolha o Amazon Managed Prometheus com o Amazon Managed Grafana se você precisar de painéis personalizados, recursos avançados de visualização ou quiser se integrar à infraestrutura de monitoramento existente baseada no Prometheus. Para obter uma lista completa das métricas disponíveis do Container Insights, consulte as métricas do Amazon EKS e do Kubernetes Container Insights.

Considere monitorar métricas de inferência on-line em tempo real

Em sistemas em tempo real, a baixa latência é fundamental para fornecer respostas oportunas aos usuários ou outros sistemas dependentes. A alta latência pode degradar a experiência do usuário ou violar os requisitos de desempenho. Os componentes que influenciam a latência da inferência incluem tempo de carregamento do modelo, tempo de pré-processamento, tempo real de previsão, tempo de pós-processamento e tempo de transmissão da rede. Recomendamos monitorar a latência de inferência para garantir respostas de baixa latência que atendam aos contratos de nível de serviço (SLAs) e desenvolver métricas personalizadas para o seguinte. Teste sob a carga esperada, inclua a latência da rede, contabilize solicitações simultâneas e teste com tamanhos de lote variados.

- Time to First Token (TTFT) — Tempo desde o momento em que um usuário envia uma solicitação até receber o início de uma resposta (a primeira palavra, token ou fragmento). Por exemplo, em chatbots, você verificaria quanto tempo leva para gerar a primeira parte da saída (token) depois que o usuário faz uma pergunta.
- End-to-End Latência — Esse é o tempo total desde o recebimento de uma solicitação até o envio da resposta. Por exemplo, meça o tempo entre a solicitação e a resposta.
- Tokens de saída por segundo (TPS) — Indica a rapidez com que seu modelo gera novos tokens depois de começar a responder. Por exemplo, em chatbots, você acompanharia a velocidade de geração de modelos de linguagem para um texto básico.
- Taxa de erro — rastreia solicitações com falha, o que pode indicar problemas de desempenho. Por exemplo, monitore solicitações malsucedidas de documentos grandes ou determinados caracteres.
- Rendimento — meça o número de solicitações ou operações que o sistema pode processar por unidade de tempo. Por exemplo, rastreie solicitações por segundo para lidar com 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/Vou um cache de sessão (uso):

- hit/miss Proporção de cache — Para configurações de inferência que utilizam o armazenamento em cache (K/V ou caches incorporados), meça a frequência com que o cache está ajudando. As baixas taxas de acerto podem indicar alterações na configuração do cache ou na carga de trabalho abaixo do ideal, o que pode aumentar a latência.

Nos tópicos subsequentes, demonstraremos a coleta de dados para algumas das métricas mencionadas acima. Forneceremos exemplos com as duas abordagens recomendadas pela AWS: _CloudWatch Container Insights nativo da AWS e Amazon Managed Prometheus de código aberto com Amazon Managed Grafana. Você escolheria uma dessas soluções com base em suas necessidades gerais de observabilidade. Consulte as métricas do Amazon EKS e do Kubernetes Container Insights para obter a lista completa das métricas do Container Insights.

Rastreando o uso da memória da GPU

Conforme discutido no the section called “Considere monitorar o treinamento básico e as métricas de ajuste fino” tópico, o uso da memória da GPU é essencial para evitar erros out-of-memory (OOM) e garantir a utilização eficiente dos recursos. Os exemplos a seguir mostram como instrumentar seu aplicativo de treinamento para expor uma métrica de histograma personalizada e calcular o uso da memória P95 para identificar o pico de consumo. `gpu_memory_usage_bytes` Certifique-se de testar com um exemplo de trabalho de treinamento (por exemplo, ajuste fino de um modelo de transformador) em um ambiente de teste.

Exemplo de insights de CloudWatch contêiner nativos da AWS

Este exemplo demonstra como instrumentar seu aplicativo de treinamento para expor `gpu_memory_usage_bytes` como um histograma usando a abordagem nativa da AWS. Observe que seu AI/ML contêiner deve ser configurado para emitir registros estruturados no formato CloudWatch Embedded Metrics Format (EMF). CloudWatch logs, analisa o EMF e publica as métricas. Use aws_embedded_metrics em seu aplicativo de treinamento para enviar registros estruturados no formato EMF para o Logs, que extrai as métricas da 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()
```

Exemplo de Prometheus e Grafana

Este exemplo demonstra como instrumentar seu aplicativo de treinamento para expor `gpu_memory_usage_bytes` como um histograma usando a biblioteca cliente Prometheus em 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()
```

Rastreie a duração da solicitação de inferência para inferência on-line em tempo real

Conforme discutido no the section called “Considere monitorar o treinamento básico e as métricas de ajuste fino” tópico, a baixa latência é fundamental para fornecer respostas oportunas aos usuários ou a outros sistemas dependentes. Os exemplos a seguir mostram como rastrear uma métrica de histograma personalizada,`inference_request_duration_seconds`, exposta por seu aplicativo de inferência. Calcule a latência do 95º percentil (P95) para se concentrar nos piores cenários, teste com solicitações de inferência sintética (por exemplo, via Locust) em um ambiente de teste e defina limites de alerta (por exemplo, >500 ms) para detectar violações de SLA.

Exemplo de insights de CloudWatch contêiner nativos da AWS

Este exemplo demonstra como criar uma métrica de histograma personalizada em seu aplicativo de inferência para inference_request_duration_seconds usando o AWS Embedded Metric Format. 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()
```

Exemplo de Prometheus e Grafana

Este exemplo demonstra como criar uma métrica de histograma personalizada em seu aplicativo de inferência para inference_request_duration_seconds usando a biblioteca de cliente Prometheus em 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)
```

No Grafana, use a consulta `histogram_quantile(0.95, sum(rate(inference_request_duration_seconds_bucket[5m])) by (le, pod))` para visualizar as tendências de latência do P95. Para saber mais, consulte a documentação do histograma do Prometheus e a documentação do cliente do Prometheus.

Rastreie a taxa de transferência de tokens para inferência on-line em tempo real

Conforme discutido no the section called “Considere monitorar o treinamento básico e as métricas de ajuste fino” tópico, recomendamos monitorar o tempo de processamento do token para avaliar o desempenho do modelo e otimizar as decisões de escalabilidade. Os exemplos a seguir mostram como rastrear uma métrica de histograma personalizada,`token_processing_duration_seconds`, exposta por seu aplicativo de inferência. Calcule a duração do 95º percentil (P95) para analisar a eficiência do processamento, testar com cargas de solicitações simuladas (por exemplo, 100 a 1000 solicitações/segundo) em um cluster que não seja de produção e ajuste os acionadores KEDA para otimizar o escalonamento.

Exemplo de insights de CloudWatch contêiner nativos da AWS

Este exemplo demonstra como criar uma métrica de histograma personalizada em seu aplicativo de inferência para token_processing_duration_seconds usando o AWS Embedded Metric Format. CloudWatch Ele usa a `) with a custom `get_duration_bucket` função dimensions (`set_dimension) para categorizar durações em compartimentos (por exemplo, “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
```

Exemplo de Prometheus e Grafana

Este exemplo demonstra como criar uma métrica de histograma personalizada em seu aplicativo de inferência para token_processing_duration_seconds usando a biblioteca de cliente Prometheus em 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)
```

No Grafana, use a consulta `histogram_quantile(0.95, sum(rate(token_processing_duration_seconds_bucket[5m])) by (le, pod))` para visualizar as tendências do tempo de processamento do P95. Para saber mais, consulte a documentação do histograma do Prometheus e a documentação do cliente do Prometheus.

Meça a latência de restauração do Checkpoint

Conforme discutido no the section called “Considere monitorar o treinamento básico e as métricas de ajuste fino” tópico, a latência do ponto de verificação é uma métrica crítica durante várias fases do ciclo de vida do modelo. Os exemplos a seguir mostram como rastrear uma métrica de histograma personalizada,`checkpoint_restore_duration_seconds`, exposta pelo seu aplicativo. Calcule a duração do 95º percentil (P95) para monitorar o desempenho da restauração, testar com interrupções pontuais em um cluster de não produção e definir limites de alerta (por exemplo, <30 segundos) para detectar atrasos.

Exemplo de insights de CloudWatch contêiner nativos da AWS

Este exemplo demonstra como instrumentar seu aplicativo em lote para expor checkpoint_restore_duration_seconds como um histograma usando o 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")
```

Exemplo de Prometheus e Grafana

Este exemplo demonstra como instrumentar seu aplicativo em lote para expor `checkpoint_restore_duration_seconds` como um histograma usando a biblioteca cliente Prometheus em 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"))
```

No Grafana, use a consulta `histogram_quantile(0.95, sum(rate(checkpoint_restore_duration_seconds_bucket[5m]) by (le))` para visualizar as tendências de latência de restauração do P95. Para saber mais, consulte a documentação do histograma do Prometheus e a documentação do cliente do Prometheus.

Escalabilidade e desempenho de aplicativos

Gerenciamento de artefatos de ML, estruturas de serviço e otimização de startups

A implantação de modelos de aprendizado de máquina (ML) no Amazon EKS exige uma análise cuidadosa de como os modelos são integrados às imagens de contêineres e aos ambientes de execução. Isso garante escalabilidade, reprodutibilidade e utilização eficiente dos recursos. Este tópico descreve as diferentes abordagens para lidar com artefatos do modelo de ML, selecionar estruturas de serviço e otimizar os tempos de inicialização de contêineres por meio de técnicas como pré-armazenamento em cache, todas adaptadas para reduzir os tempos de inicialização de contêineres.

Lidando com artefatos do modelo de ML em implantações

Uma decisão importante é como lidar com os próprios artefatos do modelo de ML (como pesos e configurações). A escolha afeta o tamanho da imagem, a velocidade de implantação, a frequência

de atualização do modelo e a sobrecarga operacional. Observe que, ao nos referirmos ao armazenamento do “modelo”, estamos nos referindo aos artefatos do modelo (como parâmetros treinados e pesos do modelo). Existem diferentes abordagens para lidar com artefatos do modelo de ML no Amazon EKS. Cada um tem suas vantagens e desvantagens, e a melhor depende do tamanho, da cadência de atualização e das necessidades de infraestrutura do seu modelo. Considere as seguintes abordagens, da menos à mais recomendada:

- Inserindo o modelo na imagem do contêiner: copie os arquivos do modelo (por exemplo, .safetensors, .pth, .h5) na imagem do contêiner (por exemplo, Dockerfile) durante a criação da imagem. O modelo faz parte da imagem imutável. Recomendamos usar essa abordagem para modelos menores com atualizações pouco frequentes. Isso garante consistência e reprodutibilidade, não atrasa o carregamento e simplifica o gerenciamento de dependências, mas resulta em tamanhos de imagem maiores, reduz a velocidade de compilação e envio, requer reconstrução e reimplantação para atualizações de modelos e não é ideal para modelos grandes devido à taxa de transferência do registro.
- Baixando o modelo em tempo de execução: na inicialização do contêiner, o aplicativo baixa o modelo do armazenamento externo (por exemplo, Amazon S3, apoiado pelo S3 CRT para transferências otimizadas de alto rendimento usando métodos como Mountpoint for S3 CSI driver, AWS S3 CLI ou s5cmd OSS CLI) por meio de scripts em um contêiner inicial ou ponto de entrada. Recomendamos começar com essa abordagem para modelos grandes com atualizações frequentes. Isso mantém as imagens do contêiner focadas no código/tempo de execução, permite atualizações fáceis do modelo sem reconstruções, oferece suporte ao controle de versão por meio de metadados de armazenamento, mas introduz possíveis falhas de rede (requer lógica de repetição), requer autenticação e armazenamento em cache.

Para saber mais, consulte Acelerando o processo de extração no AI on EKS Workshop.

Servindo modelos de ML

A implantação e a distribuição de modelos de aprendizado de máquina (ML) no Amazon EKS exigem a seleção de uma abordagem apropriada de distribuição de modelos para otimizar a latência, a taxa de transferência, a escalabilidade e a simplicidade operacional. A escolha depende do tipo de modelo (por exemplo, linguagem, modelo de visão), das demandas de carga de trabalho (por exemplo, inferência em tempo real) e da experiência da equipe. As abordagens comuns incluem configurações baseadas em Python para prototipagem, servidores de modelos dedicados para recursos de nível de produção e mecanismos de inferência especializados para alto desempenho e eficiência. Cada método envolve compensações na complexidade da configuração, no desempenho

e na utilização de recursos. Observe que as estruturas de serviço podem aumentar o tamanho das imagens de contêineres (múltiplas GBs) devido a dependências, potencialmente afetando os tempos de inicialização. Considere a desacoplamento usando técnicas de manipulação de artefatos para mitigar isso. As opções estão listadas da menos para a mais recomendada:

Usando estruturas Python (por exemplo, FastAPI, HuggingFace Transformers with PyTorch) Desenvolva um aplicativo personalizado usando estruturas Python, incorporando arquivos de modelo (pesos, configuração, tokenizador) em uma configuração de nó em contêiner.

- Prós: prototipagem fácil, somente em Python sem infraestrutura extra, compatível com todos os HuggingFace modelos, implantação simples do Kubernetes.
- Contras: restringe a um único request/simple lote, geração lenta de tokens (sem kernels otimizados), memória ineficiente, falta escalabilidade e monitoramento e envolve longos tempos de inicialização.
- Recomendação: use para prototipagem inicial ou tarefas de nó único que exijam integração lógica personalizada.

Usando estruturas dedicadas de fornecimento de modelos (por exemplo, TensorRT-LLM, TGI) Adote servidores especializados, como TensorRT-LLM ou TGI, para inferência de ML, gerenciamento de carregamento, roteamento e otimização de modelos. Esses formatos de suporte, como sensores de segurança, com compilação ou plug-ins opcionais.

- Vantagens: Oferece agrupamento em lotes (static/in-flight or continuous), quantization (INT8, FP8, GPTQ), hardware optimizations (NVIDIA, AMD, Intel, Inferentia), and multi-GPU support (Tensor/Pipelineparalelismo). O Tensorrt-LLM oferece suporte a diversos modelos (LLMs, Encoder-Decoder), enquanto o TGI aproveita a integração. HuggingFace
- Contras: o TensorRT-LLM precisa de compilação e é exclusivo para NVIDIA; o TGI pode ser menos eficiente em lotes; ambos adicionam sobrecarga de configuração e podem não se adequar a todos os tipos de modelos (por exemplo, não transformadores).
- Recomendação: Adequado para PyTorch/TensorFlow modelos que precisam de recursos de produção, como A/B testes ou alto rendimento com hardware compatível.

Usando mecanismos de inferência especializados de alto rendimento (por exemplo, vLLM) Utilize mecanismos de inferência avançados como o vLLM, otimizando o serviço de LLM com lotes em andamento e quantização (, -KV, AWQ) PagedAttention, integráveis ao escalonamento automático do EKS. INT8 FP8

- Prós: alto rendimento e eficiência de memória (40 a 60% de economia de VRAM), tratamento dinâmico de solicitações, streaming de tokens, suporte a várias GPUs Tensor Parallel de nó único e ampla compatibilidade de hardware.
- Contras: Otimizado para transformadores somente para decodificadores (por exemplo, LLa MA), menos eficaz para modelos sem transformadores, requer hardware compatível (por exemplo, GPUs NVIDIA) e esforço de configuração.
- Recomendação: A melhor opção para inferência LLM de alto volume e baixa latência no EKS, maximizando a escalabilidade e o desempenho.

Otimizando os tempos de extração da imagem do contêiner

Imagens de contêineres grandes podem causar atrasos na inicialização a frio que afetam a latência de inicialização do pod. Para cargas de trabalho sensíveis à latência, como cargas de trabalho de inferência em tempo real escaladas horizontalmente, a inicialização rápida do pod é fundamental. Considere as seguintes abordagens para otimizar os tempos de extração da imagem do contêiner:

Reduzindo tamanhos de imagens de contêiner

Reduzir o tamanho das imagens do contêiner durante a inicialização é outra forma de tornar as imagens menores. Você pode fazer reduções em cada etapa do processo de criação da imagem do contêiner. Para começar, escolha imagens básicas que contenham o menor número de dependências necessárias. Durante a criação de imagens, inclua somente as bibliotecas e artefatos essenciais necessários. Ao criar imagens, tente combinar vários `RUN` `COPY` comandos para criar um número menor de camadas maiores. Para AI/ML estruturas, use compilações de vários estágios para separar a compilação e o tempo de execução, copiando somente os artefatos necessários (por exemplo, `COPY –from=` por meio de registros ou contextos locais) e selecione variantes, como imagens somente em tempo de execução (por exemplo, `pytorch/pytorch:2.7.1-cuda11.8-cudnn9-runtime` em 3,03 GB versus desenvolvimento em 6,66 GB). Para saber mais, consulte Redução do tamanho da imagem do contêiner no AI on EKS Workshop.

Usando o snapshotter SOCI para pré-extrair imagens

Para imagens muito grandes que você não pode minimizar facilmente, você pode usar o snapshotter Seekable OCI (SOCI) de código aberto configurado no modo pull and unpack paralelo. Essa solução permite que você use imagens existentes sem reconstruir ou modificar seus pipelines de criação. Essa opção é especialmente eficaz ao implantar cargas de trabalho com imagens muito grandes em instâncias de EC2 computação de alto desempenho. Ele funciona bem com redes de alto rendimento

e configurações de armazenamento de alto desempenho, como é típico com cargas de trabalho escalonadas. AI/ML

O pull/unpack modo paralelo SOCI melhora o desempenho de extração de end-to-end imagem por meio de estratégias de paralelização configuráveis. A coleta e a preparação mais rápidas de imagens afetam diretamente a rapidez com que você pode implantar novas cargas de trabalho e escalar seu cluster com eficiência. A extração de imagens tem duas fases principais:

1. Buscando camadas do registro para o nó

   Para otimizar a busca por camada, o SOCI cria várias conexões HTTP simultâneas por camada, multiplicando a taxa de transferência de download além da limitação de conexão única. Ele divide grandes camadas em partes e as baixa simultaneamente em várias conexões. Essa abordagem ajuda a saturar a largura de banda de rede disponível e a reduzir significativamente os tempos de download. Isso é particularmente valioso para AI/ML cargas de trabalho em que uma única camada pode ter vários gigabytes.

2. Desembalando e preparando essas camadas para criar contêineres

   Para otimizar o desempacotamento de camadas, o SOCI processa várias camadas simultaneamente. Em vez de esperar que cada camada seja totalmente descompactada antes de iniciar a próxima, ele usa os núcleos de CPU disponíveis para descompactar e extrair várias camadas simultaneamente. Esse processamento paralelo transforma a fase de desempacotamento tradicionalmente vinculada à E/S em uma operação otimizada para CPU que se adapta aos núcleos disponíveis. O sistema orquestra cuidadosamente essa paralelização para manter a consistência do sistema de arquivos e, ao mesmo tempo, maximizar a taxa de transferência.

O modo SOCI parallel pull usa um sistema de controle de limite duplo com parâmetros configuráveis para simultaneidade de download e paralelismo de descompactação. Esse controle granular permite que você ajuste o comportamento do SOCI para atender aos seus requisitos específicos de desempenho e condições ambientais. A compreensão desses parâmetros ajuda você a otimizar seu tempo de execução para obter o melhor desempenho de extração.

Referências

- Para obter mais informações sobre a solução e as vantagens e desvantagens do ajuste, consulte a documentação do recurso no repositório do projeto SOCI em. GitHub

- Para ver um exemplo prático com o Karpenter no Amazon EKS, consulte o Karpenter Blueprint usando o modo paralelo do snapshotter SOCI. pull/unpack
- Para obter informações sobre como configurar o Bottlerocket para pull paralelo, consulte soci-snapshotter Parallel Pull Unpack Mode na documentação do Bottlerocket.o

Usando instantâneos do EBS para pré-extrair imagens

Você pode tirar um snapshot do Amazon Elastic Block Store (EBS) de imagens de contêineres em cache e reutilizar esse snapshot para os nós de trabalho do EKS. Isso garante que as imagens sejam pré-buscadas localmente na inicialização do nó, reduzindo o tempo de inicialização do pod. Consulte Reduzir o tempo de inicialização do contêiner no Amazon EKS com o volume de dados do Bottlerocket para obter mais informações sobre o uso do Karpenter e do EKS Terraform Blueprints para grupos de nós gerenciados.

Para saber mais, consulte Usando snapshotter containerd e Pré-carregar imagens de contêiner em volumes de dados do Bottlerocket com instantâneos do EBS no AI on EKS Workshop.

Usando o Container Runtime Cache para pré-extrair imagens

Você pode pré-inserir imagens de contêiner em nós usando recursos do Kubernetes (por exemplo, DaemonSet ou implantação) para preencher o cache de tempo de execução do contêiner do nó. O cache de tempo de execução do contêiner é o armazenamento local gerenciado pelo tempo de execução do contêiner (por exemplo, containerd), onde as imagens são armazenadas após serem retiradas de um registro. A pré-extração garante que as imagens estejam disponíveis localmente, evitando atrasos no download durante a inicialização do pod. Essa abordagem é particularmente útil quando as imagens mudam com frequência (por exemplo, atualizações frequentes), quando os instantâneos do EBS não estão pré-configurados, quando a criação de um volume do EBS seria mais demorada do que a extração direta de um registro de contêiner ou quando os nós já estão no cluster e precisam ativar pods sob demanda usando uma das várias imagens possíveis.

A pré-extração de todas as variantes garante um tempo de inicialização rápido, independentemente da imagem necessária. Por exemplo, em uma carga de trabalho de ML massivamente paralela que exige 100.000 modelos pequenos criados usando 10 técnicas diferentes, a pré-extração de 10 imagens DaemonSet em um grande cluster (por exemplo, milhares de nós) minimiza o tempo de inicialização do pod, permitindo a conclusão em menos de 10 segundos, evitando solicitações sob demanda. O uso da abordagem de cache de tempo de execução de contêiner elimina a necessidade de gerenciar instantâneos do EBS, garante que você sempre obtenha a versão mais recente da imagem do contêiner DaemonSets, mas para cargas de trabalho de inferência em tempo real

em que os nós escalam 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 padrões, isso pode despejar imagens em nós ociosos, o que requer novas extraições durante escalonamentos subsequentes e reduz a confiabilidade do cache para cargas de trabalho intermitentes.

Consulte o GitHub repositório da AWS para ver exemplos de pré-extração de imagens para o cache de tempo de execução do contêiner.

Considere o NVMe armazenamento de kubelet e containerd

Considere configurar `kubelet` e `containerd` usar discos de armazenamento de NVMe instância efêmera para aumentar o desempenho do disco. O processo de extração do contêiner envolve o download de uma imagem do contêiner de um registro e a descompactação de suas camadas em um formato utilizável. Para otimizar I/O as operações durante a descompactação, você deve avaliar o que fornece níveis mais altos de I/O desempenho e taxa de transferência para o tipo de instância do seu host de contêiner: NVMe instâncias de backup com armazenamento local versus IOPs/throughput de volume do EBS. Para EC2 instâncias com armazenamento NVMe local, considere configurar o sistema de arquivos subjacente do nó para kubelet (`/var/lib/kubelet`), containerd () e Pod logs `/var/log/pods` () para usar discos de armazenamento de instâncias `/var/lib/containerd` NVMe efêmeros para obter níveis mais altos de desempenho e taxa de transferência. I/O

O armazenamento efêmero do nó pode ser compartilhado entre pods que solicitam armazenamento efêmero e imagens de contêiner que são baixadas para o nó. Se estiver usando o Karpenter com Bottlerocket ou AL2 023 EKS Optimized, AMIs isso pode ser configurado no EC2NodeClassconfigurando instanceStorePolicy como RAID0 ou, se estiver usando grupos de nós gerenciados, definindo o in como parte dos dados do usuário. localStoragePolicy NodeConfig

Contribua com este guia

Qualquer pessoa pode contribuir com o guia de melhores práticas. O Guia de Boas Práticas do EKS está escrito no AsciiDoc formato em GitHub.

Resumo para colaboradores existentes

- Abra o `bpg-docs.code-workspace`com o VS Code para instalar automaticamente a AsciiDoc extensão.
  - Saiba mais sobre a AsciiDoc extensão no Visual Studio Marketplace.
- Os arquivos de origem do site do AWS Docs são armazenados em `latest/bpg`
- A sintaxe é muito semelhante ao markdown.
  - Analise a referência de sintaxe nos AsciiDoctor documentos.
- A plataforma docs só é `latest/bpg/images` implantada. Cada uma das seções do guia tem um link simbólico para esse diretório. Por exemplo, `latest/bpg/networking/images` aponta para`latest/bpg/images`.

Configurar um ambiente de edição local

Se você planeja editar o guia com frequência, configure um ambiente de edição local.

Bifurque e clone o repositório

Você precisa estar familiarizado com`git`,`github`, e editores de texto. Para obter informações sobre como começar a usar `git` e`github`, consulte Introdução à sua GitHub conta nos GitHub documentos.

1. Veja o Guia de Melhores Práticas do EKS em GitHub.
2. Crie uma bifurcação do repositório do projeto. Saiba como bifurcar um repositório nos GitHub documentos.
3. Clone sua bifurcação do repositório do projeto. Saiba como clonar seu repositório bifurcado.

Abra o espaço de trabalho do VS Code

A AWS recomenda usar o Visual Studio Code da Microsoft para editar o guia. Para obter mais informações sobre o VS Code, consulte Baixar o Visual Studio Code e começar a usar o Visual Studio Code na documentação do Visual Studio Code.

1. Abra o VS Code.
2. Abra o `bpg-docs.code-workspace` arquivo do repositório clonado.
3. Se for a primeira vez que você abre esse espaço de trabalho, aceite a solicitação para instalar a AsciiDoc extensão. Essa extensão verifica a sintaxe dos AsciiDoc arquivos e gera uma visualização ao vivo.
4. Navegue até o `latest/bpg` diretório. Esse diretório contém os arquivos de origem que são implantados no site de documentação da AWS. Os arquivos de origem são organizados por seção de guia, como “segurança” ou “rede”.

Editar um arquivo

1. Abra um arquivo no editor.
   - Veja a AsciiDoc sintaxe para aprender a criar cabeçalhos, links e listas.
   - Você pode usar a sintaxe Markdown para formatar texto, criar listas e cabeçalhos. Você não pode usar a sintaxe Markdown para criar links.
2. Abra uma pré-visualização ao vivo da página.
   - Primeiro, pressione `ctrl-k` ou `cmd-k` (dependendo do teclado). Em segundo lugar, pressione`v`. Isso abre uma visualização prévia na visualização dividida.

A AWS sugere o uso de ramificações de recursos para organizar suas alterações. Aprenda a criar ramificações com o git.

Enviar um Pull Request

Você pode criar uma pull request no GitHub site ou na GitHub CLI.

Saiba como criar uma pull request a partir de uma bifurcação usando o GitHub site.

Saiba como criar uma pull request usando a GitHub cli.

Use o editor baseado na web github.dev

O editor `github.dev` baseado na web é baseado no VS Code. Essa é uma ótima maneira de editar vários arquivos e visualizar conteúdo sem nenhuma configuração.

Ele tem suporte para a AsciiDoc extensão. Você pode fazer operações do git usando a GUI. O editor baseado na web não tem um shell ou terminal para executar comandos.

Você deve ter uma GitHub conta. Você será solicitado a fazer login, se necessário.

Inicie o editor GitHub baseado na web.

Editar uma única página

Você pode atualizar rapidamente páginas individuais usando GitHub o. Cada página contém um link "# Editar esta página em GitHub" na parte inferior.

1. Navegue até a página deste guia que você deseja editar
2. Clique no link “Editar esta página em GitHub” na parte inferior
3. Clique no ícone de lápis de edição no canto superior direito do visualizador de GitHub arquivos ou pressione e
4. Edite o arquivo
5. Envie suas alterações usando o botão “Confirmar alterações...”. Esse botão cria uma GitHub pull request. Os mantenedores do guia analisarão essa pull request. Um revisor aprovará a pull request ou solicitará alterações.

Exibir e definir o ID de uma página

Esta página explica como visualizar e definir o ID da página.

O ID da página é uma string exclusiva que identifica cada página no site de documentação. Você pode ver o ID da página na barra de endereço do seu navegador quando estiver em uma página específica. O ID da página é usado para o URL, o nome do arquivo e para criar links de referência cruzada.

Por exemplo, se você estiver visualizando esta página, o URL na barra de endereço do seu navegador será semelhante a:

```
https://docs.aws.amazon.com/view-set-page-id.html
```

A última parte do URL (`view-set-page-id`) é o ID da página.

Definir o ID da página

Ao criar uma nova página, você precisa definir o ID da página no arquivo de origem. O ID da página deve ser uma string concisa e hifenizada que descreva o conteúdo da página.

1. Abra o arquivo de origem da sua nova página em um editor de texto.
2. Na parte superior do arquivo, adicione a seguinte linha. Deve estar acima do primeiro título.

   ```
   [#my-new-page]
   ```

   `my-new-page`Substitua pelo ID da página da sua nova página.
3. Salve o arquivo.

> **Note**
>
> A página IDs deve ser exclusiva em todo o site de documentação. Se você tentar usar um ID de página existente, receberá um erro de compilação.

Criação de uma nova página

Saiba como criar uma nova página e atualizar o sumário do guia.

Criar metadados da página

1. Determine o título da página e o título curto da página. O título curto da página é opcional, mas recomendado se o título da página tiver mais do que algumas palavras.
2. Determine o ID da página. Isso deve ser exclusivo no Guia de Melhores Práticas do EKS. A convenção é usar todas as palavras em minúsculas e separar com. `-`
3. Crie um novo arquivo asciidoc, em uma pasta, se necessário, e adicione o seguinte texto ao arquivo:

Example

[.” tópico "] [#<page-id>] = <page-title>:info_titleabbrev: < > page-short-title

Por exemplo,

Example

[.” topic "] [#scalability] = Melhores práticas de escalabilidade do EKS:info_titleabbrev: Escalabilidade

Adicionar ao índice

1. Abra o arquivo da página principal no sumário. Para novas seções do guia de nível superior, o arquivo principal é`book.adoc`.
2. Na parte inferior do arquivo pai, atualize e insira a seguinte diretiva:

   Example

   incluem: <new-filename>[leveloffset=+1]

   Por exemplo,

   Example

   inclui: :dataplane.adoc [leveloffset=+1]

Inserir uma imagem

1. Encontre o prefixo da imagem para a página que você está editando. Revise a `:imagesdir:` propriedade no cabeçalho do arquivo. Por exemplo, `` `:imagesdir: images/reliability/ ``
2. Coloque sua imagem nesse caminho, como `latest/bpg/images/reliability`
3. Determine o texto alternativo apropriado para sua imagem. Escreva uma breve descrição de alto nível da imagem. Por exemplo, “diagrama da VPC com três zonas de disponibilidade” é o texto alternativo apropriado.
4. Atualize o exemplo a seguir com o texto alternativo e o nome do arquivo de imagem. Insira no local desejado.

Example

imagem: <image-filename>[< image-alt-text >]

Por exemplo,

Example

imagem: eks-data-plane-connectivity .jpeg [Diagrama de rede]

Verifique o estilo com a Vale

1. Instale o Vale CLI.
2. Executar `vale sync`
3. Instale a extensão Vale a partir do Visual Studio Marketplace.
4. Reinicie o VS Code e abra um AsciiDoc arquivo
5. O VS Code sublinha o texto problemático. Aprenda a trabalhar com erros e avisos nos documentos do VS Code.

Crie uma prévia local

1. Instale a `asciidoctor` ferramenta usando `brew` no Linux ou no macOS
   - Saiba como instalar o asciidoctor cli nos documentos. AsciiDoctor
   - Saiba como instalar o gerenciador de pacotes brew.
2. Abra um terminal e navegue até `latest/bpg/`
3. Executar `asciidoctor book.adoc`
   - Revise todos os avisos e erros de sintaxe
4. Abra o arquivo `book.html` de saída.
   - No macOS, você pode executar `open book.html` para abrir a visualização prévia no seu navegador padrão.

AsciiDoc Folha de dicas

Formatação básica

```
*bold text*
_italic text_
`monospace text`
```

Cabeçalhos

```
= Document Title (Header 1)
== Header 2
=== Header 3
==== Header 4
===== Header 5
====== Header 6
```

Listas

Listas não ordenadas:

```
- 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]
```

Imagens

```
image::image-file.jpg[Alt text]
```

Blocos de código

```
 [source,python]
 ----
 def hello_world():
     print("Hello, World!")
 ----
```

Tabelas

Aprenda a desenvolver uma tabela 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
|===
```

Advertências

```
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.
```

Versão prévia:

> **Note**
>
> Esta é uma advertência do tipo nota.

Inclui

```
include::filename.adoc[]
```

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.