Panduan Praktik Terbaik

Amazon EKS

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

Amazon EKS: Panduan Praktik Terbaik

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

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan di antara pelanggan, atau dengan cara apa pun yang merendahkan atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon adalah milik pemiliknya masing-masing, yang mungkin atau mungkin tidak berafiliasi dengan, terhubung ke, atau disponsori oleh Amazon

Table of Contents

Pengantar 1
Panduan terkait 2
Berkontribusi 2
Keamanan 3
Cara menggunakan panduan ini 3
Memahami Model Tanggung Jawab Bersama 3
Pengantar 1
Umpan Balik 6
Sumber Bacaan Lebih Lanjut 6
Alat dan sumber daya 6
Mode Otomatis EKS - Keamanan 6
Arsitektur Keamanan 7
Pertanyaan yang Sering Diajukan (FAQ) 17
Identity and Access Management 20
Mengontrol Akses ke Kluster EKS 20
Rekomendasi Akses Cluster 27
Identitas dan Kredensional untuk Pod EKS 34
Identitas dan Kredensyal untuk Rekomendasi Pod EKS 43
Alat dan Sumber Daya 51
Keamanan Pod 52
Kemampuan Linux 52
Solusi Keamanan Pod 54
Rekomendasi 63
Alat dan sumber daya 51
Multi-penghunian 70
Multi-tenancy lembut 70
Konstruksi Kubernetes 73
Mengurangi kontrol 75
Multi-tenancy yang keras 83
Arah masa depan 83
Alat dan sumber daya manajemen multi-cluster 84
Kontrol Detektif 84
Rekomendasi 63
Alat dan sumber daya 51

Keamanan jaringan 92
Kontrol lalu lintas 92
Enkripsi jaringan 92
Kebijakan jaringan 93
Rekomendasi 63
Grup keamanan 98
Kapan menggunakan Kebijakan Jaringan vs Grup Keamanan untuk Pod? 100
Penegakan Kebijakan Service Mesh atau kebijakan jaringan Kubernetes 102
ThirdParty Mesin Kebijakan Jaringan 103
Enkripsi saat bergerak 104
Alat dan sumber daya 51
Enkripsi data dan manajemen rahasia 115
Enkripsi saat diam 115
Manajemen rahasia 117
Alat dan sumber daya 51
Keamanan runtime 120
Konteks keamanan dan kontrol Kubernetes bawaan 121
Rekomendasi 63
Alat dan Sumber Daya 124
Keamanan Infrastruktur 125
Rekomendasi 63
Alternatif 129
Alat dan sumber daya 51
Kepatuhan Regulasi 132
Bergeser Kiri 134
Alat dan sumber daya 51
Respon insiden dan forensik 136
Contoh rencana respons insiden 136
Rekomendasi 63
Alat dan sumber daya 51
Keamanan gambar 141
Rekomendasi 63
Alat dan sumber daya 51
Strategi Multi Akun 152
Merencanakan Strategi Akun Multi Beban Kerja untuk Multi Tenant Cluster 152
Kluster EKS Terpusat 153

Cluster EKS yang tidak terpusat 159
Cluster EKS terpusat vs de-sentralisasi 161
Manajemen akses cluster 163
Opsi manajemen akses EKS 164
Opsi 1: Pusat Identitas AWS IAM dengan API Manajemen Akses Cluster (CAM) 164
Opsi 2: AWS IAM Users/Roles dipetakan ke grup Kubernetes 167
Opsi 3: Penyedia OIDC 167
AWS EKS Pod Identity vs IRSA untuk beban kerja 168
Rekomendasi 169
Penskalaan Otomatis Cluster 170
Mode Otomatis EKS 170
Alasan untuk menggunakan Mode Otomatis 171
Pertanyaan yang Sering Diajukan 171
Karpenter 177
Rekomendasi 63
Praktik terbaik Karpenter 178
Menciptakan NodePools 182
Penjadwalan Pod 186
Rekomendasi CoreDNS 190
Cetak Biru Karpenter 190
Sumber Daya Tambahan 91
Autoscaler klaster 191
Ikhtisar 191
Mengoptimalkan Kinerja dan Skalabilitas 195
Mengoptimalkan Biaya dan Ketersediaan 198
Kasus Penggunaan Tingkat Lanjut 201
Parameter Tambahan 204
Sumber Daya Tambahan 91
Referensi 82
Keandalan 208
Cara menggunakan panduan ini 3
Pengantar 1
Umpan Balik 6
Aplikasi 211
Rekomendasi 63
Penskaler Otomatis Pod Horisontal (HPA) 216

Autoscaler Pod Vertikal (VPA) 217
Memperbarui aplikasi 218
Pemeriksaan Kesehatan dan Penyembuhan Diri 220
Rekomendasi 220
Berurusan dengan gangguan 222
Rekomendasi 223
Observabilitas 225
Rekomendasi 226
Bidang Kontrol 228
Arsitektur EKS 229
Rekomendasi 230
Monitor Metrik Bidang Kontrol 230
Otentikasi Cluster 233
Penerimaan Webhooks 234
Menangani Upgrade Cluster 235
Konektivitas Titik Akhir Cluster 236
Menjalankan cluster besar 237
Sumber Daya Tambahan: 237
Bidang Data 237
Rekomendasi 63
Jaringan 245
Model Jaringan Kubernetes 245
Antarmuka Jaringan Kontainer (CNI) 246
Amazon Virtual Private Cloud (VPC) CNI 247
Kalkulator Subnet 248
Pertimbangan VPC dan Subnet 249
Ikhtisar 191
Rekomendasi 63
Amazon VPC CNI 260
Ikhtisar 191
Rekomendasi 63
Mengoptimalkan Pemanfaatan Alamat IP 273
Optimalkan konsumsi IP tingkat node 274
Mengurangi kelelahan IP 274
Menjalankan IPv6 Cluster 280
Ikhtisar 191

Rekomendasi 63
Jaringan Kustom 290
Contoh Konfigurasi 291
Rekomendasi 63
Mode Awalan untuk Linux 298
Rekomendasi 63
Mode Awalan untuk Windows 303
Rekomendasi 63
Grup Keamanan Per Pod 309
Rekomendasi 63
Penyeimbangan Beban 318
Memilih Jenis Load Balancer 319
Penyeimbang Beban Penyedia 320
Memilih Jenis Target Load Balancer 321
Mengkonfigurasi Pemeriksaan Kesehatan Load Balancer 323
Ketersediaan dan Siklus Hidup Pod 324
Referensi 82
Lampiran 329
Pemantauan untuk masalah kinerja Jaringan 330
Memantau lalu lintas CoreDNS untuk masalah pelambatan DNS 330
Memantau penundaan kueri DNS menggunakan metrik Conntrack 331
Metrik kinerja Jaringan penting lainnya 332
Menangkap metrik untuk memantau beban kerja untuk masalah kinerja jaringan 332
Menjalankan kube-proxy dalam Mode IPVS 340
Ikhtisar 191
Skalabilitas 345
Cara menggunakan panduan ini 345
Memahami dimensi penskalaan 345
Penskalaan ekstra besar 346
Bidang Kontrol 346
Batasi beban kerja dan ledakan simpul 347
Skala node dan pod dengan aman 347
Gunakan Cloent-Side Cache saat menjalankan Kubectl 348
Nonaktifkan Kompresi kubectl 348
Shard Cluster Autoscaler 349
Prioritas dan Keadilan API 351

Mengambil sumber daya di server API 358
Bidang Data 365
Penskalaan otomatis node 365
Gunakan banyak jenis instans EC2 yang berbeda 366
Lebih suka node yang lebih besar untuk mengurangi beban server API 367
Gunakan ukuran node yang serupa untuk kinerja beban kerja yang konsisten 368
Gunakan sumber daya komputasi secara efisien 369
Otomatiskan pembaruan Amazon Machine Image (AMI) 369
Gunakan beberapa volume EBS untuk wadah 370
Hindari instance dengan batas lampiran EBS rendah jika beban kerja menggunakan volume EBS 371
Nonaktifkan logging yang tidak perlu ke disk 372
Contoh tambalan di tempat ketika kecepatan pembaruan OS adalah suatu keharusan 372
Layanan Cluster 373
Skala CoreDNS 373
Skala Server Metrik Kubernetes Secara Vertikal 375
Durasi lameduck CoreDNS 375
Penyelidikan kesiapan CoreDNS 376
Agen penebangan dan pemantauan 376
Beban kerja 377
Gunakan IPv6 untuk jaringan pod 378
Batasi jumlah layanan per namespace 378
Memahami Kuota Elastic Load Balancer 378
Gunakan Route 53, Global Accelerator, atau CloudFront 379
Gunakan EndpointSlices alih-alih Titik Akhir 380
Gunakan rahasia abadi dan eksternal jika memungkinkan 380
Batasi riwayat Penerapan 381
Nonaktifkan secara enableServiceLinks default 381
Batasi webhook penerimaan dinamis per sumber daya 382
Bandingkan beban kerja di beberapa cluster 382
Teori di balik penskalaan 383
Node vs Tingkat Churn 383
Berpikir dalam Pertanyaan Per Detik 383
Penskalaan Komponen Terdistribusi 384
Kemacetan Hulu dan Hilir 385
Skala menurut Metrik 387

Kontrol Pemantauan Pesawat 390
Server API 390
Dimana masalahnya? 390
Penjadwal 396
Manajer Pengontrol Kube 398
ETCD 400
Efisiensi dan penskalaan node 401
Pemilihan Node 401
Pengepakan Node Bin 403
Pemanfaatan vs Saturasi 410
Mengatur Batas CPU 418
Kubernetes SLOs 422
Kubernetes SLOs 422
Metrik Kubernetes SLI 425
SLOs di Cluster Anda 429
Batas yang Diketahui dan Service Quotas 430
Quotas Layanan AWS Lainnya 430
Pelambatan Permintaan AWS 437
Batas Diketahui Lainnya 438
Upgrade Cluster 439
Ikhtisar 439
Sebelum Upgrade 439
Pertahankan cluster Anda up-to-date 440
Tinjau kalender rilis EKS 441
Memahami bagaimana model tanggung jawab bersama berlaku untuk peningkatan klaster 441
Tingkatkan cluster di tempat 442
Tingkatkan bidang kontrol dan bidang data Anda secara berurutan 442
Gunakan Dokumentasi EKS untuk membuat daftar periksa pemutakhiran 443
Upgrade add-on dan komponen menggunakan Kubernetes API 443
Verifikasi persyaratan EKS dasar sebelum meningkatkan 445
Verifikasi alamat IP yang tersedia 445
Verifikasi peran EKS IAM 446
Migrasi ke Pengaya EKS 447
Mengidentifikasi dan memulihkan penggunaan API yang dihapus sebelum memutakhirkan bidang kontrol 448
Wawasan Cluster 448

K ube-no-trouble 450
Pluto 451
Sumber daya 451
Perbarui beban kerja Kubernetes. Gunakan kubectl-convert untuk memperbarui manifes 452
topologySpreadConstraints Konfigurasikan PodDisruptionBudgets dan untuk memastikan ketersediaan beban kerja Anda saat bidang data ditingkatkan 453
Gunakan Grup Node Terkelola atau Karpenter untuk menyederhanakan peningkatan pesawat data 454
Konfirmasikan kompatibilitas versi dengan node yang ada dan bidang kontrol 455
Aktifkan kedaluwarsa node untuk node terkelola Karpenter 455
Gunakan fitur Drift untuk node terkelola Karpenter 456
Gunakan eksctl untuk mengotomatiskan peningkatan untuk grup node yang dikelola sendiri 456
Backup cluster sebelum upgrade 457
Mulai ulang penerapan Fargate setelah memutakhirkan bidang kontrol 457
Evaluasi Blue/Green Cluster sebagai alternatif untuk peningkatan klaster di tempat 457
Lacak perubahan besar yang direncanakan dalam proyek Kubernetes — Pikirkan ke depan ... 458
Panduan Khusus tentang Penghapusan Fitur 459
Penghapusan Dockershim di 1.25 - Gunakan Detektor untuk Docker Socket (DDS) 459
Penghapusan PodSecurityPolicy in 1.25 - Migrasi ke Standar Keamanan Pod atau solusi policy-as-code 459
Pengakhiran Driver Penyimpanan In-Tree di 1.23 - Migrasi ke Driver Container Storage Interface (CSI) 459
Sumber Daya Tambahan 460
ClowdHaus Panduan Peningkatan EKS 460
GoNoGo 460
Pengoptimalan Biaya 461
Pedoman Umum 461
Praktik Terbaik Optimalisasi Biaya EKS 461
Cara menggunakan panduan ini 3
Layanan AWS Utama dan fitur Kubernetes 462
Umpan Balik 6
Kerangka Kerja 463
Pilar See: Pengukuran dan akuntabilitas 463
Pilar Hemat: Optimalisasi biaya 464
Pilar Rencana: Perencanaan dan peramalan 465
Pilar Run 465

Referensi 82
Kesadaran 466
Rekomendasi 63
Alat-alat lainnya 473
Hitung 474
Ukuran beban kerja Anda dengan benar 475
Kurangi konsumsi 476
Kurangi kapasitas yang tidak terpakai 477
Optimalkan jenis kapasitas komputasi 483
Optimalkan Penggunaan Komputasi 487
Jaringan 488
Komunikasi Pod ke Pod 489
Load Balancer ke Komunikasi Pod 501
Transfer Data dari Registri Kontainer 503
Transfer Data ke Layanan Internet & AWS 504
Transfer Data antara VPCs 507
Menggunakan Service Mesh 510
Sumber Daya Tambahan 91
Penyimpanan 518
Gambaran Umum 191
Volume Ephemeral 518
Volume Persisten 519
Pertimbangan lainnya 526
Observabilitas 527
Pengantar 527
Pencatatan log 527
Pesawat Kontrol EKS 527
Pesawat Data EKS 528
Metrik 533
Pelacakan 536
Sumber Daya Tambahan: 91
Windows 540
Manajemen AMI 540
Mengelola Amazon EKS Anda sendiri yang dioptimalkan Windows AMI 541
Mengkonfigurasi peluncuran yang lebih cepat untuk EKS khusus yang dioptimalkan AMIs 542
Caching lapisan dasar Windows pada kustom AMIs 543

Posting blog 544
GMsa untuk Wadah Windows 544
Apa itu akun GMSA 544
Wadah Windows dan kasus penggunaan GMSA 545
Pengerasan Server Windows 547
Mengurangi permukaan serangan dengan Windows Server Core 547
Menghindari koneksi RDP 548
Amazon Inspector 549
Amazon GuardDuty 550
Keamanan di Amazon EC2 untuk Windows 550
Memindai Gambar Windows 551
Versi Windows dan Lisensi 551
Versi Windows Server 551
Lisensi 552
Pencatatan log 552
Rekomendasi Pencatatan 552
Memantau Wadah Windows 553
Jaringan Windows 558
Ikhtisar Jaringan Kontainer Windows 558
Manajemen Alamat IP 559
Opsi Antarmuka Jaringan Kontainer (CNI) 562
Kebijakan Jaringan 562
Memori dan Manajemen Sistem 563
Sistem reservasi dan memori kubelet 563
Persyaratan memori wadah Windows 564
Kesimpulan 564
Manajemen Infrastruktur 565
Mendorong dan menarik gambar Windows 565
Referensi 567
Penjadwalan 567
Menetapkan PODs ke Node Praktik terbaik 568
Memastikan beban kerja khusus OS mendarat di host kontainer yang sesuai 568
Menangani beberapa build Windows di cluster yang sama 569
Penyederhanaan NodeSelector dan Tolerasi dalam manifes Pod menggunakan RuntimeClass 570
Dukungan Grup Node Terkelola 571

Dokumentasi tambahan 572
Keamanan Pod untuk Wadah Windows 572
Opsi Penyimpanan 575
Apa itu plugin in-tree vs. out-of-tree volume? 575
Plugin Volume In-tree untuk Windows 576
Out-of-tree untuk Windows 580
Amazon FSx untuk Server File Windows 580
Pengerasan gambar wadah Windows 581
1. Konfigurasikan Kebijakan Akun (Kata Sandi atau Lockout) menggunakan Kebijakan Keamanan Lokal dan Registri 582
2. Kebijakan audit 585
3. Praktik terbaik Keamanan IIS untuk wadah Windows 588
4. Prinsip Keistimewaan Terkecil 593
Pikiran Akhir: Mengapa Mengamankan Wadah Windows Anda Harus Dimiliki di Lanskap Ancaman Hari Ini 596
Hibrida 597
Pemutusan Jaringan 597
Praktik terbaik 598
Kegagalan pod Kubernetes 603
Lalu lintas jaringan aplikasi 609
Kredensi tuan rumah 614
AI/ML 617
Umpan Balik 617
Hitung 617
Optimalisasi Sumber Daya GPU dan Manajemen Biaya 617
Ketahanan Node dan Manajemen Job Pelatihan 632
Penskalaan dan Kinerja Aplikasi 635
Alokasi sumber daya dinamis untuk manajemen GPU tingkat lanjut 637
Jaringan 683
Pertimbangkan Bandwidth Jaringan yang Lebih Tinggi atau Adaptor Kain Elastis Untuk Aplikasi dengan Komunikasi Antar Node Tinggi 684
Tingkatkan jumlah alamat IP yang tersedia untuk mengaktifkan waktu peluncuran pod yang lebih cepat 684
Keamanan 685
Keamanan dan Kepatuhan 686
Penyimpanan 687

Manajemen dan Penyimpanan Data 687
Observabilitas 700
Pemantauan dan Observabilitas 700
Observabilitas dan Metrik 701
Performa 717
Mengelola Artefak ML, Melayani Kerangka Kerja, dan Optimasi Startup 717
Mengoptimalkan waktu tarik gambar kontainer 720
Pertimbangkan NVMe untuk penyimpanan kubelet dan containerd 723
Berkontribusi 724
Ringkasan untuk kontributor yang ada 724
Siapkan lingkungan pengeditan lokal 724
Garpu dan kloning repo 724
Buka Ruang Kerja VS Code 724
Mengedit file 725
Kirim Permintaan Tarik 725
Gunakan editor berbasis web github.dev 725
Mengedit satu halaman 726
Melihat dan mengatur ID untuk halaman 726
Mengatur ID halaman 726
Buat halaman baru 727
Buat metadata halaman 727
Tambahkan ke daftar isi 728
Sisipkan gambar 728
Periksa gaya dengan Vale 728
Membangun pratinjau lokal 729
AsciiDoc Lembar Cheat 729
Pemformatan Dasar 729
Header 729
Daftar 730
Tautan 730
Citra 730
Blok Kode 730
Tabel 730
Peringatan 731
Penyertaan 731
..... dccxxxii

Panduan Praktik Terbaik Amazon EKS

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Selamat datang di Panduan Praktik Terbaik EKS. Tujuan utama dari proyek ini adalah untuk menawarkan serangkaian praktik terbaik untuk operasi hari ke-2 untuk Amazon EKS. Kami memilih untuk mempublikasikan panduan ini GitHub agar kami dapat mengulangi dengan cepat, memberikan rekomendasi yang tepat waktu dan efektif untuk berbagai masalah, dan dengan mudah memasukkan saran dari komunitas yang lebih luas.

Saat ini kami telah menerbitkan panduan untuk topik-topik berikut:

- Praktik Terbaik untuk Keamanan
- Praktik Terbaik untuk Keandalan
- Praktik Terbaik untuk Penskalaan Otomatis Cluster: Karpenter
- Praktik Terbaik untuk Penskalaan Otomatis Cluster: cluster-autoscaler
- Praktik Terbaik untuk Penskalaan Otomatis Cluster: Mode Otomatis EKS
- Praktik Terbaik untuk Jaringan
- Praktik Terbaik untuk Skalabilitas
- Praktik Terbaik untuk Upgrade Cluster
- Praktik Terbaik untuk Optimalisasi Biaya
- Praktik Terbaik untuk Menjalankan Windows Container
- Praktik Terbaik untuk Penerapan Hybrid
- Praktik Terbaik untuk Menjalankan Beban AI/ML Kerja

Kami juga membuka sumber CLI berbasis Python (Antarmuka Baris Perintah) yang disebut hardeneks untuk memeriksa beberapa rekomendasi dari panduan ini.

Di masa depan kami akan menerbitkan panduan praktik terbaik untuk kinerja, optimalisasi biaya, dan keunggulan operasional.

Panduan terkait

Selain Panduan Pengguna EKS, AWS telah menerbitkan beberapa panduan lain yang dapat membantu Anda menerapkan EKS.

- Panduan Praktik Terbaik Kontainer EMR
- Data tentang EKS
- Praktik Terbaik AWS Observability
- Cetak Biru Amazon EKS untuk Terraform
- Cetak Biru Amazon EKS Mulai Cepat

Berkontribusi

Kami mendorong Anda untuk berkontribusi pada panduan ini. Jika Anda telah menerapkan praktik yang terbukti efektif, silakan bagikan dengan kami dengan membuka masalah atau permintaan tarik. Demikian pula, jika Anda menemukan kesalahan atau cacat dalam panduan yang telah kami terbitkan, kirimkan PR untuk memperbaikinya. Pedoman pengiriman PRs dapat ditemukan di Pedoman Berkontribusi kami.

Praktik Terbaik untuk Keamanan

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Panduan ini memberikan saran tentang melindungi informasi, sistem, dan aset yang bergantung pada EKS sambil memberikan nilai bisnis melalui penilaian risiko dan strategi mitigasi. Panduan di sini adalah bagian dari serangkaian panduan praktik terbaik yang diterbitkan AWS untuk membantu pelanggan menerapkan EKS sesuai dengan praktik terbaik. Panduan untuk Kinerja, Keunggulan Operasional, Optimalisasi Biaya, dan Keandalan akan tersedia dalam beberapa bulan mendatang.

Cara menggunakan panduan ini

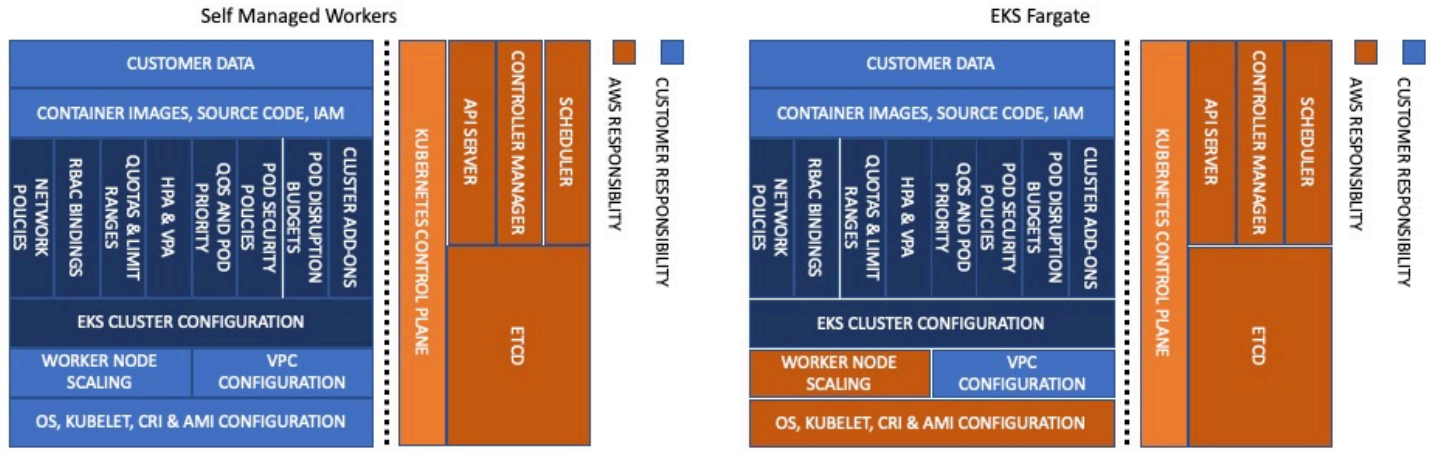
Panduan ini ditujukan untuk praktisi keamanan yang bertanggung jawab untuk menerapkan dan memantau efektivitas kontrol keamanan untuk kluster EKS dan beban kerja yang mereka dukung. Panduan ini diatur ke dalam area topik yang berbeda untuk konsumsi yang lebih mudah. Setiap topik dimulai dengan ikhtisar singkat, diikuti dengan daftar rekomendasi dan praktik terbaik untuk mengamankan kluster EKS Anda. Topik tidak perlu dibaca dalam urutan tertentu.

Memahami Model Tanggung Jawab Bersama

Keamanan dan kepatuhan dianggap sebagai tanggung jawab bersama saat menggunakan layanan terkelola seperti EKS. Secara umum, AWS bertanggung jawab atas keamanan “dari” cloud sedangkan Anda, pelanggan, bertanggung jawab atas keamanan “di” cloud. Dengan EKS, AWS bertanggung jawab untuk mengelola pesawat kontrol Kubernetes yang dikelola EKS. Ini termasuk node bidang kontrol Kubernetes, database ETCD, dan infrastruktur lain yang diperlukan AWS untuk memberikan layanan yang aman dan andal. Sebagai konsumen EKS, Anda sebagian besar bertanggung jawab atas topik dalam panduan ini, misalnya IAM, keamanan pod, keamanan runtime, keamanan jaringan, dan sebagainya.

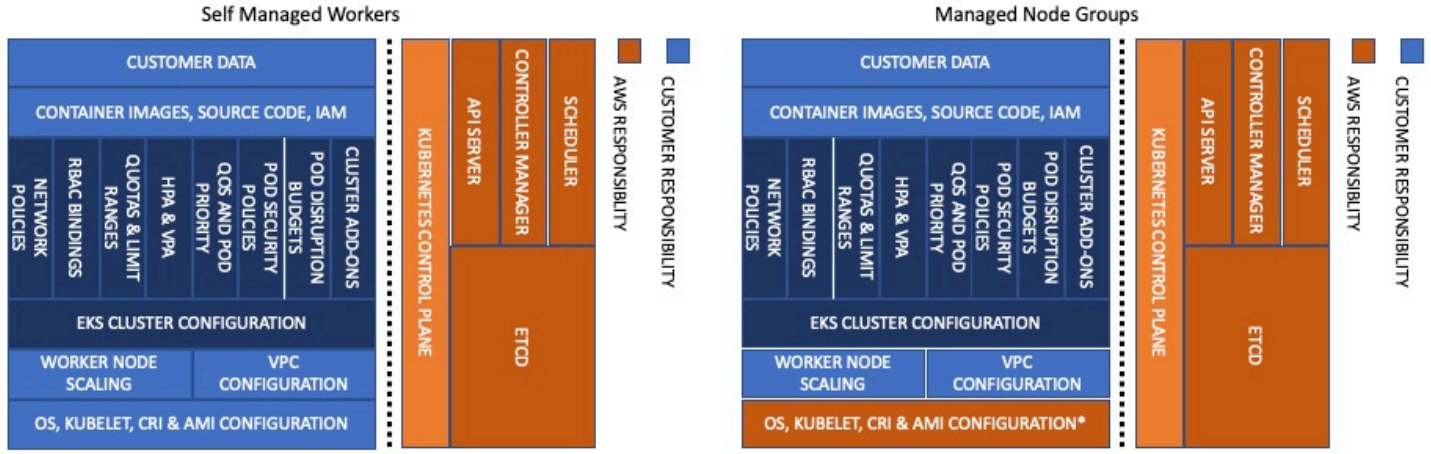
Dalam hal keamanan infrastruktur, AWS akan memikul tanggung jawab tambahan saat Anda beralih dari pekerja yang dikelola sendiri, ke grup node terkelola, ke Fargate. Misalnya, dengan Fargate, AWS bertanggung jawab untuk mengamankan dasar yang instance/runtime digunakan untuk menjalankan Pod Anda.

Model Tanggung Jawab Bersama - Fargate



AWS juga akan bertanggung jawab untuk menjaga agar AMI yang dioptimalkan EKS tetap up to date dengan versi patch Kubernetes dan patch keamanan. Pelanggan yang menggunakan Managed Node Groups (MNG) bertanggung jawab untuk meningkatkan Nodegroups mereka ke AMI terbaru melalui EKS API, CLI, Cloudformation, atau AWS Console. Juga tidak seperti Fargate, tidak akan secara otomatis MNGs menskalakan infrastruktur/cluster Anda. Itu dapat ditangani oleh cluster-autoscaler atau teknologi lain seperti Karpenter, autoscaling AWS asli, Ocean, atau Atlassian's Escalator. SpotInst

Model Tanggung Jawab Bersama - MNG



Sebelum merancang sistem Anda, penting untuk mengetahui di mana garis demarkasi antara tanggung jawab Anda dan penyedia layanan (AWS).

Untuk informasi tambahan tentang model tanggung jawab bersama, lihat https://aws.amazon.com/compliance/shared-responsibility-model/

Pengantar

Ada beberapa area praktik terbaik keamanan yang relevan saat menggunakan layanan Kubernetes yang dikelola seperti EKS:

- Identity and Access Management
- Keamanan Pod
- Keamanan Runtime
- Keamanan Jaringan
- Multi-penghunian
- Multi Akun untuk Multi-tenancy
- Kontrol Detektif
- Keamanan Infrastruktur
- Enkripsi Data dan Manajemen Rahasia
- Kepatuhan Regulasi
- Respon Insiden dan Forensik
- Keamanan Gambar

Sebagai bagian dari merancang sistem apa pun, Anda perlu memikirkan implikasi keamanannya dan praktik yang dapat memengaruhi postur keamanan Anda. Misalnya, Anda perlu mengontrol siapa yang dapat melakukan tindakan terhadap serangkaian sumber daya. Anda juga memerlukan kemampuan untuk mengidentifikasi insiden keamanan dengan cepat, melindungi sistem dan layanan Anda dari akses yang tidak sah, dan menjaga kerahasiaan dan integritas data melalui perlindungan data. Memiliki serangkaian proses yang terdefinisi dengan baik dan terlatih untuk menanggapi insiden keamanan akan meningkatkan postur keamanan Anda juga. Alat dan teknik ini penting karena dapat mendukung berbagai sasaran, seperti mencegah kerugian finansial atau mematuhi peraturan yang berlaku.

AWS membantu organisasi mencapai tujuan keamanan dan kepatuhan mereka dengan menawarkan serangkaian layanan keamanan yang kaya yang telah berkembang berdasarkan umpan balik dari serangkaian luas pelanggan yang sadar keamanan. Dengan menawarkan fondasi yang

sangat aman, pelanggan dapat menghabiskan lebih sedikit waktu untuk “angkat berat yang tidak terdiferensiasi” dan lebih banyak waktu untuk mencapai tujuan bisnis mereka.

Umpan Balik

Panduan ini sedang GitHub dirilis untuk mengumpulkan umpan balik langsung dan saran dari EKS/Kubernetes komunitas yang lebih luas. Jika Anda memiliki praktik terbaik yang menurut Anda harus kami sertakan dalam panduan ini, harap ajukan masalah atau kirimkan PR di GitHub repositori. Tujuan kami adalah memperbarui panduan secara berkala karena fitur baru ditambahkan ke layanan atau ketika praktik terbaik baru berkembang.

Sumber Bacaan Lebih Lanjut

Whitepaper Keamanan Kubernetes, yang disponsori oleh Security Audit Working Group, Whitepaper ini menjelaskan aspek-aspek kunci dari permukaan serangan Kubernetes dan arsitektur keamanan dengan tujuan membantu praktisi keamanan membuat keputusan desain dan implementasi yang baik.

CNCF juga menerbitkan white paper tentang keamanan cloud native. Paper ini meneliti bagaimana lanskap teknologi telah berkembang dan mengadvokasi adopsi praktik keamanan yang selaras dengan DevOps proses dan metodologi tangkas.

Alat dan sumber daya

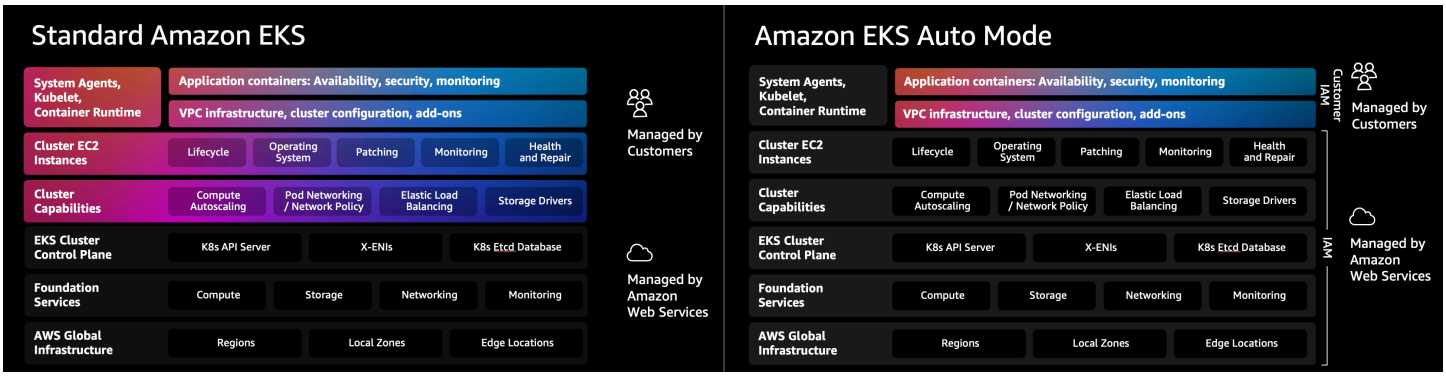
Lokakarya Perendaman Keamanan Amazon EKS

Mode Otomatis EKS - Keamanan

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Amazon EKS Auto Mode memperkenalkan kemampuan keamanan yang ditingkatkan dengan memperluas manajemen keamanan AWS di luar bidang kontrol untuk menyertakan node pekerja dan komponen cluster inti. Model keamanan komprehensif ini membantu organisasi mempertahankan postur keamanan yang kuat sekaligus mengurangi overhead operasional.

Model Tanggung Jawab Bersama - Mode Otomatis EKS



Peningkatan keamanan utama dalam Mode Otomatis EKS meliputi:

- OS minimal yang dioptimalkan kontainer dengan permukaan serangan yang dikurangi
- Praktik terbaik keamanan yang diberlakukan melalui Instans Terkelola EC2
- Manajemen patch keamanan otomatis dengan rotasi Node wajib
- Isolasi simpul bawaan dan batas keamanan
- Integrasi IAM yang efisien dengan izin minimal yang diperlukan

Mode Otomatis EKS memberlakukan kontrol keamanan ini secara default, membantu organisasi memenuhi persyaratan keamanan dan kepatuhan mereka sambil menyederhanakan operasi klaster. Pendekatan ini selaras dengan defense-in-depth prinsip, menyediakan beberapa lapisan kontrol keamanan di tingkat infrastruktur, node, dan beban kerja.

> **Important**
>
> Sementara Mode Otomatis EKS menyediakan kemampuan keamanan yang ditingkatkan, organisasi harus tetap menerapkan kontrol keamanan yang sesuai di lapisan aplikasi dan mengikuti praktik terbaik keamanan untuk beban kerja mereka yang berjalan di cluster.

Arsitektur Keamanan

Mode Otomatis EKS mengimplementasikan kontrol keamanan di beberapa lapisan infrastruktur EKS, dari bidang kontrol hingga node individual. Memahami arsitektur ini sangat penting untuk mengelola dan mengamankan cluster EKS Anda secara efektif.

Kontrol Keamanan Pesawat

Pesawat kontrol EKS dalam Mode Otomatis EKS mempertahankan standar keamanan tinggi yang sama dengan cluster EKS tradisional sambil menambahkan kemampuan keamanan baru:

- Enkripsi Amplop: Semua data API Kubernetes dienkripsi secara otomatis menggunakan enkripsi amplop.
- Integrasi KMS: Menggunakan AWS KMS dengan penyedia Kubernetes KMS v2, dengan opsi untuk kunci milik AWS atau kunci yang dikelola pelanggan (CMK).
- Manajemen Komponen yang Ditingkatkan: Komponen penting seperti auto-scaling, manajemen ENI, dan pengontrol EBS dipindahkan ke luar cluster dan dikelola oleh AWS.
- Peningkatan Kontrol Keamanan dan Kemampuan Audit: Izin yang diperlukan Mode Otomatis EKS, di luar kluster EKS standar, dikelola sepenuhnya melalui peran IAM cluster daripada peran node individual.

Integrasi IAM dan Manajemen Akses

EKS Auto Mode menyediakan integrasi yang disempurnakan dengan AWS Identity and Access Management (IAM) melalui EKS Access Entries dan EKS Pod Identity.

Manajemen Akses Cluster

EKS Auto Mode memperkenalkan peningkatan pada manajemen akses cluster melalui Cluster Access Management (CAM) API:

- Mode otentikasi standar melalui `EKS_API`
- Peningkatan keamanan melalui manajemen akses berbasis API
- Kontrol akses yang disederhanakan menggunakan Entri Akses dan Kebijakan Akses

Entri Akses dapat dibuat untuk mengelola akses klaster:

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

> **Important**
>
> Meskipun masih memungkinkan untuk membuat kluster Mode Otomatis EKS dengan mode `CONFIG_MAP_AND_API` otentikasi, ini bukan pendekatan standar, dan sangat disarankan untuk menggunakan mode `API` otentikasi default untuk cluster baru. `API`otentikasi berbasis memberikan keamanan yang ditingkatkan dan manajemen akses yang disederhanakan dibandingkan dengan pendekatan ConfigMap berbasis warisan.

Identitas Pod EKS

EKS Auto Mode hadir dengan Pod Identity Agent yang sudah di-deploy, memungkinkan cara yang efisien untuk memberikan izin AWS IAM ke Pod:

- Manajemen izin IAM yang disederhanakan tanpa konfigurasi penyedia OIDC
- Mengurangi overhead operasional dibandingkan dengan IRSA
- Keamanan yang ditingkatkan melalui penandaan sesi dan dukungan ABAC

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

> **Important**
>
> Pod Identity adalah pendekatan yang direkomendasikan untuk izin IAM dalam Mode Otomatis EKS karena menyediakan fitur keamanan yang disempurnakan dan manajemen yang disederhanakan dibandingkan dengan IRSA.

Peran IAM Node

Mode Otomatis EKS menggunakan yang baru `AmazonEKSWorkerNodeMinimalPolicy` yang hanya menyediakan izin yang diperlukan untuk node Mode Otomatis EKS untuk beroperasi. Izin tersebut:

- Berikan serangkaian izin yang dikurangi dibandingkan dengan kebijakan node tradisional

- Patuhi prinsip hak istimewa paling sedikit
- Dikelola dan diperbarui secara otomatis oleh AWS

Pendekatan kebijakan minimal ini membantu meningkatkan postur keamanan dengan membatasi izin yang tersedia untuk node dan beban kerjanya.

Keamanan Node

Mode Otomatis EKS memperkenalkan beberapa peningkatan keamanan yang signifikan di tingkat node:

Keamanan Instans Terkelola EC2

Node Mode Otomatis EKS menggunakan Instans Terkelola Amazon EC2 dengan properti keamanan yang disempurnakan:

- Pembatasan yang diberlakukan oleh IAM yang mencegah operasi yang dapat membahayakan kemampuan AWS untuk mengoperasikan node
- Pola infrastruktur yang tidak dapat diubah di mana perubahan konfigurasi memerlukan penggantian node
- Penggantian node wajib dalam waktu 21 hari untuk memastikan pembaruan keamanan reguler
- Akses terbatas ke metadata instance menggunakan IMDSv2 dengan batas hop terkontrol

Keamanan Sistem Operasi

Sistem operasi adalah varian khusus dari Bottlerocket, dioptimalkan untuk Mode Otomatis EKS, yang meliputi:

- Sistem file root hanya-baca
- SELinux diaktifkan secara default dengan kontrol akses wajib
- Isolasi Pod otomatis menggunakan label SELinux MCS unik
- Akses SSH yang dinonaktifkan dan penghapusan layanan yang tidak perlu
- Patch keamanan otomatis melalui rotasi node

Keamanan Komponen Node

Komponen node dikonfigurasi dengan praktik terbaik keamanan:

- Kubelet dikonfigurasi dengan default yang aman
- Konfigurasi pengerasan runtime kontainer
- Manajemen dan rotasi sertifikat otomatis
- node-to-control-planeKomunikasi terbatas

Keamanan Jaringan

Mode Otomatis EKS mengimplementasikan beberapa fitur keamanan jaringan untuk memastikan komunikasi yang aman di dalam cluster dan dengan sumber daya eksternal:

Kebijakan Jaringan VPC CNI

Mode Otomatis EKS memanfaatkan dukungan Kebijakan Jaringan Kubernetes asli dari Plugin CNI Amazon VPC VPC:

- Terintegrasi dengan API Kebijakan Jaringan Kubernetes hulu
- Memungkinkan kontrol halus atas komunikasi pod-to-pod
- Mendukung aturan masuk dan keluar

Untuk mengaktifkan dukungan kebijakan jaringan dalam Mode Otomatis EKS, Anda perlu mengonfigurasi add-on VPC CNI dengan manifes. `configMap` Inilah contohnya:

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

Ini juga diperlukan untuk menentukan dukungan Kebijakan Jaringan dikonfigurasi di Kelas Node, seperti yang diilustrasikan di sini:

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

```
  networkPolicyEventLogs: Enabled
```

Setelah diaktifkan, Anda dapat membuat kebijakan jaringan untuk mengontrol lalu lintas:

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

Manajemen ENI yang Ditingkatkan

Mode Otomatis EKS memberikan peningkatan keamanan untuk manajemen Elastic Network Interface (ENI):

- Lampiran dan konfigurasi ENI yang dikelola AWS
- Pemisahan lalu lintas kontrol dari lalu lintas data
- Manajemen alamat IP otomatis dengan pengurangan hak istimewa yang diperlukan pada node

Keamanan Penyimpanan

Mode Otomatis EKS menyediakan fitur keamanan yang disempurnakan untuk penyimpanan sementara dan persisten:

Penyimpanan Ephemeral

- Semua data yang ditulis ke volume fana secara otomatis dienkripsi
- Menggunakan algoritma kriptografi AES-256 standar industri
- Enkripsi dan dekripsi ditangani dengan mulus oleh layanan

Volume EBS

- Volume root dan data EBS selalu dienkripsi
- Volume dikonfigurasi untuk dihapus setelah penghentian instance
- Ada opsi untuk menentukan kunci KMS khusus untuk enkripsi

Integrasi EFS

- Support untuk enkripsi dalam perjalanan dengan EFS
- Enkripsi otomatis saat istirahat untuk sistem file EFS
- Integrasi dengan titik akses EFS untuk kontrol akses yang ditingkatkan

> **Important**
>
> Saat menggunakan EFS dengan Mode Otomatis EKS, pastikan bahwa pengaturan enkripsi yang sesuai dikonfigurasi pada tingkat sistem file EFS, karena Mode Otomatis EKS tidak mengelola enkripsi EFS secara langsung.

Pemantauan dan Pencatatan

Mode Otomatis EKS menyediakan kemampuan pemantauan dan pencatatan yang disempurnakan untuk membantu Anda mempertahankan visibilitas ke postur keamanan klaster dan kesehatan operasional Anda.

Kontrol Pencatatan Pesawat

Mode Otomatis EKS mempertahankan kemampuan logging bidang kontrol yang sama dengan EKS standar, namun memungkinkan semua log secara default untuk pemantauan yang ditingkatkan.

- Log dikirim ke Amazon CloudWatch Logs
- Secara default, Mode Otomatis EKS memungkinkan semua log bidang kontrol: server API, audit, autentikator, manajer pengontrol, dan penjadwal
- Mode Otomatis EKS memungkinkan visibilitas terperinci ke dalam operasi klaster dan peristiwa keamanan

> **Important**
>
> Kontrol logging pesawat menimbulkan biaya tambahan untuk penyimpanan log di. CloudWatch Pertimbangkan strategi pencatatan Anda dengan hati-hati untuk menyeimbangkan kebutuhan keamanan dengan manajemen biaya.

Pencatatan Tingkat Simpul

Mode Otomatis EKS meningkatkan pencatatan tingkat simpul:

- Log sistem dikumpulkan secara otomatis dan dapat diakses melalui CloudWatch Log
- Log node dipertahankan bahkan setelah penghentian node, membantu dalam analisis pasca-insiden
- Peningkatan visibilitas ke dalam peristiwa keamanan tingkat simpul dan masalah operasional

GuardDuty Integrasi Amazon

Kluster Mode Otomatis EKS terintegrasi secara mulus dengan Amazon GuardDuty untuk deteksi ancaman yang ditingkatkan. Fitur meliputi:

- Pemindaian otomatis untuk log audit bidang kontrol
- Pemantauan runtime yang dapat diaktifkan untuk pemantauan beban kerja
- Integrasi dengan GuardDuty temuan yang ada dan mekanisme peringatan

Untuk mengaktifkan perlindungan Mode Otomatis EKS di Amazon GuardDuty untuk Kubernetes Audit Logs, Anda dapat menjalankan perintah berikut:

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

GuardDuty Integrasi Amazon untuk Keamanan Runtime

Amazon GuardDuty menyediakan pemantauan keamanan runtime penting untuk kluster Mode Otomatis EKS, menawarkan kemampuan deteksi ancaman dan pemantauan keamanan yang komprehensif. Integrasi ini sangat penting karena membantu mengidentifikasi potensi ancaman keamanan dan aktivitas berbahaya secara real-time.

GuardDuty Fitur Utama untuk Mode Otomatis EKS

- Pemantauan Runtime:
  - Pemantauan perilaku runtime secara terus menerus
  - Deteksi aktivitas berbahaya atau mencurigakan

  - Identifikasi upaya pelarian kontainer potensial
  - Pemantauan eksekusi proses yang tidak biasa atau koneksi jaringan
- Deteksi Ancaman Khusus Kubernetes:
  - Identifikasi upaya penyebaran pod yang mencurigakan
  - Deteksi wadah yang dikompromikan
  - Pemantauan peluncuran kontainer istimewa
  - Identifikasi penggunaan akun layanan yang mencurigakan
- Jenis Temuan Komprehensif:
  - Policy:Kubernetes/* - Mendeteksi pelanggaran praktik terbaik keamanan
  - Dampak: Kubernetes/* - Mengidentifikasi sumber daya yang berpotensi terkena dampak
  - Discovery:Kubernetes/* - Mendeteksi aktivitas pengintaian
  - Eksekusi:Kubernetes/* - Mengidentifikasi pola eksekusi yang mencurigakan
  - Persistence:Kubernetes/* - Mendeteksi potensi ancaman persisten

Untuk mengaktifkan perlindungan Mode Otomatis EKS di Amazon GuardDuty untuk Kubernetes Audit Logs dan Runtime Monitoring, Anda dapat menjalankan perintah berikut:

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

> **Important**
>
> GuardDuty Runtime Monitoring secara otomatis didukung di kluster Mode Otomatis EKS, memberikan visibilitas keamanan yang ditingkatkan tanpa konfigurasi tambahan di tingkat node.

GuardDuty Integrasi Temuan

GuardDuty temuan dapat diintegrasikan dengan layanan AWS lainnya untuk respons otomatis:

- EventBridge Aturan:

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

- Integrasi Security Hub:

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

Praktik Terbaik untuk GuardDuty Mode Otomatis EKS

1. Aktifkan Semua Jenis Penemuan:
   - Aktifkan pemantauan log audit Kubernetes dan pemantauan runtime
   - Konfigurasikan temuan untuk semua tingkat keparahan
2. Menerapkan Respon Otomatis:
   - Buat EventBridge aturan untuk temuan tingkat keparahan tinggi
   - Integrasikan dengan AWS Security Hub untuk manajemen keamanan terpusat
   - Mengatur tindakan remediasi otomatis jika sesuai
3. Ulasan dan Penyetelan Reguler:
   - Tinjau GuardDuty temuan secara teratur
   - Menyetel ambang deteksi berdasarkan lingkungan Anda
   - Perbarui prosedur respons berdasarkan jenis temuan baru
4. Manajemen Lintas Akun:
   - Pertimbangkan untuk menggunakan akun GuardDuty administrator untuk manajemen terpusat
   - Aktifkan agregasi temuan di beberapa akun

> **Warning**
>
> Meskipun GuardDuty menyediakan pemantauan keamanan yang komprehensif, itu harus menjadi bagian dari defense-in-depth strategi yang mencakup kontrol keamanan lainnya seperti Kebijakan Jaringan, Standar Keamanan Pod, dan konfigurasi RBAC yang tepat.

Pertanyaan yang Sering Diajukan (FAQ)

T: Apa perbedaan Mode Otomatis EKS dari EKS standar dalam hal keamanan? J: Mode Otomatis EKS memberikan keamanan yang ditingkatkan melalui Instans Terkelola EC2, penambalan otomatis, rotasi node wajib, dan kontrol keamanan bawaan. Ini mengurangi overhead operasional sambil mempertahankan postur keamanan yang kuat dengan meminta AWS mengelola lebih banyak aspek keamanan.

T: Apakah saya masih dapat menggunakan alat dan kebijakan keamanan yang ada dengan Mode Otomatis EKS? J: Ya, Mode Otomatis EKS kompatibel dengan sebagian besar alat dan kebijakan keamanan yang ada. Namun, beberapa alat keamanan tingkat simpul mungkin memerlukan adaptasi karena sifat node Mode Otomatis EKS yang dikelola.

T: Bagaimana cara menggunakan agen keamanan dan alat pemantauan dalam Mode Otomatis EKS? A: Dalam Mode Otomatis EKS, agen keamanan dan alat pemantauan harus digunakan sebagai beban kerja Kubernetes (biasanya DaemonSets, yang menyebarkan satu instance Pod pada setiap node secara default) daripada diinstal langsung pada OS node. Pendekatan ini sejalan dengan model infrastruktur yang tidak dapat diubah dari Mode Otomatis EKS. Contoh:

```
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"]
```

T: Apakah solusi keamanan pihak ketiga kompatibel dengan Mode Otomatis EKS? J: Banyak solusi keamanan pihak ketiga yang populer telah diperbarui untuk mendukung Mode Otomatis EKS, namun selalu disarankan untuk memverifikasi versi spesifik dan persyaratan penerapan dengan vendor keamanan Anda, karena dukungan untuk Mode Otomatis EKS mungkin memerlukan versi terbaru atau konfigurasi penerapan tertentu.

T: Apa batasan untuk agen keamanan dalam Mode Otomatis EKS? A: Keterbatasan utama meliputi:

- Tidak ada akses langsung untuk memodifikasi sistem operasi node
- Tidak ada persistensi di seluruh rotasi simpul
- Harus kompatibel dengan penerapan berbasis kontainer
- Perlu menghormati kekekalan simpul
- Mungkin memerlukan konfigurasi hak istimewa yang berbeda
- Setiap perubahan persisten pada Node, harus dilakukan melalui `NodePools` dan `NodeClasses` sumber daya.

> **Note**
>
> Meskipun Mode Otomatis EKS mungkin memerlukan penyesuaian pada strategi penerapan alat keamanan Anda, perubahan ini sering kali menghasilkan konfigurasi yang lebih dapat dipelihara dan aman yang selaras dengan praktik terbaik cloud-native. Mode Otomatis EKS mengharapkan untuk sepenuhnya mengambil alih sebagian besar fitur yang dikelolanya. Oleh karena itu, setiap perubahan manual yang Anda buat pada fitur-fitur tersebut, jika Anda bisa mendapatkannya, dapat ditimpa atau dibuang oleh Mode Otomatis EKS.

T: Dapatkah saya menggunakan kustom AMIs dengan Mode Otomatis EKS? J: Saat ini, Mode Otomatis EKS tidak mendukung kustom AMIs. Ini dirancang karena AWS mengelola keamanan,

penambalan, dan pemeliharaan node sebagai bagian dari model tanggung jawab bersama. Node Mode Otomatis EKS menggunakan varian khusus Bottlerocket yang dioptimalkan dan dikelola oleh AWS.

T: Seberapa sering node diputar secara otomatis dalam Mode Otomatis EKS? A: Node dalam Mode Otomatis EKS memiliki masa pakai maksimum 21 hari. Mereka akan diganti secara otomatis sebelum batas ini, memastikan pembaruan keamanan reguler dan aplikasi patch.

T: Dapatkah saya SSH ke node Mode Otomatis EKS untuk pemecahan masalah? J: Tidak, akses SSH langsung tidak tersedia dalam Mode Otomatis EKS. Sebagai gantinya, Anda dapat menggunakan Definisi Sumber Daya NodeDiagnostic Kustom (CRD) untuk mengumpulkan log sistem dan informasi debugging.

T: Apakah dukungan Kebijakan Jaringan diaktifkan secara default dalam Mode Otomatis EKS? J: Untuk saat ini, dukungan Kebijakan Jaringan perlu diaktifkan secara eksplisit melalui konfigurasi add-on VPC CNI. Setelah diaktifkan, Anda dapat menggunakan Kebijakan Jaringan Kubernetes standar.

T: Haruskah saya menggunakan IRSA atau Pod Identity dengan Mode Otomatis EKS? J: Meskipun keduanya didukung, Pod Identity adalah pendekatan yang direkomendasikan dalam Mode Otomatis EKS karena sudah menyertakan add-on agen Pod Identity Security dan menyediakan fitur keamanan yang disempurnakan dan manajemen yang disederhanakan.

T: Apakah saya masih dapat menggunakan aws-auth ConfigMap dalam Mode Otomatis EKS? A: `aws-auth` ConfigMap Ini adalah fitur usang. Disarankan untuk menggunakan pendekatan default otentikasi berbasis API untuk keamanan yang ditingkatkan dan manajemen akses yang disederhanakan.

T: Bagaimana cara memantau peristiwa keamanan dalam Mode Otomatis EKS? A: Mode Otomatis EKS terintegrasi dengan beberapa solusi pemantauan termasuk GuardDuty, CloudWatch, dan CloudTrail. GuardDuty menyediakan pemantauan keamanan runtime yang ditingkatkan khusus untuk beban kerja EKS.

T: Bagaimana cara mengumpulkan log dari node Mode Otomatis EKS? J: Gunakan NodeDiagnostic CRD, yang secara otomatis mengunggah log ke bucket S3. Anda juga dapat menggunakan CloudWatch Container Insights dan AWS Distro untuk. OpenTelemetry

> **Note**
>
> Bagian FAQ ini diperbarui secara berkala karena fitur baru ditambahkan ke Mode Otomatis EKS dan saat kami menerima pertanyaan umum dari pelanggan.

Identity and Access Management

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Identity and Access Management (IAM) adalah layanan AWS yang menjalankan dua fungsi penting: Otentikasi dan Otorisasi. Otentikasi melibatkan verifikasi identitas sedangkan otorisasi mengatur tindakan yang dapat dilakukan oleh sumber daya AWS. Dalam AWS, sumber daya dapat berupa layanan AWS lain, misalnya EC2, atau prinsipal AWS seperti Pengguna atau Peran IAM. Aturan yang mengatur tindakan yang diizinkan untuk dilakukan sumber daya dinyatakan sebagai kebijakan IAM.

Mengontrol Akses ke Kluster EKS

Proyek Kubernetes mendukung berbagai strategi berbeda untuk mengautentikasi permintaan ke layanan kube-apiserver, misalnya Token Pembawa, sertifikat X.509, OIDC, dll. EKS saat ini memiliki dukungan asli untuk otentikasi token webhook, token akun layanan, dan per 21 Februari 2021, otentikasi OIDC.

Strategi otentikasi webhook memanggil webhook yang memverifikasi token pembawa. Di EKS, token pembawa ini dihasilkan oleh AWS CLI atau aws-iam-authenticatorklien saat Anda `kubectl` menjalankan perintah. Saat Anda menjalankan perintah, token diteruskan ke kube-apiserver yang meneruskannya ke webhook otentikasi. Jika permintaan terbentuk dengan baik, webhook memanggil URL yang telah ditandatangani sebelumnya yang disematkan di badan token. URL ini memvalidasi tanda tangan permintaan dan mengembalikan informasi tentang pengguna, misalnya akun pengguna, Arn, dan UserId ke kube-apiserver.

Untuk menghasilkan token otentikasi secara manual, ketik perintah berikut di jendela terminal:

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

Outputnya harus menyerupai ini:

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

```
    "status": {
        "expirationTimestamp": "2024-12-20T17:38:48Z",
        "token": "k8s-aws-
v1.aHR0cHM6Ly9zdHMudXMtd2VzdC0yLmFtYXpvbmF3cy5jb20vP0FjdGlvbj1HZ....."
    }
}
```

Anda juga bisa mendapatkan token secara terprogram. Di bawah ini adalah contoh yang ditulis dalam 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)
}
```

Outputnya harus menyerupai ini:

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

Setiap token dimulai dengan `k8s-aws-v1.` diikuti oleh string yang dikodekan base64. String, ketika diterjemahkan, harus menyerupai sesuatu yang mirip dengan ini:

```
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
```

Token terdiri dari URL yang telah ditandatangani sebelumnya yang mencakup kredensi dan tanda tangan Amazon. Untuk detail tambahan lihat https://docs.aws.amazon.com/STS/ latest/ APIReference/API _ GetCallerIdentity .html.

Token memiliki waktu untuk hidup (TTL) 15 menit setelah itu token baru perlu dibuat. Ini ditangani secara otomatis ketika Anda menggunakan klien seperti`kubectl`, namun, jika Anda menggunakan dasbor Kubernetes, Anda perlu membuat token baru dan mengautentikasi ulang setiap kali token kedaluwarsa.

Setelah identitas pengguna telah diautentikasi oleh layanan AWS IAM, kube-apiserver membaca di `kube-system` Namespace untuk menentukan grup RBAC untuk diasosiasikan dengan pengguna. `aws-auth` ConfigMap `aws-auth` ConfigMap Ini digunakan untuk membuat pemetaan statis antara prinsipal IAM, yaitu Pengguna dan Peran IAM, dan grup RBAC Kubernetes. Grup RBAC dapat direferensikan di Kubernetes atau. RoleBindings ClusterRoleBindings Mereka mirip dengan IAM Roles karena mereka mendefinisikan serangkaian tindakan (kata kerja) yang dapat dilakukan terhadap kumpulan sumber daya Kubernetes (objek).

CloudWatch kueri untuk membantu pengguna mengidentifikasi klien yang mengirim permintaan ke titik akhir STS global

Jalankan CloudWatch kueri di bawah ini untuk mendapatkan titik akhir sts. Jika stsendpoint sama dengan “sts.amazonaws.com”, maka itu adalah titik akhir STS global. Jika stsendpoint sama seperti “sts. <region>.amazonaws.com”, maka itu adalah titik akhir STS regional.

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

Manajer Akses Cluster

Cluster Access Manager, sekarang cara yang disukai untuk mengelola akses prinsipal AWS IAM ke kluster Amazon EKS, adalah fungsionalitas AWS API dan merupakan fitur keikutsertaan

untuk EKS v1.23 dan kluster yang lebih baru (baru atau yang sudah ada). Ini menyederhanakan pemetaan identitas antara AWS IAM dan KubernetesRBACs, menghilangkan kebutuhan untuk beralih antara AWS dan Kubernetes APIs atau mengedit manajemen akses `aws-auth` ConfigMap untuk, mengurangi overhead operasional, dan membantu mengatasi kesalahan konfigurasi. Alat ini juga memungkinkan administrator klaster untuk mencabut atau menyempurnakan `cluster-admin` izin yang secara otomatis diberikan kepada prinsipal AWS IAM yang digunakan untuk membuat cluster.

API ini bergantung pada dua konsep:

- Entri Akses: Identitas klaster yang ditautkan langsung ke prinsipal AWS IAM (pengguna atau peran) yang diizinkan untuk mengautentikasi ke klaster Amazon EKS.
- Kebijakan Akses: Apakah kebijakan khusus Amazon EKS yang menyediakan otorisasi untuk Entri Akses untuk melakukan tindakan di klaster Amazon EKS.

Saat peluncuran Amazon EKS hanya mendukung kebijakan yang telah ditentukan sebelumnya dan AWS yang dikelola. Kebijakan akses bukan entitas IAM dan ditentukan serta dikelola oleh Amazon EKS.

Cluster Access Manager memungkinkan kombinasi RBAC hulu dengan Kebijakan Akses yang mendukung allow dan pass (tetapi tidak ditolak) pada keputusan Kubernetes AuthZ mengenai permintaan server API. Keputusan penolakan akan terjadi ketika keduanya, otorisasi RBAC hulu dan Amazon EKS tidak dapat menentukan hasil evaluasi permintaan.

Dengan fitur ini, Amazon EKS mendukung tiga mode otentikasi:

1. `CONFIG_MAP`untuk terus menggunakan `aws-auth` ConfigMap secara eksklusif.
2. `API_AND_CONFIG_MAP`untuk sumber prinsip IAM yang diautentikasi dari Entri Akses EKS dan ConfigMap`aws-auth`, APIs memprioritaskan Entri Akses. Ideal untuk memigrasikan `aws-auth` izin yang ada ke Entri Akses.
3. `API`untuk secara eksklusif mengandalkan Entri Akses EKS APIs. Ini adalah pendekatan baru yang direkomendasikan.

Untuk memulai, administrator klaster dapat membuat atau memperbarui kluster Amazon EKS, menyetel autentikasi pilihan ke `API_AND_CONFIG_MAP` atau `API` metode, dan menentukan Entri Akses untuk memberikan akses kepada prinsipal AWS IAM yang diinginkan.

```
$ 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
```

Perintah di atas adalah contoh untuk membuat kluster Amazon EKS tanpa izin admin dari pembuat cluster.

Dimungkinkan untuk memperbarui konfigurasi kluster Amazon EKS untuk mengaktifkan `API` AuthenticationMode menggunakan `update-cluster-config` perintah, untuk melakukannya pada cluster yang ada menggunakan `CONFIG_MAP` Anda harus terlebih dahulu memperbarui ke dan kemudian ke`API_AND_CONFIG_MAP`. `API` Operasi ini tidak dapat dikembalikan, artinya tidak mungkin untuk beralih dari `API` ke `API_AND_CONFIG_MAP` atau`CONFIG_MAP`, dan juga dari `API_AND_CONFIG_MAP` ke`CONFIG_MAP`.

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

Perintah dukungan API untuk menambah dan mencabut akses ke klaster, serta memvalidasi Kebijakan Akses dan Entri Akses yang ada untuk klaster yang ditentukan. Kebijakan default dibuat untuk mencocokkan Kubernetes RBACs sebagai berikut.

Kebijakan Akses EKS	Kubernetes RBAC
Amazon EKSCluster AdminPolicy	klaster admin
EKSAdminKebijakan Amazon	admin
EKSEditKebijakan Amazon	edit
EKSViewKebijakan Amazon	lihat

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

Entri Akses tidak tersedia saat klaster dibuat tanpa izin admin pembuat klaster, yang merupakan satu-satunya entri yang dibuat secara default.

aws-auth ConfigMap (usang)

Salah satu cara integrasi Kubernetes dengan autentikasi AWS dapat dilakukan adalah melalui `aws-auth` ConfigMap, yang berada di Namespace. `kube-system` Ini bertanggung jawab untuk memetakan autentikasi AWS IAM Identities (Users, Groups, and Roles), ke otorisasi kontrol akses berbasis peran (RBAC) Kubernetes. `aws-auth` ConfigMap Ini secara otomatis dibuat di kluster Amazon EKS Anda selama fase penyediaannya. Awalnya dibuat untuk memungkinkan node bergabung dengan cluster Anda, tetapi seperti yang disebutkan Anda juga dapat menggunakan ini ConfigMap untuk menambahkan RBACs akses ke prinsipal IAM.

Untuk memeriksa cluster Anda `aws-auth` ConfigMap, Anda dapat menggunakan perintah berikut.

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

Ini adalah contoh konfigurasi default dari file `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
```

Sesi utama ini ConfigMap, berada `data` di bawah `mapRoles` blok, yang pada dasarnya disusun oleh 3 parameter.

- grup: Kubernetes group/groups untuk memetakan Peran IAM. Ini bisa berupa grup default, atau grup kustom yang ditentukan dalam `clusterrolebinding` atau`rolebinding`. Dalam contoh di atas kita hanya memiliki grup sistem yang dideklarasikan.
- rolearn: ARN dari AWS IAM Role dipetakan ke grup/grup Kubernetes add, menggunakan format berikut. `arn:<PARTITION>:iam::<AWS_ACCOUNT_ID>:role/role-name`
- username: Nama pengguna dalam Kubernetes untuk dipetakan ke peran AWS IAM. Ini bisa berupa nama khusus apa pun.

Dimungkinkan juga untuk memetakan izin untuk Pengguna AWS IAM, menentukan blok konfigurasi baru untuk`mapUsers`, `data` di bawah `aws-auth`ConfigMap, mengganti parameter rolearn untuk userarn, namun sebagai Praktik Terbaik selalu disarankan untuk pengguna. `mapRoles`

Untuk mengelola izin, Anda dapat mengedit `aws-auth` ConfigMap penambahan atau penghapusan akses ke klaster Amazon EKS Anda. Meskipun dimungkinkan untuk mengedit `aws-auth` ConfigMap secara manual, disarankan menggunakan alat seperti, karena ini adalah konfigurasi yang sangat sensitif`eksctl`, dan konfigurasi yang tidak akurat dapat mengunci Anda di luar Cluster Amazon EKS

Anda. Periksa subbagian Gunakan alat untuk membuat perubahan pada aws-auth di ConfigMap bawah ini untuk detail selengkapnya.

Manfaat dibandingkan manajemen akses ConfigMap berbasis

1. Mengurangi risiko kesalahan konfigurasi: Manajemen berbasis API langsung menghilangkan kesalahan umum yang terkait dengan pengeditan manual. ConfigMap Ini membantu mencegah penghapusan yang tidak disengaja atau kesalahan sintaks yang dapat mengunci pengguna keluar dari cluster.
2. Prinsip hak istimewa terkecil yang disempurnakan: Menghapus kebutuhan akan izin cluster-admin dari identitas pembuat klaster dan memungkinkan penetapan izin yang lebih terperinci dan sesuai. Anda dapat memilih untuk menambahkan izin ini untuk kasus penggunaan break-glass.
3. Model keamanan yang disempurnakan: Menyediakan validasi bawaan entri akses sebelum diterapkan. Selain itu, menawarkan integrasi yang lebih ketat dengan AWS IAM untuk otentikasi.
4. Operasi yang disederhanakan: Menawarkan cara yang lebih intuitif untuk mengelola izin melalui perkakas asli AWS.

Rekomendasi Akses Cluster

Kombinasikan IAM Identity Center dengan CAM API

- Manajemen yang disederhanakan: Dengan menggunakan API Manajemen Akses Cluster bersama dengan IAM Identity Center, administrator dapat mengelola akses kluster EKS bersama layanan AWS lainnya, mengurangi kebutuhan untuk beralih di antara antarmuka yang berbeda atau mengedit secara manual. ConfigMaps
- Gunakan entri akses untuk mengelola izin Kubernetes dari prinsipal IAM dari luar klaster. Anda dapat menambahkan dan mengelola akses ke cluster dengan menggunakan EKS API, AWS Command Line Interface, AWS, AWS SDKs CloudFormation, dan AWS Management Console. Ini berarti Anda dapat mengelola pengguna dengan alat yang sama dengan yang Anda buat dengan cluster.
- Manfaatkan otomatisasi seperti yang ditunjukkan dalam contoh ini untuk menerapkan cluster dengan AWS IAM Identity Center sebagai idP yang memiliki CAM API sebagai titik masuk.
- Izin Kubernetes granular dapat diterapkan dengan memetakan pengguna atau grup Kubernetes dengan prinsipal IAM yang terkait dengan identitas SSO melalui entri akses dan kebijakan akses.
- Untuk memulai, ikuti Ubah mode autentikasi untuk menggunakan entri akses, lalu Memigrasi entri ConfigMap aws-auth yang ada untuk mengakses entri.

Jadikan EKS Cluster Endpoint pribadi

Secara default saat Anda menyediakan kluster EKS, titik akhir klaster API disetel ke publik, yaitu dapat diakses dari Internet. Meskipun dapat diakses dari Internet, titik akhir masih dianggap aman karena mengharuskan semua permintaan API untuk diautentikasi oleh IAM dan kemudian disahkan oleh Kubernetes RBAC. Karena itu, jika kebijakan keamanan perusahaan Anda mengamanatkan bahwa Anda membatasi akses ke API dari Internet atau mencegah Anda merutekan lalu lintas di luar VPC klaster, Anda dapat:

- Konfigurasikan titik akhir kluster EKS menjadi pribadi. Lihat Memodifikasi Akses Titik Akhir Cluster untuk informasi lebih lanjut tentang topik ini.
- Biarkan titik akhir cluster publik dan tentukan blok CIDR mana yang dapat berkomunikasi dengan titik akhir cluster. Blok tersebut secara efektif merupakan kumpulan alamat IP publik yang masuk daftar putih yang diizinkan untuk mengakses titik akhir cluster.
- Konfigurasikan akses publik dengan satu set blok CIDR yang masuk daftar putih dan setel akses titik akhir pribadi ke diaktifkan. Ini akan memungkinkan akses publik dari rentang publik tertentu IPs sambil memaksa semua lalu lintas jaringan antara kubelet (pekerja) dan Kubernetes API melalui akun silang yang akan disediakan ke dalam ENIs VPC klaster ketika bidang kontrol disediakan.

Jangan gunakan token akun layanan untuk otentikasi

Token akun layanan adalah kredensi statis yang berumur panjang. Jika dikompromikan, hilang, atau dicuri, penyerang mungkin dapat melakukan semua tindakan yang terkait dengan token tersebut hingga akun layanan dihapus. Terkadang, Anda mungkin perlu memberikan pengecualian untuk aplikasi yang harus menggunakan Kubernetes API dari luar klaster, misalnya aplikasi pipeline CI/CD. Jika aplikasi tersebut berjalan pada infrastruktur AWS, seperti instans EC2, pertimbangkan untuk menggunakan profil instans dan memetakannya ke peran Kubernetes RBAC.

Gunakan akses yang paling tidak istimewa ke AWS Resources

Pengguna IAM tidak perlu diberi hak istimewa ke sumber daya AWS untuk mengakses Kubernetes API. Jika Anda perlu memberikan akses pengguna IAM ke kluster EKS, buat entri `aws-auth` ConfigMap untuk pengguna yang memetakan ke grup RBAC Kubernetes tertentu.

Hapus izin cluster-admin dari prinsipal pembuat klaster

Secara default, klaster Amazon EKS dibuat dengan `cluster-admin` izin permanen yang terikat pada prinsipal pembuat klaster. Dengan Cluster Access Manager API,

dimungkinkan untuk membuat cluster tanpa izin ini menyetel `--access-config bootstrapClusterCreatorAdminPermissions kefalse`, saat menggunakan `API_AND_CONFIG_MAP` atau mode `API` otentikasi. Cabut akses ini yang dianggap sebagai praktik terbaik untuk menghindari perubahan yang tidak diinginkan pada konfigurasi cluster. Proses untuk mencabut akses ini, mengikuti proses yang sama untuk mencabut akses lain ke cluster.

API memberi Anda fleksibilitas untuk hanya memisahkan prinsipal IAM dari Kebijakan Akses, dalam hal ini. `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
```

Atau sepenuhnya menghapus Entri Akses yang terkait dengan `cluster-admin` izin.

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

{
    "accessEntries": []
}

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

Akses ini dapat diberikan lagi jika diperlukan selama skenario insiden, darurat atau pecah kaca di mana cluster tidak dapat diakses.

Jika klaster masih dikonfigurasi dengan metode `CONFIG_MAP` otentikasi, semua pengguna tambahan harus diberikan akses ke cluster melalui `aws-auth` ConfigMap, dan setelah `aws-auth` ConfigMap dikonfigurasi, peran yang ditetapkan ke entitas yang membuat cluster, dapat dihapus dan hanya dibuat ulang jika terjadi insiden, skenario darurat atau pecah kaca, atau di mana rusak dan klaster `aws-auth` ConfigMap tidak dapat diakses. Ini bisa sangat berguna dalam kelompok produksi.

Gunakan Peran IAM saat beberapa pengguna membutuhkan akses identik ke cluster

Daripada membuat entri untuk setiap Pengguna IAM individu, izinkan pengguna tersebut untuk mengambil Peran IAM dan memetakan peran itu ke grup RBAC Kubernetes. Ini akan lebih mudah dipertahankan, terutama karena jumlah pengguna yang membutuhkan akses bertambah.

> **Important**
>
> Saat mengakses kluster EKS dengan entitas IAM yang dipetakan oleh `aws-auth` ConfigMap, nama pengguna yang dijelaskan dicatat di bidang pengguna log audit Kubernetes. Jika Anda menggunakan peran IAM, pengguna aktual yang menganggap peran tersebut tidak direkam dan tidak dapat diaudit.

Jika masih menggunakan `aws-auth` ConfigMap sebagai metode otentikasi, saat menetapkan izin K8s RBAC ke peran IAM, Anda harus menyertakan\ {{}} di nama pengguna Anda. SessionName Dengan begitu, log audit akan merekam nama sesi sehingga Anda dapat melacak siapa pengguna yang sebenarnya mengambil peran ini bersama dengan CloudTrail log.

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

Di Kubernetes 1.20 dan di atasnya, perubahan ini tidak lagi diperlukan, karena `user.extra.sessionName.0` telah ditambahkan ke log audit Kubernetes.

Gunakan akses yang paling tidak memiliki hak istimewa saat membuat dan RoleBindings ClusterRoleBindings

Seperti poin sebelumnya tentang pemberian akses ke AWS Resources, RoleBindings dan ClusterRoleBindings seharusnya hanya menyertakan serangkaian izin yang diperlukan untuk menjalankan fungsi tertentu. Hindari menggunakan `["*"]` peran Anda dan ClusterRoles kecuali itu benar-benar diperlukan. Jika Anda tidak yakin izin apa yang akan ditetapkan, pertimbangkan untuk menggunakan alat seperti audit2rbac untuk secara otomatis menghasilkan Peran dan pengikatan berdasarkan panggilan API yang diamati di Log Audit Kubernetes.

Buat cluster menggunakan proses otomatis

Seperti yang terlihat pada langkah sebelumnya, saat membuat klaster Amazon EKS, jika tidak menggunakan mode penggunaan `API_AND_CONFIG_MAP` atau `API` otentikasi, dan tidak memilih untuk mendelegasikan `cluster-admin` izin ke pembuat klaster, pengguna atau peran entitas IAM, seperti pengguna federasi yang membuat cluster, secara otomatis diberikan `system:masters` izin dalam konfigurasi RBAC cluster. Bahkan menjadi praktik terbaik untuk menghapus izin ini, seperti yang dijelaskan di sini jika menggunakan metode `CONFIG_MAP` otentikasi, mengandalkan `aws-auth` ConfigMap, akses ini tidak dapat dicabut. Oleh karena itu, merupakan ide yang baik untuk membuat cluster dengan pipa otomatisasi infrastruktur yang terkait dengan peran IAM khusus, tanpa izin untuk diasumsikan oleh pengguna atau entitas lain dan secara teratur mengaudit izin, kebijakan, dan siapa yang memiliki akses untuk memicu pipeline. Selain itu, peran ini tidak boleh digunakan untuk melakukan tindakan rutin di cluster, dan secara eksklusif digunakan untuk tindakan tingkat cluster yang dipicu oleh pipeline, melalui perubahan kode SCM misalnya.

Buat cluster dengan peran IAM khusus

Saat Anda membuat klaster Amazon EKS, pengguna atau peran entitas IAM, seperti pengguna federasi yang membuat klaster, secara otomatis diberikan `system:masters` izin dalam konfigurasi RBAC cluster. Akses ini tidak dapat dihapus dan tidak dikelola melalui `aws-auth` ConfigMap. Oleh karena itu merupakan ide yang baik untuk membuat cluster dengan peran IAM khusus dan secara teratur mengaudit siapa yang dapat mengambil peran ini. Peran ini tidak boleh digunakan untuk melakukan tindakan rutin di cluster, dan sebagai gantinya pengguna tambahan harus diberikan akses ke cluster melalui `aws-auth` ConfigMap untuk tujuan ini. Setelah `aws-auth` ConfigMap dikonfigurasi, peran harus diamankan dan hanya digunakan dalam mode hak istimewa sementara yang ditinggikan /break glass untuk skenario di mana cluster tidak dapat diakses. Ini bisa sangat berguna dalam cluster yang tidak memiliki akses pengguna langsung yang dikonfigurasi.

Secara teratur mengaudit akses ke cluster

Siapa yang membutuhkan akses kemungkinan akan berubah seiring waktu. Berencana untuk mengaudit secara berkala `aws-auth` ConfigMap untuk melihat siapa yang telah diberikan akses dan hak yang telah diberikan kepada mereka. Anda juga dapat menggunakan alat open source seperti kubectl-who-can, atau rbac-lookup untuk memeriksa peran yang terikat pada akun layanan, pengguna, atau grup tertentu. Kami akan mengeksplorasi topik ini lebih lanjut ketika kami sampai ke bagian tentang audit. Ide tambahan dapat ditemukan di artikel ini dari NCC Group.

Jika mengandalkan **`aws-auth`** ConfigMap gunakan alat untuk membuat perubahan

Aws-auth yang diformat dengan tidak benar ConfigMap dapat menyebabkan Anda kehilangan akses ke cluster. Jika Anda perlu membuat perubahan pada ConfigMap, gunakan alat.

eksctl `eksctl` CLI menyertakan perintah untuk menambahkan pemetaan identitas ke aws-auth. ConfigMap

Lihat Bantuan CLI:

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

Periksa identitas yang dipetakan ke Amazon EKS Cluster Anda.

```
$ 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
```

Jadikan Peran IAM sebagai Admin Cluster:

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

Untuk informasi lebih lanjut, tinjau `eksctl`dokumen

aws-auth oleh keikoproj

`aws-auth`oleh keikoproj mencakup perpustakaan cli dan go.

Unduh dan lihat bantuan bantuan CLI:

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

Atau, instal `aws-auth` dengan pengelola plugin krew untuk kubectl.

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

Tinjau dokumen aws-auth GitHub untuk informasi lebih lanjut, termasuk perpustakaan go.

AWS IAM Authenticator CLI

`aws-iam-authenticator`Proyek ini mencakup CLI untuk memperbarui. ConfigMap

Unduh rilis di GitHub.

Tambahkan izin klaster ke Peran 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
...
```

Pendekatan Alternatif untuk Otentikasi dan Manajemen Akses

Meskipun IAM adalah cara yang lebih disukai untuk mengautentikasi pengguna yang membutuhkan akses ke cluster EKS, dimungkinkan untuk menggunakan penyedia identitas OIDC seperti GitHub menggunakan proxy otentikasi dan peniruan identitas Kubernetes. Posting untuk dua solusi tersebut telah dipublikasikan di blog AWS Open Source:

- Mengautentikasi ke EKS Menggunakan GitHub Kredensyal dengan Teleport
- Otentikasi OIDC yang konsisten di beberapa kluster EKS menggunakan kube-oidc-proxy

> **Important**
>
> EKS secara native mendukung otentikasi OIDC tanpa menggunakan proxy. Untuk informasi lebih lanjut, silakan baca blog peluncuran, Memperkenalkan otentikasi penyedia identitas OIDC untuk Amazon EKS. Untuk contoh yang menunjukkan cara mengonfigurasi EKS dengan Dex, penyedia OIDC open source populer dengan konektor untuk berbagai metode keaslian yang berbeda, lihat Menggunakan Dex & dex-k8s-authenticator untuk mengautentikasi ke Amazon EKS. Seperti yang dijelaskan di blog, pengguna yang username/group diautentikasi oleh penyedia OIDC akan muncul di log audit Kubernetes.

Anda juga dapat menggunakan AWS SSO untuk menggabungkan AWS dengan penyedia identitas eksternal, misalnya Azure AD. Jika Anda memutuskan untuk menggunakan ini, AWS CLI v2.0 menyertakan opsi untuk membuat profil bernama yang memudahkan untuk mengaitkan sesi SSO dengan sesi CLI Anda saat ini dan mengambil peran IAM. Ketahuilah bahwa Anda harus mengambil peran sebelum menjalankan `kubectl` karena peran IAM digunakan untuk menentukan grup RBAC Kubernetes pengguna.

Identitas dan Kredensional untuk Pod EKS

Aplikasi tertentu yang berjalan di dalam klaster Kubernetes memerlukan izin untuk memanggil Kubernetes API agar berfungsi dengan baik. Misalnya, AWS Load Balancer Controller harus dapat mencantumkan Endpoint Layanan. Pengontrol juga harus dapat memanggil AWS APIs untuk menyediakan dan mengonfigurasi ALB. Di bagian ini kita akan mengeksplorasi praktik terbaik untuk menetapkan hak dan hak istimewa ke Pod.

Akun Layanan Kubernetes

Service account adalah jenis objek khusus yang memungkinkan Anda untuk menetapkan peran Kubernetes RBAC ke sebuah pod. Akun layanan default dibuat secara otomatis untuk setiap Namespace dalam sebuah cluster. Saat Anda menerapkan pod ke Namespace tanpa mereferensikan akun layanan tertentu, akun layanan default untuk Namespace tersebut akan secara otomatis ditetapkan ke Pod dan Secret, yaitu token akun layanan (JWT) untuk akun layanan tersebut, akan dipasang ke pod sebagai volume di. `/var/run/secrets/kubernetes.io/serviceaccount` Menguraikan kode token akun layanan di direktori itu akan mengungkapkan metadata berikut:

```
{
  "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"
}
```

Akun layanan default memiliki izin berikut untuk Kubernetes API.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
```

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

Peran ini memberi wewenang kepada pengguna yang tidak diautentikasi dan diautentikasi untuk membaca informasi API dan dianggap aman agar dapat diakses publik.

Ketika sebuah aplikasi yang berjalan di dalam sebuah Pod memanggil Kubernetes APIs, Pod tersebut perlu diberi service account yang secara eksplisit memberikan izin untuk memanggilnya. APIs Mirip dengan pedoman untuk akses pengguna, Peran atau ClusterRole terikat ke akun layanan harus dibatasi pada sumber daya API dan metode yang diperlukan aplikasi untuk berfungsi dan tidak ada yang lain. Untuk menggunakan akun layanan non-default cukup atur `spec.serviceAccountName` bidang Pod ke nama akun layanan yang ingin Anda gunakan. Untuk informasi tambahan tentang membuat akun layanan, lihat https://kubernetes. io/docs/reference/access-authn-authz/rbac/#service-account-permissions.

> **Note**
>
> Sebelum Kubernetes 1.24, Kubernetes akan secara otomatis membuat rahasia untuk setiap akun layanan. Rahasia ini dipasang ke pod di/var/run/secrets/kubernetes.io/serviceaccountdan akan digunakan oleh pod untuk mengautentikasi ke server API

Kubernetes. Di Kubernetes 1.24, token akun layanan dibuat secara dinamis ketika pod berjalan dan hanya berlaku selama satu jam secara default. Rahasia untuk akun layanan tidak akan dibuat. Jika Anda memiliki aplikasi yang berjalan di luar klaster yang perlu mengautentikasi ke Kubernetes API, misalnya Jenkins, Anda perlu membuat secret of type `kubernetes.io/service-account-token` bersama dengan anotasi yang mereferensikan akun layanan seperti. `metadata.annotations.kubernetes.io/service-account.name: <SERVICE_ACCOUNT_NAME>` Rahasia yang dibuat dengan cara ini tidak kedaluwarsa.

Peran IAM untuk Akun Layanan (IRSA)

IRSA adalah fitur yang memungkinkan Anda untuk menetapkan peran IAM ke akun layanan Kubernetes. Ia bekerja dengan memanfaatkan fitur Kubernetes yang dikenal sebagai Proyeksi Volume Token Akun Layanan. Ketika Pod dikonfigurasi dengan Service Account yang mereferensikan Peran IAM, server API Kubernetes akan memanggil titik akhir penemuan OIDC publik untuk klaster saat startup. Titik akhir secara kriptografis menandatangani token OIDC yang dikeluarkan oleh Kubernetes dan token yang dihasilkan dipasang sebagai volume. Token yang ditandatangani ini memungkinkan Pod untuk memanggil peran IAM APIs terkait AWS. Saat AWS API dipanggil, AWS akan SDKs memanggil`sts:AssumeRoleWithWebIdentity`. Setelah memvalidasi tanda tangan token, IAM menukar token yang dikeluarkan Kubernetes dengan kredensi peran AWS sementara.

Saat menggunakan IRSA, penting untuk menggunakan kembali sesi AWS SDK untuk menghindari panggilan yang tidak diperlukan ke AWS STS.

Decoding token (JWT) untuk IRSA akan menghasilkan output yang mirip dengan contoh yang Anda lihat di bawah ini:

```
{
  "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"
}
```

Token khusus ini memberikan hak istimewa hanya tampilan Pod ke S3 dengan mengasumsikan peran IAM. Saat aplikasi mencoba membaca dari S3, token ditukar dengan seperangkat kredensyal IAM sementara yang menyerupai ini:

```
{
    "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"
    }
}
```

Webhook yang bermutasi yang berjalan sebagai bagian dari control plane EKS akan menyuntikkan AWS Role ARN dan path ke file token identitas web ke dalam Pod sebagai variabel lingkungan. Nilai-nilai ini juga dapat diberikan secara manual.

```
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
```

Kubelet akan secara otomatis memutar token yang diproyeksikan ketika lebih tua dari 80% dari total TTL, atau setelah 24 jam. AWS SDKs bertanggung jawab untuk memuat ulang token saat diputar. Untuk informasi lebih lanjut tentang IRSA, lihat https://docs.aws.amazon.com/eks/ latest/userguide/ iam - roles-for-service-accounts -technical-overview.html.

Identitas Pod EKS

EKS Pod Identities adalah fitur yang diluncurkan di re:Invent 2023 yang memungkinkan Anda menetapkan peran IAM ke akun layanan kubernetes, tanpa perlu mengonfigurasi penyedia identitas (IDP) Open Id Connect (OIDC) untuk setiap cluster di akun AWS Anda. Untuk menggunakan EKS Pod Identity, Anda harus menerapkan agen yang berjalan sebagai DaemonSet pod pada setiap node pekerja yang memenuhi syarat. Agen ini tersedia untuk Anda sebagai Add-on EKS dan merupakan prasyarat untuk menggunakan fitur EKS Pod Identity. Aplikasi Anda harus menggunakan AWS SDK versi yang didukung untuk menggunakan fitur ini.

Ketika EKS Pod Identities dikonfigurasi untuk sebuah Pod, EKS akan me-mount dan me-refresh token identitas pod di`/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token`. Token ini akan digunakan oleh AWS SDK untuk berkomunikasi dengan Agen Identitas Pod EKS, yang menggunakan token identitas pod dan peran IAM agen untuk membuat kredensil sementara untuk pod Anda dengan memanggil API. AssumeRoleForPodIdentity Token identitas pod yang dikirimkan ke pod Anda adalah JWT yang dikeluarkan dari cluster EKS Anda dan ditandatangani secara kriptografis, dengan klaim JWT yang sesuai untuk digunakan dengan Identitas Pod EKS.

Untuk mempelajari lebih lanjut tentang EKS Pod Identities, silakan lihat blog ini.

Anda tidak perlu melakukan modifikasi apa pun pada kode aplikasi Anda untuk menggunakan EKS Pod Identities. Versi AWS SDK yang didukung akan secara otomatis menemukan kredensyal yang tersedia dengan EKS Pod Identities dengan menggunakan rantai penyedia kredensyal. Seperti IRSA, identitas pod EKS menetapkan variabel di dalam pod Anda untuk mengarahkan mereka cara menemukan kredensyal AWS.

Bekerja dengan peran IAM untuk EKS Pod Identities

- EKS Pod Identities hanya dapat secara langsung mengambil peran IAM yang termasuk dalam akun AWS yang sama dengan cluster EKS. Untuk mengakses peran IAM di akun AWS lain, Anda harus mengambil peran tersebut dengan mengonfigurasi profil dalam konfigurasi SDK Anda, atau dalam kode aplikasi Anda.
- Ketika EKS Pod Identities sedang dikonfigurasi untuk Akun Layanan, orang atau proses yang mengonfigurasi Asosiasi Identitas Pod harus memiliki `iam:PassRole` hak untuk peran tersebut.
- Setiap Akun Layanan hanya dapat memiliki satu peran IAM yang terkait dengannya melalui Identitas Pod EKS, namun Anda dapat mengaitkan peran IAM yang sama dengan beberapa akun layanan.
- Peran IAM yang digunakan dengan EKS Pod Identities harus mengizinkan `pods.eks.amazonaws.com` Service Principal untuk mengasumsikannya, dan menetapkan tag sesi. Berikut ini adalah contoh kebijakan kepercayaan peran yang memungkinkan EKS Pod Identities menggunakan peran IAM:

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

AWS merekomendasikan penggunaan kunci kondisi seperti `aws:SourceOrgId` membantu melindungi dari masalah deputi lintas layanan yang membingungkan. Dalam contoh kebijakan

kepercayaan peran di atas, variabel `ResourceOrgId` adalah variabel yang sama dengan AWS Organizations Organization ID dari AWS Organization yang menjadi milik akun AWS. EKS akan memberikan nilai yang `aws:SourceOrgId` sama dengan itu ketika mengasumsikan peran dengan EKS Pod Identities.

Identitas Pod ABAC dan EKS

Ketika EKS Pod Identities mengasumsikan peran IAM, ia menetapkan tag sesi berikut:

Tag Sesi Identitas Pod EKS	Nilai
kubernetes-namespace	Namespace yang digunakan pod yang terkait dengan EKS Pod Identities.
kubernetes-service-account	Nama akun layanan kubernetes yang terkait dengan EKS Pod Identities
eks-cluster-arn	ARN dari kluster EKS, mis. `arn:${Partition}:eks:${Region}:${Account}:cluster/${ClusterName}` ARN cluster unik, tetapi jika cluster dihapus dan dibuat ulang di wilayah yang sama dengan nama yang sama, dalam akun AWS yang sama, ia akan memiliki ARN yang sama.
eks-cluster-name	Nama klaster EKS. Harap dicatat bahwa nama klaster EKS dapat sama di dalam akun AWS Anda, dan kluster EKS di akun AWS lainnya.
kubernetes-pod-name	Nama pod di EKS.
kubernetes-pod-uid	UID pod di EKS.

Tag sesi ini memungkinkan Anda menggunakan Attribute Based Access Control (ABAC) untuk memberikan akses ke resource AWS Anda hanya ke akun layanan kubernetes tertentu. Saat melakukannya, sangat penting untuk dipahami bahwa akun layanan kubernetes hanya unik di dalam namespace, dan namespace kubernetes hanya unik di dalam cluster EKS. Tag sesi ini dapat diakses

dalam kebijakan AWS dengan menggunakan kunci kondisi `aws:PrincipalTag/<tag-key>` global, seperti `aws:PrincipalTag/eks-cluster-arn`

Misalnya, jika Anda ingin memberikan akses ke hanya akun layanan tertentu untuk mengakses sumber daya AWS di akun Anda dengan IAM atau kebijakan sumber daya, Anda perlu memeriksa `eks-cluster-arn` dan memberi `kubernetes-namespace` tag serta memastikan bahwa hanya akun layanan dari kluster yang dituju yang memiliki akses ke sumber daya tersebut karena cluster lain dapat memiliki identik `kubernetes-service-accounts` dan. `kubernetes-service-account` `kubernetes-namespaces`

Contoh kebijakan Bucket S3 ini hanya memberikan akses ke objek di bucket S3 yang dilampirkan, hanya jika, `kubernetes-service-accountkubernetes-namespace`, `eks-cluster-arn` semuanya memenuhi nilai yang diharapkan, tempat cluster EKS di-host di akun 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"
                }
            }
        }
    ]
}
```

EKS Pod Identitas dibandingkan dengan IRSA

Baik EKS Pod Identities dan IRSA adalah cara yang lebih disukai untuk mengirimkan kredensyal AWS sementara ke pod EKS Anda. Kecuali Anda memiliki kasus penggunaan khusus untuk IRSA, kami sarankan Anda menggunakan EKS Pod Identities saat menggunakan EKS. Tabel ini membantu membandingkan dua fitur.

#	Identitas Pod EKS	IRSA
Memerlukan izin untuk membuat IDP OIDC di akun AWS Anda?	Tidak	Ya
Membutuhkan pengaturan IDP unik per cluster	Tidak	Ya
Menetapkan tag sesi yang relevan untuk digunakan dengan ABAC	Ya	Tidak
Membutuhkan iam: PassRole Periksa?	Ya	Tidak
Menggunakan Kuota AWS STS dari akun AWS Anda?	Tidak	Ya
Dapat mengakses akun AWS lainnya	Secara tidak langsung dengan rantai peran	Langsung dengan sts: AssumeRoleWithWebIdentity
Kompatibel dengan AWS SDKs	Ya	Ya
Memerlukan Daemonset Agen Identitas Pod pada node?	Ya	Tidak

Identitas dan Kredensyal untuk Rekomendasi Pod EKS

Perbarui daemonset aws-node untuk menggunakan IRSA

Saat ini, daemonset aws-node dikonfigurasi untuk menggunakan peran yang ditetapkan ke instans EC2 untuk ditetapkan ke pod. IPs Peran ini mencakup beberapa kebijakan yang dikelola AWS, misalnya Amazoneks_CNI_Policy EC2 ContainerRegistryReadOnly dan yang secara efektif memungkinkan semua pod yang berjalan pada node ke, alamat IP, atau menarik gambar dari ECR. attach/detach ENIs assign/unassign Karena ini menimbulkan risiko bagi cluster Anda, Anda disarankan untuk memperbarui daemonset aws-node untuk menggunakan IRSA. Skrip untuk melakukan ini dapat ditemukan di repositori untuk panduan ini.

Daemonset aws-node mendukung EKS Pod Identities dalam versi v1.15.5 dan yang lebih baru.

Batasi akses ke profil instance yang ditetapkan ke node pekerja

Ketika Anda menggunakan IRSA atau EKS Pod Identities, Pod akan memperbarui rantai kredensialnya untuk menggunakan IRSA atau EKS Pod Identities terlebih dahulu, namun pod tersebut masih dapat mewarisi hak dari profil instance yang ditetapkan ke node pekerja. Untuk pod yang tidak memerlukan izin ini, Anda dapat memblokir akses ke metadata instance untuk membantu memastikan bahwa aplikasi Anda hanya memiliki izin yang diperlukan, dan bukan node mereka.

> **Warning**
>
> Memblokir akses ke metadata instance akan mencegah pod yang tidak menggunakan IRSA atau EKS Pod Identities mewarisi peran yang ditetapkan ke node pekerja.

Anda dapat memblokir akses ke metadata instance dengan mengharuskan instance IMDSv2 hanya menggunakan dan memperbarui jumlah hop ke 1 seperti pada contoh di bawah ini. Anda juga dapat menyertakan pengaturan ini dalam template peluncuran grup node. Jangan nonaktifkan metadata instance karena ini akan mencegah komponen seperti pengendali terminasi node dan hal-hal lain yang mengandalkan metadata instance agar tidak berfungsi dengan baik.

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

Jika Anda menggunakan Terraform untuk membuat templat peluncuran untuk digunakan dengan Grup Node Terkelola, tambahkan blok metadata untuk mengonfigurasi jumlah hop seperti yang terlihat dalam cuplikan kode ini:

```
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" } …
```

Anda juga dapat memblokir akses pod ke metadata EC2 dengan memanipulasi iptables pada node. Untuk informasi lebih lanjut tentang metode ini, lihat Membatasi akses ke layanan metadata instance.

Jika Anda memiliki aplikasi yang menggunakan AWS SDK versi lama yang tidak mendukung IRSA atau EKS Pod Identities, Anda harus memperbarui versi SDK.

Cakupan kebijakan kepercayaan Peran IAM untuk Peran IRSA ke nama akun layanan, namespace, dan klaster

Kebijakan kepercayaan dapat dicakup ke Namespace atau akun layanan tertentu dalam Namespace. Saat menggunakan IRSA, yang terbaik adalah membuat kebijakan kepercayaan peran seeksplisit mungkin dengan memasukkan nama akun layanan. Ini akan secara efektif mencegah Pod lain dalam Namespace yang sama untuk mengambil peran tersebut. CLI `eksctl` akan melakukan ini secara otomatis ketika Anda menggunakannya untuk membuat peran layanan accounts/IAM . Lihat https:// eksctl. io/usage/iamserviceaccounts/untuk informasi lebih lanjut.

Saat bekerja dengan IAM secara langsung, ini menambahkan kondisi ke dalam kebijakan kepercayaan peran yang menggunakan kondisi untuk memastikan `:sub` klaim adalah namespace dan akun layanan yang Anda harapkan. Sebagai contoh, sebelum kita memiliki token IRSA dengan sub klaim “system:serviceaccount:default:s3-read-only”. Ini adalah `default` namespace dan akun layanannya. `s3-read-only` Anda akan menggunakan kondisi seperti berikut ini untuk memastikan bahwa hanya akun layanan Anda di namespace tertentu dari cluster Anda yang dapat mengambil peran itu:

```
 "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"
     }
```

```
  }
```

Gunakan satu peran IAM per aplikasi

Dengan IRSA dan EKS Pod Identity, ini adalah praktik terbaik untuk memberikan masing-masing aplikasi peran IAM sendiri. Ini memberi Anda isolasi yang lebih baik karena Anda dapat memodifikasi satu aplikasi tanpa memengaruhi yang lain, dan memungkinkan Anda menerapkan prinsip hak istimewa paling sedikit dengan hanya memberikan izin yang dibutuhkan aplikasi.

Saat menggunakan ABAC dengan EKS Pod Identity, Anda dapat menggunakan peran IAM umum di beberapa akun layanan dan mengandalkan atribut sesi mereka untuk kontrol akses. Ini sangat berguna saat beroperasi dalam skala besar, karena ABAC memungkinkan Anda beroperasi dengan peran IAM yang lebih sedikit.

Saat aplikasi Anda membutuhkan akses ke IMDS, gunakan IMDSv2 dan tingkatkan batas hop pada instans EC2 menjadi 2

IMDSv2mengharuskan Anda menggunakan permintaan PUT untuk mendapatkan token sesi. Permintaan PUT awal harus menyertakan TTL untuk token sesi. Versi AWS SDK yang lebih baru akan menangani ini dan pembaruan token tersebut secara otomatis. Penting juga untuk diperhatikan bahwa batas hop default pada instans EC2 sengaja disetel ke 1 untuk mencegah penerusan IP. Akibatnya, Pod yang meminta token sesi yang dijalankan pada instans EC2 pada akhirnya dapat habis waktu dan mundur menggunakan aliran data. IMDSv1 EKS menambahkan dukungan IMDSv2 dengan mengaktifkan v1 dan v2 dan mengubah batas hop menjadi 2 pada node yang disediakan oleh eksctl atau dengan templat resmi. CloudFormation

Nonaktifkan pemasangan otomatis token akun layanan

Jika aplikasi Anda tidak perlu memanggil Kubernetes API, setel `automountServiceAccountToken` atribut ke `false` in PodSpec for your application atau patch akun layanan default di setiap namespace sehingga tidak lagi terpasang ke pod secara otomatis. Contoh:

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

Gunakan akun layanan khusus untuk setiap aplikasi

Setiap aplikasi harus memiliki akun layanan khusus sendiri. Ini berlaku untuk akun layanan untuk Kubernetes API serta IRSA dan EKS Pod Identity.

> **Important**
>
> Jika Anda menggunakan blue/green pendekatan untuk upgrade klaster alih-alih melakukan upgrade klaster di tempat saat menggunakan IRSA, Anda perlu memperbarui kebijakan kepercayaan dari masing-masing peran IAM IRSA dengan titik akhir OIDC dari cluster baru. Upgrade blue/green klaster adalah tempat Anda membuat klaster yang menjalankan versi Kubernetes yang lebih baru di samping klaster lama dan menggunakan penyeimbang beban atau mesh layanan untuk mengalihkan lalu lintas dari layanan yang berjalan di cluster lama ke cluster baru dengan mulus. Saat menggunakan upgrade blue/green cluster dengan EKS Pod Identity, Anda akan membuat asosiasi identitas pod antara peran IAM dan akun layanan di cluster baru. Dan perbarui kebijakan kepercayaan peran IAM jika Anda memiliki `sourceArn` kondisi.

Jalankan aplikasi sebagai pengguna non-root

Kontainer berjalan sebagai root secara default. Meskipun ini memungkinkan mereka untuk membaca file token identitas web, menjalankan wadah sebagai root tidak dianggap sebagai praktik terbaik. Sebagai alternatif, pertimbangkan untuk menambahkan `spec.securityContext.runAsUser` atribut ke PodSpec. Nilai `runAsUser` adalah nilai arbitrer.

Dalam contoh berikut, semua proses di dalam Pod akan berjalan di bawah ID pengguna yang ditentukan dalam `runAsUser` bidang.

```
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" ]
```

Saat Anda menjalankan penampung sebagai pengguna non-root, penampung mencegah penampung membaca token akun layanan IRSA karena token diberi izin 0600 [root] secara default.

Jika Anda memperbarui SecurityContext untuk container Anda untuk menyertakan fsgroup=65534 [Nobody] itu akan memungkinkan penampung untuk membaca token.

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

Di Kubernetes 1.19 dan di atasnya, perubahan ini tidak lagi diperlukan dan aplikasi dapat membaca token akun layanan IRSA tanpa menambahkannya ke grup Nobody.

Berikan akses paling tidak istimewa ke aplikasi

Action Hero adalah utilitas yang dapat Anda jalankan bersama aplikasi Anda untuk mengidentifikasi panggilan AWS API dan izin IAM terkait yang dibutuhkan aplikasi Anda untuk berfungsi dengan baik. Ini mirip dengan IAM Access Advisor karena membantu Anda secara bertahap membatasi ruang lingkup peran IAM yang ditetapkan untuk aplikasi. Konsultasikan dokumentasi tentang pemberian akses istimewa paling sedikit ke sumber daya AWS untuk informasi lebih lanjut.

Pertimbangkan untuk menetapkan batas izin pada peran IAM yang digunakan dengan IRSA dan Pod Identities. Anda dapat menggunakan batas izin untuk memastikan bahwa peran yang digunakan oleh IRSA atau Pod Identities tidak dapat melebihi tingkat izin maksimum. Untuk panduan contoh tentang memulai batas izin dengan contoh kebijakan batas izin, silakan lihat repo github ini.

Tinjau dan cabut akses anonim yang tidak perlu ke kluster EKS Anda

Idealnya akses anonim harus dinonaktifkan untuk semua tindakan API. Akses anonim diberikan dengan membuat RoleBinding atau ClusterRoleBinding untuk sistem pengguna bawaan Kubernetes:anonim. Anda dapat menggunakan alat rbac-lookup untuk mengidentifikasi izin yang dimiliki pengguna system:anonymous di cluster Anda:

```
./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
```

Peran apa pun atau ClusterRole selain sistem: tidak public-info-viewer boleh terikat pada sistem: pengguna anonim atau grup system:unauthenticated.

Mungkin ada beberapa alasan yang sah untuk mengaktifkan akses anonim secara spesifik APIs. Jika hal ini terjadi pada klaster Anda, pastikan bahwa hanya yang spesifik APIs yang dapat diakses oleh pengguna anonim dan mengekspos mereka APIs tanpa otentikasi tidak membuat cluster Anda rentan.

Sebelum Kubernetes/EKS Versi 1.14, grup system:unauthenticated dikaitkan dengan system:discovery dan system:basic-user secara default. ClusterRoles Perhatikan bahwa meskipun Anda telah memperbarui klaster ke versi 1.14 atau lebih tinggi, izin ini mungkin masih diaktifkan di klaster Anda, karena pembaruan klaster tidak mencabut izin ini. Untuk memeriksa yang ClusterRoles memiliki “system:unauthenticated” kecuali system: public-info-viewer Anda dapat menjalankan perintah berikut (membutuhkan jq util):

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

Dan “system:unauthenticated” dapat dihapus dari semua peran kecuali “system:" menggunakan: 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 -
```

Atau, Anda dapat memeriksa dan menghapusnya secara manual dengan kubectl describe dan kubectl edit. Untuk memeriksa apakah grup system:unauthenticated memiliki izin system:discovery di cluster Anda, jalankan perintah berikut:

```
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
```

Untuk memeriksa apakah system:unauthenticated group memiliki system:basic-user permission pada cluster Anda, jalankan perintah berikut:

```
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
```

Jika grup system:unauthenticated terikat pada system:discovery dan/atau system:basic-user di cluster Anda, Anda harus memisahkan peran ini dari grup ClusterRoles system:unauthenticated. Edit system:discovery ClusterRoleBinding menggunakan perintah berikut:

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

Perintah di atas akan membuka definisi system:discovery saat ini di editor seperti yang ditunjukkan ClusterRoleBinding di bawah ini:

```
# 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
```

Hapus entri untuk grup system:unauthenticated dari bagian “subyek” di layar editor di atas.

Ulangi langkah yang sama untuk ClusterRoleBinding system:basic-user.

Gunakan kembali sesi AWS SDK dengan IRSA

Saat Anda menggunakan IRSA, aplikasi yang ditulis menggunakan AWS SDK menggunakan token yang dikirimkan ke pod Anda `sts:AssumeRoleWithWebIdentity` untuk menelepon guna menghasilkan kredensyal AWS sementara. Ini berbeda dengan layanan komputasi AWS lainnya, di mana layanan komputasi memberikan kredensyal AWS sementara langsung ke sumber daya komputasi AWS, seperti fungsi lambda. Ini berarti bahwa setiap kali sesi AWS SDK diinisialisasi, panggilan ke AWS STS untuk `AssumeRoleWithWebIdentity` dilakukan. Jika aplikasi Anda menskalakan dengan cepat dan menginisialisasi banyak sesi AWS SDK, Anda mungkin mengalami throttling dari AWS STS karena kode Anda akan membuat banyak panggilan. `AssumeRoleWithWebIdentity`

Untuk menghindari skenario ini, sebaiknya gunakan kembali sesi AWS SDK dalam aplikasi Anda sehingga panggilan yang tidak perlu `AssumeRoleWithWebIdentity` tidak dilakukan.

Dalam contoh kode berikut, sesi dibuat menggunakan boto3 python SDK, dan sesi yang sama digunakan untuk membuat klien dan berinteraksi dengan Amazon S3 dan Amazon SQS. `AssumeRoleWithWebIdentity`hanya dipanggil sekali, dan AWS SDK akan menyegarkan kredensyal `my_session` kapan mereka kedaluwarsa secara otomatis.

```
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)
```

Jika Anda memigrasikan aplikasi dari layanan komputasi AWS lain, seperti EC2, ke EKS dengan IRSA, ini adalah detail yang sangat penting. Pada layanan komputasi lain yang menginisialisasi sesi AWS SDK tidak memanggil AWS STS kecuali Anda menginstruksikannya.

Pendekatan alternatif

Meskipun IRSA dan EKS Pod Identities adalah cara yang lebih disukai untuk menetapkan identitas AWS ke pod, mereka mengharuskan Anda menyertakan AWS versi terbaru SDKs dalam aplikasi Anda. Untuk daftar lengkap SDKs yang saat ini mendukung IRSA, lihat 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. Jika Anda memiliki aplikasi yang tidak dapat segera diperbarui dengan SDK yang kompatibel, ada beberapa solusi buatan komunitas yang tersedia untuk menetapkan peran IAM ke pod Kubernetes, termasuk kube2iam dan kiam. Meskipun AWS tidak mendukung, memaafkan, atau mendukung penggunaan solusi ini, mereka sering digunakan oleh komunitas pada umumnya untuk mencapai hasil yang sama seperti IRSA dan EKS Pod Identities.

Jika Anda perlu menggunakan salah satu solusi yang disediakan non-aws ini, harap lakukan uji tuntas dan pastikan Anda memahami implikasi keamanan dari melakukannya.

Alat dan Sumber Daya

- Workshop Perendaman Keamanan Amazon EKS - Identity and Access Management
- Pola Cetak Biru Terraform EKS - Cluster Amazon EKS Sepenuhnya Pribadi
- Pola Cetak Biru Terraform EKS - Pusat Identitas IAM Masuk Tunggal untuk Amazon EKS Cluster
- Pola Cetak Biru Terraform EKS - Okta Single Sign-On untuk Amazon EKS Cluster

- audit2rbac
- rbac.dev Daftar sumber daya tambahan, termasuk blog dan alat, untuk Kubernetes RBAC
- Pahlawan Aksi
- kube2iam
- kiam

Keamanan Pod

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Spesifikasi pod mencakup berbagai atribut berbeda yang dapat memperkuat atau melemahkan postur keamanan Anda secara keseluruhan. Sebagai praktisi Kubernetes, perhatian utama Anda adalah mencegah proses yang berjalan dalam wadah agar tidak lolos dari batas isolasi runtime container dan mendapatkan akses ke host yang mendasarinya.

Kemampuan Linux

Proses yang berjalan dalam wadah berjalan di bawah konteks pengguna root [Linux] secara default. Meskipun tindakan root dalam wadah sebagian dibatasi oleh serangkaian kemampuan Linux yang ditetapkan runtime kontainer ke wadah, hak istimewa default ini dapat memungkinkan penyerang untuk meningkatkan hak istimewa mereka.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

Pod EC2 dan Fargate diberi kemampuan yang disebutkan di atas secara default. Selain itu, kemampuan Linux hanya dapat dihapus dari pod Fargate.

Pod yang dijalankan sebagai hak istimewa, mewarisi semua kemampuan Linux yang terkait dengan root pada host. Ini harus dihindari jika memungkinkan.

Otorisasi Node

Semua node pekerja Kubernetes menggunakan mode otorisasi yang disebut Node Authorization. Otorisasi Node mengotorisasi semua permintaan API yang berasal dari kubelet dan memungkinkan node untuk melakukan tindakan berikut:

Baca operasi:

- layanan
- titik akhir
- simpul
- polong
- secret, configmaps, klaim volume persisten, dan volume persisten yang terkait dengan pod yang terikat pada node kubelet

Tulis operasi:

- node dan status node (aktifkan plugin `NodeRestriction` penerimaan untuk membatasi kubelet untuk memodifikasi simpulnya sendiri)
- pod dan status pod (aktifkan plugin `NodeRestriction` penerimaan untuk membatasi kubelet untuk memodifikasi pod yang terikat pada dirinya sendiri)
- acara

Operasi terkait Author:

- Akses baca/tulis ke API CertificateSigningRequest (CSR) untuk bootstrap TLS
- kemampuan untuk membuat TokenReview dan SubjectAccessReview untuk pemeriksaan yang didelegasikan authentication/authorization

EKS menggunakan node restriction admission controller yang hanya memungkinkan node untuk memodifikasi set terbatas atribut node dan objek pod yang terikat ke node. Namun demikian, penyerang yang berhasil mendapatkan akses ke host masih dapat mengumpulkan informasi sensitif

tentang lingkungan dari Kubernetes API yang memungkinkan mereka untuk bergerak secara lateral di dalam cluster.

Solusi Keamanan Pod

Kebijakan Keamanan Pod (PSP)

Di masa lalu, sumber daya Pod Security Policy (PSP) digunakan untuk menentukan serangkaian persyaratan yang harus dipenuhi oleh Pod sebelum dapat dibuat. Pada Kubernetes versi 1.21, PSP sudah tidak digunakan lagi. Mereka dijadwalkan untuk dihapus di Kubernetes versi 1.25.

> **Important**
>
> PSPs tidak digunakan lagi di Kubernetes versi 1.21. Anda akan memiliki hingga versi 1.25 atau kira-kira 2 tahun untuk transisi ke alternatif. Dokumen ini menjelaskan motivasi untuk penghentian ini.

Migrasi ke solusi keamanan pod baru

Karena PSPs telah dihapus pada Kubernetes v1.25, administrator dan operator klaster harus mengganti kontrol keamanan tersebut. Dua solusi dapat memenuhi kebutuhan ini:

- Policy-as-code (PAC) solusi dari ekosistem Kubernetes
- Standar Keamanan Pod Kubernetes (PSS)

Baik solusi PAC dan PSS dapat hidup berdampingan dengan PSP; mereka dapat digunakan dalam cluster sebelum PSP dihapus. Ini memudahkan adopsi saat bermigrasi dari PSP. Silakan lihat dokumen ini saat mempertimbangkan migrasi dari PSP ke PSS.

Kyverno, salah satu solusi PAC yang diuraikan di bawah ini, memiliki panduan khusus yang diuraikan dalam posting blog saat bermigrasi dari PSPs ke solusinya termasuk kebijakan analog, perbandingan fitur, dan prosedur migrasi. Informasi dan panduan tambahan tentang migrasi ke Kyverno sehubungan dengan Pod Security Admission (PSA) telah dipublikasikan di blog AWS di sini.

Policy-as-code (PAC)

Policy-as-code Solusi (PAC) menyediakan pagar pembatas untuk memandu pengguna klaster, dan mencegah perilaku yang tidak diinginkan, melalui kontrol yang ditentukan dan otomatis. PAC

menggunakan Kubernetes Dynamic Admission Controllers untuk mencegat alur permintaan server API Kubernetes, melalui panggilan webhook, dan mengubah dan memvalidasi payload permintaan, berdasarkan kebijakan yang ditulis dan disimpan sebagai kode. Mutasi dan validasi terjadi sebelum permintaan server API menghasilkan perubahan pada cluster. Solusi PAC menggunakan kebijakan untuk mencocokkan dan bertindak berdasarkan muatan permintaan server API, berdasarkan taksonomi dan nilai.

Ada beberapa solusi PAC open source yang tersedia untuk Kubernetes. Solusi ini bukan bagian dari proyek Kubernetes; mereka bersumber dari ekosistem Kubernetes. Beberapa solusi PAC tercantum di bawah ini.

- OPA/Penjaga gerbang
- Agen Kebijakan Terbuka (OPA)
- Kyverno
- Kubewarden
- JSPolicy

Untuk informasi lebih lanjut tentang solusi PAC dan cara membantu Anda memilih solusi yang tepat untuk kebutuhan Anda, lihat tautan di bawah ini.

- Penanggulangan berbasis kebijakan untuk Kubernetes — Bagian 1
- Penanggulangan berbasis kebijakan untuk Kubernetes — Bagian 2

Standar Keamanan Pod (PSS) dan Penerimaan Keamanan Pod (PSA)

Menanggapi penghentian PSP dan kebutuhan berkelanjutan untuk mengontrol keamanan pod out-of-the-box, dengan solusi Kubernetes bawaan, Kubernetes Auth Special Interest Group membuat Pod Security Standards (PSS) dan Pod Security Admission (PSA). Upaya PSA mencakup proyek webhook pengontrol penerimaan yang mengimplementasikan kontrol yang didefinisikan dalam PSS. Pendekatan pengontrol penerimaan ini menyerupai yang digunakan dalam solusi PAC.

Menurut dokumentasi Kubernetes, PSS “mendefinisikan tiga kebijakan berbeda untuk mencakup spektrum keamanan secara luas. Kebijakan ini bersifat kumulatif dan berkisar dari yang sangat permisif hingga sangat membatasi. `”

Kebijakan ini didefinisikan sebagai:

- Hak istimewa: Kebijakan tidak terbatas (tidak aman), memberikan tingkat izin seluas mungkin. Kebijakan ini memungkinkan eskalasi hak istimewa yang diketahui. Ini adalah tidak adanya kebijakan. Ini bagus untuk aplikasi seperti agen logging, driver penyimpanan CNIs, dan aplikasi luas sistem lainnya yang membutuhkan akses istimewa.
- Baseline: Kebijakan pembatasan minimal yang mencegah eskalasi hak istimewa yang diketahui. Mengizinkan konfigurasi Pod default (minimal ditentukan). Kebijakan dasar melarang penggunaan HostNetwork, HostPid, HostiPC, HostPath, HostPort, ketidakmampuan untuk menambahkan kemampuan Linux, bersama dengan beberapa batasan lainnya.
- Dibatasi: Kebijakan yang sangat dibatasi, mengikuti praktik terbaik pengerasan Pod saat ini. Kebijakan ini mewarisi dari baseline dan menambahkan pembatasan lebih lanjut seperti ketidakmampuan untuk berjalan sebagai root atau root-group. Kebijakan yang dibatasi dapat memengaruhi kemampuan aplikasi untuk berfungsi. Mereka terutama ditargetkan untuk menjalankan aplikasi keamanan kritis.

Kebijakan ini mendefinisikan profil untuk eksekusi pod, disusun menjadi tiga tingkat akses istimewa vs terbatas.

Untuk mengimplementasikan kontrol yang ditentukan oleh PSS, PSA beroperasi dalam tiga mode:

- menegakkan: Pelanggaran kebijakan akan menyebabkan pod ditolak.
- Audit: Pelanggaran kebijakan akan memicu penambahan anotasi audit pada peristiwa yang dicatat dalam log audit, tetapi sebaliknya diizinkan.
- peringatan: Pelanggaran kebijakan akan memicu peringatan yang dihadapi pengguna, tetapi sebaliknya diizinkan.

Mode-mode ini dan level profil (pembatasan) dikonfigurasi pada level Namespace Kubernetes, menggunakan label, seperti yang terlihat pada contoh di bawah ini.

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

Ketika digunakan secara independen, mode operasional ini memiliki respons berbeda yang menghasilkan pengalaman pengguna yang berbeda. Mode enforce akan mencegah pod dibuat jika

masing-masing PodSpecs melanggar level pembatasan yang dikonfigurasi. Namun, dalam mode ini, objek Kubernetes non-pod yang membuat pod, seperti Deployments, tidak akan dicegah untuk diterapkan ke cluster, bahkan jika PodSpec di dalamnya melanggar PSS yang diterapkan. Dalam hal ini Deployment akan diterapkan, sedangkan pod akan dicegah agar tidak diterapkan.

Ini adalah pengalaman pengguna yang sulit, karena tidak ada indikasi langsung bahwa objek Deployment yang berhasil diterapkan memungkiri pembuatan pod yang gagal. PodSpecs yang menyinggung tidak akan membuat pod. Memeriksa sumber daya Deployment dengan `kubectl get deploy <DEPLOYMENT_NAME> -oyaml` akan mengekspos pesan dari `.status.conditions` elemen pod yang gagal, seperti yang terlihat di bawah ini.

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

Baik dalam mode audit maupun warning, pembatasan pod tidak mencegah agar Pod tidak dibuat dan dimulai. Namun, dalam mode ini, anotasi audit pada server API, peristiwa log audit dan peringatan kepada klien server API, seperti kubectl, dipicu, masing-masing, ketika pod, serta objek yang membuat pod, berisi PodSpecs dengan pelanggaran. Pesan `kubectl` peringatan terlihat di bawah ini.

```
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
```

Mode audit dan peringatan PSA berguna saat memperkenalkan PSS tanpa berdampak negatif pada operasi klaster.

Mode operasional PSA tidak saling eksklusif, dan dapat digunakan secara kumulatif. Seperti yang terlihat di bawah ini, beberapa mode dapat dikonfigurasi dalam satu 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
```

Dalam contoh di atas, peringatan yang mudah digunakan dan anotasi audit diberikan saat menerapkan Deployment, sedangkan pemberlakuan pelanggaran juga disediakan di tingkat pod. Bahkan beberapa label PSA dapat menggunakan tingkat profil yang berbeda, seperti yang terlihat di bawah ini.

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

Dalam contoh di atas, PSA dikonfigurasi untuk memungkinkan pembuatan semua pod yang memenuhi tingkat profil dasar, dan kemudian memperingatkan pada pod (dan objek yang membuat pod) yang melanggar level profil terbatas. Ini adalah pendekatan yang berguna untuk menentukan dampak yang mungkin terjadi saat mengubah dari baseline ke profil terbatas.

Pod yang ada

Jika namespace dengan pod yang ada dimodifikasi untuk menggunakan profil PSS yang lebih ketat, mode audit dan warning akan menghasilkan pesan yang sesuai; namun, mode enforce tidak akan menghapus pod. Pesan peringatan terlihat di bawah ini.

```
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
```

Pengecualian

PSA menggunakan Pengecualian untuk mengecualikan penegakan pelanggaran terhadap pod yang seharusnya diterapkan. Pengecualian ini tercantum di bawah ini.

- Nama pengguna: permintaan dari pengguna dengan nama pengguna yang dikecualikan yang diautentikasi (atau ditiru) diabaikan.
- RuntimeClassNames: Pod dan sumber daya beban kerja yang menentukan nama kelas runtime yang dikecualikan akan diabaikan.
- Namespaces: Pod dan sumber daya beban kerja dalam namespace yang dikecualikan diabaikan.

Pengecualian ini diterapkan secara statis dalam konfigurasi pengontrol penerimaan PSA sebagai bagian dari konfigurasi server API.

Dalam implementasi Validating Webhook, pengecualian dapat dikonfigurasi dalam ConfigMapresource Kubernetes yang akan dipasang sebagai volume ke dalam container. 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"]
```

Seperti yang terlihat pada ConfigMap YAMAL di atas, level PSS default di seluruh cluster telah diatur untuk dibatasi untuk semua mode PSA, mengaudit, menegakkan, dan memperingatkan. Ini memengaruhi semua ruang nama, kecuali yang dikecualikan:. `namespaces: ["kube-system","policy-test1"]` Selain itu, dalam ValidatingWebhookConfigurationsumber daya, terlihat di bawah, pod-security-webhooknamespace juga dikecualikan dari PSS yang dikonfigurasi.

```
...
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

Penerimaan Pod Security lulus menjadi stabil di Kubernetes v1.25. Jika Anda ingin menggunakan fitur Pod Security Admission sebelum diaktifkan secara default, Anda perlu

menginstal pengontrol penerimaan dinamis (mutating webhook). Petunjuk untuk menginstal dan mengkonfigurasi webhook dapat ditemukan di sini.

Memilih antara policy-as-code dan Standar Keamanan Pod

Pod Security Standards (PSS) dikembangkan untuk menggantikan Pod Security Policy (PSP), dengan menyediakan solusi yang sudah ada di dalam Kubernetes dan tidak memerlukan solusi dari ekosistem Kubernetes. Karena itu, solusi policy-as-code (PAC) jauh lebih fleksibel.

Daftar Pro dan Kontra berikut ini dirancang untuk membantu Anda membuat keputusan yang lebih tepat tentang solusi keamanan pod Anda.

Policy-as-code (dibandingkan dengan Standar Keamanan Pod)

Kelebihan:

- Lebih fleksibel dan lebih terperinci (turun ke atribut sumber daya jika perlu)
- Tidak hanya berfokus pada pod, dapat digunakan untuk melawan sumber daya dan tindakan yang berbeda
- Tidak hanya diterapkan di tingkat namespace
- Lebih matang dari Standar Keamanan Pod
- Keputusan dapat didasarkan pada apa pun dalam payload permintaan server API, serta sumber daya cluster yang ada dan data eksternal (tergantung solusi)
- Mendukung mutasi permintaan server API sebelum validasi (tergantung solusi)
- Dapat menghasilkan kebijakan pelengkap dan sumber daya Kubernetes (tergantung solusi - Dari kebijakan pod, Kyverno dapat secara otomatis membuat kebijakan untuk pengontrol tingkat yang lebih tinggi, seperti Deployment. Kyverno juga dapat menghasilkan sumber daya Kubernetes tambahan “`ketika sumber daya baru dibuat atau ketika sumber diperbarui`” dengan menggunakan Generate Rules.)
- Dapat digunakan untuk beralih ke kiri, ke pipeline CICD, sebelum melakukan panggilan ke server API Kubernetes (tergantung solusi)
- Dapat digunakan untuk menerapkan perilaku yang belum tentu terkait keamanan, seperti praktik terbaik, standar organisasi, dll.
- Dapat digunakan dalam kasus penggunaan non-Kubernetes (tergantung solusi)
- Karena fleksibilitas, pengalaman pengguna dapat disesuaikan dengan kebutuhan pengguna

Kontra:

- Tidak dibangun di Kubernetes
- Lebih kompleks untuk dipelajari, dikonfigurasi, dan didukung
- Penulisan kebijakan mungkin memerlukan yang baru skills/languages/capabilities

Penerimaan Keamanan Pod (dibandingkan dengan policy-as-code)

Kelebihan:

- Dibangun di Kubernetes
- Lebih sederhana untuk dikonfigurasi
- Tidak ada bahasa baru untuk digunakan atau kebijakan untuk penulis
- Jika tingkat penerimaan default cluster dikonfigurasi ke privileged, label namespace dapat digunakan untuk memilih namespace ke dalam profil keamanan pod.

Kontra:

- Tidak sefleksibel atau granular policy-as-code
- Hanya 3 tingkat pembatasan
- Terutama berfokus pada pod

Ringkasan

Jika saat ini Anda tidak memiliki solusi keamanan pod, di luar PSP, dan postur keamanan pod yang Anda butuhkan sesuai dengan model yang ditentukan dalam Standar Keamanan Pod (PSS), maka jalur yang lebih mudah mungkin adalah mengadopsi PSS, sebagai pengganti solusi. policy-as-code Namun, jika postur keamanan pod Anda tidak sesuai dengan model PSS, atau Anda membayangkan menambahkan kontrol tambahan, di luar yang ditentukan oleh PSS, maka policy-as-code solusi akan tampak lebih cocok.

Rekomendasi

Gunakan beberapa mode Pod Security Admission (PSA) untuk pengalaman pengguna yang lebih baik

Seperti disebutkan sebelumnya, mode penegakan PSA mencegah Pod dengan pelanggaran PSS diterapkan, tetapi tidak menghentikan pengontrol tingkat yang lebih tinggi, seperti Deployment. Faktanya, Deployment akan berhasil diterapkan tanpa indikasi bahwa pod gagal diterapkan. Meskipun Anda dapat menggunakan kubectl untuk memeriksa objek Deployment, dan menemukan pesan pod yang gagal dari PSA, pengalaman pengguna bisa lebih baik. Untuk membuat pengalaman pengguna lebih baik, beberapa mode PSA (audit, menegakkan, memperingatkan) harus digunakan.

```
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
```

Dalam contoh di atas, dengan mode enforce yang ditentukan, ketika manifes Deployment dengan pelanggaran PSS di PodSpec masing-masing mencoba diterapkan ke server API Kubernetes, Deployment akan berhasil diterapkan, tetapi pod tidak akan berhasil. Dan, karena mode audit dan peringatan juga diaktifkan, klien server API akan menerima pesan peringatan dan peristiwa log audit server API akan dianotasi dengan pesan juga.

Batasi kontainer yang dapat berjalan sebagai hak istimewa

Seperti disebutkan, kontainer yang berjalan sebagai hak istimewa mewarisi semua kemampuan Linux yang ditugaskan untuk melakukan root pada host. Jarang kontainer membutuhkan jenis hak istimewa ini untuk berfungsi dengan baik. Ada beberapa metode yang dapat digunakan untuk membatasi izin dan kemampuan kontainer.

> **Important**
>
> Fargate adalah tipe peluncuran yang memungkinkan Anda menjalankan container “tanpa server” di mana kontainer pod dijalankan pada infrastruktur yang dikelola AWS. Dengan

Fargate, Anda tidak dapat menjalankan kontainer istimewa atau mengonfigurasi pod Anda untuk menggunakan HostNetwork atau HostPort.

Jangan menjalankan proses dalam wadah sebagai root

Semua kontainer berjalan sebagai root secara default. Ini bisa menjadi masalah jika penyerang dapat mengeksploitasi kerentanan dalam aplikasi dan mendapatkan akses shell ke wadah yang sedang berjalan. Anda dapat mengurangi risiko ini dengan berbagai cara. Pertama, dengan menghapus shell dari gambar kontainer. Kedua, menambahkan direktif USER ke Dockerfile Anda atau menjalankan container di pod sebagai pengguna non-root. Kubernetes PodSpec menyertakan sekumpulan bidang, di bawahnya`spec.securityContext`, yang memungkinkan Anda menentukan and/or grup pengguna untuk menjalankan aplikasi Anda. Bidang ini adalah `runAsUser` dan `runAsGroup` masing-masing.

Untuk menegakkan penggunaan`spec.securityContext`, dan elemen-elemen yang terkait, dalam Kubernetes PodSpec, policy-as-code atau Standar Keamanan Pod dapat ditambahkan ke klaster. Solusi ini memungkinkan Anda untuk menulis dan/atau menggunakan kebijakan atau profil yang dapat memvalidasi muatan permintaan server API Kubernetes masuk, sebelum disimpan ke etcd. Selain itu, policy-as-code solusi dapat mengubah permintaan masuk, dan dalam beberapa kasus, menghasilkan permintaan baru.

Jangan pernah menjalankan Docker di Docker atau memasang soket di wadah

Meskipun ini memungkinkan Anda untuk membuat build/run gambar dalam wadah Docker, pada dasarnya Anda melepaskan kendali penuh node ke proses yang berjalan di wadah. Jika Anda perlu membuat image kontainer di Kubernetes, gunakan Kaniko, buildah, atau layanan build seperti. CodeBuild

> **Note**
>
> Cluster Kubernetes yang digunakan untuk pemrosesan CICD, seperti membangun image container, harus diisolasi dari cluster yang menjalankan beban kerja yang lebih umum.

Batasi penggunaan hostPath atau jika hostPath diperlukan batasi awalan mana yang dapat digunakan dan konfigurasikan volume sebagai hanya-baca

`hostPath`adalah volume yang memasang direktori dari host langsung ke wadah. Jarang pod membutuhkan jenis akses ini, tetapi jika mereka melakukannya, Anda harus menyadari risikonya. Secara default pod yang berjalan sebagai root akan memiliki akses tulis ke sistem file yang diekspos oleh HostPath. Ini dapat memungkinkan penyerang untuk memodifikasi pengaturan kubelet, membuat tautan simbolis ke direktori atau file yang tidak langsung diekspos oleh HostPath, misalnya /etc/shadow, menginstal kunci ssh, membaca rahasia yang dipasang ke host, dan hal-hal berbahaya lainnya. Untuk mengurangi risiko dari HostPath, konfigurasikan `spec.containers.volumeMounts` as`readOnly`, misalnya:

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

Anda juga harus menggunakan policy-as-code solusi untuk membatasi direktori yang dapat digunakan oleh `hostPath` volume, atau mencegah `hostPath` penggunaan sama sekali. Anda dapat menggunakan Pod Security Standards Baseline atau kebijakan Restricted untuk mencegah penggunaan. `hostPath`

Untuk informasi lebih lanjut tentang bahaya eskalasi istimewa, baca blog Seth Art Bad Pods: Kubernetes Pod Privilege Escalation.

Tetapkan permintaan dan batasan untuk setiap kontainer untuk menghindari pertentangan sumber daya dan serangan DoS

Pod tanpa permintaan atau batasan secara teoritis dapat menggunakan semua sumber daya yang tersedia di host. Karena pod tambahan dijadwalkan ke sebuah node, node mungkin mengalami CPU atau tekanan memori yang dapat menyebabkan Kubelet menghentikan atau mengusir pod dari node. Meskipun Anda tidak dapat mencegah hal ini terjadi bersama-sama, menetapkan permintaan dan batasan akan membantu meminimalkan pertentangan sumber daya dan mengurangi risiko dari aplikasi yang ditulis dengan buruk yang mengkonsumsi sumber daya dalam jumlah berlebihan.

`podSpec`Ini memungkinkan Anda untuk menentukan permintaan dan batas untuk CPU dan memori. CPU dianggap sebagai sumber daya yang dapat dikompresi karena dapat kelebihan langganan. Memori tidak dapat dimampatkan, yaitu tidak dapat dibagi di antara beberapa wadah.

Saat Anda menentukan permintaan untuk CPU atau memori, pada dasarnya Anda menentukan jumlah memori yang dijamin akan didapat oleh kontainer. Kubernetes mengumpulkan permintaan dari semua kontainer dalam sebuah pod untuk menentukan node mana yang akan menjadwalkan pod tersebut. Jika sebuah wadah melebihi jumlah memori yang diminta, mungkin akan dihentikan jika ada tekanan memori pada node.

Batas adalah jumlah maksimum CPU dan sumber daya memori yang diizinkan untuk dikonsumsi oleh wadah dan secara langsung sesuai dengan `memory.limit_in_bytes` nilai cgroup yang dibuat untuk wadah. Sebuah wadah yang melebihi batas memori akan OOM dimatikan. Jika sebuah wadah melebihi batas CPU-nya, itu akan dibatasi.

> **Note**
>
> Saat menggunakan kontainer, sangat `resources.limits` disarankan agar penggunaan sumber daya kontainer (alias Resource Footprints) digerakkan oleh data dan akurat, berdasarkan pengujian beban. Tanpa jejak sumber daya yang akurat dan tepercaya, wadah `resources.limits` dapat empuk. Misalnya, `resources.limits.memory` dapat diberi empuk 20-30% lebih tinggi dari maksimum yang dapat diamati, untuk memperhitungkan potensi ketidakakuratan batas sumber daya memori.

Kubernetes menggunakan tiga kelas Quality of Service (QoS) untuk memprioritaskan beban kerja yang berjalan pada sebuah node. Ini termasuk:

- terjamin
- bisa meledak
- upaya terbaik

Jika batas dan permintaan tidak disetel, pod dikonfigurasi sebagai upaya terbaik (prioritas terendah). Pod dengan upaya terbaik adalah yang pertama terbunuh ketika memori tidak mencukupi. Jika limit ditetapkan pada semua kontainer di dalam pod, atau jika permintaan dan limit disetel ke nilai yang sama dan tidak sama dengan 0, pod dikonfigurasi sebagai jaminan (prioritas tertinggi). Pod yang dijamin tidak akan dimatikan kecuali melebihi batas memori yang dikonfigurasi. Jika batas dan permintaan dikonfigurasi dengan nilai yang berbeda dan tidak sama dengan 0, atau satu kontainer dalam pod menetapkan batas dan yang lainnya tidak atau memiliki batas yang ditetapkan untuk sumber daya yang berbeda, pod dikonfigurasi sebagai burstable (prioritas menengah). Pod ini

memiliki beberapa jaminan sumber daya, tetapi dapat dimatikan setelah melebihi memori yang diminta.

> **Important**
>
> Permintaan tidak memengaruhi `memory_limit_in_bytes` nilai cgroup container; batas cgroup diatur ke jumlah memori yang tersedia di host. Namun demikian, pengaturan nilai permintaan terlalu rendah dapat menyebabkan pod ditargetkan untuk dihentikan oleh kubelet jika node mengalami tekanan memori.

Kelas	Prioritas	Kondisi	Kondisi Bunuh
Terjamin	tertinggi	batas = permintaan! = 0	Hanya melebihi batas memori
Terbakar	medium	batasi! = permintaan! = 0	Dapat dimatikan jika melebihi memori permintaan
Upaya Terbaik	terendah	batas & permintaan Tidak Ditetapkan	Pertama terbunuh ketika memori tidak mencukupi

Untuk informasi tambahan tentang resource QoS, silakan lihat dokumentasi Kubernetes.

Anda dapat memaksa penggunaan permintaan dan batasan dengan menetapkan kuota sumber daya pada namespace atau dengan membuat rentang batas. Kuota sumber daya memungkinkan Anda menentukan jumlah total sumber daya, misalnya CPU dan RAM, yang dialokasikan ke namespace. Saat diterapkan ke namespace, ini memaksa Anda untuk menentukan permintaan dan batasan untuk semua kontainer yang digunakan ke namespace tersebut. Sebaliknya, rentang batas memberi Anda kontrol yang lebih terperinci dari alokasi sumber daya. Dengan rentang batas, Anda dapat min/max menggunakan CPU dan sumber daya memori per pod atau per kontainer dalam namespace. Anda juga dapat menggunakannya untuk mengatur nilai permintaan/batas default jika tidak ada yang disediakan.

Policy-as-code solusi dapat digunakan menerapkan permintaan dan batasan. atau bahkan untuk membuat kuota sumber daya dan rentang batas saat ruang nama dibuat.

Jangan biarkan eskalasi istimewa

Eskalasi istimewa memungkinkan proses untuk mengubah konteks keamanan di mana ia berjalan. Sudo adalah contoh yang baik dari ini seperti binari dengan bit SUID atau SGID. Eskalasi istimewa pada dasarnya adalah cara bagi pengguna untuk mengeksekusi file dengan izin pengguna atau grup lain. Anda dapat mencegah penampung menggunakan eskalasi istimewa dengan menerapkan kebijakan policy-as-code mutasi yang disetel `allowPrivilegeEscalation` ke `false` atau dengan `securityContext.allowPrivilegeEscalation` menyetel di. `podSpec` Policy-as-code kebijakan juga dapat digunakan untuk mencegah permintaan server API berhasil jika pengaturan yang salah terdeteksi. Standar Keamanan Pod juga dapat digunakan untuk mencegah Pod menggunakan eskalasi hak istimewa.

Nonaktifkan ServiceAccount token mount

Untuk pod yang tidak perlu mengakses Kubernetes API, Anda dapat menonaktifkan pemasangan ServiceAccount token secara otomatis pada spesifikasi pod, atau untuk semua pod yang menggunakan tertentu. 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
```

Nonaktifkan penemuan layanan

Untuk pod yang tidak perlu mencari atau memanggil layanan in-cluster, Anda dapat mengurangi jumlah informasi yang diberikan ke pod. Anda dapat mengatur kebijakan DNS Pod agar tidak menggunakan CoreDNS, dan tidak mengekspos layanan di namespace pod sebagai variabel

lingkungan. Lihat dokumen Kubernetes tentang variabel lingkungan untuk informasi selengkapnya tentang tautan layanan. Nilai default untuk kebijakan DNS pod adalah "ClusterFirst" yang menggunakan DNS in-cluster, sedangkan nilai non-default “Default” menggunakan resolusi DNS node yang mendasarinya. Lihat dokumen Kubernetes tentang kebijakan DNS Pod untuk informasi selengkapnya.

Example

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

Konfigurasikan gambar Anda dengan sistem file root read-only

Mengkonfigurasi gambar Anda dengan sistem file root read-only mencegah penyerang menimpa biner pada sistem file yang digunakan aplikasi Anda. Jika aplikasi Anda harus menulis ke sistem file, pertimbangkan untuk menulis ke direktori sementara atau melampirkan dan memasang volume. Anda dapat menerapkan ini dengan menyetel pod SecurityContext sebagai berikut:

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

Policy-as-code dan Standar Keamanan Pod dapat digunakan untuk menegakkan perilaku ini.

Example

Sesuai kontainer Windows di Kubernetes `securityContext.readOnlyRootFilesystem` tidak dapat diatur `true` untuk wadah yang berjalan di Windows karena akses tulis diperlukan untuk registri dan proses sistem untuk berjalan di dalam wadah.

Alat dan sumber daya

- Lokakarya Perendaman Keamanan Amazon EKS - Keamanan Pod

- open-policy-agent/gatekeeper-library: Perpustakaan kebijakan OPA Gatekeeper adalah pustaka kebijakan yang dapat Anda gunakan sebagai pengganti. OPA/Gatekeeper PSPs
- Perpustakaan Kebijakan Kyverno
- Kumpulan kebijakan OPA dan Kyverno umum untuk EKS.
- Penanggulangan berbasis kebijakan: bagian 1
- Penanggulangan berbasis kebijakan: bagian 2
- Pod Security Policy Migrator adalah alat yang mengkonversi PSPs ke kebijakan OPA/GateKeeper,, atau Kyverno KubeWarden
- NeuVector oleh SUSE open source, platform keamanan wadah zero-trust, menyediakan kebijakan proses dan sistem file serta aturan kontrol penerimaan.

Isolasi Penyewa

Ketika kita memikirkan multi-tenancy, kita sering ingin mengisolasi pengguna atau aplikasi dari pengguna atau aplikasi lain yang berjalan pada infrastruktur bersama.

Kubernetes adalah orkestrator penyewa tunggal, yaitu satu instance dari pesawat kontrol dibagi di antara semua penyewa dalam sebuah cluster. Namun, ada berbagai objek Kubernetes yang dapat Anda gunakan untuk membuat kemiripan multi-tenancy. Misalnya, Namespaces dan Role-based access control (RBAC) dapat diimplementasikan untuk secara logis mengisolasi penyewa satu sama lain. Demikian pula, Quotas dan Limit Ranges dapat digunakan untuk mengontrol jumlah sumber daya cluster yang dapat dikonsumsi setiap penyewa. Namun demikian, cluster adalah satu-satunya konstruksi yang menyediakan batas keamanan yang kuat. Ini karena penyerang yang berhasil mendapatkan akses ke host di dalam cluster dapat mengambil semua Rahasia,, dan VolumeConfigMaps, yang dipasang pada host itu. Mereka juga bisa meniru Kubelet yang memungkinkan mereka memanipulasi atribut node yang and/or bergerak secara lateral di dalam cluster.

Bagian berikut akan menjelaskan bagaimana menerapkan isolasi penyewa sambil mengurangi risiko menggunakan orkestrator penyewa tunggal seperti Kubernetes.

Multi-tenancy lembut

Dengan soft multi-tenancy, Anda menggunakan konstruksi Kubernetes asli, misalnya namespace, role dan role binding, dan kebijakan jaringan, untuk membuat pemisahan logis antar penyewa. RBAC, misalnya, dapat mencegah penyewa mengakses atau memanipulasi sumber daya masing-masing.

Kuota dan rentang batas mengontrol jumlah sumber daya cluster yang dapat dikonsumsi setiap penyewa sementara kebijakan jaringan dapat membantu mencegah aplikasi yang digunakan ke ruang nama yang berbeda berkomunikasi satu sama lain.

Namun, tak satu pun dari kontrol ini mencegah pod dari penyewa yang berbeda berbagi node. Jika diperlukan isolasi yang lebih kuat, Anda dapat menggunakan pemilih node, aturan anti-afinitas, and/or taint, dan toleransi untuk memaksa pod dari penyewa yang berbeda dijadwalkan ke node terpisah; sering disebut sebagai node penyewa tunggal. Ini bisa menjadi agak rumit, dan biayanya mahal, di lingkungan dengan banyak penyewa.

> **Important**
>
> Soft multi-tenancy yang diimplementasikan dengan Namespaces tidak memungkinkan Anda untuk menyediakan penyewa dengan daftar Namespaces yang difilter karena Namespace adalah Tipe cakupan global. Jika penyewa memiliki kemampuan untuk melihat Namespace tertentu, ia dapat melihat semua Namespace dalam cluster.

> **Warning**
>
> Dengan soft-multi-tenancy, penyewa mempertahankan kemampuan untuk menanyakan CoreDNS untuk semua layanan yang berjalan di dalam cluster secara default.
> Seorang penyerang dapat mengeksploitasi ini dengan menjalankan dig SRV `..svc.cluster.local` dari pod mana pun di cluster. Jika Anda perlu membatasi akses ke catatan DNS layanan yang berjalan dalam cluster Anda, pertimbangkan untuk menggunakan plugin Firewall atau Kebijakan untuk CoreDNS. Untuk informasi tambahan, lihat https://github.com/coredns/ kebijakan # -policykubernetes-metadata-multi-tenancy.

Kiosk adalah proyek open source yang dapat membantu dalam implementasi soft multi-tenancy. Ini diimplementasikan sebagai serangkaian CRDs dan pengontrol yang menyediakan kemampuan berikut:

- Akun & Pengguna Akun untuk memisahkan penyewa di klaster Kubernetes bersama
- Penyediaan Namespace Layanan Mandiri untuk pengguna akun
- Batas Akun untuk memastikan kualitas layanan dan keadilan saat berbagi klaster
- Template Namespace untuk isolasi penyewa yang aman dan inisialisasi namespace swalayan

Loft adalah penawaran komersial dari pengelola Kios dan DevSpaceyang menambahkan kemampuan berikut:

- Akses multi-cluster untuk memberikan akses ke ruang di cluster yang berbeda
- Mode tidur menurunkan penyebaran di ruang selama periode tidak aktif
- Sistem masuk tunggal dengan penyedia otentikasi OIDC seperti GitHub

Ada tiga kasus penggunaan utama yang dapat diatasi dengan soft multi-tenancy.

Pengaturan Perusahaan

Yang pertama adalah dalam pengaturan Perusahaan di mana “penyewa” semi-dipercaya karena mereka adalah karyawan, kontraktor, atau diberi wewenang oleh organisasi. Setiap penyewa biasanya akan menyelaraskan ke divisi administratif seperti departemen atau tim.

Dalam jenis pengaturan ini, administrator klaster biasanya akan bertanggung jawab untuk membuat ruang nama dan mengelola kebijakan. Mereka juga dapat menerapkan model administrasi yang didelegasikan di mana individu tertentu diberi pengawasan atas namespace, memungkinkan mereka untuk melakukan operasi CRUD untuk objek terkait non-kebijakan seperti penerapan, layanan, pod, pekerjaan, dll.

Isolasi yang disediakan oleh runtime kontainer mungkin dapat diterima dalam pengaturan ini atau mungkin perlu ditambah dengan kontrol tambahan untuk keamanan pod. Mungkin juga perlu untuk membatasi komunikasi antar layanan di ruang nama yang berbeda jika diperlukan isolasi yang lebih ketat.

Kubernetes sebagai Layanan

Sebaliknya, soft multi-tenancy dapat digunakan dalam pengaturan di mana Anda ingin menawarkan Kubernetes as a service (KaaS). Dengan KaaS, aplikasi Anda di-host di cluster bersama bersama dengan koleksi pengontrol dan CRDs yang menyediakan satu set layanan PaaS. Penyewa berinteraksi langsung dengan server API Kubernetes dan diizinkan untuk melakukan operasi CRUD pada objek non-kebijakan. Ada juga elemen layanan mandiri di mana penyewa dapat diizinkan untuk membuat dan mengelola ruang nama mereka sendiri. Dalam jenis lingkungan ini, penyewa diasumsikan menjalankan kode yang tidak tepercaya.

Untuk mengisolasi penyewa dalam jenis lingkungan ini, Anda mungkin perlu menerapkan kebijakan jaringan yang ketat serta pod sandboxing. Sandboxing adalah tempat Anda menjalankan kontainer

pod di dalam VM mikro seperti Firecracker atau di kernel ruang pengguna. Hari ini, Anda dapat membuat polong kotak pasir dengan EKS Fargate.

Perangkat Lunak sebagai Layanan (SaaS)

Kasus penggunaan terakhir untuk soft multi-tenancy adalah dalam pengaturan (Software-as-a-ServiceSaaS). Dalam lingkungan ini, setiap penyewa dikaitkan dengan instance tertentu dari aplikasi yang berjalan di dalam cluster. Setiap instance sering memiliki datanya sendiri dan menggunakan kontrol akses terpisah yang biasanya independen dari Kubernetes RBAC.

Berbeda dengan kasus penggunaan lainnya, penyewa dalam pengaturan SaaS tidak secara langsung berinteraksi dengan Kubernetes API. Sebaliknya, aplikasi SaaS bertanggung jawab untuk berinteraksi dengan Kubernetes API untuk membuat objek yang diperlukan untuk mendukung setiap penyewa.

Konstruksi Kubernetes

Dalam setiap contoh ini konstruksi berikut digunakan untuk mengisolasi penyewa dari satu sama lain:

Namespace

Ruang nama sangat penting untuk menerapkan soft multi-tenancy. Mereka memungkinkan Anda untuk membagi cluster menjadi partisi logis. Kuota, kebijakan jaringan, akun layanan, dan objek lain yang diperlukan untuk mengimplementasikan multi-tenancy dicakup ke namespace.

Kebijakan jaringan

Secara default, semua pod dalam klaster Kubernetes diizinkan untuk berkomunikasi satu sama lain. Perilaku ini dapat diubah menggunakan kebijakan jaringan.

Kebijakan jaringan membatasi komunikasi antar pod menggunakan label atau rentang alamat IP. Dalam lingkungan multi-tenant di mana isolasi jaringan yang ketat antara penyewa diperlukan, sebaiknya mulai dengan aturan default yang menolak komunikasi antar pod, dan aturan lain yang memungkinkan semua Pod melakukan query pada server DNS untuk resolusi nama. Dengan itu, Anda dapat mulai menambahkan aturan yang lebih permisif yang memungkinkan komunikasi dalam namespace. Ini dapat disempurnakan lebih lanjut sesuai kebutuhan.

> **Note**
>
> Amazon VPC CNI sekarang mendukung Kebijakan Jaringan Kubernetes untuk membuat kebijakan yang dapat mengisolasi beban kerja sensitif dan melindunginya dari akses tidak

sah saat menjalankan Kubernetes di AWS. Ini berarti Anda dapat menggunakan semua kemampuan API Kebijakan Jaringan dalam kluster Amazon EKS Anda. Tingkat kontrol granular ini memungkinkan Anda untuk menerapkan prinsip hak istimewa terkecil, yang memastikan bahwa hanya Pod yang diotorisasi yang diizinkan untuk berkomunikasi satu sama lain.

Important

Kebijakan jaringan diperlukan tetapi tidak cukup. Penegakan kebijakan jaringan membutuhkan mesin kebijakan seperti Calico atau Cilium.

Kontrol akses berbasis peran (RBAC)

Roles dan role binding adalah objek Kubernetes yang digunakan untuk menegakkan role-based access control (RBAC) di Kubernetes. Peran berisi daftar tindakan yang dapat dilakukan terhadap objek di klaster Anda. Ikatan peran menentukan individu atau kelompok kepada siapa peran tersebut berlaku. Dalam pengaturan perusahaan dan KaaS, RBAC dapat digunakan untuk mengizinkan administrasi objek oleh kelompok atau individu yang dipilih.

Kuota

Kuota digunakan untuk menentukan batasan beban kerja yang dihosting di klaster Anda. Dengan kuota, Anda dapat menentukan jumlah maksimum CPU dan memori yang dapat dikonsumsi pod, atau Anda dapat membatasi jumlah sumber daya yang dapat dialokasikan dalam klaster atau namespace. Rentang batas memungkinkan Anda untuk mendeklarasikan nilai minimum, maksimum, dan default untuk setiap batas.

Sumber daya yang berlebihan dalam cluster bersama seringkali bermanfaat karena memungkinkan Anda memaksimalkan sumber daya Anda. Namun, akses tak terbatas ke cluster dapat menyebabkan kelaparan sumber daya, yang dapat menyebabkan penurunan kinerja dan hilangnya ketersediaan aplikasi. Jika permintaan pod disetel terlalu rendah dan pemanfaatan sumber daya aktual melebihi kapasitas node, node akan mulai mengalami tekanan CPU atau memori. Ketika ini terjadi, pod dapat dimulai ulang and/or diusir dari node.

Untuk mencegah hal ini terjadi, Anda harus merencanakan untuk menerapkan kuota pada namespace di lingkungan multi-tenant untuk memaksa penyewa menentukan permintaan dan

batasan saat menjadwalkan pod mereka di klaster. Ini juga akan mengurangi potensi penolakan layanan dengan membatasi jumlah sumber daya yang dapat dikonsumsi pod.

Anda juga dapat menggunakan kuota untuk membagi sumber daya klaster agar selaras dengan pengeluaran penyewa. Ini sangat berguna dalam skenario KaaS.

Prioritas dan preemption pod

Prioritas dan preemption Pod dapat berguna ketika Anda ingin lebih mementingkan Pod dibandingkan dengan Pod lainnya. Misalnya, dengan prioritas pod, Anda dapat mengonfigurasi pod dari pelanggan A agar berjalan pada prioritas yang lebih tinggi daripada pelanggan B. Ketika kapasitas yang tersedia tidak mencukupi, penjadwal akan mengusir pod dengan prioritas lebih rendah dari pelanggan B untuk mengakomodasi pod dengan prioritas lebih tinggi dari pelanggan A. Ini bisa sangat berguna di lingkungan SaaS di mana pelanggan yang bersedia membayar premi menerima prioritas yang lebih tinggi.

> **Important**
>
> Prioritas Pod dapat memiliki efek yang tidak diinginkan pada Pod lain dengan prioritas yang lebih rendah. Misalnya, meskipun pod korban dihentikan dengan baik tetapi tidak PodDisruptionBudget dijamin, yang dapat merusak aplikasi dengan prioritas lebih rendah yang bergantung pada kuorum Pod, lihat Batasan preemption.

Mengurangi kontrol

Perhatian utama Anda sebagai administrator lingkungan multi-penyewa adalah mencegah penyerang mendapatkan akses ke host yang mendasarinya. Kontrol berikut harus dipertimbangkan untuk mengurangi risiko ini:

Lingkungan eksekusi kotak pasir untuk kontainer

Sandboxing adalah teknik dimana setiap kontainer dijalankan di mesin virtual terisolasi sendiri. Teknologi yang melakukan pod sandboxing termasuk Firecracker dan Weave's Firekube.

Untuk informasi tambahan tentang upaya menjadikan Firecracker sebagai runtime yang didukung untuk EKS, lihat 1238496944684597248.html. https://threadreaderapp.com/thread/

Agen Kebijakan Terbuka (OPA) & Penjaga Gerbang

Gatekeeper adalah pengontrol penerimaan Kubernetes yang memberlakukan kebijakan yang dibuat dengan OPA. Dengan OPA, Anda dapat membuat kebijakan yang menjalankan Pod dari penyewa pada instance terpisah atau dengan prioritas yang lebih tinggi daripada penyewa lainnya. Kumpulan kebijakan OPA umum dapat ditemukan di GitHub repositori untuk proyek ini.

Ada juga plugin OPA eksperimental untuk CoreDNS yang memungkinkan Anda menggunakan OPA filter/control ke catatan yang dikembalikan oleh CoreDNS.

Kyverno

Kyverno adalah mesin kebijakan asli Kubernetes yang dapat memvalidasi, mengubah, dan menghasilkan konfigurasi dengan kebijakan sebagai sumber daya Kubernetes. Kyverno menggunakan overlay gaya Kustomize untuk validasi, mendukung JSON Patch dan patch gabungan strategis untuk mutasi, dan dapat mengkloning sumber daya di seluruh ruang nama berdasarkan pemicu fleksibel.

Anda dapat menggunakan Kyverno untuk mengisolasi ruang nama, menerapkan keamanan pod dan praktik terbaik lainnya, dan menghasilkan konfigurasi default seperti kebijakan jaringan. Beberapa contoh disertakan dalam GitHub repositori untuk proyek ini. Banyak lainnya termasuk dalam perpustakaan kebijakan di situs web Kyverno.

Mengisolasi beban kerja penyewa ke node tertentu

Membatasi beban kerja penyewa untuk berjalan pada node tertentu dapat digunakan untuk meningkatkan isolasi dalam model multi-tenancy lunak. Dengan pendekatan ini, beban kerja khusus penyewa hanya dijalankan pada node yang disediakan untuk penyewa masing-masing. Untuk mencapai isolasi ini, properti Kubernetes asli (afinitas node, dan taints dan toleransi) digunakan untuk menargetkan node tertentu untuk penjadwalan pod, dan mencegah Pod, dari penyewa lain, agar tidak dijadwalkan pada node khusus penyewa.

Bagian 1 - Afinitas simpul

Afinitas node Kubernetes digunakan untuk menargetkan node untuk penjadwalan, berdasarkan label node. Dengan aturan afinitas node, pod tertarik ke node tertentu yang cocok dengan istilah pemilih. Dalam spesifikasi pod di bawah ini, afinitas `requiredDuringSchedulingIgnoredDuringExecution` node diterapkan ke masing-masing pod. Hasilnya adalah pod akan menargetkan node yang diberi label dengan kunci/nilai berikut:. `node-restriction.kubernetes.io/tenant: tenants-x`

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

Dengan afinitas node ini, label diperlukan selama penjadwalan, tetapi tidak selama eksekusi; jika label node yang mendasarinya berubah, pod tidak akan diusir hanya karena perubahan label tersebut. Namun, penjadwalan masa depan dapat terpengaruh.

> ⚠ Warning
>
> Awalan label `node-restriction.kubernetes.io/` memiliki arti khusus di Kubernetes. NodeRestrictionyang diaktifkan untuk kluster EKS `kubelet` mencegah adding/removing/updating label dengan awalan ini. Penyerang tidak dapat menggunakan apa yang `kubelet's credentials to update the node object or modify the system setup to pass these labels into `kubelet` kubelet` tidak diizinkan untuk memodifikasi label ini. Jika awalan ini digunakan untuk semua penjadwalan pod ke node, ini mencegah skenario di mana penyerang mungkin ingin menarik serangkaian beban kerja yang berbeda ke node dengan memodifikasi label node.

Example

Alih-alih afinitas simpul, kita bisa menggunakan pemilih simpul. Namun, afinitas node lebih ekspresif dan memungkinkan lebih banyak kondisi dipertimbangkan selama penjadwalan pod. Untuk informasi tambahan tentang perbedaan dan pilihan penjadwalan yang lebih lanjut, silakan lihat posting blog CNCF ini di Advanced Kubernetes pod to node scheduling.

Bagian 2 - Noda dan toleransi

Menarik pod ke node hanyalah bagian pertama dari pendekatan tiga bagian ini. Agar pendekatan ini berhasil, kita harus mengusir pod dari penjadwalan ke node yang pod tidak diotorisasi. Untuk

mengusir pod yang tidak diinginkan atau tidak sah, Kubernetes menggunakan node taints. Taint digunakan untuk menempatkan kondisi pada node yang mencegah pod dijadwalkan. Taint di bawah ini menggunakan pasangan kunci-nilai dari. `tenant: tenants-x`

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

Mengingat node di atas`taint`, hanya pod yang mentolerir taint yang akan diizinkan untuk dijadwalkan pada node. Untuk memungkinkan pod yang diotorisasi dijadwalkan ke node, spesifikasi pod masing-masing harus menyertakan a `toleration` to the taint, seperti yang terlihat di bawah ini.

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

Pod dengan hal di atas tidak `toleration` akan dihentikan dari penjadwalan pada node, setidaknya bukan karena taint khusus itu. Taints juga digunakan oleh Kubernetes untuk menghentikan sementara penjadwalan pod selama kondisi tertentu, seperti tekanan sumber daya node. Dengan afinitas node, dan taint dan toleransi, kita dapat secara efektif menarik pod yang diinginkan ke node tertentu dan mengusir pod yang tidak diinginkan.

> **Important**
>
> Pod Kubernetes tertentu diperlukan untuk berjalan di semua node. Contoh pod ini adalah yang dimulai oleh Container Network Interface (CNI) dan daemonset kube-proxy. Untuk itu, spesifikasi untuk pod ini mengandung toleransi yang sangat permisif, untuk mentolerir noda yang berbeda. Perhatian harus diambil untuk tidak mengubah toleransi ini. Mengubah toleransi ini dapat mengakibatkan operasi cluster yang salah. Selain itu, alat manajemen kebijakan, seperti OPA/GateKeeper dan Kyverno dapat digunakan untuk menulis kebijakan validasi yang mencegah pod yang tidak sah menggunakan toleransi permisif ini.

Bagian 3 - Manajemen berbasis kebijakan untuk pemilihan simpul

Ada beberapa alat yang dapat digunakan untuk membantu mengelola afinitas node dan toleransi spesifikasi pod, termasuk penegakan aturan di pipeline CICD. Namun, penegakan isolasi juga harus dilakukan di tingkat cluster Kubernetes. Untuk tujuan ini, alat manajemen kebijakan dapat digunakan untuk mengubah permintaan server API Kubernetes masuk, berdasarkan payload permintaan, untuk menerapkan aturan afinitas node masing-masing dan toleransi yang disebutkan di atas.

Misalnya, pod yang ditujukan untuk namespace tenants-x dapat dicap dengan afinitas dan toleransi node yang benar untuk mengizinkan penjadwalan pada node tenants-x. Menggunakan alat manajemen kebijakan yang dikonfigurasi menggunakan Kubernetes Mutating Admission Webhook, kebijakan dapat digunakan untuk mengubah spesifikasi pod masuk. Mutasi menambahkan elemen yang diperlukan untuk memungkinkan penjadwalan yang diinginkan. Contoh OPA/Gatekeeper kebijakan yang menambahkan afinitas simpul terlihat di bawah ini.

```
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"
```

Kebijakan di atas diterapkan pada permintaan server API Kubernetes, untuk menerapkan pod ke namespace tenants-x. Kebijakan ini menambahkan aturan afinitas `requiredDuringSchedulingIgnoredDuringExecution` node, sehingga pod tertarik ke node dengan `tenant: tenants-x` label.

Kebijakan kedua, terlihat di bawah, menambahkan toleransi ke spesifikasi pod yang sama, menggunakan kriteria pencocokan yang sama dari namespace target dan grup, jenis, dan versi.

```
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"
```

Kebijakan di atas khusus untuk pod; ini disebabkan oleh jalur ke elemen yang bermutasi dalam elemen kebijakan`location`. Kebijakan tambahan dapat ditulis untuk menangani sumber daya yang membuat pod, seperti sumber daya Deployment dan Job. Kebijakan yang tercantum dan contoh lainnya dapat dilihat dalam GitHubproyek pendamping untuk panduan ini.

Hasil dari dua mutasi ini adalah bahwa pod tertarik ke node yang diinginkan, sementara pada saat yang sama, tidak ditolak oleh noda node tertentu. Untuk memverifikasi ini, kita dapat melihat cuplikan output dari dua `kubectl` panggilan untuk mendapatkan node yang diberi label`tenant=tenants-x`, dan mendapatkan pod di 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...
```

Seperti yang dapat kita lihat dari output di atas, semua pod dijadwalkan pada node yang diberi label. `tenant=tenants-x` Sederhananya, pod hanya akan berjalan pada node yang diinginkan, dan pod lainnya (tanpa afinitas dan toleransi yang diperlukan) tidak akan. Beban kerja penyewa secara efektif diisolasi.

Contoh spesifikasi pod bermutasi terlihat di bawah ini.

```
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**
>
> Alat manajemen kebijakan yang terintegrasi dengan alur permintaan server API Kubernetes, menggunakan webhook masuk yang bermutasi dan memvalidasi, dirancang untuk menanggapi permintaan server API dalam jangka waktu tertentu. Ini biasanya 3 detik atau kurang. Jika panggilan webhook gagal mengembalikan respons dalam waktu yang dikonfigurasi, and/or validasi mutasi dari permintaan pemutusan API masuk mungkin atau mungkin tidak terjadi. Perilaku ini didasarkan pada apakah konfigurasi webhook penerimaan disetel ke Fail Buka atau Gagal Tutup.

Dalam contoh di atas, kami menggunakan kebijakan yang ditulis untuk OPA/GateKeeper. Namun, ada alat manajemen kebijakan lain yang menangani kasus penggunaan pemilihan simpul kami juga. Misalnya, kebijakan Kyverno ini dapat digunakan untuk menangani mutasi afinitas simpul.

> **Note**
>
> Jika beroperasi dengan benar, kebijakan mutasi akan memengaruhi perubahan yang diinginkan pada muatan permintaan server API masuk. Namun, kebijakan validasi juga harus disertakan untuk memverifikasi bahwa perubahan yang diinginkan terjadi, sebelum perubahan diizinkan untuk bertahan. Ini sangat penting ketika menggunakan kebijakan ini untuk tenant-to-node isolasi. Ini juga merupakan ide yang baik untuk menyertakan kebijakan Audit untuk secara rutin memeriksa klaster Anda untuk konfigurasi yang tidak diinginkan.

Referensi

- k-rail Dirancang untuk membantu Anda mengamankan lingkungan multi-penyewa melalui penegakan kebijakan tertentu.
- Praktik Keamanan untuk Aplikasi MultiTenant SaaS menggunakan Amazon EKS

Multi-tenancy yang keras

Hard multi-tenancy dapat diimplementasikan dengan menyediakan cluster terpisah untuk setiap penyewa. Meskipun ini memberikan isolasi yang sangat kuat antara penyewa, ia memiliki beberapa kelemahan.

Pertama, ketika Anda memiliki banyak penyewa, pendekatan ini dapat dengan cepat menjadi mahal. Anda tidak hanya harus membayar biaya pesawat kontrol untuk setiap cluster, Anda tidak akan dapat berbagi sumber daya komputasi antar cluster. Ini pada akhirnya akan menyebabkan fragmentasi di mana sebagian dari cluster Anda kurang dimanfaatkan sementara yang lain digunakan secara berlebihan.

Kedua, Anda mungkin perlu membeli atau membangun perkakas khusus untuk mengelola semua cluster ini. Pada waktunya, mengelola ratusan atau ribuan cluster mungkin menjadi terlalu berat.

Akhirnya, membuat cluster per penyewa akan lambat relatif terhadap pembuatan namespace. Namun demikian, pendekatan penyewaan keras mungkin diperlukan di industri yang sangat diatur atau di lingkungan SaaS di mana isolasi yang kuat diperlukan.

Arah masa depan

Komunitas Kubernetes telah menyadari kekurangan soft multi-tenancy saat ini dan tantangan dengan hard multi-tenancy. Multi-Tenancy Special Interest Group (SIG) berusaha mengatasi kekurangan ini melalui beberapa proyek inkubasi, termasuk Hierarchical Namespace Controller (HNC) dan Virtual Cluster.

Proposal HNC (KEP) menjelaskan cara untuk membuat hubungan orangtua-anak antara namespace dengan warisan objek [policy] bersama dengan kemampuan administrator penyewa untuk membuat sub-namespace.

Proposal Virtual Cluster menjelaskan mekanisme untuk membuat instance terpisah dari layanan bidang kontrol, termasuk server API, manajer pengontrol, dan penjadwal, untuk setiap penyewa di dalam klaster (juga dikenal sebagai “Kubernetes on Kubernetes”).

Proposal Multi-Tenancy Benchmarks menyediakan panduan untuk berbagi cluster menggunakan namespace untuk isolasi dan segmentasi, dan alat baris perintah kubectl-mtb untuk memvalidasi kesesuaian dengan pedoman.

Alat dan sumber daya manajemen multi-cluster

- Awan Banzai
- Kommander
- Lensa
- Nirmata
- Rafay
- Peternak
- Menenun Fluks

Audit dan pencatatan

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Mengumpulkan dan menganalisis log [audit] berguna untuk berbagai alasan yang berbeda. Log dapat membantu dengan analisis akar penyebab dan atribusi, yaitu menganggap perubahan pada pengguna tertentu. Ketika cukup banyak log telah dikumpulkan, mereka dapat digunakan untuk mendeteksi perilaku anomali juga. Di EKS, log audit dikirim ke Amazon Cloudwatch Logs. Kebijakan audit untuk EKS adalah sebagai berikut:

```
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"
```

Rekomendasi

Aktifkan log audit

Log audit adalah bagian dari log pesawat kontrol Kubernetes yang dikelola EKS yang dikelola oleh EKS. Petunjuk untuk enabling/disabling log bidang kontrol, yang mencakup log untuk server API Kubernetes, manajer pengontrol, dan penjadwal, bersama dengan log audit, dapat ditemukan di sini, -plane-logs.html# -export. https://docs.aws.amazon.com/eks/ latest/userguide/control enabling-control-plane-log

> **Note**
>
> Saat Anda mengaktifkan pencatatan pesawat kontrol, Anda akan dikenakan biaya untuk menyimpan log masuk CloudWatch. Hal ini menimbulkan masalah yang lebih luas tentang biaya keamanan yang sedang berlangsung. Pada akhirnya Anda harus mempertimbangkan biaya-biaya tersebut terhadap biaya pelanggaran keamanan, misalnya kerugian finansial, kerusakan reputasi Anda, dll. Anda mungkin menemukan bahwa Anda dapat mengamankan lingkungan Anda secara memadai dengan menerapkan hanya beberapa rekomendasi dalam panduan ini.

> **Warning**
>
> Ukuran maksimum untuk entri CloudWatch Log adalah 1MB sedangkan ukuran permintaan API Kubernetes maksimum adalah 1.5MiB. Entri log yang lebih besar dari 1MB akan dipotong atau hanya menyertakan metadata permintaan.

Memanfaatkan metadata audit

Log audit Kubernetes mencakup dua anotasi yang menunjukkan apakah suatu permintaan diotorisasi atau tidak `authorization.k8s.io/decision` dan alasan keputusan tersebut. `authorization.k8s.io/reason` Gunakan atribut ini untuk memastikan mengapa panggilan API tertentu diizinkan.

Buat alarm untuk acara mencurigakan

Buat alarm untuk secara otomatis memberi tahu Anda di mana ada peningkatan 403 Forbidden dan 401 Respons Tidak Sah, lalu gunakan atribut sepertihost,`sourceIPs`, dan `k8s_user.username` untuk mencari tahu dari mana permintaan tersebut berasal.

Menganalisis log dengan Wawasan Log

Gunakan Wawasan CloudWatch Log untuk memantau perubahan pada objek RBAC, misalnya Peran, RoleBindings, ClusterRoles dan. ClusterRoleBindings Beberapa contoh pertanyaan muncul di bawah ini:

Daftar pembaruan ke `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
```

Daftar pembuatan baru atau perubahan pada webhook validasi:

```
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
```

Daftar membuat, memperbarui, menghapus operasi ke Peran:

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

Daftar membuat, memperbarui, menghapus operasi ke RoleBindings:

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

Daftar membuat, memperbarui, menghapus operasi ke ClusterRoles:

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

Daftar membuat, memperbarui, menghapus operasi ke ClusterRoleBindings:

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

Plot operasi baca yang tidak sah terhadap Rahasia:

```
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)
```

Daftar permintaan anonim yang gagal:

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

Audit CloudTrail log Anda

AWS yang APIs dipanggil oleh pod yang menggunakan IAM Roles for Service Accounts (IRSA) secara otomatis dicatat CloudTrail bersama dengan nama akun layanan. Jika nama akun layanan yang tidak secara eksplisit diizinkan untuk memanggil API muncul di log, itu mungkin merupakan indikasi bahwa kebijakan kepercayaan peran IAM salah dikonfigurasi. Secara umum, Cloudtrail adalah cara yang bagus untuk menganggap panggilan AWS API ke prinsip IAM tertentu.

Gunakan CloudTrail Wawasan untuk menggali aktivitas yang mencurigakan

CloudTrail wawasan secara otomatis menganalisis peristiwa manajemen penulisan dari CloudTrail jejak dan memberi tahu Anda tentang aktivitas yang tidak biasa. Ini dapat membantu Anda mengidentifikasi kapan ada peningkatan volume panggilan saat menulis APIs di akun AWS Anda, termasuk dari pod yang menggunakan IRSA untuk mengambil peran IAM. Lihat Mengumumkan CloudTrail Wawasan: Mengidentifikasi dan Menanggapi Aktivitas API yang Tidak Biasa untuk informasi lebih lanjut.

Sumber daya tambahan

Ketika volume log meningkat, mengurai dan memfilternya dengan Wawasan Log atau alat analisis log lainnya mungkin menjadi tidak efektif. Sebagai alternatif, Anda mungkin ingin mempertimbangkan untuk menjalankan Sysdig Falco dan ekscloudwatch. Falco menganalisis log audit dan menandai anomali atau penyalahgunaan selama periode waktu yang lama. Proyek ekscloudwatch meneruskan peristiwa log audit dari Falco CloudWatch untuk analisis. Falco menyediakan seperangkat aturan audit default bersama dengan kemampuan untuk menambahkan aturan Anda sendiri.

Namun pilihan lain mungkin untuk menyimpan log audit di S3 dan menggunakan algoritma SageMaker Random Cut Forest untuk perilaku anomali yang memerlukan penyelidikan lebih lanjut.

Alat dan sumber daya

Proyek komersial dan open source berikut dapat digunakan untuk menilai keselarasan klaster Anda dengan praktik terbaik yang telah ditetapkan:

- Lokakarya Perendaman Keamanan Amazon EKS - Kontrol Detektif
- kubeaudit
- kube-scan Menetapkan skor risiko pada beban kerja yang berjalan di klaster Anda sesuai dengan framework Kubernetes Common Configuration Scoring System
- kubesec.io
- polaris
- Kanan
- Snyk
- Kubescape Kubescape adalah alat keamanan kubernetes open source yang memindai cluster, file YAMB, dan bagan Helm. Ini mendeteksi kesalahan konfigurasi sesuai dengan beberapa kerangka kerja (termasuk NSA-CISA dan MITRE ATT&CK®.)

Keamanan jaringan

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Keamanan jaringan memiliki beberapa aspek. Yang pertama melibatkan penerapan aturan yang membatasi arus lalu lintas jaringan antar layanan. Yang kedua melibatkan enkripsi lalu lintas saat sedang dalam perjalanan. Mekanisme untuk menerapkan langkah-langkah keamanan ini pada EKS bervariasi tetapi sering kali mencakup item-item berikut:

Kontrol lalu lintas

- Kebijakan Jaringan
- Grup Keamanan

Enkripsi jaringan

- Layanan Mesh
- Antarmuka Jaringan Kontainer () CNIs
- Pengontrol Ingress dan Load Balancer

- Contoh Nitro
- ACM Private CA dengan manajer sertifikat

Kebijakan jaringan

Dalam klaster Kubernetes, semua komunikasi Pod ke Pod diperbolehkan secara default. Meskipun fleksibilitas ini dapat membantu mempromosikan eksperimen, itu tidak dianggap aman. Kebijakan jaringan Kubernetes memberi Anda mekanisme untuk membatasi lalu lintas jaringan antar Pod (sering disebut sebagai lalu lintas Timur/Barat) serta antara Pod dan layanan eksternal. Kebijakan jaringan Kubernetes beroperasi pada lapisan 3 dan 4 dari model OSI. Kebijakan jaringan menggunakan pod, pemilih namespace, dan label untuk mengidentifikasi pod sumber dan tujuan, tetapi juga dapat menyertakan alamat IP, nomor port, protokol, atau kombinasi keduanya. Kebijakan Jaringan dapat diterapkan pada koneksi Inbound atau Outbound ke pod, sering disebut aturan Ingress dan Egress.

Dengan dukungan kebijakan jaringan native dari Amazon VPC CNI Plugin, Anda dapat menerapkan kebijakan jaringan untuk mengamankan lalu lintas jaringan di klaster kubernetes. Ini terintegrasi dengan API Kebijakan Jaringan Kubernetes hulu, memastikan kompatibilitas dan kepatuhan terhadap standar Kubernetes. Anda dapat menentukan kebijakan menggunakan pengidentifikasi berbeda yang didukung oleh API upstream. Secara default, semua lalu lintas masuk dan keluar diizinkan ke pod. Ketika kebijakan jaringan dengan PolicyType Ingress ditentukan, hanya koneksi yang diizinkan ke dalam pod adalah yang berasal dari node pod dan yang diizinkan oleh aturan ingress. Hal yang sama berlaku untuk aturan jalan keluar. Jika beberapa aturan didefinisikan, maka penyatuan semua aturan diperhitungkan saat membuat keputusan. Dengan demikian, urutan evaluasi tidak mempengaruhi hasil kebijakan.

> **Important**
>
> Saat pertama kali menyediakan kluster EKS, fungsionalitas Kebijakan Jaringan VPC CNI tidak diaktifkan secara default. Pastikan Anda menerapkan versi Add-on VPC CNI yang didukung dan `ENABLE_NETWORK_POLICY` setel flag `true` ke add-on vpc-cni untuk mengaktifkannya. Lihat Panduan Pengguna Amazon EKS untuk petunjuk terperinci.

Rekomendasi

Memulai dengan Kebijakan Jaringan - Ikuti Prinsip Hak Istimewa Terkecil

Buat kebijakan penolakan default

Seperti halnya kebijakan RBAC, disarankan untuk mengikuti prinsip akses yang paling tidak memiliki hak istimewa dengan kebijakan jaringan. Mulailah dengan membuat kebijakan tolak semua yang membatasi semua lalu lintas masuk dan keluar dengan namespace.

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

default-menyangkal

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

> **Note**
>
> Gambar di atas dibuat oleh penampil kebijakan jaringan dari Tufin.

Buat aturan untuk mengizinkan kueri DNS

Setelah Anda memiliki aturan penolakan semua default, Anda dapat mulai membuat layering pada aturan tambahan, seperti aturan yang memungkinkan pod untuk menanyakan CoreDNS untuk resolusi nama.

```
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	

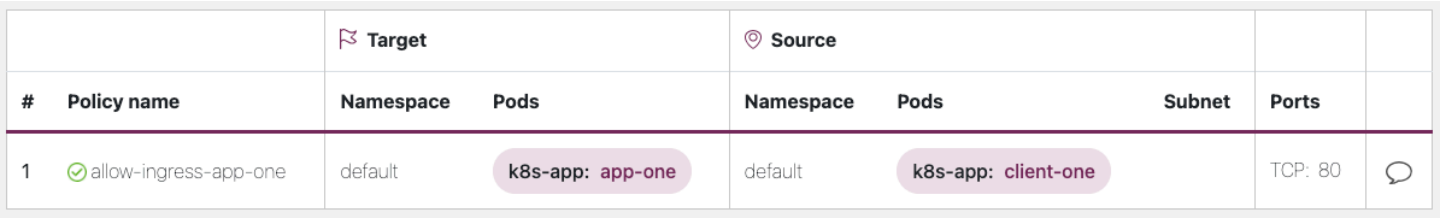
Tambahkan aturan secara bertahap untuk memungkinkan aliran lalu lintas antar namespace/pod secara selektif

Memahami persyaratan aplikasi dan membuat aturan masuk dan keluar berbutir halus sesuai kebutuhan. Contoh di bawah ini menunjukkan cara membatasi lalu lintas masuk pada port 80 ke `app-one` dari. `client-one` Ini membantu meminimalkan permukaan serangan dan mengurangi risiko akses yang tidak sah.

```
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	

Pemantauan penegakan kebijakan jaringan

- Gunakan editor Kebijakan Jaringan
  - Editor kebijakan jaringan membantu visualisasi, skor keamanan, pembuatan otomatis dari log aliran jaringan
  - Membangun kebijakan jaringan dengan cara yang interaktif
- Log Audit
  - Tinjau log audit cluster EKS Anda secara teratur
  - Log audit menyediakan banyak informasi tentang tindakan apa yang telah dilakukan pada klaster Anda termasuk perubahan kebijakan jaringan
  - Gunakan informasi ini untuk melacak perubahan pada kebijakan jaringan Anda dari waktu ke waktu dan mendeteksi perubahan yang tidak sah atau tidak terduga
- Pengujian otomatis
  - Terapkan pengujian otomatis dengan membuat lingkungan pengujian yang mencerminkan lingkungan produksi Anda dan menerapkan beban kerja secara berkala yang mencoba melanggar kebijakan jaringan Anda.
- Metrik pemantauan
  - Konfigurasikan agen observabilitas Anda untuk mengikis metrik prometheus dari agen node VPC CNI, yang memungkinkan untuk memantau kesehatan agen, dan kesalahan sdk.

- Kebijakan Jaringan Audit secara teratur
  - Audit Kebijakan Jaringan Anda secara berkala untuk memastikan bahwa mereka memenuhi persyaratan aplikasi Anda saat ini. Saat aplikasi Anda berkembang, audit memberi Anda kesempatan untuk menghapus aturan masuk, keluar yang berlebihan dan memastikan bahwa aplikasi Anda tidak memiliki izin yang berlebihan.
- Pastikan Kebijakan Jaringan ada menggunakan Open Policy Agent (OPA)
  - Gunakan Kebijakan OPA seperti yang ditunjukkan di bawah ini untuk memastikan Kebijakan Jaringan selalu ada sebelum melakukan orientasi pada pod aplikasi. Kebijakan ini menolak orientasi pod k8s dengan label `k8s-app: sample-app` jika kebijakan jaringan terkait tidak ada.

```
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
}
```

Pemecahan masalah

Pantau log vpc-network-policy-controller agen simpul

Aktifkan log manajer pengontrol bidang Kontrol EKS untuk mendiagnosis fungsionalitas kebijakan jaringan. Anda dapat mengalirkan log bidang kontrol ke grup CloudWatch log dan menggunakan wawasan CloudWatch Log untuk melakukan kueri lanjutan. Dari log, Anda dapat melihat objek titik akhir pod yang diselesaikan menjadi Kebijakan Jaringan, status rekonsiliasi kebijakan, dan debug jika kebijakan berfungsi seperti yang diharapkan.

Selain itu, Amazon VPC CNI memungkinkan Anda mengaktifkan pengumpulan dan ekspor log penegakan kebijakan ke Amazon Cloudwatch dari node pekerja EKS. Setelah diaktifkan, Anda dapat

memanfaatkan CloudWatchWawasan Kontainer untuk memberikan wawasan tentang penggunaan yang terkait dengan Kebijakan Jaringan.

Amazon VPC CNI juga mengirimkan SDK yang menyediakan antarmuka untuk berinteraksi dengan program eBPF pada node. SDK diinstal saat `aws-node` dikerahkan ke node. Anda dapat menemukan biner SDK yang diinstal di bawah `/opt/cni/bin` direktori pada node. Saat diluncurkan, SDK menyediakan dukungan untuk fungsionalitas mendasar seperti memeriksa program dan peta eBPF.

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

Log metadata lalu lintas jaringan

AWS VPC Flow Logs menangkap metadata tentang lalu lintas yang mengalir melalui VPC, seperti alamat IP sumber dan tujuan serta port bersama dengan paket yang diterima/dijatuhkan. Informasi ini dapat dianalisis untuk mencari aktivitas yang mencurigakan atau tidak biasa antara sumber daya dalam VPC, termasuk Pod. Namun, karena alamat IP pod sering berubah saat diganti, Flow Logs mungkin tidak cukup dengan sendirinya. Calico Enterprise memperluas Flow Logs dengan label pod dan metadata lainnya, sehingga lebih mudah untuk menguraikan arus lalu lintas antar pod.

Grup keamanan

EKS menggunakan AWS VPC Security Groups (SGs) untuk mengontrol lalu lintas antara bidang kontrol Kubernetes dan node pekerja klaster. Grup keamanan juga digunakan untuk mengontrol lalu lintas antara node pekerja, dan sumber daya VPC lainnya, dan alamat IP eksternal. Saat Anda menyediakan kluster EKS (dengan Kubernetes versi 1.14-eks.3 atau lebih tinggi), grup keamanan klaster akan dibuat secara otomatis untuk Anda. Grup keamanan ini memungkinkan komunikasi yang tidak terkekang antara bidang kontrol EKS dan node dari grup node terkelola. Untuk mempermudah, Anda disarankan untuk menambahkan cluster SG ke semua grup node, termasuk grup node yang tidak dikelola.

Sebelum Kubernetes versi 1.14 dan EKS versi eks.3, ada grup keamanan terpisah yang dikonfigurasi untuk bidang kontrol EKS dan grup node. Aturan minimum dan yang disarankan untuk control plane dan grup keamanan grup node dapat ditemukan di https://docs.aws.amazon.com/eks/latest/userguide/sec -group-reqs.html. Aturan minimum untuk grup keamanan bidang kontrol memungkinkan port 443 masuk dari node pekerja SG. Aturan inilah yang memungkinkan kubelet berkomunikasi dengan server API Kubernetes. Ini juga mencakup port 10250 untuk lalu lintas keluar ke node pekerja SG; 10250 adalah port yang didengarkan kubelet. Demikian pula, aturan grup node minimum memungkinkan port 10250 masuk dari bidang kontrol SG dan 443 keluar ke bidang kontrol

SG. Akhirnya ada aturan yang memungkinkan komunikasi tak terkekang antara node dalam grup node.

Jika Anda perlu mengontrol komunikasi antara layanan yang berjalan di dalam klaster dan layanan yang dijalankan di luar klaster seperti database RDS, pertimbangkan grup keamanan untuk Pod. Dengan grup keamanan untuk Pod, Anda dapat menetapkan grup keamanan yang ada ke kumpulan pod.

> **Warning**
>
> Jika Anda mereferensikan grup keamanan yang tidak ada sebelum pembuatan pod, pod tidak akan dijadwalkan.

Anda dapat mengontrol Pod mana yang ditetapkan ke grup keamanan dengan membuat `SecurityGroupPolicy` objek dan menentukan sebuah `PodSelector` atau sebuah`ServiceAccountSelector`. Menyetel selector ke `{}` akan menetapkan SGs referensi `SecurityGroupPolicy` ke semua pod di namespace atau semua Account Layanan di namespace. Pastikan Anda telah membiasakan diri dengan semua pertimbangan sebelum menerapkan grup keamanan untuk pod.

> **Important**
>
> Jika Anda menggunakan SGs Pod, Anda harus membuat SGs yang memungkinkan port 53 keluar ke grup keamanan klaster. Demikian pula, Anda harus memperbarui grup keamanan klaster untuk menerima lalu lintas masuk port 53 dari grup keamanan pod.

> **Important**
>
> Batasan untuk grup keamanan masih berlaku saat menggunakan grup keamanan untuk pod, jadi gunakan dengan bijaksana.

> **Important**
>
> Anda harus membuat aturan untuk lalu lintas masuk dari grup keamanan klaster (kubelet) untuk semua probe yang dikonfigurasi untuk pod.

> **Important**
>
> Grup keamanan untuk pod bergantung pada fitur yang dikenal sebagai trunking ENI yang dibuat untuk meningkatkan kepadatan ENI dari instans EC2. Ketika sebuah pod ditugaskan ke SG, pengontrol VPC mengaitkan ENI cabang dari grup node dengan pod. Jika tidak ada cukup cabang yang ENIs tersedia di grup node pada saat pod dijadwalkan, pod akan tetap dalam status tertunda. Jumlah cabang yang dapat didukung oleh ENIs instance bervariasi menurut instance type/family. See https://docs.aws.amazon.com/eks/latest/userguide/security - groups-for-pods .html# supported-instance-types untuk detail lebih lanjut.

Meskipun grup keamanan untuk pod menawarkan cara asli AWS untuk mengontrol lalu lintas jaringan di dalam dan di luar klaster Anda tanpa overhead daemon kebijakan, opsi lain tersedia. Misalnya, mesin kebijakan Cilium memungkinkan Anda untuk mereferensikan nama DNS dalam kebijakan jaringan. Calico Enterprise menyertakan opsi untuk memetakan kebijakan jaringan ke grup keamanan AWS. Jika Anda telah menerapkan mesh layanan seperti Istio, Anda dapat menggunakan gateway keluar untuk membatasi jalan keluar jaringan ke domain atau alamat IP tertentu yang sepenuhnya memenuhi syarat. Untuk informasi lebih lanjut tentang opsi ini, baca seri tiga bagian tentang kontrol lalu lintas jalan keluar di Istio.

Kapan menggunakan Kebijakan Jaringan vs Grup Keamanan untuk Pod?

Kapan menggunakan kebijakan jaringan Kubernetes

- Mengontrol pod-to-pod lalu lintas
  - Cococok untuk mengontrol lalu lintas jaringan antar pod di dalam cluster (lalu lintas timur-barat)
- Kontrol lalu lintas di alamat IP atau tingkat port (lapisan OSI 3 atau 4)

Kapan menggunakan grup AWS Security untuk pod (SGP)

- Manfaatkan konfigurasi AWS yang ada
  - Jika Anda sudah memiliki kumpulan grup keamanan EC2 kompleks yang mengelola akses ke layanan AWS dan Anda memigrasikan aplikasi dari instans EC2 ke EKS, SGPs dapat menjadi pilihan yang sangat baik yang memungkinkan Anda untuk menggunakan kembali sumber daya grup keamanan dan menerapkannya ke pod Anda.
- Kontrol akses ke layanan AWS

  - Aplikasi Anda yang berjalan dalam kluster EKS ingin berkomunikasi dengan layanan AWS lainnya (database RDS), digunakan SGPs sebagai mekanisme yang efisien untuk mengontrol lalu lintas dari pod ke layanan AWS.
- Isolasi lalu lintas Pod & Node
  - Jika Anda ingin benar-benar memisahkan lalu lintas pod dari lalu lintas node lainnya, gunakan SGP dalam `POD_SECURITY_GROUP_ENFORCING_MODE=strict` mode.

Praktik terbaik menggunakan grup Keamanan untuk Pod dan Kebijakan Jaringan

- Keamanan berlapis
  - Gunakan kombinasi kebijakan jaringan SGP dan kubernetes untuk pendekatan keamanan berlapis
  - Gunakan SGPs untuk membatasi akses tingkat jaringan ke layanan AWS yang bukan bagian dari klaster, sementara kebijakan jaringan kubernetes dapat membatasi lalu lintas jaringan antar pod di dalam klaster
- Prinsip hak istimewa yang paling kecil
  - Hanya izinkan lalu lintas yang diperlukan antara pod atau ruang nama
- Segmentasikan aplikasi Anda
  - Jika memungkinkan, segmentasikan aplikasi berdasarkan kebijakan jaringan untuk mengurangi radius ledakan jika aplikasi dikompromikan
- Jaga kebijakan tetap sederhana dan jelas
  - Kebijakan jaringan Kubernetes bisa sangat terperinci dan kompleks, yang terbaik adalah membuatnya sesederhana mungkin untuk mengurangi risiko kesalahan konfigurasi dan memudahkan overhead manajemen
- Kurangi permukaan serangan
  - Minimalkan permukaan serangan dengan membatasi eksposur aplikasi Anda

> **Important**
>
> Grup Keamanan untuk Pod menyediakan dua mode penegakan: `strict` dan`standard`. Anda harus menggunakan `standard` mode saat menggunakan Kebijakan Jaringan dan Grup Keamanan untuk fitur pod di klaster EKS.

Ketika datang ke keamanan jaringan, pendekatan berlapis seringkali merupakan solusi yang paling efektif. Menggunakan kebijakan jaringan kubernetes dan SGP dalam kombinasi dapat memberikan defense-in-depth strategi yang kuat untuk aplikasi Anda yang berjalan di EKS.

Penegakan Kebijakan Service Mesh atau kebijakan jaringan Kubernetes

A `service mesh` adalah lapisan infrastruktur khusus yang dapat Anda tambahkan ke aplikasi Anda. Ini memungkinkan Anda untuk secara transparan menambahkan kemampuan seperti observabilitas, manajemen lalu lintas, dan keamanan, tanpa menambahkannya ke kode Anda sendiri.

Service mesh memberlakukan kebijakan pada Layer 7 (aplikasi) model OSI sedangkan kebijakan jaringan kubernetes beroperasi pada Layer 3 (network) dan Layer 4 (transport). Ada banyak penawaran di ruang ini seperti AWSAppMesh, Istio, Linkerd, dll.,

Kapan menggunakan Service mesh untuk penegakan kebijakan

- Memiliki investasi yang ada di service mesh
- Membutuhkan kemampuan yang lebih canggih seperti manajemen lalu lintas, observabilitas & keamanan
  - Kontrol lalu lintas, penyeimbangan beban, pemutusan sirkuit, pembatasan laju, batas waktu, dll.
  - Wawasan terperinci tentang kinerja layanan Anda (latensi, tingkat kesalahan, permintaan per detik, volume permintaan, dll.)
  - Anda ingin menerapkan dan memanfaatkan mesh layanan untuk fitur keamanan seperti mTL

Pilih kebijakan jaringan Kubernetes untuk kasus penggunaan yang lebih sederhana

- Batasi pod mana yang dapat berkomunikasi satu sama lain
- Kebijakan jaringan membutuhkan sumber daya yang lebih sedikit daripada mesh layanan sehingga cocok untuk kasus penggunaan yang lebih sederhana atau untuk cluster yang lebih kecil di mana overhead menjalankan dan mengelola mesh layanan mungkin tidak dapat dibenarkan

> **Note**
>
> Kebijakan jaringan dan Service mesh juga dapat digunakan bersama. Gunakan kebijakan jaringan untuk memberikan tingkat keamanan dan isolasi dasar antara pod Anda dan

kemudian gunakan mesh layanan untuk menambahkan kemampuan tambahan seperti manajemen lalu lintas, observabilitas, dan keamanan.

ThirdParty Mesin Kebijakan Jaringan

Pertimbangkan Mesin Kebijakan Jaringan Pihak Ketiga ketika Anda memiliki persyaratan kebijakan lanjutan seperti Kebijakan Jaringan Global, dukungan untuk aturan berbasis DNS Hostname, aturan Layer 7, aturan ServiceAccount berbasis, dan deny/log tindakan eksplisit, dll., Calico, adalah mesin kebijakan open source dari Tigera yang bekerja dengan baik dengan EKS. Selain menerapkan set lengkap fitur kebijakan jaringan Kubernetes, Calico mendukung kebijakan jaringan yang diperluas dengan serangkaian fitur yang lebih kaya, termasuk dukungan untuk aturan lapisan 7, misalnya HTTP, ketika terintegrasi dengan Istio. Kebijakan Calico dapat dicakup ke Namespace, Pod, akun layanan, atau secara global. Ketika kebijakan dicakup ke akun layanan, kebijakan tersebut mengaitkan seperangkat ingress/egress aturan dengan akun layanan tersebut. Dengan aturan RBAC yang tepat, Anda dapat mencegah tim mengesampingkan aturan ini, memungkinkan profesional keamanan TI untuk mendelegasikan administrasi ruang nama dengan aman. Isovalent, pengelola Cilium, juga telah memperluas kebijakan jaringan untuk menyertakan dukungan sebagian untuk aturan lapisan 7, misalnya HTTP. Cilium juga memiliki dukungan untuk nama host DNS yang dapat berguna untuk membatasi lalu lintas antara Kubernetes Services/Pods dan sumber daya yang berjalan di dalam atau di luar VPC Anda. Sebaliknya, Calico Enterprise menyertakan fitur yang memungkinkan Anda memetakan kebijakan jaringan Kubernetes ke grup keamanan AWS, serta nama host DNS.

Anda dapat menemukan daftar kebijakan jaringan Kubernetes yang umum di. https://github.com/ahmetb/ kubernetes-network-policy-recipes Seperangkat aturan serupa untuk Calico tersedia di https://docs.projectcalico. org/security/calico-kebijakan jaringan.

Migrasi ke Mesin Kebijakan Jaringan Amazon VPC CNI

Untuk menjaga konsistensi dan menghindari perilaku komunikasi pod yang tidak terduga, disarankan untuk menggunakan hanya satu Network Policy Engine di klaster Anda. Jika Anda ingin bermigrasi dari 3P ke VPC CNI Network Policy Engine, sebaiknya Anda mengonversi 3P NetworkPolicy CRDs yang ada menjadi resource Kubernetes sebelum mengaktifkan dukungan kebijakan jaringan VPC CNI. NetworkPolicy Dan, uji kebijakan yang dimigrasi dalam klaster pengujian terpisah sebelum menerapkannya di lingkungan produksi Anda. Ini memungkinkan Anda untuk mengidentifikasi dan mengatasi potensi masalah atau ketidakkonsistenan dalam perilaku komunikasi pod.

Alat Migrasi

Untuk membantu proses migrasi Anda, kami telah mengembangkan alat bernama K8s Network Policy Migrator yang mengubah kebijakan jaringan Anda yang ada menjadi kebijakan Calico/Cilium jaringan asli CRDs Kubernetes. Setelah konversi, Anda dapat langsung menguji kebijakan jaringan yang dikonversi pada cluster baru Anda yang menjalankan pengontrol kebijakan jaringan VPC CNI. Alat ini dirancang untuk membantu Anda merampingkan proses migrasi dan memastikan transisi yang lancar.

> **Important**
>
> Alat migrasi hanya akan mengonversi kebijakan 3P yang kompatibel dengan api kebijakan jaringan kubernetes asli. Jika Anda menggunakan fitur kebijakan jaringan lanjutan yang ditawarkan oleh plugin 3P, alat Migrasi akan melewati dan melaporkannya.

Harap dicatat bahwa alat migrasi saat ini tidak didukung oleh tim rekayasa kebijakan AWS VPC CNI Network, alat ini tersedia untuk pelanggan dengan upaya terbaik. Kami mendorong Anda untuk menggunakan alat ini untuk memfasilitasi proses migrasi Anda. Jika Anda mengalami masalah atau bug dengan alat ini, kami mohon Anda membuat GitHubmasalah. Umpan balik Anda sangat berharga bagi kami dan akan membantu dalam peningkatan berkelanjutan dari layanan kami.

Sumber Daya Tambahan

- Kubernetes & Tigera: Kebijakan Jaringan, Keamanan, dan Audit
- Perusahaan Calico
- silia
- NetworkPolicyEditor editor kebijakan interaktif dari Cilium
- Inspektor Gadget menyarankan gadget kebijakan jaringan Menyarankan kebijakan jaringan berdasarkan analisis lalu lintas jaringan

Enkripsi saat bergerak

Aplikasi yang perlu mematuhi PCI, HIPAA, atau peraturan lain mungkin perlu mengenkripsi data saat sedang dalam perjalanan. Saat ini TLS adalah pilihan de facto untuk mengenkripsi lalu lintas di kawat. TLS, seperti pendahulunya SSL, menyediakan komunikasi yang aman melalui jaringan

menggunakan protokol kriptografi. TLS menggunakan enkripsi simetris di mana kunci untuk mengenkripsi data dihasilkan berdasarkan rahasia bersama yang dinegosiasikan pada awal sesi. Berikut ini adalah beberapa cara untuk mengenkripsi data di lingkungan Kubernetes.

Contoh Nitro

Lalu lintas yang dipertukarkan antara jenis instans Nitro berikut, misalnya C5n, G4, I3en, M5dn, M5n, P3dn, R5dn, dan R5n, secara otomatis dienkripsi secara default. Ketika ada hop perantara, seperti gateway transit atau penyeimbang beban, lalu lintas tidak dienkripsi. Lihat Enkripsi dalam perjalanan untuk detail lebih lanjut tentang enkripsi dalam perjalanan serta daftar lengkap jenis instance yang mendukung enkripsi jaringan secara default.

Antarmuka Jaringan Kontainer () CNIs

WeaveNetdapat dikonfigurasi untuk secara otomatis mengenkripsi semua lalu lintas menggunakan NaCl enkripsi untuk lalu lintas lengan, dan IPsec ESP untuk lalu lintas datapath cepat.

Layanan Mesh

Enkripsi dalam perjalanan juga dapat diimplementasikan dengan mesh layanan seperti App Mesh, Linkerd v2, dan Istio. AppMesh mendukung mTL dengan sertifikat X.509 atau Envoy's Secret Discovery Service (SDS). Linkerd dan Istio keduanya memiliki dukungan untuk mTL.

aws-app-mesh-examplesGitHub Repositori menyediakan panduan untuk mengonfigurasi mTL menggunakan sertifikat X.509 dan SPIRE sebagai penyedia SDS dengan wadah Utusan Anda:

- Mengkonfigurasi mTL menggunakan sertifikat X.509
- Mengkonfigurasi TLS menggunakan SPIRE (SDS)

App Mesh juga mendukung enkripsi TLS dengan sertifikat pribadi yang dikeluarkan oleh AWS Certificate Manager (ACM) atau sertifikat yang disimpan di sistem file lokal node virtual.

aws-app-mesh-examplesGitHub Repositori menyediakan panduan untuk mengonfigurasi TLS menggunakan sertifikat yang dikeluarkan oleh ACM dan sertifikat yang dikemas dengan wadah Utusan Anda:

- Mengkonfigurasi TLS dengan Sertifikat TLS yang Disediakan File
- Mengkonfigurasi TLS dengan AWS Certificate Manager

Pengontrol Ingress dan Load Balancer

Pengontrol Ingress adalah cara bagi Anda untuk secara cerdas merutekan 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 pemrosesan yang sangat intensif CPU. Akibatnya, jika Anda memiliki fleksibilitas, coba lakukan pembongkaran SSL di Ingress atau penyeimbang beban.

Gunakan enkripsi dengan penyeimbang beban AWS Elastic

AWS Application Load Balancer (ALB) dan Network Load Balancer (NLB) keduanya memiliki dukungan untuk enkripsi transport (SSL dan TLS). `alb.ingress.kubernetes.io/certificate-arn`Anotasi untuk ALB memungkinkan Anda menentukan sertifikat mana yang akan ditambahkan ke ALB. Jika Anda menghilangkan anotasi, pengontrol akan mencoba menambahkan sertifikat ke pendengar yang memerlukannya dengan mencocokkan sertifikat AWS Certificate Manager (ACM) yang tersedia menggunakan bidang host. Dimulai dengan EKS v1.15 Anda dapat menggunakan `service.beta.kubernetes.io/aws-load-balancer-ssl-cert` anotasi dengan NLB seperti yang ditunjukkan pada contoh di bawah ini.

```
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
```

Berikut ini adalah contoh tambahan untuk SSL/TLS penghentian.

- Mengamankan Ingress EKS Dengan Kontur Dan Mari Enkripsi Jalan GitOps
- Bagaimana cara menghentikan lalu lintas HTTPS di beban kerja Amazon EKS dengan ACM?

> **Important**
>
> Beberapa Ingress, seperti pengontrol AWS LB, mengimplementasikan SSL/TLS penggunaan Anotasi alih-alih sebagai bagian dari Spesifikasi Ingress.

ACM Private CA dengan manajer sertifikat

Anda dapat mengaktifkan TLS dan mTL untuk mengamankan beban kerja aplikasi EKS Anda di ingress, di pod, dan di antara pod menggunakan ACM Private Certificate Authority (CA) dan cert-manager, add-on Kubernetes populer untuk mendistribusikan, memperbarui, dan mencabut sertifikat. ACM Private CA adalah CA yang sangat tersedia, aman, terkelola tanpa biaya dimuka dan pemeliharaan untuk mengelola CA Anda sendiri. Jika Anda menggunakan otoritas sertifikat Kubernetes default, ada kesempatan untuk meningkatkan keamanan Anda dan memenuhi persyaratan kepatuhan dengan ACM Private CA. ACM Private CA mengamankan kunci pribadi dalam modul keamanan perangkat keras FIPS 140-2 Level 3 (sangat aman), dibandingkan dengan kunci penyimpanan CA default yang dikodekan dalam memori (kurang aman). CA terpusat juga memberi Anda lebih banyak kontrol dan peningkatan auditabilitas untuk sertifikat pribadi baik di dalam maupun di luar lingkungan Kubernetes.

Mode CA Berumur Pendek untuk TLS Bersama Antar Beban Kerja

Saat menggunakan ACM Private CA untuk mTL di EKS, disarankan agar Anda menggunakan sertifikat berumur pendek dengan mode CA berumur pendek. Meskipun dimungkinkan untuk mengeluarkan sertifikat berumur pendek dalam mode CA tujuan umum, menggunakan mode CA berumur pendek menghasilkan lebih hemat biaya (~ 75% lebih murah daripada mode umum) untuk kasus penggunaan di mana sertifikat baru perlu sering dikeluarkan. Selain itu, Anda harus mencoba menyelaraskan masa berlaku sertifikat privat dengan masa pakai pod di klaster EKS Anda. Pelajari lebih lanjut tentang ACM Private CA dan manfaatnya di sini.

Petunjuk Pengaturan ACM

Mulailah dengan membuat CA Pribadi dengan mengikuti prosedur yang disediakan dalam dokumen teknologi ACM Private CA. Setelah Anda memiliki CA Pribadi, instal cert-manager menggunakan instruksi instalasi reguler. Setelah menginstal cert-manager, instal plugin Cert-Manager Private CA Kubernetes dengan mengikuti petunjuk penyiapan di. GitHub Plugin ini memungkinkan manajer sertifikat meminta sertifikat pribadi dari ACM Private CA.

Sekarang setelah Anda memiliki Private CA dan kluster EKS dengan cert-manager dan plugin yang diinstal, saatnya untuk mengatur izin dan membuat penerbit. Perbarui izin IAM dari peran node EKS untuk memungkinkan akses ke ACM Private CA. Ganti `<CA_ARN>` dengan nilai dari CA Pribadi Anda:

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

Peran Layanan untuk Akun IAM, atau IRSA juga dapat digunakan. Silakan lihat bagian Sumber Daya Tambahan di bawah ini untuk contoh lengkapnya.

Buat Penerbit di Amazon EKS dengan membuat file Definisi Sumber Daya Kustom bernama cluster-issuer.yaml dengan teks berikut di dalamnya, ganti dan informasikan dengan CA Pribadi Anda. `<CA_ARN>` `<Region>`

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

Terapkan Penerbit yang Anda buat.

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

Kluster EKS Anda dikonfigurasi untuk meminta sertifikat dari Private CA. Sekarang Anda dapat menggunakan `Certificate` sumber daya cert-manager untuk menerbitkan sertifikat dengan mengubah nilai `issuerRef` bidang menjadi Penerbit CA Pribadi yang Anda buat di atas. Untuk detail selengkapnya tentang cara menentukan dan meminta sumber daya Sertifikat, silakan periksa panduan Sumber Daya Sertifikat manajer sertifikat. Lihat contoh di sini.

ACM Private CA dengan Istio dan manajer sertifikat

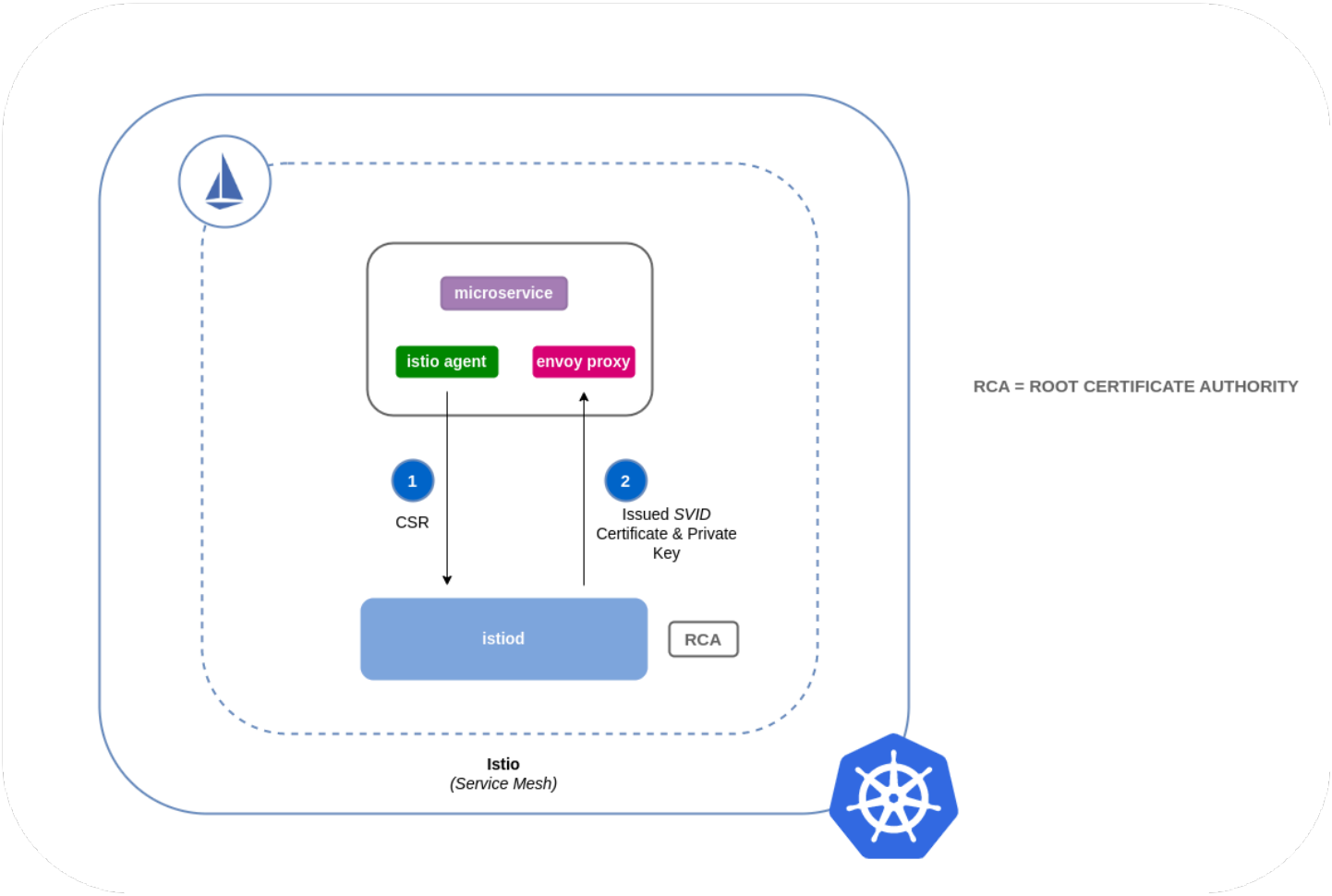
Jika Anda menjalankan Istio di cluster EKS Anda, Anda dapat menonaktifkan bidang kontrol Istio (khususnya`aistiod`) agar tidak berfungsi sebagai root Certificate Authority (CA), dan mengonfigurasi ACM Private CA sebagai root CA untuk mTL di antara beban kerja. Jika Anda menggunakan pendekatan ini, pertimbangkan untuk menggunakan mode CA berumur pendek di ACM Private CA. Lihat bagian sebelumnya dan posting blog ini untuk lebih jelasnya.

Cara Kerja Penandatanganan Sertifikat di Istio (Default)

Beban kerja di Kubernetes diidentifikasi menggunakan akun layanan. Jika Anda tidak menentukan akun layanan, Kubernetes akan secara otomatis menetapkan satu ke beban kerja Anda. Selain itu, akun layanan secara otomatis memasang token terkait. Token ini digunakan oleh akun layanan untuk beban kerja untuk mengautentikasi terhadap Kubernetes API. Akun layanan mungkin cukup sebagai identitas untuk Kubernetes tetapi Istio memiliki sistem manajemen identitas dan CA sendiri. Ketika beban kerja dimulai dengan proxy sespan utusannya, ia membutuhkan identitas yang ditetapkan dari Istio agar dapat dianggap dapat dipercaya dan diizinkan untuk berkomunikasi dengan layanan lain di mesh.

Untuk mendapatkan identitas ini dari Istio, `istio-agent` mengirimkan permintaan yang dikenal sebagai permintaan penandatanganan sertifikat (atau CSR) ke pesawat kontrol Istio. CSR ini berisi token akun layanan sehingga identitas beban kerja dapat diverifikasi sebelum diproses. Proses verifikasi ini ditangani oleh`histiod`, yang bertindak sebagai Otoritas Registrasi (atau RA) dan CA. RA berfungsi sebagai penjaga gerbang yang memastikan hanya CSR terverifikasi yang berhasil lolos ke CA. Setelah CSR diverifikasi, CSR akan diteruskan ke CA yang kemudian akan mengeluarkan sertifikat yang berisi identitas SPIFFE dengan akun layanan. Sertifikat ini disebut dokumen identitas yang dapat diverifikasi SPIFFE (atau SVID). SVID ditugaskan ke layanan yang meminta untuk tujuan identifikasi dan untuk mengenkripsi lalu lintas dalam perjalanan antara layanan yang berkomunikasi.

Alur default untuk Permintaan Penandatanganan Sertifikat Istio:



Cara Kerja Penandatanganan Sertifikat di Istio dengan ACM Private CA

Anda dapat menggunakan add-on cert-manager yang disebut agen Permintaan Penandatanganan Sertifikat Istio (istio-csr) untuk mengintegrasikan Istio dengan ACM Private CA. Agen ini memungkinkan beban kerja Istio dan komponen bidang kontrol diamankan dengan emiten manajer sertifikat, dalam hal ini ACM Private CA. Agen istio-csr mengekspos layanan yang sama dengan yang disajikan istiod dalam konfigurasi default untuk memvalidasi masuk. CSRs Kecuali, setelah verifikasi, itu akan mengubah permintaan menjadi sumber daya yang didukung manajer sertifikat (yaitu integrasi dengan penerbit CA eksternal).

Setiap kali ada CSR dari beban kerja, itu akan diteruskan ke istio-csr, yang akan meminta sertifikat dari ACM Private CA. Komunikasi antara istio-csr dan ACM Private CA ini diaktifkan oleh plugin penerbit AWS Private CA. Manajer sertifikat menggunakan plugin ini untuk meminta sertifikat TLS dari ACM Private CA. Plugin penerbit akan berkomunikasi dengan layanan ACM Private CA untuk meminta sertifikat yang ditandatangani untuk beban kerja. Setelah sertifikat ditandatangani,

sertifikat akan dikembalikan ke istio-csr, yang akan membaca permintaan yang ditandatangani, dan mengembalikannya ke beban kerja yang memulai CSR.

Alur untuk Permintaan Penandatanganan Sertifikat Istio dengan istio-csr

image:: istio-csr-with-acm -private-ca.png [Alur untuk Permintaan Penandatanganan Sertifikat Istio dengan istio-csr]

Istio dengan Petunjuk Pengaturan CA Pribadi

1. Mulailah dengan mengikuti petunjuk pengaturan yang sama di bagian ini untuk menyelesaikan yang berikut:
2. Buat CA Pribadi
3. Instal cert-manager
4. Instal plugin penerbit
5. Tetapkan izin dan buat penerbit. Penerbit mewakili CA dan digunakan untuk menandatangani `istiod` dan menyaring sertifikat beban kerja. Ini akan berkomunikasi dengan ACM Private CA.
6. Buat `istio-system` namespace. Di sinilah sumber daya Istio `istiod certificate` dan lainnya akan digunakan.
7. Instal Istio CSR yang dikonfigurasi dengan AWS Private CA Issuer Plugin. Anda dapat mempertahankan permintaan penandatanganan sertifikat untuk beban kerja guna memverifikasi bahwa permintaan tersebut disetujui dan ditandatangani (`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. Instal Istio dengan konfigurasi khusus untuk diganti `istiod` dengan `cert-manager istio-csr` sebagai penyedia sertifikat untuk mesh. Proses ini dapat dilakukan dengan menggunakan Operator Istio.

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

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

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

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

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

9. Terapkan sumber daya kustom di atas yang Anda buat.

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

10. Sekarang Anda dapat menerapkan beban kerja ke mesh di kluster EKS Anda dan menerapkan mTL.

Permintaan penandatanganan sertifikat Istio

image:: istio-csr-requests .png [Permintaan penandatanganan sertifikat Istio]

Alat dan sumber daya

- Lokakarya Perendaman Keamanan Amazon EKS - Keamanan jaringan
- Bagaimana menerapkan cert-manager dan plugin ACM Private CA untuk mengaktifkan TLS di EKS.
- Menyiapkan enkripsi end-to-end TLS di Amazon EKS dengan AWS Load Balancer Controller baru dan ACM Private CA.
- Plugin pengelola sertifikat CA Kubernetes pribadi aktif. GitHub
- Panduan pengguna plugin pengelola sertifikat CA Kubernetes pribadi.
- Cara menggunakan mode sertifikat berumur pendek AWS Private Certificate Authority

- egress-operator Operator dan plugin DNS untuk mengontrol lalu lintas keluar dari cluster Anda tanpa inspeksi protokol
- NeuVector oleh SUSE open source, platform keamanan wadah zero-trust, menyediakan aturan jaringan kebijakan, pencegahan kehilangan data (DLP), firewall aplikasi web (WAF) dan tanda tangan ancaman jaringan.

Enkripsi data dan manajemen rahasia

Enkripsi saat diam

Ada tiga opsi penyimpanan asli AWS yang berbeda yang dapat Anda gunakan dengan Kubernetes: EBS, EFS, dan untuk Lustre. FSx Ketiganya menawarkan enkripsi at rest menggunakan kunci terkelola layanan atau kunci master pelanggan (CMK). Untuk EBS Anda dapat menggunakan driver penyimpanan in-tree atau driver EBS CSI. Keduanya termasuk parameter untuk mengenkripsi volume dan memasok CMK. Untuk EFS, Anda dapat menggunakan driver EFS CSI, namun, tidak seperti EBS, driver EFS CSI tidak mendukung penyediaan dinamis. Jika Anda ingin menggunakan EFS dengan EKS, Anda perlu menyediakan dan mengonfigurasi enkripsi saat istirahat untuk sistem file sebelum membuat PV. Untuk informasi lebih lanjut tentang enkripsi file EFS, silakan merujuk ke Enkripsi Data saat Istirahat. Selain menawarkan enkripsi saat istirahat, EFS dan FSx Lustre menyertakan opsi untuk mengenkripsi data dalam perjalanan. FSx untuk Lustre melakukan ini secara default. Untuk EFS, Anda dapat menambahkan enkripsi transport dengan menambahkan `tls` parameter ke `mountOptions` dalam PV Anda seperti dalam contoh ini:

```
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>
```

Driver FSx CSI mendukung penyediaan dinamis sistem file Lustre. Ini mengenkripsi data dengan kunci yang dikelola layanan secara default, meskipun ada opsi untuk menyediakan CMK Anda sendiri seperti dalam contoh ini:

```
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**
>
> Per 28 Mei 2020 semua data yang ditulis ke volume fana di pod EKS Fargate dienkripsi secara default menggunakan algoritme kriptografi AES-256 standar industri. Tidak ada modifikasi pada aplikasi Anda yang diperlukan karena enkripsi dan dekripsi ditangani dengan mulus oleh layanan.

Enkripsi data saat istirahat

Mengenkripsi data saat istirahat dianggap sebagai praktik terbaik. Jika Anda tidak yakin apakah enkripsi diperlukan, enkripsi data Anda.

Putar Anda CMKs secara berkala

Konfigurasikan KMS untuk memutar secara otomatis. CMKs Ini akan memutar kunci Anda setahun sekali sambil menyimpan kunci lama tanpa batas waktu sehingga data Anda masih dapat didekripsi. Untuk informasi tambahan lihat Memutar kunci master pelanggan

Gunakan titik akses EFS untuk menyederhanakan akses ke kumpulan data bersama

Jika Anda telah berbagi kumpulan data dengan izin file POSIX yang berbeda atau ingin membatasi akses ke bagian dari sistem file bersama dengan membuat titik pemasangan yang berbeda,

pertimbangkan untuk menggunakan titik akses EFS. Untuk mempelajari lebih lanjut tentang bekerja dengan titik akses, lihat https://docs.aws.amazon.com/efs/ latest/ug/efs -access-points.html. Hari ini, jika Anda ingin menggunakan titik akses (AP), Anda harus mereferensikan AP dalam `volumeHandle` parameter PV.

> **Important**
>
> Per 23 Maret 2021 driver EFS CSI mendukung penyediaan dinamis EFS Access Points. Access point adalah titik masuk khusus aplikasi ke dalam sistem file EFS yang membuatnya lebih mudah untuk berbagi sistem file antara beberapa pod. Setiap sistem file EFS dapat memiliki hingga 120 PVs. Lihat Memperkenalkan penyediaan dinamis Amazon EFS CSI untuk informasi tambahan.

Manajemen rahasia

Rahasia Kubernetes digunakan untuk menyimpan informasi sensitif, seperti sertifikat pengguna, kata sandi, atau kunci API. Mereka bertahan dalam etcd sebagai string yang dikodekan base64. Pada EKS, volume EBS untuk node etcd dienkripsi dengan enkripsi EBS. Sebuah pod dapat mengambil objek rahasia Kubernetes dengan mereferensikan rahasia di. `podSpec` Rahasia ini dapat dipetakan ke variabel lingkungan atau dipasang sebagai volume. Untuk informasi tambahan tentang membuat rahasia, lihat https://kubernetes. io/docs/concepts/configuration/secret/.

> **Warning**
>
> Rahasia dalam namespace tertentu dapat direferensikan oleh semua pod di namespace rahasia.

> **Warning**
>
> Node authorizer memungkinkan Kubelet untuk membaca semua rahasia yang dipasang ke node.

Gunakan AWS KMS untuk enkripsi amplop rahasia Kubernetes

Ini memungkinkan Anda untuk mengenkripsi rahasia Anda dengan kunci enkripsi data unik (DEK). DEK kemudian dienkripsi menggunakan kunci enkripsi kunci (KEK) dari AWS KMS yang dapat diputar secara otomatis pada jadwal berulang. Dengan plugin KMS untuk Kubernetes, semua rahasia Kubernetes disimpan dalam etcd dalam ciphertext alih-alih teks biasa dan hanya dapat didekripsi oleh server API Kubernetes. Untuk detail tambahan, lihat menggunakan dukungan penyedia enkripsi EKS untuk pertahanan secara mendalam

Audit penggunaan Rahasia Kubernetes

Pada EKS, aktifkan pencatatan audit dan buat filter CloudWatch metrik dan alarm untuk mengingatkan Anda saat rahasia digunakan (opsional). Berikut ini adalah contoh filter metrik untuk log audit Kubernetes,. `{($.verb="get") && ($.objectRef.resource="secret")}` Anda juga dapat menggunakan kueri berikut dengan Wawasan CloudWatch Log:

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

Query di atas akan menampilkan berapa kali rahasia telah diakses dalam jangka waktu tertentu.

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

Kueri ini akan menampilkan rahasia, bersama dengan namespace dan nama pengguna pengguna yang mencoba mengakses rahasia dan kode respons.

Putar rahasia Anda secara berkala

Kubernetes tidak secara otomatis memutar rahasia. Jika Anda harus memutar rahasia, pertimbangkan untuk menggunakan toko rahasia eksternal, misalnya Vault atau AWS Secrets Manager.

Gunakan ruang nama terpisah sebagai cara untuk mengisolasi rahasia dari aplikasi yang berbeda

Jika Anda memiliki rahasia yang tidak dapat dibagi antara aplikasi dalam namespace, buat namespace terpisah untuk aplikasi tersebut.

Gunakan volume mount alih-alih variabel lingkungan

Nilai variabel lingkungan dapat secara tidak sengaja muncul di log. Rahasia yang dipasang sebagai volume dipakai sebagai volume tmpfs (sistem file yang didukung RAM) yang secara otomatis dihapus dari node saat pod dihapus.

Gunakan penyedia rahasia eksternal

Ada beberapa alternatif yang layak untuk menggunakan rahasia Kubernetes, termasuk AWS Secrets Manager dan Hashicorp's Vault. Layanan ini menawarkan fitur-fitur seperti kontrol akses berbutir halus, enkripsi yang kuat, dan rotasi otomatis rahasia yang tidak tersedia dengan Rahasia Kubernetes. Rahasia Tertutup Bitnami adalah pendekatan lain yang menggunakan enkripsi asimetris untuk membuat “rahasia tertutup”. Kunci publik digunakan untuk mengenkripsi rahasia sementara kunci pribadi yang digunakan untuk mendekripsi rahasia disimpan di dalam cluster, memungkinkan Anda untuk menyimpan rahasia yang disegel dengan aman dalam sistem kontrol sumber seperti Git. Lihat Mengelola penyebaran rahasia di Kubernetes menggunakan Rahasia Tertutup untuk informasi lebih lanjut.

Karena penggunaan toko rahasia eksternal telah berkembang, begitu juga kebutuhan untuk mengintegrasikannya dengan Kubernetes. Secret Store CSI Driver adalah proyek komunitas yang menggunakan model driver CSI untuk mengambil rahasia dari toko rahasia eksternal. Saat ini, Driver memiliki dukungan untuk AWS Secrets Manager, Azure, Vault, dan GCP. Penyedia AWS mendukung AWS Secrets Manager dan AWS Parameter Store. Ini juga dapat dikonfigurasi untuk memutar rahasia ketika mereka kedaluwarsa dan dapat menyinkronkan rahasia AWS Secrets Manager ke Rahasia Kubernetes. Sinkronisasi rahasia dapat berguna ketika Anda perlu merujuk rahasia sebagai variabel lingkungan alih-alih membacanya dari volume.

> **Note**
>
> Ketika driver CSI toko rahasia harus mengambil rahasia, ia mengasumsikan peran IRSA yang ditugaskan ke pod yang mereferensikan rahasia. Kode untuk operasi ini dapat ditemukan di sini.

Untuk informasi tambahan tentang AWS Secrets & Configuration Provider (ASCP) lihat sumber daya berikut:

- Cara menggunakan AWS Secrets Configuration Provider dengan Kubernetes Secret Store CSI Driver
- Mengintegrasikan rahasia Secrets Manager dengan Kubernetes Secrets Store CSI Driver

rahasia eksternal adalah cara lain untuk menggunakan toko rahasia eksternal dengan Kubernetes. Seperti Driver CSI, rahasia eksternal bekerja melawan berbagai backend yang berbeda, termasuk AWS Secrets Manager. Perbedaannya adalah, daripada mengambil rahasia dari toko rahasia eksternal, rahasia eksternal menyalin rahasia dari backend ini ke Kubernetes sebagai Rahasia. Ini memungkinkan Anda mengelola rahasia menggunakan toko rahasia pilihan Anda dan berinteraksi dengan rahasia dengan cara asli Kubernetes.

Alat dan sumber daya

- Lokakarya Perendaman Keamanan Amazon EKS - Enkripsi Data dan Manajemen Rahasia

Keamanan runtime

Keamanan runtime memberikan perlindungan aktif untuk kontainer Anda saat sedang berjalan. Idenya adalah untuk mendeteksi and/or mencegah aktivitas jahat terjadi di dalam wadah. Hal ini dapat dicapai dengan sejumlah mekanisme dalam kernel Linux atau ekstensi kernel yang terintegrasi dengan Kubernetes, seperti kemampuan Linux, komputasi aman (seccomp), atau. AppArmor SELinux Ada juga opsi seperti Amazon GuardDuty dan alat pihak ketiga yang dapat membantu menetapkan garis dasar dan mendeteksi aktivitas anomali dengan konfigurasi mekanisme kernel Linux yang lebih sedikit secara manual.

> **Important**
>
> Kubernetes saat ini tidak menyediakan mekanisme asli untuk memuat seccomp, AppArmor, atau profil ke Nodes. SELinux Mereka harus dimuat secara manual atau diinstal ke Node ketika mereka di-bootstrap. Ini harus dilakukan sebelum mereferensikannya di Pod Anda karena scheduler tidak mengetahui node mana yang memiliki profil. Lihat di bawah ini bagaimana alat seperti Operator Profil Keamanan dapat membantu mengotomatiskan penyediaan profil ke node.

Konteks keamanan dan kontrol Kubernetes bawaan

Banyak mekanisme keamanan runtime Linux terintegrasi erat dengan Kubernetes dan dapat dikonfigurasi melalui konteks keamanan Kubernetes. Salah satu opsi tersebut adalah `privileged` bendera, yang secara `false` default dan jika diaktifkan pada dasarnya setara dengan root pada host. Hampir selalu tidak tepat untuk mengaktifkan mode istimewa dalam beban kerja produksi, tetapi ada lebih banyak kontrol yang dapat memberikan hak istimewa yang lebih terperinci ke wadah yang sesuai.

Kemampuan Linux

Kemampuan Linux memungkinkan Anda untuk memberikan kemampuan tertentu ke Pod atau wadah tanpa menyediakan semua kemampuan pengguna root. Contohnya termasuk`CAP_NET_ADMIN`, yang memungkinkan konfigurasi antarmuka jaringan atau firewall, atau`CAP_SYS_TIME`, yang memungkinkan manipulasi jam sistem.

Seccomp

Dengan komputasi aman (seccomp) Anda dapat mencegah aplikasi kontainer membuat syscall tertentu ke kernel sistem operasi host yang mendasarinya. Sementara sistem operasi Linux memiliki beberapa ratus panggilan sistem, bagian terbesar dari mereka tidak diperlukan untuk menjalankan kontainer. Dengan membatasi syscall apa yang dapat dibuat oleh wadah, Anda dapat secara efektif mengurangi permukaan serangan aplikasi Anda.

Seccomp bekerja dengan mencegat syscalls dan hanya mengizinkan mereka yang telah diizinkan untuk melewatinya. Docker memiliki profil seccomp default yang cocok untuk sebagian besar beban kerja tujuan umum, dan runtime kontainer lainnya seperti containerd menyediakan default yang sebanding. Anda dapat mengonfigurasi container atau Pod untuk menggunakan profil seccomp default container runtime dengan menambahkan hal berikut ke `securityContext` bagian spesifikasi Pod:

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

Pada 1.22 (dalam alpha, stable pada 1.27), di atas `RuntimeDefault` dapat digunakan untuk semua Pod pada Node menggunakan flag kubelet tunggal,. `--seccomp-default` Maka profil yang ditentukan hanya `securityContext` diperlukan untuk profil lain.

Anda juga dapat membuat profil Anda sendiri untuk hal-hal yang memerlukan hak istimewa tambahan. Ini bisa sangat membosankan untuk dilakukan secara manual, tetapi ada alat seperti Inspektor Gadget (juga direkomendasikan di bagian keamanan jaringan untuk menghasilkan kebijakan jaringan) dan Operator Profil Keamanan yang mendukung penggunaan alat seperti eBPF atau log untuk merekam persyaratan hak istimewa dasar sebagai profil seccomp. Operator Profil Keamanan selanjutnya memungkinkan otomatisasi penyebaran profil yang direkam ke node untuk digunakan oleh Pod dan kontainer.

AppArmor dan SELinux

AppArmor dan SELinux dikenal sebagai kontrol akses wajib atau sistem MAC. Mereka mirip dalam konsep untuk seccomp tetapi dengan kemampuan yang berbeda APIs , memungkinkan kontrol akses untuk misalnya jalur sistem file tertentu atau port jaringan. Support untuk alat-alat ini tergantung pada distribusi Linux, dengan Debian/Ubuntu dukungan AppArmor dan dukungan RHEL/CentOS/Bottlerocket/Amazon Linux 2023. SELinux Lihat juga bagian keamanan infrastruktur untuk pembahasan lebih lanjut SELinux.

Keduanya AppArmor dan SELinux terintegrasi dengan Kubernetes, tetapi pada AppArmor profil Kubernetes 1.28 harus ditentukan melalui anotasi sementara SELinux label dapat diatur melalui bidang Opsi pada SELinux konteks keamanan secara langsung.

Seperti profil seccomp, Operator Profil Keamanan yang disebutkan di atas dapat membantu menyebarkan profil ke node di cluster. (Di masa depan, proyek ini juga bertujuan untuk menghasilkan profil untuk AppArmor dan SELinux seperti halnya untuk seccomp.)

Rekomendasi

Gunakan Amazon GuardDuty untuk memantau runtime dan mendeteksi ancaman ke lingkungan EKS Anda

Jika saat ini Anda tidak memiliki solusi untuk terus memantau runtime EKS dan menganalisis log audit EKS, dan memindai malware dan aktivitas mencurigakan lainnya, Amazon sangat merekomendasikan penggunaan Amazon GuardDuty untuk pelanggan yang menginginkan cara satu-klik yang sederhana, cepat, aman, terukur, dan hemat biaya untuk melindungi lingkungan AWS mereka. Amazon GuardDuty adalah layanan pemantauan keamanan yang menganalisis dan memproses sumber data dasar, seperti peristiwa CloudTrail manajemen AWS, log CloudTrail peristiwa AWS, log aliran VPC (dari instans Amazon EC2), log audit Kubernetes, dan log DNS. Ini juga termasuk pemantauan runtime EKS. Ini menggunakan umpan intelijen ancaman yang

terus diperbarui, seperti daftar alamat IP dan domain berbahaya, dan pembelajaran mesin untuk mengidentifikasi aktivitas yang tidak terduga, berpotensi tidak sah, dan berbahaya dalam lingkungan AWS Anda. Hal ini dapat mencakup masalah seperti peningkatan hak istimewa, penggunaan kredensyal yang terbuka, atau komunikasi dengan alamat IP berbahaya, domain, keberadaan malware di instans Amazon EC2 dan beban kerja kontainer EKS, atau penemuan aktivitas API yang mencurigakan. GuardDuty memberi tahu Anda tentang status lingkungan AWS Anda dengan menghasilkan temuan keamanan yang dapat Anda lihat di GuardDuty konsol atau melalui Amazon EventBridge. GuardDuty juga menyediakan dukungan bagi Anda untuk mengekspor temuan Anda ke bucket Amazon Simple Storage Service (S3) Amazon Simple Storage Service (S3), dan berintegrasi dengan layanan lain seperti AWS Security Hub dan Detective.

Tonton AWS Online Tech Talk ini "Deteksi ancaman yang ditingkatkan untuk Amazon EKS dengan Amazon GuardDuty - AWS Online Tech Talks" untuk melihat cara mengaktifkan fitur keamanan EKS tambahan ini step-by-step dalam hitungan menit.

Opsional: Gunakan solusi pihak ke-3 untuk pemantauan runtime

Membuat dan mengelola profil seccomp dan Apparmor bisa jadi sulit jika Anda tidak terbiasa dengan keamanan Linux. Jika Anda tidak punya waktu untuk menjadi mahir, pertimbangkan untuk menggunakan solusi komersial pihak ketiga. Banyak dari mereka telah bergerak melampaui profil statis seperti Apparmor dan seccomp dan telah mulai menggunakan pembelajaran mesin untuk memblokir atau memperingatkan aktivitas yang mencurigakan. Beberapa solusi ini dapat ditemukan di bawah di bagian alat. Opsi tambahan dapat ditemukan di AWS Marketplace for Container.

Pertimbangkan kemampuan add/dropping Linux sebelum menulis kebijakan seccomp

Kemampuan melibatkan berbagai pemeriksaan dalam fungsi kernel yang dapat dijangkau oleh syscalls. Jika pemeriksaan gagal, syscall biasanya mengembalikan kesalahan. Pemeriksaan dapat dilakukan tepat di awal syscall tertentu, atau lebih dalam di kernel di area yang mungkin dapat dijangkau melalui beberapa syscall yang berbeda (seperti menulis ke file istimewa tertentu). Seccomp, di sisi lain, adalah filter syscall yang diterapkan ke semua syscalls sebelum dijalankan. Sebuah proses dapat mengatur filter yang memungkinkan mereka untuk mencabut hak mereka untuk menjalankan syscalls tertentu, atau argumen khusus untuk syscall tertentu.

Sebelum menggunakan seccomp, pertimbangkan apakah kemampuan adding/removing Linux memberi Anda kontrol yang Anda butuhkan. Lihat Menyetel kemampuan untuk- kontainer untuk informasi lebih lanjut.

Lihat apakah Anda dapat mencapai tujuan Anda dengan menggunakan Kebijakan Keamanan Pod () PSPs

Kebijakan Keamanan Pod menawarkan banyak cara berbeda untuk meningkatkan postur keamanan Anda tanpa menimbulkan kerumitan yang tidak semestinya. Jelajahi opsi yang tersedia PSPs sebelum menjelajah ke dalam membangun profil seccomp dan Apparmor.

> ⚠ Warning
>
> Pada Kubernetes 1.25, PSPs telah dihapus dan diganti dengan Pod Security Admission controller. Alternatif pihak ketiga yang ada termasuk OPA/Gatekeeper dan Kyverno. Kumpulan batasan Gatekeeper dan template kendala untuk menerapkan kebijakan yang biasa ditemukan di PSPs dapat ditarik dari repositori pustaka Gatekeeper pada. GitHub Dan banyak pengganti PSPs dapat ditemukan di pustaka kebijakan Kyverno termasuk koleksi lengkap Standar Keamanan Pod.

Alat dan Sumber Daya

- 7 hal yang harus Anda ketahui sebelum memulai
- AppArmorLoader
- Menyiapkan node dengan profil
- Security Profiles Operator adalah perangkat tambahan Kubernetes yang bertujuan untuk memudahkan pengguna untuk menggunakan SELinux, seccomp dan di klaster Kubernetes. AppArmor Ini menyediakan kemampuan untuk menghasilkan profil dari menjalankan beban kerja dan memuat profil ke node Kubernetes untuk digunakan di Pod.
- Inspektor Gadget memungkinkan inspeksi, penelusuran, dan pembuatan profil banyak aspek perilaku runtime di Kubernetes, termasuk membantu dalam pembuatan profil seccomp.
- Aqua
- Qualys
- Stackrox
- Sysdig Aman
- Prisma
- NeuVector oleh SUSE open source, platform keamanan wadah zero-trust, menyediakan aturan profil proses dan aturan akses file.

Melindungi infrastruktur (host)

Karena penting untuk mengamankan gambar kontainer Anda, sama pentingnya untuk menjaga infrastruktur yang menjalankannya. Bagian ini mengeksplorasi berbagai cara untuk mengurangi risiko dari serangan yang diluncurkan langsung terhadap tuan rumah. Pedoman ini harus digunakan bersama dengan yang diuraikan di bagian Keamanan Runtime.

Rekomendasi

Gunakan OS yang dioptimalkan untuk menjalankan kontainer

Pertimbangkan untuk menggunakan Flatcar Linux, Project Atomic, RancherOS, dan Bottlerocket, OS tujuan khusus dari AWS yang dirancang untuk menjalankan wadah Linux. Ini termasuk permukaan serangan yang dikurangi, gambar disk yang diverifikasi saat boot, dan batasan izin yang diberlakukan menggunakan SELinux.

Sebagai alternatif, gunakan AMI yang dioptimalkan EKS untuk node pekerja Kubernetes Anda. AMI yang dioptimalkan EKS dirilis secara teratur dan berisi kumpulan minimal paket OS dan binari yang diperlukan untuk menjalankan beban kerja kontainer Anda.

Silakan lihat Amazon EKS AMI RHEL Build Specification untuk contoh skrip konfigurasi yang dapat digunakan untuk membangun Amazon EKS AMI khusus yang berjalan di Red Hat Enterprise Linux menggunakan Hashicorp Packer. Skrip ini dapat dimanfaatkan lebih lanjut untuk membangun kustom EKS yang sesuai dengan STIG. AMIs

Tetap perbarui OS node pekerja Anda

Terlepas dari apakah Anda menggunakan OS host yang dioptimalkan kontainer seperti Bottlerocket atau yang lebih besar, tetapi masih minimalis, Gambar Mesin Amazon seperti EKS yang dioptimalkan AMIs, praktik terbaik untuk menjaga gambar OS host ini tetap up to date dengan patch keamanan terbaru.

Untuk EKS yang dioptimalkan AMIs, periksa saluran CHANGELOG dan/atau catatan rilis secara teratur dan otomatiskan peluncuran gambar node pekerja yang diperbarui ke dalam klaster Anda.

Perlakukan infrastruktur Anda sebagai tidak dapat diubah dan otomatiskan penggantian node pekerja Anda

Daripada melakukan peningkatan di tempat, ganti pekerja Anda saat patch atau pembaruan baru tersedia. Ini bisa didekati dengan beberapa cara. Anda dapat menambahkan instance ke grup

penskalaan otomatis yang ada menggunakan AMI terbaru saat Anda secara berurutan mengikat dan menguras node hingga semua node dalam grup diganti dengan AMI terbaru. Atau, Anda dapat menambahkan instance ke grup node baru saat Anda secara berurutan mengikat dan menguras node dari grup node lama sampai semua node telah diganti. Grup node terkelola EKS menggunakan pendekatan pertama dan akan menampilkan pesan di konsol untuk meningkatkan pekerja Anda saat AMI baru tersedia. `eksctl`juga memiliki mekanisme untuk membuat grup node dengan AMI terbaru dan untuk mengepung dan menguras pod dengan anggun dari grup node sebelum instance dihentikan. Jika Anda memutuskan untuk menggunakan metode yang berbeda untuk mengganti node pekerja Anda, sangat disarankan agar Anda mengotomatiskan proses untuk meminimalkan pengawasan manusia karena Anda mungkin perlu mengganti pekerja secara teratur saat baru updates/patches dirilis dan ketika bidang kontrol ditingkatkan.

Dengan EKS Fargate, AWS akan secara otomatis memperbarui infrastruktur yang mendasarinya saat pembaruan tersedia. Seringkali hal ini dapat dilakukan dengan mulus, tetapi mungkin ada kalanya pembaruan akan menyebabkan pod Anda dijadwalkan ulang. Oleh karena itu, kami menyarankan Anda membuat deployment dengan beberapa replika saat menjalankan aplikasi Anda sebagai pod Fargate.

Jalankan kube-bench secara berkala untuk memverifikasi kepatuhan terhadap benchmark CIS untuk Kubernetes

kube-bench adalah proyek open source dari Aqua yang mengevaluasi cluster Anda terhadap benchmark CIS untuk Kubernetes. Tolok ukur ini menjelaskan praktik terbaik untuk mengamankan klaster Kubernetes yang tidak dikelola. Benchmark CIS Kubernetes mencakup bidang kontrol dan bidang data. Karena Amazon EKS menyediakan pesawat kontrol yang dikelola sepenuhnya, tidak semua rekomendasi dari Tolok Ukur CIS Kubernetes dapat diterapkan. Untuk memastikan cakupan ini mencerminkan bagaimana Amazon EKS diimplementasikan, AWS membuat Benchmark CIS Amazon EKS. Benchmark EKS mewarisi dari CIS Kubernetes Benchmark dengan masukan tambahan dari komunitas dengan pertimbangan konfigurasi khusus untuk kluster EKS.

Saat menjalankan kube-bench terhadap cluster EKS, ikuti instruksi ini dari Aqua Security. Untuk informasi lebih lanjut lihat Memperkenalkan Benchmark CIS Amazon EKS.

Minimalkan akses ke node pekerja

Alih-alih mengaktifkan akses SSH, gunakan SSM Session Manager saat Anda perlu remote ke host. Tidak seperti kunci SSH yang dapat hilang, disalin, atau dibagikan, Session Manager memungkinkan Anda untuk mengontrol akses ke instans EC2 menggunakan IAM. Selain itu, ia menyediakan jejak audit dan log perintah yang dijalankan pada instance.

Per 19 Agustus 2020 Grup Node Terkelola mendukung AMI khusus dan Template Peluncuran EC2. Ini memungkinkan Anda untuk menyematkan agen SSM ke AMI atau menginstalnya saat node pekerja sedang di-bootstrap. Jika Anda lebih suka tidak memodifikasi AMI yang Dioptimalkan atau template peluncuran ASG, Anda dapat menginstal agen SSM dengan DaemonSet seperti dalam contoh ini.

Kebijakan IAM minimal untuk Akses SSH berbasis SSM

Kebijakan `AmazonSSMManagedInstanceCore` AWS managed berisi sejumlah izin yang tidak diperlukan untuk SSM Session Manager/SSM RunCommand jika Anda hanya ingin menghindari akses SSH. Yang menjadi perhatian khusus adalah `*` izin `ssm:GetParameter(s)` yang memungkinkan peran mengakses semua parameter di Parameter Store (termasuk SecureStrings dengan kunci KMS terkelola AWS yang dikonfigurasi).

Kebijakan IAM berikut berisi kumpulan izin minimal untuk mengaktifkan akses node melalui 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": "*"
    }
  ]
}
```

Dengan kebijakan ini dan plugin Session Manager diinstal, Anda kemudian dapat menjalankannya

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

untuk mengakses node.

> **Note**
>
> Anda mungkin juga ingin mempertimbangkan untuk menambahkan izin untuk mengaktifkan pencatatan Manajer Sesi.

Menyebarkan pekerja ke subnet pribadi

Dengan mengerahkan pekerja ke subnet pribadi, Anda meminimalkan eksposur mereka ke Internet di mana serangan sering berasal. Mulai 22 April 2020, penugasan alamat IP publik ke node dalam grup node yang dikelola akan dikendalikan oleh subnet tempat mereka digunakan. Sebelum ini, node dalam Grup Node Terkelola secara otomatis diberi IP publik. Jika Anda memilih untuk menerapkan node pekerja ke subnet publik, terapkan aturan grup keamanan AWS yang membatasi untuk membatasi eksposur mereka.

Jalankan Amazon Inspector untuk menilai host untuk eksposur, kerentanan, dan penyimpangan dari praktik terbaik

Anda dapat menggunakan Amazon Inspector untuk memeriksa akses jaringan yang tidak diinginkan ke node dan kerentanan pada instans Amazon EC2 yang mendasarinya.

Amazon Inspector dapat memberikan data kerentanan dan eksposur umum (CVE) untuk instans Amazon EC2 Anda hanya jika agen Amazon EC2 Systems Manager (SSM) diinstal dan diaktifkan. Agen ini sudah diinstal sebelumnya di beberapa Amazon Machine Images (AMIs) termasuk Amazon Linux AMIs yang dioptimalkan EKS. Terlepas dari status agen SSM, semua instans Amazon EC2 Anda dipindai untuk masalah jangkauan jaringan. Untuk informasi selengkapnya tentang mengonfigurasi pemindaian untuk Amazon EC2, lihat Memindai instans Amazon EC2.

> **Important**
>
> Inspector tidak dapat dijalankan pada infrastruktur yang digunakan untuk menjalankan pod Fargate.

Alternatif

Jalankan SELinux

> **Note**
>
> Tersedia di Red Hat Enterprise Linux (RHEL), CentOS, Bottlerocket, dan Amazon Linux 2023

SELinux menyediakan lapisan keamanan tambahan untuk menjaga kontainer terisolasi satu sama lain dan dari host. SELinux memungkinkan administrator untuk menegakkan kontrol akses wajib (MAC) untuk setiap pengguna, aplikasi, proses, dan file. Anggap saja sebagai backstop yang membatasi operasi yang dapat dilakukan terhadap sumber daya tertentu berdasarkan serangkaian label. Pada EKS, SELinux dapat digunakan untuk mencegah kontainer mengakses sumber daya masing-masing.

SELinux Kebijakan kontainer didefinisikan dalam paket container-selinux. Docker CE memerlukan paket ini (bersama dengan dependensinya) sehingga proses dan file yang dibuat oleh Docker (atau runtime kontainer lainnya) berjalan dengan akses sistem yang terbatas. Container memanfaatkan `container_t` label yang merupakan alias untuk`svirt_lxc_net_t`. Kebijakan ini secara efektif mencegah kontainer mengakses fitur tertentu dari host.

Saat Anda mengonfigurasi SELinux untuk Docker, Docker secara otomatis memberi label beban kerja `container_t` sebagai tipe dan memberi setiap kontainer tingkat MCS yang unik. Ini akan mengisolasi wadah satu sama lain. Jika Anda memerlukan pembatasan yang lebih longgar, Anda dapat membuat profil Anda sendiri SElinux yang memberikan izin kontainer ke area tertentu dari sistem file. Ini mirip dengan PSPs di mana Anda dapat membuat profil yang berbeda untuk wadah/pod yang berbeda. Misalnya, Anda dapat memiliki profil untuk beban kerja umum dengan serangkaian kontrol terbatas dan lainnya untuk hal-hal yang memerlukan akses istimewa.

SELinux untuk Kontainer memiliki serangkaian opsi yang dapat dikonfigurasi untuk memodifikasi batasan default. SELinux Boolean berikut dapat diaktifkan atau dinonaktifkan berdasarkan kebutuhan Anda:

Boolean	Default	Deskripsi
`container_connect_any`	`off`	Izinkan kontainer mengakses port istimewa di host. Misalnya, jika Anda memiliki wadah yang perlu memetakan port ke 443 atau 80 pada host.
`container_manage_cgroup`	`off`	Izinkan kontainer untuk mengelola konfigurasi cgroup. Misalnya, wadah yang menjalankan systemd akan membutuhkan ini untuk diaktifkan.
`container_use_cephfs`	`off`	Izinkan kontainer menggunakan sistem file ceph.

Secara default, kontainer diizinkan untuk read/execute di bawah `/usr` dan membaca sebagian besar konten dari`/etc`. File di bawah `/var/lib/docker` dan `/var/lib/containers` memiliki label`container_var_lib_t`. Untuk melihat daftar lengkap default, label melihat file 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
```

File berlabel `container_file_t` adalah satu-satunya file yang dapat ditulis oleh kontainer. Jika Anda ingin volume mount dapat ditulis, Anda harus menentukan `:z` atau `:Z` di akhir.

- `:z`akan memberi label ulang file sehingga wadah dapat membaca/menulis

- `:`Zakan memberi label ulang file sehingga hanya wadah yang dapat membaca/menulis

```
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
```

Di Kubernetes, pelabelan ulang sedikit berbeda. Daripada meminta Docker memberi label ulang file secara otomatis, Anda dapat menentukan label MCS khusus untuk menjalankan pod. Volume yang mendukung pelabelan ulang secara otomatis akan diberi label ulang sehingga dapat diakses. Pod dengan label MCS yang cocok akan dapat mengakses volume. Jika Anda membutuhkan isolasi yang ketat, tetapkan label MCS yang berbeda untuk setiap 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
```

Dalam contoh ini `s0:c144:c154` sesuai dengan label MCS yang ditetapkan ke file yang wadah diizinkan untuk diakses.

Di EKS Anda dapat membuat kebijakan yang memungkinkan wadah istimewa berjalan, seperti fluentD dan membuat SELinux kebijakan untuk mengizinkannya membaca dari/var/log di host tanpa perlu memberi label ulang pada direktori host. Pod dengan label yang sama akan dapat mengakses volume host yang sama.

Kami telah menerapkan sampel AMIs untuk Amazon EKS yang telah SELinux dikonfigurasi pada CentOS 7 dan RHEL 7. Ini AMIs dikembangkan untuk menunjukkan implementasi sampel yang memenuhi persyaratan pelanggan yang sangat diatur.

> ⚠ Warning
>
> SELinux akan mengabaikan wadah di mana tipenya tidak terbatas.

Alat dan sumber daya

- SELinuxKubernetes RBAC dan Kebijakan Keamanan Pengiriman untuk Aplikasi On-PREM
- Pengerasan Iteratif Kubernetes
- Audit2Izinkan
- SEAlert
- Buat SELinux kebijakan untuk kontainer dengan Udica menjelaskan alat yang melihat file spesifikasi kontainer untuk kemampuan Linux, port, dan titik pemasangan, dan menghasilkan seperangkat SELinux aturan yang memungkinkan penampung berjalan dengan benar
- Playbook AMI Hardening untuk pengerasan OS untuk memenuhi persyaratan peraturan yang berbeda
- Keiko Upgrade Manager proyek open source dari Intuit yang mengatur rotasi node pekerja.
- Sysdig Aman
- eksctl

Kepatuhan

Kepatuhan adalah tanggung jawab bersama antara AWS dan konsumen layanannya. Secara umum, AWS bertanggung jawab atas “keamanan cloud” sedangkan penggunanya bertanggung jawab atas “keamanan di cloud.” Garis yang menggambarkan apa yang AWS dan penggunanya bertanggung jawab akan bervariasi tergantung pada layanan. Misalnya, dengan Fargate, AWS bertanggung jawab untuk mengelola keamanan fisik pusat datanya, perangkat keras, infrastruktur virtual (Amazon EC2), dan runtime kontainer (Docker). Pengguna Fargate bertanggung jawab untuk mengamankan gambar kontainer dan aplikasinya. Mengetahui siapa yang bertanggung jawab atas apa yang menjadi pertimbangan penting saat menjalankan beban kerja yang harus mematuhi standar kepatuhan.

Tabel berikut menunjukkan program kepatuhan yang sesuai dengan layanan kontainer yang berbeda.

Program Kepatuhan	Orkestra Amazon ECS	Orkestra Amazon EKS	ECS Fargate	Amazon ECR
PCI DSS Tingkat 1	1	1	1	1
Memenuhi Syarat HIPAA	1	1	1	1
SOC SAYA	1	1	1	1
SOC II	1	1	1	1
SOC III	1	1	1	1
ISO 27001:2013	1	1	1	1
ISO 9001:2015	1	1	1	1
ISO 27017:2015	1	1	1	1
ISO 27018:2019	1	1	1	1
IRAP	1	1	1	1
FedRAMP Moderat (Timur/Barat)	1	1	0	1
FedRAMP Tinggi () GovCloud	1	1	0	1
DOD CC SRG	1	Ulasan DISA () IL5	0	1
HIPAA BAA	1	1	1	1
MTCS	1	1	0	1
C5	1	1	0	1

Program Kepatuhan	Orkestra Amazon ECS	Orkestra Amazon EKS	ECS Fargate	Amazon ECR
K-ISMS	1	1	0	1
ENS High	1	1	0	1
OSPAR	1	1	0	1
HITRUST CSF	1	1	1	1

Status kepatuhan berubah seiring waktu. Untuk status terbaru, selalu lihat https://aws.amazon.com/compliance/ services-in-scope /.

Untuk informasi lebih lanjut tentang model akreditasi cloud dan praktik terbaik, lihat whitepaper AWS, Model Akreditasi untuk Adopsi Cloud Aman

Bergeser Kiri

Konsep pergeseran ke kiri melibatkan penangkapan pelanggaran kebijakan dan kesalahan sebelumnya dalam siklus hidup pengembangan perangkat lunak. Dari perspektif keamanan, ini bisa sangat bermanfaat. Pengembang, misalnya, dapat memperbaiki masalah dengan konfigurasi mereka sebelum aplikasi mereka diterapkan ke cluster. Menangkap kesalahan seperti ini sebelumnya akan membantu mencegah konfigurasi yang melanggar kebijakan Anda agar tidak diterapkan.

Kebijakan sebagai Kode

Kebijakan dapat dianggap sebagai seperangkat aturan untuk mengatur perilaku, yaitu perilaku yang diizinkan atau yang dilarang. Misalnya, Anda mungkin memiliki kebijakan yang mengatakan bahwa semua Dockerfiles harus menyertakan direktif USER yang menyebabkan container berjalan sebagai pengguna non-root. Sebagai dokumen, kebijakan seperti ini bisa sulit ditemukan dan ditegakkan. Ini juga bisa menjadi usang karena kebutuhan Anda berubah. Dengan solusi Policy as Code (PAC), Anda dapat mengotomatiskan kontrol keamanan, kepatuhan, dan privasi yang mendeteksi, mencegah, mengurangi, dan menangkal ancaman yang diketahui dan terus-menerus. Selain itu, mereka memberi Anda mekanisme untuk mengkodifikasi kebijakan Anda dan mengelolanya saat Anda melakukan artefak kode lainnya. Manfaat dari pendekatan ini adalah Anda dapat menggunakan kembali GitOps strategi Anda untuk mengelola DevOps dan menerapkan kebijakan secara konsisten

di seluruh armada klaster Kubernetes. Silakan merujuk ke Pod Security untuk informasi tentang opsi PAC dan masa depan. PSPs

Gunakan policy-as-code alat di saluran pipa untuk mendeteksi pelanggaran sebelum penerapan

- OPA adalah mesin kebijakan open source yang merupakan bagian dari CNCF. Ini digunakan untuk membuat keputusan kebijakan dan dapat dijalankan dengan berbagai cara yang berbeda, misalnya sebagai perpustakaan bahasa atau layanan. Kebijakan OPA ditulis dalam Domain Specific Language (DSL) yang disebut Rego. Meskipun sering dijalankan sebagai bagian dari Kubernetes Dynamic Admission Controller sebagai proyek Gatekeeper, OPA juga dapat dimasukkan ke dalam pipeline Anda. CI/CD Hal ini memungkinkan pengembang untuk mendapatkan umpan balik tentang konfigurasi mereka di awal siklus rilis yang selanjutnya dapat membantu mereka menyelesaikan masalah sebelum mereka sampai ke produksi. Kumpulan kebijakan OPA umum dapat ditemukan di GitHub repositori untuk proyek ini.
- Conftest dibangun di atas OPA dan memberikan pengalaman yang berfokus pada pengembang untuk menguji konfigurasi Kubernetes.
- Kyverno adalah mesin kebijakan yang dirancang untuk Kubernetes. Dengan Kyverno, kebijakan dikelola sebagai sumber daya Kubernetes dan tidak diperlukan bahasa baru untuk menulis kebijakan. Ini memungkinkan penggunaan alat yang sudah dikenal seperti kubectl, git, dan kustomize untuk mengelola kebijakan. Kebijakan Kyverno dapat memvalidasi, mengubah, dan menghasilkan sumber daya Kubernetes plus memastikan keamanan rantai pasokan gambar OCI. CLI Kyverno dapat digunakan untuk menguji kebijakan dan memvalidasi sumber daya sebagai bagian dari pipeline. CI/CD Semua kebijakan komunitas Kyverno dapat ditemukan di situs web Kyverno, dan untuk contoh menggunakan CLI Kyverno untuk menulis tes di saluran pipa, lihat repositori kebijakan.

Alat dan sumber daya

- Lokakarya Perendaman Keamanan Amazon EKS - Kepatuhan Peraturan
- kube-bangku
- docker-bench-security
- AWS Inspector
- Tinjauan Keamanan Kubernetes Penilaian keamanan pihak ketiga Kubernetes 1.13.4 (2019)

- NeuVector oleh SUSE open source, platform keamanan wadah tanpa kepercayaan, menyediakan pelaporan kepatuhan dan pemeriksaan kepatuhan khusus

Respon insiden dan forensik

Kemampuan Anda untuk bereaksi cepat terhadap suatu insiden dapat membantu meminimalkan kerusakan yang disebabkan oleh pelanggaran. Memiliki sistem peringatan yang andal yang dapat memperingatkan Anda tentang perilaku mencurigakan adalah langkah pertama dalam rencana respons insiden yang baik. Ketika sebuah insiden muncul, Anda harus segera memutuskan apakah akan menghancurkan dan mengganti wadah yang terkena dampak, atau mengisolasi dan memeriksa wadah. Jika Anda memilih untuk mengisolasi wadah sebagai bagian dari penyelidikan forensik dan analisis akar penyebab, maka serangkaian kegiatan berikut harus diikuti:

Contoh rencana respons insiden

Identifikasi Pod yang menyinggung dan node pekerja

Tindakan pertama Anda harus mengisolasi kerusakan. Mulailah dengan mengidentifikasi di mana pelanggaran terjadi dan mengisolasi Pod tersebut dan simpulnya dari infrastruktur lainnya.

Identifikasi Pod dan node pekerja yang menyinggung menggunakan nama beban kerja

Jika Anda mengetahui nama dan namespace dari pod yang menyinggung, Anda dapat mengidentifikasi node pekerja yang menjalankan pod sebagai berikut:

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

Jika Sumber Daya Beban Kerja seperti Deployment telah dikompromikan, kemungkinan semua Pod yang merupakan bagian dari sumber daya beban kerja dikompromikan. Gunakan perintah berikut untuk membuat daftar semua pod dari Workload Resource dan node yang sedang dijalankan:

```
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)"'
```

Perintah di atas adalah untuk penerapan. Anda dapat menjalankan perintah yang sama untuk sumber daya beban kerja lainnya seperti replicasets,, statefulsets, dll.

Identifikasi Pod dan node pekerja yang melanggar menggunakan nama akun layanan

Dalam beberapa kasus, Anda dapat mengidentifikasi bahwa akun layanan disusupi. Kemungkinan pod yang menggunakan akun layanan yang diidentifikasi dikompromikan. Anda dapat mengidentifikasi semua pod menggunakan akun layanan dan node yang dijalankan dengan perintah berikut:

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

Identifikasi Pod dengan gambar dan node pekerja yang rentan atau disusupi

Dalam beberapa kasus, Anda mungkin menemukan bahwa image kontainer yang digunakan dalam pod di klaster Anda berbahaya atau disusupi. Gambar kontainer berbahaya atau dikompromikan, jika ditemukan mengandung malware, adalah gambar buruk yang diketahui atau memiliki CVE yang telah dieksploitasi. Anda harus mempertimbangkan semua pod menggunakan image kontainer yang dikompromikan. Anda dapat mengidentifikasi pod menggunakan image dan node yang mereka jalankan dengan perintah berikut:

```
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)"'
```

Mengisolasi Pod dengan membuat Kebijakan Jaringan yang menolak semua lalu lintas masuk dan keluar ke pod

Menyangkal semua aturan lalu lintas dapat membantu menghentikan serangan yang sudah berlangsung dengan memutuskan semua koneksi ke pod. Kebijakan Jaringan berikut akan berlaku untuk pod dengan label`app=web`.

```
apiVersion: networking.k8s.io/v1
```

```
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector:
    matchLabels:
      app: web
  policyTypes:
  - Ingress
  - Egress
```

Important

Kebijakan Jaringan mungkin terbukti tidak efektif jika penyerang telah mendapatkan akses ke host yang mendasarinya. Jika Anda mencurigai hal itu terjadi, Anda dapat menggunakan AWS Security Groups untuk mengisolasi host yang disusupi dari host lain. Saat mengubah grup keamanan host, ketahuilah bahwa itu akan memengaruhi semua kontainer yang berjalan di host tersebut.

Mencabut kredensi keamanan sementara yang ditetapkan ke pod atau node pekerja jika perlu

Jika node pekerja telah diberi peran IAM yang memungkinkan Pod mendapatkan akses ke sumber daya AWS lainnya, hapus peran tersebut dari instance untuk mencegah kerusakan lebih lanjut dari serangan. Demikian pula, jika Pod telah diberi peran IAM, evaluasi apakah Anda dapat menghapus kebijakan IAM dari peran dengan aman tanpa memengaruhi beban kerja lainnya.

Cordon simpul pekerja

Dengan menutup node pekerja yang terkena dampak, Anda memberi tahu penjadwal untuk menghindari penjadwalan pod ke node yang terpengaruh. Ini akan memungkinkan Anda untuk menghapus node untuk studi forensik tanpa mengganggu beban kerja lainnya.

Note

Panduan ini tidak berlaku untuk Fargate di mana setiap pod Fargate berjalan di lingkungan kotak pasirnya sendiri. Alih-alih mengepung, serapkan pod Fargate yang terpengaruh dengan menerapkan kebijakan jaringan yang menyangkal semua lalu lintas masuk dan keluar.

Aktifkan perlindungan penghentian pada node pekerja yang terkena dampak

Seorang penyerang dapat mencoba untuk menghapus kesalahan mereka dengan mengakhiri node yang terpengaruh. Mengaktifkan perlindungan terminasi dapat mencegah hal ini terjadi. Perlindungan skala dalam instance akan melindungi node dari peristiwa scale-in.

> **Warning**
>
> Anda tidak dapat mengaktifkan perlindungan penghentian pada instans Spot.

Beri label pelanggaran Pod/Node dengan label yang menunjukkan bahwa itu adalah bagian dari penyelidikan aktif

Ini akan berfungsi sebagai peringatan bagi administrator klaster untuk tidak mengutak-atik yang terkena dampak Pods/Nodes sampai penyelidikan selesai.

Tangkap artefak yang mudah menguap pada node pekerja

- Tangkap memori sistem operasi. Ini akan menangkap daemon Docker (atau runtime kontainer lainnya) dan subprosesnya per kontainer. Ini dapat dicapai dengan menggunakan alat seperti LiME dan Volatilitas, atau melalui alat tingkat yang lebih tinggi seperti Automated Forensics Orchestrator untuk Amazon EC2 yang dibangun di atasnya.
- Lakukan dump pohon netstat dari proses yang berjalan dan port terbuka. Ini akan menangkap daemon docker dan subprosesnya per kontainer.
- Jalankan perintah untuk menyimpan status tingkat kontainer sebelum bukti diubah. Anda dapat menggunakan kemampuan runtime kontainer untuk menangkap informasi tentang kontainer yang sedang berjalan. Misalnya, dengan Containerd, Anda dapat melakukan hal berikut:
  - `crictl ps`untuk proses yang berjalan.
  - `crictl logs CONTAINER`untuk log yang dimiliki tingkat daemon.

    Hal yang sama dapat dicapai dengan containerd menggunakan nerdctl CLI, sebagai pengganti (misalnya). `docker` `nerdctl inspect` Beberapa perintah tambahan tersedia tergantung pada runtime kontainer. Misalnya, Docker `docker diff` harus melihat perubahan pada sistem file kontainer atau `docker checkpoint` untuk menyimpan semua status kontainer termasuk memori volatil (RAM). Lihat posting blog Kubernetes ini untuk diskusi tentang kemampuan serupa dengan runtime containerd atau CRI-O.

- Jeda wadah untuk penangkapan forensik.
- Cuplikan volume EBS instans.

Menerapkan ulang Pod atau Sumber Daya Beban Kerja yang dikompromikan

Setelah mengumpulkan data untuk analisis forensik, Anda dapat menerapkan ulang pod atau sumber daya beban kerja yang dikompromikan.

Pertama-tama luncurkan perbaikan untuk kerentanan yang dikompromikan dan mulai pod pengganti baru. Kemudian hapus pod yang rentan.

Jika pod yang rentan dikelola oleh sumber daya beban kerja Kubernetes tingkat tinggi (misalnya, Deployment atau DaemonSet), menghapusnya akan menjadwalkan yang baru. Jadi pod yang rentan akan diluncurkan lagi. Dalam hal ini, Anda harus menerapkan sumber daya beban kerja pengganti baru setelah memperbaiki kerentanan. Maka Anda harus menghapus beban kerja yang rentan.

Rekomendasi

Tinjau Whitepaper Respons Insiden Keamanan AWS

Meskipun bagian ini memberikan gambaran singkat bersama dengan beberapa rekomendasi untuk menangani dugaan pelanggaran keamanan, topik ini dibahas secara mendalam dalam white paper, AWS Security Incident Response.

Berlatih hari permainan keamanan

Bagilah praktisi keamanan Anda menjadi 2 tim: merah dan biru. Tim merah akan fokus pada menyelidiki sistem yang berbeda untuk kerentanan sementara tim biru akan bertanggung jawab untuk bertahan melawan mereka. Jika Anda tidak memiliki cukup praktisi keamanan untuk membuat tim terpisah, pertimbangkan untuk menyewa entitas luar yang memiliki pengetahuan tentang eksploitasi Kubernetes.

Kubesploit adalah kerangka pengujian penetrasi CyberArk yang dapat Anda gunakan untuk melakukan hari permainan. Tidak seperti alat lain yang memindai cluster Anda untuk kerentanan, kubesploit mensimulasikan serangan dunia nyata. Ini memberi tim biru Anda kesempatan untuk melatih responsnya terhadap serangan dan mengukur efektivitasnya.

Jalankan tes penetrasi terhadap cluster Anda

Menyerang cluster Anda secara berkala dapat membantu Anda menemukan kerentanan dan kesalahan konfigurasi. Sebelum memulai, ikuti pedoman pengujian penetrasi sebelum melakukan tes terhadap cluster Anda.

Alat dan sumber daya

- kube-hunter, alat pengujian penetrasi untuk Kubernetes.
- Gremlin, toolkit rekayasa kekacauan yang dapat Anda gunakan untuk mensimulasikan serangan terhadap aplikasi dan infrastruktur Anda.
- Menyerang dan Mempertahankan Instalasi Kubernetes
- kubesploit
- NeuVector oleh SUSE open source, platform keamanan wadah tanpa kepercayaan, menyediakan pelaporan kerentanan dan risiko serta pemberitahuan peristiwa keamanan
- Ancaman Persisten Tingkat Lanjut
- Serangan dan Pertahanan Praktis Kubernetes
- Mengkompromikan Cluster Kubernetes dengan Memanfaatkan Izin RBAC

Keamanan gambar

Anda harus mempertimbangkan gambar kontainer sebagai garis pertahanan pertama Anda terhadap serangan. Gambar yang tidak aman dan dibangun dengan buruk dapat memungkinkan penyerang melarikan diri dari batas-batas wadah dan mendapatkan akses ke host. Setelah berada di host, penyerang dapat memperoleh akses ke informasi sensitif atau bergerak secara lateral di dalam cluster atau dengan akun AWS Anda. Praktik terbaik berikut akan membantu mengurangi risiko terjadinya hal ini.

Rekomendasi

Buat gambar minimal

Mulailah dengan menghapus semua binari asing dari gambar kontainer. Jika Anda menggunakan gambar yang tidak dikenal dari Dockerhub, periksa gambar menggunakan aplikasi seperti Dive yang dapat menampilkan konten dari setiap lapisan wadah. Hapus semua binari dengan bit SETUID

dan SETGID karena dapat digunakan untuk meningkatkan hak istimewa dan pertimbangkan untuk menghapus semua shell dan utilitas seperti nc dan curl yang dapat digunakan untuk tujuan jahat. Anda dapat menemukan file dengan bit SETUID dan SETGID dengan perintah berikut:

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

Untuk menghapus izin khusus dari file-file ini, tambahkan direktif berikut ke gambar kontainer Anda:

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

Dalam bahasa sehari-hari, ini dikenal sebagai de-fanging your image.

Gunakan build multi-tahap

Menggunakan build multi-tahap adalah cara untuk membuat gambar minimal. Seringkali, build multi-tahap digunakan untuk mengotomatiskan bagian-bagian dari siklus Integrasi Berkelanjutan. Misalnya, build multi-tahap dapat digunakan untuk lint kode sumber Anda atau melakukan analisis kode statis. Ini memberi pengembang kesempatan untuk mendapatkan umpan balik langsung daripada menunggu pipeline untuk dijalankan. Build multi-tahap menarik dari sudut pandang keamanan karena memungkinkan Anda meminimalkan ukuran gambar akhir yang didorong ke registri kontainer Anda. Gambar kontainer tanpa alat build dan binari asing lainnya meningkatkan postur keamanan Anda dengan mengurangi permukaan serangan gambar. Untuk informasi tambahan tentang build multi-tahap, lihat dokumentasi build multi-tahap Docker.

Buat Software Bill of Materials (SBOMs) untuk gambar kontainer Anda

Sebuah “software bill of materials” (SBOM) adalah inventaris bersarang dari artefak perangkat lunak yang membentuk gambar kontainer Anda. SBOM adalah blok bangunan utama dalam keamanan perangkat lunak dan manajemen risiko rantai pasokan perangkat lunak. Menghasilkan, menyimpan SBOMS di repositori pusat dan memindai SBOMs kerentanan membantu mengatasi masalah berikut:

- Visibilitas: pahami komponen apa yang membentuk gambar kontainer Anda. Menyimpan di repositori pusat memungkinkan SBOMs untuk diaudit dan dipindai kapan saja, bahkan pasca penyebaran untuk mendeteksi dan menanggapi kerentanan baru seperti kerentanan nol hari.
- Verifikasi Asal: jaminan bahwa asumsi yang ada tentang di mana dan bagaimana artefak berasal adalah benar dan bahwa artefak atau metadata yang menyertainya belum dirusak selama proses pembuatan atau pengiriman.

- Kepercayaan: jaminan bahwa artefak tertentu dan isinya dapat dipercaya untuk melakukan apa yang seharusnya dilakukan, yaitu cocok untuk suatu tujuan. Ini melibatkan penilaian apakah kode tersebut aman untuk dieksekusi dan membuat keputusan berdasarkan informasi tentang risiko yang terkait dengan pelaksanaan kode. Kepercayaan terjamin dengan membuat laporan eksekusi pipa yang dibuktikan bersama dengan SBOM yang telah dibuktikan dan laporan pemindaian CVE yang dibuktikan untuk meyakinkan konsumen gambar bahwa gambar ini sebenarnya dibuat melalui sarana aman (pipeline) dengan komponen yang aman.
- Verifikasi Kepercayaan Ketergantungan: pemeriksaan rekursif pohon ketergantungan artefak untuk kepercayaan dan asal artefak yang digunakannya. Drift in SBOMs dapat membantu mendeteksi aktivitas berbahaya termasuk dependensi yang tidak sah, tidak tepercaya, upaya infiltrasi.

Alat-alat berikut dapat digunakan untuk menghasilkan SBOM:

- Amazon Inspector dapat digunakan untuk membuat dan mengekspor. SBOMs
- Syft dari Anchore juga dapat digunakan untuk generasi SBOM. Untuk pemindaian kerentanan yang lebih cepat, SBOM yang dihasilkan untuk gambar kontainer dapat digunakan sebagai input untuk memindai. Laporan SBOM dan pemindaian kemudian dibuktikan dan dilampirkan pada gambar sebelum mendorong gambar ke repositori OCI pusat seperti Amazon ECR untuk tujuan peninjauan dan audit.

Pelajari lebih lanjut tentang mengamankan rantai pasokan perangkat lunak Anda dengan meninjau panduan Praktik Terbaik Rantai Pasokan Perangkat Lunak CNCF.

Pindai gambar untuk kerentanan secara teratur

Seperti rekan-rekan mesin virtual mereka, gambar kontainer dapat berisi binari dan pustaka aplikasi dengan kerentanan atau mengembangkan kerentanan dari waktu ke waktu. Cara terbaik untuk melindungi terhadap eksploitasi adalah dengan memindai gambar Anda secara teratur dengan pemindai gambar. Gambar yang disimpan di Amazon ECR dapat dipindai saat push atau on-demand (sekali selama periode 24 jam). ECR saat ini mendukung dua jenis pemindaian - Basic dan Enhanced. Pemindaian dasar memanfaatkan Clair solusi pemindaian gambar open source tanpa biaya. Pemindaian yang disempurnakan menggunakan Amazon Inspector untuk menyediakan pemindaian berkelanjutan otomatis dengan biaya tambahan. Setelah gambar dipindai, hasilnya dicatat ke aliran acara untuk ECR di. EventBridge Anda juga dapat melihat hasil pemindaian dari dalam konsol ECR. Gambar dengan kerentanan TINGGI atau KRITIS harus dihapus atau dibangun kembali. Jika gambar yang telah digunakan mengembangkan kerentanan, itu harus diganti sesegera mungkin.

Mengetahui di mana gambar dengan kerentanan telah digunakan sangat penting untuk menjaga lingkungan Anda tetap aman. Meskipun Anda dapat membuat sendiri solusi pelacakan gambar, sudah ada beberapa penawaran komersial yang menyediakan ini dan kemampuan canggih lainnya di luar kotak, termasuk:

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

Webhook validasi Kubernetes juga dapat digunakan untuk memvalidasi bahwa gambar bebas dari kerentanan kritis. Webhook validasi dipanggil sebelum API Kubernetes. Mereka biasanya digunakan untuk menolak permintaan yang tidak sesuai dengan kriteria validasi yang ditentukan dalam webhook. Ini adalah contoh webhook tanpa server yang memanggil API describeImageScan Temuan ECR untuk menentukan apakah sebuah pod menarik gambar dengan kerentanan kritis. Jika kerentanan ditemukan, pod ditolak dan pesan dengan daftar CVEs dikembalikan sebagai Event.

Gunakan pengesahan untuk memvalidasi integritas artefak

Pengesahan adalah “pernyataan” yang ditandatangani secara kriptografis yang mengklaim sesuatu - “predikat” misalnya pipeline run atau SBOM atau laporan pemindaian kerentanan benar tentang hal lain - “subjek” yaitu gambar wadah.

Pengesahan membantu pengguna untuk memvalidasi bahwa artefak berasal dari sumber tepercaya dalam rantai pasokan perangkat lunak. Sebagai contoh, kita dapat menggunakan gambar kontainer tanpa mengetahui semua komponen perangkat lunak atau dependensi yang disertakan dalam gambar itu. Namun, jika kita mempercayai apa pun yang dikatakan produsen gambar kontainer tentang perangkat lunak apa yang ada, kita dapat menggunakan pengesahan produsen untuk mengandalkan artefak itu. Ini berarti bahwa kita dapat melanjutkan untuk menggunakan artefak dengan aman di alur kerja kita daripada melakukan analisis sendiri.

- Pengesahan dapat dibuat menggunakan AWS Signer atau Sigstore cosign.
- Pengontrol penerimaan Kubernetes seperti Kyverno dapat digunakan untuk memverifikasi pengesahan.

- Lihat lokakarya ini untuk mempelajari lebih lanjut tentang praktik terbaik manajemen rantai pasokan perangkat lunak di AWS menggunakan alat sumber terbuka dengan topik termasuk membuat dan melampirkan pengesahan ke gambar kontainer.

Buat kebijakan IAM untuk repositori ECR

Saat ini, tidak jarang sebuah organisasi memiliki beberapa tim pengembangan yang beroperasi secara independen dalam akun AWS bersama. Jika tim ini tidak perlu berbagi aset, Anda mungkin ingin membuat serangkaian kebijakan IAM yang membatasi akses ke repositori yang dapat berinteraksi dengan setiap tim. Cara yang baik untuk mengimplementasikan ini adalah dengan menggunakan ruang nama ECR. Ruang nama adalah cara untuk mengelompokkan repositori serupa bersama-sama. Misalnya, semua registrasi untuk tim A dapat diawali dengan tim-a/ sedangkan untuk tim B dapat menggunakan awalan tim-b/. Kebijakan untuk membatasi akses mungkin terlihat seperti berikut:

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

Pertimbangkan untuk menggunakan titik akhir pribadi ECR

ECR API memiliki titik akhir publik. Akibatnya, pendaftar ECR dapat diakses dari Internet selama permintaan tersebut telah diautentikasi dan disahkan oleh IAM. Bagi mereka yang perlu beroperasi di lingkungan kotak pasir di mana VPC cluster tidak memiliki Internet Gateway (IGW), Anda dapat mengonfigurasi titik akhir pribadi untuk ECR. Membuat endpoint pribadi memungkinkan Anda mengakses ECR API secara pribadi melalui alamat IP pribadi alih-alih merutekan lalu lintas di Internet. Untuk informasi tambahan tentang topik ini, lihat Titik akhir VPC antarmuka Amazon ECR.

Menerapkan kebijakan endpoint untuk ECR

Kebijakan endpoint default untuk memungkinkan akses ke semua repositori ECR dalam suatu wilayah. Ini mungkin memungkinkan attacker/insider untuk mengekstrasi data dengan mengemasnya sebagai gambar kontainer dan mendorongnya ke registri di akun AWS lain. Mengurangi risiko ini melibatkan pembuatan kebijakan endpoint yang membatasi akses API ke repositori ECR. Misalnya, kebijakan berikut memungkinkan semua prinsip AWS di akun Anda untuk melakukan semua tindakan terhadap repositori ECR Anda dan hanya Anda:

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

Anda dapat menyempurnakannya lebih lanjut dengan menetapkan kondisi yang menggunakan `PrincipalOrgID` atribut baru yang akan mencegah pushing/pulling gambar dengan prinsip IAM yang bukan bagian dari Organisasi AWS Anda. Lihat, aws: PrincipalOrg ID untuk detail tambahan. Kami merekomendasikan menerapkan kebijakan yang sama untuk kedua titik akhir `com.amazonaws.<region>.ecr.dkr` dan `com.amazonaws.<region>.ecr.api` titik akhir. Karena EKS menarik gambar untuk kube-proxy, coredns, dan aws-node dari ECR, Anda perlu menambahkan ID akun registri, misalnya `602401143452.dkr.ecr.us-west-2.amazonaws.com/` ke daftar sumber daya dalam kebijakan titik akhir atau mengubah kebijakan untuk mengizinkan penarikan dari dan membatasi dorongan ke ID akun Anda. Tabel di

bawah ini mengungkapkan pemetaan antara akun AWS tempat gambar EKS dijual dan wilayah cluster.

Nomor Rekening	Region
602401143452	Semua wilayah komersial kecuali yang tercantum di bawah ini
—	—
800184023465	ap-east-1 - Asia Pasifik (Hong Kong)
558608220178	me-south-1 - Timur Tengah (Bahrain)
918309763551	cn-utara-1 - Tiongkok (Beijing)
961992271922	cn-barat laut-1 - Tiongkok (Ningxia)

Untuk informasi lebih lanjut tentang penggunaan kebijakan titik akhir, lihat Menggunakan kebijakan titik akhir VPC untuk mengontrol akses ECR Amazon.

Menerapkan kebijakan siklus hidup untuk ECR

Panduan Keamanan Kontainer Aplikasi NIST memperingatkan tentang risiko “gambar basi dalam pendaftar”, mencatat bahwa seiring waktu gambar lama dengan paket out-of-date perangkat lunak yang rentan harus dihapus untuk mencegah penyebaran dan paparan yang tidak disengaja. Setiap repositori ECR dapat memiliki kebijakan siklus hidup yang menetapkan aturan kapan gambar kedaluwarsa. Dokumentasi resmi AWS menjelaskan cara mengatur aturan pengujian, mengevaluasinya, dan kemudian menerapkannya. Ada beberapa contoh kebijakan siklus hidup di dokumen resmi yang menunjukkan berbagai cara memfilter gambar dalam repositori:

- Memfilter berdasarkan usia gambar atau hitungan
- Memfilter berdasarkan gambar yang ditandai atau tidak ditandai
- Memfilter berdasarkan tag gambar, baik dalam beberapa aturan atau satu aturan

??? + peringatan Jika gambar untuk aplikasi yang berjalan lama dibersihkan dari ECR, itu dapat menyebabkan kesalahan penarikan gambar saat aplikasi dipindahkan atau diskalakan secara horizontal. Saat menggunakan kebijakan siklus hidup gambar, pastikan Anda memiliki CI/CD

praktik yang baik untuk menjaga penerapan dan gambar yang mereka referensikan tetap mutakhir dan selalu membuat aturan kedaluwarsa [image] yang memperhitungkan seberapa sering Anda melakukan rilis/penerapan.

Buat satu set gambar yang dikuratori

Daripada mengizinkan pengembang untuk membuat gambar mereka sendiri, pertimbangkan untuk membuat satu set gambar yang diperiksa untuk tumpukan aplikasi yang berbeda di organisasi Anda. Dengan demikian, pengembang dapat mengabaikan belajar bagaimana menulis Dockerfiles dan berkonsentrasi pada penulisan kode. Saat perubahan digabungkan menjadi Master, CI/CD pipeline dapat secara otomatis mengkompilasi aset, menyimpannya dalam repositori artefak dan menyalin artefak ke gambar yang sesuai sebelum mendorongnya ke registri Docker seperti ECR. Setidaknya Anda harus membuat satu set gambar dasar dari mana pengembang untuk membuat Dockerfiles mereka sendiri. Idealnya, Anda ingin menghindari pengambilan gambar dari Dockerhub karena 1/ Anda tidak selalu tahu apa yang ada di gambar dan 2/sekitar seperlima dari 1000 gambar teratas memiliki kerentanan. Daftar gambar-gambar itu dan kerentanannya dapat ditemukan di sini.

Tambahkan direktif USER ke Dockerfiles Anda untuk dijalankan sebagai pengguna non-root

Seperti yang disebutkan di bagian keamanan pod, Anda harus menghindari menjalankan container sebagai root. Meskipun Anda dapat mengonfigurasi ini sebagai bagian dari PodSpec, itu adalah kebiasaan yang baik untuk menggunakan `USER` arahan ke Dockerfiles Anda. `USER`Direktif menetapkan UID untuk digunakan saat menjalankan`RUN`,`ENTRYPOINT`, atau `CMD` instruksi yang muncul setelah direktif USER.

Lint Dockerfiles Anda

Linting dapat digunakan untuk memverifikasi bahwa Dockerfiles Anda mengikuti serangkaian pedoman yang telah ditentukan sebelumnya, misalnya penyertaan `USER` arahan atau persyaratan bahwa semua gambar diberi tag. dockerfile_lint adalah proyek open source dari yang memverifikasi praktik terbaik umum dan menyertakan mesin aturan RedHat yang dapat Anda gunakan untuk membuat aturan Anda sendiri untuk linting Dockerfiles. Itu dapat dimasukkan ke dalam pipa CI, di mana build dengan Dockerfiles yang melanggar aturan akan secara otomatis gagal.

Membangun gambar dari Scratch

Mengurangi permukaan serangan gambar kontainer Anda harus menjadi tujuan utama saat membuat gambar. Cara ideal untuk melakukan ini adalah dengan membuat gambar minimal yang tanpa binari

yang dapat digunakan untuk mengeksploitasi kerentanan. Untungnya, Docker memiliki mekanisme untuk membuat gambar dari `scratch`. Dengan bahasa seperti Go, Anda dapat membuat biner tertaut statis dan mereferensikannya di Dockerfile Anda seperti dalam contoh ini:

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

Ini menciptakan gambar kontainer yang terdiri dari aplikasi Anda dan tidak ada yang lain, membuatnya sangat aman.

Gunakan tag yang tidak dapat diubah dengan ECR

Tag yang tidak dapat diubah memaksa Anda untuk memperbarui tag gambar pada setiap dorongan ke repositori gambar. Ini dapat menggagalkan penyerang dari menimpa gambar dengan versi berbahaya tanpa mengubah tag gambar. Selain itu, ini memberi Anda cara untuk mengidentifikasi gambar dengan mudah dan unik.

Tanda tangani gambar Anda SBOMs, proses pipeline, dan laporan kerentanan

Ketika Docker pertama kali diperkenalkan, tidak ada model kriptografi untuk memverifikasi gambar kontainer. Dengan v2, Docker menambahkan intisari ke manifes gambar. Ini memungkinkan konfigurasi gambar untuk di-hash dan untuk hash yang akan digunakan untuk menghasilkan ID untuk gambar. Saat penandatanganan gambar diaktifkan, mesin Docker memverifikasi tanda tangan manifes, memastikan bahwa konten dihasilkan dari sumber tepercaya dan tidak ada gangguan yang terjadi. Setelah setiap lapisan diunduh, mesin memverifikasi intisari lapisan, memastikan bahwa

konten cocok dengan konten yang ditentukan dalam manifes. Penandatanganan gambar secara efektif memungkinkan Anda membuat rantai pasokan yang aman, melalui verifikasi tanda tangan digital yang terkait dengan gambar.

Kami dapat menggunakan AWS Signer atau Sigstore Cosign, untuk menandatangani gambar kontainer, membuat pengesahan, laporan pemindaian kerentanan SBOMs, dan laporan proses pipeline. Pengesahan ini menjamin kepercayaan dan integritas gambar, bahwa itu sebenarnya dibuat oleh pipa tepercaya tanpa gangguan atau gangguan, dan bahwa itu hanya berisi komponen perangkat lunak yang didokumentasikan (dalam SBOM) yang diverifikasi dan dipercaya oleh penerbit gambar. Pengesahan ini dapat dilampirkan ke gambar kontainer dan didorong ke repositori.

Pada bagian selanjutnya kita akan melihat bagaimana menggunakan artefak yang dibuktikan untuk audit dan verifikasi pengontrol penerimaan.

Verifikasi integritas gambar menggunakan pengontrol penerimaan Kubernetes

Kami dapat memverifikasi tanda tangan gambar, artefak yang dibuktikan secara otomatis sebelum menerapkan gambar untuk menargetkan klaster Kubernetes menggunakan pengontrol penerimaan dinamis dan mengakui penerapan hanya jika metadata keamanan artefak mematuhi kebijakan pengontrol penerimaan.

Misalnya kita dapat menulis kebijakan yang secara kriptografis memverifikasi tanda tangan gambar, SBOM yang dibuktikan, laporan pipeline run yang dibuktikan, atau laporan pemindaian CVE yang dibuktikan. Kami dapat menulis kondisi dalam kebijakan untuk memeriksa data dalam laporan, misalnya pemindaian CVE seharusnya tidak kritisCVEs. Penerapan hanya diperbolehkan untuk gambar yang memenuhi kondisi ini dan semua penerapan lainnya akan ditolak oleh pengontrol penerimaan.

Contoh pengontrol penerimaan meliputi:

- Kyverno
- Penjaga Gerbang OPA
- Portieris
- Ratifikasi
- Kritis
- Tutorial Grafea
- Voucher

Perbarui paket dalam gambar kontainer Anda

Anda harus menyertakan RUN `apt-get update && apt-get upgrade` di Dockerfiles Anda untuk memutakhirkan paket dalam gambar Anda. Meskipun pemutakhiran mengharuskan Anda untuk menjalankan sebagai root, ini terjadi selama fase pembuatan gambar. Aplikasi tidak perlu dijalankan sebagai root. Anda dapat menginstal pembaruan dan kemudian beralih ke pengguna lain dengan arahan USER. Jika gambar dasar Anda berjalan sebagai pengguna non-root, beralihlah ke root dan back; jangan hanya mengandalkan pengelola gambar dasar untuk menginstal pembaruan keamanan terbaru.

Jalankan `apt-get clean` untuk menghapus file installer dari`/var/cache/apt/archives/`. Anda juga dapat menjalankan `rm -rf /var/lib/apt/lists/*` setelah menginstal paket. Ini menghapus file indeks atau daftar paket yang tersedia untuk diinstal. Ketahuilah bahwa perintah ini mungkin berbeda untuk setiap manajer paket. Contoh:

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

Alat dan sumber daya

- Lokakarya Perendaman Keamanan Amazon EKS - Keamanan Gambar
- docker-slim Bangun gambar minimal yang aman
- dockle Memverifikasi bahwa Dockerfile Anda selaras dengan praktik terbaik untuk membuat gambar aman
- linter berbasis Aturan dockerfile-lint untuk Dockerfiles
- hadolint Sebuah linter dockerfile yang cerdas
- Gatekeeper dan OPA A pengontrol penerimaan berbasis kebijakan
- Kyverno Mesin kebijakan asli Kubernetes
- in-toto Memungkinkan pengguna untuk memverifikasi apakah langkah dalam rantai pasokan dimaksudkan untuk dilakukan, dan apakah langkah itu dilakukan oleh aktor yang tepat
- Notaris Sebuah proyek untuk menandatangani gambar kontainer
- Notaris v2

- Grafeas API metadata artefak terbuka untuk mengaudit dan mengatur rantai pasokan perangkat lunak Anda
- NeuVector oleh SUSE open source, platform keamanan wadah zero-trust, menyediakan pemindaian kontainer, gambar, dan registri untuk kerentanan, rahasia, dan kepatuhan.

Strategi Multi Akun

AWS merekomendasikan penggunaan strategi multi akun dan organisasi AWS untuk membantu mengisolasi dan mengelola aplikasi dan data bisnis Anda. Ada banyak manfaat menggunakan strategi multi akun:

- Peningkatan kuota layanan AWS API. Kuota diterapkan ke akun AWS, dan menggunakan beberapa akun untuk beban kerja Anda meningkatkan keseluruhan kuota yang tersedia untuk beban kerja Anda.
- Kebijakan Identity and Access Management (IAM) yang lebih sederhana. Memberikan beban kerja dan operator yang mendukung mereka akses hanya ke akun AWS mereka sendiri berarti lebih sedikit waktu menyusun kebijakan IAM berbutir halus untuk mencapai prinsip hak istimewa yang paling rendah.
- Peningkatan Isolasi sumber daya AWS. Secara desain, semua sumber daya yang disediakan dalam akun secara logis diisolasi dari sumber daya yang disediakan di akun lain. Batas isolasi ini memberi Anda cara untuk membatasi risiko masalah terkait aplikasi, kesalahan konfigurasi, atau tindakan jahat. Jika masalah terjadi dalam satu akun, dampak terhadap beban kerja yang terdapat di akun lain dapat dikurangi atau dihilangkan.
- Lebih banyak manfaat, seperti yang dijelaskan dalam Whitepaper Strategi Multi Akun AWS

Bagian berikut akan menjelaskan cara menerapkan strategi multi akun untuk beban kerja EKS Anda menggunakan pendekatan kluster EKS terpusat atau terpusat.

Merencanakan Strategi Akun Multi Beban Kerja untuk Multi Tenant Cluster

Dalam strategi AWS multi akun, sumber daya yang termasuk dalam beban kerja tertentu seperti bucket S3, ElastiCache cluster, dan DynamoDB Tables semuanya dibuat di akun AWS yang berisi semua sumber daya untuk beban kerja tersebut. Ini disebut sebagai akun beban kerja, dan kluster EKS dikerahkan ke akun yang disebut sebagai akun cluster. Akun cluster akan dieksplorasi di bagian selanjutnya. Menerapkan sumber daya ke akun beban kerja khusus mirip dengan menerapkan sumber daya kubernetes ke dalam namespace khusus.

Akun beban kerja kemudian dapat dipecah lebih lanjut berdasarkan siklus hidup pengembangan perangkat lunak atau persyaratan lain jika sesuai. Misalnya beban kerja tertentu dapat memiliki akun produksi, akun pengembangan, atau akun untuk hosting instance beban kerja tersebut di wilayah tertentu. Informasi lebih lanjut tersedia di whitepaper AWS ini.

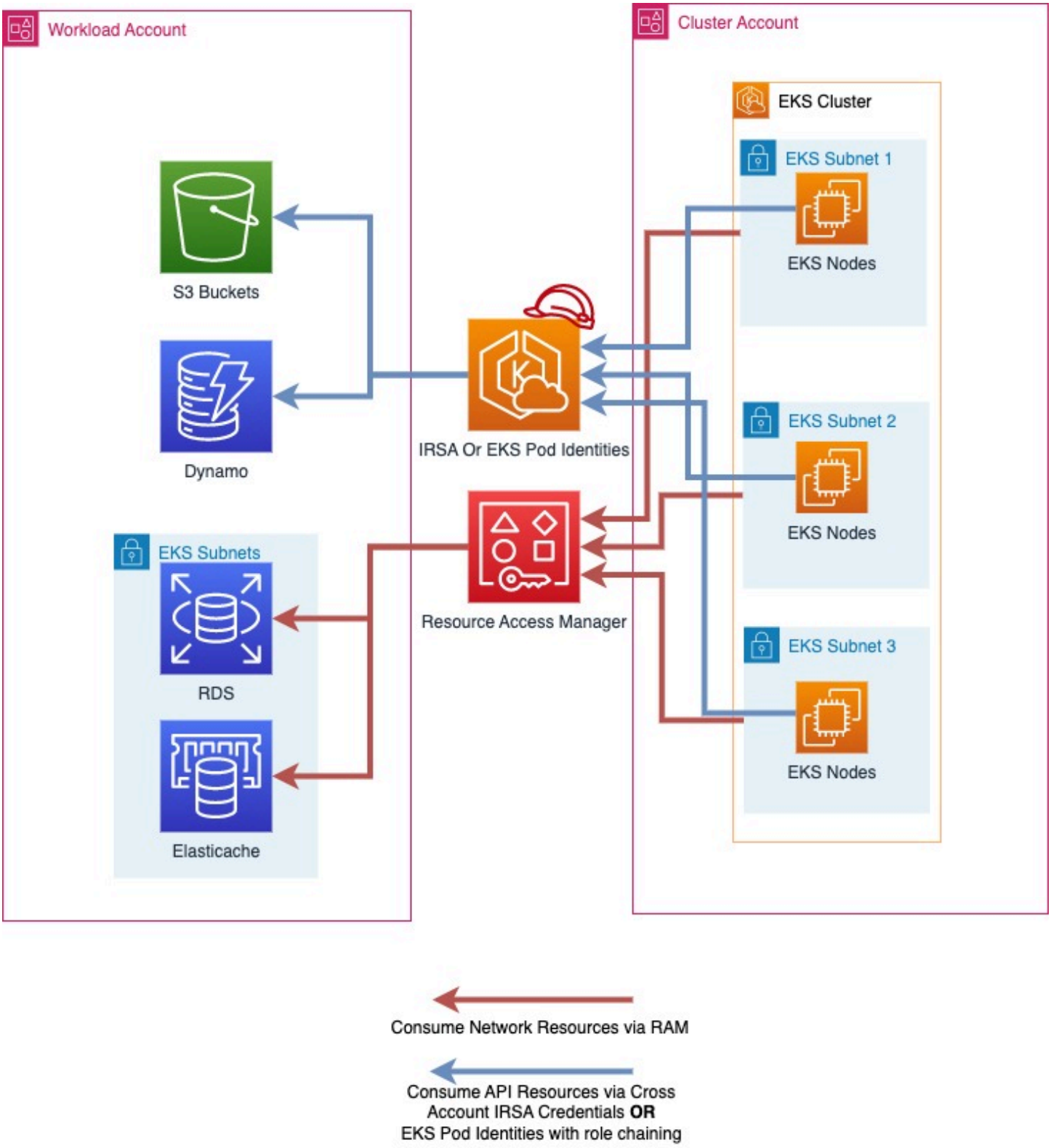
Anda dapat mengadopsi pendekatan berikut saat menerapkan strategi Multi akun EKS:

Kluster EKS Terpusat

Dalam pendekatan ini, Kluster EKS Anda akan digunakan dalam satu akun AWS yang disebut. `Cluster Account` Menggunakan peran IAM untuk Akun Layanan (IRSA) atau EKS Pod Identities untuk memberikan kredensyal AWS sementara dan AWS Resource Access Manager (RAM) untuk menyederhanakan akses jaringan, Anda dapat mengadopsi strategi multi akun untuk klaster EKS multi tenant Anda. Akun cluster akan berisi VPC, subnet, kluster EKS, sumber daya komputasi EC2/Fargate (node pekerja), dan konfigurasi jaringan tambahan apa pun yang diperlukan untuk menjalankan kluster EKS Anda.

Dalam strategi akun multi beban kerja untuk klaster multi tenant, akun AWS biasanya selaras dengan ruang nama kubernetes sebagai mekanisme untuk mengisolasi grup sumber daya. Praktik terbaik untuk isolasi penyewa dalam cluster EKS harus tetap diikuti saat menerapkan strategi multi akun untuk klaster EKS multi penyewa.

Dimungkinkan untuk memiliki beberapa `Cluster Accounts` di organisasi AWS Anda, dan merupakan praktik terbaik untuk memiliki beberapa `Cluster Accounts` yang selaras dengan kebutuhan siklus hidup pengembangan perangkat lunak Anda. Untuk beban kerja yang beroperasi pada skala yang sangat besar, Anda mungkin memerlukan beberapa `Cluster Accounts` untuk memastikan bahwa ada cukup kubernetes dan kuota layanan AWS yang tersedia untuk semua beban kerja Anda.



|Pada diagram di atas, AWS RAM digunakan untuk berbagi subnet dari akun cluster ke akun beban kerja. Kemudian beban kerja yang berjalan di pod EKS menggunakan IRSA atau EKS Pod Identities

dan role chaining untuk mengambil peran dalam akun beban kerja mereka dan mengakses sumber daya AWS mereka.

Menerapkan Strategi Akun Multi Beban Kerja untuk Multi Tenant Cluster

Berbagi Subnet Dengan AWS Resource Access Manager

AWS Resource Access Manager (RAM) memungkinkan Anda berbagi sumber daya di seluruh akun AWS.

Jika RAM diaktifkan untuk AWS Organization, Anda dapat membagikan Subnet VPC dari akun Cluster ke akun beban kerja Anda. Ini akan memungkinkan sumber daya AWS yang dimiliki oleh akun beban kerja Anda, seperti Amazon ElastiCache Clusters atau Amazon Relational Database Service (RDS) Database Service (RDS) untuk diterapkan ke VPC yang sama dengan kluster EKS Anda, dan dapat dikonsumsi oleh beban kerja yang berjalan di kluster EKS Anda.

Untuk berbagi sumber daya melalui RAM, buka RAM di konsol AWS akun cluster dan pilih “Berbagi Sumber Daya” dan “Buat Berbagi Sumber Daya”. Beri nama Resource Share Anda dan Pilih subnet yang ingin Anda bagikan. Pilih Berikutnya lagi dan masukkan akun 12 digit IDs untuk akun beban kerja yang ingin Anda bagikan subnet, pilih berikutnya lagi, dan klik Buat berbagi sumber daya untuk menyelesaikannya. Setelah langkah ini, akun beban kerja dapat menyebarkan sumber daya ke subnet tersebut.

Berbagi RAM juga dapat dibuat secara terprogram, atau dengan infrastruktur sebagai kode.

Memilih Antara Identitas EKS Pod dan IRSA

Di re:invent 2023, AWS meluncurkan EKS Pod Identities sebagai cara yang lebih sederhana untuk mengirimkan kredensil AWS sementara ke pod Anda di EKS. Identitas Pod IRSA dan EKS adalah metode yang valid untuk mengirimkan kredensil AWS sementara ke pod EKS Anda dan akan terus didukung. Anda harus mempertimbangkan metode pengiriman mana yang paling sesuai dengan kebutuhan Anda.

Saat bekerja dengan kluster EKS dan beberapa akun AWS, IRSA dapat langsung mengambil peran di akun AWS selain akun tempat cluster EKS di-host secara langsung, sementara identitas EKS Pod mengharuskan Anda untuk mengonfigurasi rantai peran. Lihat dokumentasi EKS untuk perbandingan mendalam.

Mengakses Sumber Daya AWS API dengan Peran IAM Untuk Akun Layanan

Peran IAM untuk Akun Layanan (IRSA) memungkinkan Anda mengirimkan kredensyal AWS sementara ke beban kerja Anda yang berjalan di EKS. IRSA dapat digunakan untuk mendapatkan kredensyal sementara untuk peran IAM di akun beban kerja dari akun cluster. Hal ini memungkinkan beban kerja Anda berjalan di kluster EKS Anda di akun klaster untuk menggunakan sumber daya AWS API, seperti bucket S3 yang dihosting di akun beban kerja, dan menggunakan autentikasi IAM untuk sumber daya seperti Amazon RDS Databases atau Amazon EFS. FileSystems

Sumber daya AWS API dan Sumber Daya lainnya yang menggunakan autentikasi IAM di akun beban kerja hanya dapat diakses oleh kredensyal untuk peran IAM di akun beban kerja yang sama, kecuali jika akses lintas akun mampu dan telah diaktifkan secara eksplisit.

Mengaktifkan IRSA untuk akses lintas akun

Untuk mengaktifkan IRSA untuk beban kerja di Akun Cluster Anda untuk mengakses sumber daya di akun Beban Kerja, Anda harus terlebih dahulu membuat penyedia identitas IAM OIDC di akun beban kerja Anda. Ini dapat dilakukan dengan prosedur yang sama untuk mengatur IRSA, kecuali Penyedia Identitas akan dibuat di akun beban kerja.

Kemudian saat mengonfigurasi IRSA untuk beban kerja Anda di EKS, Anda dapat mengikuti langkah yang sama seperti dokumentasi, tetapi gunakan id akun 12 digit dari akun beban kerja seperti yang disebutkan di bagian “Contoh Buat penyedia identitas dari cluster akun lain”.

Setelah ini dikonfigurasi, aplikasi Anda yang berjalan di EKS akan dapat langsung menggunakan akun layanannya untuk mengambil peran dalam akun beban kerja, dan menggunakan sumber daya di dalamnya.

Mengakses Sumber Daya AWS API dengan Identitas Pod EKS

EKS Pod Identities adalah cara baru untuk mengirimkan kredensyal AWS ke beban kerja Anda yang berjalan di EKS. Identitas pod EKS menyederhanakan konfigurasi sumber daya AWS karena Anda tidak perlu lagi mengelola konfigurasi OIDC untuk mengirimkan kredensil AWS ke pod Anda di EKS.

Mengaktifkan Identitas Pod EKS untuk akses lintas akun

Tidak seperti IRSA, EKS Pod Identities hanya dapat digunakan untuk secara langsung memberikan akses ke peran dalam akun yang sama dengan cluster EKS. Untuk mengakses peran di akun AWS lain, pod yang menggunakan Identitas Pod EKS harus melakukan Role Chaining.

Role chaining dapat dikonfigurasi dalam profil aplikasi dengan file konfigurasi awsnya menggunakan Penyedia Kredensial Proses yang tersedia di berbagai AWS. SDKs `credential_process`dapat digunakan sebagai sumber kredensi saat mengonfigurasi profil, seperti:

```
# 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
```

Sumber skrip yang disebut oleh 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}'
```

Anda dapat membuat file konfigurasi aws seperti yang ditunjukkan di atas dengan peran Akun A dan B dan menentukan vars AWS_CONFIG_FILE dan AWS_PROFILE env dalam spesifikasi pod Anda. EKS Pod identity webhook tidak diganti jika env vars sudah ada dalam spesifikasi 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
```

Saat mengonfigurasi kebijakan kepercayaan peran untuk rantai peran dengan identitas pod EKS, Anda dapat mereferensikan atribut khusus EKS sebagai tag sesi dan menggunakan kontrol akses berbasis atribut (ABAC) untuk membatasi akses ke peran IAM Anda hanya pada sesi identitas EKS Pod tertentu, seperti Akun Layanan Kubernetes yang dimiliki pod.

Harap dicatat bahwa beberapa atribut ini mungkin tidak unik secara universal, misalnya dua kluster EKS mungkin memiliki ruang nama yang identik, dan satu cluster mungkin memiliki akun layanan bernama identik di seluruh ruang nama. Jadi ketika memberikan akses melalui EKS Pod Identities dan ABAC, itu adalah praktik terbaik untuk selalu mempertimbangkan cluster arn dan namespace saat memberikan akses ke akun layanan.

Identitas ABAC dan EKS Pod untuk akses lintas akun

Saat menggunakan EKS Pod Identities untuk mengambil peran (role chaining) di akun lain sebagai bagian dari strategi multi akun, Anda memiliki opsi untuk menetapkan peran IAM unik untuk setiap akun layanan yang perlu mengakses akun lain, atau menggunakan peran IAM umum di beberapa akun layanan dan menggunakan ABAC untuk mengontrol akun apa yang dapat diakses.

Untuk menggunakan ABAC untuk mengontrol akun layanan apa yang dapat mengambil peran ke akun lain dengan rantai peran, Anda membuat pernyataan kebijakan kepercayaan peran yang hanya memungkinkan peran diasumsikan oleh sesi peran ketika nilai yang diharapkan ada. Kebijakan kepercayaan peran berikut hanya akan membiarkan peran dari akun klaster EKS (ID akun 111122223333) mengambil peran jika`kubernetes-service-account`, `eks-cluster-arn` dan `kubernetes-namespace` tag semuanya memiliki nilai yang diharapkan.

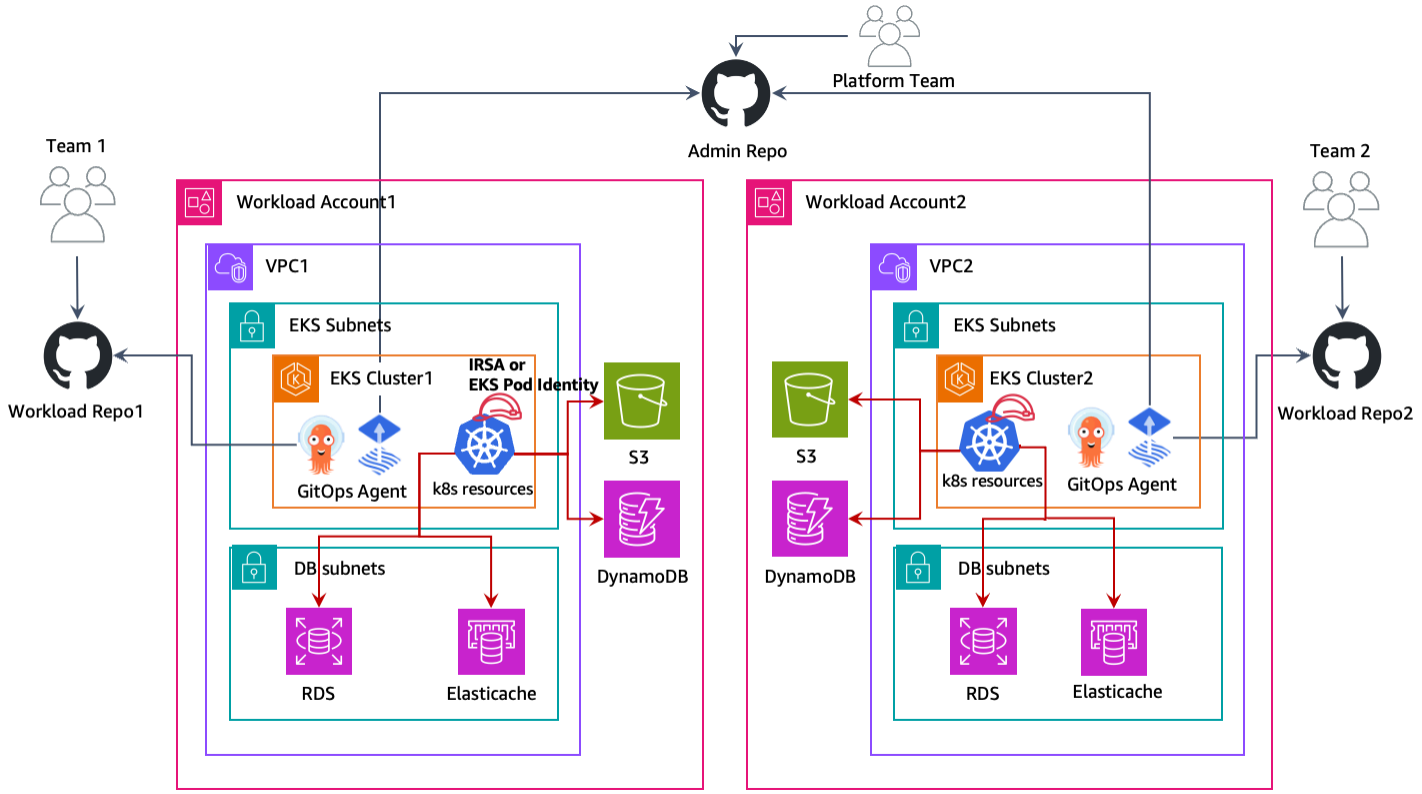
```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "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"
                }
            }
        }
    ]
}
```

Saat menggunakan strategi ini, merupakan praktik terbaik untuk memastikan bahwa peran IAM umum hanya memiliki `sts:AssumeRole` izin dan tidak ada akses AWS lainnya.

Penting saat menggunakan ABAC bahwa Anda mengontrol siapa yang memiliki kemampuan untuk menandai peran IAM dan pengguna hanya kepada mereka yang memiliki kebutuhan ketat untuk melakukannya. Seseorang dengan kemampuan untuk menandai peran IAM atau pengguna akan dapat mengatur tag yang roles/users identik dengan apa yang akan ditetapkan oleh EKS Pod Identities dan mungkin dapat meningkatkan hak istimewa mereka. Anda dapat membatasi siapa yang memiliki akses untuk menyetel tag `kubernetes-` dan `eks-` tag pada peran IAM dan pengguna menggunakan kebijakan IAM, atau Kebijakan Kontrol Layanan (SCP).

Cluster EKS yang tidak terpusat

Dalam pendekatan ini, kluster EKS diterapkan ke Akun AWS beban kerja masing-masing dan hidup bersama dengan sumber daya AWS lainnya seperti bucket Amazon S3 VPCs, tabel Amazon DynamoDB, dll., Setiap akun beban kerja independen, mandiri, dan dioperasikan oleh cluster Unit/Application teams. This model allows the creation of reusable blueprints for various cluster capabilities — AI/ML Bisnis masing-masing, Pemrosesan Batch, Tujuan umum, dll, — dan menjual klaster berdasarkan persyaratan tim aplikasi. Tim aplikasi dan platform beroperasi dari GitOpsrepositori masing-masing untuk mengelola penerapan ke cluster beban kerja.



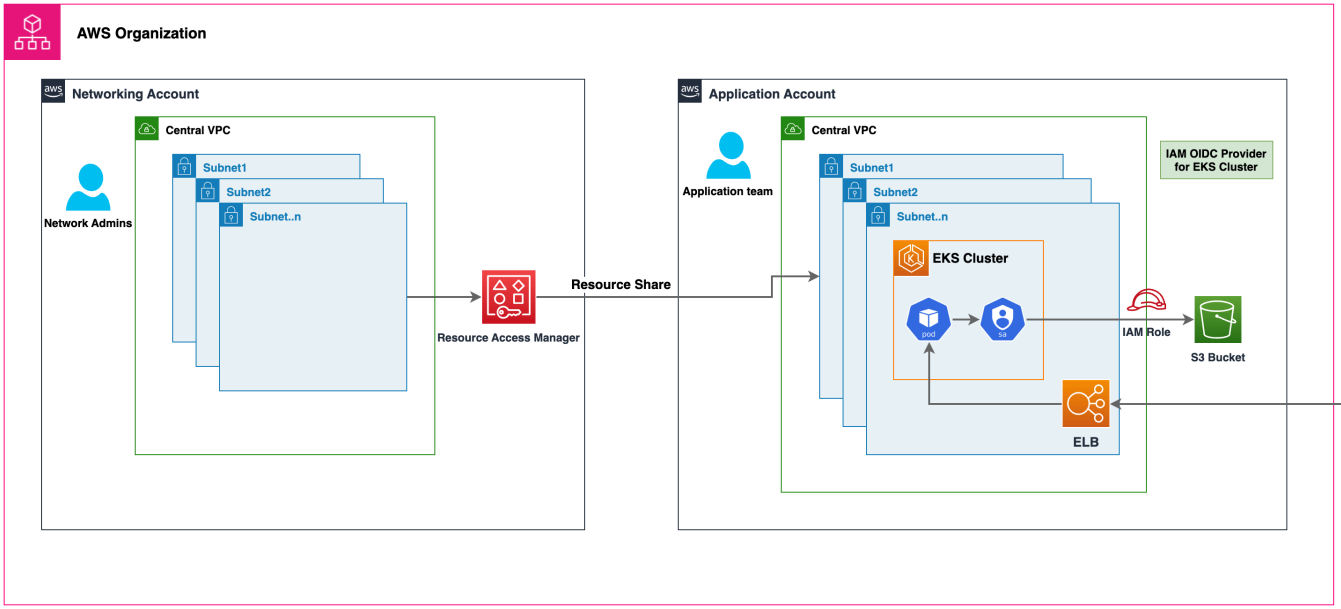
Dalam diagram di atas, kluster Amazon EKS dan sumber daya AWS lainnya diterapkan ke akun beban kerja masing-masing. Kemudian beban kerja yang berjalan di pod EKS menggunakan IRSA atau EKS Pod Identities untuk mengakses sumber daya AWS mereka.

GitOps adalah cara mengelola penerapan aplikasi dan infrastruktur sehingga seluruh sistem dijelaskan secara deklaratif dalam repositori Git. Ini adalah model operasional yang menawarkan Anda kemampuan untuk mengelola status beberapa klaster Kubernetes menggunakan praktik terbaik kontrol versi, artefak yang tidak dapat diubah, dan otomatisasi. Dalam model multi cluster ini, setiap klaster beban kerja di-bootstrap dengan beberapa repo Git, memungkinkan setiap tim (aplikasi, platform, keamanan, dll.,) untuk menerapkan perubahan masing-masing pada cluster.

Anda akan menggunakan peran IAM untuk Akun Layanan (IRSA) atau Identitas Pod EKS di setiap akun untuk memungkinkan beban kerja EKS Anda mendapatkan kredensyal aws sementara untuk mengakses sumber daya AWS lainnya dengan aman. Peran IAM dibuat di Akun AWS beban kerja masing-masing dan memetakannya ke akun layanan k8s untuk menyediakan akses IAM sementara. Jadi, tidak diperlukan akses lintas akun dalam pendekatan ini. Ikuti peran IAM untuk dokumentasi Akun Layanan tentang cara penyiapan di setiap beban kerja untuk IRSA, dan dokumentasi EKS Pod Identities tentang cara mengatur identitas pod EKS di setiap akun.

Jaringan Terpusat

Anda juga dapat menggunakan AWS RAM untuk membagikan Subnet VPC ke akun beban kerja dan meluncurkan kluster Amazon EKS dan sumber daya AWS lainnya di dalamnya. Ini memungkinkan manajemen/administrasi jaringan terpusat, konektivitas jaringan yang disederhanakan, dan kluster EKS yang tidak terpusat. Lihat blog AWS ini untuk panduan dan pertimbangan terperinci tentang pendekatan ini.



Dalam diagram di atas, AWS RAM digunakan untuk berbagi subnet dari akun jaringan pusat ke akun beban kerja. Kemudian kluster EKS dan sumber daya AWS lainnya diluncurkan di subnet tersebut di masing-masing akun beban kerja. Pod EKS menggunakan IRSA atau EKS Pod Identities untuk mengakses sumber daya AWS mereka.

Cluster EKS terpusat vs de-sentralisasi

Keputusan untuk menjalankan dengan Sentralisasi atau De-sentralisasi akan tergantung pada kebutuhan Anda. Tabel ini menunjukkan perbedaan utama dengan masing-masing strategi.

#	Kluster EKS terpusat	Cluster EKS yang tidak terpusat
Manajemen Cluster:	Mengelola satu kluster EKS lebih mudah daripada mengelola beberapa cluster	Otomatisasi manajemen cluster yang efisien diperluka n untuk mengurangi overhead operasional pengelolaan beberapa kluster EKS
Efisiensi Biaya:	Memungkinkan penggunaa n kembali cluster EKS dan sumber daya jaringan, yang	Membutuhkan pengaturan jaringan dan cluster per beban kerja, yang membutuhkan sumber daya tambahan

#	Kluster EKS terpusat	Cluster EKS yang tidak terpusat
	mempromosikan efisiensi biaya	
Ketahanan:	Beberapa beban kerja pada cluster terpusat dapat terpengaruh jika klaster menjadi terganggu	Jika cluster menjadi terganggu, kerusakan terbatas hanya pada beban kerja yang berjalan di cluster itu. Semua beban kerja lainnya tidak terpengaruh
Isolasi & Keamanan:	Isolasi/Soft Multi-tenancy dicapai dengan menggunak an konstruksi asli k8s seperti. `Namespaces` Beban kerja dapat berbagi sumber daya yang mendasarinya seperti CPU, memori, dll. Sumber daya AWS diisolasi ke dalam akun beban kerjanya sendiri yang secara default tidak dapat diakses dari akun AWS lainnya.	Isolasi yang lebih kuat pada sumber daya komputasi karena beban kerja berjalan di masing-masing cluster dan node yang tidak berbagi sumber daya apa pun. Sumber daya AWS diisolasi ke dalam akun beban kerjanya sendiri yang secara default tidak dapat diakses dari akun AWS lainnya.
Kinerja & Skalabitas:	Saat beban kerja bertambah menjadi skala yang sangat besar, Anda mungkin menemukan kubernetes dan kuota layanan AWS di akun klaster. Anda dapat menerapkan akun klaster tambahan untuk menskalakan lebih jauh	Semakin banyak cluster dan VPCs hadir, setiap beban kerja memiliki lebih banyak kuota layanan k8 dan AWS yang tersedia

#	Kluster EKS terpusat	Cluster EKS yang tidak terpusat
Jaringan:	VPC tunggal digunakan per cluster, memungkin kan konektivitas yang lebih sederhana untuk aplikasi di cluster itu	Perutean harus dibuat antara kluster EKS yang tidak terpusat VPCs
Manajemen Akses Kubernetes:	Perlu mempertahankan banyak peran dan pengguna yang berbeda di cluster untuk menyediakan akses ke semua tim beban kerja dan memastikan sumber daya kubernetes dipisahkan dengan benar	Manajemen akses yang disederhanakan karena setiap cluster didedikasikan untuk beban kerja/tim
Manajemen Akses AWS:	Sumber daya AWS diterapkan ke akun mereka sendiri yang hanya dapat diakses secara default dengan peran IAM di akun beban kerja. Peran IAM dalam akun beban kerja diasumsikan lintas akun baik dengan IRSA atau EKS Pod Identities.	Sumber daya AWS diterapka n ke akun mereka sendiri yang hanya dapat diakses secara default dengan peran IAM di akun beban kerja. Peran IAM dalam akun beban kerja dikirim langsung ke pod dengan IRSA atau EKS Pod Identities

Manajemen akses cluster

Manajemen akses yang efektif sangat penting untuk menjaga keamanan dan integritas kluster Amazon EKS Anda. Panduan ini membahas berbagai opsi untuk manajemen akses EKS, dengan fokus pada penggunaan AWS IAM Identity Center (sebelumnya AWS SSO). Kami akan membandingkan pendekatan yang berbeda, mendiskusikan trade-off mereka, dan menyoroti batasan dan pertimbangan yang diketahui.

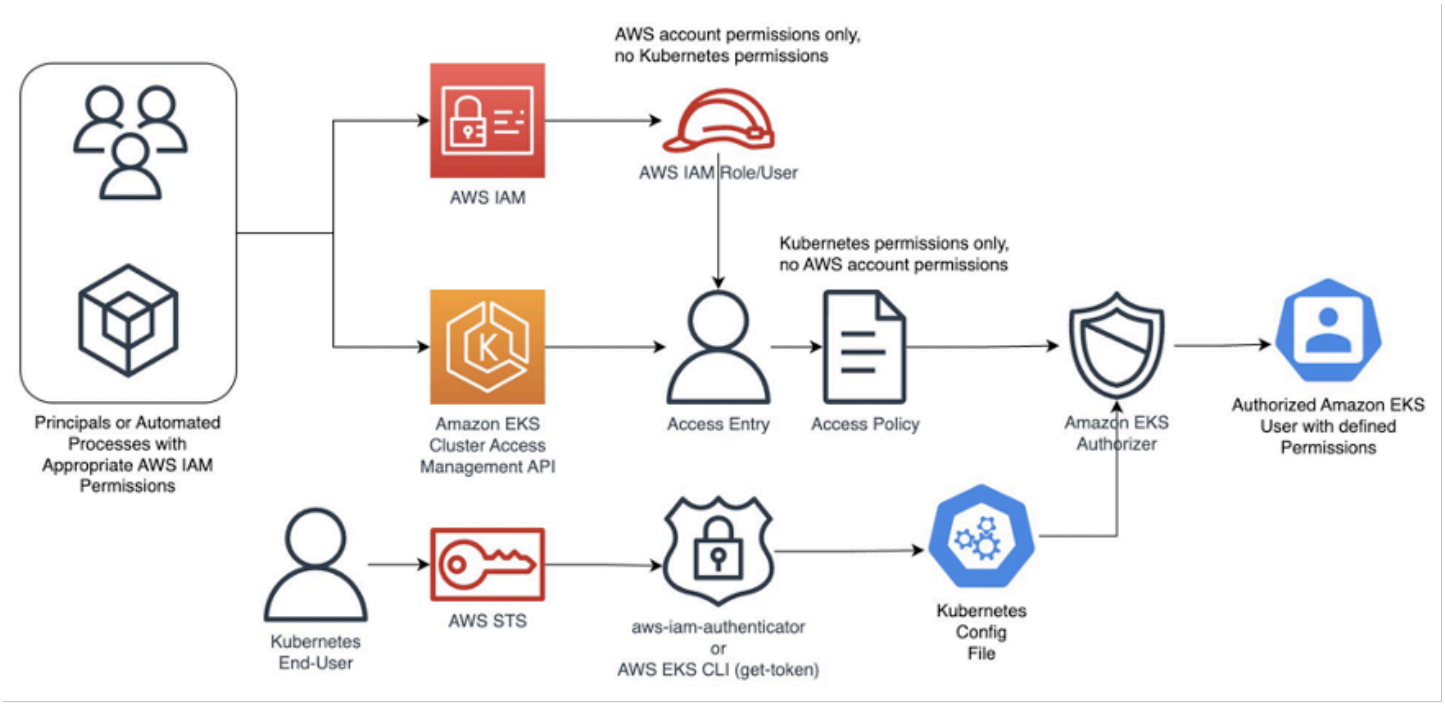
Opsi manajemen akses EKS

> **Note**
>
> ConfigMapmanajemen akses berbasis (aws-auth ConfigMap) tidak digunakan lagi dan digantikan oleh Cluster Access Management (CAM) API. Untuk kluster EKS baru, terapkan CAM API untuk mengelola akses cluster. Untuk cluster yang ada menggunakan aws-auth ConfigMap, migrasi ke menggunakan CAM API.

Opsi 1: Pusat Identitas AWS IAM dengan API Manajemen Akses Cluster (CAM)

- Pengguna terpusat dan manajemen izin
- Integrasi dengan penyedia identitas yang ada (misalnya Microsoft AD, Okta, PingId dan lainnya)
- CAM API menggunakan Entri Akses untuk menautkan prinsipal AWS IAM (pengguna atau peran) ke kluster EKS. Entri ini bekerja dengan identitas terkelola IAM Identity Center, memungkinkan administrator untuk mengontrol akses klaster untuk pengguna dan grup yang ditentukan di Pusat Identitas.

Aliran otentikasi kluster EKS:



1. Prinsipal (pengguna manusia) atau proses otomatis mengautentikasi melalui AWS IAM dengan menghadirkan izin akun AWS yang sesuai. Pada langkah ini, mereka dipetakan ke prinsipal AWS IAM yang sesuai (peran atau pengguna).
2. Selanjutnya, entri akses EKS memetakan prinsipal IAM ini ke prinsipal Kubernetes RBAC (pengguna atau grup) dengan mendefinisikan kebijakan akses yang sesuai, yang hanya berisi izin Kubernetes.
3. Ketika pengguna akhir Kubernetes mencoba mengakses klaster, permintaan otentikasi diproses oleh aws-iam-authenticator AWS EKS CLI dan divalidasi terhadap konteks klaster dalam file kubeconfig.
4. Akhirnya, otorisasi EKS memverifikasi izin yang terkait dengan entri akses pengguna yang diautentikasi dan memberikan atau menolak akses yang sesuai.
    - API menggunakan Kebijakan Akses Khusus Amazon EKS untuk menentukan tingkat otorisasi untuk setiap Entri Akses. Kebijakan ini dapat dipetakan ke peran dan izin yang disiapkan di Pusat Identitas IAM, memastikan kontrol akses yang konsisten di seluruh layanan AWS dan kluster EKS.

Manfaat dibandingkan manajemen akses ConfigMap berbasis:

1. Mengurangi risiko kesalahan konfigurasi: Manajemen berbasis API langsung menghilangkan kesalahan umum yang terkait dengan pengeditan manual. ConfigMap Ini membantu mencegah penghapusan yang tidak disengaja atau kesalahan sintaks yang dapat mengunci pengguna keluar dari cluster.
2. Prinsip hak istimewa terkecil yang disempurnakan: Menghapus kebutuhan akan izin cluster-admin dari identitas pembuat klaster dan memungkinkan penetapan izin yang lebih terperinci dan sesuai. Anda dapat memilih untuk menambahkan izin ini untuk kasus penggunaan break-glass.
3. Model keamanan yang disempurnakan: Menyediakan validasi bawaan entri akses sebelum diterapkan. Selain itu, menawarkan integrasi yang lebih ketat dengan AWS IAM untuk otentikasi.
4. Operasi yang disederhanakan: Menawarkan cara yang lebih intuitif untuk mengelola izin melalui perkakas asli AWS.

Praktik terbaik:

1. Gunakan AWS Organizations untuk mengelola beberapa akun dan menerapkan kebijakan kontrol layanan (SCPs).
2. Terapkan prinsip hak istimewa terkecil dengan membuat set izin khusus untuk peran EKS yang berbeda (misalnya admin, pengembang, hanya-baca).
3. Memanfaatkan Attribute-based Access Control (ABAC) untuk secara dinamis menetapkan izin ke Pod berdasarkan atribut pengguna.
4. Secara teratur mengaudit dan meninjau izin akses.

Pertimbangan/keterbatasan:

1. Peran ARNs yang dihasilkan oleh Identity Center memiliki sufiks acak, membuatnya sulit untuk digunakan dalam konfigurasi statis.
2. Dukungan terbatas untuk izin berbutir halus di tingkat sumber daya Kubernetes. Konfigurasi tambahan diperlukan untuk peran Kubernetes RBAC kustom. Bersama dengan Kubernetes-native RBAC, pertimbangkan untuk menggunakan Kyverno untuk manajemen izin lanjutan di kluster EKS.

Opsi 2: AWS IAM Users/Roles dipetakan ke grup Kubernetes

Kelebihan:

1. Kontrol halus atas izin IAM.
2. Peran yang dapat diprediksi dan statis ARNs

Kontra:

1. Peningkatan overhead manajemen untuk akun pengguna
2. Kurangnya manajemen identitas terpusat
3. Potensi proliferasi entitas IAM

Praktik terbaik:

1. Gunakan peran IAM alih-alih pengguna IAM untuk meningkatkan keamanan dan pengelolaan
2. Menerapkan konvensi penamaan untuk peran untuk memastikan konsistensi dan kemudahan manajemen
3. Memanfaatkan kondisi kebijakan IAM untuk membatasi akses berdasarkan tag atau atribut lainnya.
4. Putar tombol akses secara teratur dan tinjau izin.

Pertimbangan/keterbatasan:

1. Masalah skalabilitas saat mengelola sejumlah besar pengguna atau peran
2. Tidak ada kemampuan masuk tunggal bawaan

Opsi 3: Penyedia OIDC

Kelebihan:

1. Integrasi dengan sistem manajemen identitas yang ada
2. Mengurangi overhead manajemen untuk akun pengguna

Kontra:

1. Kompleksitas konfigurasi tambahan
2. Potensi peningkatan latensi selama otentikasi
3. Ketergantungan pada penyedia identitas eksternal

Praktik Terbaik:

1. Konfigurasikan penyedia OIDC dengan hati-hati untuk memastikan validasi token yang aman.
2. Gunakan token berumur pendek dan terapkan mekanisme penyegaran token.
3. Secara teratur mengaudit dan memperbarui konfigurasi OIDC.

Tinjau panduan ini untuk implementasi referensi mengintegrasikan penyedia Single Sign-On eksternal dengan Amazon EKS

Pertimbangan/keterbatasan:

1. Integrasi native terbatas dengan layanan AWS dibandingkan dengan IAM.
2. URL penerbit penyedia OIDC harus dapat diakses publik agar EKS dapat menemukan kunci penandatanganan.

AWS EKS Pod Identity vs IRSA untuk beban kerja

Amazon EKS menyediakan dua cara untuk memberikan izin AWS IAM untuk beban kerja yang berjalan di klaster Amazon EKS: peran IAM untuk akun layanan (IRSA), dan Identitas Pod EKS.

Meskipun IRSA dan EKS Pod Identities memberikan manfaat dari akses hak istimewa yang paling sedikit, isolasi kredensyal, dan auditabilitas, EKS Pod Identity adalah cara yang disarankan untuk memberikan izin untuk beban kerja.

Untuk panduan rinci tentang Identitas dan kredensyal untuk pod EKS, silakan merujuk ke bagian Identitas dan Kredensial dari praktik terbaik Keamanan.

Rekomendasi

Kombinasikan IAM Identity Center dengan CAM API

- Manajemen yang disederhanakan: Dengan menggunakan API Manajemen Akses Cluster bersama dengan IAM Identity Center, administrator dapat mengelola akses kluster EKS bersama layanan AWS lainnya, mengurangi kebutuhan untuk beralih di antara antarmuka yang berbeda atau mengedit secara manual. ConfigMaps
- Gunakan entri akses untuk mengelola izin Kubernetes dari prinsipal IAM dari luar klaster. Anda dapat menambahkan dan mengelola akses ke cluster dengan menggunakan EKS API, AWS Command Line Interface, AWS, AWS SDKs CloudFormation, dan AWS Management Console. Ini berarti Anda dapat mengelola pengguna dengan alat yang sama dengan yang Anda buat dengan cluster.
- Izin Kubernetes granular dapat diterapkan dengan memetakan pengguna atau grup Kubernetes dengan prinsipal IAM yang terkait dengan identitas SSO melalui entri akses dan kebijakan akses.
- Untuk memulai, ikuti Ubah mode autentikasi untuk menggunakan entri akses, lalu Memigrasi entri ConfigMap aws-auth yang ada untuk mengakses entri.

Praktik Terbaik untuk Penskalaan Otomatis Cluster

Panduan ini memberikan saran tentang Cluster Autoscaling, termasuk panduan untuk Auto Mode, Karpenter dan Kubernetes Cluster Autoscaler.

Topik

- Mode Otomatis EKS
- Karpenter
- Autoscaler klaster

Mode Otomatis EKS

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Amazon EKS Auto Mode mewakili evolusi signifikan dalam manajemen infrastruktur Kubernetes, menggabungkan infrastruktur klaster yang aman dan dapat diskalakan dengan kemampuan Kubernetes terintegrasi yang dikelola oleh AWS. Layanan ini menyediakan operasi node pekerja yang dikelola sepenuhnya, menghilangkan kebutuhan pelanggan untuk menyiapkan Grup atau AutoScaling grup Node Terkelola.

Perbedaan arsitektur utama adalah bahwa EKS Auto Mode menggunakan sistem berbasis Karpenter yang secara otomatis menyediakan instans EC2 sebagai respons terhadap permintaan pod. Instans ini berjalan di Bottlerocket AMIs dengan add-on pra-instal seperti driver EBS CSI, membuat infrastruktur benar-benar dikelola oleh AWS. Berbeda dengan metode penskalaan tradisional:

- Traditional Cluster Autoscaler (CAS) membutuhkan manajemen grup node manual dan hanya dapat membuat node dengan satu tipe instance per grup node
- Karpenter yang dikelola sendiri menawarkan lebih banyak fleksibilitas dengan bekerja dengan EC2 Fleet API dan dapat menyediakan berbagai jenis instans, tetapi memerlukan manajemen pelanggan
- Mode Otomatis EKS menangani semua operasi penskalaan secara otomatis melalui terkelola NodePools dan NodeClasses

Sistem baru ini memperkenalkan beberapa perbaikan operasional:

- Penskalaan berbasis pod otomatis tanpa konfigurasi grup node manual
- Pengontrol penyeimbang beban terkelola bawaan yang dibuat secara otomatis ALB/NLB berdasarkan sumber daya Ingress
- Fitur keamanan terintegrasi dengan identitas Pod yang telah dikonfigurasi sebelumnya
- Runtime node maksimum 21 hari dengan penggantian otomatis

Dari perspektif biaya, Mode Otomatis EKS mempertahankan harga EC2 standar sambil menambahkan biaya manajemen hanya untuk node yang dikelola Mode Otomatis. Yang penting, pelanggan masih dapat mencampur node terkelola Mode Otomatis dengan node yang dikelola sendiri di cluster yang sama.

Meskipun AWS menangani sebagian besar aspek operasional, pelanggan tetap bertanggung jawab atas manajemen versi cluster dan dapat melakukan peningkatan terkontrol yang memicu pembaruan bergulir node pekerja.

Alasan untuk menggunakan Mode Otomatis

Mode Otomatis ditujukan untuk pengguna yang menginginkan manfaat Kubernetes dan EKS tetapi perlu meminimalkan beban operasional di sekitar Kubernetes seperti upgrade dan bagian platform installation/maintenance penting seperti auto-scaling, load balancing, dan storage. Mode Otomatis membawa EKS selangkah lebih maju dalam meminimalkan angkat berat yang tidak berdiferensiasi yang sejalan dengan pemeliharaan Kubernetes

Pertanyaan yang Sering Diajukan

Apa perbedaan antara EKS Auto Mode dan Open Source Karpenter?

EKS Auto Mode adalah serangkaian besar fitur yang membuat menjalankan Kubernetes tingkat produksi menjadi sederhana. Salah satu fitur ini adalah manfaat auto-scaling dari Karpenter, sepenuhnya dikelola. Dari sudut pandang operasi, satu-satunya perbedaan adalah dalam Mode Otomatis EKS Anda tidak perlu mengelola penyebaran, penskalaan, dan peningkatan pod Karpenter itu sendiri. Semua operasi lainnya, seperti dikelola NodeClasses dan NodePools bekerja sama seperti dengan open source Karpenter.

Bisakah saya menjalankan grup node terkelola bersama node yang dikelola Mode Otomatis?

Ya, Anda dapat menjalankan node statis melalui grup node terkelola di samping node penskalaan otomatis Anda yang dilengkapi dengan Mode Otomatis

Bisakah saya memigrasikan cluster dari EKS standar ke Mode Otomatis EKS?

Ya, petunjuk untuk mengaktifkan Mode Otomatis EKS pada klaster yang ada dapat ditemukan di Dokumentasi AWS resmi

Hal-hal yang perlu diperhatikan: 1. Setelah mengaktifkan Mode Otomatis, Anda akan ingin menghapus semua komponen yang telah Anda instal yang sekarang dikelola oleh Mode Otomatis, seperti Karpenter atau AWS Load Balancer Controller 2. Anda perlu memastikan add-on yang Anda instal. up-to-date Lihat dokumentasi.

Bagaimana cara mengkonfigurasi NodePools dalam Mode Otomatis EKS?

Sebuah cluster baru akan datang pra-konfigurasi dengan dua NodePools

tujuan umum

```
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
```

Ini NodePool menginstruksikan Karpenter untuk meluncurkan node dengan karakteristik sebagai berikut:

1. Jenis Kapasitas “Sesuai Permintaan”
2. Jenis contoh C, M, atau R
3. Generasi Instance dari 4
4. Arsitektur AMD
5. OS Linux

Ini juga mendefinisikan apa logika skala ke bawah dengan menyatakan bahwa hanya 10% dari semua node yang mungkin berada dalam keadaan terganggu pada waktu tertentu dan konsolidasi hanya boleh terjadi ketika node kosong atau kurang dimanfaatkan.

sistem

```yaml
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  generation: 1
  labels:
    app.kubernetes.io/managed-by: eks
  name: system
spec:
  disruption:
    budgets:
      - nodes: 10%
    consolidateAfter: 30s
    consolidationPolicy: WhenEmptyOrUnderutilized
  template:
    spec:
      expireAfter: 336h
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values:
            - on-demand
        - key: eks.amazonaws.com/instance-category
          operator: In
          values:
            - c
            - m
            - r
        - key: eks.amazonaws.com/instance-generation
          operator: Gt
          values:
            - '4'
        - key: kubernetes.io/arch
          operator: In
          values:
            - amd64
            - arm64
        - key: kubernetes.io/os
          operator: In
          values:
            - linux
      taints:
        - effect: NoSchedule
```

NodePool Ini mirip dengan “tujuan umum” kecuali untuk perbedaan berikut:

1. Hal ini memungkinkan untuk node dengan arsitektur ARM serta arsitektur AMD
2. Ini menodai node ini dengan NoSchedule kecuali ada toleransi untuk “”CriticalAddonsOnly. Ini untuk penggunaan internal oleh add-on EKS

kebiasaan

Anda dapat membuat kustom Anda sendiri NodePools tergantung pada kebutuhan Anda. Untuk mempelajari lebih lanjut tentang NodePools silakan berkonsultasi dengan Dokumentasi Karpenter.

Dapatkah saya menyesuaikan AMI yang digunakan oleh Mode Otomatis saat node baru diluncurkan?

Tidak, saat ini satu-satunya yang didukung AMIs adalah untuk Bottlerocket yang disediakan Amazon

Bagaimana cara menginstal perkakas atau agen khusus di host Kubernetes saya?

Karena kustomisasi AMI tidak didukung, jika Anda membutuhkan perangkat lunak tingkat host untuk hal-hal seperti pemindaian keamanan, Anda harus menerapkan beban kerja sebagai Kubernetes. DaemonSet

Komponen apa yang berjalan di bidang data klaster saya saat saya menyediakan kluster Mode Otomatis EKS baru?

Jika Anda membuat klaster dengan eksctl atau konsol AWS, satu-satunya pod yang berjalan di kluster Mode Otomatis EKS adalah pod Kubernetes Metrics Server. Komponen lain dari Mode Otomatis EKS seperti Karpenter, AWS Load Balancer Controller, dan EBS CSI Driver semuanya berjalan dan dikelola di luar cluster.

Komponen terkelola apa yang berjalan untuk mendukung kluster Mode Otomatis EKS baru saya?

Mode Otomatis EKS sepenuhnya mengotomatiskan penerapan sebagian besar bagian bidang data yang diperlukan untuk Kubernetes tingkat produksi. Hal ini mencakup:

- Karpenter, untuk auto-scaling komputasi cluster Anda
- AWS Load Balancer Controller untuk memungkinkan Anda mengekspos layanan Kubernetes dengan mudah melalui integrasi Elastic Load Balancer otomatis

- EBS CSI
- VPC CNI
- Agen Identitas Pod EKS

Bagaimana cara memecahkan masalah komponen Mode Otomatis yang dulu dijalankan sebagai pod di cluster saya?

Dengan Mode Otomatis EKS, banyak komponen seperti AWS Load Balancer Controller dan Karpenter dikelola untuk Anda di luar klaster, oleh karena itu Anda tidak akan memiliki visibilitas yang sama ke log yang biasa Anda gunakan saat mengelola sendiri. Jika Anda berada dalam situasi di mana Anda perlu memecahkan masalah fungsionalitas bagian dari fungsionalitas Mode Otomatis, buat Tiket AWS Support.

Karpenter

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Karpenter adalah proyek sumber terbuka yang dirancang untuk meningkatkan manajemen siklus hidup node dalam klaster Kubernetes. Ini mengotomatiskan penyediaan dan deprovisioning node berdasarkan kebutuhan penjadwalan spesifik pod, memungkinkan penskalaan yang efisien dan pengoptimalan biaya. Fungsi utamanya adalah:

- Pantau pod yang tidak dapat dijadwalkan oleh penjadwal Kubernetes karena kendala sumber daya.
- Evaluasi persyaratan penjadwalan (permintaan sumber daya, pemilih node, afinitas, toleransi, dll.) dari Pod yang tidak dapat dijadwalkan.
- Menyediakan node baru yang memenuhi persyaratan pod tersebut.
- Hapus node saat tidak lagi dibutuhkan.

Dengan Karpenter, Anda dapat menentukan NodePools dengan batasan pada penyediaan node seperti taints, label, persyaratan (tipe instance, zona, dll.), Dan batasan total sumber daya yang disediakan. Saat menerapkan beban kerja, Anda dapat menentukan berbagai kendala penjadwalan

dalam spesifikasi pod seperti requests/limits, node selectors, node/pod afinitas sumber daya, toleransi, dan batasan penyebaran topologi. Karpenter kemudian akan menyediakan node berukuran tepat berdasarkan spesifikasi ini.

Alasan menggunakan Karpenter

Sebelum peluncuran Karpenter, pengguna Kubernetes mengandalkan terutama pada grup Amazon EC2 Auto Scaling dan Kubernetes Cluster Autoscaler (CAS) untuk secara dinamis menyesuaikan kapasitas komputasi klaster mereka. Dengan Karpenter, Anda tidak perlu membuat lusinan grup simpul untuk mencapai fleksibilitas dan keragaman yang Anda dapatkan dengan Karpenter. Tidak seperti CAS, Karpenter tidak digabungkan erat dengan versi Kubernetes dan tidak mengharuskan Anda untuk melompat antara AWS dan Kubernetes. APIs

Karpenter mengkonsolidasikan tanggung jawab orkestrasi instance dalam satu sistem, yang lebih sederhana, lebih stabil, dan sadar cluster. Karpenter dirancang untuk mengatasi beberapa tantangan yang disajikan oleh Cluster Autoscaler dengan menyediakan cara yang disederhanakan untuk:

- Node penyediaan berdasarkan persyaratan beban kerja.
- Buat konfigurasi node yang beragam berdasarkan jenis instance, menggunakan NodePool opsi fleksibel. Alih-alih mengelola banyak grup node khusus tertentu, Karpenter dapat membiarkan Anda mengelola kapasitas beban kerja yang beragam dengan satu, fleksibel. NodePool
- Raih penjadwalan pod yang ditingkatkan dalam skala besar dengan meluncurkan node dan menjadwalkan pod dengan cepat.

Untuk informasi dan dokumentasi tentang penggunaan Karpenter, kunjungi situs karpenter.sh.

Rekomendasi

Praktik terbaik dibagi menjadi beberapa bagian tentang Karpenter itu sendiri, NodePools, dan penjadwalan pod.

Praktik terbaik Karpenter

Praktik terbaik berikut mencakup topik yang berkaitan dengan Karpenter itu sendiri.

Kunci AMIs di cluster produksi

Kami sangat menyarankan Anda menyematkan Gambar Mesin Amazon (AMIs) yang terkenal yang digunakan oleh Karpenter untuk kluster produksi. Menggunakan `amiSelector` dengan alias yang

disetel ke`@latest`, atau menggunakan beberapa metode lain yang menghasilkan penerapan yang belum teruji AMIs saat dirilis, menawarkan risiko kegagalan beban kerja dan waktu henti di kluster produksi Anda. Sebagai hasilnya, kami sangat menyarankan untuk menyematkan versi kerja yang telah diuji AMIs untuk kluster produksi Anda saat Anda menguji versi yang lebih baru di kluster non-produksi. Misalnya, Anda dapat menetapkan alias di Anda NodeClass sebagai berikut:

```
amiSelectorTerms
  - alias: al2023@v20240807
```

Untuk informasi tentang mengelola dan menyematkan AMIs di Karpenter, lihat Mengelola AMIs dalam dokumentasi Karpenter.

Gunakan Karpenter untuk beban kerja dengan kebutuhan kapasitas yang berubah

Karpenter membawa manajemen penskalaan lebih dekat ke Kubernetes native APIs daripada Autoscaling Groups () dan Managed Node Groups (ASGsMNG). ASG dan MNG adalah abstraksi asli AWS di mana penskalaan dipicu berdasarkan metrik level AWS, seperti beban CPU EC2. Cluster Autoscaler menjembatani abstraksi Kubernetes ke dalam abstraksi AWS, tetapi kehilangan beberapa fleksibilitas karena itu, seperti penjadwalan untuk zona ketersediaan tertentu.

Karpenter menghapus lapisan abstraksi AWS untuk membawa beberapa fleksibilitas langsung ke Kubernetes. Karpenter paling baik digunakan untuk cluster dengan beban kerja yang menghadapi periode permintaan tinggi dan runcing atau memiliki persyaratan komputasi yang beragam. MNGs dan ASGs bagus untuk cluster yang menjalankan beban kerja yang cenderung lebih statis dan konsisten. Anda dapat menggunakan campuran node yang dikelola secara dinamis dan statis, tergantung pada kebutuhan Anda.

Pertimbangkan proyek penskalaan otomatis lainnya ketika...

Anda membutuhkan fitur yang masih dikembangkan di Karpenter. Karena Karpenter adalah proyek yang relatif baru, pertimbangkan proyek penskalaan otomatis lainnya untuk saat ini jika Anda membutuhkan fitur yang belum menjadi bagian dari Karpenter.

Jalankan pengontrol Karpenter di EKS Fargate atau pada node pekerja yang termasuk dalam grup node

Karpenter diinstal menggunakan bagan Helm. Bagan Helm menginstal pengontrol Karpenter dan pod webhook sebagai Deployment yang perlu dijalankan sebelum controller dapat digunakan untuk penskalaan cluster Anda. Kami merekomendasikan minimal satu grup node kecil dengan setidaknya

satu node pekerja. Sebagai alternatif, Anda dapat menjalankan pod ini di EKS Fargate dengan membuat profil Fargate untuk namespace. `karpenter` Melakukannya akan menyebabkan semua pod yang diterapkan ke namespace ini berjalan di EKS Fargate. Jangan menjalankan Karpenter pada node yang dikelola oleh Karpenter.

Tidak ada dukungan template peluncuran khusus dengan Karpenter

Tidak ada dukungan template peluncuran khusus dengan v1 APIs. Anda dapat menggunakan data pengguna kustom and/or secara langsung menentukan kustom AMIs di. EC2 NodeClass Informasi lebih lanjut tentang cara melakukan ini tersedia di NodeClasses.

Kecualikan jenis instans yang tidak sesuai dengan beban kerja Anda

Pertimbangkan untuk mengecualikan jenis instance tertentu dengan `node.kubernetes.io/instance-type` kunci jika tidak diperlukan oleh beban kerja yang berjalan di klaster Anda.

Contoh berikut menunjukkan bagaimana menghindari penyediaan instance Graviton besar.

```
- key: node.kubernetes.io/instance-type
  operator: NotIn
  values:
  - m6g.16xlarge
  - m6gd.16xlarge
  - r6g.16xlarge
  - r6gd.16xlarge
  - c6g.16xlarge
```

Aktifkan Penanganan Gangguan saat menggunakan Spot

Karpenter mendukung penanganan interupsi asli dan dapat menangani peristiwa interupsi yang tidak disengaja seperti interupsi Instans Spot, peristiwa pemeliharaan terjadwal, penghentian/penghentian instance peristiwa yang dapat mengganggu beban kerja Anda. Ketika Karpenter mendeteksi peristiwa semacam itu untuk node, ia secara otomatis menodai, menguras, dan menghentikan node yang terpengaruh sebelumnya untuk memulai pembersihan beban kerja yang anggun sebelum gangguan. Untuk interupsi Spot dengan pemberitahuan 2 menit, Karpenter dengan cepat memulai node baru sehingga pod dapat dipindahkan sebelum instance direklamasi. Untuk mengaktifkan penanganan interupsi, Anda mengonfigurasi argumen `--interruption-queue` CLI dengan nama antrian SQS yang disediakan untuk tujuan ini. Tidak disarankan untuk menggunakan penanganan interupsi Karpenter bersama Node Termination Handler seperti yang dijelaskan di sini.

Pod yang memerlukan checkpointing atau bentuk lain dari pengeringan yang anggun, membutuhkan waktu 2 menit sebelum shutdown harus memungkinkan penanganan interupsi Karpenter di klaster mereka.

Cluster pribadi Amazon EKS tanpa akses internet keluar

Saat menyediakan Kluster EKS ke dalam VPC tanpa rute ke internet, Anda harus memastikan bahwa Anda telah mengonfigurasi lingkungan Anda sesuai dengan persyaratan klaster pribadi yang muncul dalam dokumentasi EKS. Selain itu, Anda perlu memastikan bahwa Anda telah membuat titik akhir regional STS VPC di VPC Anda. Jika tidak, Anda akan melihat kesalahan yang mirip dengan yang muncul di bawah ini.

```
{"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"}
```

Perubahan ini diperlukan dalam cluster pribadi karena Karpenter Controller menggunakan IAM Roles for Service Accounts (IRSA). Pod yang dikonfigurasi dengan IRSA memperoleh kredensyal dengan memanggil AWS Security Token Service (AWS STS) API. Jika tidak ada akses internet keluar, Anda harus membuat dan menggunakan titik akhir AWS STS VPC di VPC Anda.

Cluster pribadi juga mengharuskan Anda untuk membuat titik akhir VPC untuk SSM. Ketika Karpenter mencoba menyediakan node baru, ia menanyakan konfigurasi template Launch dan parameter SSM. Jika Anda tidak memiliki titik akhir VPC SSM di VPC Anda, itu akan menyebabkan kesalahan berikut:

```
{"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"}
```

Tidak ada titik akhir VPC untuk API Kueri Daftar Harga. Akibatnya, data harga akan basi seiring waktu. Karpenter menyiasatinya dengan memasukkan data harga sesuai permintaan dalam binernya,

tetapi hanya memperbarui data itu ketika Karpenter ditingkatkan. Permintaan data harga yang gagal akan menghasilkan pesan galat berikut:

```
{"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"}
```

Lihat dokumentasi ini untuk menggunakan Karpenter dalam Kluster EKS yang sepenuhnya Pribadi dan untuk mengetahui titik akhir VPC mana yang akan dibuat.

Menciptakan NodePools

Praktik terbaik berikut mencakup topik yang terkait dengan pembuatan NodePools.

Buat beberapa NodePools saat...

Ketika tim yang berbeda berbagi cluster dan perlu menjalankan beban kerja mereka pada node pekerja yang berbeda, atau memiliki persyaratan OS atau tipe instans yang berbeda, buat beberapa NodePools. Misalnya, satu tim mungkin ingin menggunakan Bottlerocket, sementara yang lain mungkin ingin menggunakan Amazon Linux. Demikian juga, satu tim mungkin memiliki akses ke perangkat keras GPU mahal yang tidak akan dibutuhkan oleh tim lain. Menggunakan beberapa NodePools memastikan bahwa aset yang paling tepat tersedia untuk setiap tim.

Buat NodePools yang saling eksklusif atau berbobot

Disarankan untuk membuat NodePools yang saling eksklusif atau berbobot untuk memberikan perilaku penjadwalan yang konsisten. Jika tidak dan beberapa NodePools dicocokkan, Karpenter akan secara acak memilih mana yang akan digunakan, menyebabkan hasil yang tidak terduga. Contoh yang berguna untuk membuat beberapa NodePools termasuk yang berikut:

Membuat NodePool dengan GPU dan hanya mengizinkan beban kerja khusus berjalan pada node (mahal) ini:

```
# 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"
```

Penerapan dengan toleransi untuk noda:

```
# 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"
```

Untuk penerapan umum untuk tim lain, NodePool spesifikasi dapat menyertakan NodeAffinity. Deployment kemudian dapat digunakan nodeSelectorTerms untuk mencocokkan`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
```

Penerapan menggunakan 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"]
```

Gunakan timer (TTL) untuk menghapus node secara otomatis dari cluster

Anda dapat menggunakan timer pada node yang disediakan untuk mengatur kapan harus menghapus node yang tidak memiliki pod beban kerja atau telah mencapai waktu kedaluwarsa. Kedaluwarsa node dapat digunakan sebagai sarana upgrade, sehingga node dihentikan dan diganti dengan versi yang diperbarui. Lihat Kedaluwarsa dalam dokumentasi Karpenter untuk informasi tentang penggunaan untuk mengonfigurasi `spec.template.spec` kedaluwarsa node.

Hindari terlalu membatasi Jenis Instance yang dapat disediakan Karpenter, terutama saat menggunakan Spot

Saat menggunakan Spot, Karpenter menggunakan strategi alokasi Price Capacity Optimized untuk menyediakan instans EC2. Strategi ini menginstruksikan EC2 untuk menyediakan instance dari kumpulan terdalam untuk jumlah instance yang Anda luncurkan dan memiliki risiko gangguan

terendah. Armada EC2 kemudian meminta instans Spot dari harga terendah dari kumpulan ini. Semakin banyak jenis instans yang Anda izinkan Karpenter gunakan, semakin baik EC2 dapat mengoptimalkan runtime instans spot Anda. Secara default, Karpenter akan menggunakan semua jenis instans yang ditawarkan EC2 di wilayah dan zona ketersediaan kluster Anda digunakan. Karpenter secara cerdas memilih dari kumpulan semua jenis instance berdasarkan pod yang tertunda untuk memastikan pod Anda dijadwalkan ke instance yang berukuran dan dilengkapi dengan tepat. Misalnya, jika pod Anda tidak memerlukan GPU, Karpenter tidak akan menjadwalkan pod Anda ke jenis instans EC2 yang mendukung GPU. Jika tidak yakin jenis instans mana yang akan digunakan, Anda dapat menjalankan Amazon ec2-instance-selector untuk membuat daftar jenis instance yang sesuai dengan persyaratan komputasi Anda. Misalnya, CLI mengambil memori vCPU, arsitektur, dan wilayah sebagai parameter input dan memberi Anda daftar instans EC2 yang memenuhi batasan tersebut.

```
$ 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
```

Anda tidak boleh menempatkan terlalu banyak kendala pada Karpenter saat menggunakan instance Spot karena hal itu dapat memengaruhi ketersediaan aplikasi Anda. Katakanlah, misalnya, semua contoh dari jenis tertentu direklamasi dan tidak ada alternatif yang cocok yang tersedia untuk menggantikannya. Pod Anda akan tetap dalam status tertunda hingga kapasitas spot untuk tipe instance yang dikonfigurasi diisi ulang. Anda dapat mengurangi risiko kesalahan kapasitas yang tidak mencukupi dengan menyebarkan instans Anda di berbagai zona ketersediaan, karena kumpulan spot berbeda di seluruh AZs area. Meskipun demikian, praktik terbaik umum adalah mengizinkan Karpenter menggunakan beragam jenis instance saat menggunakan Spot.

Penjadwalan Pod

Praktik terbaik berikut ini berkaitan dengan penerapan Pod Dalam klaster yang menggunakan Karpenter untuk penyediaan node.

Ikuti praktik terbaik EKS untuk ketersediaan tinggi

Jika Anda perlu menjalankan aplikasi yang sangat tersedia, ikuti rekomendasi praktik terbaik EKS umum. Lihat Topologi Spread di dokumentasi Karpenter untuk detail tentang cara menyebarkan pod di seluruh node dan zona. Gunakan Anggaran Gangguan untuk mengatur minimum pod yang tersedia yang perlu dipertahankan, jika ada upaya untuk mengusir atau menghapus pod.

Gunakan Kendala berlapis untuk membatasi fitur komputasi yang tersedia dari penyedia cloud Anda

Model kendala berlapis Karpenter memungkinkan Anda membuat serangkaian kendala penerapan NodePool dan pod yang kompleks untuk mendapatkan kecocokan terbaik untuk penjadwalan pod. Contoh kendala yang dapat diminta oleh spesifikasi pod meliputi:

- Perlu berjalan di zona ketersediaan di mana hanya aplikasi tertentu yang tersedia. Misalnya, Anda memiliki pod yang harus berkomunikasi dengan aplikasi lain yang berjalan pada instans EC2 yang berada di zona ketersediaan tertentu. Jika tujuan Anda adalah untuk mengurangi lalu lintas lintas lintas AZ di VPC Anda, Anda mungkin ingin menempatkan pod bersama di AZ tempat instans EC2 berada. Penargetan semacam ini sering dilakukan dengan menggunakan pemilih node. Untuk informasi tambahan tentang pemilih Node, silakan merujuk ke dokumentasi Kubernetes.
- Membutuhkan beberapa jenis prosesor atau perangkat keras lainnya. Lihat bagian Accelerators dari dokumen Karpenter untuk contoh spesifikasi pod yang mengharuskan pod berjalan pada GPU.

Buat alarm penagihan untuk memantau pengeluaran komputasi Anda

Ketika Anda mengonfigurasi klaster Anda untuk skala otomatis, Anda harus membuat alarm penagihan untuk memperingatkan Anda ketika pengeluaran Anda telah melampaui ambang batas dan menambahkan batas sumber daya ke konfigurasi Karpenter Anda. Menetapkan batas sumber daya dengan Karpenter mirip dengan menetapkan kapasitas maksimum grup penskalaan otomatis AWS karena mewakili jumlah maksimum sumber daya komputasi yang dapat digunakan oleh Karpenter. NodePool

> **Note**
>
> Tidak mungkin menetapkan batas global untuk seluruh cluster. Batas berlaku untuk spesifik NodePools.

Cuplikan di bawah ini memberi tahu Karpenter untuk hanya menyediakan maksimum 1000 core CPU dan 1000Gi memori. Karpenter akan berhenti menambah kapasitas hanya ketika batas terpenuhi atau terlampaui. Ketika batas terlampaui, pengontrol Karpenter akan menulis `memory resource usage of 1001 exceeds limit of 1000` atau pesan yang mirip dengan log pengontrol. Jika Anda merutekan log kontainer Anda ke CloudWatch log, Anda dapat membuat filter metrik untuk mencari pola atau istilah tertentu dalam log Anda dan kemudian membuat CloudWatchalarm untuk mengingatkan Anda ketika ambang metrik yang dikonfigurasi dilanggar.

Untuk informasi lebih lanjut menggunakan limit dengan Karpenter, lihat Menetapkan Batas Sumber Daya dalam dokumentasi Karpenter.

```
spec:
  limits:
    cpu: 1000
    memory: 1000Gi
```

Jika Anda tidak menggunakan batas atau membatasi jenis instance yang dapat disediakan Karpenter, Karpenter akan terus menambahkan kapasitas komputasi ke cluster Anda sesuai kebutuhan. Meskipun mengonfigurasi Karpenter dengan cara ini memungkinkan klaster Anda untuk menskalakan secara bebas, itu juga dapat memiliki implikasi biaya yang signifikan. Karena alasan inilah kami merekomendasikan untuk mengonfigurasi alarm penagihan. Alarm penagihan memungkinkan Anda untuk diperingatkan dan diberitahukan secara proaktif ketika perkiraan biaya yang dihitung di akun Anda melebihi ambang batas yang ditentukan. Lihat Menyiapkan Alarm CloudWatch Penagihan Amazon untuk Memantau Perkiraan Biaya Secara Proaktif untuk informasi tambahan.

Anda mungkin juga ingin mengaktifkan Deteksi Anomali Biaya yang merupakan fitur AWS Cost Management yang menggunakan pembelajaran mesin untuk terus memantau biaya dan penggunaan Anda untuk mendeteksi pengeluaran yang tidak biasa. Informasi lebih lanjut dapat ditemukan di panduan Memulai Deteksi Anomali Biaya AWS. Jika Anda telah membuat anggaran di AWS Budgets, Anda juga dapat mengonfigurasi tindakan untuk memberi tahu Anda ketika ambang batas tertentu telah dilanggar. Dengan tindakan anggaran, Anda dapat mengirim email, mengirim pesan ke topik SNS, atau mengirim pesan ke chatbot seperti Slack. Untuk informasi lebih lanjut, lihat Mengonfigurasi tindakan AWS Budgets.

Gunakan do-not-disrupt anotasi karpenter.sh/ untuk mencegah Karpenter dari deprovisioning node

Jika Anda menjalankan aplikasi penting pada node yang disediakan Karpenter, seperti pekerjaan batch yang berjalan lama atau aplikasi stateful, dan TTL node telah kedaluwarsa, aplikasi akan terganggu saat instance dihentikan. Dengan menambahkan `karpenter.sh/do-not-disrupt` anotasi ke pod, Anda menginstruksikan Karpenter untuk mempertahankan node sampai Pod dihentikan atau anotasi dihapus. `karpenter.sh/do-not-disrupt` Lihat dokumentasi Distruption untuk informasi lebih lanjut.

Jika satu-satunya pod non-daemonset yang tersisa di node adalah yang terkait dengan pekerjaan, Karpenter dapat menargetkan dan menghentikan node tersebut selama status pekerjaan berhasil atau gagal.

Konfigurasikan requests=limits untuk semua sumber daya non-CPU saat menggunakan konsolidasi

Konsolidasi dan penjadwalan secara umum bekerja dengan membandingkan permintaan sumber daya pod vs jumlah sumber daya yang dapat dialokasikan pada sebuah node. Batas sumber daya tidak dipertimbangkan. Sebagai contoh, pod yang memiliki batas memori yang lebih besar dari permintaan memori dapat meledak di atas permintaan. Jika beberapa pod pada node yang sama meledak pada saat yang sama, ini dapat menyebabkan beberapa pod dihentikan karena kondisi out of memory (OOM). Konsolidasi dapat membuat ini lebih mungkin terjadi karena berfungsi untuk mengemas pod ke node hanya dengan mempertimbangkan permintaannya.

Gunakan LimitRanges untuk mengonfigurasi default untuk permintaan dan batasan sumber daya

Karena Kubernetes tidak menetapkan permintaan atau batasan default, konsumsi sumber daya kontainer dari host, CPU, dan memori yang mendasarinya tidak terikat. Penjadwal Kubernetes melihat total permintaan sebuah pod (jumlah permintaan yang lebih tinggi dari kontainer pod atau total sumber daya dari kontainer Init pod) untuk menentukan node pekerja mana yang akan menjadwalkan pod tersebut. Demikian pula, Karpenter mempertimbangkan permintaan pod untuk menentukan jenis instance yang disediakannya. Anda dapat menggunakan rentang batas untuk menerapkan default yang masuk akal untuk namespace, jika permintaan sumber daya tidak ditentukan oleh beberapa pod.

Lihat Mengkonfigurasi Permintaan dan Batas Memori Default untuk Namespace

Terapkan permintaan sumber daya yang akurat ke semua beban kerja

Karpenter dapat meluncurkan node yang paling sesuai dengan beban kerja Anda ketika informasi tentang persyaratan beban kerja Anda akurat. Ini sangat penting jika menggunakan fitur konsolidasi Karpenter.

Lihat Mengkonfigurasi dan Mengukur Sumber Daya Requests/Limits untuk semua Beban Kerja

Rekomendasi CoreDNS

Perbarui konfigurasi CoreDNS untuk menjaga keandalan

Saat menerapkan pod CoreDNS pada node yang dikelola oleh Karpenter, mengingat sifat dinamis Karpenter dalam node baru yang terminating/creating cepat agar selaras dengan permintaan, disarankan untuk mematuhi praktik terbaik berikut:

Durasi lameduck CoreDNS

Penyelidikan kesiapan CoreDNS

Ini akan memastikan bahwa query DNS tidak diarahkan ke Pod CoreDNS yang belum siap atau telah dihentikan.

Cetak Biru Karpenter

Karena Karpenter mengambil pendekatan aplikasi-pertama untuk menyediakan kapasitas komputasi untuk bidang data Kubernetes, ada skenario beban kerja umum yang mungkin membuat Anda bertanya-tanya bagaimana mengonfigurasinya dengan benar. Karpenter Blueprints adalah repositori yang mencakup daftar skenario beban kerja umum mengikuti praktik terbaik yang dijelaskan di sini. Anda akan memiliki semua sumber daya yang Anda butuhkan untuk membuat cluster EKS dengan Karpenter yang dikonfigurasi, dan menguji setiap cetak biru yang disertakan dalam repositori. Anda dapat menggabungkan cetak biru yang berbeda untuk akhirnya membuat yang Anda butuhkan untuk beban kerja Anda.

Sumber Daya Tambahan

- Lokakarya Hari Perendaman Karpenter
- Lokakarya Optimalisasi Biaya Karpenter
- Lokakarya EKS - Karpenter
- Karpenter vs Cluster Autoscaler

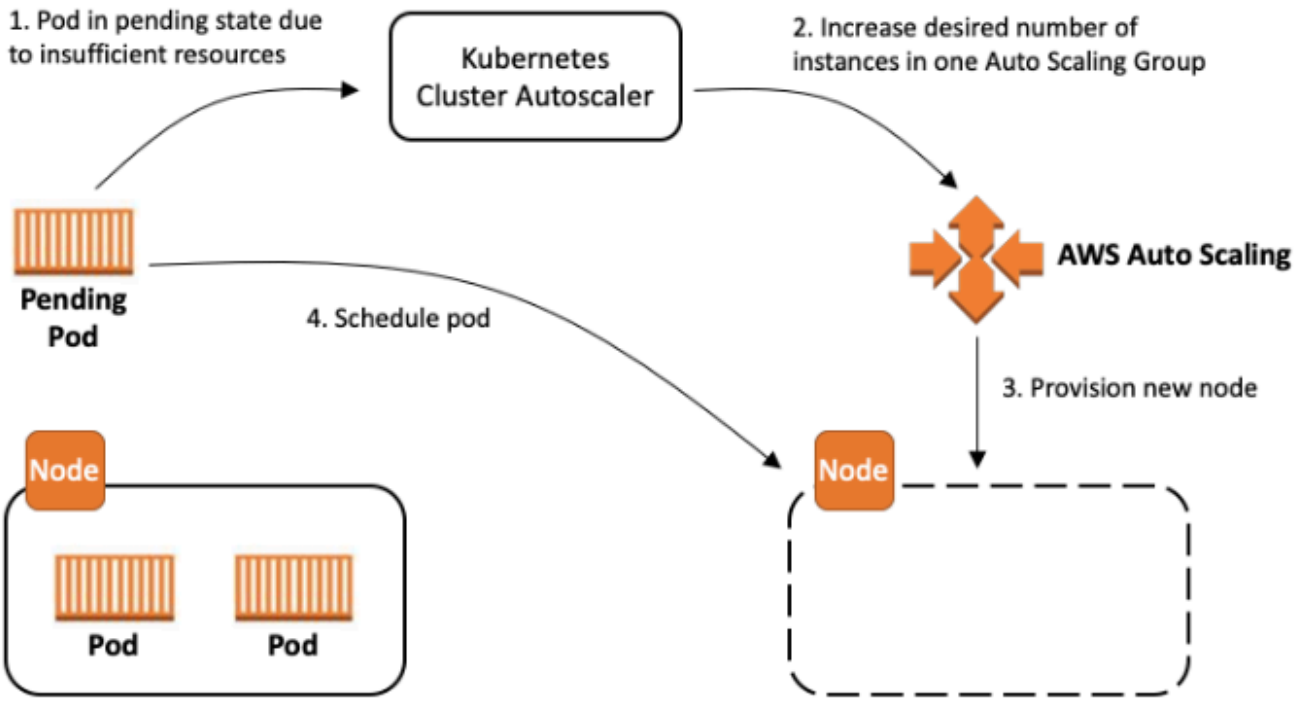
- Sesi Karpenter di re:Invent 2023
- Tutorial: Jalankan Kubernetes Clusters dengan Harga Lebih Murah dengan Amazon EC2 Spot dan Karpenter

Autoscaler klaster

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Ikhtisar

Kubernetes Cluster Autoscaler adalah solusi Cluster Autoscaling populer yang dikelola oleh SIG Autoscaling. Ini bertanggung jawab untuk memastikan bahwa klaster Anda memiliki cukup node untuk menjadwalkan pod Anda tanpa membuang sumber daya. Ini mengawasi pod yang gagal menjadwalkan dan untuk node yang kurang dimanfaatkan. Kemudian mensimulasikan penambahan atau penghapusan node sebelum menerapkan perubahan ke cluster Anda. Implementasi AWS Cloud Provider dalam Cluster Autoscaler mengontrol `.DesiredReplicas` bidang Grup Auto Scaling EC2 Anda.



Panduan ini akan memberikan model mental untuk mengonfigurasi Cluster Autoscaler dan memilih rangkaian pengorbanan terbaik untuk memenuhi persyaratan organisasi Anda. Meskipun tidak ada satu konfigurasi terbaik, ada serangkaian opsi konfigurasi yang memungkinkan Anda menukar kinerja, skalabilitas, biaya, dan ketersediaan. Selain itu, panduan ini akan memberikan tips dan praktik terbaik untuk mengoptimalkan konfigurasi AWS Anda.

Glosarium

Terminologi berikut akan sering digunakan di seluruh dokumen ini. Istilah-istilah ini dapat memiliki arti yang luas, tetapi terbatas pada definisi di bawah ini untuk keperluan dokumen ini.

Skalabilitas mengacu pada seberapa baik kinerja Cluster Autoscaler saat Cluster Kubernetes Anda bertambah dalam jumlah pod dan node. Saat batas skalabilitas tercapai, kinerja dan fungsionalitas Cluster Autoscaler menurun. Karena Cluster Autoscaler melebihi batas skalabilitasnya, ia mungkin tidak lagi menambah atau menghapus node di cluster Anda.

Kinerja mengacu pada seberapa cepat Cluster Autoscaler mampu membuat dan melaksanakan keputusan penskalaan. Cluster Autoscaler yang berkinerja sempurna akan langsung membuat keputusan dan memicu tindakan penskalaan sebagai respons terhadap rangsangan, seperti pod menjadi tidak dapat dijadwalkan.

Ketersediaan berarti pod dapat dijadwalkan dengan cepat dan tanpa gangguan. Ini termasuk ketika pod yang baru dibuat perlu dijadwalkan dan ketika node yang diperkecil menghentikan semua pod yang tersisa yang dijadwalkan untuk itu.

Biaya ditentukan oleh keputusan di balik skala dan skala dalam peristiwa. Sumber daya terbuang jika node yang ada kurang dimanfaatkan atau node baru ditambahkan yang terlalu besar untuk pod yang masuk. Bergantung pada kasus penggunaan, mungkin ada biaya yang terkait dengan penghentian pod sebelum waktunya karena keputusan penurunan skala yang agresif.

Node Groups adalah konsep Kubernetes abstrak untuk sekelompok node dalam sebuah cluster. Ini bukan sumber daya Kubernetes yang sebenarnya, tetapi ada sebagai abstraksi di Cluster Autoscaler, Cluster API, dan komponen lainnya. Node dalam Grup Node berbagi properti seperti label dan taints, tetapi dapat terdiri dari beberapa Availability Zone atau Instance Type.

EC2 Auto Scaling Groups dapat digunakan sebagai implementasi Node Groups pada EC2. Grup Auto Scaling EC2 dikonfigurasi untuk meluncurkan instans yang secara otomatis bergabung dengan Kluster Kubernetes mereka dan menerapkan label dan taint ke sumber daya Node yang sesuai di Kubernetes API.

EC2 Managed Node Groups adalah implementasi lain dari Node Groups pada EC2. Mereka mengabstraksikan kompleksitas secara manual mengonfigurasi Grup Penskalaan Otomatis EC2 dan menyediakan fitur manajemen tambahan seperti peningkatan versi node dan penghentian node yang anggun.

Mengoperasikan Cluster Autoscaler

Cluster Autoscaler biasanya diinstal sebagai Deployment di klaster Anda. Ini menggunakan pemilihan pemimpin untuk memastikan ketersediaan tinggi, tetapi pekerjaan dilakukan oleh satu replika pada satu waktu. Ini tidak dapat diskalakan secara horizontal. Untuk pengaturan dasar, default itu harus bekerja di luar kotak menggunakan instruksi instalasi yang disediakan, tetapi ada beberapa hal yang perlu diingat.

Pastikan bahwa:

- Versi Cluster Autoscaler cocok dengan Versi Cluster. Kompatibilitas lintas versi tidak diuji atau didukung.
- Penemuan Otomatis diaktifkan, kecuali jika Anda memiliki kasus penggunaan lanjutan khusus yang mencegah penggunaan mode ini.

Gunakan akses paling tidak istimewa ke peran IAM

Saat Auto Discovery digunakan, kami sangat menyarankan agar Anda menggunakan akses hak istimewa paling sedikit dengan membatasi Tindakan `autoscaling:SetDesiredCapacity` dan `autoscaling:TerminateInstanceInAutoScalingGroup` grup Auto Scaling yang tercakup pada klaster saat ini.

Ini akan mencegah Cluster Autoscaler yang berjalan di satu cluster memodifikasi nodegroup di cluster yang berbeda bahkan jika `--node-group-auto-discovery` argumen tidak dicakup ke nodegroup cluster menggunakan tag (misalnya). `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": "*"
        }
    ]
}
```

Mengonfigurasi Grup Node Anda

Penskalaan otomatis yang efektif dimulai dengan mengonfigurasi satu set Grup Node dengan benar untuk klaster Anda. Memilih kumpulan Grup Node yang tepat adalah kunci untuk memaksimalkan ketersediaan dan mengurangi biaya di seluruh beban kerja Anda. AWS mengimplementasikan Grup Node menggunakan EC2 Auto Scaling Groups, yang fleksibel untuk sejumlah besar kasus penggunaan. Namun, Cluster Autoscaler membuat beberapa asumsi tentang Grup Node Anda. Menjaga konfigurasi Grup Auto Scaling EC2 Anda konsisten dengan asumsi ini akan meminimalkan perilaku yang tidak diinginkan.

Pastikan bahwa:

- Setiap Node dalam Grup Node memiliki properti penjadwalan yang identik, seperti Label, Taints, dan Resources.

  - Untuk MixedInstancePolicies, Jenis Instance harus memiliki bentuk yang sama untuk CPU, Memori, dan GPU
  - Jenis Instance pertama yang ditentukan dalam kebijakan akan digunakan untuk mensimulasikan penjadwalan.
  - Jika kebijakan Anda memiliki Jenis Instance tambahan dengan lebih banyak sumber daya, sumber daya mungkin terbuang setelah skala habis.
  - Jika kebijakan Anda memiliki Tipe Instance tambahan dengan sumber daya yang lebih sedikit, pod mungkin gagal menjadwalkan instans.
- Grup Node dengan banyak node lebih disukai daripada banyak Grup Node dengan lebih sedikit node. Ini akan memiliki dampak terbesar pada skalabilitas.
- Jika memungkinkan, pilih fitur EC2 ketika kedua sistem memberikan dukungan (misalnya Wilayah, MixedInstancePolicy)

> **Note**
>
> Kami merekomendasikan menggunakan Grup Node Terkelola EKS. Grup Node Terkelola hadir dengan fitur manajemen yang kuat, termasuk fitur untuk Cluster Autoscaler seperti penemuan Grup EC2 Auto Scaling otomatis dan penghentian node yang anggun.

Mengoptimalkan Kinerja dan Skalabilitas

Memahami kompleksitas runtime algoritma autoscaling akan membantu Anda menyetel Cluster Autoscaler untuk terus beroperasi dengan lancar di cluster besar dengan lebih dari 1.000 node.

Tombol-tombol utama untuk menyetel skalabilitas Cluster Autoscaler adalah sumber daya yang disediakan untuk proses, interval pemindaian algoritma, dan jumlah Grup Node di cluster. Ada faktor lain yang terlibat dalam kompleksitas runtime sebenarnya dari algoritma ini, seperti menjadwalkan kompleksitas plugin dan jumlah pod. Ini dianggap sebagai parameter yang tidak dapat dikonfigurasi karena alami untuk beban kerja cluster dan tidak dapat dengan mudah disetel.

Cluster Autoscaler memuat seluruh status klaster ke dalam memori, termasuk Pod, Nodes, dan Node Groups. Pada setiap interval pemindaian, algoritme mengidentifikasi pod yang tidak dapat dijadwalkan dan mensimulasikan penjadwalan untuk setiap Grup Node. Menyetel faktor-faktor ini datang dengan pengorbanan yang berbeda yang harus dipertimbangkan dengan cermat untuk kasus penggunaan Anda.

Penskalaan Otomatis Secara Vertikal Cluster Autoscaler

Cara termudah untuk menskalakan Cluster Autoscaler ke cluster yang lebih besar adalah dengan meningkatkan permintaan sumber daya untuk penerapannya. Baik memori dan CPU harus ditingkatkan untuk cluster besar, meskipun ini bervariasi secara signifikan dengan ukuran cluster. Algoritma penskalaan otomatis menyimpan semua pod dan node dalam memori, yang dapat menghasilkan footprint memori yang lebih besar dari satu gigabyte dalam beberapa kasus. Meningkatkan sumber daya biasanya dilakukan secara manual. Jika Anda menemukan bahwa penyetelan sumber daya konstan menciptakan beban operasional, pertimbangkan untuk menggunakan Addon Resizer atau Vertical Pod Autoscaler.

Mengurangi jumlah Grup Node

Meminimalkan jumlah grup node adalah salah satu cara untuk memastikan bahwa Cluster Autoscaler akan terus berkinerja baik pada cluster besar. Ini mungkin menantang bagi beberapa organisasi yang menyusun grup node mereka per tim atau per aplikasi. Meskipun ini sepenuhnya didukung oleh Kubernetes API, ini dianggap sebagai anti-pola Cluster Autoscaler dengan dampak untuk skalabilitas. Ada banyak alasan untuk menggunakan beberapa grup node (misalnya Spot atau GPUs), tetapi dalam banyak kasus ada desain alternatif yang mencapai efek yang sama saat menggunakan sejumlah kecil grup.

Pastikan bahwa:

- Isolasi Pod dilakukan dengan menggunakan Namespaces daripada Node Groups.
  - Ini mungkin tidak mungkin dilakukan di cluster multi-tenant dengan kepercayaan rendah.
  - Pod ResourceRequests dan ResourceLimits diatur dengan benar untuk menghindari pertentangan sumber daya.
  - Jenis instans yang lebih besar akan menghasilkan pengemasan bin yang lebih optimal dan pengurangan overhead pod sistem.
- NodeTaints atau NodeSelectors digunakan untuk menjadwalkan pod sebagai pengecualian, bukan sebagai aturan.
- Sumber daya regional didefinisikan sebagai Grup Auto Scaling EC2 tunggal dengan beberapa Availability Zone.

Mengurangi Interval Pemindaian

Interval pemindaian yang rendah (misalnya 10 detik) akan memastikan bahwa Cluster Autoscaler merespons secepat mungkin ketika pod menjadi tidak terjadwal. Namun, setiap pemindaian menghasilkan banyak panggilan API ke Kubernetes API dan EC2 Auto Scaling Group atau EKS Managed Node Group. APIs Panggilan API ini dapat mengakibatkan pembatasan tarif atau bahkan tidak tersedianya layanan untuk Kubernetes Control Plane Anda.

Interval pemindaian default adalah 10 detik, tetapi di AWS, peluncuran node membutuhkan waktu lebih lama untuk meluncurkan instance baru. Ini berarti bahwa ada kemungkinan untuk meningkatkan interval tanpa secara signifikan meningkatkan waktu untuk menaikkan skala keseluruhan. Misalnya, jika dibutuhkan 2 menit untuk meluncurkan node, mengubah interval menjadi 1 menit akan menghasilkan tradeoff 6x panggilan API berkurang untuk peningkatan skala 38% lebih lambat.

Sharding Lintas Grup Node

Cluster Autoscaler dapat dikonfigurasi untuk beroperasi pada satu set Grup Node tertentu. Dengan menggunakan fungsi ini, dimungkinkan untuk menerapkan beberapa instance Cluster Autoscaler, masing-masing dikonfigurasi untuk beroperasi pada kumpulan Grup Node yang berbeda. Strategi ini memungkinkan Anda menggunakan sejumlah besar Grup Node secara sewenang-wenang, biaya perdagangan untuk skalabilitas. Kami hanya merekomendasikan menggunakan ini sebagai upaya terakhir untuk meningkatkan kinerja.

Cluster Autoscaler awalnya tidak dirancang untuk konfigurasi ini, jadi ada beberapa efek samping. Karena pecahan tidak berkomunikasi, beberapa autoscaler dapat mencoba menjadwalkan pod yang tidak dapat dijadwalkan. Hal ini dapat mengakibatkan skala yang tidak perlu dari beberapa Grup Node. Node tambahan ini akan menskalakan kembali setelah`scale-down-delay`.

```
metadata:
  name: cluster-autoscaler
  namespace: cluster-autoscaler-1

...

--nodes=1:10:k8s-worker-asg-1
--nodes=1:10:k8s-worker-asg-2

---

metadata:
```

```
  name: cluster-autoscaler
  namespace: cluster-autoscaler-2

...

--nodes=1:10:k8s-worker-asg-3
--nodes=1:10:k8s-worker-asg-4
```

Pastikan bahwa:

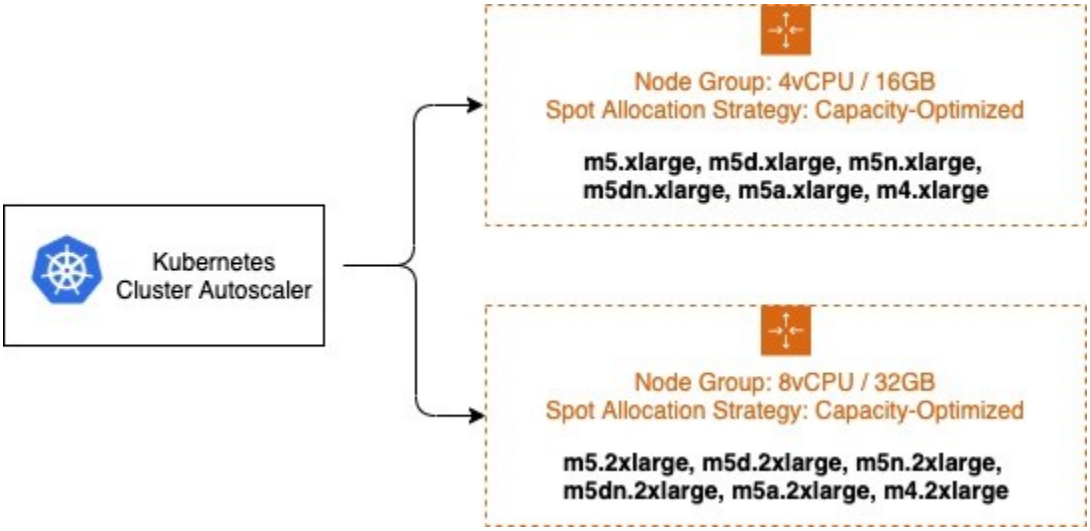
- Setiap pecahan dikonfigurasi untuk menunjuk ke satu set unik Grup Auto Scaling EC2
- Setiap pecahan dikerahkan ke namespace terpisah untuk menghindari konflik pemilihan pemimpin

Mengoptimalkan Biaya dan Ketersediaan

Instans Spot

Anda dapat menggunakan Instans Spot di grup node Anda dan menghemat hingga 90% dari harga sesuai permintaan, dengan trade-off Instans Spot dapat terganggu kapan saja ketika EC2 membutuhkan kapasitas kembali. Kesalahan Kapasitas Tidak Cukup akan terjadi ketika grup EC2 Auto Scaling Anda tidak dapat meningkatkan skala karena kurangnya kapasitas yang tersedia. Memaksimalkan keragaman dengan memilih banyak kelompok instans dapat meningkatkan peluang Anda untuk mencapai skala yang Anda inginkan dengan memanfaatkan banyak kumpulan kapasitas Spot, dan mengurangi dampak interupsi Instans Spot pada ketersediaan klaster Anda. Kebijakan Instans Campuran dengan Instans Spot adalah cara yang bagus untuk meningkatkan keragaman tanpa meningkatkan jumlah grup simpul. Perlu diingat, jika Anda membutuhkan sumber daya yang terjamin, gunakan Instans Sesuai Permintaan, bukan Instans Spot.

Sangat penting bahwa semua Jenis Instance memiliki kapasitas sumber daya yang sama saat mengonfigurasi Kebijakan Instance Campuran. Simulator penjadwalan autoscaler menggunakan yang pertama di. InstanceType MixedInstancePolicy Jika Jenis Instance berikutnya lebih besar, sumber daya mungkin terbuang setelah peningkatan skala. Jika lebih kecil, pod Anda mungkin gagal menjadwalkan instans baru karena kapasitas yang tidak mencukupi. Misalnya, instance M4, M5, M5a, dan M5n semuanya memiliki jumlah CPU dan Memori yang sama dan merupakan kandidat yang bagus untuk file. MixedInstancePolicy Alat Pemilih Instance EC2 dapat membantu Anda mengidentifikasi jenis instans yang serupa.



Disarankan untuk mengisolasi kapasitas On-Demand dan Spot ke dalam grup Auto Scaling EC2 yang terpisah. Ini lebih disukai daripada menggunakan strategi kapasitas dasar karena properti penjadwalan pada dasarnya berbeda. Karena Instans Spot terputus kapan saja (ketika EC2 membutuhkan kapasitas kembali), pengguna akan sering mencemari node preemptable mereka, memerlukan toleransi pod eksplisit terhadap perilaku preemption. Taints ini menghasilkan properti penjadwalan yang berbeda untuk node, sehingga mereka harus dipisahkan menjadi beberapa Grup Auto Scaling EC2.

Cluster Autoscaler memiliki konsep Expander, yang menyediakan strategi berbeda untuk memilih Grup Node mana yang akan diskalakan. Strateginya `--expander=least-waste` adalah default tujuan umum yang baik, dan jika Anda akan menggunakan beberapa grup simpul untuk diversifikasi Instans Spot (seperti yang dijelaskan pada gambar di atas), ini dapat membantu mengoptimalkan biaya lebih lanjut grup node dengan menskalakan grup yang paling baik digunakan setelah aktivitas penskalaan.

Memprioritaskan grup simpul/ASG

Anda juga dapat mengonfigurasi penskalaan otomatis berbasis prioritas dengan menggunakan Expander Prioritas. `--expander=priority`memungkinkan klaster Anda untuk memprioritaskan grup simpul/ASG, dan jika tidak dapat menskalakan karena alasan apa pun, ia akan memilih grup simpul berikutnya dalam daftar yang diprioritaskan. Ini berguna dalam situasi di mana, misalnya, Anda ingin menggunakan jenis instans P3 karena GPU mereka memberikan kinerja optimal untuk beban kerja Anda, tetapi sebagai opsi kedua Anda juga dapat menggunakan jenis instans P2.

```
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: cluster-autoscaler-priority-expander
  namespace: kube-system
data:
  priorities: |-
    10:
      - .*p2-node-group.*
    50:
      - .*p3-node-group.*
```

Cluster Autoscaler akan mencoba meningkatkan grup Auto Scaling EC2 yang cocok dengan nama p3-node-group. Jika operasi ini tidak berhasil di dalamnya`--max-node-provision-time`, ia akan mencoba untuk menskalakan grup Auto Scaling EC2 yang cocok dengan nama p2-node-group. Nilai ini default menjadi 15 menit dan dapat dikurangi untuk pemilihan grup node yang lebih responsif, meskipun jika nilainya terlalu rendah, itu dapat menyebabkan pemutusan skala yang tidak perlu.

Penyediaan berlebihan

Cluster Autoscaler meminimalkan biaya dengan memastikan bahwa node hanya ditambahkan ke cluster bila diperlukan dan dihapus ketika tidak digunakan. Ini secara signifikan memengaruhi latensi penerapan karena banyak pod akan dipaksa untuk menunggu skala node sebelum dapat dijadwalkan. Simpul membutuhkan waktu beberapa menit untuk tersedia, yang dapat meningkatkan latensi penjadwalan pod melalui urutan besarnya.

Ini dapat dikurangi menggunakan kelebihan penyediaan, yang membutuhkan biaya transaksi untuk latensi penjadwalan. Overprovisioning diimplementasikan menggunakan pod temporer dengan prioritas negatif, yang menempati ruang di cluster. Ketika pod yang baru dibuat tidak dapat dijadwalkan dan memiliki prioritas yang lebih tinggi, pod sementara akan didahului untuk memberi ruang. Pod sementara kemudian menjadi tidak dapat dijadwalkan, memicu Cluster Autoscaler untuk menskalakan node baru yang telah dilebih-lebihkan.

Ada manfaat lain yang kurang jelas untuk penyediaan berlebihan. Tanpa overprovisioning, salah satu efek samping dari cluster yang sangat dimanfaatkan adalah bahwa pod akan membuat keputusan penjadwalan yang kurang optimal menggunakan `preferredDuringSchedulingIgnoredDuringExecution` aturan Pod atau Node Affinity. Kasus penggunaan umum untuk ini adalah memisahkan pod untuk aplikasi yang sangat tersedia di seluruh zona ketersediaan yang menggunakan AntiAffinity. Penyediaan berlebihan dapat secara signifikan meningkatkan kemungkinan node dari zona yang benar tersedia.

Jumlah kapasitas yang dilebih-lebihkan adalah keputusan bisnis yang cermat untuk organisasi Anda. Pada intinya, ini adalah tradeoff antara kinerja dan biaya. Salah satu cara untuk membuat keputusan ini adalah dengan menentukan frekuensi skala rata-rata Anda dan membaginya dengan jumlah waktu yang diperlukan untuk meningkatkan node baru. Misalnya, jika rata-rata Anda memerlukan node baru setiap 30 detik dan EC2 membutuhkan waktu 30 detik untuk menyediakan node baru, satu node overprovisioning akan memastikan bahwa selalu ada node tambahan yang tersedia, mengurangi latensi penjadwalan hingga 30 detik dengan biaya satu Instans EC2 tambahan. Untuk meningkatkan keputusan penjadwalan zona, berikan berlebihan sejumlah node yang sama dengan jumlah zona ketersediaan di Grup Penskalaan Otomatis EC2 Anda untuk memastikan bahwa penjadwal dapat memilih zona terbaik untuk pod yang masuk.

Mencegah Penggusuran Scale Down

Beberapa beban kerja terlalu mahal untuk dikosongkan. Analisis data besar, tugas pembelajaran mesin, dan pelari uji pada akhirnya akan selesai, tetapi harus dimulai ulang jika terganggu. Cluster Autoscaler akan mencoba untuk menurunkan setiap node di bawah scale-down-utilization-threshold, yang akan mengganggu setiap pod yang tersisa pada node. Hal ini dapat dicegah dengan memastikan bahwa pod yang mahal untuk diusir dilindungi oleh label yang dikenali oleh Cluster Autoscaler.

Pastikan bahwa:

- Mahal untuk mengusir pod memiliki anotasi `cluster-autoscaler.kubernetes.io/safe-to-evict=false`

Kasus Penggunaan Tingkat Lanjut

Volume EBS

Penyimpanan persisten sangat penting untuk membangun aplikasi stateful, seperti database atau cache terdistribusi. Volume EBS mengaktifkan kasus penggunaan ini di Kubernetes, tetapi terbatas pada zona tertentu. Aplikasi ini dapat sangat tersedia jika di-sharded di beberapa AZ menggunakan Volume EBS terpisah untuk setiap AZ. Cluster Autoscaler kemudian dapat menyeimbangkan penskalaan Grup Autoscaling EC2.

Pastikan bahwa:

- Penyeimbangan grup simpul diaktifkan dengan pengaturan `balance-similar-node-groups=true`.

- Grup Node dikonfigurasi dengan pengaturan yang identik kecuali untuk zona ketersediaan dan Volume EBS yang berbeda.

Penjadwalan Bersama

Tugas-tugas pelatihan yang didistribusikan machine learning mendapatkan manfaat signifikan dari latensi yang diminimalkan atas konfigurasi simpul zona yang sama. Beban kerja ini menyebarkan beberapa pod ke zona tertentu. Hal ini dapat dicapai dengan menetapkan Pod Affinity untuk semua Pod yang dijadwalkan bersama atau Node Affinity menggunakan. `topologyKey: failure-domain.beta.kubernetes.io/zone` Cluster Autoscaler kemudian akan menskalakan zona tertentu agar sesuai dengan permintaan. Anda mungkin ingin mengalokasikan beberapa Grup Auto Scaling EC2, satu per zona ketersediaan untuk mengaktifkan failover untuk seluruh beban kerja yang dijadwalkan bersama.

Pastikan bahwa:

- Penyeimbang group simpul diaktifkan dengan pengaturan `balance-similar-node-groups=false`
- Node Affinity dan/atau Pod Preemption digunakan ketika cluster mencakup Grup Node Regional dan Zonal.
  - Gunakan Node Affinity untuk memaksa atau mendorong pod regional untuk menghindari Zonal Node Groups, dan sebaliknya.
  - Jika pod zonal dijadwalkan ke grup node regional, ini akan mengakibatkan ketidakseimbangan kapasitas untuk pod regional Anda.
  - Jika beban kerja zonal Anda dapat mentolerir gangguan dan relokasi, konfigurasikan Pod Preemption untuk mengaktifkan pod yang diskalakan secara regional untuk memaksa preemption dan penjadwalan ulang pada zona yang kurang diperebutkan.

Akselerator

Beberapa cluster memanfaatkan akselerator perangkat keras khusus seperti GPU. Saat menskalakan, plugin perangkat akselerator dapat memakan waktu beberapa menit untuk mengiklankan sumber daya ke cluster. Cluster Autoscaler telah mensimulasikan bahwa node ini akan memiliki akselerator, tetapi sampai akselerator menjadi siap dan memperbarui sumber daya node yang tersedia, pod yang tertunda tidak dapat dijadwalkan pada node. Hal ini dapat mengakibatkan menskalakan keluar tidak diperlukan yang berulang.

Selain itu, node dengan akselerator dan pemanfaatan CPU atau Memori yang tinggi tidak akan dipertimbangkan untuk menurunkan skala, bahkan jika akselerator tidak digunakan. Perilaku ini bisa mahal karena biaya relatif akselerator. Sebagai gantinya, Cluster Autoscaler dapat menerapkan aturan khusus untuk mempertimbangkan node untuk menurunkan skala jika mereka memiliki akselerator kosong.

Untuk memastikan perilaku yang benar untuk kasus-kasus ini, Anda dapat mengonfigurasi kubelet pada node akselerator Anda untuk memberi label pada node sebelum bergabung dengan cluster. Cluster Autoscaler akan menggunakan pemilih label ini untuk memicu perilaku akselerator yang dioptimalkan.

Pastikan bahwa:

- Kubelet untuk node GPU dikonfigurasi dengan `--node-labels k8s.amazonaws.com/accelerator=$ACCELERATOR_TYPE`
- Node dengan Accelerators mematuhi aturan properti penjadwalan identik yang disebutkan di atas.

Penskalaan dari 0

Cluster Autoscaler mampu menskalakan Grup Node ke dan dari nol, yang dapat menghasilkan penghematan biaya yang signifikan. Ini mendeteksi sumber daya CPU, memori, dan GPU dari Grup Auto Scaling dengan memeriksa yang ditentukan InstanceType dalam atau. LaunchConfiguration LaunchTemplate Beberapa pod memerlukan sumber daya tambahan seperti `WindowsENI` atau `PrivateIPv4Address` atau spesifik NodeSelectors atau Taints yang tidak dapat ditemukan dari. LaunchConfiguration Cluster Autoscaler dapat menjelaskan faktor-faktor ini dengan menemukannya dari tag pada EC2 Auto Scaling Group. Contoh:

```
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**
>
> Perlu diingat, saat penskalaan ke nol kapasitas Anda dikembalikan ke EC2 dan mungkin tidak tersedia di masa mendatang.

Parameter Tambahan

Ada banyak opsi konfigurasi yang dapat digunakan untuk menyesuaikan perilaku dan kinerja Cluster Autoscaler. Daftar lengkap parameter tersedia di GitHub.

Parameter	Deskripsi	Default
interval pemindaian	Seberapa sering cluster dievaluasi ulang untuk skala naik atau turun	10 detik
max-empty-bulk-delete	Jumlah maksimum node kosong yang dapat dihapus secara bersamaan.	10
scale-down-delay-after-tambahkan	Berapa lama setelah skala naik, evaluasi skala ke bawah dilanjutkan	10 menit
scale-down-delay-after-hapus	Berapa lama setelah penghapusan node yang menurunkan evaluasi dilanjutkan, default ke interval pemindaian	interval pemindaian
scale-down-delay-after-kegagalan	Berapa lama setelah penurunan skala kegagalan yang menurunkan evaluasi dilanjutkan	3 menit
scale-down-unneeded-time	Berapa lama sebuah node seharusnya tidak dibutuhkan sebelum memenuhi syarat untuk menurunkan skala	10 menit
scale-down-unready-time	Berapa lama node yang belum siap seharusnya tidak dibutuhkan sebelum	20 menit

Parameter	Deskripsi	Default
	memenuhi syarat untuk menurunkan skala	
scale-down-utilization-threshold	Tingkat pemanfaatan node, didefinisikan sebagai jumlah sumber daya yang diminta dibagi dengan kapasitas , di bawahnya node dapat dipertimbangkan untuk menurunkan skala	0,5
scale-down-non-empty-kandidat-hitung	Jumlah maksimum node yang tidak kosong dipertimbangkan dalam satu iterasi sebagai kandidat untuk menurunkan skala dengan drain. Nilai yang lebih rendah berarti respons CA yang lebih baik tetapi kemungkinan penurunan latensi skala lebih lambat. Nilai yang lebih tinggi dapat mempengaruhi kinerja CA dengan cluster besar (ratusan node). Setel ke nilai non positif untuk mematikan heuristik ini - CA tidak akan membatasi jumlah node yang dianggapnya. ”	30

Parameter	Deskripsi	Default
scale-down-candidates-pool-rasio	Rasio node yang dianggap sebagai kandidat tambahan yang tidak kosong untuk menurunkan skala ketika beberapa kandidat dari iterasi sebelumnya tidak lagi valid. Nilai yang lebih rendah berarti respons CA yang lebih baik tetapi kemungkinan penurunan latensi skala lebih lambat. Nilai yang lebih tinggi dapat mempengaruhi kinerja CA dengan cluster besar (ratusan node). Setel ke 1.0 untuk menonaktifkan heuristik ini - CA akan mengambil semua node sebagai kandidat tambahan.	0.1
scale-down-candidates-pool-min-hitung	Jumlah minimum node yang dianggap sebagai kandidat tambahan yang tidak kosong untuk menurunkan skala ketika beberapa kandidat dari iterasi sebelumnya tidak lagi valid. Saat menghitung ukuran kolam untuk kandidat tambahan yang kami ambil `max(#nodes * scale-down-candidates-pool-ratio, scale-down-candidates-pool-min-count)`	50

Sumber Daya Tambahan

Halaman ini berisi daftar presentasi dan demo Cluster Autoscaler. Jika Anda ingin menambahkan presentasi atau demo di sini, silakan kirim permintaan tarik.

Presentasi/Demo	Penyaji
Penskalaan Otomatis dan Optimalisasi Biaya di Kubernetes: Dari 0 hingga 100	Guy Templeton, Skyscanner & Jiaxin Shan, Amazon
Penyelaman Dalam Penskalaan Otomatis SIGG	Maciek Pytel & Marcin Wielgus

Referensi

- https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md
- https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/aws/README.md
- https://github.com/aws/amazon-ec2-pemilih instans
- https://github.com/aws/aws-node-termination-handler

Praktik Terbaik untuk Keandalan

Bagian ini memberikan panduan tentang membuat beban kerja berjalan di EKS tangguh dan sangat tersedia

Cara menggunakan panduan ini

Panduan ini ditujukan untuk pengembang dan arsitek yang ingin mengembangkan dan mengoperasikan layanan yang sangat tersedia dan toleran terhadap kesalahan di EKS. Panduan ini diatur ke dalam area topik yang berbeda untuk konsumsi yang lebih mudah. Setiap topik dimulai dengan ikhtisar singkat, diikuti dengan daftar rekomendasi dan praktik terbaik untuk keandalan kluster EKS Anda.

Pengantar

Praktik terbaik keandalan untuk EKS telah dikelompokkan di bawah topik-topik berikut:

- Aplikasi
- Bidang Kontrol
- Bidang Data

Apa yang membuat sistem dapat diandalkan? Jika suatu sistem dapat berfungsi secara konsisten dan memenuhi tuntutan terlepas dari perubahan lingkungannya selama periode waktu tertentu, itu dapat disebut andal. Untuk mencapai hal ini, sistem harus mendeteksi kegagalan, secara otomatis menyembuhkan dirinya sendiri, dan memiliki kemampuan untuk skala berdasarkan permintaan.

Pelanggan dapat menggunakan Kubernetes sebagai fondasi untuk mengoperasikan aplikasi dan layanan mission-critical dengan andal. Namun selain menggabungkan prinsip desain aplikasi berbasis kontainer, menjalankan beban kerja dengan andal juga membutuhkan infrastruktur yang andal. Di Kubernetes, infrastruktur terdiri dari bidang kontrol dan pesawat data.

EKS menyediakan pesawat kontrol Kubernetes tingkat produksi yang dirancang agar sangat tersedia dan toleran terhadap kesalahan.

Di EKS, AWS bertanggung jawab atas keandalan bidang kontrol Kubernetes. EKS menjalankan bidang kontrol Kubernetes di tiga zona ketersediaan di Wilayah AWS. Ini secara otomatis mengelola ketersediaan dan skalabilitas server API Kubernetes dan cluster etcd.

Tanggung jawab atas keandalan pesawat data dibagi antara Anda, pelanggan, dan AWS. EKS menawarkan empat opsi node pekerja untuk menerapkan bidang data Kubernetes.

Mode Otomatis EKS, yang merupakan opsi paling terkelola, menangani penyediaan, penskalaan, dan pembaruan bidang data bersama dengan menyediakan kemampuan Compute, Networking, dan Storage yang dikelola. Mode Otomatis AMIs sering dirilis dan cluster diperbarui ke AMI terbaru secara otomatis untuk menerapkan perbaikan CVE dan patch keamanan. Anda memiliki kemampuan untuk mengontrol kapan ini terjadi dengan mengonfigurasi kontrol gangguan pada Mode Otomatis Anda. NodePools

Fargate menangani provisioning dan scaling data plane dengan menjalankan satu Pod per Node. Opsi ketiga, grup node terkelola, menangani penyediaan, dan pembaruan bidang data. Dan akhirnya, node yang dikelola sendiri adalah opsi yang paling tidak dikelola untuk bidang data. Semakin banyak pesawat data yang dikelola AWS yang Anda gunakan, semakin sedikit tanggung jawab yang Anda miliki.

Grup node terkelola mengotomatiskan penyediaan dan manajemen siklus hidup node EC2. Anda dapat menggunakan EKS API (menggunakan konsol EKS, AWS API, AWS CLI,, TerraformCloudFormation, atau`eksctl`), untuk membuat, menskalakan, dan meningkatkan node terkelola. Node terkelola menjalankan instans Amazon Linux 2 EC2 yang dioptimalkan EKS di akun Anda, dan Anda dapat menginstal paket perangkat lunak khusus dengan mengaktifkan akses SSH. Saat Anda menyediakan node terkelola, node tersebut berjalan sebagai bagian dari Grup Auto Scaling yang dikelola EKS yang dapat menjangkau beberapa Availability Zone; Anda mengontrol ini melalui subnet yang Anda berikan saat membuat node terkelola. EKS juga secara otomatis menandai node yang dikelola sehingga dapat digunakan dengan Cluster Autoscaler.

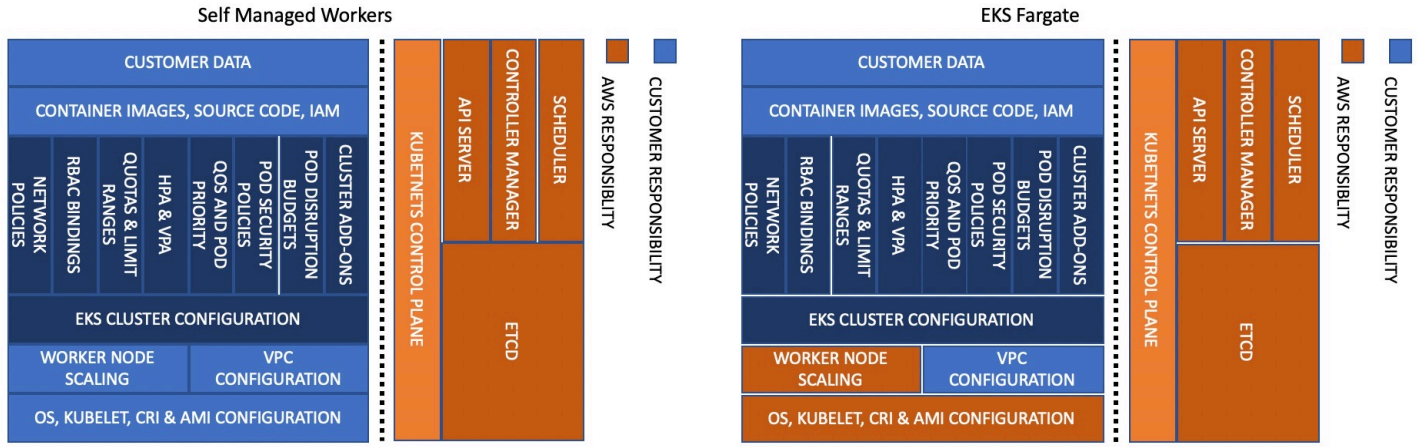
Amazon EKS mengikuti model tanggung jawab bersama untuk patch keamanan CVEs dan pada grup node terkelola. Karena node terkelola menjalankan Amazon EKS yang dioptimalkan, AMIs Amazon EKS bertanggung jawab untuk membuat versi yang ditambal AMIs saat perbaikan bug. Namun, Anda bertanggung jawab untuk menerapkan versi AMI yang ditambal ini ke grup node terkelola Anda.

EKS juga mengelola pembaruan node meskipun Anda harus memulai proses pembaruan. Proses memperbarui node terkelola dijelaskan dalam dokumentasi EKS.

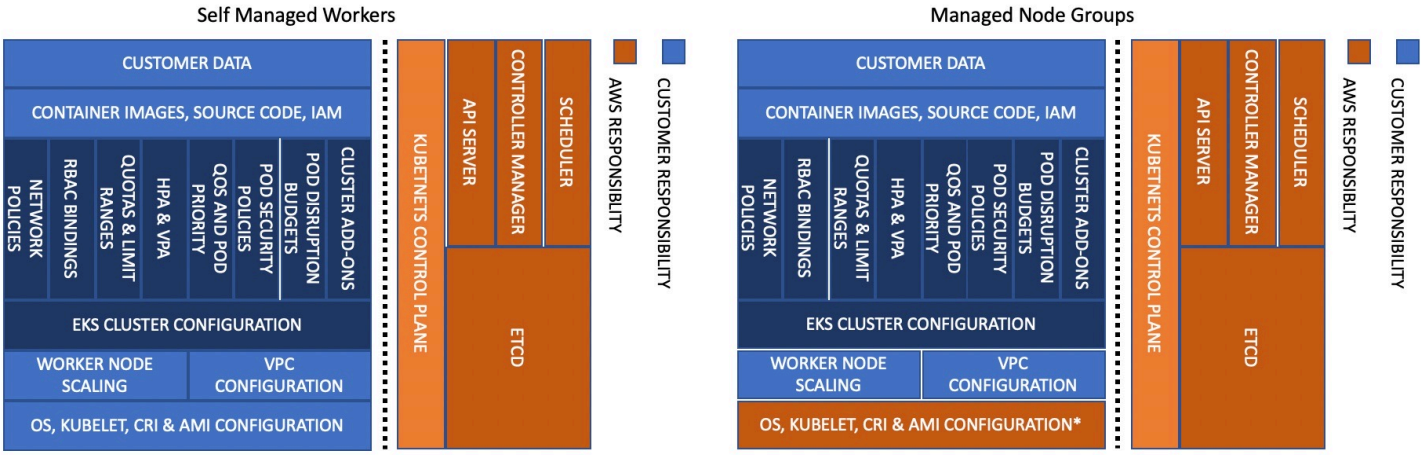
Jika Anda menjalankan node yang dikelola sendiri, Anda dapat menggunakan AMI Linux yang dioptimalkan Amazon EKS untuk membuat node pekerja. Anda bertanggung jawab untuk menambal dan meningkatkan AMI dan node. Ini adalah praktik terbaik untuk menggunakan`eksctl`, CloudFormation, atau infrastruktur sebagai alat kode untuk menyediakan node yang dikelola sendiri karena ini akan memudahkan Anda untuk meningkatkan node yang dikelola sendiri. Pertimbangkan

untuk bermigrasi ke node baru saat memperbarui node pekerja karena proses migrasi menodai grup node lama `NoSchedule` dan menguras node setelah tumpukan baru siap menerima beban kerja pod yang ada. Namun, Anda juga dapat melakukan peningkatan node yang dikelola sendiri di tempat.

Model Tanggung Jawab Bersama - Fargate



Model Tanggung Jawab Bersama - MNG



Panduan ini mencakup serangkaian rekomendasi yang dapat Anda gunakan untuk meningkatkan keandalan bidang data EKS Anda, komponen inti Kubernetes, dan aplikasi Anda.

Umpan Balik

Panduan ini dirilis GitHub untuk mengumpulkan umpan balik dan saran langsung dari EKS/ Kubernetes komunitas yang lebih luas. Jika Anda memiliki praktik terbaik yang menurut Anda harus kami sertakan dalam panduan ini, harap ajukan masalah atau kirimkan PR di GitHub repositori. Kami

bermaksud untuk memperbarui panduan secara berkala karena fitur baru ditambahkan ke layanan atau ketika praktik terbaik baru berkembang.

Menjalankan aplikasi yang sangat tersedia

Pelanggan Anda mengharapkan aplikasi Anda selalu tersedia, termasuk ketika Anda membuat perubahan dan terutama selama lonjakan lalu lintas. Arsitektur yang skalabel dan tangguh membuat aplikasi dan layanan Anda berjalan tanpa gangguan, yang membuat pengguna Anda senang. Infrastruktur yang terukur tumbuh dan menyusut berdasarkan kebutuhan bisnis. Menghilangkan satu titik kegagalan adalah langkah penting untuk meningkatkan ketersediaan aplikasi dan membuatnya tangguh.

Dengan Kubernetes, Anda dapat mengoperasikan aplikasi dan menjalankannya dengan cara yang sangat tersedia dan tangguh. Manajemen deklaratifnya memastikan bahwa setelah Anda menyiapkan aplikasi, Kubernetes akan terus mencoba mencocokkan status saat ini dengan status yang diinginkan.

Rekomendasi

Konfigurasikan Anggaran Gangguan Pod

Anggaran Gangguan Pod digunakan untuk membatasi jumlah gangguan bersamaan yang akan dialami aplikasi. Mereka harus dikonfigurasi untuk beban kerja jika penting untuk selalu memiliki sebagian dari beban kerja yang tersedia. EKS Auto Mode, Karpenter, dan Cluster Autoscaler menyadari dan mematuhi Anggaran Gangguan Pod yang dikonfigurasi saat menurunkan skala. Mode Otomatis EKS, Karpenter, dan Grup Node Terkelola juga mematuhi Anggaran Gangguan Pod saat memperbarui Node

Hindari menjalankan Pod tunggal

Jika seluruh aplikasi Anda berjalan dalam satu Pod, maka aplikasi Anda tidak akan tersedia jika Pod tersebut dihentikan. Alih-alih menerapkan aplikasi menggunakan pod individual, buat Deployment. Jika Pod yang dibuat oleh Deployment gagal atau dihentikan, pengontrol Deployment akan memulai sebuah pod baru untuk memastikan jumlah Pod replika yang ditentukan selalu berjalan.

Jalankan beberapa replika

Menjalankan beberapa replika Pod aplikasi menggunakan Deployment membantunya berjalan dengan cara yang sangat tersedia. Jika salah satu replika gagal, replika yang tersisa akan tetap berfungsi, meskipun pada kapasitas berkurang sampai Kubernetes membuat Pod lain untuk

menebus kerugian. Selanjutnya, Anda dapat menggunakan Horizontal Pod Autoscaler untuk menskalakan replika secara otomatis berdasarkan permintaan beban kerja.

Jadwalkan replika di seluruh node

Menjalankan beberapa replika tidak akan sangat berguna jika semua replika berjalan pada node yang sama, dan node menjadi tidak tersedia. Pertimbangkan untuk menggunakan kendala penyebaran anti-afinitas pod atau topologi pod untuk menyebarkan replika Deployment di beberapa node pekerja.

Anda dapat lebih meningkatkan keandalan aplikasi tipikal dengan menjalankannya di beberapa AZs.

Menggunakan aturan anti-afinitas Pod

Manifes di bawah ini memberi tahu penjadwal Kubernetes untuk memilih menempatkan pod pada node terpisah dan. AZs Itu tidak memerlukan node atau AZ yang berbeda karena jika itu terjadi, maka Kubernetes tidak akan dapat menjadwalkan pod apa pun setelah ada pod yang berjalan di setiap AZ. Jika aplikasi Anda hanya membutuhkan tiga replika, Anda dapat menggunakannya `requiredDuringSchedulingIgnoredDuringExecution` untuk`topologyKey: topology.kubernetes.io/zone`, dan penjadwal Kubernetes tidak akan menjadwalkan dua pod dalam AZ yang sama.

```
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
```

Menggunakan kendala penyebaran topologi Pod

Mirip dengan aturan anti-afinitas pod, kendala penyebaran topologi pod memungkinkan Anda membuat aplikasi Anda tersedia di berbagai domain kegagalan (atau topologi) seperti host atau. AZs Pendekatan ini bekerja sangat baik ketika Anda mencoba memastikan toleransi kesalahan serta ketersediaan dengan memiliki beberapa replika di setiap domain topologi yang berbeda. Aturan anti-afinitas pod, di sisi lain, dapat dengan mudah menghasilkan hasil di mana Anda memiliki replika tunggal dalam domain topologi karena pod dengan anti-afinitas terhadap satu sama lain memiliki efek menolak. Dalam kasus seperti itu, replika tunggal pada node khusus tidak ideal untuk toleransi kesalahan juga bukan penggunaan sumber daya yang baik. Dengan kendala penyebaran topologi, Anda memiliki kontrol lebih besar atas penyebaran atau distribusi yang harus coba diterapkan oleh penjadwal di seluruh domain topologi. Berikut adalah beberapa properti penting untuk digunakan dalam pendekatan ini:

1. `maxSkew`Ini digunakan untuk mengontrol atau menentukan titik maksimum di mana hal-hal dapat tidak merata di seluruh domain topologi. Misalnya, jika aplikasi memiliki 10 replika dan digunakan di 3 AZs, Anda tidak bisa mendapatkan spread yang merata, tetapi Anda dapat memengaruhi seberapa tidak merata distribusinya. Dalam hal ini, `maxSkew` bisa apa saja antara 1 dan 10. Nilai 1 berarti Anda berpotensi berakhir dengan spread seperti`4,3,3`, `3,4,3` atau `3,3,4` di 3 AZs.

Sebaliknya, nilai 10 berarti Anda berpotensi berakhir dengan spread seperti`10,0,0`, `0,10,0` atau `0,0,10` di 3 AZs.

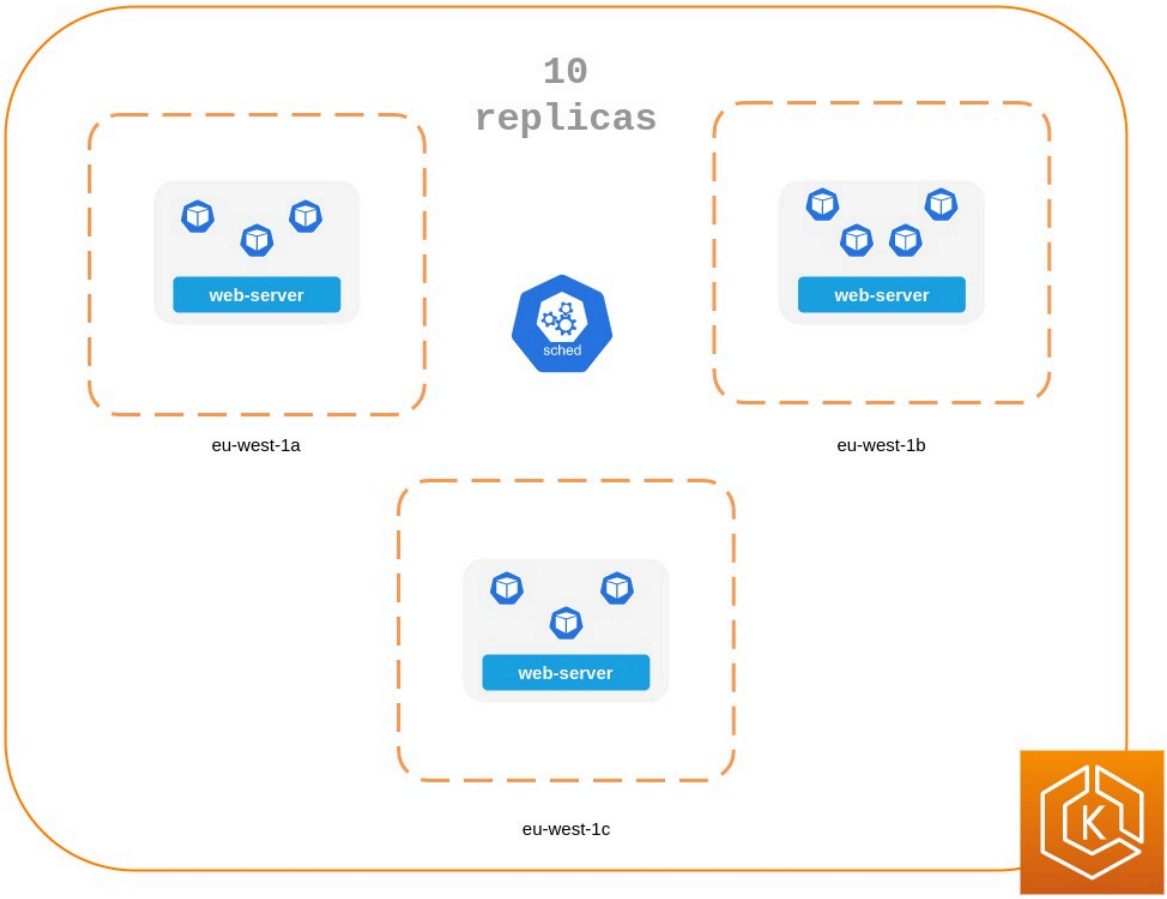
2. `topologyKey`Ini adalah kunci untuk salah satu label node dan mendefinisikan jenis domain topologi yang harus digunakan untuk distribusi pod. Misalnya, spread zona akan memiliki pasangan kunci-nilai berikut:

```
topologyKey: "topology.kubernetes.io/zone"
```

3. `whenUnsatisfiable`Properti ini digunakan untuk menentukan bagaimana Anda ingin penjadwal merespons jika kendala yang diinginkan tidak dapat dipenuhi.
4. Pod `labelSelector` digunakan untuk menemukan pod yang cocok sehingga scheduler dapat mengetahuinya saat memutuskan di mana menempatkan Pod sesuai dengan batasan yang Anda tentukan.

Selain ini di atas, ada bidang lain yang dapat Anda baca lebih lanjut dalam dokumentasi Kubernetes.

Topologi pod menyebarkan kendala di 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
```

Jalankan Kubernetes Metrics Server

Instal server metrik Kubernetes untuk membantu menskalakan aplikasi Anda. Add-on autoscaler Kubernetes seperti HPA dan VPA perlu melacak metrik aplikasi untuk menskalakannya. Server metrik mengumpulkan metrik sumber daya yang dapat digunakan untuk membuat keputusan penskalaan. Metrik dikumpulkan dari kubelet dan disajikan dalam format Metrik API.

Server metrik tidak menyimpan data apa pun, dan ini bukan solusi pemantauan. Tujuannya adalah untuk mengekspos metrik penggunaan CPU dan memori ke sistem lain. Jika Anda ingin melacak status aplikasi Anda dari waktu ke waktu, Anda memerlukan alat pemantauan seperti Prometheus atau Amazon. CloudWatch

Ikuti dokumentasi EKS untuk menginstal metrik-server di kluster EKS Anda.

Penskaler Otomatis Pod Horisontal (HPA)

HPA dapat secara otomatis menskalakan aplikasi Anda sebagai respons terhadap permintaan dan membantu Anda menghindari dampak pelanggan Anda selama lalu lintas puncak. Ini diimplementasikan sebagai loop kontrol di Kubernetes yang secara berkala menanyakan metrik yang menyediakan metrik sumber APIs daya.

HPA dapat mengambil metrik dari berikut ini: APIs 1. `metrics.k8s.io`Juga dikenal sebagai Resource Metrics API — Menyediakan penggunaan CPU dan memori untuk pod 2. `custom.metrics.k8s.io` — Menyediakan metrik dari kolektor metrik lain seperti Prometheus; metrik ini bersifat internal untuk klaster Kubernetes Anda. 3. `external.metrics.k8s.io` — Menyediakan metrik yang berada di luar klaster Kubernetes Anda (Misalnya, SQS Queue Depth, ELB latency).

Anda harus menggunakan salah satu dari ketiganya APIs untuk memberikan metrik untuk menskalakan aplikasi Anda.

Menskalakan aplikasi berdasarkan metrik khusus atau eksternal

Anda dapat menggunakan metrik khusus atau eksternal untuk menskalakan aplikasi Anda pada metrik selain pemanfaatan CPU atau memori. Server API Metrik Kustom menyediakan `custom-metrics.k8s.io` API yang dapat digunakan HPA untuk aplikasi skala otomatis.

Anda dapat menggunakan Adaptor Prometheus untuk Metrik Kubernetes untuk mengumpulkan APIs metrik dari Prometheus dan digunakan dengan HPA. Dalam hal ini, adaptor Prometheus akan mengekspos metrik Prometheus dalam format Metrik API.

Setelah Anda menerapkan Adaptor Prometheus, Anda dapat melakukan kueri metrik kustom menggunakan kubectl. `kubectl get –raw /apis/custom.metrics.k8s.io/v1beta1/`

Metrik eksternal, seperti namanya, memberikan Horizontal Pod Autoscaler kemampuan untuk menskalakan deployment menggunakan metrik yang berada di luar klaster Kubernetes. Misalnya, dalam beban kerja pemrosesan batch, adalah umum untuk menskalakan jumlah replika berdasarkan jumlah pekerjaan dalam penerbangan dalam antrian SQS.

Untuk melakukan autoscale beban kerja Kubernetes, Anda dapat menggunakan KEDA (Kubernetes Event-driven Autoscaling), sebuah proyek sumber terbuka yang dapat mendorong penskalaan kontainer berdasarkan sejumlah peristiwa khusus. Blog AWS ini menguraikan cara menggunakan Amazon Managed Service untuk Prometheus untuk auto-scaling beban kerja Kubernetes.

Autoscaler Pod Vertikal (VPA)

VPA secara otomatis menyesuaikan reservasi CPU dan memori untuk Pod Anda untuk membantu Anda “ukuran yang tepat” aplikasi Anda. Untuk aplikasi yang perlu diskalakan secara vertikal - yang dilakukan dengan meningkatkan alokasi sumber daya - Anda dapat menggunakan VPA untuk secara otomatis menskalakan replika Pod atau memberikan rekomendasi penskalaan.

Aplikasi Anda mungkin menjadi tidak tersedia untuk sementara jika VPA perlu menskalakannya karena implementasi VPA saat ini tidak melakukan penyesuaian di tempat pada Pod; sebaliknya, itu akan membuat ulang Pod yang perlu diskalakan.

Dokumentasi EKS mencakup panduan untuk menyiapkan VPA.

Proyek Fairwinds Goldilocks menyediakan dasbor untuk memvisualisasikan rekomendasi VPA untuk permintaan dan batasan CPU dan memori. Mode pembaruan VPA-nya memungkinkan Anda untuk menskalakan Pod secara otomatis berdasarkan rekomendasi VPA.

Memperbarui aplikasi

Aplikasi modern membutuhkan inovasi cepat dengan tingkat stabilitas dan ketersediaan yang tinggi. Kubernetes memberi Anda alat untuk memperbarui aplikasi Anda secara terus menerus tanpa mengganggu pelanggan Anda.

Mari kita lihat beberapa praktik terbaik yang memungkinkan penerapan perubahan dengan cepat tanpa mengorbankan ketersediaan.

Memiliki mekanisme untuk melakukan rollback

Memiliki tombol undo dapat menghindari bencana. Ini adalah praktik terbaik untuk menguji penerapan di lingkungan bawah yang terpisah (lingkungan pengujian atau pengembangan) sebelum memperbarui cluster produksi. Menggunakan pipa CI/CD dapat membantu Anda mengotomatiskan dan menguji penerapan. Dengan pipeline penerapan berkelanjutan, Anda dapat dengan cepat kembali ke versi yang lebih lama jika pemutakhiran terjadi rusak.

Anda dapat menggunakan Deployment untuk memperbarui aplikasi yang sedang berjalan. Ini biasanya dilakukan dengan memperbarui gambar kontainer. Anda dapat menggunakan `kubectl` untuk memperbarui Deployment seperti ini:

```
kubectl --record deployment.apps/nginx-deployment set image nginx-deployment
 nginx=nginx:1.16.1
```

`--record`Argumen merekam perubahan pada Deployment dan membantu Anda jika Anda perlu melakukan rollback. `kubectl rollout history deployment`menunjukkan kepada Anda perubahan yang direkam pada Deployment di klaster Anda. Anda dapat mengembalikan perubahan menggunakan. `kubectl rollout undo deployment <DEPLOYMENT_NAME>`

Secara default, saat Anda memperbarui Deployment yang memerlukan rekreasi pod, Deployment akan melakukan pembaruan bergulir. Dengan kata lain, Kubernetes hanya akan memperbarui sebagian dari Pod yang sedang berjalan dalam Deployment dan tidak semua Pod sekaligus. Anda dapat mengontrol bagaimana Kubernetes melakukan pembaruan bergulir melalui properti. `RollingUpdateStrategy`

Saat melakukan pembaruan bergulir dari Deployment, Anda dapat menggunakan `Max Unavailable`properti untuk menentukan jumlah maksimum Pod yang tidak tersedia selama pembaruan. `Max Surge`Properti Deployment memungkinkan Anda untuk mengatur jumlah maksimum Pod yang dapat dibuat di atas jumlah Pod yang diinginkan.

Pertimbangkan `max unavailable` untuk menyesuaikan untuk memastikan bahwa peluncuran tidak mengganggu pelanggan Anda. Misalnya, Kubernetes menetapkan 25% secara `max unavailable` default, yang berarti jika Anda memiliki 100 Pod, Anda mungkin hanya memiliki 75 Pod yang aktif bekerja selama peluncuran. Jika aplikasi Anda membutuhkan minimal 80 Pod, peluncuran ini dapat mengganggu. Sebagai gantinya, Anda dapat mengatur `max unavailable` hingga 20% untuk memastikan bahwa setidaknya ada 80 Pod fungsional selama peluncuran.

Gunakan blue/green penerapan

Perubahan pada dasarnya berisiko, tetapi perubahan yang tidak dapat dibatalkan dapat berpotensi menjadi bencana besar. Ubah prosedur yang memungkinkan Anda memutar kembali waktu secara efektif melalui rollback membuat penyempurnaan dan eksperimen lebih aman. Blue/green penerapan memberi Anda metode untuk menarik kembali perubahan dengan cepat jika terjadi kesalahan. Dalam strategi penyebaran ini, Anda membuat lingkungan untuk versi baru. Lingkungan ini identik dengan versi aplikasi saat ini yang sedang diperbarui. Setelah lingkungan baru disediakan, lalu lintas diarahkan ke lingkungan baru. Jika versi baru menghasilkan hasil yang diinginkan tanpa menghasilkan kesalahan, lingkungan lama dihentikan. Jika tidak, lalu lintas dikembalikan ke versi lama.

Anda dapat melakukan blue/green deployment di Kubernetes dengan membuat Deployment baru yang identik dengan Deployment versi yang ada. Setelah Anda memverifikasi bahwa Pod dalam Deployment baru berjalan tanpa kesalahan, Anda dapat mulai mengirim lalu lintas ke Deployment baru dengan mengubah `selector` spesifikasi di Layanan yang merutekan lalu lintas ke Pod aplikasi Anda.

Banyak alat integrasi berkelanjutan seperti Flux, Jenkins, dan Spinnaker memungkinkan Anda mengotomatiskan penerapan. blue/green AWS Containers Blog menyertakan panduan menggunakan AWS Load Balancer Controller: Menggunakan AWS Load Balancer Controller blue/green untuk penerapan, penerapan canary, dan pengujian A/B

Gunakan penerapan Canary

Penerapan Canary adalah varian blue/green penerapan yang dapat secara signifikan menghilangkan risiko dari perubahan. Dalam strategi penerapan ini, Anda membuat Deployment baru dengan Pod yang lebih sedikit di samping Deployment lama Anda, dan mengalihkan sebagian kecil lalu lintas ke Deployment baru. Jika metrik menunjukkan bahwa versi baru berkinerja baik atau lebih baik dari versi yang ada, Anda secara progresif meningkatkan lalu lintas ke Deployment baru sambil menskalakannya hingga semua lalu lintas dialihkan ke Deployment baru. Jika ada masalah,

Anda dapat merutekan semua lalu lintas ke Deployment lama dan berhenti mengirim lalu lintas ke Deployment baru.

Meskipun Kubernetes tidak menawarkan cara asli untuk melakukan penerapan kenari, Anda dapat menggunakan alat seperti Flagger dengan Istio.

Pemeriksaan Kesehatan dan Penyembuhan Diri

Tidak ada perangkat lunak yang bebas bug, tetapi Kubernetes dapat membantu Anda meminimalkan dampak kegagalan perangkat lunak. Di masa lalu, jika aplikasi macet, seseorang harus memperbaiki situasi dengan memulai ulang aplikasi secara manual. Kubernetes memberi Anda kemampuan untuk mendeteksi kegagalan perangkat lunak di Pod Anda dan secara otomatis menggantinya dengan replika baru. Dengan Kubernetes Anda dapat memantau kesehatan aplikasi Anda dan secara otomatis mengganti instans yang tidak sehat.

Kubernetes mendukung tiga jenis pemeriksaan kesehatan:

1. Probe keaktifan
2. Probe startup (didukung di Kubernetes versi 1.16+)
3. Probe kesiapan

Kubelet, agen Kubernetes, bertanggung jawab untuk menjalankan semua pemeriksaan yang disebutkan di atas. Kubelet dapat memeriksa kesehatan Pods dengan tiga cara: kubelet dapat menjalankan perintah shell di dalam container Pod, mengirim permintaan HTTP GET ke kontainernya, atau membuka soket TCP pada port tertentu.

Jika Anda memilih probe `exec` berbasis, yang menjalankan skrip shell di dalam wadah, pastikan perintah shell keluar sebelum `timeoutSeconds` nilainya kedaluwarsa. Jika tidak, node Anda akan memiliki `<defunct>` proses, yang mengarah ke kegagalan node.

Rekomendasi

Gunakan Liveness Probe untuk menghilangkan pod yang tidak sehat

Probe Liveness dapat mendeteksi kondisi kebuntuan di mana proses terus berjalan, tetapi aplikasi menjadi tidak responsif. Misalnya, jika Anda menjalankan layanan web yang mendengarkan pada port 80, Anda dapat mengonfigurasi probe Liveness untuk mengirim permintaan HTTP GET pada port 80 Pod. Kubelet akan secara berkala mengirimkan permintaan GET ke Pod dan mengharapkan respons; jika Pod merespons antara 200-399 maka kubelet menganggap bahwa Pod sehat; jika

tidak, Pod akan ditandai sebagai tidak sehat. Jika Pod gagal melakukan pemeriksaan kesehatan secara terus menerus, kubelet akan menghentikannya.

Anda dapat menggunakan `initialDelaySeconds` untuk menunda probe pertama.

Saat menggunakan Liveness Probe, pastikan aplikasi Anda tidak mengalami situasi di mana semua Pod secara bersamaan gagal dalam Liveness Probe karena Kubernetes akan mencoba mengganti semua Pod Anda, yang akan membuat aplikasi Anda offline. Selanjutnya, Kubernetes akan terus membuat Pod baru yang juga akan gagal Liveness Probe, menempatkan ketegangan yang tidak perlu pada bidang kontrol. Hindari mengonfigurasi Liveness Probe agar bergantung pada faktor eksternal Pod Anda, misalnya database eksternal. Dengan kata lain, external-to-your-Pod database yang tidak responsif seharusnya tidak membuat Pod Anda gagal dalam Liveness Probe mereka.

Posting Sandor Szücs LIVENESS PROBES ARE DANGEROUS menjelaskan masalah yang dapat disebabkan oleh probe yang salah konfigurasi.

Gunakan Startup Probe untuk aplikasi yang membutuhkan waktu lebih lama untuk memulai

Saat aplikasi membutuhkan waktu tambahan untuk memulai, Anda dapat menggunakan Startup Probe untuk menunda Liveness and Readiness Probe. Misalnya, aplikasi Java yang perlu menghidrasi cache dari database mungkin memerlukan waktu hingga dua menit sebelum berfungsi penuh. Setiap Liveness atau Readiness Probe sampai berfungsi penuh mungkin gagal. Mengkonfigurasi Probe Startup akan memungkinkan aplikasi Java menjadi sehat sebelum Liveness atau Readiness Probe dijalankan.

Sampai Startup Probe berhasil, semua Probe lainnya dinonaktifkan. Anda dapat menentukan waktu maksimum Kubernetes harus menunggu startup aplikasi. Jika, setelah waktu maksimum yang dikonfigurasi, Pod masih gagal Startup Probe, maka akan dihentikan, dan Pod baru akan dibuat.

Startup Probe mirip dengan Liveness Probe — jika gagal, Pod dibuat ulang. Seperti yang dijelaskan Ricardo A. dalam postingannya Fantastic Probes And How To Configuration Them, Startup Probe harus digunakan ketika waktu startup suatu aplikasi tidak dapat diprediksi. Jika Anda tahu aplikasi Anda membutuhkan sepuluh detik untuk memulai, Anda harus menggunakan Liveness/Readiness Probe sebagai gantinya. `initialDelaySeconds`

Gunakan Readiness Probe untuk mendeteksi ketidaktersediaan sebagian

Sementara probe Liveness mendeteksi kegagalan dalam aplikasi yang diselesaikan dengan menghentikan Pod (karenanya, memulai ulang aplikasi), Readiness Probe mendeteksi kondisi di

mana aplikasi mungkin tidak tersedia untuk sementara. Dalam situasi ini, aplikasi mungkin menjadi tidak responsif sementara; Namun, diharapkan akan sehat kembali setelah operasi ini selesai.

Misalnya, selama I/O operasi disk yang intens, aplikasi mungkin sementara tidak tersedia untuk menangani permintaan. Di sini, menghentikan Pod aplikasi bukanlah solusi; pada saat yang sama, permintaan tambahan yang dikirim ke Pod dapat gagal.

Anda dapat menggunakan Readiness Probe untuk mendeteksi ketidaktersediaan sementara di aplikasi Anda dan berhenti mengirim permintaan ke Pod-nya hingga menjadi berfungsi kembali. Tidak seperti Liveness Probe, di mana kegagalan akan mengakibatkan rekreasi Pod, Readiness Probe yang gagal berarti bahwa Pod tidak akan menerima lalu lintas apa pun dari Layanan Kubernetes. Ketika Readiness Probe berhasil, Pod akan melanjutkan menerima lalu lintas dari Service.

Sama seperti Liveness Probe, hindari mengonfigurasi Readiness Probe yang bergantung pada sumber daya yang berada di luar Pod (seperti database). Berikut adalah skenario di mana Kesiapan yang dikonfigurasi dengan buruk dapat membuat aplikasi tidak berfungsi - jika Probe Kesiapan Pod gagal ketika database aplikasi tidak dapat dijangkau, replika Pod lainnya juga akan gagal secara bersamaan karena mereka memiliki kriteria pemeriksaan kesehatan yang sama. Menyetel probe dengan cara ini akan memastikan bahwa kapan pun database tidak tersedia, Probe Kesiapan Pod akan gagal, dan Kubernetes akan menghentikan pengiriman lalu lintas ke semua Pod.

Efek samping dari penggunaan Probe Kesiapan adalah bahwa mereka dapat meningkatkan waktu yang diperlukan untuk memperbarui Deployment. Replika baru tidak akan menerima lalu lintas kecuali Probe Kesiapan berhasil; sampai saat itu, replika lama akan terus menerima lalu lintas.

Berurusan dengan gangguan

Pod memiliki masa pakai yang terbatas - bahkan jika Anda memiliki Pod yang berjalan lama, sebaiknya pastikan Pod dihentikan dengan benar ketika saatnya tiba. Bergantung pada strategi upgrade Anda, upgrade klaster Kubernetes mungkin mengharuskan Anda untuk membuat node pekerja baru, yang mengharuskan semua Pod dibuat ulang pada node yang lebih baru. Penanganan penghentian yang tepat dan Anggaran Gangguan Pod dapat membantu Anda menghindari gangguan layanan karena Pod dihapus dari node yang lebih lama dan dibuat ulang pada node yang lebih baru.

Cara yang lebih disukai untuk memutakhirkan node pekerja adalah dengan membuat node pekerja baru dan menghentikan yang lama. Sebelum menghentikan node pekerja, Anda harus `drain` melakukannya. Ketika node pekerja terkuras, semua polong-nya akan diusir dengan aman. Aman adalah kata kunci di sini; ketika pod pada pekerja diusir, mereka tidak hanya dikirim sinyal`SIGKILL`. Sebagai gantinya, `SIGTERM` sinyal dikirim ke proses utama (PID 1) dari setiap kontainer dalam

Pod yang diusir. Setelah `SIGTERM` sinyal dikirim, Kubernetes akan memberikan proses beberapa waktu (masa tenggang) sebelum `SIGKILL` sinyal dikirim. Masa tenggang ini adalah 30 detik secara default; Anda dapat mengganti default dengan menggunakan `grace-period` flag di kubectl atau deklarasikan `terminationGracePeriodSeconds` di Podspec Anda.

```
kubectl delete pod <pod name> –grace-period=<seconds>
```

Adalah umum untuk memiliki wadah di mana proses utama tidak memiliki PID 1. Pertimbangkan wadah sampel berbasis Python ini:

```
$ 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
```

Dalam contoh ini, skrip shell menerima`SIGTERM`, proses utama, yang kebetulan merupakan aplikasi Python dalam contoh ini, tidak mendapatkan sinyal`SIGTERM`. Ketika Pod dihentikan, aplikasi Python akan mati secara tiba-tiba. Ini dapat diperbaiki dengan mengubah wadah untuk meluncurkan aplikasi Python. `ENTRYPOINT` Atau, Anda dapat menggunakan alat seperti dumb-init untuk memastikan bahwa aplikasi Anda dapat menangani sinyal.

Anda juga dapat menggunakan Container hook untuk mengeksekusi script atau permintaan HTTP pada container start atau stop. Tindakan `PreStop` hook berjalan sebelum kontainer menerima `SIGTERM` sinyal dan harus selesai sebelum sinyal ini dikirim. `terminationGracePeriodSeconds`Nilai berlaku dari saat aksi `PreStop` hook mulai dijalankan, bukan saat `SIGTERM` sinyal dikirim.

Rekomendasi

Lindungi beban kerja kritis dengan Anggaran Gangguan Pod

Pod Disruption Budget atau PDB dapat menghentikan sementara proses penggusuran jika jumlah replika aplikasi turun di bawah ambang batas yang dinyatakan. Proses penggusuran akan berlanjut setelah jumlah replika yang tersedia melebihi ambang batas. Anda dapat menggunakan PDB untuk mendeklarasikan `minAvailable` dan `maxUnavailable` jumlah replika. Misalnya, jika Anda ingin setidaknya tiga salinan aplikasi tersedia, Anda dapat membuat PDB.

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
```

```
  name: my-svc-pdb
spec:
  minAvailable: 3
  selector:
    matchLabels:
      app: my-svc
```

Kebijakan PDB di atas memberitahu Kubernetes untuk menghentikan proses penggusuran sampai tiga atau lebih replika tersedia. Pengerasan simpul menghormati`PodDisruptionBudgets`. Selama upgrade grup node terkelola EKS, node dikeringkan dengan batas waktu lima belas menit. Setelah lima belas menit, jika pembaruan tidak dipaksakan (opsi ini disebut Pembaruan bergulir di konsol EKS), pembaruan gagal. Jika pembaruan dipaksakan, pod akan dihapus.

Untuk node yang dikelola sendiri, Anda juga dapat menggunakan alat seperti AWS Node Termination Handler, yang memastikan bahwa bidang kontrol Kubernetes merespons dengan tepat peristiwa yang dapat menyebabkan instans EC2 Anda menjadi tidak tersedia, seperti peristiwa pemeliharaan EC2 dan interupsi EC2 Spot. Ia menggunakan API Kubernetes untuk mengikat node untuk memastikan tidak ada Pod baru yang dijadwalkan, kemudian mengurasnya, menghentikan semua Pod yang sedang berjalan.

Anda dapat menggunakan anti-afinitas Pod untuk menjadwalkan Pod Deployment pada node yang berbeda dan menghindari penundaan terkait PDB selama upgrade node.

Berlatih rekayasa kekacauan

Chaos Engineering adalah disiplin bereksperimen pada sistem terdistribusi untuk membangun kepercayaan pada kemampuan sistem untuk menahan kondisi turbulen dalam produksi.

Dalam blognya, Dominik Tornow menjelaskan bahwa Kubernetes adalah sistem deklaratif di mana "pengguna memasok representasi dari keadaan sistem yang diinginkan ke sistem. Sistem kemudian mempertimbangkan keadaan saat ini dan keadaan yang diinginkan untuk menentukan urutan perintah untuk transisi dari keadaan saat ini ke keadaan yang diinginkan. “ Ini berarti Kubernetes selalu menyimpan status yang diinginkan dan jika sistem menyimpang, Kubernetes akan mengambil tindakan untuk memulihkan status. Misalnya, jika node pekerja menjadi tidak tersedia, Kubernetes akan menjadwal ulang Pod tersebut ke node pekerja lain. Demikian pula, jika `replica` crash, Deployment Contoller akan membuat yang baru. `replica` Dengan cara ini, pengendali Kubernetes secara otomatis memperbaiki kegagalan.

Alat rekayasa kekacauan seperti Gremlin membantu Anda menguji ketahanan cluster Kubernetes Anda dan mengidentifikasi satu titik kegagalan. Alat yang memperkenalkan kekacauan buatan di

cluster Anda (dan seterusnya) dapat mengungkap kelemahan sistemik, memberikan kesempatan untuk mengidentifikasi kemacetan dan kesalahan konfigurasi, dan memperbaiki masalah di lingkungan yang terkendali. Filosofi Chaos Engineering menganjurkan pemecahan hal-hal dengan sengaja dan infrastruktur pengujian stres untuk meminimalkan waktu henti yang tidak terduga.

Gunakan Service Mesh

Anda dapat menggunakan mesh layanan untuk meningkatkan ketahanan aplikasi Anda. Jaring layanan memungkinkan service-to-service komunikasi dan meningkatkan pengamatan jaringan layanan mikro Anda. Sebagian besar produk mesh layanan bekerja dengan menjalankan proxy jaringan kecil di samping setiap layanan yang mencegat dan memeriksa lalu lintas jaringan aplikasi. Anda dapat menempatkan aplikasi Anda di mesh tanpa memodifikasi aplikasi Anda. Dengan menggunakan fitur bawaan proxy layanan, Anda dapat membuatnya menghasilkan statistik jaringan, membuat log akses, dan menambahkan header HTTP ke permintaan keluar untuk penelusuran terdistribusi.

Service mesh dapat membantu Anda membuat layanan mikro Anda lebih tangguh dengan fitur-fitur seperti percobaan ulang permintaan otomatis, batas waktu, pemutusan sirkuit, dan pembatasan laju.

Jika Anda mengoperasikan beberapa cluster, Anda dapat menggunakan mesh layanan untuk mengaktifkan komunikasi lintas cluster service-to-service.

Jala Layanan

- Istio
- LinkerD
- Konsul

Observabilitas

Observabilitas adalah istilah umum yang mencakup pemantauan, penebangan, dan penelusuran. Aplikasi berbasis layanan mikro didistribusikan secara alami. Tidak seperti aplikasi monolitik di mana pemantauan satu sistem sudah cukup, dalam arsitektur aplikasi terdistribusi, Anda perlu memantau kinerja setiap komponen. Anda dapat menggunakan sistem pemantauan, pencatatan, dan penelusuran terdistribusi tingkat cluster untuk mengidentifikasi masalah di klaster Anda sebelum mengganggu pelanggan Anda.

Alat bawaan Kubernetes untuk pemecahan masalah dan pemantauan terbatas. Server metrik mengumpulkan metrik sumber daya dan menyimpannya dalam memori tetapi tidak bertahan. Kamu dapat melihat log dari sebuah Pod menggunakan kubectl, tetapi Kubernetes tidak secara otomatis menyimpan log. Dan implementasi penelusuran terdistribusi dilakukan baik di tingkat kode aplikasi atau menggunakan jerat layanan.

Ekstensibilitas Kubernetes bersinar di sini. Kubernetes memungkinkan Anda menghadirkan solusi pemantauan, pencatatan, dan penelusuran terpusat pilihan Anda.

Rekomendasi

Pantau aplikasi Anda

Jumlah metrik yang perlu Anda pantau dalam aplikasi modern terus bertambah. Ini membantu jika Anda memiliki cara otomatis untuk melacak aplikasi Anda sehingga Anda dapat fokus pada pemecahan tantangan pelanggan Anda. Alat pemantauan di seluruh cluster seperti Prometheus atau CloudWatchContainer Insights dapat memantau klaster dan beban kerja Anda dan memberi Anda sinyal kapan, atau lebih disukai, sebelum terjadi kesalahan.

Alat pemantauan memungkinkan Anda membuat peringatan yang dapat dilangganan oleh tim operasi Anda. Pertimbangkan aturan untuk mengaktifkan alarm untuk peristiwa yang dapat, ketika diperburuk, menyebabkan pemadaman atau memengaruhi kinerja aplikasi.

Jika Anda tidak jelas metrik mana yang harus Anda pantau, Anda dapat mengambil inspirasi dari metode ini:

- Metode RED. Singkatan dari permintaan, kesalahan, dan durasi.
- Gunakan metode. Singkatan dari pemanfaatan, saturasi, dan kesalahan.

Postingan Sysdig Praktik terbaik untuk peringatan di Kubernetes mencakup daftar lengkap komponen yang dapat memengaruhi ketersediaan aplikasi Anda.

Gunakan pustaka klien Prometheus untuk mengekspos metrik aplikasi

Selain memantau status aplikasi dan menggabungkan metrik standar, Anda juga dapat menggunakan pustaka klien Prometheus untuk mengekspos metrik khusus aplikasi khusus untuk meningkatkan observabilitas aplikasi.

Gunakan alat logging terpusat untuk mengumpulkan dan mempertahankan log

Logging di EKS termasuk dalam dua kategori: log bidang kontrol dan log aplikasi. Pencatatan bidang kontrol EKS menyediakan log audit dan diagnostik langsung dari bidang kontrol ke CloudWatch Log di akun Anda. Log aplikasi adalah log yang dihasilkan oleh Pod yang berjalan di dalam klaster Anda. Log aplikasi mencakup log yang dihasilkan oleh Pod yang menjalankan aplikasi logika bisnis dan komponen sistem Kubernetes seperti CoreDNS, Cluster Autoscaler, Prometheus, dll.

EKS menyediakan lima jenis log bidang kontrol:

1. Log komponen server API Kubernetes
2. Audit
3. Authenticator
4. Manajer pengontrol
5. Penjadwal

Manajer pengontrol dan log penjadwal dapat membantu mendiagnosis masalah bidang kontrol seperti kemacetan dan kesalahan. Secara default, log bidang kontrol EKS tidak dikirim ke CloudWatch Log. Anda dapat mengaktifkan pencatatan bidang kontrol dan memilih jenis log bidang kontrol EKS yang ingin Anda tangkap untuk setiap cluster di akun Anda

Mengumpulkan log aplikasi memerlukan pemasangan alat agregator log seperti Fluent Bit, Fluentd, atau CloudWatchContainer Insights di cluster Anda.

Alat agregator log Kubernetes berjalan sebagai DaemonSets dan mengikis log kontainer dari node. Log aplikasi kemudian dikirim ke tujuan terpusat untuk penyimpanan. Misalnya, CloudWatch Container Insights dapat menggunakan Fluent Bit atau Fluentd untuk mengumpulkan log dan mengirimkannya ke Log untuk CloudWatch penyimpanan. Fluent Bit dan Fluentd mendukung banyak sistem analisis log populer seperti Elasticsearch dan InfluxDB yang memberi Anda kemampuan untuk mengubah backend penyimpanan untuk log Anda dengan memodifikasi konfigurasi log Fluent bit atau Fluentd.

Gunakan sistem penelusuran terdistribusi untuk mengidentifikasi kemacetan

Aplikasi modern yang khas memiliki komponen yang didistribusikan melalui jaringan, dan keandalannya tergantung pada berfungsinya masing-masing komponen yang membentuk aplikasi. Anda dapat menggunakan solusi penelusuran terdistribusi untuk memahami bagaimana permintaan mengalir dan bagaimana sistem berkomunikasi. Jejak dapat menunjukkan kepada Anda di mana

kemacetan ada di jaringan aplikasi Anda dan mencegah masalah yang dapat menyebabkan kegagalan cascading.

Anda memiliki dua opsi untuk menerapkan penelusuran dalam aplikasi Anda: Anda dapat menerapkan penelusuran terdistribusi pada tingkat kode menggunakan pustaka bersama atau menggunakan mesh layanan.

Menerapkan penelusuran pada tingkat kode bisa merugikan. Dalam metode ini, Anda harus membuat perubahan pada kode Anda. Ini semakin rumit jika Anda memiliki aplikasi polyglot. Anda juga bertanggung jawab untuk memelihara perpustakaan lain, di seluruh layanan Anda.

Service Meshes seperti LinkerD dan Istio dapat digunakan untuk mengimplementasikan penelusuran terdistribusi dalam aplikasi Anda dengan sedikit perubahan pada kode aplikasi. Anda dapat menggunakan mesh layanan untuk menstandarisasi pembuatan metrik, pencatatan, dan penelusuran.

Alat penelusuran seperti AWS X-Ray, Jaeger mendukung perpustakaan bersama dan implementasi mesh layanan.

Pertimbangkan untuk menggunakan alat penelusuran seperti AWS X-Ray atau Jaeger yang mendukung implementasi keduanya (library bersama dan mesh layanan) sehingga Anda tidak perlu mengganti alat jika nanti Anda mengadopsi mesh layanan.

Pesawat Kontrol EKS

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Amazon Elastic Kubernetes Service (EKS) adalah layanan Kubernetes terkelola yang memudahkan Anda menjalankan Kubernetes di AWS tanpa perlu menginstal, mengoperasikan, dan memelihara pesawat kontrol Kubernetes atau node pekerja Anda sendiri. Ini berjalan di hulu Kubernetes dan bersertifikat kesesuaian Kubernetes. Kesesuaian ini memastikan bahwa EKS mendukung API Kubernetes, seperti versi komunitas open-source yang dapat Anda instal di EC2 atau lokal. Aplikasi yang ada yang berjalan di Kubernetes upstream kompatibel dengan Amazon EKS.

EKS secara otomatis mengelola ketersediaan dan skalabilitas node bidang kontrol Kubernetes, dan secara otomatis menggantikan node bidang kontrol yang tidak sehat.

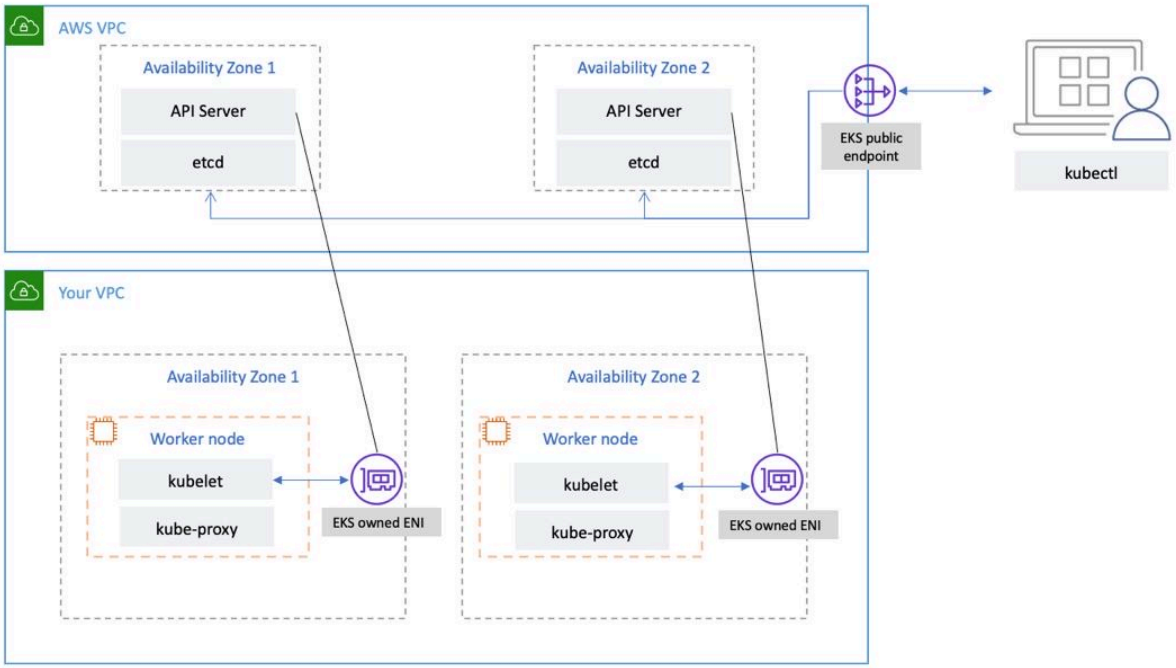
Arsitektur EKS

Arsitektur EKS dirancang untuk menghilangkan titik kegagalan tunggal yang dapat membahayakan ketersediaan dan daya tahan bidang kontrol Kubernetes.

Pesawat kontrol Kubernetes yang dikelola oleh EKS berjalan di dalam VPC yang dikelola EKS. Bidang kontrol EKS terdiri dari node server API Kubernetes, klaster etcd. Node server API Kubernetes yang menjalankan komponen seperti server API, scheduler, dan berjalan `kube-controller-manager` dalam grup auto-scaling. EKS menjalankan minimal dua node server API di Availability Zones (AZs) yang berbeda di dalam wilayah AWS. Demikian juga, untuk daya tahan, node server etcd juga berjalan dalam grup auto-scaling yang mencakup tiga. AZs EKS menjalankan NAT Gateway di setiap AZ, dan server API dan server etcd berjalan di subnet pribadi. Arsitektur ini memastikan bahwa peristiwa dalam satu AZ tidak memengaruhi ketersediaan kluster EKS.

Saat Anda membuat klaster baru, Amazon EKS membuat titik akhir yang sangat tersedia untuk server API Kubernetes terkelola yang Anda gunakan untuk berkomunikasi dengan klaster Anda (menggunakan alat seperti). `kubectl` Endpoint terkelola menggunakan NLB untuk load balance Kubernetes API server. EKS juga menyediakan dua ENI berbeda AZs untuk memfasilitasi komunikasi ke node pekerja Anda.

EKS Konektivitas jaringan pesawat data



Anda dapat mengonfigurasi apakah server API klaster Kubernetes Anda dapat dijangkau dari internet publik (menggunakan titik akhir publik) atau melalui VPC Anda (menggunakan yang dikelola EKS) atau keduanya. ENIs

Baik pengguna dan node pekerja terhubung ke server API menggunakan titik akhir publik atau ENI yang dikelola EKS, ada jalur redundan untuk koneksi.

Rekomendasi

Tinjau rekomendasi berikut.

Monitor Metrik Bidang Kontrol

Memantau metrik API Kubernetes dapat memberi Anda wawasan tentang kinerja bidang kontrol dan mengidentifikasi masalah. Bidang kontrol yang tidak sehat dapat membahayakan ketersediaan beban kerja yang berjalan di dalam cluster. Misalnya, pengontrol yang ditulis dengan buruk dapat membebani server API, memengaruhi ketersediaan aplikasi Anda.

Kubernetes mengekspos metrik bidang kontrol di titik akhir. `/metrics`

Anda dapat melihat metrik yang diekspos menggunakan`kubectl`:

```
kubectl get --raw /metrics
```

Metrik ini direpresentasikan dalam format teks Prometheus.

Anda dapat menggunakan Prometheus untuk mengumpulkan dan menyimpan metrik ini. Pada Mei 2020, CloudWatch menambahkan dukungan untuk memantau metrik Prometheus di Wawasan Kontainer. CloudWatch Jadi Anda juga dapat menggunakan Amazon CloudWatch untuk memantau bidang kontrol EKS. Anda dapat menggunakan Tutorial untuk Menambahkan Target Scrape Prometheus Baru: Metrik Server Prometheus KPI untuk mengumpulkan metrik dan membuat dasbor untuk memantau bidang kontrol cluster Anda. CloudWatch

Anda dapat menemukan metrik server API Kubernetes di sini. Misalnya, `apiserver_request_duration_seconds` dapat menunjukkan berapa lama permintaan API dijalankan.

Pertimbangkan untuk memantau metrik bidang kontrol ini:

Server API

Metrik	Deskripsi
`apiserver_request_total`	Penghitung permintaan apiserver dipecah untuk setiap kata kerja, nilai dry run, grup, versi, sumber daya, ruang lingkup, komponen, dan kode respons HTTP.
`apiserver_request_duration_seconds*`	Histogram latensi respons dalam hitungan detik untuk setiap kata kerja, nilai dry run, grup, versi, sumber daya, subsumber daya, ruang lingkup, dan komponen.
`apiserver_admission_controller_admission_duration_seconds*`	Histogram latensi pengontrol penerimaan dalam hitungan detik, diidentifikasi berdasarkan nama dan dipecah untuk setiap operasi dan sumber daya API dan jenis (memvalidasi atau mengakui).
`apiserver_admission_webhook_rejection_count`	Hitungan penerimaan penolakan webhook. Diidentifikasi berdasarkan nama, operasi, rejection_code, jenis (memvalidasi atau mengakui), error_type (calling_webhook_error, apiserver_internal_error, no_error)
`rest_client_request_duration_seconds*`	Minta histogram latensi dalam hitungan detik. Diuraikan berdasarkan kata kerja dan URL.
`rest_client_requests_total`	Jumlah permintaan HTTP, dipartisi berdasarkan kode status, metode, dan host.

- Metrik histogram mencakup sufiks _bucket, _sum, dan _count.

etcd

Metrik	Deskri
`etcd_request_duration_seconds*`	Etcd memir histogr latensi dalam hitung detik untuk setiap operas dan jenis objek.
`apiserver_storage_db_total_size_in_bytes` atau `apiserver_storage_size_bytes` (dimulai dengan EKS v1.28)	Ukuran databa Etcd.

- Metrik histogram mencakup sufiks _bucket, _sum, dan _count.

Pertimbangkan untuk menggunakan Dasbor Ikhtisar Pemantauan Kubernetes untuk memvisualisasikan dan memantau permintaan server API Kubernetes serta metrik latensi dan latensi etcd.

> **Important**
>
> Ketika batas ukuran database terlampaui, etcd memancarkan alarm tanpa spasi dan berhenti mengambil permintaan tulis lebih lanjut. Dengan kata lain, klaster menjadi hanya-baca, dan semua permintaan untuk mengubah objek seperti membuat pod baru, penyebaran skala, dll., akan ditolak oleh server API klaster.

Otentikasi Cluster

EKS saat ini mendukung dua jenis otentikasi: token akun bearer/service dan autentikasi IAM yang menggunakan otentikasi token webhook. Saat pengguna memanggil Kubernetes API, webhook meneruskan token otentikasi yang disertakan dalam permintaan ke IAM. Token, URL bertanda dasar 64, dihasilkan oleh AWS Command Line Interface (AWS CLI).

Pengguna IAM atau peran yang membuat EKS Cluster secara otomatis mendapatkan akses penuh ke cluster. Anda dapat mengelola akses ke cluster EKS dengan mengedit configmap aws-auth.

Jika Anda salah mengonfigurasi `aws-auth` configmap dan kehilangan akses ke cluster, Anda masih dapat menggunakan pengguna atau peran pembuat klaster untuk mengakses kluster EKS Anda.

Jika Anda tidak dapat menggunakan layanan IAM di wilayah AWS, Anda juga dapat menggunakan token pembawa akun layanan Kubernetes untuk mengelola klaster.

Buat `super-admin` akun yang diizinkan untuk melakukan semua tindakan di cluster:

```
kubectl -n kube-system create serviceaccount super-admin
```

Buat pengikatan peran yang memberikan peran cluster-admin super-admin:

```
kubectl create clusterrolebinding super-admin-rb --clusterrole=cluster-admin --
serviceaccount=kube-system:super-admin
```

Dapatkan rahasia akun layanan:

```
SECRET_NAME=`kubectl -n kube-system get serviceaccount/super-admin -o
 jsonpath='{.secrets[0].name}'`
```

Dapatkan token yang terkait dengan rahasia:

```
TOKEN=`kubectl -n kube-system get secret $SECRET_NAME -o jsonpath='{.data.token}'|
 base64 --decode`
```

Tambahkan akun layanan dan token kekubeconfig:

```
kubectl config set-credentials super-admin --token=$TOKEN
```

Atur konteks saat ini `kubeconfig` untuk menggunakan akun super-admin:

```
kubectl config set-context --current --user=super-admin
```

Final `kubeconfig` akan terlihat seperti ini:

```
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>>
```

Penerimaan Webhooks

Kubernetes memiliki dua jenis webhook penerimaan: memvalidasi webhook masuk dan memutasi webhook masuk. Ini memungkinkan pengguna untuk memperluas API kubernetes dan memvalidasi atau mengubah objek sebelum diterima oleh API. Konfigurasi yang buruk dari webhook ini dapat mengacaukan bidang kontrol EKS dengan memblokir operasi kritis cluster.

Untuk menghindari dampak operasi kritis klaster, hindari pengaturan webhook “catch-all” seperti berikut:

```
- name: "pod-policy.example.com"
  rules:
  - apiGroups:   ["*"]
    apiVersions: ["*"]
    operations:  ["*"]
    resources:   ["*"]
    scope: "*"
```

Atau pastikan webhook memiliki kebijakan buka yang gagal dengan batas waktu lebih pendek dari 30 detik untuk memastikan bahwa jika webhook Anda tidak tersedia, itu tidak akan mengganggu beban kerja kritis klaster.

Blokir Pod dengan tidak aman **`sysctls`**

`Sysctl`adalah utilitas Linux yang memungkinkan pengguna untuk memodifikasi parameter kernel selama runtime. Parameter kernel ini mengontrol berbagai aspek perilaku sistem operasi, seperti jaringan, sistem file, memori virtual, dan manajemen proses.

Kubernetes memungkinkan `sysctl` penetapan profil untuk Pod. Kubernetes dikategorikan sebagai aman `systcls` dan tidak aman. Safe `sysctls` adalah namespaced di container atau Pod, dan menyetelnya tidak berdampak pada Pod lain pada node atau node itu sendiri. Sebaliknya, sysctl yang tidak aman dinonaktifkan secara default karena berpotensi mengganggu Pod lain atau membuat node tidak stabil.

Karena unsafe dinonaktifkan `sysctls` secara default, kubelet tidak akan membuat Pod dengan profil yang tidak aman. `sysctl` Jika Anda membuat Pod seperti itu, scheduler akan berulang kali menetapkan Pod tersebut ke node, sementara node gagal meluncurkannya. Loop tak terbatas ini akhirnya membebani bidang kontrol cluster, membuat cluster tidak stabil.

Pertimbangkan untuk menggunakan OPA Gatekeeper atau Kyverno untuk menolak Pod dengan unsafe. `sysctls`

Menangani Upgrade Cluster

Sejak April 2021, siklus rilis Kubernetes telah diubah dari empat rilis setahun (sekali seperempat) menjadi tiga rilis setahun. Versi minor baru (seperti 1. 21 atau 1. 22) dirilis kira-kira setiap lima

belas minggu. Dimulai dengan Kubernetes 1.19, setiap versi minor didukung selama kurang lebih dua belas bulan setelah pertama kali dirilis. Dengan munculnya Kubernetes v1.28, kecondongan kompatibilitas antara bidang kontrol dan node pekerja telah berkembang dari n-2 ke n-3 versi minor. Untuk mempelajari lebih lanjut, lihat Praktik Terbaik untuk Peningkatan Kluster.

Konektivitas Titik Akhir Cluster

Saat bekerja dengan Amazon EKS (Elastic Kubernetes Service), Anda mungkin mengalami batas waktu koneksi atau kesalahan selama peristiwa seperti penskalaan atau patching bidang kontrol Kubernetes. Peristiwa ini dapat menyebabkan instance kube-apiserver diganti, berpotensi mengakibatkan alamat IP yang berbeda dikembalikan saat menyelesaikan FQDN. Dokumen ini menguraikan praktik terbaik bagi konsumen API Kubernetes untuk menjaga konektivitas yang andal.

> **Note**
>
> Menerapkan praktik terbaik ini mungkin memerlukan pembaruan pada konfigurasi atau skrip klien untuk menangani resolusi ulang DNS baru dan mencoba lagi strategi secara efektif.

Masalah utama berasal dari cache sisi klien DNS dan potensi alamat IP basi dari titik akhir EKS - NLB publik untuk titik akhir publik atau X-ENI untuk titik akhir pribadi. Ketika instans kube-apiserver diganti, Fully Qualified Domain Name (FQDN) dapat diselesaikan ke alamat IP baru. Namun, karena pengaturan DNS Time to Live (TTL), yang diatur ke 60 detik di zona Route 53 AWS yang dikelola AWS, klien dapat terus menggunakan alamat IP yang sudah ketinggalan zaman untuk waktu yang singkat.

Untuk mengatasi masalah ini, konsumen Kubernetes API (seperti kubectl, CI/CD pipelines, dan aplikasi kustom) harus menerapkan praktik terbaik berikut:

- Menerapkan resolusi ulang DNS
- Implementasikan Retries dengan Backoff dan Jitter. Misalnya, lihat artikel ini berjudul Kegagalan Terjadi
- Menerapkan Timeout Klien. Tetapkan batas waktu yang sesuai untuk mencegah permintaan yang berjalan lama memblokir aplikasi Anda. Ketahuilah bahwa beberapa pustaka klien Kubernetes, terutama yang dihasilkan oleh generator OpenAPI, mungkin tidak mengizinkan pengaturan batas waktu kustom dengan mudah.
  - Contoh 1 dengan kubectl:

```
kubectl get pods --request-timeout 10s # default: no timeout
```

- Contoh 2 dengan Python: Klien Kubernetes menyediakan parameter _request_timeout

Dengan menerapkan praktik terbaik ini, Anda dapat secara signifikan meningkatkan keandalan dan ketahanan aplikasi Anda saat berinteraksi dengan Kubernetes API. Ingatlah untuk menguji implementasi ini secara menyeluruh, terutama di bawah kondisi kegagalan simulasi, untuk memastikan implementasi tersebut berperilaku seperti yang diharapkan selama peristiwa penskalaan atau penambalan aktual.

Menjalankan cluster besar

EKS secara aktif memonitor beban pada instans bidang kontrol dan secara otomatis menskalakannya untuk memastikan kinerja tinggi. Namun, Anda harus memperhitungkan potensi masalah dan batasan kinerja dalam Kubernetes dan kuota di layanan AWS saat menjalankan klaster besar.

- Cluster dengan lebih dari 1000 layanan dapat mengalami latensi jaringan dengan menggunakan `kube-proxy` dalam `iptables` mode sesuai dengan tes yang dilakukan oleh tim. ProjectCalico Solusinya adalah beralih ke `kube-proxy` `ipvs` mode berjalan.
- Anda juga dapat mengalami pembatasan permintaan API EC2 jika CNI perlu meminta alamat IP untuk Pod atau jika Anda perlu sering membuat instans EC2 baru. Anda dapat mengurangi panggilan EC2 API dengan mengonfigurasi CNI untuk menyimpan alamat IP cache. Anda dapat menggunakan jenis instans EC2 yang lebih besar untuk mengurangi peristiwa penskalaan EC2.

Sumber Daya Tambahan:

- Jaringan cluster de-mistik untuk node pekerja Amazon EKS
- Kontrol akses titik akhir kluster Amazon EKS
- AWS re: Invent 2019: Amazon EKS di bawah tenda (-R1) CON421

Pesawat Data EKS

Untuk mengoperasikan aplikasi yang tersedia tinggi dan tangguh, Anda memerlukan bidang data yang sangat tersedia dan tangguh. Bidang data elastis memastikan Kubernetes dapat menskalakan dan menyembuhkan aplikasi Anda secara otomatis. Bidang data tangguh terdiri dari dua atau lebih

node pekerja, dapat tumbuh dan menyusut dengan beban kerja, dan secara otomatis pulih dari kegagalan.

Anda memiliki beberapa pilihan untuk node pekerja dengan EKS: EKS Auto Mode managed nodes, EC2 Instances dan Fargate.

Mode Otomatis EKS menawarkan jalur termudah ke bidang data yang tangguh. Mode Otomatis memperluas pengelolaan AWS klaster Kubernetes di luar klaster itu sendiri, untuk memungkinkan AWS juga menyiapkan dan mengelola infrastruktur yang memungkinkan kelancaran pengoperasian beban kerja Anda. Mode Otomatis secara otomatis menskalakan bidang data ke atas atau ke bawah saat Kubernetes menskalakan Pod dan bekerja untuk terus memastikan bahwa Node di klaster Anda berukuran tepat dan hemat biaya untuk beban kerja yang sedang berjalan.

Jika Anda memilih instans EC2, Anda dapat mengelola node pekerja sendiri atau menggunakan grup node terkelola EKS. Anda dapat memiliki klaster dengan campuran Mode Otomatis, node pekerja yang dikelola, dikelola sendiri, dan Fargate.

Fargate menjalankan setiap Pod dalam lingkungan komputasi yang terisolasi. Setiap Pod yang berjalan di Fargate mendapatkan node pekerjanya sendiri. Fargate secara otomatis menskalakan bidang data saat Kubernetes menskalakan pod. Anda dapat menskalakan bidang data dan beban kerja Anda dengan menggunakan autoscaler pod horizontal.

Cara yang lebih disukai untuk menskalakan node pekerja EC2 (jika tidak menggunakan Mode Otomatis EKS di mana ini dilakukan secara otomatis oleh AWS) adalah dengan menggunakan grup Karpenter, Kubernetes Cluster Autoscaler, atau EC2 Auto Scaling.

Rekomendasi

Sebarkan node pekerja dan beban kerja di beberapa AZs

Anda dapat melindungi beban kerja Anda dari kegagalan di AZ individual dengan menjalankan node pekerja dan Pod dalam beberapa AZs. Anda dapat mengontrol AZ tempat node pekerja dibuat menggunakan subnet tempat Anda membuat node.

Metode yang disarankan untuk menyebarkan pod ke seluruh Pod AZs adalah dengan menggunakan Topology Spread Constraints untuk Pod. Kemampuan penskalaan otomatis seperti Mode Otomatis EKS dan Karpenter menyadari kendala penyebaran topologi dan akan secara otomatis meluncurkan Node di tempat yang benar untuk memungkinkan kendala Anda dipenuhi. AZs

Penerapan di bawah ini menyebarkan pod ke seluruh AZs jika memungkinkan, membiarkan pod tersebut tetap berjalan jika tidak:

```
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`hanya mengetahui domain topologi melalui node yang ada dengan label tersebut. Jika penerapan di atas di-deploy ke cluster dengan node hanya dalam satu zona, semua pod akan menjadwalkan pada node tersebut karena `kube-scheduler` tidak mengetahui zona lainnya. Agar penyebaran topologi ini berfungsi seperti yang diharapkan dengan penjadwal, node harus sudah ada di semua zona. `minDomains`Properti kendala penyebaran topologi digunakan untuk menginformasikan penjadwal jumlah domain yang memenuhi syarat, bahkan jika ada Node yang berjalan di sana untuk menghindari masalah ini.

> **Warning**
>
> Pengaturan `whenUnsatisfiable` ke `DoNotSchedule` akan menyebabkan pod tidak dapat dijadwalkan jika batasan penyebaran topologi tidak dapat dipenuhi. Seharusnya hanya disetel jika pod lebih disukai untuk tidak berjalan daripada melanggar batasan penyebaran topologi.

Pada versi Kubernetes yang lebih lama, Anda dapat menggunakan aturan anti-afinitas pod untuk menjadwalkan pod di beberapa pod. AZs Manifes di bawah ini memberi tahu penjadwal Kubernetes untuk memilih pod penjadwalan yang berbeda. 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**
>
> Jangan mengharuskan Pod dijadwalkan secara berbeda, AZs jika tidak, jumlah Pod dalam penerapan tidak akan pernah melebihi jumlah AZs Pod.

Pastikan kemampuan untuk meluncurkan Node di setiap AZ saat menggunakan volume EBS

Jika Anda menggunakan Amazon EBS untuk menyediakan Volume Persisten, Anda perlu memastikan bahwa pod dan volume EBS terkait berada di AZ yang sama. Sebuah Pod tidak dapat mengakses volume persisten yang didukung EBS yang terletak di AZ yang berbeda. Penjadwal Kubernetes mengetahui AZ mana node pekerja berada dari label yang ada di Node dan akan selalu menjadwalkan Pod yang membutuhkan volume EBS dalam AZ yang sama dengan volume. Namun, jika tidak ada node pekerja yang tersedia di AZ tempat volume berada, maka Pod tidak dapat dijadwalkan.

Jika menggunakan Mode Otomatis EKS atau Karpenter, Anda perlu memastikan bahwa subnet NodeClass pilihan Anda di setiap AZ. Jika menggunakan Grup Node Terkelola, Anda perlu memastikan bahwa Anda memiliki Grup Node di setiap AZ.

Kemampuan penyimpanan EBS dibangun ke dalam Mode Otomatis EKS, tetapi jika menggunakan Karpenter atau Grup Node Terkelola, EBS CSI juga perlu diinstal.

Gunakan Mode Otomatis EKS untuk mengelola node pekerja

Mode Otomatis EKS merampingkan manajemen EKS dengan menyediakan klaster siap produksi dengan overhead operasional minimal. Mode Otomatis bertanggung jawab untuk menskalakan jumlah Node naik atau turun tergantung pada Pod yang berjalan di klaster. Node selalu diperbarui dengan patch dan perbaikan perangkat lunak secara otomatis, dengan pembaruan dilakukan sesuai dengan pengaturan NodePoolgangguan yang dikonfigurasi dan Anggaran Gangguan Pod.

Jalankan Node Monitoring Agent

Agen Pemantau Node memantau dan bereaksi terhadap masalah kesehatan Node dengan mempublikasikan peristiwa Kubernetes dan memperbarui kondisi status pada Node. Agen

Pemantauan Node disertakan dengan Node Mode Otomatis EKS, dan dapat diinstal sebagai Addon EKS untuk Node yang tidak dikelola oleh Mode Otomatis.

EKS Auto Mode, Managed Node Groups, dan Karpenter semuanya memiliki kemampuan untuk mendeteksi kondisi Node fatal yang dilaporkan oleh Node Monitoring Agent dan memperbaiki Node tersebut secara otomatis ketika kondisi tersebut terjadi.

Menerapkan QoS

Untuk aplikasi kritis, pertimbangkan untuk mendefinisikan `requests` = `limits` untuk kontainer di dalam Pod. Ini akan memastikan bahwa kontainer tidak akan dimatikan jika Pod lain meminta sumber daya.

Ini adalah praktik terbaik untuk menerapkan batas CPU dan memori untuk semua kontainer karena mencegah wadah secara tidak sengaja mengkonsumsi sumber daya sistem yang memengaruhi ketersediaan proses lain yang ditempatkan bersama.

Konfigurasi dan Ukuran Sumber Daya Requests/Limits untuk semua Beban Kerja

Beberapa panduan umum dapat diterapkan untuk mengukur permintaan sumber daya dan batasan untuk beban kerja:

- Jangan tentukan batas sumber daya pada CPU. Dengan tidak adanya batasan, permintaan bertindak sebagai bobot pada berapa banyak kontainer waktu CPU relatif yang didapat. Hal ini memungkinkan beban kerja Anda untuk menggunakan CPU penuh tanpa batas buatan atau kelaparan.
- Untuk sumber daya non-CPU, konfigurasi `requests` = `limits` memberikan perilaku yang paling dapat diprediksi. Jika`requests`! `=limits`, kontainer juga mengurangi QOS-nya dari Dijamin menjadi Burstable sehingga lebih mungkin untuk diusir jika terjadi tekanan simpul.
- Untuk sumber daya non-CPU, jangan tentukan batas yang jauh lebih besar dari permintaan. Semakin besar `limits` dikonfigurasi relatif terhada`requests`, semakin besar kemungkinan node akan terlalu berkomitmen yang mengarah ke kemungkinan tinggi gangguan beban kerja.
- Permintaan berukuran benar sangat penting saat menggunakan solusi auto-scaling node seperti Karpenter atau Cluster. AutoScaler Alat-alat ini melihat permintaan beban kerja Anda untuk menentukan jumlah dan ukuran node yang akan disediakan. Jika permintaan Anda terlalu kecil dengan batas yang lebih besar, Anda mungkin menemukan beban kerja Anda diusir atau OOM dimatikan jika mereka telah dikemas rapat pada sebuah node.

Menentukan permintaan sumber daya bisa jadi sulit, tetapi alat seperti Vertical Pod Autoscaler dapat membantu Anda "mengukur" permintaan dengan mengamati penggunaan sumber daya kontainer saat runtime. Alat lain yang mungkin berguna untuk menentukan ukuran permintaan meliputi:

- Goldilocks
- Parca
- Prodfiler
- rsg

Konfigurasikan kuota sumber daya untuk ruang nama

Ruang nama dimaksudkan untuk digunakan di lingkungan dengan banyak pengguna yang tersebar di beberapa tim, atau proyek. Mereka menyediakan ruang lingkup untuk nama dan merupakan cara untuk membagi sumber daya cluster antara beberapa tim, proyek, beban kerja. Anda dapat membatasi konsumsi sumber daya agregat di namespace. `ResourceQuota`Objek dapat membatasi jumlah objek yang dapat dibuat dalam namespace berdasarkan jenis, serta jumlah total sumber daya komputasi yang dapat dikonsumsi oleh sumber daya dalam proyek itu. Anda dapat membatasi jumlah total sumber daya and/or komputasi penyimpanan (CPU dan memori) yang dapat diminta dalam namespace tertentu.

Jika kuota sumber daya diaktifkan untuk namespace untuk sumber daya komputasi seperti CPU dan memori, pengguna harus menentukan permintaan atau batasan untuk setiap kontainer di namespace tersebut.

Pertimbangkan untuk mengonfigurasi kuota untuk setiap namespace. Pertimbangkan `LimitRanges` untuk menggunakan untuk secara otomatis menerapkan batas yang telah dikonfigurasi sebelumnya ke kontainer dalam ruang nama.

Batasi penggunaan sumber daya kontainer dalam namespace

Resource Quotas membantu membatasi jumlah sumber daya yang dapat digunakan namespace. `LimitRange`Objek dapat membantu Anda menerapkan sumber daya minimum dan maksimum yang dapat diminta oleh wadah. Menggunakan `LimitRange` Anda dapat menetapkan permintaan default dan batasan untuk kontainer, yang sangat membantu jika menyetel batas sumber daya komputasi bukanlah praktik standar di organisasi Anda. Seperti namanya, `LimitRange` dapat menerapkan penggunaan sumber daya komputasi minimum dan maksimum per Pod atau Container dalam namespace. Selain itu, terapkan permintaan penyimpanan minimum dan maksimum per PersistentVolumeClaim di namespace.

Pertimbangkan untuk menggunakan `LimitRange` bersama dengan `ResourceQuota` untuk menegakkan batasan pada wadah serta tingkat namespace. Menetapkan batas ini akan memastikan bahwa wadah atau namespace tidak memengaruhi sumber daya yang digunakan oleh penyewa lain di cluster.

Gunakan NodeLocal DNSCache

Anda dapat meningkatkan kinerja DNS Cluster dengan menjalankan NodeLocalDNSCache. Fitur ini menjalankan agen caching DNS pada node cluster sebagai file. DaemonSet Semua pod menggunakan agen caching DNS yang berjalan di node untuk resolusi nama alih-alih menggunakan `kube-dns` Service. Fitur ini secara otomatis disertakan dalam Mode Otomatis EKS.

Konfigurasikan CoreDNS auto-scaling

Metode lain untuk meningkatkan kinerja DNS Cluster adalah dengan mengaktifkan auto-scaling bawaan Pod CoreDNS.

Fitur ini terus memantau status cluster, termasuk jumlah node dan core CPU. Berdasarkan informasi tersebut, pengontrol akan secara dinamis menyesuaikan jumlah replika penerapan CoreDNS di cluster EKS.

Praktik Terbaik untuk Jaringan

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Sangat penting untuk memahami jaringan Kubernetes untuk mengoperasikan klaster dan aplikasi Anda secara efisien. Jaringan Pod, juga disebut jaringan cluster, adalah pusat dari jaringan Kubernetes. Kubernetes mendukung plugin Container Network Interface (CNI) untuk jaringan cluster.

Amazon EKS secara resmi mendukung plugin Amazon Virtual Private Cloud (VPC) CNI untuk mengimplementasikan jaringan Kubernetes Pod. VPC CNI menyediakan integrasi asli dengan AWS VPC dan bekerja dalam mode underlay. Dalam mode underlay, Pod dan host berada di lapisan jaringan yang sama dan berbagi namespace jaringan. Alamat IP Pod konsisten dari perspektif cluster dan VPC.

Panduan ini memperkenalkan Amazon VPC Container Network Interface (VPC CNI) dalam konteks jaringan cluster Kubernetes. VPC CNI adalah plugin jaringan default yang didukung oleh EKS dan karenanya menjadi fokus panduan ini. VPC CNI sangat dapat dikonfigurasi untuk mendukung kasus penggunaan yang berbeda. Panduan ini selanjutnya mencakup bagian khusus pada kasus penggunaan VPC CNI yang berbeda, mode operasi, sub-komponen, diikuti oleh rekomendasi.

Amazon EKS menjalankan Kubernetes hulu dan bersertifikat kesesuaian Kubernetes. Meskipun Anda dapat menggunakan plugin CNI alternatif, panduan ini tidak memberikan rekomendasi untuk mengelola alternatif. CNIs Periksa dokumentasi EKS Alternate CNI untuk daftar mitra dan sumber daya untuk mengelola alternatif CNIs secara efektif.

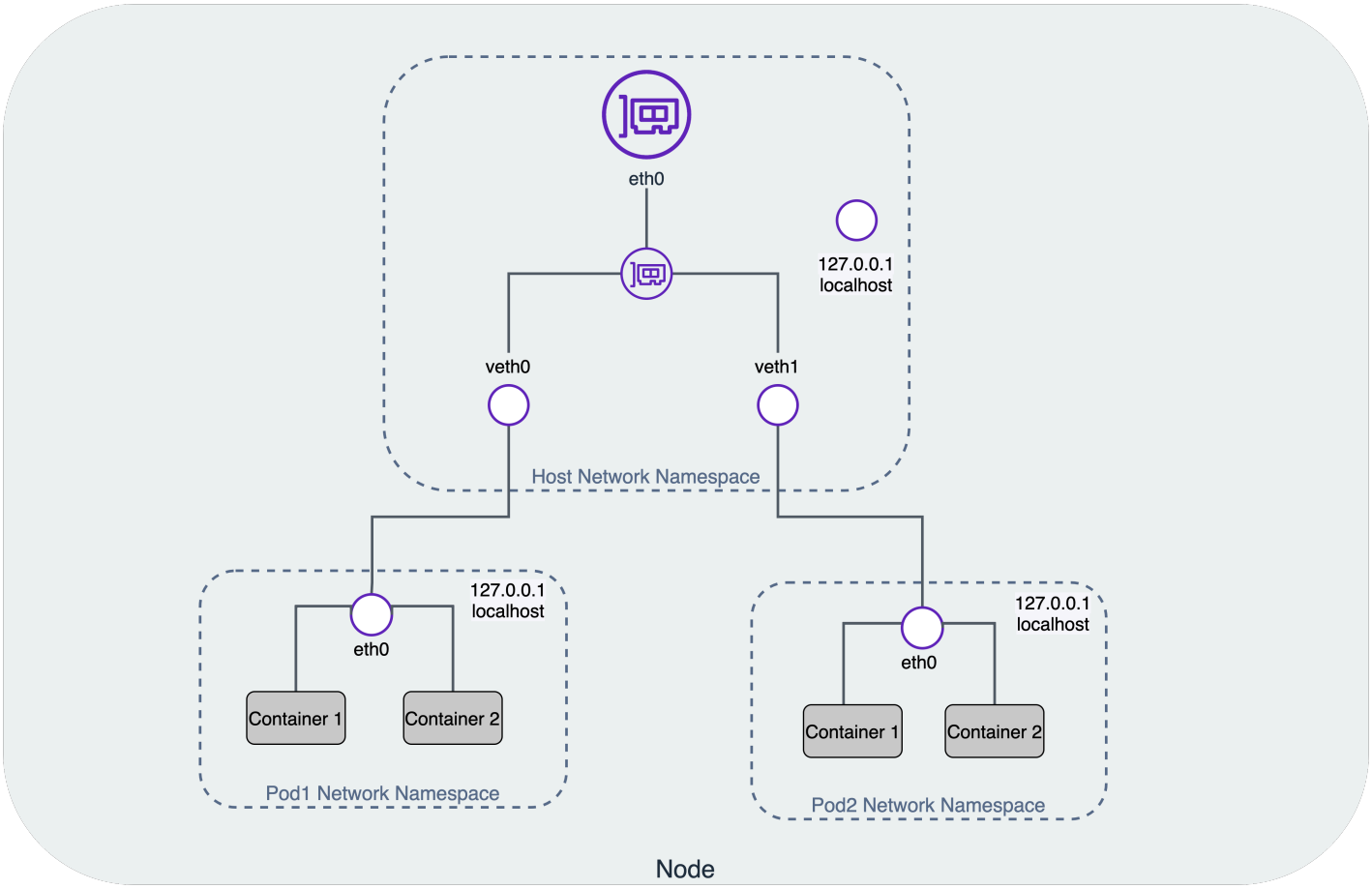
Model Jaringan Kubernetes

Kubernetes menetapkan persyaratan berikut pada jaringan klaster:

- Pod yang dijadwalkan pada node yang sama harus dapat berkomunikasi dengan Pod lain tanpa menggunakan NAT (Network Address Translation).
- Semua daemon sistem (proses latar belakang, misalnya, kubelet) yang berjalan pada node tertentu dapat berkomunikasi dengan Pod yang berjalan pada node yang sama.

- Pod yang menggunakan jaringan host harus dapat menghubungi semua Pod lain di semua node lain tanpa menggunakan NAT.

Lihat model jaringan Kubernetes untuk detail tentang apa yang Kubernetes harapkan dari implementasi jaringan yang kompatibel. Gambar berikut menggambarkan hubungan antara namespace jaringan Pod dan namespace jaringan host.



Antarmuka Jaringan Kontainer (CNI)

Kubernetes mendukung spesifikasi dan plugin CNI untuk mengimplementasikan model jaringan Kubernetes. CNI terdiri dari spesifikasi (versi saat ini 1.0.0) dan pustaka untuk menulis plugin untuk mengkonfigurasi antarmuka jaringan dalam wadah, bersama dengan sejumlah plugin yang didukung. CNI hanya menyangkut konektivitas jaringan kontainer dan menghapus sumber daya yang dialokasikan saat wadah dihapus.

Plugin CNI diaktifkan dengan meneruskan kubelet opsi baris perintah. `--network-plugin=cni` Kubelet membaca file dari `--cni-conf-dir` (default/etc/cni/net.d) dan menggunakan konfigurasi

CNI dari file tersebut untuk menyiapkan jaringan masing-masing Pod. File konfigurasi CNI harus sesuai dengan spesifikasi CNI (minimum v0.4.0) dan plugin CNI yang diperlukan yang direferensikan oleh konfigurasi harus ada di direktori (default/). `--cni-bin-dir` opt/cni/bin Jika ada beberapa file konfigurasi CNI di direktori, kubelet menggunakan file konfigurasi yang muncul pertama dengan nama dalam urutan leksikografis.

Amazon Virtual Private Cloud (VPC) CNI

VPC CNI yang disediakan AWS adalah add-on jaringan default untuk kluster EKS. Add-on VPC CNI diinstal secara default saat Anda menyediakan kluster EKS. VPC CNI berjalan pada node pekerja Kubernetes. Add-on VPC CNI terdiri dari biner CNI dan plugin IP Address Management (ipamd). CNI memberikan alamat IP dari jaringan VPC ke Pod. Ipamd mengelola AWS Elastic Networking Interfaces (ENIs) ke setiap node Kubernetes dan mempertahankan kolam hangat. IPs VPC CNI menyediakan opsi konfigurasi untuk pra-alokasi ENIs dan alamat IP untuk waktu startup Pod yang cepat. Lihat Amazon VPC CNI untuk praktik terbaik manajemen plugin yang direkomendasikan.

Amazon EKS menyarankan Anda menentukan subnet di setidaknya dua zona ketersediaan saat Anda membuat klaster. Amazon VPC CNI mengalokasikan alamat IP ke Pod dari subnet node. Kami sangat menyarankan memeriksa subnet untuk alamat IP yang tersedia. Harap pertimbangkan rekomendasi VPC dan Subnet sebelum menerapkan kluster EKS.

Amazon VPC CNI mengalokasikan kumpulan hangat ENIs dan alamat IP sekunder dari subnet yang dilampirkan ke ENI primer node. Mode VPC CNI ini disebut mode IP sekunder. Jumlah alamat IP dan karenanya jumlah Pod (kepadatan Pod) ditentukan oleh jumlah ENIs dan alamat IP per ENI (limit) sebagaimana didefinisikan oleh jenis instance. Mode sekunder adalah default dan berfungsi dengan baik untuk cluster kecil dengan tipe instance yang lebih kecil. Harap pertimbangkan untuk menggunakan mode awalan jika Anda mengalami tantangan kepadatan pod. Anda juga dapat meningkatkan alamat IP yang tersedia pada node untuk Pod dengan menetapkan prefiks ke. ENIs

Amazon VPC CNI terintegrasi secara native dengan AWS VPC dan memungkinkan pengguna menerapkan praktik terbaik keamanan dan jaringan AWS VPC yang ada untuk membangun kluster Kubernetes. Ini termasuk kemampuan untuk menggunakan log aliran VPC, kebijakan perutean VPC, dan grup keamanan untuk isolasi lalu lintas jaringan. Secara default, Amazon VPC CNI menerapkan grup keamanan yang terkait dengan ENI primer pada node ke Pod. Pertimbangkan untuk mengaktifkan grup keamanan untuk Pod saat Anda ingin menetapkan aturan jaringan yang berbeda untuk sebuah Pod.

Secara default, VPC CNI memberikan alamat IP ke Pod dari subnet yang ditetapkan ke ENI primer dari sebuah node. Adalah umum untuk mengalami kekurangan IPv4 alamat saat menjalankan cluster besar dengan ribuan beban kerja. AWS VPC memungkinkan Anda memperpanjang ketersediaan IPs dengan menetapkan sekunder CIDRs untuk mengatasi kehabisan blok CIDR. IPv4 AWS VPC CNI memungkinkan Anda menggunakan rentang CIDR subnet yang berbeda untuk Pod. Fitur VPC CNI ini disebut jaringan khusus. Anda dapat mempertimbangkan untuk menggunakan jaringan khusus untuk digunakan `100.64.0.0/10` dan `198.19.0.0/16` CIDRs (CG-NAT) dengan EKS. Ini secara efektif memungkinkan Anda untuk membuat lingkungan di mana Pod tidak lagi menggunakan alamat RFC1918 IP apa pun dari VPC Anda.

Jaringan khusus adalah salah satu opsi untuk mengatasi masalah kelelahan IPv4 alamat, tetapi memerlukan overhead operasional. Kami merekomendasikan IPv6 cluster melalui jaringan khusus untuk mengatasi masalah ini. Secara khusus, kami sarankan untuk bermigrasi ke IPv6 cluster jika Anda benar-benar kehabisan semua ruang IPv4 alamat yang tersedia untuk VPC Anda. Evaluasi rencana organisasi Anda untuk mendukung IPv6, dan pertimbangkan apakah berinvestasi IPv6 mungkin memiliki nilai jangka panjang yang lebih besar.

Dukungan EKS untuk IPv6 difokuskan pada pemecahan masalah kelelahan IP yang disebabkan oleh ruang IPv4 alamat yang terbatas. Menanggapi masalah pelanggan yang IPv4 kelelahan, EKS telah memprioritaskan Pod IPv6 -only di atas Pod dual-stack. Artinya, Pod mungkin dapat mengakses IPv4 sumber daya, tetapi mereka tidak diberi IPv4 alamat dari rentang VPC CIDR. VPC CNI memberikan IPv6 alamat ke Pod dari blok CIDR VPC yang dikelola AWS. IPv6

Kalkulator Subnet

Proyek ini mencakup Dokumen Excel Kalkulator Subnet. Dokumen kalkulator ini mensimulasikan konsumsi alamat IP dari beban kerja tertentu di bawah opsi konfigurasi ENI yang berbeda, seperti `WARM_IP_TARGET` dan. `WARM_ENI_TARGET` Dokumen ini mencakup dua lembar, yang pertama untuk mode ENI Hangat, dan yang kedua untuk mode IP Hangat. Tinjau panduan VPC CNI untuk informasi lebih lanjut tentang mode ini.

Masukan:

- Ukuran CIDR Subnet
- Target ENI Hangat atau Target IP Hangat
- Daftar contoh
  - jenis, jumlah, dan jumlah pod beban kerja yang dijadwalkan per instance

Keluaran:

- Jumlah total Pod yang di-host
- Jumlah Subnet yang Dikonsumsi IPs
- Jumlah Subnet IPs yang tersisa
- Detail Tingkat Instance
  - Jumlah Hangat IPs/ENIs per instance
  - Jumlah Aktif IPs/ENIs per instans

Pertimbangan VPC dan Subnet

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Mengoperasikan kluster EKS membutuhkan pengetahuan tentang jaringan AWS VPC, selain jaringan Kubernetes.

Kami menyarankan Anda memahami mekanisme komunikasi pesawat kontrol EKS sebelum Anda mulai merancang VPC atau menyebarkan cluster ke dalam yang sudah ada. VPCs

Lihat pertimbangan VPC Cluster dan pertimbangan grup keamanan Amazon EKS saat merancang VPC dan subnet yang akan digunakan dengan EKS.

Ikhtisar

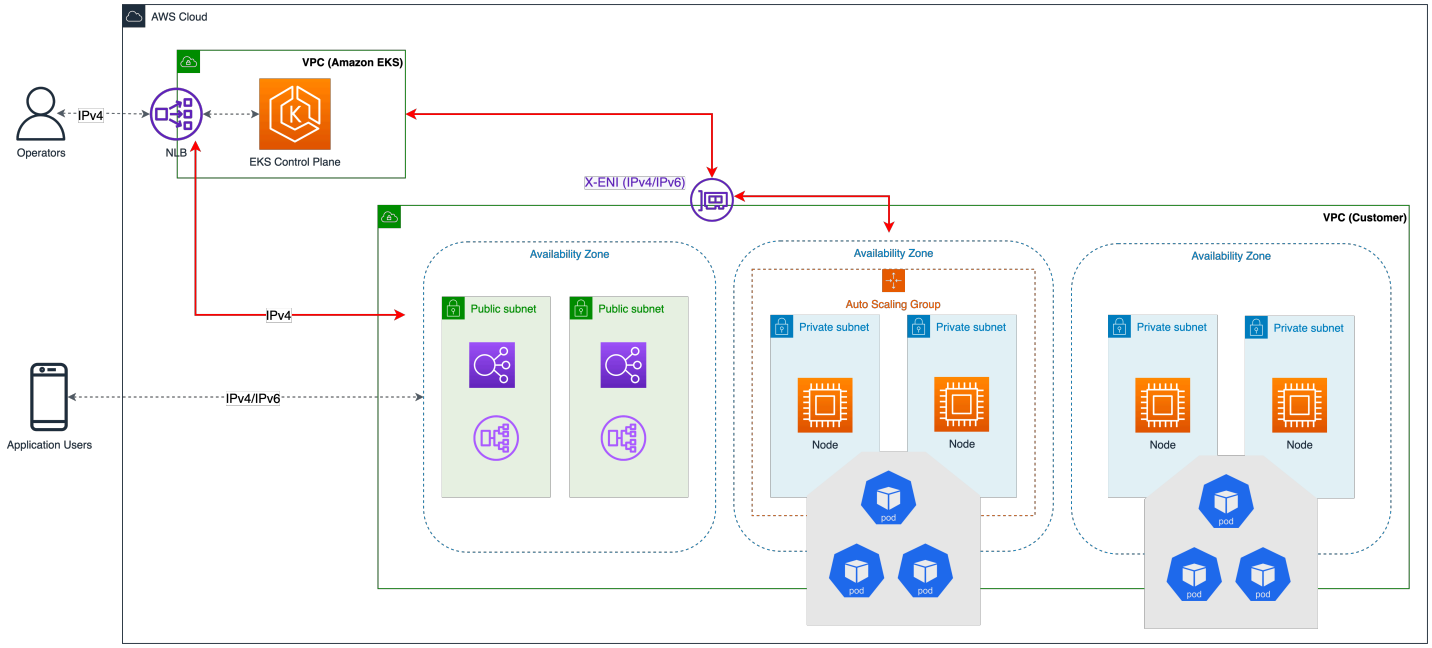
Arsitektur Kluster EKS

Kluster EKS terdiri dari dua VPCs:

- VPC yang dikelola AWS yang menampung pesawat kontrol Kubernetes. VPC ini tidak muncul di akun pelanggan.
- VPC yang dikelola pelanggan yang menghosting node Kubernetes. Di sinilah container berjalan, serta infrastruktur AWS yang dikelola pelanggan lainnya seperti load balancer yang digunakan oleh cluster. VPC ini muncul di akun pelanggan. Anda perlu membuat VPC yang dikelola pelanggan sebelum membuat cluster. Eksctl membuat VPC jika Anda tidak menyediakannya.

Node di VPC pelanggan memerlukan kemampuan untuk terhubung ke titik akhir server API terkelola di AWS VPC. Hal ini memungkinkan node untuk mendaftar dengan control plane Kubernetes dan menerima permintaan untuk menjalankan Pod aplikasi.

Node terhubung ke bidang kontrol EKS melalui (a) titik akhir publik EKS atau (b) antarmuka jaringan elastis Cross-Account (X-ENI) yang dikelola oleh EKS. Ketika sebuah cluster dibuat, Anda perlu menentukan setidaknya dua subnet VPC. EKS menempatkan X-ENI di setiap subnet yang ditentukan selama pembuatan cluster (juga disebut subnet cluster). Server API Kubernetes menggunakan Cross-Account ini ENIs untuk berkomunikasi dengan node yang digunakan pada subnet VPC klaster yang dikelola pelanggan.



Saat node dimulai, skrip bootstrap EKS dijalankan dan file konfigurasi node Kubernetes diinstal. Sebagai bagian dari proses boot pada setiap instance, agen runtime container, kubelet, dan agen node Kubernetes diluncurkan.

Untuk mendaftarkan sebuah node, Kubelet menghubungi titik akhir klaster Kubernetes. Ini membuat koneksi dengan titik akhir publik di luar VPC atau titik akhir pribadi dalam VPC. Kubelet menerima instruksi API dan menyediakan pembaruan status dan detak jantung ke titik akhir secara teratur.

Komunikasi Pesawat Kontrol EKS

EKS memiliki dua cara untuk mengontrol akses ke titik akhir cluster. Kontrol akses endpoint memungkinkan Anda memilih apakah titik akhir dapat dijangkau dari internet publik atau hanya

melalui VPC Anda. Anda dapat mengaktifkan titik akhir publik (yang merupakan default), titik akhir pribadi, atau keduanya sekaligus.

Konfigurasi titik akhir API cluster menentukan jalur yang diambil node untuk berkomunikasi dengan bidang kontrol. Perhatikan bahwa pengaturan titik akhir ini dapat diubah kapan saja melalui konsol EKS atau API.

Titik Akhir Publik

Ini adalah perilaku default untuk klaster Amazon EKS yang baru. Ketika hanya titik akhir publik untuk klaster yang diaktifkan, permintaan Kubernetes API yang berasal dari dalam VPC klaster Anda (seperti node pekerja untuk mengontrol komunikasi pesawat) meninggalkan VPC, tetapi bukan jaringan Amazon. Agar node dapat terhubung ke bidang kontrol, mereka harus memiliki alamat IP publik dan rute ke gateway internet atau rute ke gateway NAT di mana mereka dapat menggunakan alamat IP publik dari gateway NAT.

Titik Akhir Publik dan Pribadi

Ketika endpoint publik dan pribadi diaktifkan, Kubernetes API request dari dalam VPC berkomunikasi ke control plane melalui X- dalam VPC Anda. ENIs Server API klaster Anda dapat diakses dari internet.

Titik Akhir Pribadi

Tidak ada akses publik ke server API Anda dari internet ketika hanya titik akhir pribadi yang diaktifkan. Semua lalu lintas ke server API cluster Anda harus berasal dari dalam VPC klaster Anda atau jaringan yang terhubung. Node berkomunikasi ke server API melalui X- ENIs dalam VPC Anda. Perhatikan bahwa alat manajemen klaster harus memiliki akses ke titik akhir pribadi. Pelajari lebih lanjut cara menyambung ke titik akhir klaster Amazon EKS pribadi dari luar VPC Amazon.

Perhatikan bahwa titik akhir server API cluster diselesaikan oleh server DNS publik ke alamat IP pribadi dari VPC. Di masa lalu, titik akhir hanya dapat diselesaikan dari dalam VPC.

Konfigurasi VPC

Amazon VPC mendukung IPv4 dan IPv6 menangani. Amazon EKS mendukung secara IPv4 default. VPC harus memiliki blok IPv4 CIDR yang terkait dengannya. Anda dapat secara opsional mengaitkan beberapa blok IPv4 Classless Inter-Domain Routing (CIDR) dan beberapa blok CIDR ke VPC Anda. IPv6 Saat Anda membuat VPC, Anda harus menentukan blok IPv4 CIDR untuk VPC dari rentang

IPv4 alamat pribadi seperti yang ditentukan dalam RFC 1918. Ukuran blok yang diizinkan adalah antara /16 awalan (65.536 alamat IP) dan /28 awalan (16 alamat IP).

Saat membuat VPC baru, Anda dapat melampirkan satu blok IPv6 CIDR, dan hingga lima saat mengubah VPC yang ada. Panjang awalan ukuran blok IPv6 CIDR dapat antara /44 dan /60 dan untuk IPv6 subnet dapat antara/44/ dan /64. Anda dapat meminta blok IPv6 CIDR dari kumpulan IPv6 alamat yang dikelola oleh Amazon. Silakan merujuk ke bagian blok VPC CIDR dari Panduan Pengguna VPC untuk informasi lebih lanjut.

Cluster Amazon EKS mendukung keduanya IPv4 dan IPv6. Secara default, kluster EKS menggunakan IPv4 IP. Menentukan IPv6 pada waktu pembuatan cluster akan memungkinkan IPv6 cluster penggunaan. IPv6 cluster membutuhkan dual-stack VPCs dan subnet.

Amazon EKS merekomendasikan Anda menggunakan setidaknya dua subnet yang berada di Availability Zone berbeda selama pembuatan klaster. Subnet yang Anda lewati selama pembuatan cluster dikenal sebagai subnet cluster. Saat Anda membuat klaster, Amazon EKS membuat hingga 4 akun silang (x-account atau x-ENIs) ENIs di subnet yang Anda tentukan. X- selalu ENIs digunakan dan digunakan untuk lalu lintas administrasi cluster seperti pengiriman log, exec, dan proxy. Silakan merujuk ke panduan pengguna EKS untuk detail persyaratan VPC dan subnet lengkap.

Node pekerja Kubernetes dapat berjalan di subnet klaster, tetapi tidak disarankan. Selama upgrade klaster Amazon EKS menyediakan tambahan ENIs di subnet cluster. Saat klaster Anda keluar, node dan pod pekerja dapat menggunakan yang tersedia IPs di subnet klaster. Oleh karena itu untuk memastikan ada cukup tersedia, IPs Anda mungkin ingin mempertimbangkan untuk menggunakan subnet cluster khusus dengan/28 netmask.

Node pekerja Kubernetes dapat berjalan baik di subnet publik atau pribadi. Apakah subnet bersifat publik atau pribadi mengacu pada apakah lalu lintas dalam subnet dirutekan melalui gateway internet. Subnet publik memiliki entri tabel rute ke internet melalui gateway internet, tetapi subnet pribadi tidak.

Lalu lintas yang berasal dari tempat lain dan mencapai node Anda disebut ingress. Lalu lintas yang berasal dari node dan meninggalkan jaringan disebut jalan keluar. Node dengan alamat IP publik atau elastis (EIPs) dalam subnet yang dikonfigurasi dengan gateway internet memungkinkan masuknya dari luar VPC. Subnet pribadi biasanya memiliki routing ke gateway NAT, yang tidak memungkinkan lalu lintas masuk ke node di subnet dari luar VPC sementara masih memungkinkan lalu lintas dari node untuk meninggalkan VPC (jalan keluar).

Di IPv6 dunia, setiap alamat dapat dirutekan melalui internet. IPv6 Alamat yang terkait dengan node dan pod bersifat publik. Subnet pribadi didukung dengan menerapkan gateway internet egress-only

(EIGW) di VPC, memungkinkan lalu lintas keluar sambil memblokir semua lalu lintas masuk. Praktik terbaik untuk menerapkan IPv6 subnet dapat ditemukan di panduan pengguna VPC.

Anda dapat mengkonfigurasi VPC dan Subnet dalam tiga cara berbeda:

Hanya menggunakan subnet publik

Dalam subnet publik yang sama, node dan sumber daya ingress (seperti penyeimbang beban) dibuat. Tandai subnet publik `kubernetes.io/role/elb`untuk membangun penyeimbang beban yang menghadap internet. Dalam konfigurasi ini, titik akhir cluster dapat dikonfigurasi menjadi publik, pribadi, atau keduanya (publik dan pribadi).

Menggunakan subnet pribadi dan publik

Node dibuat pada subnet pribadi, sedangkan sumber daya Ingress dipakai di subnet publik. Anda dapat mengaktifkan akses publik, pribadi, atau keduanya (publik dan pribadi) ke titik akhir cluster. Tergantung pada konfigurasi titik akhir cluster, lalu lintas node akan masuk melalui gateway NAT atau ENI.

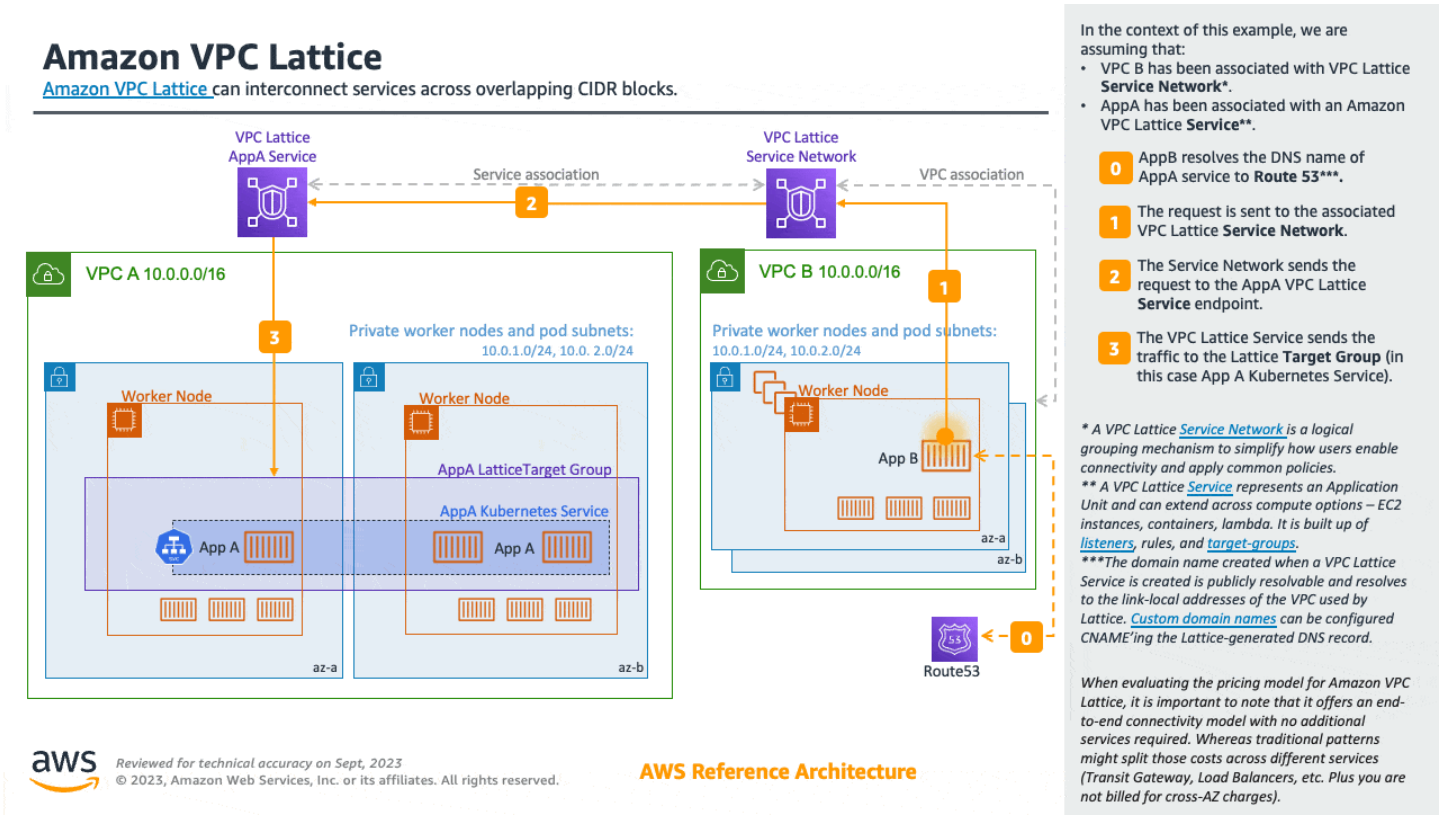
Hanya menggunakan subnet pribadi

Kedua node dan ingress dibuat dalam subnet pribadi. Menggunakan tag `kubernetes.io/role/internal-elb`subnet untuk membangun penyeimbang beban internal. Mengakses titik akhir klaster Anda akan membutuhkan koneksi VPN. Anda harus mengaktifkan AWS PrivateLink untuk EC2 dan semua repositori Amazon ECR dan S3. Hanya titik akhir pribadi cluster yang harus diaktifkan. Kami menyarankan untuk memeriksa persyaratan cluster pribadi EKS sebelum menyediakan cluster pribadi.

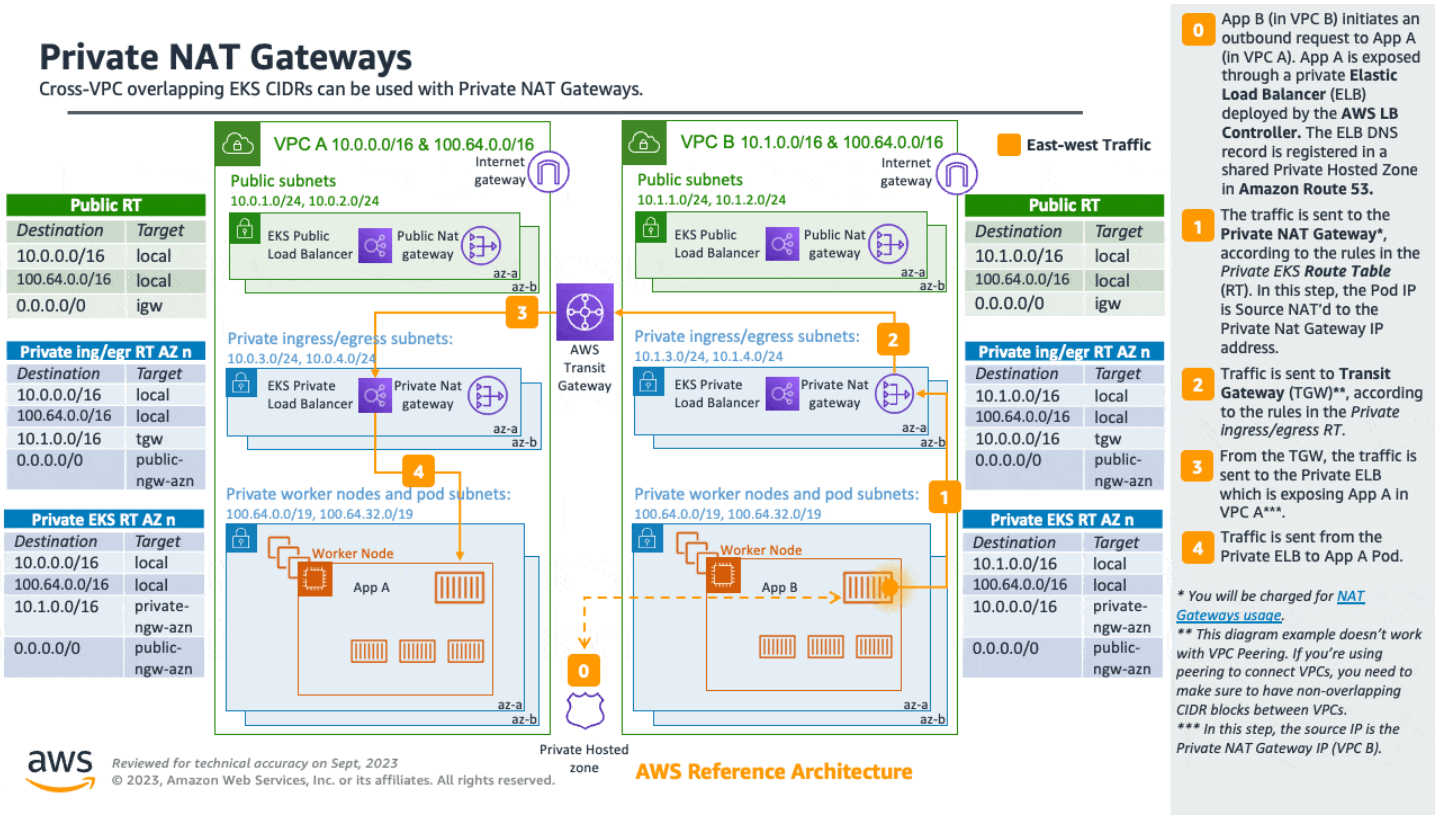
Komunikasi lintas VPCs

Ada banyak skenario ketika Anda memerlukan beberapa VPCs dan terpisah kluster EKS digunakan untuk ini. VPCs

Anda dapat menggunakan Amazon VPC Lattice untuk menghubungkan layanan secara konsisten dan aman di beberapa VPCs akun (tanpa memerlukan konektivitas tambahan yang disediakan oleh layanan seperti peering VPC, AWS, atau AWS Transit Gateway). PrivateLink Pelajari lebih lanjut di sini.



Amazon VPC Lattice beroperasi di ruang alamat link-lokal di IPv4 dan IPv6, menyediakan konektivitas antar layanan yang mungkin memiliki alamat yang tumpang tindih. IPv4 Untuk efisiensi operasional, kami sangat menyarankan untuk menggunakan kluster dan node EKS ke rentang IP yang tidak tumpang tindih. Jika infrastruktur Anda mencakup rentang IP VPCs yang tumpang tindih, Anda perlu merancang jaringan Anda sesuai dengan itu. Kami menyarankan Private NAT Gateway, atau VPC CNI dalam mode jaringan khusus bersama dengan gateway transit untuk mengintegrasikan beban kerja di EKS untuk menyelesaikan tantangan CIDR yang tumpang tindih sambil mempertahankan alamat IP yang dapat dirutekan. RFC1918

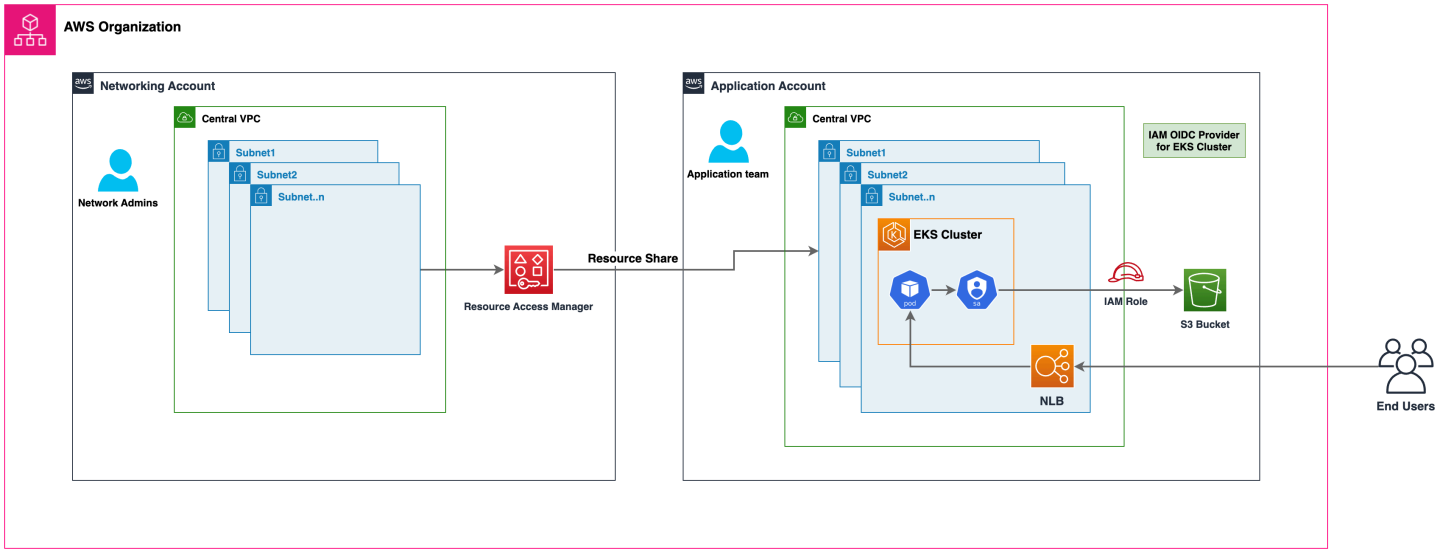


Pertimbangkan untuk menggunakan AWS PrivateLink, yang juga dikenal sebagai layanan endpoint, jika Anda adalah penyedia layanan dan ingin membagikan layanan dan ingress Kubernetes Anda (baik ALB atau NLB) dengan VPC pelanggan Anda di akun terpisah.

Berbagi VPC di beberapa akun

Banyak perusahaan mengadopsi Amazon bersama VPCs sebagai sarana untuk merampingkan administrasi jaringan, mengurangi biaya, dan meningkatkan keamanan di beberapa Akun AWS di Organisasi AWS. Mereka menggunakan AWS Resource Access Manager (RAM) untuk berbagi sumber daya AWS yang didukung secara aman dengan Akun AWS individual, unit organisasi (OUs), atau seluruh AWS Organization.

Anda dapat menerapkan kluster Amazon EKS, grup node terkelola, dan sumber daya AWS pendukung lainnya (seperti LoadBalancers, grup keamanan, titik akhir, dll.,) di Subnet VPC bersama dari Akun AWS lain menggunakan AWS RAM. Gambar di bawah ini menggambarkan contoh arsitektur tingkat tinggi. Hal ini memungkinkan tim jaringan pusat mengontrol konstruksi jaringan seperti VPCs, Subnet, dll., sambil memungkinkan tim aplikasi atau platform untuk menyebarkan kluster Amazon EKS di Akun AWS masing-masing. Panduan lengkap dari skenario ini tersedia di repositori github ini.



Pertimbangan saat menggunakan Subnet Bersama

- Cluster Amazon EKS dan node pekerja dapat dibuat dalam subnet bersama yang semuanya merupakan bagian dari VPC yang sama. Amazon EKS tidak mendukung pembuatan cluster di beberapa VPCs.
- Amazon EKS menggunakan AWS VPC Security Groups (SGs) untuk mengontrol lalu lintas antara bidang kontrol Kubernetes dan node pekerja klaster. Grup keamanan juga digunakan untuk mengontrol lalu lintas antara node pekerja, dan sumber daya VPC lainnya, dan alamat IP eksternal. Anda harus membuat grup keamanan ini di application/participant akun. Pastikan bahwa grup keamanan yang ingin Anda gunakan untuk pod Anda juga berada di akun peserta. Anda dapat mengonfigurasi aturan masuk dan keluar dalam grup keamanan Anda untuk mengizinkan lalu lintas yang diperlukan ke dan dari grup keamanan yang terletak di akun VPC Pusat.
- Buat peran IAM dan kebijakan terkait dalam akun peserta tempat klaster Amazon EKS Anda berada. Peran dan kebijakan IAM ini sangat penting untuk memberikan izin yang diperlukan ke klaster Kubernetes yang dikelola oleh Amazon EKS, serta node dan pod yang berjalan di Fargate. Izin memungkinkan Amazon EKS melakukan panggilan ke layanan AWS lain atas nama Anda.
- Anda dapat mengikuti pendekatan berikut untuk mengizinkan akses lintas Akun ke sumber daya AWS seperti bucket Amazon S3, tabel Dynamodb, dll., dari pod k8s:
  - Pendekatan kebijakan berbasis sumber daya: Jika layanan AWS mendukung kebijakan sumber daya, Anda dapat menambahkan kebijakan berbasis sumber daya yang sesuai untuk mengizinkan akses lintas akun ke Peran IAM yang ditetapkan ke pod kubernetes. Dalam skenario ini, penyedia OIDC, Peran IAM, dan kebijakan izin ada di akun aplikasi. Untuk menemukan Layanan AWS yang mendukung kebijakan berbasis Sumber Daya, rujuk layanan

AWS yang bekerja dengan IAM dan cari layanan yang memiliki Ya di kolom Berbasis Sumber Daya.

- Pendekatan Penyedia OIDC: Sumber daya IAM seperti Penyedia OIDC, Peran IAM, Izin, dan kebijakan Trust akan dibuat di Akun AWS peserta lain di mana sumber daya ada. Peran ini akan ditetapkan ke pod Kubernetes di akun aplikasi, sehingga mereka dapat mengakses sumber daya lintas akun. Lihat peran IAM Cross account untuk blog akun layanan Kubernetes untuk panduan lengkap tentang pendekatan ini.

- Anda dapat menerapkan resource Amazon Elastic Loadbalancer (ELB) (ALB atau NLB) untuk merutekan lalu lintas ke pod k8s baik di aplikasi maupun akun jaringan pusat. Lihat mengekspos Pod Amazon EKS Melalui panduan Load Balancer Lintas Akun untuk petunjuk terperinci tentang penerapan sumber daya ELB di akun jaringan pusat. Opsi ini menawarkan fleksibilitas yang ditingkatkan, karena memberikan akun Central Networking kontrol penuh atas konfigurasi keamanan sumber daya Load Balancer.
- Saat menggunakan `custom networking feature` Amazon VPC CNI, Anda perlu menggunakan pemetaan ID Availability Zone (AZ) yang tercantum di akun jaringan pusat untuk membuatnya. `ENIConfig` Ini karena pemetaan acak fisik AZs ke nama AZ di setiap akun AWS.

Grup Keamanan

Grup keamanan mengontrol lalu lintas yang diizinkan untuk mencapai dan meninggalkan sumber daya yang terkait dengannya. Amazon EKS menggunakan grup keamanan untuk mengelola komunikasi antara bidang kontrol dan node. Saat Anda membuat klaster, Amazon EKS membuat grup keamanan yang diberi nama`eks-cluster-sg-my-cluster-uniqueID`. EKS mengaitkan kelompok-kelompok keamanan ini ke yang dikelola ENIs dan node. Aturan default memungkinkan semua lalu lintas mengalir bebas antara cluster dan node Anda, dan memungkinkan semua lalu lintas keluar ke tujuan mana pun.

Saat membuat klaster, Anda dapat menentukan grup keamanan Anda sendiri. Silakan lihat rekomendasi untuk grup keamanan saat Anda menentukan grup keamanan sendiri.

Rekomendasi

Pertimbangkan Penerapan Multi-AZ

AWS Regions menyediakan beberapa Availability Zone (AZ) yang terpisah secara fisik dan terisolasi, yang terhubung dengan latensi rendah, throughput tinggi, dan jaringan yang sangat redundan. Dengan Availability Zones, Anda dapat merancang dan mengoperasikan aplikasi yang secara

otomatis gagal di antara Availability Zones tanpa gangguan. Amazon EKS sangat menyarankan untuk menerapkan kluster EKS ke beberapa zona ketersediaan. Harap pertimbangkan untuk menentukan subnet di setidaknya dua zona ketersediaan saat Anda membuat cluster.

Kubelet yang berjalan pada node secara otomatis menambahkan label ke objek node seperti. `topology.kubernetes.io/region=us-west-2` Kami merekomendasikan untuk menggunakan label node dalam hubungannya dengan batasan penyebaran topologi Pod untuk mengontrol bagaimana Pod tersebar di seluruh zona. Petunjuk ini memungkinkan penjadwal Kubernetes untuk menempatkan Pod untuk ketersediaan yang diharapkan lebih baik, mengurangi risiko kegagalan yang berkorelasi memengaruhi seluruh beban kerja Anda. Silakan lihat Menetapkan Node ke Pod untuk melihat contoh untuk pemilih node dan batasan spread AZ.

Anda dapat menentukan subnet atau zona ketersediaan saat Anda membuat node. Node ditempatkan di subnet cluster jika tidak ada subnet yang dikonfigurasi. Dukungan EKS untuk grup node terkelola secara otomatis menyebarkan node di beberapa zona ketersediaan pada kapasitas yang tersedia. Karpenter akan menghormati penempatan spread AZ dengan menskalakan node untuk ditentukan AZs jika beban kerja menentukan batas penyebaran topologi.

AWS Elastic Load Balancer dikelola oleh AWS Load Balancer Controller untuk klaster Kubernetes. Ini menyediakan Application Load Balancer (ALB) untuk sumber daya ingress Kubernetes dan Network Load Balancer (NLB) untuk layanan Kubernetes tipe Loadbalancer. Pengontrol Elastic Load Balancer menggunakan tag untuk menemukan subnet. Pengontrol ELB membutuhkan minimal dua zona ketersediaan (AZs) untuk menyediakan sumber daya masuk dengan sukses. Pertimbangkan pengaturan subnet di setidaknya dua AZs untuk memanfaatkan keamanan dan keandalan redundansi geografis.

Menyebarkan Node ke Subnet Pribadi

VPC yang mencakup subnet pribadi dan publik adalah metode ideal untuk menerapkan beban kerja Kubernetes di EKS. Pertimbangkan untuk menetapkan minimal dua subnet publik dan dua subnet pribadi dalam dua zona ketersediaan yang berbeda. Tabel rute terkait subnet publik berisi rute ke gateway internet. Pod dapat berinteraksi dengan Internet melalui gateway NAT. Subnet pribadi didukung oleh gateway internet khusus egres di lingkungan (EIGW). IPv6

Membuat instantiasi node dalam subnet pribadi menawarkan kontrol maksimal atas lalu lintas ke node dan efektif untuk sebagian besar aplikasi Kubernetes. Sumber daya masuk (seperti penyeimbang beban) dipakai di subnet publik dan mengarahkan lalu lintas ke Pod yang beroperasi pada subnet pribadi.

Pertimbangkan mode pribadi saja jika Anda menuntut keamanan yang ketat dan isolasi jaringan. Dalam konfigurasi ini, tiga subnet pribadi digunakan di Availability Zone yang berbeda dalam VPC Wilayah AWS. Sumber daya yang dikerahkan ke subnet tidak dapat mengakses internet, internet juga tidak dapat mengakses sumber daya dalam subnet. Agar aplikasi Kubernetes Anda dapat mengakses layanan AWS lainnya, Anda harus mengonfigurasi titik akhir gateway PrivateLink antarmuka and/or . Anda dapat mengatur penyeimbang beban internal untuk mengarahkan lalu lintas ke Pod menggunakan AWS Load Balancer Controller. Subnet pribadi harus diberi tag (`kubernetes.io/role/internal-elb: 1`) agar pengontrol menyediakan penyeimbang beban. Agar node dapat mendaftar dengan cluster, titik akhir cluster harus diatur ke mode pribadi. Silakan kunjungi panduan klaster pribadi untuk persyaratan dan pertimbangan lengkap.

Pertimbangkan Mode Publik dan Pribadi untuk Titik Akhir Cluster

Amazon EKS menawarkan mode titik akhir klaster khusus publik public-and-private, dan pribadi. Mode default hanya publik, namun kami menyarankan untuk mengonfigurasi titik akhir cluster dalam mode publik dan pribadi. Opsi ini memungkinkan panggilan Kubernetes API di dalam VPC klaster Anda (seperti node-to-control-plane komunikasi) untuk memanfaatkan titik akhir dan lalu lintas VPC pribadi agar tetap berada dalam VPC klaster Anda. Server API cluster Anda, di sisi lain, dapat dijangkau dari internet. Namun, kami sangat menyarankan untuk membatasi blok CIDR yang dapat menggunakan titik akhir publik. Pelajari cara mengonfigurasi akses titik akhir publik dan pribadi, termasuk membatasi blok CIDR.

Kami menyarankan titik akhir khusus pribadi ketika Anda memerlukan keamanan dan isolasi jaringan. Sebaiknya gunakan salah satu opsi yang tercantum dalam panduan pengguna EKS untuk terhubung ke server API secara pribadi.

Konfigurasikan Grup Keamanan dengan Hati-hati

Amazon EKS mendukung penggunaan grup keamanan khusus. Setiap grup keamanan khusus harus mengizinkan komunikasi antara node dan bidang kontrol Kubernetes. Silakan periksa persyaratan port dan konfigurasikan aturan secara manual ketika organisasi Anda tidak mengizinkan komunikasi terbuka.

EKS menerapkan grup keamanan kustom yang Anda berikan selama pembuatan klaster ke antarmuka terkelola (X-ENIs). Namun, itu tidak segera mengaitkannya dengan node. Saat membuat grup node, sangat disarankan untuk mengaitkan grup keamanan khusus secara manual. Harap pertimbangkan untuk mengaktifkan securityGroupSelectorKetentuan untuk mengaktifkan penemuan templat simpul Karpenter dari grup keamanan khusus selama penskalaan otomatis node.

Kami sangat menyarankan membuat grup keamanan untuk memungkinkan semua lalu lintas komunikasi antar simpul. Selama proses bootstrap, node memerlukan konektivitas Internet keluar untuk mengakses titik akhir cluster. Evaluasi persyaratan akses luar, seperti koneksi on-premise dan akses registri kontainer, dan tetapkan aturan dengan tepat. Sebelum memasukkan perubahan ke dalam produksi, kami sangat menyarankan agar Anda memeriksa koneksi dengan cermat di lingkungan pengembangan Anda.

Terapkan Gateway NAT di setiap Availability Zone

Jika Anda menerapkan node di subnet pribadi (IPv4 dan IPv6), pertimbangkan untuk membuat Gateway NAT di setiap Availability Zone (AZ) untuk memastikan arsitektur zona independen dan mengurangi pengeluaran lintas AZ. Setiap gateway NAT di AZ diimplementasikan dengan redundansi.

Amazon VPC CNI

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Amazon EKS mengimplementasikan jaringan cluster melalui plugin Amazon VPC Container Network Interface, juga dikenal sebagai VPC CNI. Plugin CNI memungkinkan Kubernetes Pods memiliki alamat IP yang sama seperti yang mereka lakukan di jaringan VPC. Lebih khusus lagi, semua kontainer di dalam Pod berbagi namespace jaringan, dan mereka dapat berkomunikasi satu sama lain menggunakan port lokal.

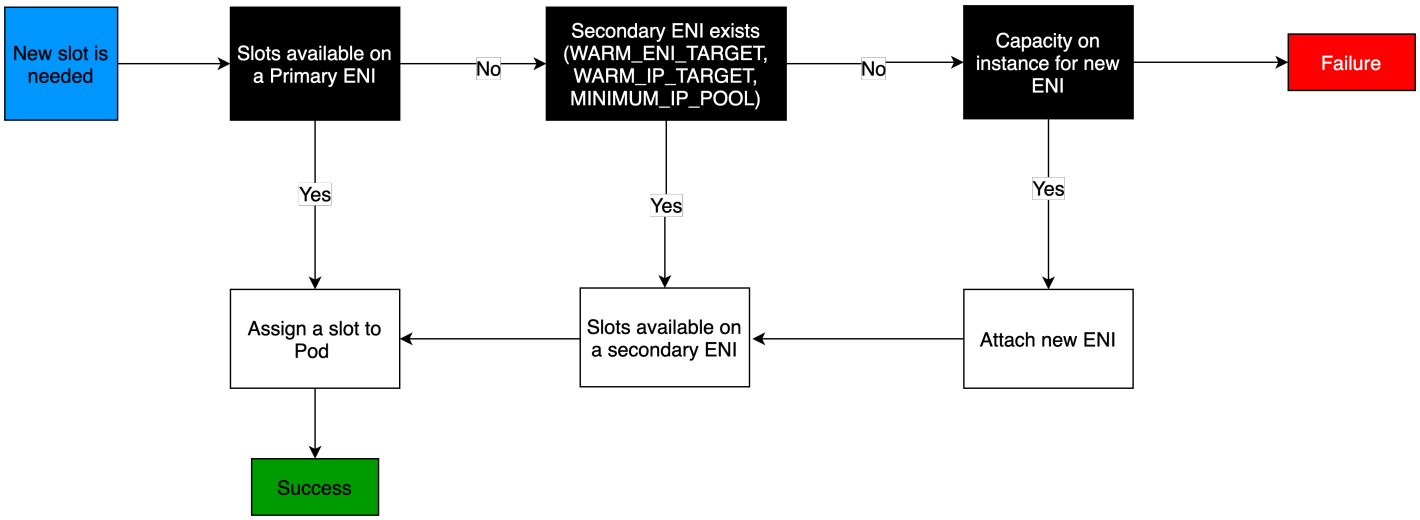
Amazon VPC CNI memiliki dua komponen:

- CNI Binary, yang akan mengatur jaringan Pod untuk mengaktifkan Pod-to-Pod komunikasi. Biner CNI berjalan pada sistem file root node dan dipanggil oleh kubelet ketika Pod baru ditambahkan, atau Pod yang sudah ada dihapus dari node.
- ipamd, daemon Manajemen Alamat IP (IPAM) node-lokal yang sudah berjalan lama dan bertanggung jawab untuk:
  - mengelola ENIs pada node, dan

- mempertahankan kumpulan hangat dari alamat IP atau awalan yang tersedia

Ketika sebuah instance dibuat, EC2 membuat dan melampirkan ENI primer yang terkait dengan subnet primer. Subnet utama mungkin publik atau pribadi. Pod yang berjalan dalam mode HostNetwork menggunakan alamat IP utama yang ditetapkan ke ENI primer node dan berbagi namespace jaringan yang sama dengan host.

Plugin CNI mengelola Elastic Network Interfaces (ENI) pada node. Ketika sebuah node disediakan, plugin CNI secara otomatis mengalokasikan kumpulan slot (IPs atau Prefix) dari subnet node ke ENI primer. Kolam ini dikenal sebagai kolam hangat, dan ukurannya ditentukan oleh tipe instance node. Tergantung pada pengaturan CNI, slot dapat berupa alamat IP atau awalan. Ketika slot pada ENI telah ditetapkan, CNI dapat melampirkan tambahan ENIs dengan kolam hangat slot ke node. Tambahan ini ENIs disebut Sekunder ENIs. Setiap ENI hanya dapat mendukung sejumlah slot tertentu, berdasarkan jenis instans. CNI lebih banyak ENIs menempel pada instance berdasarkan jumlah slot yang dibutuhkan, yang biasanya sesuai dengan jumlah Pod. Proses ini berlanjut sampai node tidak dapat lagi mendukung ENI tambahan. CNI juga mengalokasikan “warm” ENIs dan slot untuk startup Pod yang lebih cepat. Perhatikan setiap jenis instance memiliki jumlah maksimum ENIs yang dapat dilampirkan. Ini adalah salah satu kendala pada kepadatan Pod (jumlah Pod per node), selain sumber daya komputasi.



Jumlah maksimum antarmuka jaringan, dan jumlah maksimum slot yang dapat Anda gunakan bervariasi menurut jenis Instans EC2. Karena setiap Pod menggunakan alamat IP pada slot, jumlah Pod yang dapat Anda jalankan pada Instance EC2 tertentu tergantung pada berapa banyak yang ENIs dapat dilampirkan padanya dan berapa banyak slot yang didukung setiap ENI. Kami menyarankan pengaturan Pod maksimum per panduan pengguna EKS untuk menghindari kehabisan

sumber daya CPU dan memori instans. Pod `hostNetwork` yang menggunakan dikecualikan dari perhitungan ini. Anda dapat mempertimbangkan untuk menggunakan skrip yang disebut max-pod-calculator.sh untuk menghitung Pod maksimum yang direkomendasikan EKS untuk jenis instance tertentu.

Ikhtisar

Mode IP sekunder adalah mode default untuk VPC CNI. Panduan ini memberikan gambaran umum tentang perilaku CNI VPC saat mode IP Sekunder diaktifkan. Fungsionalitas ipamd (alokasi alamat IP) dapat bervariasi tergantung pada pengaturan konfigurasi untuk VPC CNI, seperti,, dan. the section called “Mode Awalan untuk Linux” the section called “Grup Keamanan Per Pod” the section called “Jaringan Kustom”

Amazon VPC CNI digunakan sebagai Kubernetes Daemonset bernama aws-node pada node pekerja. Ketika node pekerja disediakan, ia memiliki ENI default, yang disebut ENI primer, melekat padanya. CNI mengalokasikan kumpulan hangat ENIs dan alamat IP sekunder dari subnet yang melekat pada ENI primer node. Secara default, ipamd mencoba mengalokasikan ENI tambahan ke node. IPAMD mengalokasikan ENI tambahan ketika satu Pod dijadwalkan dan diberi alamat IP sekunder dari ENI primer. ENI “hangat” ini memungkinkan jaringan Pod yang lebih cepat. Karena kumpulan alamat IP sekunder habis, CNI menambahkan ENI lain untuk menetapkan lebih banyak.

Jumlah ENIs dan alamat IP dalam kumpulan dikonfigurasi melalui variabel lingkungan yang disebut WARM_ENI_TARGET, WARM_IP_TARGET, MINIMUM_IP_TARGET. `aws-node`Daemonset akan secara berkala memeriksa bahwa jumlah yang cukup ENIs terlampir. Jumlah yang cukup ENIs terlampir ketika semua`WARM_ENI_TARGET`, atau `WARM_IP_TARGET` dan `MINIMUM_IP_TARGET` kondisi terpenuhi. Jika ada ENI yang tidak mencukupi, CNI akan melakukan panggilan API ke EC2 untuk melampirkan lebih banyak hingga `MAX_ENI` batas tercapai.

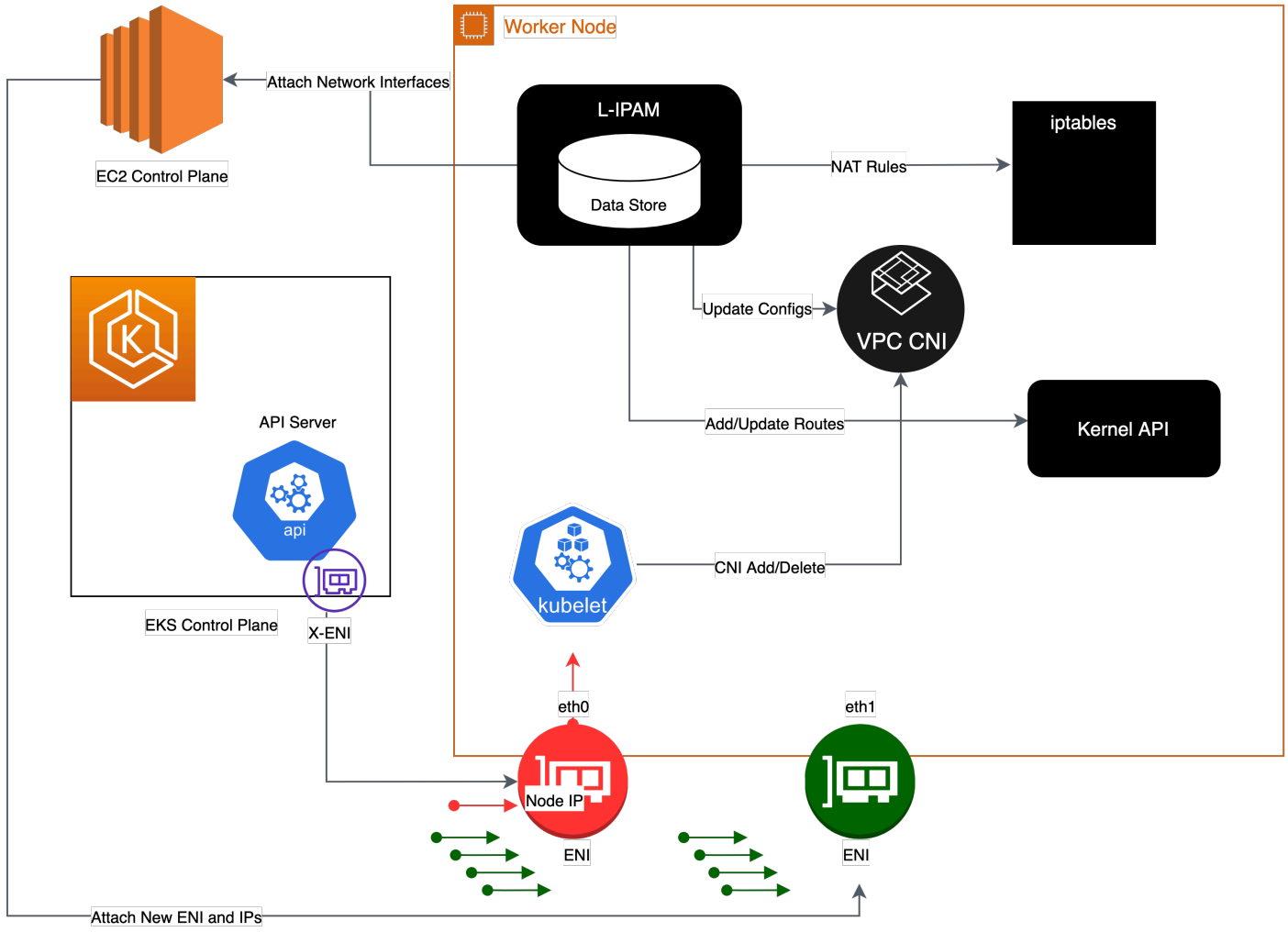
- `WARM_ENI_TARGET`- Integer, Nilai lebih besar dari 0 menunjukkan persyaratan Diaktifkan
  - Jumlah hangat yang ENIs harus dipertahankan. ENI adalah “hangat” ketika dilekatkan sebagai ENI sekunder ke sebuah node, tetapi tidak digunakan oleh Pod manapun. Lebih khusus lagi, tidak ada alamat IP ENI yang dikaitkan dengan Pod.
  - Contoh: Pertimbangkan sebuah instance dengan 2 ENIs, masing-masing ENI mendukung 5 alamat IP. WARM_ENI_TARGET diatur ke 1. Jika tepat 5 alamat IP dikaitkan dengan instance, CNI mempertahankan 2 yang ENIs melekat pada instance. ENI pertama sedang digunakan, dan semua 5 alamat IP yang mungkin dari ENI ini digunakan. ENI kedua adalah “hangat” dengan semua 5 alamat IP di kolam renang. Jika Pod lain diluncurkan pada instance, alamat IP ke-6 akan dibutuhkan. CNI akan menetapkan Pod ke-6 ini alamat IP dari ENI kedua dan dari 5 IPs

dari kolam. ENI kedua sekarang digunakan, dan tidak lagi dalam status “hangat”. CNI akan mengalokasikan ENI ke-3 untuk mempertahankan setidaknya 1 ENI hangat.

> **Note**
>
> Yang hangat ENIs masih mengkonsumsi alamat IP dari CIDR VPC Anda. Alamat IP adalah “tidak terpakai” atau “hangat” sampai mereka dikaitkan dengan beban kerja, seperti Pod.

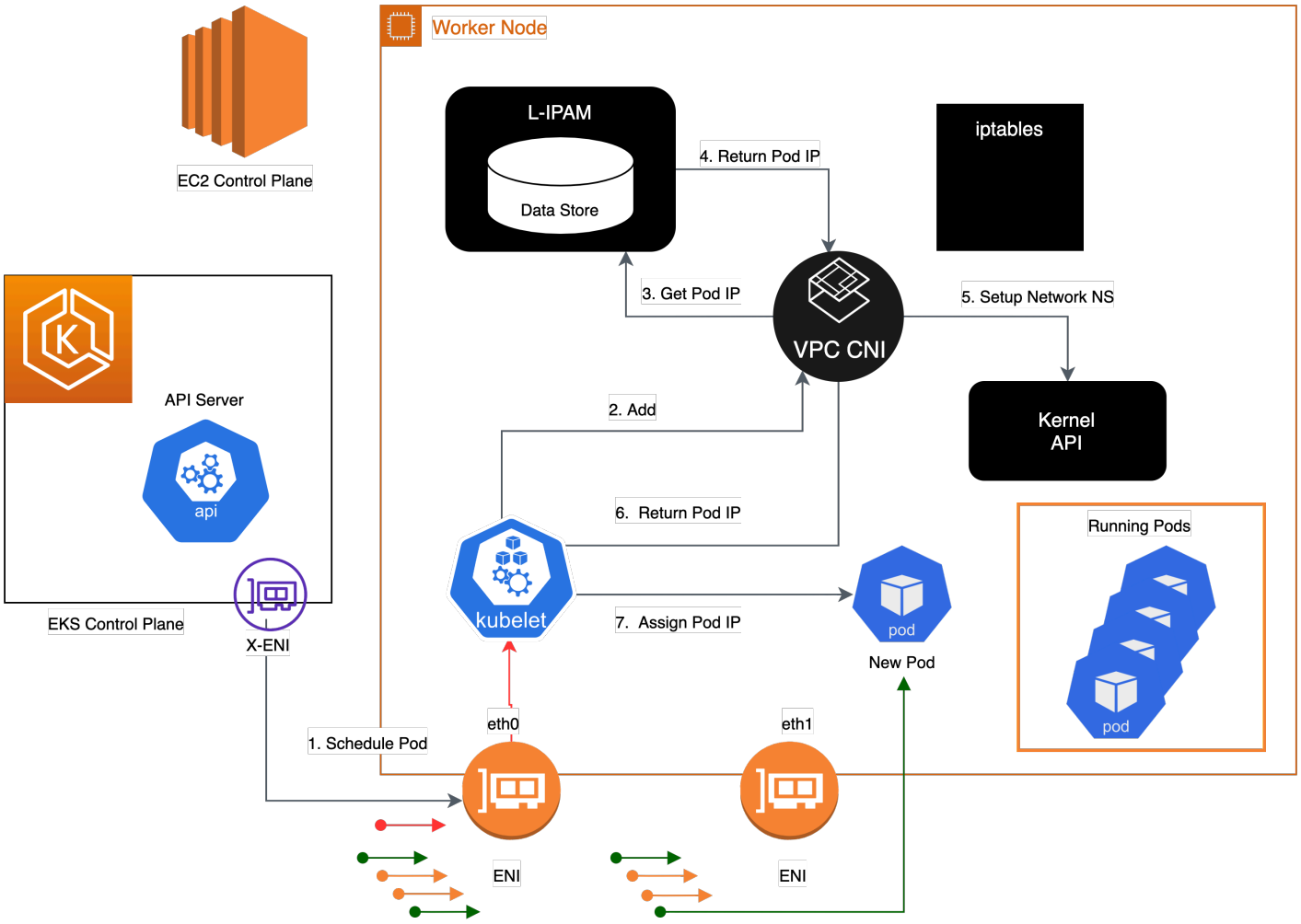
- `WARM_IP_TARGET`, Integer, Nilai lebih besar dari 0 menunjukkan persyaratan Diaktifkan
  - Jumlah alamat IP Warm yang harus dipertahankan. IP Warm tersedia pada ENI yang terpasang secara aktif, tetapi belum ditetapkan ke Pod. Dengan kata lain, jumlah Warm IPs yang tersedia adalah jumlah IPs yang dapat ditetapkan ke Pod tanpa memerlukan ENI tambahan.
- Contoh: Pertimbangkan contoh dengan 1 ENI, masing-masing ENI mendukung 20 alamat IP. WARM_IP_TARGET diatur ke 5. WARM_ENI_TARGET diatur ke 0. Hanya 1 ENI yang akan dilampirkan sampai alamat IP ke-16 diperlukan. Kemudian, CNI akan melampirkan ENI kedua, mengkonsumsi 20 kemungkinan alamat dari subnet CIDR.
- `MINIMUM_IP_TARGET`, Integer, Nilai lebih besar dari 0 menunjukkan persyaratan Diaktifkan
  - Jumlah minimum alamat IP yang akan dialokasikan setiap saat. Hal ini biasanya digunakan untuk front-load penugasan beberapa ENIs saat peluncuran instance.
  - Contoh: Pertimbangkan instance yang baru diluncurkan. Ini memiliki 1 ENI dan setiap ENI mendukung 10 alamat IP. MINIMUM_IP_TARGET diatur ke 100. ENI segera melampirkan 9 lagi ENIs untuk total 100 alamat. Ini terjadi terlepas dari nilai WARM_IP_TARGET atau WARM_ENI_TARGET.

Proyek ini mencakup Dokumen Excel Kalkulator Subnet. Dokumen kalkulator ini mensimulasikan konsumsi alamat IP dari beban kerja tertentu di bawah opsi konfigurasi ENI yang berbeda, seperti `WARM_IP_TARGET` dan. `WARM_ENI_TARGET`

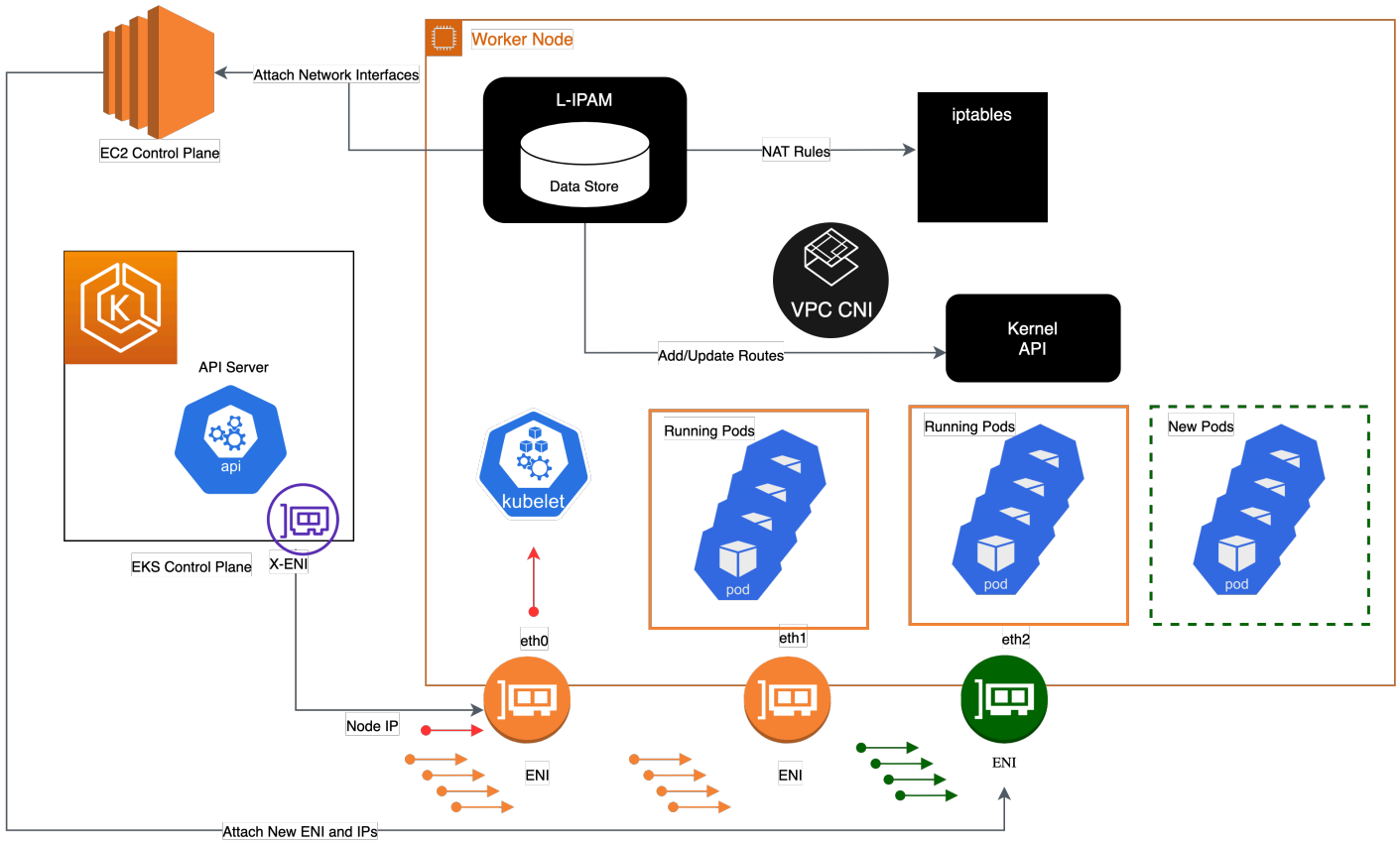


Ketika Kubelet menerima permintaan add Pod, biner CNI menanyakan ipamd untuk alamat IP yang tersedia, yang kemudian diberikan ipamd ke Pod. Biner CNI menghubungkan jaringan host dan Pod.

Pod yang digunakan pada sebuah node, secara default, ditetapkan ke grup keamanan yang sama dengan ENI utama. Atau, Pod dapat dikonfigurasi dengan grup keamanan yang berbeda.



Saat kumpulan alamat IP habis, plugin secara otomatis melampirkan antarmuka jaringan elastis lain ke instans dan mengalokasikan set alamat IP sekunder lainnya ke antarmuka itu. Proses ini berlanjut sampai simpul tidak dapat lagi mendukung antarmuka jaringan elastis tambahan.



Ketika sebuah Pod dihapus, VPC CNI menempatkan alamat IP Pod dalam cache pendinginan 30 detik. Cache IPs dalam cool down tidak ditetapkan ke Pod baru. Ketika periode pendinginan selesai, VPC CNI memindahkan Pod IP kembali ke kolam hangat. Periode pendinginan mencegah alamat IP Pod didaur ulang sebelum waktunya dan memungkinkan kube-proxy pada semua node cluster untuk menyelesaikan pembaruan aturan iptables. Ketika jumlah IPs atau ENIs melebihi jumlah pengaturan kolam hangat, plugin ipamd kembali IPs dan ENIs ke VPC.

Seperti dijelaskan di atas dalam mode IP Sekunder, setiap Pod menerima satu alamat IP pribadi sekunder dari salah satu ENI yang dilampirkan ke sebuah instance. Karena setiap Pod menggunakan alamat IP, jumlah Pod yang dapat Anda jalankan pada Instans EC2 tertentu tergantung pada berapa banyak yang ENIs dapat dilampirkan padanya dan berapa banyak alamat IP yang didukungnya. VPC CNI memeriksa batas file untuk mengetahui berapa banyak ENIs dan alamat IP yang diizinkan untuk setiap jenis instance.

Anda dapat menggunakan rumus berikut untuk menentukan jumlah maksimum Pod yang dapat Anda gunakan pada sebuah node.

```
(Number of network interfaces for the instance type * (the number of IP addresses per
 network interface - 1)) + 2
```

+2 menunjukkan Pod yang membutuhkan jaringan host, seperti kube-proxy dan VPC CNI. Amazon EKS membutuhkan kube-proxy dan VPC CNI untuk beroperasi pada setiap node, dan persyaratan ini diperhitungkan dalam nilai max-pod. Jika Anda ingin menjalankan pod jaringan host tambahan, pertimbangkan untuk memperbarui nilai max-pod. Anda dapat menentukan `--kubelet-extra-args "—max-pods=110"` sebagai data pengguna di template peluncuran.

Sebagai contoh, pada cluster dengan 3 node c5.large (3 ENIs dan max 10 IPs per ENI), ketika cluster dimulai dan memiliki 2 pod CoreDNS, CNI akan mengkonsumsi 50 alamat IP dan menyimpan 43 di kolam hangat. IPs Kolam hangat memungkinkan peluncuran Pod lebih cepat saat aplikasi dikerahkan.

Node 1 (dengan pod CoreDNS): ENIs 2, 20 ditetapkan IPs

Node 2 (dengan pod CoreDNS): ENIs 2, 20 ditetapkan IPs

Node 3 (tanpa Pod): 1 ENI. 10 IPs ditugaskan.

Untuk Node 1 dan Node 2 (konfigurasi identik):

- 2 ENIs × 10 IPs per ENI = 20 IPs total
- Kurangi 2 primer IPs (1 per ENI) = 18 IPs
- Kurangi 1 IP untuk pod CoreDNS = 17 tersedia IPs
- Jadi masing-masing node ini memiliki 17 IPs di kolam hangat

Untuk Node 3:

- 1 ENI × 10 IPs = 10 IPs total
- Kurangi 1 IP primer = 9 IPs tersedia di kolam hangat

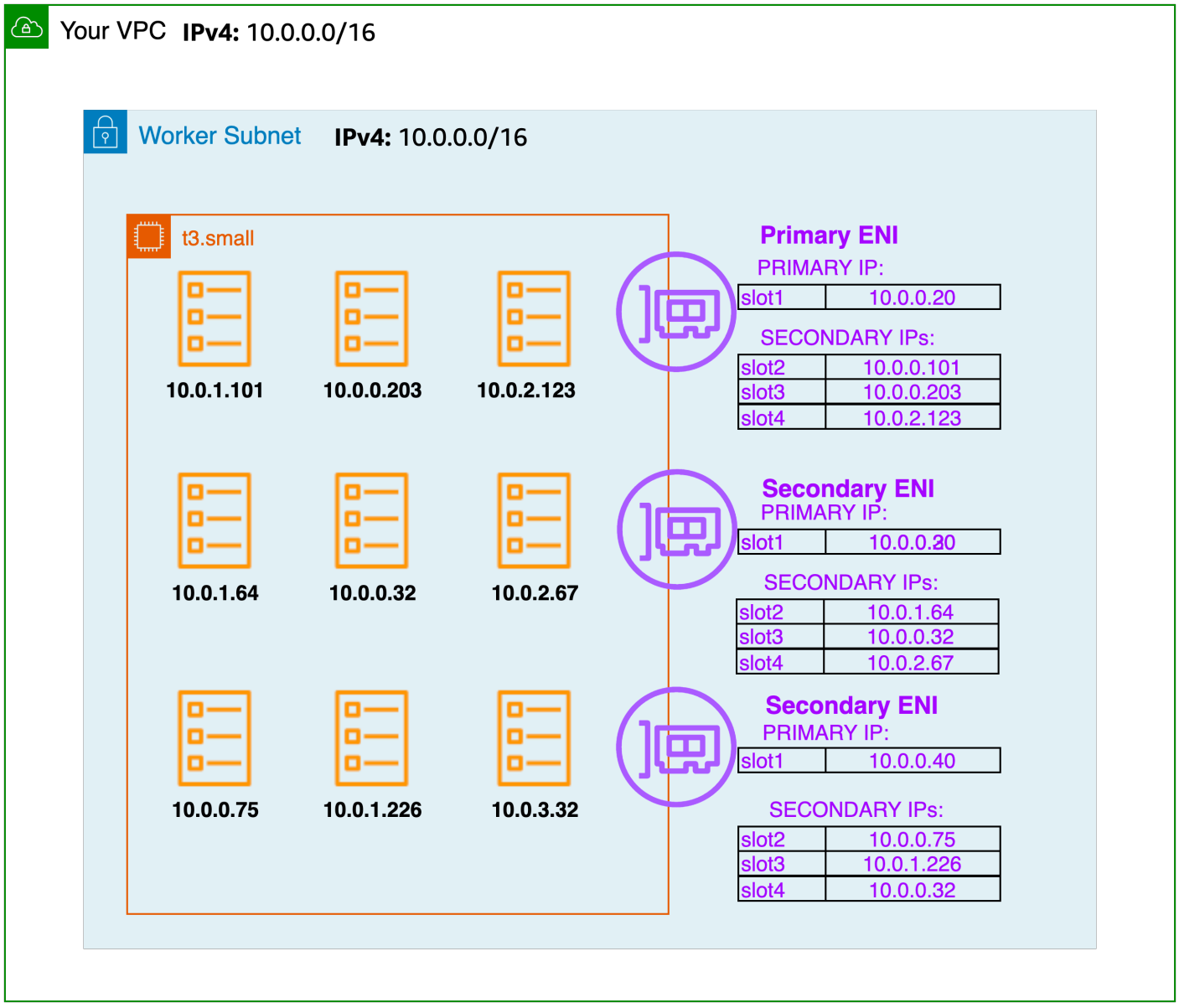
Total perhitungan kolam hangat: - 17 (Node 1) + 17 (Node 2) + 9 (Node 3) = 43 IPs

Perlu diingat bahwa pod infrastruktur, yang sering berjalan sebagai kumpulan daemon, masing-masing berkontribusi pada jumlah max-pod. Ini dapat mencakup:

- CoreDNS

- Amazon Elastis LoadBalancer
- Pod operasional untuk server metrik

Kami menyarankan Anda merencanakan infrastruktur Anda dengan menggabungkan kapasitas Pods ini. Untuk daftar jumlah maksimum Pod yang didukung oleh setiap jenis instance, lihat eni-max-pods.txt di. GitHub



Rekomendasi

Menyebarkan kluster EKS dengan Mode Otomatis

Saat Anda menggunakan Mode Otomatis EKS untuk membuat klaster, AWS mengelola konfigurasi VPC Container Network Interface (CNI) untuk cluster Anda. Dengan Amazon EKS Auto Mode, Anda tidak perlu menginstal atau meningkatkan add-on jaringan. Namun, pastikan beban kerja Anda kompatibel dengan konfigurasi CNI VPC yang dikelola.

Menyebarkan Add-On Terkelola VPC CNI

Saat Anda menyediakan klaster, Amazon EKS menginstal VPC CNI secara otomatis. Amazon EKS tetap mendukung add-on terkelola yang memungkinkan cluster berinteraksi dengan sumber daya AWS yang mendasarinya seperti komputasi, penyimpanan, dan jaringan. Kami sangat menyarankan Anda menerapkan cluster dengan add-on terkelola termasuk VPC CNI.

Add-on terkelola Amazon EKS menawarkan instalasi dan manajemen VPC CNI untuk kluster Amazon EKS. Add-on Amazon EKS menyertakan patch keamanan terbaru, perbaikan bug, dan divalidasi oleh AWS untuk bekerja dengan Amazon EKS. Add-on VPC CNI memungkinkan Anda untuk terus memastikan keamanan dan stabilitas kluster Amazon EKS Anda dan mengurangi jumlah upaya yang diperlukan untuk menginstal, mengonfigurasi, dan memperbarui add-on. Selain itu, add-on terkelola dapat ditambahkan, diperbarui, atau dihapus melalui Amazon EKS API, AWS Management Console, AWS CLI, dan eksctl.

Anda dapat menemukan bidang terkelola VPC CNI menggunakan `--show-managed-fields` bendera dengan perintah. `kubectl get`

```
kubectl get daemonset aws-node --show-managed-fields -n kube-system -o yaml
```

Add-on terkelola mencegah penyimpangan konfigurasi dengan menimpa konfigurasi secara otomatis setiap 15 menit. Ini berarti bahwa setiap perubahan pada add-on terkelola, yang dilakukan melalui Kubernetes API setelah pembuatan add-on, akan menimpa oleh proses pencegahan drift otomatis dan juga disetel ke default selama proses pembaruan add-on.

Bidang yang dikelola oleh EKS terdaftar di bawah ManagedFields dengan manajer sebagai EKS. Bidang yang dikelola oleh EKS mencakup akun layanan, gambar, url gambar, probe keaktifan, probe kesiapan, label, volume, dan pemasangan volume.

> **Note**
>
> Bidang yang paling sering digunakan seperti WARM_ENI_TARGET, WARM_IP_TARGET, dan MINIMUM_IP_TARGET tidak dikelola dan tidak akan direkonsiliasi. Perubahan pada bidang ini akan dipertahankan setelah memperbarui add-on.

Kami menyarankan untuk menguji perilaku add-on di kluster non-produksi Anda untuk konfigurasi tertentu sebelum memperbarui kluster produksi. Selain itu, ikuti langkah-langkah dalam panduan pengguna EKS untuk konfigurasi add-on.

Migrasi ke Add-On Terkelola

Anda akan mengelola kompatibilitas versi dan memperbarui tambalan keamanan CNI VPC yang dikelola sendiri. Untuk memperbarui add-on yang dikelola sendiri, Anda harus menggunakan Kubernetes APIs dan instruksi yang diuraikan dalam panduan pengguna EKS. Kami merekomendasikan migrasi ke add-on terkelola untuk kluster EKS yang ada dan sangat menyarankan untuk membuat cadangan pengaturan CNI Anda saat ini sebelum migrasi. Untuk mengonfigurasi add-on terkelola, Anda dapat menggunakan Amazon EKS API, AWS Management Console, atau AWS Command Line Interface.

```
kubectl apply view-last-applied daemonset aws-node -n kube-system  aws-k8s-cni-old.yaml
```

Amazon EKS akan mengganti pengaturan konfigurasi CNI jika bidang terdaftar sebagai dikelola dengan pengaturan default. Kami berhati-hati agar tidak memodifikasi bidang yang dikelola. Add-on tidak merekonsiliasi bidang konfigurasi seperti variabel lingkungan hangat dan mode CNI. Pod dan aplikasi akan terus berjalan saat Anda bermigrasi ke CNI yang dikelola.

Backup Pengaturan CNI Sebelum Pembaruan

VPC CNI berjalan pada pesawat data pelanggan (node), dan karenanya Amazon EKS tidak secara otomatis memperbarui add-on (dikelola dan dikelola sendiri) ketika versi baru dirilis atau setelah Anda memperbarui klaster Anda ke versi minor Kubernetes yang baru. Untuk memperbarui add-on untuk kluster yang ada, Anda harus memicu pembaruan melalui update-addon API atau mengklik tautan pembaruan sekarang di konsol EKS untuk add-on. Jika Anda telah menerapkan add-on yang dikelola sendiri, ikuti langkah-langkah yang disebutkan dalam memperbarui add-on VPC CNI yang dikelola sendiri.

Kami sangat menyarankan Anda memperbarui satu versi minor sekaligus. Misalnya, jika versi minor Anda saat ini `1.9` dan Anda ingin memperbaruinya`1.11`, Anda harus memperbarui ke versi patch terbaru `1.10` terlebih dahulu, lalu perbarui ke versi patch terbaru`1.11`.

Lakukan inspeksi Daemonset aws-node sebelum memperbarui Amazon VPC CNI. Ambil cadangan pengaturan yang ada. Jika menggunakan add-on terkelola, konfirmasikan bahwa Anda belum memperbarui pengaturan apa pun yang mungkin diganti Amazon EKS. Kami merekomendasikan kait pembaruan pos dalam alur kerja otomatisasi Anda atau langkah penerapan manual setelah pembaruan add-on.

```
kubectl apply view-last-applied daemonset aws-node -n kube-system  aws-k8s-cni-old.yaml
```

Untuk add-on yang dikelola sendiri, bandingkan cadangan dengan `releases` on GitHub untuk melihat versi yang tersedia dan biasakan diri Anda dengan perubahan dalam versi yang ingin Anda perbarui. Sebaiknya gunakan Helm untuk mengelola add-on yang dikelola sendiri dan memanfaatkan file nilai untuk menerapkan pengaturan. Setiap operasi pembaruan yang melibatkan penghapusan Daemonset akan mengakibatkan downtime aplikasi dan harus dihindari.

Memahami Konteks Keamanan

Kami sangat menyarankan Anda untuk memahami konteks keamanan yang dikonfigurasi untuk mengelola VPC CNI secara efisien. Amazon VPC CNI memiliki dua komponen biner CNI dan ipamd (aws-node) Daemonset. CNI berjalan sebagai biner pada node dan memiliki akses ke sistem file root node, juga memiliki akses istimewa karena berhubungan dengan iptables di tingkat node. Biner CNI dipanggil oleh kubelet ketika Pod ditambahkan atau dihapus.

Daemonset aws-node adalah proses yang berjalan lama yang bertanggung jawab untuk manajemen alamat IP di tingkat node. Aws-node berjalan dalam `hostNetwork` mode dan memungkinkan akses ke perangkat loopback, dan aktivitas jaringan pod lain pada node yang sama. Init-container aws-node berjalan dalam mode privileged dan memasang soket CRI yang memungkinkan Daemonset untuk memantau penggunaan IP oleh Pod yang berjalan pada node. Amazon EKS sedang bekerja untuk menghapus persyaratan istimewa dari aws-node init container. Selain itu, aws-node perlu memperbarui entri NAT dan memuat modul iptables dan karenanya berjalan dengan hak istimewa NET_ADMIN.

Amazon EKS merekomendasikan penerapan kebijakan keamanan seperti yang ditentukan oleh manifes aws-node untuk manajemen IP untuk pengaturan Pod dan jaringan. Harap pertimbangkan untuk memperbarui ke versi terbaru VPC CNI. Selain itu, harap pertimbangkan untuk membuka GitHub masalah jika Anda memiliki persyaratan keamanan tertentu.

Gunakan peran IAM terpisah untuk CNI

AWS VPC CNI memerlukan izin AWS Identity and Access Management (IAM). Kebijakan CNI perlu diatur sebelum peran IAM dapat digunakan. Anda dapat menggunakan `AmazonEKS_CNI_Policy`, yang merupakan kebijakan terkelola AWS untuk IPv4 klaster. Kebijakan terkelola Amazoneks CNI hanya memiliki izin untuk klaster. IPv4 Anda harus membuat kebijakan IAM terpisah untuk IPv6 cluster dengan izin yang tercantum di sini.

Secara default, VPC CNI mewarisi peran IAM node Amazon EKS (baik grup node terkelola maupun yang dikelola sendiri).

Sangat disarankan untuk mengonfigurasi peran IAM terpisah dengan kebijakan yang relevan untuk Amazon VPC CNI. Jika tidak, pod Amazon VPC CNI mendapatkan izin yang ditetapkan ke peran IAM node dan memiliki akses ke profil instance yang ditetapkan ke node.

Plugin VPC CNI membuat dan mengonfigurasi akun layanan yang disebut aws-node. Secara default, akun layanan mengikat peran IAM node Amazon EKS dengan kebijakan Amazon EKS CNI terlampir. Untuk menggunakan peran IAM terpisah, kami sarankan Anda membuat akun layanan baru dengan kebijakan Amazon EKS CNI terlampir. Untuk menggunakan akun layanan baru, Anda harus menerapkan ulang pod CNI. Pertimbangkan untuk menentukan add-on terkelola `--service-account-role-arn` untuk VPC CNI saat membuat cluster baru. Pastikan Anda menghapus kebijakan Amazon EKS CNI untuk keduanya IPv4 dan IPv6 dari peran node Amazon EKS.

Disarankan agar Anda memblokir metadata instance akses untuk meminimalkan radius ledakan pelanggaran keamanan.

Menangani kegagalan Liveness/Readiness Probe

Kami menyarankan untuk meningkatkan nilai batas waktu probe keaktifan dan kesiapan (default`timeoutSeconds: 10`) untuk EKS 1.20 dan cluster yang lebih baru untuk mencegah kegagalan probe menyebabkan Pod aplikasi Anda macet dalam status ContainerCreating. Masalah ini telah terlihat pada cluster data intensif dan pemrosesan batch. Penggunaan CPU yang tinggi menyebabkan kegagalan kesehatan probe aws-node, yang menyebabkan permintaan CPU Pod tidak terpenuhi. Selain memodifikasi batas waktu probe, pastikan bahwa permintaan sumber daya CPU (default`CPU: 25m`) untuk aws-node dikonfigurasi dengan benar. Kami tidak menyarankan memperbarui pengaturan kecuali node Anda mengalami masalah.

Kami sangat menyarankan Anda untuk menjalankan sudo `bash /opt/cni/bin/aws-cni-support.sh` pada node saat Anda menggunakan dukungan Amazon EKS. Script akan membantu

dalam mengevaluasi log kubelet dan pemanfaatan memori pada node. Harap pertimbangkan untuk menginstal Agen SSM di node pekerja Amazon EKS untuk menjalankan skrip.

Konfigurasi Kebijakan IPTables Penerusan pada Instans AMI yang Dioptimalkan Non-EKS

Jika Anda menggunakan AMI kustom, pastikan untuk mengatur kebijakan penerusan iptables ke ACCEPT di bawah kubelet.service. Banyak sistem menetapkan kebijakan iptables forward ke DROP. Anda dapat membuat AMI kustom menggunakan HashiCorp Packer dan spesifikasi build dengan sumber daya dan skrip konfigurasi dari repositori Amazon EKS AMI di AWS. GitHub Anda dapat memperbarui kubelet.service dan mengikuti petunjuk yang ditentukan di sini untuk membuat AMI kustom.

Upgrade Versi CNI Secara Rutin

VPC CNI kompatibel ke belakang. Versi terbaru bekerja dengan semua versi Kubernetes yang didukung Amazon EKS. Selain itu, VPC CNI ditawarkan sebagai add-on EKS (lihat “Menyebarkan Add-On Terkelola VPC CNI” di atas). Sementara add-on EKS mengatur peningkatan add-on, itu tidak akan secara otomatis memutakhirkan add-on seperti CNI karena mereka berjalan di bidang data. Anda bertanggung jawab untuk memutakhirkan add-on VPC CNI setelah upgrade node pekerja yang dikelola dan dikelola sendiri.

Mengoptimalkan Pemanfaatan Alamat IP

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Lingkungan kontainer tumbuh dalam skala dengan cepat, berkat modernisasi aplikasi. Ini berarti semakin banyak node dan pod pekerja yang sedang di-deploy.

Plugin Amazon VPC CNI memberikan setiap pod alamat IP dari CIDR (s) VPC. Pendekatan ini memberikan visibilitas penuh dari alamat Pod dengan alat seperti VPC Flow Logs dan solusi pemantauan lainnya. Bergantung pada jenis beban kerja Anda, hal ini dapat menyebabkan sejumlah besar alamat IP dikonsumsi oleh pod.

Saat mendesain arsitektur jaringan AWS Anda, penting untuk mengoptimalkan konsumsi IP Amazon EKS di VPC dan di tingkat node. Ini akan membantu Anda mengurangi masalah kelelahan IP dan meningkatkan kepadatan pod per node.

Pada bagian ini, kita akan membahas teknik yang dapat membantu Anda mencapai tujuan ini.

Optimalkan konsumsi IP tingkat node

Delegasi awalan adalah fitur Amazon Virtual Private Cloud (Amazon VPC) yang memungkinkan Anda menetapkan awalan IPv4 atau IPv6 ke instans Amazon Elastic Compute Cloud (Amazon EC2). Ini meningkatkan alamat IP per antarmuka jaringan (ENI), yang meningkatkan kepadatan pod per node dan meningkatkan efisiensi komputasi Anda. Delegasi awalan juga didukung dengan Custom Networking.

Untuk informasi lebih lanjut, silakan lihat Delegasi Awalan dengan node Linux dan Delegasi Awalan dengan bagian node Windows.

Mengurangi kelelahan IP

Untuk mencegah cluster Anda mengkonsumsi semua alamat IP yang tersedia, kami sangat menyarankan untuk mengukur ukuran VPCs dan subnet Anda dengan mempertimbangkan pertumbuhan.

Mengadopsi IPv6adalah cara yang bagus untuk menghindari masalah ini sejak awal. Namun, untuk organisasi yang kebutuhan skalabilitasnya melebihi perencanaan awal dan tidak dapat mengadopsi IPv6, meningkatkan desain VPC adalah respons yang disarankan terhadap kelelahan alamat IP. Teknik yang paling umum digunakan di antara pelanggan Amazon EKS adalah menambahkan Secondary non-routable CIDRs ke VPC dan mengonfigurasi VPC CNI untuk menggunakan ruang IP tambahan ini saat mengalokasikan alamat IP ke Pod. Hal ini biasa disebut sebagai Custom Networking.

Kami akan membahas variabel mana dari Amazon VPC CNI yang dapat Anda gunakan untuk mengoptimalkan kumpulan hangat yang IPs ditugaskan ke node Anda. Kami akan menutup bagian ini dengan beberapa pola arsitektur lain yang tidak intrinsik untuk Amazon EKS tetapi dapat membantu mengurangi kelelahan IP.

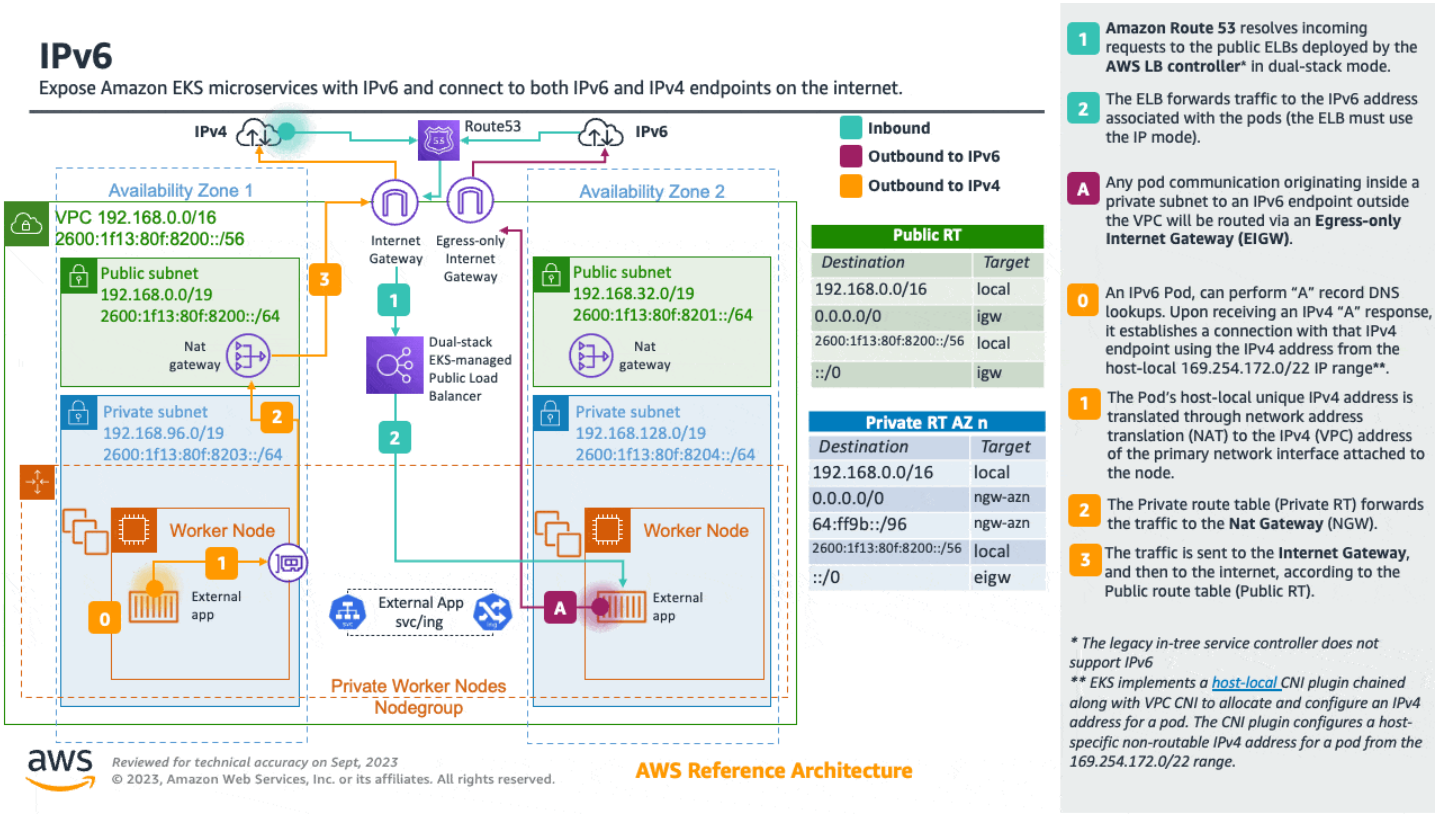
Gunakan IPv6 (disarankan)

Mengadopsi IPv6 adalah cara termudah untuk mengatasi RFC1918 keterbatasan; kami sangat menyarankan Anda mempertimbangkan untuk mengadopsi IPv6 sebagai opsi pertama Anda ketika

memilih arsitektur jaringan. IPv6 menyediakan total ruang alamat IP yang jauh lebih besar, dan administrator cluster dapat fokus pada migrasi dan penskalaan aplikasi tanpa mencurahkan upaya untuk mengatasi batas. IPv4

Cluster Amazon EKS mendukung keduanya IPv4 dan IPv6. Secara default, kluster EKS menggunakan ruang IPv4 alamat. Menentukan ruang alamat IPv6 berbasis pada waktu pembuatan cluster akan memungkinkan penggunaan. IPv6 Dalam klaster IPv6 EKS, pod dan layanan menerima IPv6 alamat sambil mempertahankan kemampuan IPv4 endpoint lama untuk terhubung ke layanan yang berjalan di IPv6 cluster dan sebaliknya. Semua pod-to-pod komunikasi dalam sebuah cluster selalu terjadi IPv6. Dalam VPC (/56), ukuran blok IPv6 CIDR untuk IPv6 subnet ditetapkan pada /64. Ini menyediakan 2 ^ 64 (sekitar 18 triliun) IPv6 alamat yang memungkinkan untuk menskalakan penerapan Anda di EKS.

Untuk informasi terperinci, silakan lihat bagian Menjalankan Kluster IPv6 EKS dan untuk pengalaman langsung, silakan lihat bagian Memahami tentang IPv6 Amazon EKS dari lokakarya Dapatkan langsung dengan. IPv6



Optimalkan konsumsi IP dalam IPv4 cluster

Bagian ini didedikasikan untuk pelanggan yang menjalankan aplikasi lama, tidak and/or siap untuk bermigrasi. IPv6 Meskipun kami mendorong semua organisasi untuk bermigrasi IPv6 sesegera

mungkin, kami menyadari bahwa beberapa mungkin masih perlu melihat pendekatan alternatif untuk meningkatkan beban kerja kontainer mereka. IPv4 Untuk alasan ini, kami juga akan memandu Anda melalui pola arsitektur untuk mengoptimalkan IPv4 (RFC1918) mengatasi konsumsi ruang dengan cluster Amazon EKS.

Rencana untuk Pertumbuhan

Sebagai garis pertahanan pertama terhadap kelelahan IP, kami sangat menyarankan untuk mengukur subnet Anda IPv4 VPCs dan dengan mempertimbangkan pertumbuhan, untuk mencegah cluster Anda mengkonsumsi semua alamat IP yang tersedia. Anda tidak akan dapat membuat Pod atau node baru jika subnet tidak memiliki cukup alamat IP yang tersedia.

Sebelum membangun VPC dan subnet, disarankan untuk bekerja mundur dari skala beban kerja yang diperlukan. Misalnya, ketika cluster dibangun menggunakan eksctl (alat CLI sederhana untuk membuat dan mengelola cluster di EKS) /19 subnet dibuat secara default. Netmask dari /19 cocok untuk sebagian besar jenis beban kerja yang memungkinkan lebih dari 8000 alamat dialokasikan.

> **Important**
>
> Saat Anda mengukur VPCs dan subnet, mungkin ada sejumlah elemen (selain pod dan node) yang dapat menggunakan alamat IP, misalnya Load Balancer, Database RDS, dan layanan in-vpc lainnya.

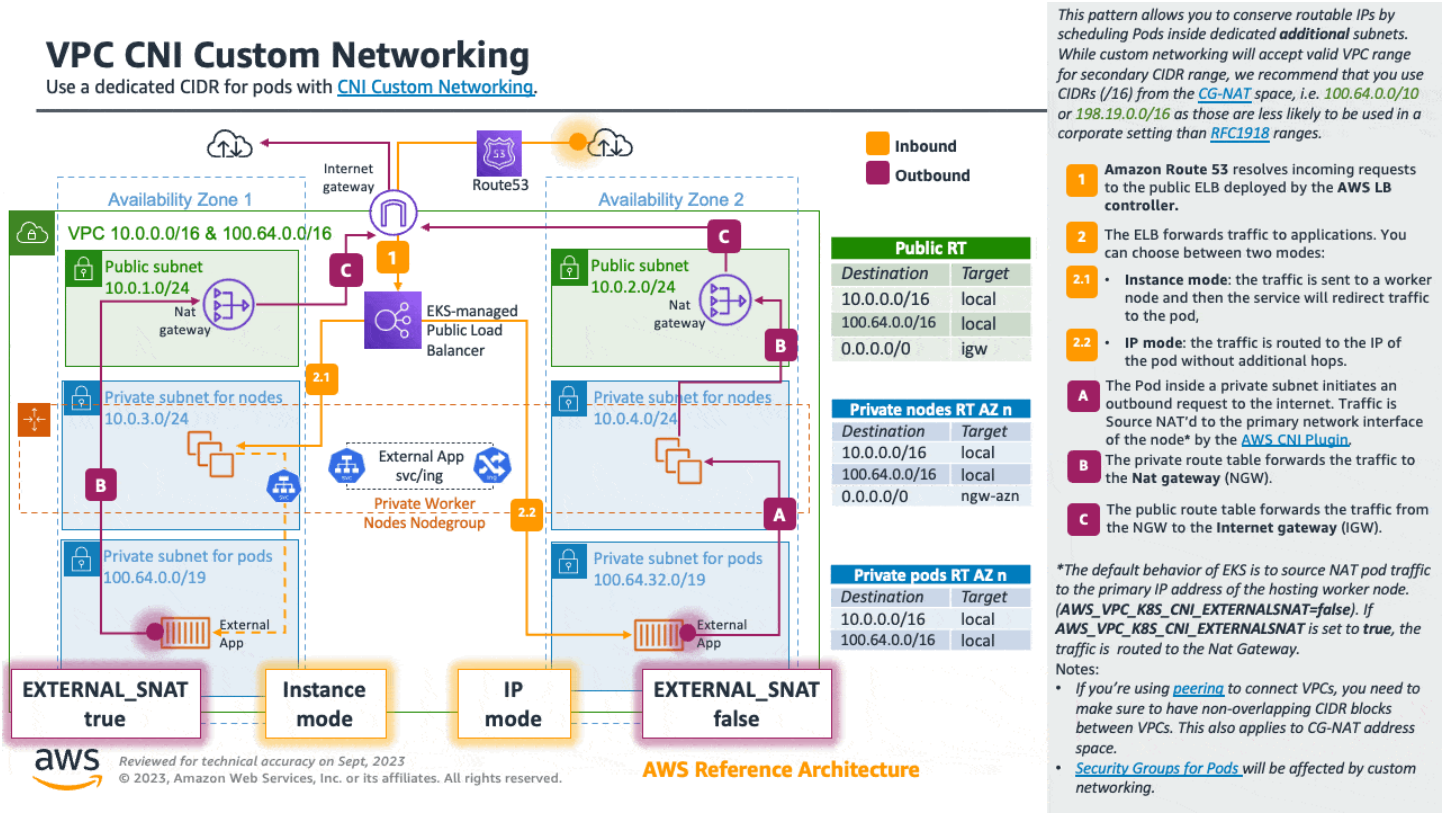
Selain itu, Amazon EKS, dapat membuat hingga 4 antarmuka jaringan elastis (X-ENI) yang diperlukan untuk memungkinkan komunikasi menuju bidang kontrol (info lebih lanjut di sini). Selama upgrade cluster, Amazon EKS membuat X- baru ENIs dan menghapus yang lama ketika upgrade berhasil. Untuk alasan ini kami merekomendasikan netmask minimal /28 (16 alamat IP) untuk subnet yang terkait dengan cluster EKS.

Anda dapat menggunakan contoh spreadsheet EKS Subnet Calculator untuk merencanakan jaringan Anda. Spreadsheet menghitung penggunaan IP berdasarkan beban kerja dan konfigurasi VPC ENI. Penggunaan IP dibandingkan dengan IPv4 subnet untuk menentukan apakah konfigurasi dan ukuran subnet cukup untuk beban kerja Anda. Perlu diingat bahwa, jika subnet di VPC Anda kehabisan alamat IP yang tersedia, kami sarankan untuk membuat subnet baru menggunakan blok CIDR asli VPC. Perhatikan bahwa sekarang Amazon EKS sekarang memungkinkan modifikasi subnet cluster dan grup keamanan.

Jaringan Kustom

Jika Anda akan menghabiskan ruang RFC1918 IP, Anda dapat menggunakan pola Custom Networking untuk menghemat routable IPs dengan menjadwalkan Pod di dalam subnet tambahan khusus. Meskipun jaringan khusus akan menerima rentang VPC yang valid untuk rentang CIDR sekunder, kami menyarankan Anda menggunakan CIDRs (/16) dari ruang CG-NAT, yaitu `100.64.0.0/10` atau `198.19.0.0/16` karena mereka cenderung tidak digunakan dalam pengaturan perusahaan daripada rentang. RFC1918

Untuk informasi rinci silahkan lihat bagian khusus untuk Custom Networking.



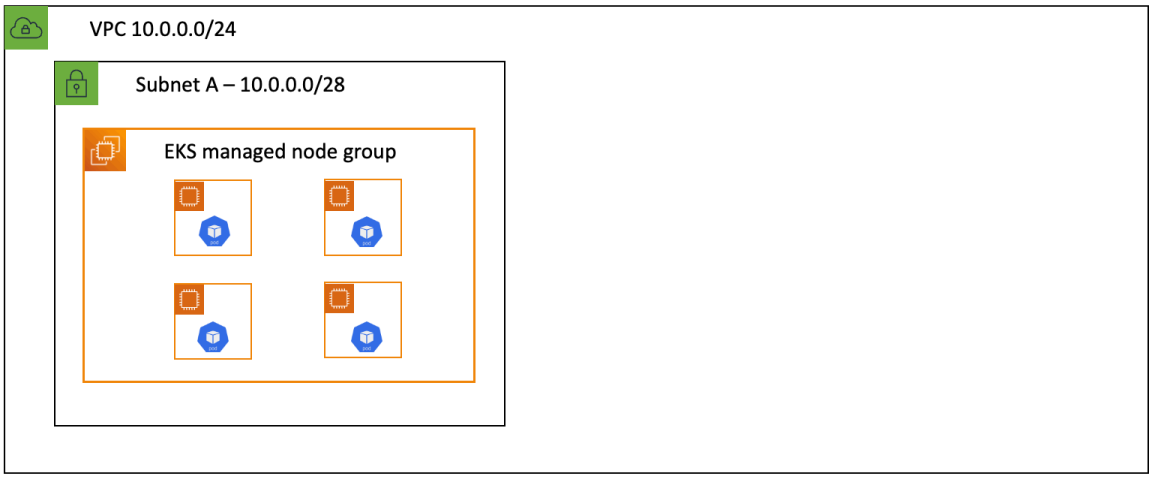
Penemuan Subnet yang Ditingkatkan

Enhanced Subnet Discovery menyediakan alternatif konfigurasi jaringan yang efisien untuk kelelahan IP, dengan menandai subnet baru sehingga dapat ditemukan oleh Amazon VPC CNI. Dengan Enhanced Subnet Discovery, beban kerja saat ini dapat terus berjalan pada subnet yang sama dan Amazon Elastic Kubernetes Service (Amazon EKS) sekarang dapat menjadwalkan pod tambahan pada “subnet yang dapat digunakan” baru.

Jika subnet klaster Anda saat ini kehabisan alamat IP, Anda cukup menambahkan subnet tambahan ke cluster Amazon EKS Anda sebagai berikut:

1. Kaitkan blok CIDR baru ke VPC Anda.
2. Buat subnet baru di blok CIDR baru dan beri tag dengan “kubernetes. io/role/cni” = “1".
3. Aktifkan konfigurasi ENABLE_SUBNET_DISCOVERY dari add-on Amazon VPC CNI menjadi “true” (default sejak versi 1.18.0).

Setelah Enhanced Subnet Discovery diaktifkan pada kluster VPC dan Amazon EKS Anda, Elastic Network Interfaces ENIs () baru akan dilampirkan ke node Amazon EKS Anda seperti yang dijelaskan dalam diagram berikut:



Untuk informasi selengkapnya, lihat Amazon VPC CNI memperkenalkan Enhanced Subnet Discovery di blog AWS container.

Optimalkan kolam IPs hangat

Dengan konfigurasi default, VPC CNI menyimpan seluruh ENI (dan terkait IPs) di kolam hangat. Ini mungkin menghabiskan sejumlah besar IPs, terutama pada jenis instance yang lebih besar.

Jika subnet cluster Anda memiliki jumlah alamat IP terbatas yang tersedia, teliti variabel lingkungan konfigurasi VPC CNI ini:

- `WARM_IP_TARGET`

- `MINIMUM_IP_TARGET`
- `WARM_ENI_TARGET`

Anda dapat mengonfigurasi nilai `MINIMUM_IP_TARGET` agar sesuai dengan jumlah Pod yang Anda harapkan untuk dijalankan di node Anda. Melakukannya akan memastikan bahwa saat Pod dibuat, dan CNI dapat menetapkan alamat IP dari kolam hangat tanpa memanggil API EC2.

Harap diperhatikan bahwa menyetel nilai `WARM_IP_TARGET` terlalu rendah, akan menyebabkan panggilan tambahan ke EC2 API, dan itu dapat menyebabkan pembatasan permintaan. Untuk cluster besar gunakan bersama dengan `MINIMUM_IP_TARGET` untuk menghindari pembatasan permintaan.

Untuk mengonfigurasi opsi ini, Anda dapat mengunduh `aws-k8s-cni.yaml` manifes dan mengatur variabel lingkungan. Pada saat penulisan, rilis terbaru terletak di sini. Periksa versi nilai konfigurasi yang cocok dengan versi VPC CNI yang diinstal.

> **Warning**
>
> Pengaturan ini akan diatur ulang ke default saat Anda memperbarui CNI. Silakan ambil cadangan CNI, sebelum Anda memperbaruinya. Tinjau pengaturan konfigurasi untuk menentukan apakah Anda perlu menerapkannya kembali setelah pembaruan berhasil.

Anda dapat menyesuaikan parameter CNI dengan cepat tanpa downtime untuk aplikasi yang ada, tetapi Anda harus memilih nilai yang akan mendukung kebutuhan skalabilitas Anda. Misalnya, jika Anda bekerja dengan beban kerja batch, sebaiknya update default `WARM_ENI_TARGET` agar sesuai dengan kebutuhan skala Pod. Pengaturan `WARM_ENI_TARGET` ke nilai tinggi selalu mempertahankan kumpulan IP hangat yang diperlukan untuk menjalankan beban kerja batch besar dan karenanya menghindari penundaan pemrosesan data.

> **Warning**
>
> Meningkatkan desain VPC Anda adalah respons yang disarankan terhadap kelelahan alamat IP. Pertimbangkan solusi seperti IPv6 dan Sekunder CIDRs. Menyesuaikan nilai-nilai ini untuk meminimalkan jumlah Warm IPs harus menjadi solusi sementara setelah opsi lain dikecualikan. Kesalahan konfigurasi nilai-nilai ini dapat mengganggu operasi cluster. Sebelum membuat perubahan apa pun pada sistem produksi, pastikan untuk meninjau pertimbangan di halaman ini.

Pantau Inventaris Alamat IP

Selain solusi yang dijelaskan di atas, penting juga untuk memiliki visibilitas melalui pemanfaatan IP. Anda dapat memantau inventaris alamat IP subnet menggunakan CNI Metrics Helper. Beberapa metrik yang tersedia adalah:

- Jumlah maksimum ENIs cluster dapat mendukung
- jumlah yang ENIs sudah dialokasikan
- jumlah alamat IP yang saat ini ditetapkan ke Pod
- Jumlah total dan maksimum alamat IP yang tersedia

Anda juga dapat mengatur CloudWatch alarm untuk mendapatkan pemberitahuan jika subnet kehabisan alamat IP.

> ⚠ Warning
>
> Pastikan `DISABLE_METRICS` variabel untuk VPC CNI disetel ke false.

Pertimbangan lebih lanjut

Ada pola arsitektur lain yang tidak intrinsik untuk Amazon EKS yang dapat membantu mengatasi kelelahan IP. Misalnya, Anda dapat mengoptimalkan komunikasi di seluruh VPCs atau berbagi VPC di beberapa akun untuk membatasi alokasi IPv4 alamat.

Pelajari lebih lanjut tentang pola-pola ini di sini:

- Merancang jaringan VPC Amazon hyperscale,
- Bangun konektivitas multi-VPC multi-akun yang aman dengan Amazon VPC Lattice.

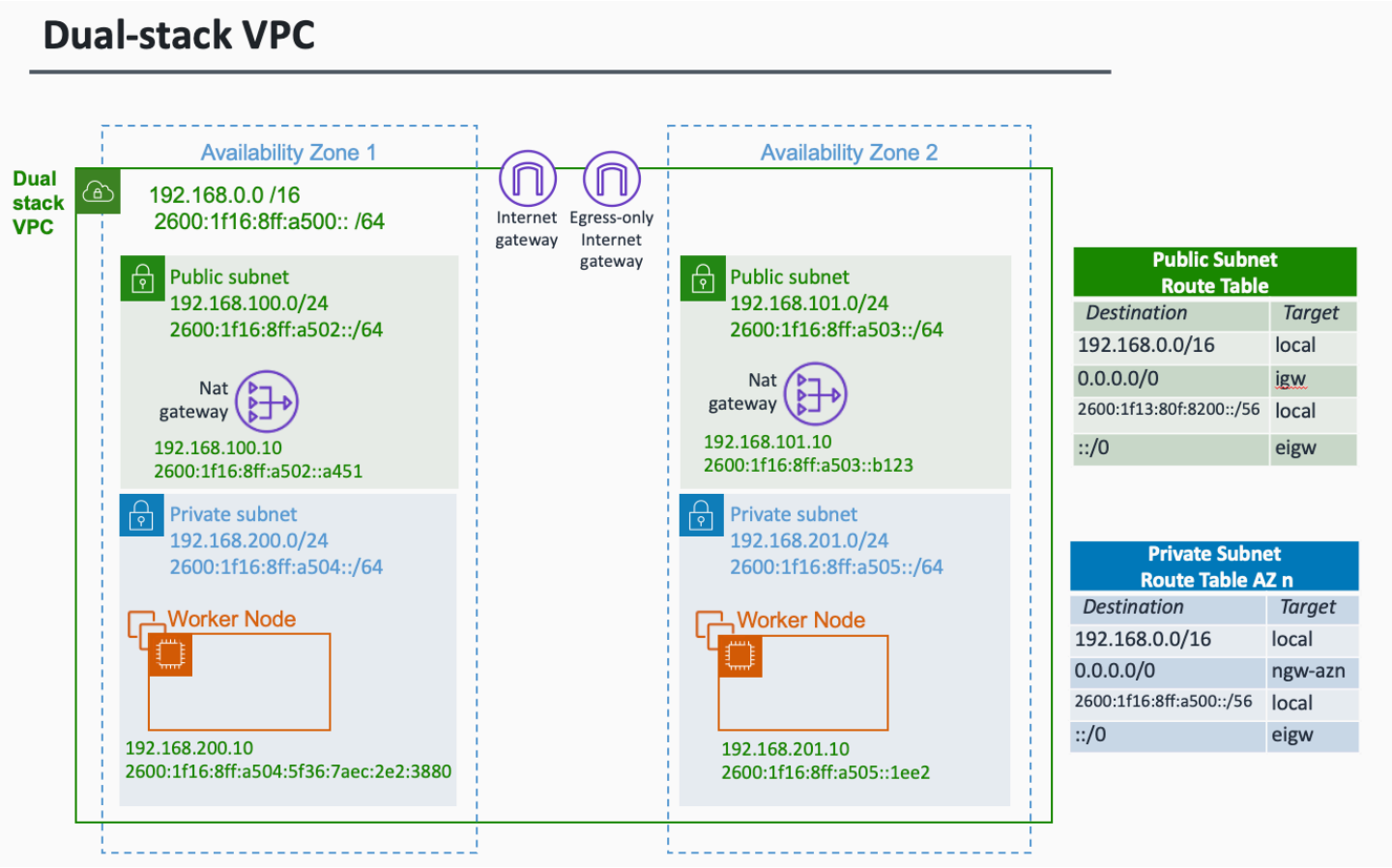
Menjalankan Kluster IPv6 EKS

EKS dalam IPv6 mode memecahkan tantangan IPv4 kelelahan yang sering dimanifestasikan dalam kelompok EKS skala besar. Dukungan EKS untuk IPv6 difokuskan pada penyelesaian masalah IPv4 kelelahan, yang berasal dari ukuran ruang alamat yang terbatas. IPv4 Ini adalah kekhawatiran signifikan yang diangkat oleh sejumlah pelanggan kami dan berbeda dari fitur IPv4IPv6 Kubernetes/ dual-stack. EKS/ juga IPv6 akan memberikan fleksibilitas untuk menghubungkan batas-batas jaringan menggunakan IPv6 CIDRs sehingga meminimalkan peluang untuk menderita CIDR tumpang tindih,

sehingga memecahkan masalah 2-Fold (In-Cluster, Cross-Cluster). Saat menerapkan kluster EKS dalam IPv6 mode (--ip-family ipv6), tindakannya tidak dapat dibalik. Dengan kata sederhana IPv6 dukungan EKS diaktifkan untuk seluruh masa hidup cluster Anda.

Dalam klaster IPv6 EKS, Pod dan Layanan akan menerima IPv6 alamat sambil mempertahankan kompatibilitas dengan IPv4 Endpoint lama. Ini termasuk kemampuan IPv4 endpoint eksternal untuk mengakses layanan in-cluster, dan Pod untuk mengakses endpoint eksternal IPv4 .

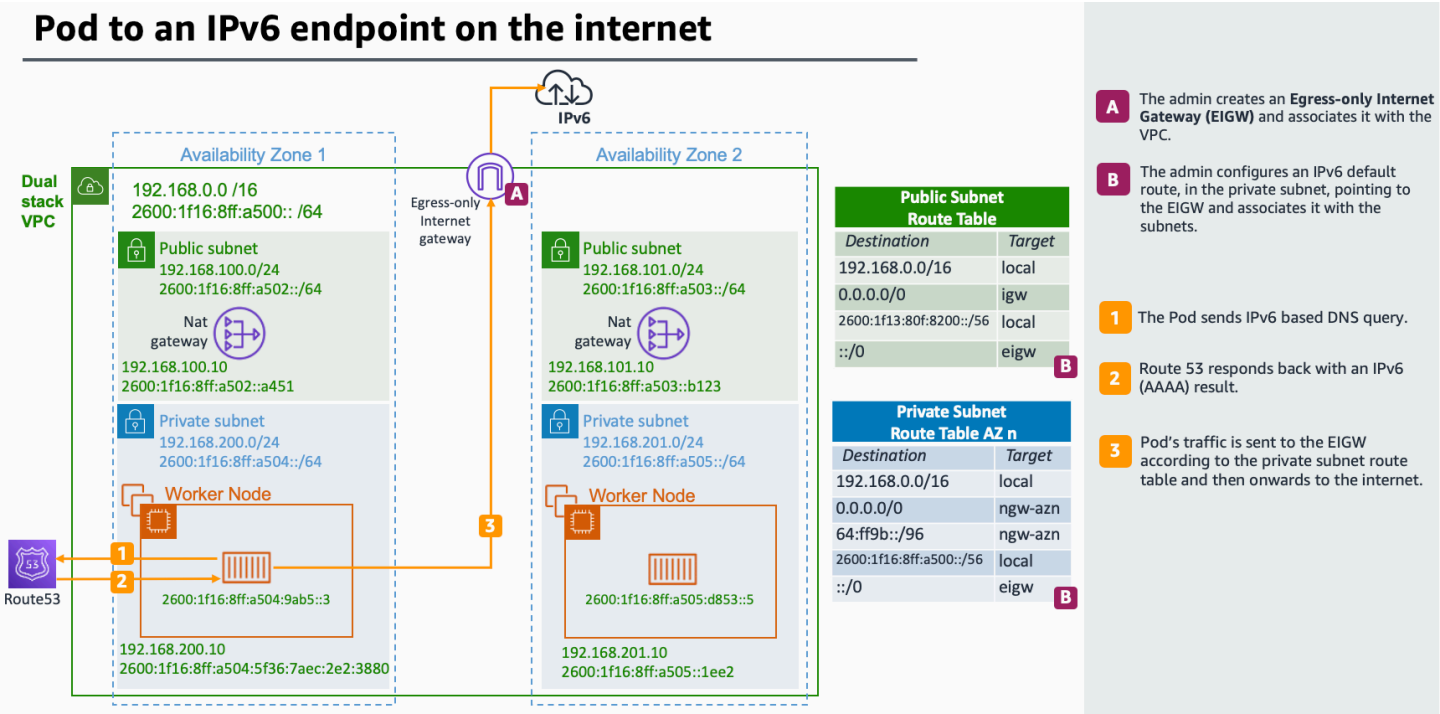
IPv6 Dukungan Amazon EKS memanfaatkan kemampuan IPv6 VPC asli. Setiap VPC dialokasikan dengan awalan IPv4 alamat (ukuran blok CIDR dapat dari /16 hingga /28) dan awalan alamat/56 unik (tetap) dari dalam GUA Amazon (IPv6 Alamat Unicast Global); Anda dapat menetapkan awalan alamat/64 ke setiap subnet di VPC Anda. IPv4 fitur, seperti Tabel Rute, Daftar Kontrol Akses Jaringan, Peering, dan resolusi DNS, bekerja dengan cara yang sama di IPv6 VPC yang diaktifkan. VPC kemudian disebut sebagai VPC dual-stack, mengikuti subnet dual-stack, diagram berikut menggambarkan pola pondasi VPC yang mendukung cluster berbasis IPV4 IPv6 EKS/: IPv6



Di IPv6 dunia, setiap alamat dapat dirutekan melalui internet. Secara default, VPC mengalokasikan IPv6 CIDR dari jangkauan GUA publik. Namun sejak Agustus 2024 Anda juga dapat menggunakan

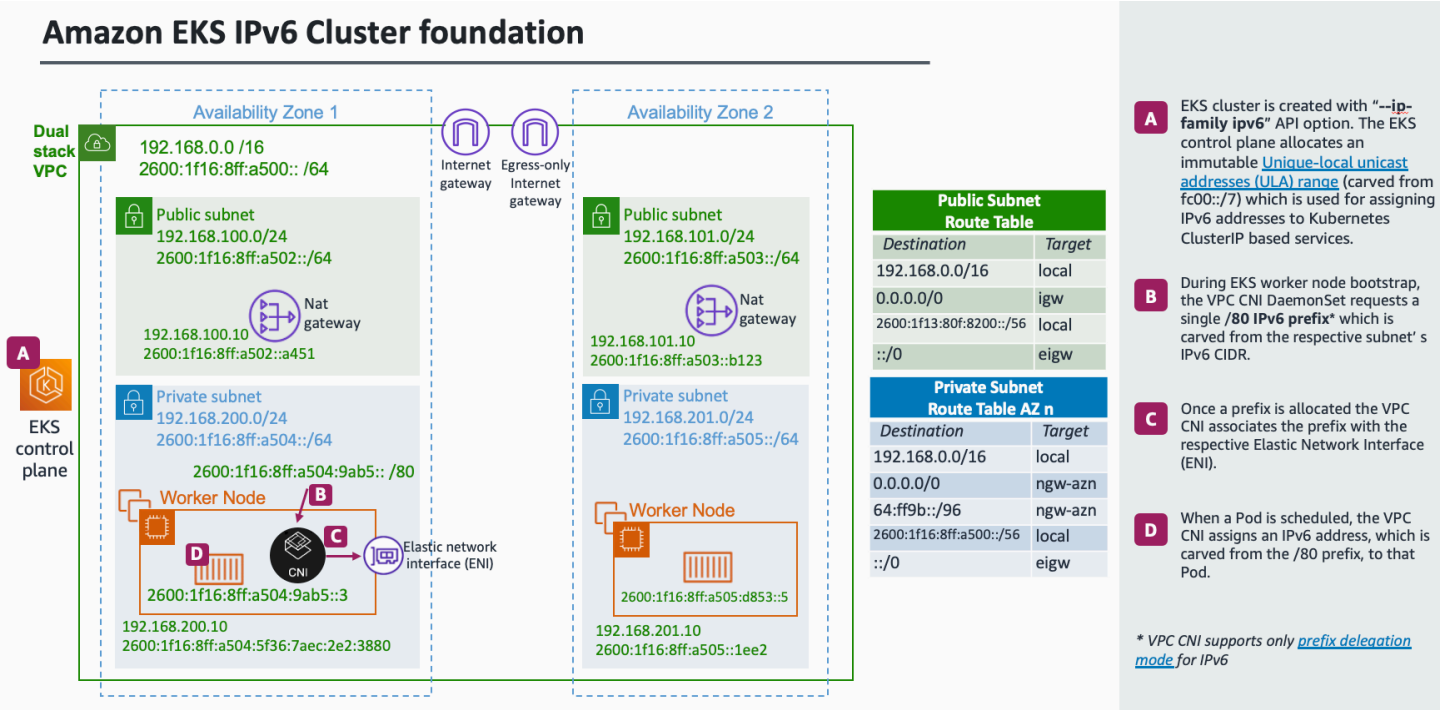
IPv6 pengalamatan pribadi untuk VPCs dan subnet dengan Amazon VPC IP Address Manager (IPAM). Silakan lihat posting blog AWS Networking dan dokumentasi VPC ini untuk informasi lebih lanjut.

Diagram berikut menggambarkan aliran keluar IPv6 Internet Pod di dalam EXS/cluster: IPv6



Praktik terbaik untuk menerapkan IPv6 subnet dapat ditemukan di panduan pengguna VPC.

Dalam klaster IPv6 EKS, node dan Pod menerima IPv6 alamat publik. EKS memberikan IPv6 alamat ke layanan berdasarkan Alamat IPv6 Unicast Lokal Unik (ULA). CIDR Layanan ULA untuk IPv6 cluster secara otomatis ditetapkan selama tahap pembuatan cluster dan tidak dapat ditentukan, tidak seperti IPv4. Diagram berikut menggambarkan pola dasar rencana data bidang kontrol cluster IPv6 berbasis EKS/:



Ikhtisar

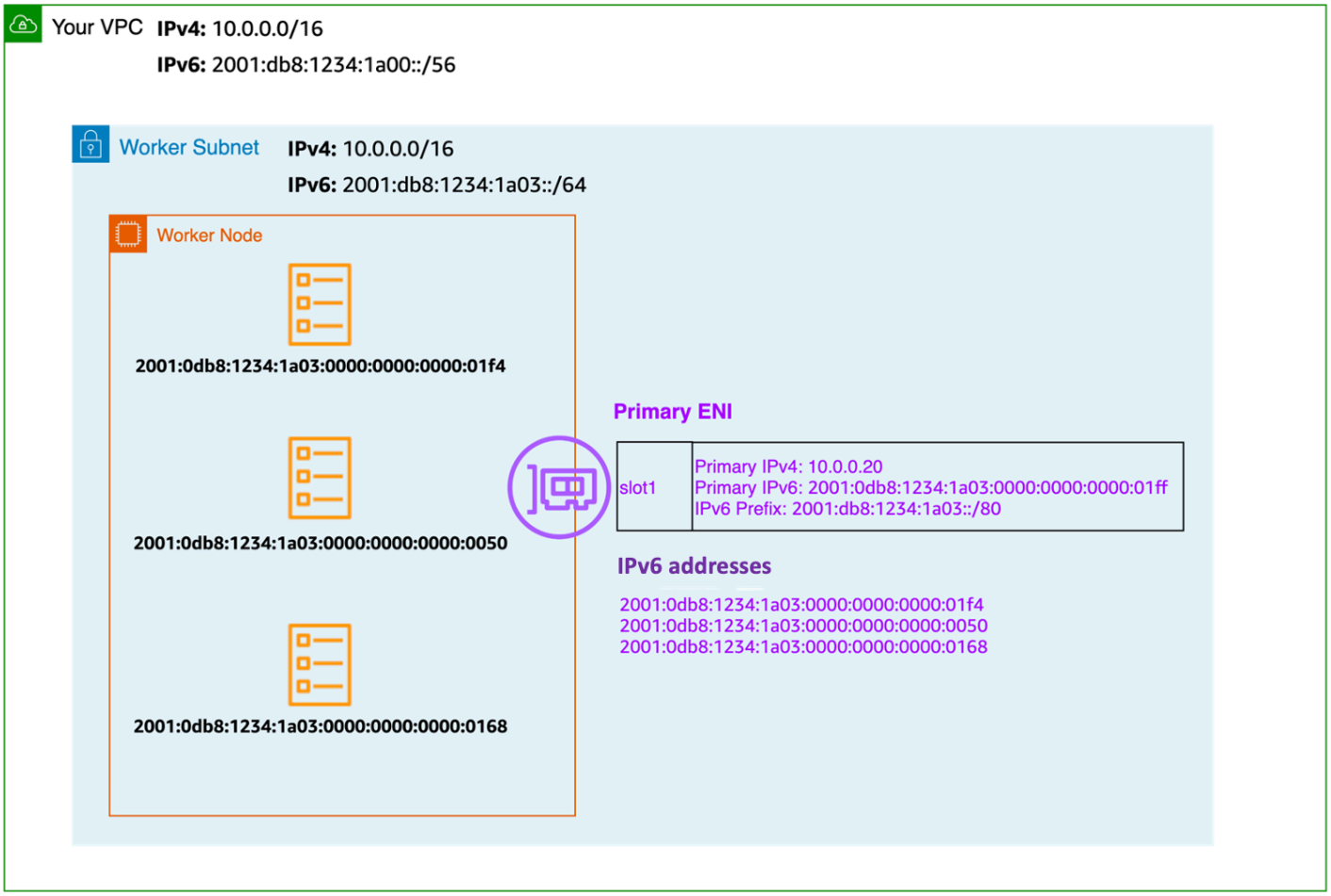
EKS/ hanya IPv6 didukung dalam mode awalan (mode penetapan IP ENI Plug-in VPC-CNI Plug-in). Pelajari lebih lanjut tentang Mode Awalan.

Penetapan awalan hanya berfungsi pada instans EC2 berbasis Nitro, oleh karena itu EKS/IPv6 hanya didukung ketika bidang data cluster menggunakan instans berbasis EC2 Nitro.

Dengan kata sederhana IPv6 awalan /80 (Per worker-node) akan menghasilkan ~10^14 IPv6 alamat, faktor pembatasnya tidak lagi adalah kepadatan Pod (Dari segi sumber daya). IPs

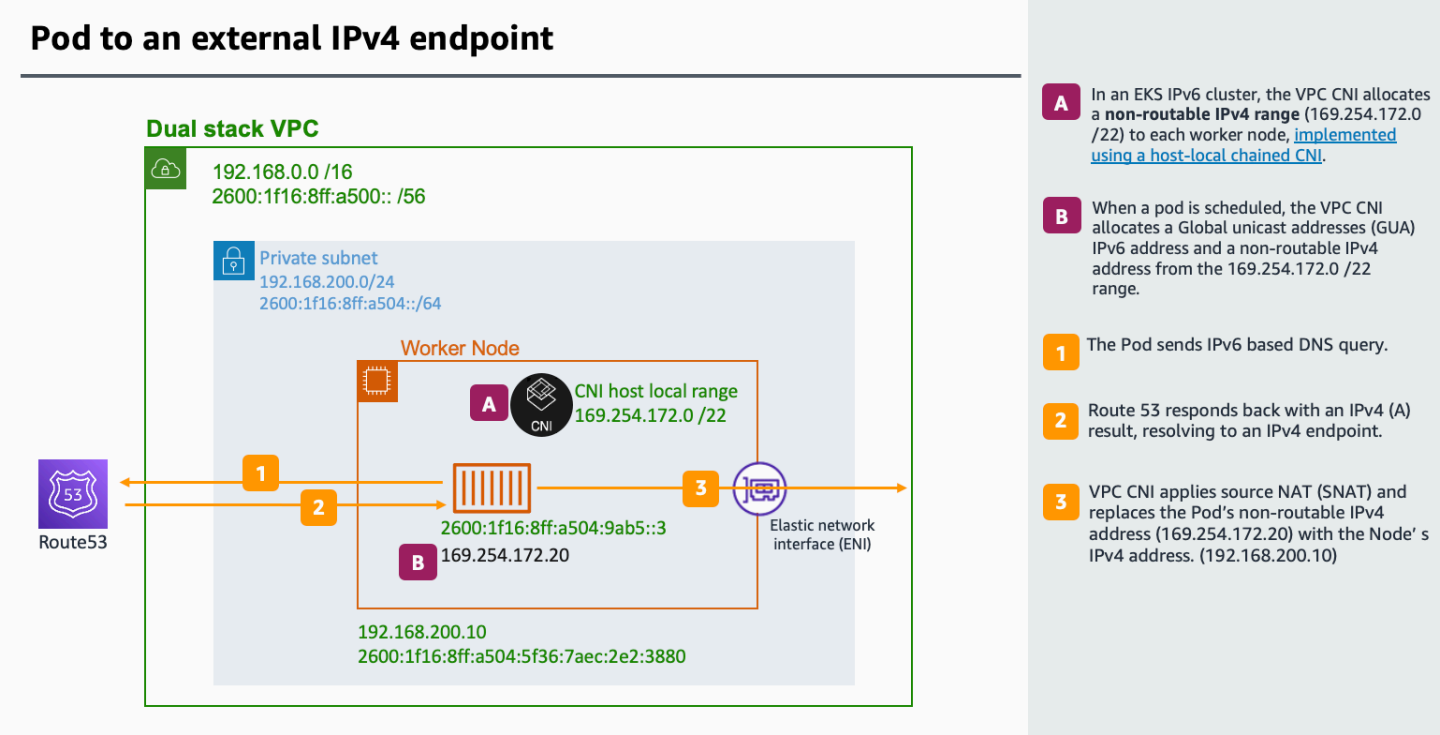
IPv6 penetapan awalan hanya terjadi pada waktu bootstrap worker-node EKS. Perilaku ini diketahui dapat mengurangi skenario di mana EX/ IPv4 cluster Pod churn tinggi sering tertunda dalam penjadwalan Pod karena panggilan API yang dibatasi yang dihasilkan oleh plug-in VPC CNI (ipamd) yang bertujuan untuk mengalokasikan alamat Private secara tepat waktu. IPv4 Hal ini juga dikenal untuk membuat kenop canggih plug-in VPC-CNI menyetel WARM_IP/ENI, MINIMUM_IP tidak perlu.

Diagram berikut memperbesar ke IPv6 Worker-node Elastic Network Interface (ENI):



Setiap node pekerja EKS ditugaskan dengan IPv4 dan IPv6 alamat, bersama dengan entri DNS yang sesuai. Untuk node pekerja tertentu, hanya satu IPv4 alamat dari subnet dual-stack yang digunakan. Dukungan EKS untuk IPv6 memungkinkan Anda berkomunikasi dengan IPv4 titik akhir (AWS, on-premise, internet) melalui model egress-only yang sangat berpendirian tinggi. IPv4 EKS mengimplementasikan plugin CNI host-lokal, sekunder dari plugin VPC CNI, yang mengalokasikan dan mengonfigurasi alamat untuk sebuah Pod. IPv4 Plugin CNI mengonfigurasi IPv4 alamat non-routable khusus host untuk sebuah Pod dari rentang 169.254.172.0/22. IPv4 Alamat yang ditetapkan ke Pod adalah unik untuk worker-node dan tidak diiklankan di luar worker-node. 169.254.172.0/22 menyediakan hingga 1024 alamat unik yang dapat mendukung tipe instance besar. IPv4

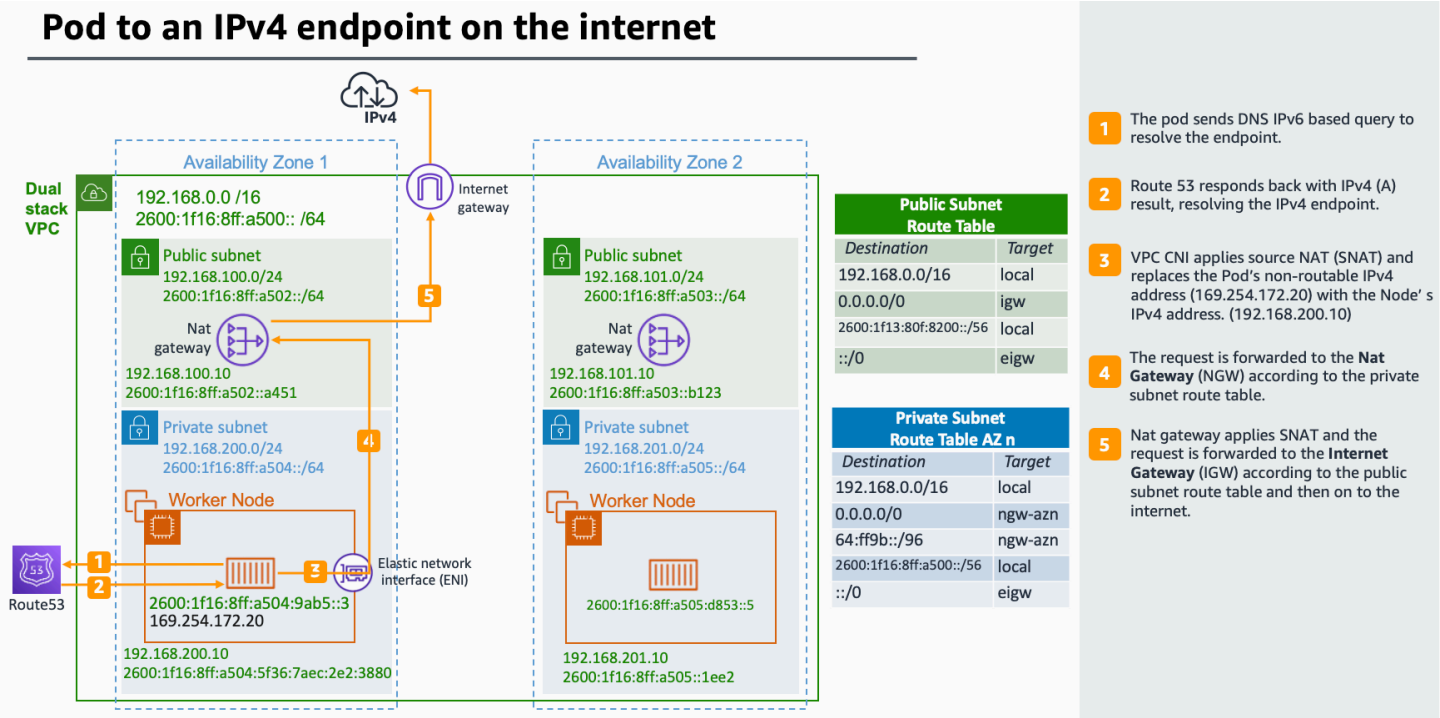
Diagram berikut menggambarkan aliran IPv6 Pod yang terhubung ke IPv4 titik akhir di luar batas cluster (non-internet):



Pada diagram di atas, Pod akan melakukan pencarian DNS untuk titik akhir dan, setelah menerima respons IPv4 “A”, alamat IPv4 unik khusus node Pod diterjemahkan melalui terjemahan alamat jaringan sumber (SNAT) ke alamat Private IPv4 (VPC) dari antarmuka jaringan utama yang dilampirkan ke EC2 Worker-Node.

> **Note**
>
> Pola di atas mengharuskan DNS64 dinonaktifkan pada subnet tempat EKS/ IPv6 Pod berjalan. Ketika DNS64 diaktifkan, resolver DNS mengembalikan alamat yang disintesis untuk titik akhir IPv4 -only bersama dengan IPv6 alamat. IPv4 Akibatnya, rute lalu lintas melalui NAT64 fungsionalitas NAT Gateway (jika disertakan dalam arsitektur) alih-alih tetap berada di dalam VPC seperti yang ditunjukkan pada pola di atas. Hal ini dapat menyebabkan penggunaan NAT Gateway yang tidak terduga dan biaya terkait.

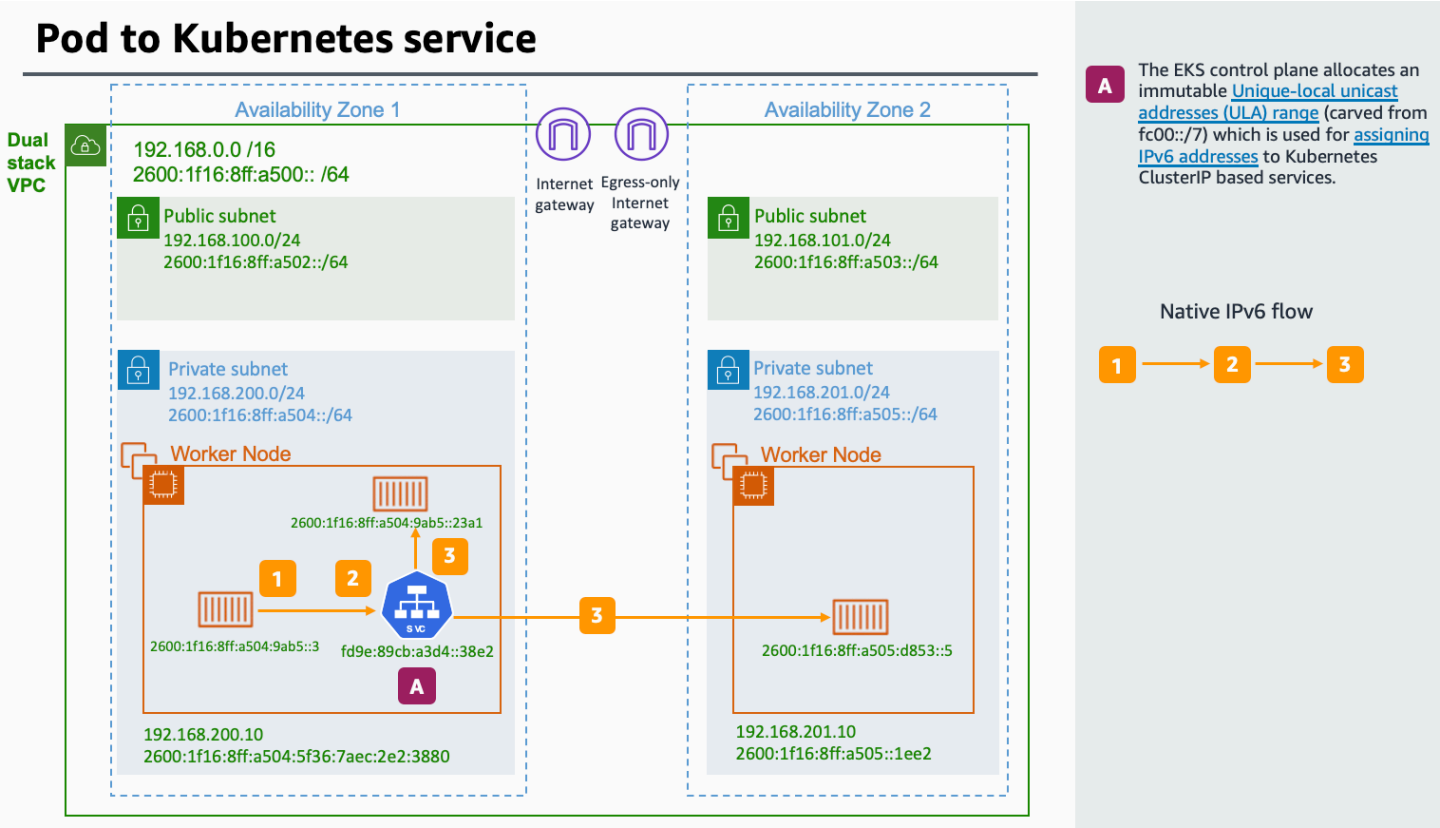
EKS/ IPv6 Pod juga perlu terhubung ke IPv4 titik akhir melalui internet menggunakan IPv4 Alamat publik, untuk mencapai bahwa aliran serupa ada. Diagram berikut menggambarkan aliran IPv6 Pod yang terhubung ke IPv4 titik akhir di luar batas cluster (internet routable):



Pada diagram di atas, Pod akan melakukan pencarian DNS untuk titik akhir dan, setelah menerima respons IPv4 “A”, alamat IPv4 unik khusus node Pod diterjemahkan melalui terjemahan alamat jaringan sumber (SNAT) ke alamat Private IPv4 (VPC) dari antarmuka jaringan utama yang dilampirkan ke EC2 Worker-Node. Alamat IPv4 Pod (Sumber IPv4: EC2 Primary IP) kemudian dirutekan ke IPv4 NAT Gateway di mana IP Primer EC2 diterjemahkan (SNAT) ke Internet IPv4 routable Public IP Address (NAT Gateway Assigned Public IP) yang valid.

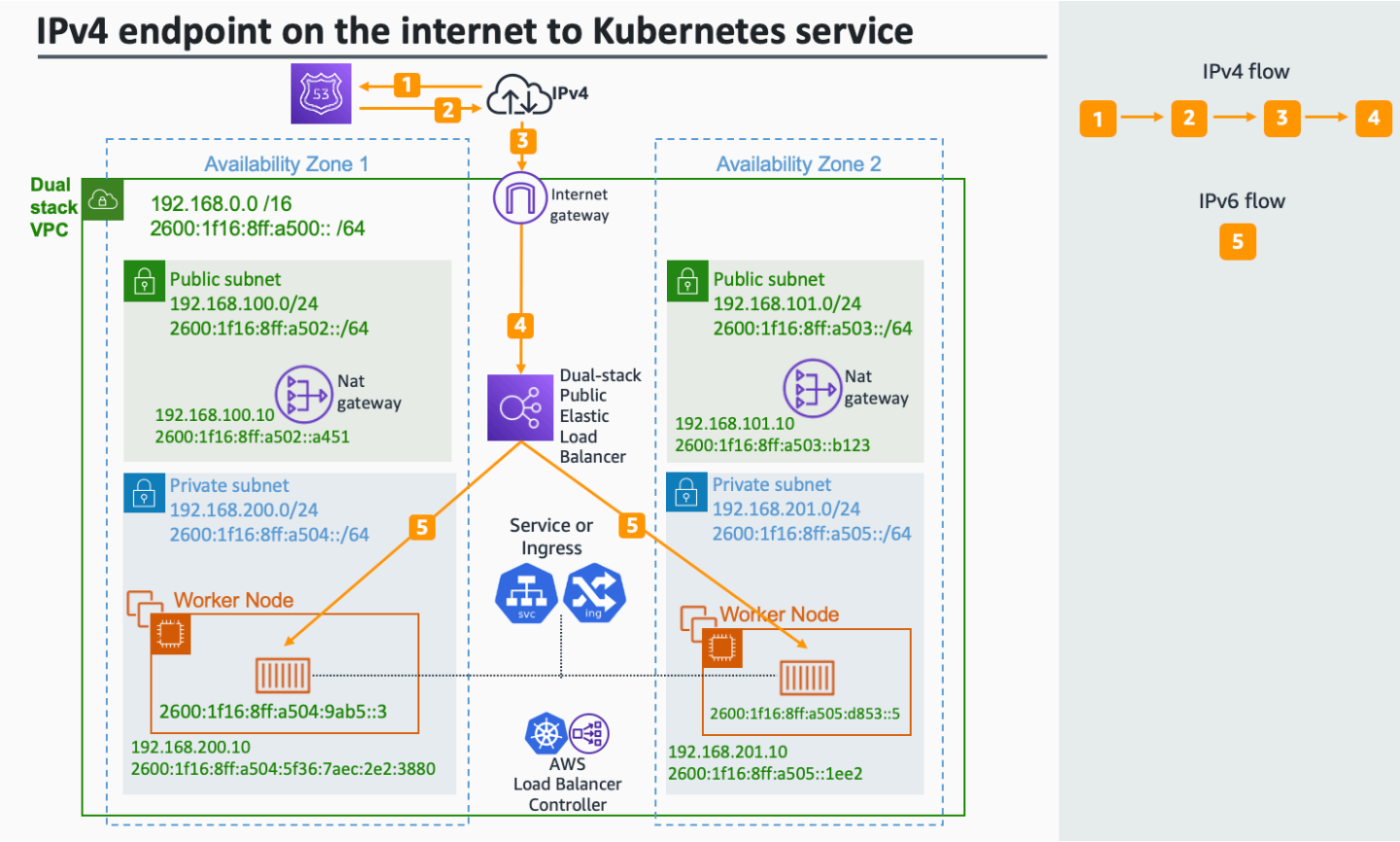
Setiap Pod-to-Pod komunikasi di seluruh node selalu menggunakan IPv6 alamat. VPC CNI mengonfigurasi iptables untuk ditangani IPv6 sambil memblokir koneksi apa pun. IPv4

Layanan Kubernetes hanya akan menerima alamat IPv6 (ClusterIP) dari Unique Local Unicast Address (ULA). IPv6 CIDR Layanan ULA untuk IPv6 cluster secara otomatis ditetapkan selama tahap pembuatan klaster EKS dan tidak dapat dimodifikasi. Diagram berikut menggambarkan alur Pod ke Kubernetes Service:



Layanan diekspos ke internet menggunakan penyeimbang beban AWS. Penyeimbang beban menerima publik IPv4 dan IPv6 alamat, alias penyeimbang beban tumpukan ganda. Untuk IPv4 klien yang mengakses layanan IPv6 cluster kubernetes, load balancer melakukan terjemahan. IPv4 IPv6

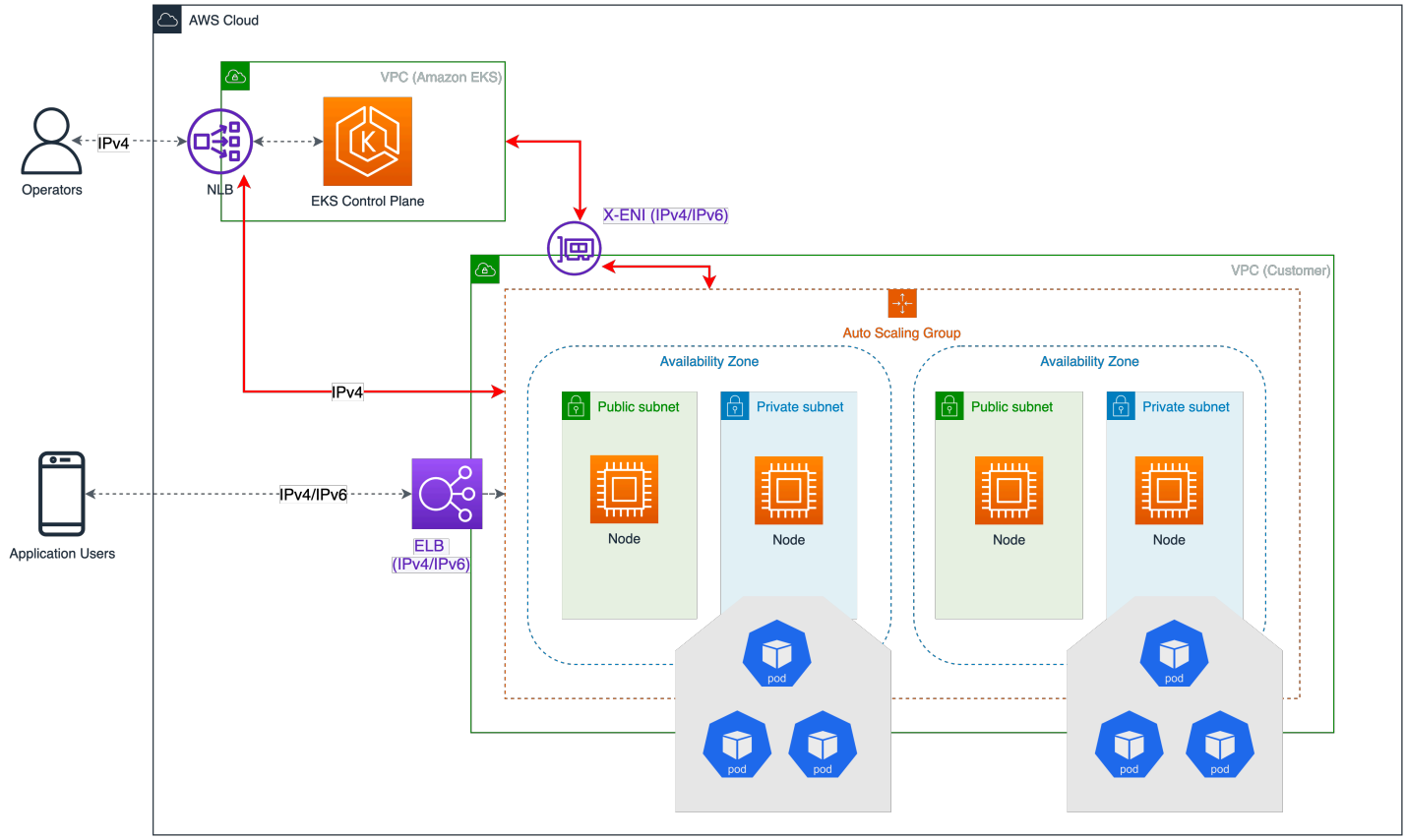
Amazon EKS merekomendasikan untuk menjalankan node pekerja dan Pod dalam subnet pribadi. Anda dapat membuat penyeimbang beban publik di subnet publik yang memuat lalu lintas keseimbangan ke Pod yang berjalan pada node yang berada di subnet pribadi. Diagram berikut menggambarkan IPv4 pengguna internet yang mengakses layanan berbasis IPv6 EKS/Ingress:



> **Note**
>
> Pola di atas memerlukan penerapan versi terbaru dari pengontrol penyeimbang beban AWS

EKS Kontrol Pesawat Data Pesawat Komunikasi

EKS akan menyediakan Cross-Account ENIs (X-ENIs) dalam mode tumpukan ganda (IPv4/IPv6). Komponen node Kubernetes seperti kubelet dan kube-proxy dikonfigurasi untuk mendukung dual stack. Kubelet dan kube-proxy berjalan dalam mode HostNetwork dan mengikat keduanya IPv4 dan IPv6 alamat yang dilampirkan ke antarmuka jaringan utama dari sebuah node. Api-server Kubernetes berkomunikasi dengan Pod dan komponen node melalui X- berbasis. ENIs IPv6 Pod berkomunikasi dengan api-server melalui X-ENIs, dan komunikasi Pod ke api-server selalu menggunakan mode. IPv6



Rekomendasi

Jadwal Berdasarkan Sumber Daya Komputasi

Sebuah IPv6 awalan tunggal cukup untuk menjalankan banyak Pod pada satu node. Ini juga secara efektif menghilangkan batasan ENI dan IP pada jumlah maksimum Pod pada sebuah node. Meskipun IPv6 menghapus ketergantungan langsung pada Max-pod, saat menggunakan lampiran awalan dengan tipe instans yang lebih kecil seperti m5.large, Anda cenderung menghabiskan CPU dan sumber daya memori instans jauh sebelum Anda menghabiskan alamat IP-nya. Anda harus menetapkan nilai Pod maksimum yang direkomendasikan EKS dengan tangan jika Anda menggunakan grup node yang dikelola sendiri atau grup node terkelola dengan ID AMI kustom.

Anda dapat menggunakan rumus berikut untuk menentukan jumlah maksimum Pod yang dapat Anda gunakan pada sebuah node untuk klaster 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)
```

Grup node terkelola secara otomatis menghitung jumlah maksimum Pod untuk Anda. Hindari mengubah nilai rekomendasi EKS untuk jumlah maksimum Pod untuk menghindari kegagalan penjadwalan Pod karena keterbatasan sumber daya.

Mengevaluasi Tujuan Jaringan Kustom yang Ada

Jika jaringan khusus saat ini diaktifkan, Amazon EKS merekomendasikan untuk mengevaluasi kembali kebutuhan Anda. IPv6 Jika Anda memilih untuk menggunakan jaringan khusus untuk mengatasi masalah IPv4 kelelahan, itu tidak lagi diperlukan. IPv6 Jika Anda menggunakan jaringan khusus untuk memenuhi persyaratan keamanan, seperti jaringan terpisah untuk node dan Pod, Anda disarankan untuk mengirimkan permintaan peta jalan EKS.

Pod Fargate di EKS/ Cluster IPv6

EKS mendukung IPv6 Pod yang berjalan di Fargate. Pod yang berjalan di Fargate akan menggunakan dan alamat IPv6 VPC Routable Private yang IPv4 diukir dari rentang CIDR VPC (). IPv4 IPv6 Dengan kata sederhana kepadatan luas cluster EKS/Fargate Pod Anda akan terbatas pada yang tersedia IPv4 dan IPv6 alamat. Disarankan untuk mengukur dual-stack Anda untuk pertumbuhan subnets/VPC CIDRs masa depan. Anda tidak akan dapat menjadwalkan Pod Fargate baru jika subnet yang mendasarinya tidak berisi alamat yang tersedia, terlepas dari IPv4 alamat yang tersedia. IPv6

Menerapkan AWS Load Balancer Controller (LBC)

Pengontrol layanan Kubernetes in-tree upstream tidak mendukung. IPv6 Sebaiknya gunakan add-on AWS Load Balancer Controller versi terbaru. LBC hanya akan menerapkan NLB dual-stack atau ALB dual-stack setelah menggunakan definisi kubernetes terkait yang dianotasi dengan: dan service/ingress `"alb.ingress.kubernetes.io/ip-address-type: dualstack"` `"alb.ingress.kubernetes.io/target-type: ip"`

Jaringan Kustom

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Secara default, Amazon VPC CNI akan menetapkan Pod alamat IP yang dipilih dari subnet utama. Subnet primer adalah subnet CIDR yang melekat pada ENI primer, biasanya subnet dari node/host.

Jika subnet CIDR terlalu kecil, CNI mungkin tidak dapat memperoleh cukup alamat IP sekunder untuk ditetapkan ke Pod Anda. Ini adalah tantangan umum untuk IPv4 kluster EKS.

Jaringan khusus adalah salah satu solusi untuk masalah ini.

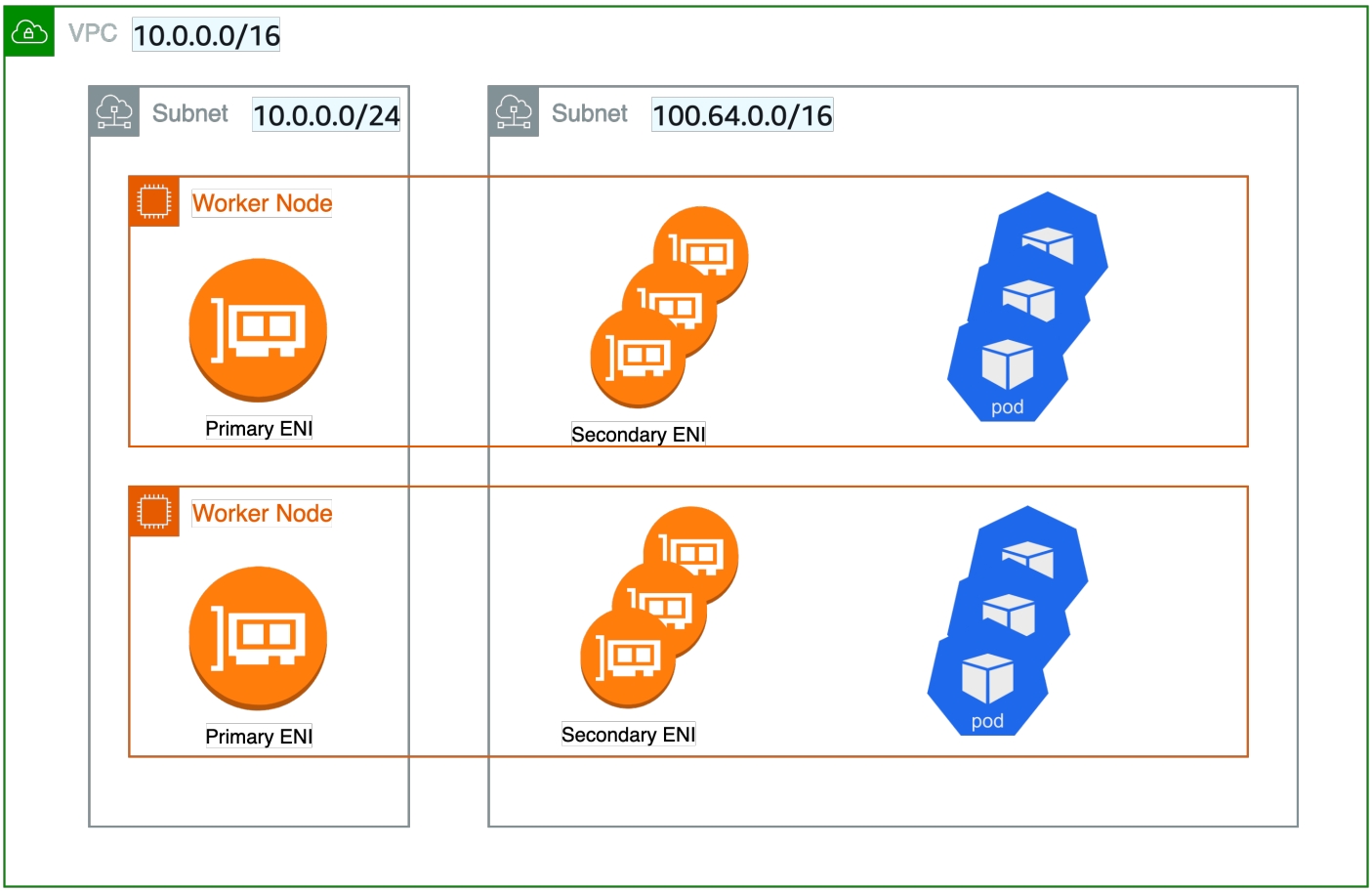
Jaringan khusus mengatasi masalah kelelahan IP dengan menetapkan node dan Pod dari ruang alamat VPC IPs sekunder (CIDR). Dukungan jaringan khusus mendukung sumber daya ENIConfig khusus. ENIConfig Termasuk rentang CIDR subnet alternatif (diukir dari CIDR VPC sekunder), bersama dengan grup keamanan yang akan dimiliki oleh Pod. Saat jaringan khusus diaktifkan, CNI VPC membuat sekunder ENIs di subnet yang ditentukan di bawah. ENIConfig CNI memberikan Pod alamat IP dari rentang CIDR yang ditentukan dalam CRD. ENIConfig

Karena ENI primer tidak digunakan oleh jaringan kustom, jumlah maksimum Pod yang dapat Anda jalankan pada sebuah node lebih rendah. Pod jaringan host terus menggunakan alamat IP yang ditetapkan ke ENI primer. Selain itu, ENI primer digunakan untuk menangani terjemahan jaringan sumber dan merutekan lalu lintas Pod di luar node.

Contoh Konfigurasi

Meskipun jaringan khusus akan menerima rentang VPC yang valid untuk rentang CIDR sekunder, kami menyarankan Anda menggunakan CIDRs (/16) dari ruang CG-NAT, yaitu 100.64.0.0/10 atau 198.19.0.0/16 karena cenderung tidak digunakan dalam pengaturan perusahaan daripada rentang lainnya. RFC1918 Untuk informasi tambahan tentang asosiasi blok CIDR yang diizinkan dan dibatasi yang dapat Anda gunakan dengan VPC Anda, IPv4 lihat Pembatasan asosiasi blok CIDR di bagian ukuran VPC dan subnet dari dokumentasi VPC.

Seperti yang ditunjukkan pada diagram di bawah ini, Antarmuka Jaringan Elastis primer (ENI) dari node pekerja masih menggunakan rentang CIDR VPC primer (dalam hal ini 10.0.0.0/16) tetapi sekunder menggunakan ENIs Rentang CIDR VPC sekunder (dalam hal ini 100.64.0.0/16). Sekarang, agar Pod menggunakan rentang CIDR 100.64.0.0/16, Anda harus mengkonfigurasi plugin CNI untuk menggunakan jaringan kustom. Anda dapat mengikuti langkah-langkah seperti yang didokumentasikan di sini.



Jika Anda ingin CNI menggunakan jaringan khusus, atur variabel `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG` lingkungan `ketrue`.

```
kubectl set env daemonset aws-node -n kube-system
 AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

Kapan`AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true`, CNI akan menetapkan alamat IP Pod dari subnet yang ditentukan dalam. `ENIConfig` Sumber daya `ENIConfig` kustom digunakan untuk menentukan subnet di mana Pod akan dijadwalkan.

```
apiVersion : crd.k8s.amazonaws.com/v1alpha1
kind : ENIConfig
metadata:
  name: us-west-2a
spec:
  securityGroups:
    - sg-0dff111a1d11c1c11
```

```
  subnet: subnet-011b111c1f11fdf11
```

Setelah membuat sumber daya `ENIconfig` khusus, Anda perlu membuat node pekerja baru dan menguras node yang ada. Node dan Pod pekerja yang ada akan tetap tidak terpengaruh.

Rekomendasi

Gunakan Jaringan Kustom Saat

Kami menyarankan Anda untuk mempertimbangkan jaringan khusus jika Anda berurusan dengan IPv4 kelelahan dan belum dapat menggunakannya IPv6 . Dukungan Amazon EKS untuk RFC6598ruang memungkinkan Anda menskalakan Pod di luar RFC1918mengatasi tantangan kelelahan. Harap pertimbangkan untuk menggunakan delegasi awalan dengan jaringan khusus untuk meningkatkan kepadatan Pod pada sebuah node.

Anda dapat mempertimbangkan jaringan khusus jika Anda memiliki persyaratan keamanan untuk menjalankan Pod di jaringan yang berbeda dengan persyaratan grup keamanan yang berbeda. Ketika jaringan kustom diaktifkan, pod menggunakan subnet atau grup keamanan yang berbeda seperti yang ENIConfig didefinisikan dalam antarmuka jaringan utama node.

Jaringan khusus memang merupakan pilihan ideal untuk menyebarkan beberapa kluster dan aplikasi EKS untuk menghubungkan layanan pusat data di lokasi. Anda dapat menambah jumlah alamat pribadi (RFC1918) yang dapat diakses EKS di VPC Anda untuk layanan seperti Amazon Elastic Load Balancing dan NAT-GW, sambil menggunakan ruang CG-NAT yang tidak dapat dirutekan untuk Pod Anda di beberapa cluster. Jaringan khusus dengan gateway transit dan VPC Layanan Bersama (termasuk gateway NAT di beberapa Availability Zone untuk ketersediaan tinggi) memungkinkan Anda memberikan arus lalu lintas yang terukur dan dapat diprediksi. Posting blog ini menjelaskan pola arsitektur yang merupakan salah satu cara yang paling direkomendasikan untuk menghubungkan Pod EKS ke jaringan pusat data menggunakan jaringan kustom.

Hindari Jaringan Kustom Saat

Siap untuk Implementasi IPv6

Jaringan khusus dapat mengurangi masalah kelelahan IP, tetapi membutuhkan overhead operasional tambahan. Jika saat ini Anda sedang menerapkan IPv6 VPC dual-stack (/IPv4) atau jika paket Anda menyertakan IPv6 dukungan, sebaiknya implementasikan cluster sebagai gantinya. IPv6 Anda dapat mengatur kluster IPv6 EKS dan memigrasikan aplikasi Anda. Dalam kluster IPv6 EKS, Kubernetes

dan Pod mendapatkan IPv6 alamat dan dapat berkomunikasi masuk dan keluar ke keduanya IPv4 dan titik akhir. IPv6 Harap tinjau praktik terbaik untuk Menjalankan Kluster IPv6 EKS.

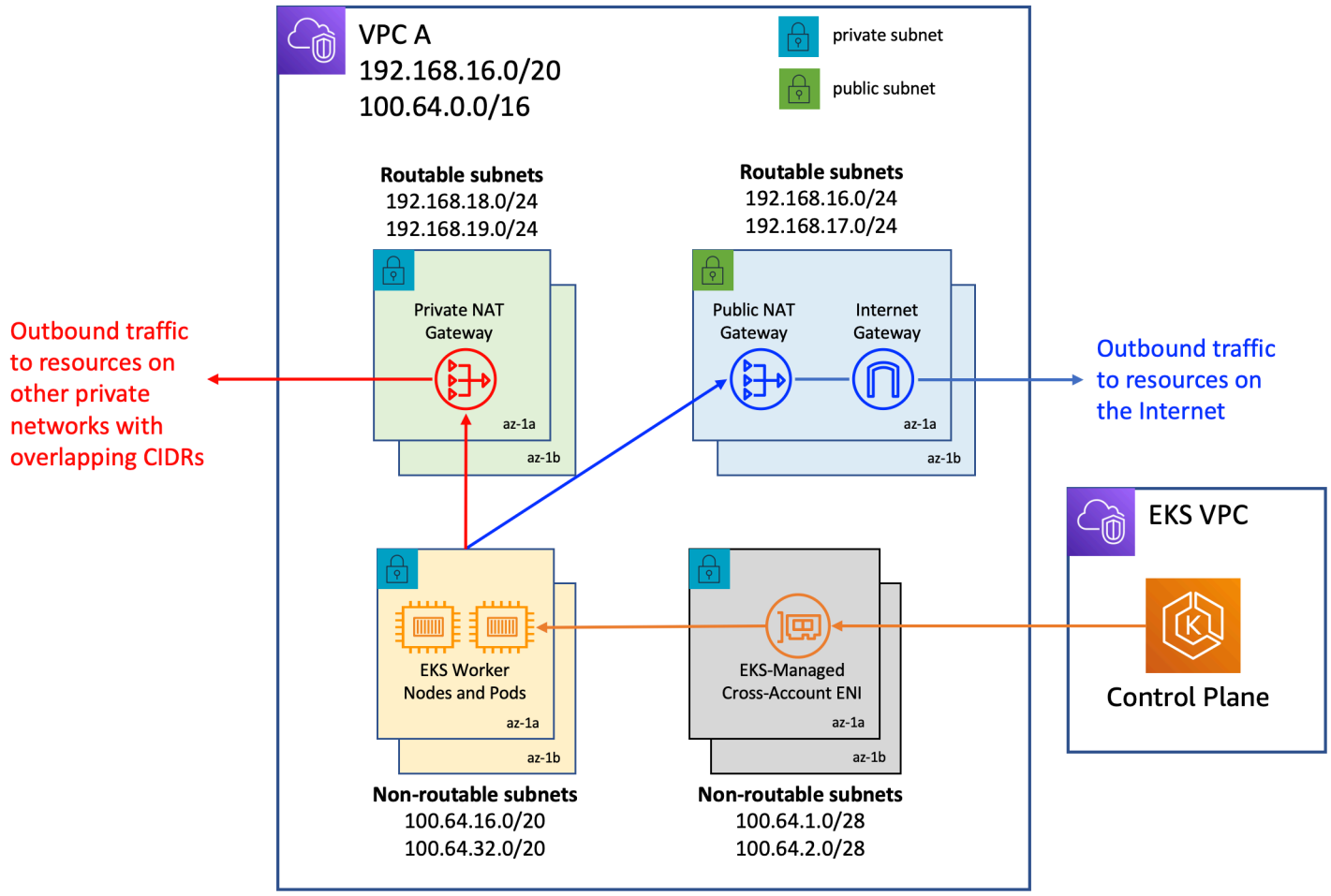
Ruang CG-NAT yang Lelah

Selain itu, jika saat ini Anda menggunakan CIDRs dari ruang CG-NAT atau tidak dapat menautkan CIDR sekunder dengan VPC cluster Anda, Anda mungkin perlu menjelajahi opsi lain, seperti menggunakan CNI alternatif. Kami sangat menyarankan agar Anda mendapatkan dukungan komersial atau memiliki pengetahuan internal untuk men-debug dan mengirimkan tambalan ke proyek plugin CNI open source. Lihat panduan pengguna Plugin CNI Alternatif untuk detail selengkapnya.

Gunakan Gateway NAT Pribadi

Amazon VPC sekarang menawarkan kemampuan gateway NAT pribadi. Gateway NAT pribadi Amazon memungkinkan instance di subnet pribadi untuk terhubung ke jaringan lain VPCs dan lokal dengan tumpang tindih. CIDRs Pertimbangkan untuk menggunakan metode yang dijelaskan pada posting blog ini untuk menggunakan gateway NAT pribadi untuk mengatasi masalah komunikasi untuk beban kerja EKS yang disebabkan oleh tumpang tindih CIDRs, keluhan signifikan yang diungkapkan oleh klien kami. Jaringan khusus tidak dapat mengatasi kesulitan CIDR yang tumpang tindih dengan sendirinya, dan itu menambah tantangan konfigurasi.

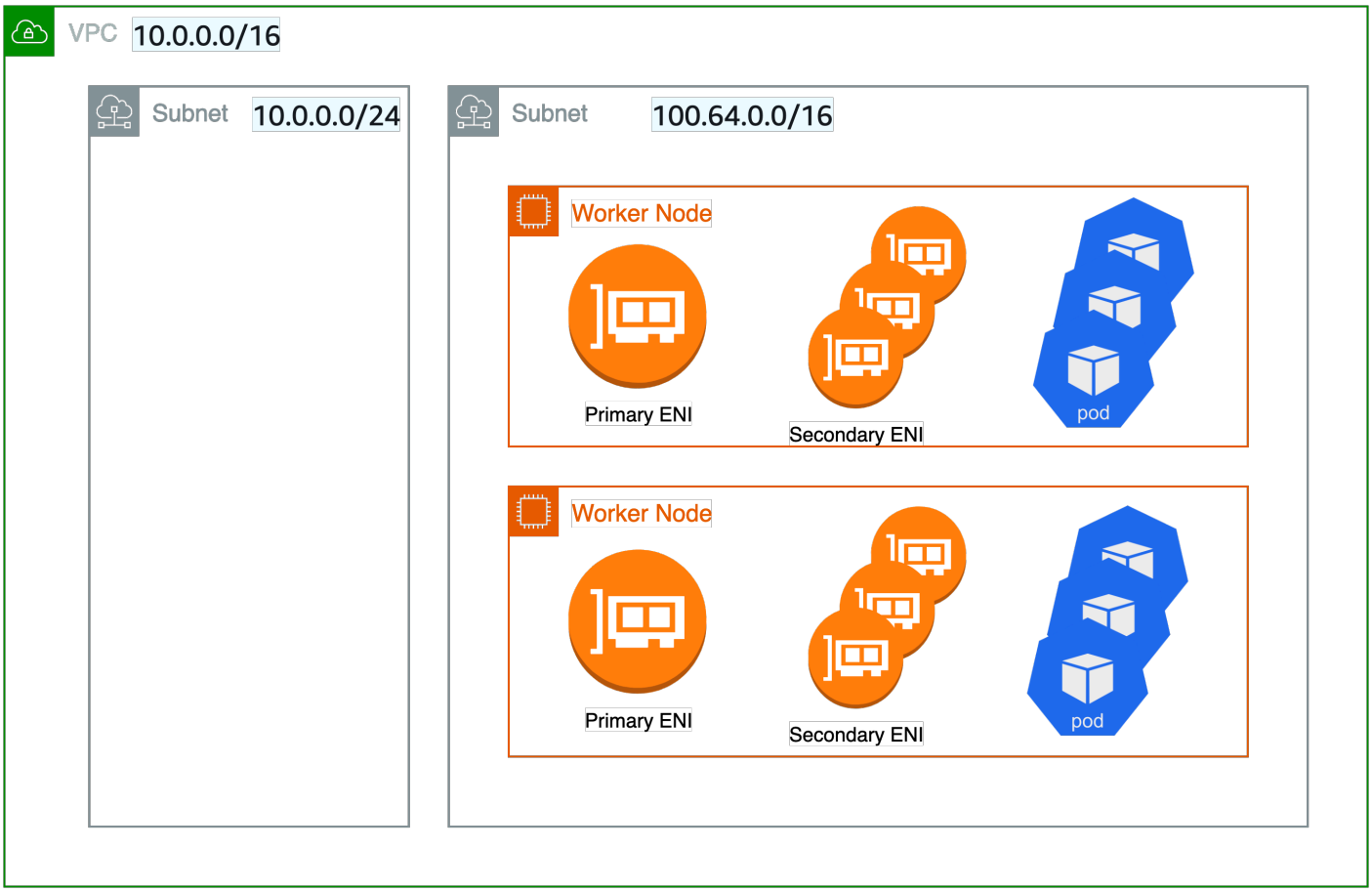
Arsitektur jaringan yang digunakan dalam implementasi posting blog ini mengikuti rekomendasi di bawah Aktifkan komunikasi antara jaringan yang tumpang tindih dalam dokumentasi Amazon VPC. Seperti yang ditunjukkan dalam posting blog ini, Anda dapat memperluas penggunaan Gateway NAT pribadi bersama dengan RFC6598 alamat untuk mengelola masalah kelelahan IP pribadi pelanggan. Kluster EKS, node pekerja dikerahkan dalam rentang CIDR sekunder VPC 100.64.0.0/16 yang tidak dapat dirutekan, sedangkan gateway NAT pribadi, gateway NAT diterapkan ke rentang CIDR yang dapat dirutekan. RFC1918 Blog menjelaskan bagaimana gateway transit digunakan untuk terhubung untuk memfasilitasi komunikasi lintas VPCs dengan rentang VPCs CIDR yang tidak dapat dirutekan yang tumpang tindih. Untuk kasus penggunaan di mana sumber daya EKS dalam rentang alamat non-routable VPC perlu berkomunikasi dengan orang lain yang tidak memiliki rentang alamat VPCs yang tumpang tindih, pelanggan memiliki opsi untuk menggunakan VPC Peering untuk menghubungkannya. VPCs Metode ini dapat memberikan potensi penghematan biaya karena semua transit data dalam Availability Zone melalui koneksi peering VPC sekarang gratis.



Jaringan unik untuk node dan Pod

Jika Anda perlu mengisolasi node dan Pod ke jaringan tertentu untuk alasan keamanan, sebaiknya Anda menyebarkan node dan Pod ke subnet dari blok CIDR sekunder yang lebih besar (mis. 100.64.0.0/8). Setelah instalasi CIDR baru di VPC Anda, Anda dapat menerapkan grup node lain menggunakan CIDR sekunder dan menguras node asli untuk secara otomatis menerapkan ulang pod ke node pekerja baru. Untuk informasi lebih lanjut tentang cara menerapkan ini, lihat posting blog ini.

Jaringan khusus tidak digunakan dalam pengaturan yang diwakili dalam diagram di bawah ini. Sebaliknya, node pekerja Kubernetes di-deploy pada subnet dari rentang CIDR VPC sekunder VPC Anda, seperti 100.64.0.0/10. Anda dapat menjaga cluster EKS tetap berjalan (bidang kontrol akan tetap pada aslinyasubnet/s), but the nodes and Pods will be moved to a secondary subnet/s. Ini adalah teknik lain, meskipun tidak konvensional, untuk mengurangi bahaya kelelahan IP dalam VPC. Kami mengusulkan untuk menguras node lama sebelum memindahkan pod ke node pekerja baru.



Otomatiskan Konfigurasi dengan Label Availability Zone

Anda dapat mengaktifkan Kubernetes untuk secara otomatis menerapkan yang sesuai ENIConfig untuk node pekerja Availability Zone (AZ).

Kubernetes secara otomatis menambahkan tag topology.kubernetes.io/zoneke node pekerja Anda. Amazon EKS merekomendasikan penggunaan zona ketersediaan sebagai nama konfigurasi ENI Anda ketika Anda hanya memiliki satu subnet sekunder (CIDR alternatif) per AZ. Anda kemudian dapat mengatur label yang digunakan untuk menemukan nama konfigurasi ENI. `topology.kubernetes.io/zone` Perhatikan bahwa tag `failure-domain.beta.kubernetes.io/zone` tidak digunakan lagi dan diganti dengan tag. `topology.kubernetes.io/zone`

1. Setel `name` bidang ke Availability Zone dari VPC Anda.
2. Aktifkan konfigurasi otomatis melalui perintah berikut
3. Mengatur label konfigurasi melalui perintah berikut

```
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"
```

Jika Anda memiliki beberapa subnet sekunder per zona ketersediaan, Anda perlu membuat yang spesifikENI_CONFIG_LABEL_DEF. Anda dapat mempertimbangkan untuk mengonfigurasi ENI_CONFIG_LABEL_DEF as k8s.amazonaws.com/eniConfigdan memberi label node dengan nama eniconFig khusus, seperti dan. k8s.amazonaws.com/eniConfig=us-west-2a-subnet-1k8s.amazonaws.com/eniConfig=us-west-2a-subnet-2

Ganti Pod saat Mengonfigurasi Jaringan Sekunder

Mengaktifkan jaringan kustom tidak mengubah node yang ada. Jaringan khusus adalah tindakan yang mengganggu. Daripada melakukan penggantian bergulir dari semua node pekerja di cluster Anda setelah mengaktifkan jaringan khusus, kami sarankan memperbarui CloudFormation template AWS di Panduan Memulai EKS dengan sumber daya khusus yang memanggil fungsi Lambda untuk memperbarui aws-node Daemonset dengan variabel lingkungan untuk mengaktifkan jaringan khusus sebelum node pekerja disediakan.

Jika Anda memiliki node di klaster dengan menjalankan Pod sebelum beralih ke fitur jaringan CNI kustom, Anda harus mem-cordon dan menguras node untuk mematikan Pod dengan anggun dan kemudian menghentikan node. Hanya node baru yang cocok dengan ENIConfig label atau anotasi yang menggunakan jaringan kustom, dan karenanya Pod yang dijadwalkan pada node baru ini dapat diberi IP dari CIDR sekunder.

Hitung Pod Maks per Node

Karena ENI primer node tidak lagi digunakan untuk menetapkan alamat IP Pod, ada penurunan jumlah Pod yang dapat Anda jalankan pada jenis instans EC2 tertentu. Untuk mengatasi batasan ini, Anda dapat menggunakan penetapan awalan dengan jaringan khusus. Dengan penugasan awalan, setiap IP sekunder diganti dengan awalan /28 pada sekunder. ENIs

Pertimbangkan jumlah maksimum Pod untuk instance m5.large dengan jaringan kustom.

Jumlah maksimum Pod yang dapat Anda jalankan tanpa penetapan awalan adalah 29

- 3 ENIs - 1) * (10 secondary IPs per ENI - 1 + 2 = 20

Mengaktifkan lampiran awalan meningkatkan jumlah Pod menjadi 290.

- `(3 ENIs - 1) * ((10 secondary IPs per ENI - 1) * 16 + 2 = 290`

Namun, kami menyarankan pengaturan max-pod ke 110 daripada 290 karena instance memiliki jumlah virtual yang agak kecil. CPUs Pada instance yang lebih besar, EKS merekomendasikan nilai pod maksimal 250. Saat menggunakan lampiran awalan dengan tipe instans yang lebih kecil (misalnya m5.large), ada kemungkinan Anda akan menghabiskan CPU dan sumber daya memori instans jauh sebelum alamat IP-nya.

> **Note**
>
> Ketika awalan CNI mengalokasikan awalan /28 ke ENI, itu harus berupa blok alamat IP yang berdekatan. Jika subnet tempat awalan dihasilkan sangat terfragmentasi, lampiran awalan mungkin gagal. Anda dapat mengurangi hal ini terjadi dengan membuat VPC khusus baru untuk cluster atau dengan memesan subnet satu set CIDR secara eksklusif untuk lampiran awalan. Kunjungi reservasi Subnet CIDR untuk informasi lebih lanjut tentang topik ini.

Identifikasi Penggunaan Ruang CG-NAT yang Ada

Jaringan khusus memungkinkan Anda untuk mengurangi masalah kelelahan IP, namun tidak dapat menyelesaikan semua tantangan. Jika Anda sudah menggunakan ruang CG-NAT untuk cluster Anda, atau tidak memiliki kemampuan untuk mengaitkan CIDR sekunder dengan VPC cluster Anda, kami sarankan Anda untuk menjelajahi opsi lain, seperti menggunakan CNI alternatif atau pindah ke cluster. IPv6

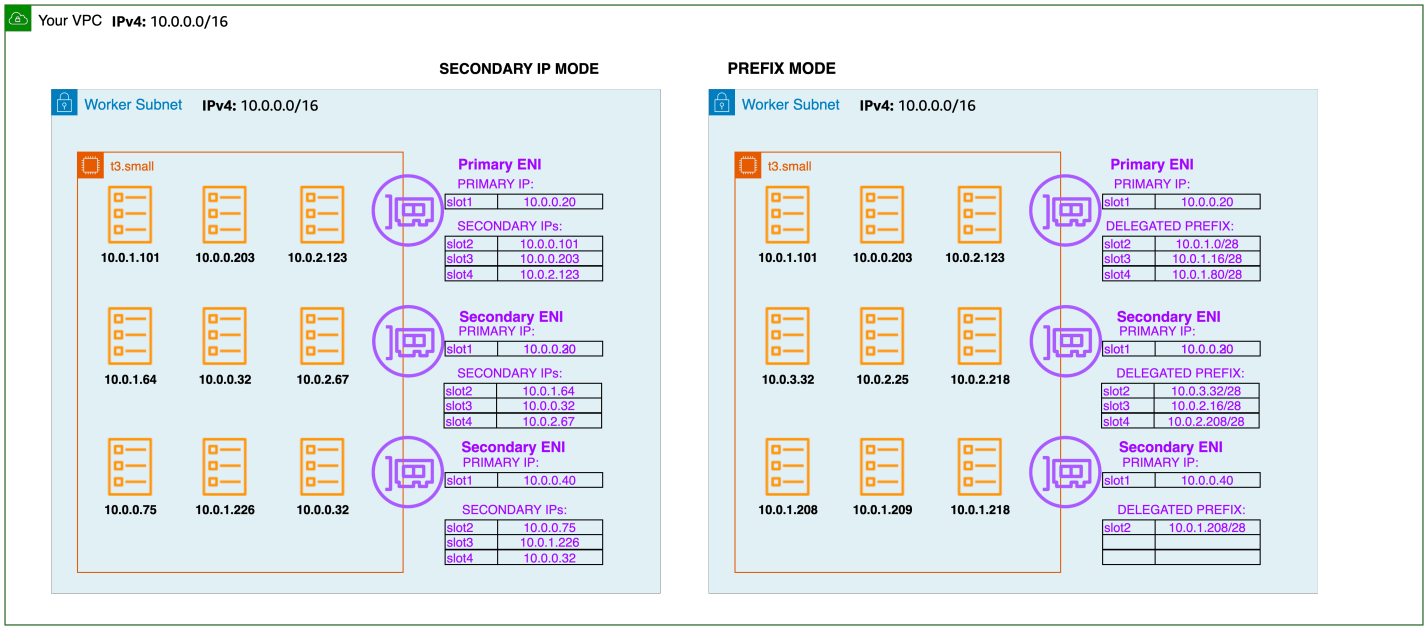
Mode Awalan untuk Linux

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Amazon VPC CNI memberikan awalan jaringan ke antarmuka jaringan Amazon EC2 untuk meningkatkan jumlah alamat IP yang tersedia untuk node dan meningkatkan kepadatan pod per node. Anda dapat mengonfigurasi versi 1.9.0 atau yang lebih baru dari add-on Amazon VPC CNI untuk menetapkan IPv4 dan IPv6 CIDRs alih-alih menetapkan alamat IP sekunder individual ke antarmuka jaringan.

Mode awalan diaktifkan secara default pada IPv6 cluster dan merupakan satu-satunya pilihan yang didukung. VPC CNI memberikan IPv6 awalan /80 ke slot pada ENI. Silakan merujuk ke IPv6 bagian panduan ini untuk informasi lebih lanjut.

Dengan mode penetapan awalan, jumlah maksimum antarmuka jaringan elastis per jenis instans tetap sama, tetapi Anda sekarang dapat mengonfigurasi Amazon VPC CNI untuk menetapkan awalan alamat /28 (16 alamat IP), alih-alih menetapkan IPv4 alamat individual ke slot pada antarmuka jaringan. IPv4 Kapan `ENABLE_PREFIX_DELEGATION` disetel ke VPC true, CNI mengalokasikan alamat IP ke Pod dari awalan yang ditetapkan ke ENI. Ikuti petunjuk yang disebutkan dalam panduan pengguna EKS untuk mengaktifkan mode IP Awalan.

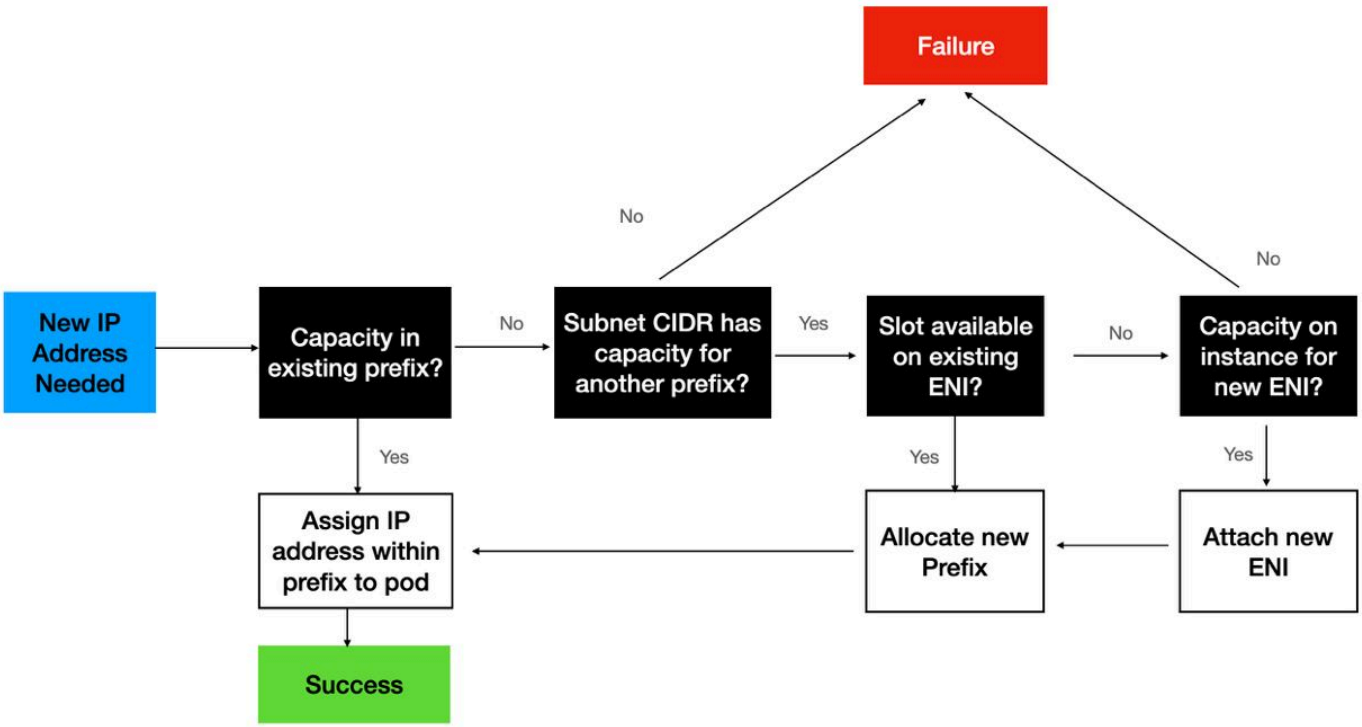


Jumlah maksimum alamat IP yang dapat Anda tetapkan ke antarmuka jaringan tergantung pada tipe instans. Setiap prefiks yang Anda tetapkan ke antarmuka jaringan dihitung sebagai satu alamat IP. Misalnya, `c5.large` instans memiliki batas alamat `10` IPv4 per antarmuka jaringan. Setiap antarmuka jaringan untuk instans ini memiliki alamat IPv4 primer. Jika antarmuka jaringan tidak memiliki alamat IPv4 sekunder, Anda dapat menetapkan hingga 9 prefiks ke antarmuka jaringan. Untuk setiap alamat IPv4 tambahan yang Anda tetapkan ke antarmuka jaringan, Anda dapat menetapkan satu prefiks kurang ke antarmuka jaringan. Tinjau dokumentasi AWS EC2 tentang alamat IP per antarmuka jaringan per jenis instans dan tetapkan awalan ke antarmuka jaringan.

Selama inisialisasi node pekerja, VPC CNI memberikan satu atau lebih awalan ke ENI primer. CNI mengalokasikan awalan untuk startup pod yang lebih cepat dengan mempertahankan kolam yang hangat. Jumlah awalan yang akan disimpan di kolam hangat dapat dikontrol dengan mengatur variabel lingkungan.

- `WARM_PREFIX_TARGET`, jumlah awalan yang akan dialokasikan melebihi kebutuhan saat ini.
- `WARM_IP_TARGET`, jumlah alamat IP yang akan dialokasikan melebihi kebutuhan saat ini.
- `MINIMUM_IP_TARGET`, jumlah minimum alamat IP yang tersedia kapan saja.
- `WARM_IP_TARGET`dan `MINIMUM_IP_TARGET` if set akan menimpa`WARM_PREFIX_TARGET`.

Karena semakin banyak Pod yang dijadwalkan, prefiks tambahan akan diminta untuk ENI yang ada. Pertama, VPC CNI mencoba mengalokasikan awalan baru ke ENI yang ada. Jika ENI berkapasitas, VPC CNI mencoba mengalokasikan ENI baru ke node. Baru ENIs akan dilampirkan sampai batas ENI maksimum (ditentukan oleh jenis instance) tercapai. Ketika ENI baru dilampirkan, ipamd akan mengalokasikan satu atau lebih awalan yang diperlukan untuk mempertahankan,, dan pengaturan. `WARM_PREFIX_TARGET` `WARM_IP_TARGET` `MINIMUM_IP_TARGET`



Rekomendasi

Gunakan Mode Awalan saat

Gunakan mode awalan jika Anda mengalami masalah kepadatan Pod pada node pekerja. Untuk menghindari kesalahan VPC CNI, sebaiknya periksa subnet untuk blok alamat yang berdekatan

untuk awalan /28 sebelum bermigrasi ke mode awalan. Silakan lihat bagian "Gunakan Reservasi Subnet untuk Menghindari Fragmentasi Subnet (IPv4)" untuk rincian reservasi Subnet.

Untuk kompatibilitas mundur, batas max-pod diatur untuk mendukung mode IP sekunder. Untuk meningkatkan kepadatan pod, tentukan `max-pods` nilainya ke Kubelet dan `--use-max-pods=false` sebagai data pengguna untuk node. Anda dapat mempertimbangkan untuk menggunakan max-pod-calculatorskrip.sh untuk menghitung jumlah maksimum Pod yang direkomendasikan EKS untuk jenis instance tertentu. Lihat panduan pengguna EKS misalnya data pengguna.

```
./max-pods-calculator.sh --instance-type m5.large --cni-version ``1.9``.0 --cni-prefix-delegation-enabled
```

Mode penetapan awalan sangat relevan bagi pengguna jaringan kustom CNI di mana ENI primer tidak digunakan untuk pod. Dengan penetapan awalan, Anda masih dapat melampirkan lebih banyak IPs pada hampir setiap jenis instans Nitro, bahkan tanpa ENI primer yang digunakan untuk pod.

Hindari Mode Awalan saat

Jika subnet Anda sangat terfragmentasi dan tidak memiliki alamat IP yang tersedia untuk membuat awalan /28, hindari menggunakan mode awalan. Lampiran awalan mungkin gagal jika subnet dari mana awalan dihasilkan terfragmentasi (subnet yang banyak digunakan dengan alamat IP sekunder yang tersebar). Masalah ini dapat dihindari dengan membuat subnet baru dan memesan awalan.

Dalam mode awalan, grup keamanan yang ditetapkan ke node pekerja dibagikan oleh Pod. Pertimbangkan untuk menggunakan Grup keamanan untuk Pod jika Anda memiliki persyaratan keamanan untuk mencapai kepatuhan dengan menjalankan aplikasi dengan berbagai persyaratan keamanan jaringan pada sumber daya komputasi bersama.

Gunakan Jenis Instance Serupa di Grup Node yang sama

Grup node Anda mungkin berisi instance dari banyak jenis. Jika sebuah instance memiliki jumlah pod maksimum yang rendah, nilai tersebut diterapkan ke semua node dalam grup node. Pertimbangkan untuk menggunakan jenis instance serupa dalam grup node untuk memaksimalkan penggunaan node. Kami merekomendasikan untuk mengonfigurasi node.kubernetes.io/instance-type di bagian persyaratan API penyedia jika Anda menggunakan Karpenter untuk penskalaan node otomatis.

> **Warning**
>
> Jumlah pod maksimum untuk semua node dalam grup node tertentu ditentukan oleh jumlah pod maksimum terendah dari setiap jenis instance tunggal dalam grup node.

`WARM_PREFIX_TARGET`Konfigurasikan untuk menghemat alamat IPv4

Nilai default manifes instalasi untuk `WARM_PREFIX_TARGET` adalah 1. Dalam kebanyakan kasus, nilai yang direkomendasikan dari 1 for `WARM_PREFIX_TARGET` akan memberikan campuran yang baik dari waktu peluncuran pod cepat sambil meminimalkan alamat IP yang tidak digunakan yang ditetapkan ke instance.

Jika Anda memiliki kebutuhan untuk lebih menghemat IPv4 alamat per penggunaan `WARM_IP_TARGET` dan `MINIMUM_IP_TARGET` pengaturan node, yang akan menimpa `WARM_PREFIX_TARGET` saat dikonfigurasi. Dengan menyetel `WARM_IP_TARGET` ke nilai kurang dari 16, Anda dapat mencegah CNI menjaga seluruh awalan berlebih terpasang.

Lebih suka mengalokasikan awalan baru daripada melampirkan ENI baru

Mengalokasikan awalan tambahan ke ENI yang ada adalah operasi API EC2 yang lebih cepat daripada membuat dan melampirkan ENI baru ke instance. Menggunakan awalan meningkatkan kinerja sekaligus hemat dengan IPv4 alokasi alamat. Melampirkan awalan biasanya selesai dalam waktu kurang dari satu detik, sedangkan melampirkan ENI baru dapat memakan waktu hingga 10 detik. Untuk sebagian besar kasus penggunaan, CNI hanya memerlukan satu ENI per node pekerja saat berjalan dalam mode awalan. Jika Anda mampu (dalam kasus terburuk) hingga 15 yang tidak digunakan IPs per node, kami sangat menyarankan untuk menggunakan mode jaringan penetapan awalan yang lebih baru, dan menyadari peningkatan kinerja dan efisiensi yang menyertainya.

Gunakan Reservasi Subnet untuk Menghindari Fragmentasi Subnet () IPv4

Ketika EC2 mengalokasikan IPv4 awalan /28 ke ENI, itu harus berupa blok alamat IP yang berdekatan dari subnet Anda. Jika subnet tempat awalan dihasilkan terfragmentasi (subnet yang sangat digunakan dengan alamat IP sekunder yang tersebar), lampiran awalan mungkin gagal, dan Anda akan melihat pesan kesalahan berikut di log CNI VPC:

```
failed to allocate a private IP/Prefix address: InsufficientCidrBlocks: There are not
 enough free cidr blocks in the specified subnet to satisfy the request.
```

Untuk menghindari fragmentasi dan memiliki ruang berdekatan yang cukup untuk membuat awalan, Anda dapat menggunakan reservasi VPC Subnet CIDR untuk memesan ruang IP dalam subnet untuk penggunaan eksklusif dengan awalan. Setelah Anda membuat reservasi, plugin VPC CNI akan memanggil EC2 APIs untuk menetapkan awalan yang secara otomatis dialokasikan dari ruang yang dipesan.

Disarankan untuk membuat subnet baru, mencadangkan ruang untuk awalan, dan mengaktifkan penetapan awalan dengan VPC CNI untuk node pekerja yang berjalan di subnet tersebut. Jika subnet baru hanya didedikasikan untuk Pod yang berjalan di klaster EKS Anda dengan penetapan awalan VPC CNI diaktifkan, maka Anda dapat melewati langkah reservasi awalan.

Hindari menurunkan versi VPC CNI

Mode awalan bekerja dengan VPC CNI versi 1.9.0 dan yang lebih baru. Penurunan versi add-on Amazon VPC CNI ke versi yang lebih rendah dari 1.9.0 harus dihindari setelah mode awalan diaktifkan dan awalan ditetapkan. ENIs Anda harus menghapus dan membuat ulang node jika Anda memutuskan untuk menurunkan versi VPC CNI.

Ganti semua node selama transisi ke Delegasi Awalan

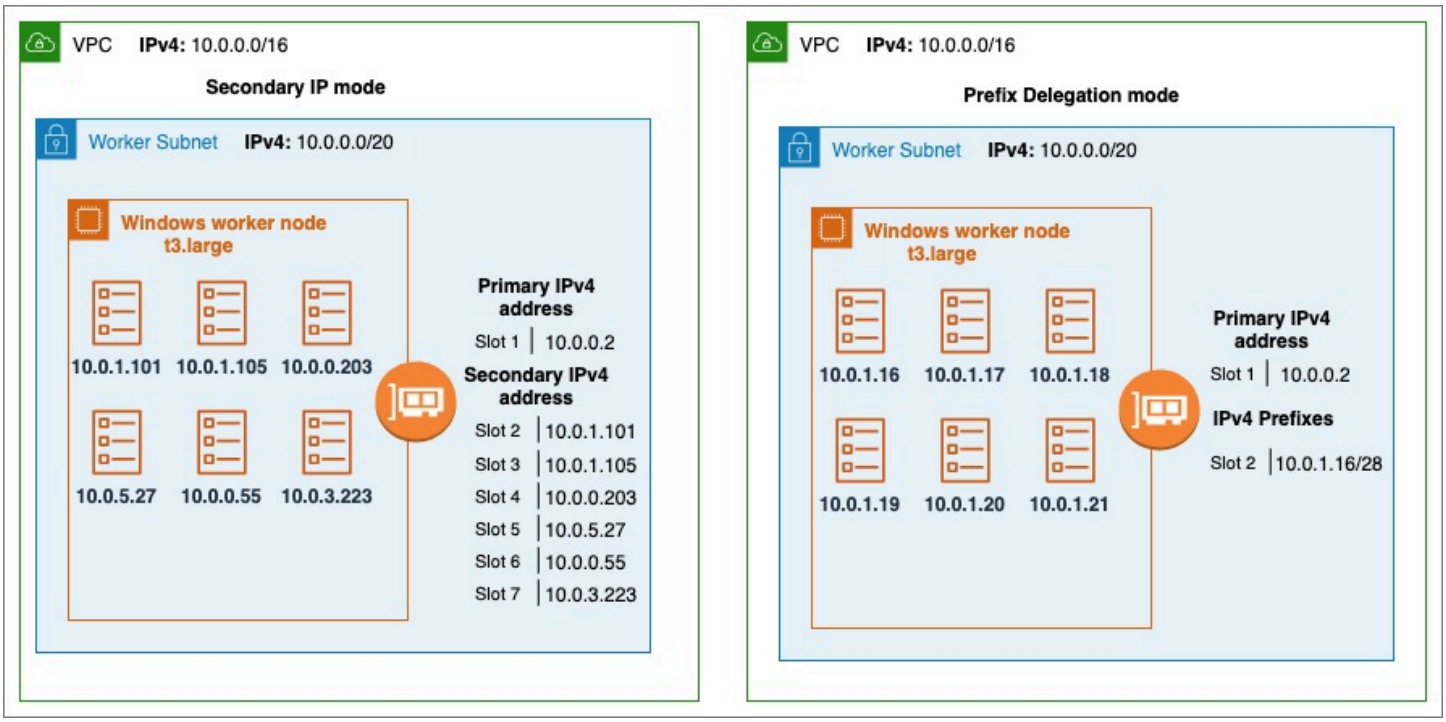
Sangat disarankan agar Anda membuat grup node baru untuk meningkatkan jumlah alamat IP yang tersedia daripada melakukan penggantian bergulir node pekerja yang ada. Cordon dan tiriskan semua node yang ada untuk mengusir semua Pod yang ada dengan aman. Untuk mencegah gangguan layanan, kami sarankan untuk menerapkan Anggaran Gangguan Pod pada klaster produksi Anda untuk beban kerja yang kritis. Pod pada node baru akan diberi IP dari awalan yang ditetapkan ke ENI. Setelah Anda mengonfirmasi bahwa Pod sedang berjalan, Anda dapat menghapus node lama dan grup node. Jika Anda menggunakan grup node terkelola, ikuti langkah-langkah yang disebutkan di sini untuk menghapus grup node dengan aman.

Mode Awalan untuk Windows

Di Amazon EKS, setiap Pod yang berjalan pada host Windows diberi alamat IP sekunder oleh pengontrol sumber daya VPC secara default. Alamat IP ini adalah alamat VPC-routable yang dialokasikan dari subnet host. Di Linux, setiap ENI yang melekat pada instance memiliki beberapa slot yang dapat diisi oleh alamat IP sekunder atau /28 CIDR (awalan). Host Windows, bagaimanapun, hanya mendukung satu ENI dan slot yang tersedia. Hanya menggunakan alamat IP sekunder dapat membatasi jumlah pod yang dapat Anda jalankan di host Windows, bahkan ketika ada banyak alamat IP yang tersedia untuk penugasan.

Untuk meningkatkan kepadatan pod pada host Windows, terutama saat menggunakan tipe instans yang lebih kecil, Anda dapat mengaktifkan Delegasi Awalan untuk node Windows. Ketika delegasi awalan diaktifkan, IPv4 awalan /28 ditetapkan ke slot ENI daripada alamat IP sekunder. Delegasi awalan dapat diaktifkan dengan menambahkan `enable-windows-prefix-delegation: "true"` entri ke peta `amazon-vpc-cni` konfigurasi. Ini adalah peta konfigurasi yang sama di mana Anda perlu mengatur `enable-windows-ipam: "true"` entri untuk mengaktifkan dukungan Windows.

Ikuti petunjuk yang disebutkan dalam panduan pengguna EKS untuk mengaktifkan mode Delegasi Awalan untuk node Windows.



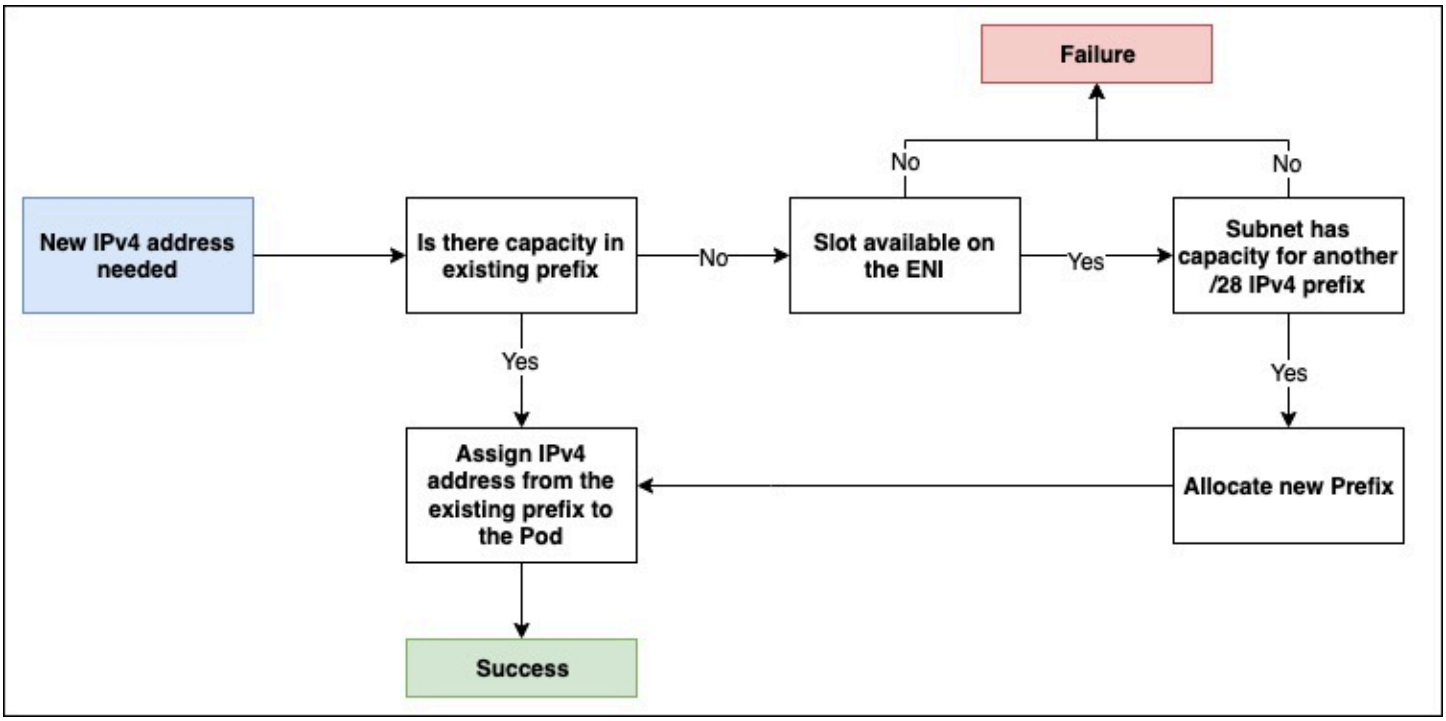
Gambar: Perbandingan mode IP Sekunder dengan mode Delegasi Awalan

Jumlah maksimum alamat IP yang dapat Anda tetapkan ke antarmuka jaringan tergantung pada jenis instans dan ukurannya. Setiap awalan yang ditetapkan ke antarmuka jaringan mengkonsumsi slot yang tersedia. Misalnya, sebuah `c5.large` instance memiliki batas `10` slot per antarmuka jaringan. Slot pertama pada antarmuka jaringan selalu dikonsumsi oleh alamat IP utama antarmuka, meninggalkan Anda dengan 9 slot untuk awalan alamat IP and/or sekunder. Jika slot ini diberi awalan, node dapat mendukung (9 * 16) 144 alamat IP sedangkan jika mereka diberi alamat IP sekunder hanya dapat mendukung 9 alamat IP. Lihat dokumentasi alamat IP per antarmuka jaringan per jenis instans dan tetapkan awalan ke antarmuka jaringan untuk informasi lebih lanjut.

Selama inisialisasi node pekerja, VPC Resource Controller memberikan satu atau lebih awalan ke ENI primer untuk startup pod yang lebih cepat dengan mempertahankan kumpulan alamat IP yang hangat. Jumlah awalan yang akan disimpan di kolam hangat dapat dikontrol dengan mengatur parameter konfigurasi berikut di `amazon-vpc-cni` peta konfigurasi.

- `warm-prefix-target`, jumlah awalan yang akan dialokasikan melebihi kebutuhan saat ini.
- `warm-ip-target`, jumlah alamat IP yang akan dialokasikan melebihi kebutuhan saat ini.
- `minimum-ip-target`, jumlah minimum alamat IP yang tersedia kapan saja.
- `warm-ip-target` and/or `minimum-ip-target`jika set akan menimpa`warm-prefix-target`.

Karena semakin banyak Pod yang dijadwalkan pada node, prefiks tambahan akan diminta untuk ENI yang ada. Ketika sebuah Pod dijadwalkan pada node, VPC Resource Controller pertama-tama akan mencoba untuk menetapkan IPv4 alamat dari prefiks yang ada pada node. Jika itu tidak memungkinkan, maka IPv4 awalan baru akan diminta selama subnet memiliki kapasitas yang diperlukan.



Gambar: Alur kerja selama penetapan IPv4 alamat ke Pod

Rekomendasi

Gunakan Delegasi Awalan saat

Gunakan delegasi awalan jika Anda mengalami masalah kepadatan Pod pada node pekerja. Untuk menghindari kesalahan, sebaiknya periksa subnet untuk blok alamat yang berdekatan untuk awalan /28 sebelum bermigrasi ke mode awalan. Silakan lihat bagian "Gunakan Reservasi Subnet untuk Menghindari Fragmentasi Subnet (IPv4)" untuk rincian reservasi Subnet.

Secara default, node `max-pods` pada Windows diatur ke`110`. Untuk sebagian besar jenis instance, ini sudah cukup. Jika Anda ingin menambah atau mengurangi batas ini, tambahkan yang berikut ini ke perintah bootstrap di data pengguna Anda:

```
-KubeletExtraArgs '--max-pods=example-value'
```

Untuk detail lebih lanjut tentang parameter konfigurasi bootstrap untuk node Windows, silakan kunjungi dokumentasi di sini.

Hindari Delegasi Awalan saat

Jika subnet Anda sangat terfragmentasi dan tidak memiliki alamat IP yang tersedia untuk membuat awalan /28, hindari menggunakan mode awalan. Lampiran awalan mungkin gagal jika subnet dari mana awalan dihasilkan terfragmentasi (subnet yang banyak digunakan dengan alamat IP sekunder yang tersebar). Masalah ini dapat dihindari dengan membuat subnet baru dan memesan awalan.

Konfigurasikan parameter untuk delegasi awalan untuk menghemat alamat IPv4

`warm-prefix-target`,`warm-ip-target`, dan `minimum-ip-target` dapat digunakan untuk menyempurnakan perilaku pra-penskalaan dan penskalaan dinamis dengan awalan. Secara default, nilai-nilai berikut digunakan:

```
warm-ip-target: "1"
minimum-ip-target: "3"
```

Dengan menyempurnakan parameter konfigurasi ini, Anda dapat mencapai keseimbangan optimal dalam melestarikan alamat IP dan memastikan latensi Pod menurun karena penetapan alamat IP. Untuk informasi lebih lanjut tentang parameter konfigurasi ini, kunjungi dokumentasi di sini.

Gunakan Reservasi Subnet untuk Menghindari Fragmentasi Subnet () IPv4

Ketika EC2 mengalokasikan IPv4 awalan /28 ke ENI, itu harus berupa blok alamat IP yang berdekatan dari subnet Anda. Jika subnet tempat awalan dihasilkan terfragmentasi (subnet yang sangat digunakan dengan alamat IP sekunder yang tersebar), lampiran awalan mungkin gagal, dan Anda akan melihat peristiwa node berikut:

```
InsufficientCidrBlocks: The specified subnet does not have enough free cidr blocks to
 satisfy the request
```

Untuk menghindari fragmentasi dan memiliki ruang berdekatan yang cukup untuk membuat awalan, gunakan reservasi VPC Subnet CIDR untuk memesan ruang IP dalam subnet untuk penggunaan eksklusif dengan awalan. Setelah Anda membuat reservasi, alamat IP dari blok yang dicadangkan tidak akan ditetapkan ke sumber daya lain. Dengan begitu, VPC Resource Controller akan bisa mendapatkan awalan yang tersedia selama panggilan penugasan ke node ENI.

Disarankan untuk membuat subnet baru, ruang cadangan untuk awalan, dan mengaktifkan penetapan awalan untuk node pekerja yang berjalan di subnet tersebut. Jika subnet baru hanya didedikasikan untuk Pod yang berjalan di klaster EKS Anda dengan delegasi awalan diaktifkan, maka Anda dapat melewati langkah reservasi awalan.

Ganti semua node saat bermigrasi dari mode IP Sekunder ke mode Delegasi Awalan atau sebaliknya

Sangat disarankan agar Anda membuat grup node baru untuk meningkatkan jumlah alamat IP yang tersedia daripada melakukan penggantian bergulir node pekerja yang ada.

Saat menggunakan grup simpul yang dikelola sendiri, langkah-langkah untuk transisi adalah:

- Tingkatkan kapasitas di klaster Anda sehingga node baru dapat mengakomodasi beban kerja Anda
- Aktifkan/Nonaktifkan fitur Delegasi Awalan untuk Windows
- Cordon dan tiriskan semua node yang ada untuk mengusir semua Pod yang ada dengan aman. Untuk mencegah gangguan layanan, kami sarankan untuk menerapkan Anggaran Gangguan Pod pada klaster produksi Anda untuk beban kerja yang kritis.
- Setelah Anda mengonfirmasi bahwa Pod sedang berjalan, Anda dapat menghapus node lama dan grup node. Pod pada node baru akan diberi IPv4 alamat dari awalan yang ditetapkan ke node ENI.

Saat menggunakan grup simpul terkelola, langkah-langkah untuk transisi adalah:

- Aktifkan/Nonaktifkan fitur Delegasi Awalan untuk Windows
- Perbarui grup node menggunakan langkah-langkah yang disebutkan di sini. Ini melakukan langkah-langkah serupa seperti di atas tetapi dikelola oleh EKS.

> **Warning**
>
> Jalankan semua Pod pada sebuah node dalam mode yang sama

Untuk Windows, kami menyarankan agar Anda menghindari menjalankan Pod dalam mode IP sekunder dan mode delegasi awalan secara bersamaan. Situasi seperti itu dapat muncul ketika Anda bermigrasi dari mode IP sekunder ke mode delegasi awalan atau sebaliknya dengan menjalankan beban kerja Windows.

Meskipun hal ini tidak akan memengaruhi Pod yang sedang berjalan, mungkin ada ketidakkonsistenan sehubungan dengan kapasitas alamat IP node. Misalnya, pertimbangkan bahwa node t3.xlarge yang memiliki 14 slot untuk alamat sekunder. IPv4 Jika Anda menjalankan 10 Pod, maka 10 slot pada ENI akan dikonsumsi oleh alamat IP sekunder. Setelah Anda mengaktifkan delegasi awalan, kapasitas yang diiklankan ke server kube-api adalah (14 slot * 16 alamat ip per awalan) 244 tetapi kapasitas sebenarnya pada saat itu adalah (4 slot yang tersisa * 16 alamat per awalan) 64. Ketidakkonsistenan antara jumlah kapasitas yang diiklankan dan jumlah kapasitas aktual (slot yang tersisa) ini dapat menyebabkan masalah jika Anda menjalankan lebih banyak Pod daripada alamat IP yang tersedia untuk penetapan.

Karena itu, Anda dapat menggunakan strategi migrasi seperti yang dijelaskan di atas untuk mentransisikan Pod Anda dengan aman dari alamat IP sekunder ke alamat yang diperoleh dari awalan. Saat beralih di antara mode, Pod akan terus berjalan normal dan:

- Saat beralih dari mode IP sekunder ke mode delegasi awalan, alamat IP sekunder yang ditetapkan ke pod yang sedang berjalan tidak akan dirilis. Awalan akan ditetapkan ke slot gratis. Setelah pod dihentikan, IP sekunder dan slot yang digunakannya akan dirilis.
- Saat beralih dari mode delegasi awalan ke mode IP sekunder, awalan akan dirilis ketika semua yang berada IPs dalam jangkauannya tidak lagi dialokasikan ke pod. Jika ada IP dari awalan yang ditetapkan ke sebuah pod maka prefiks tersebut akan disimpan sampai pod dihentikan.

Masalah Debugging dengan Delegasi Awalan

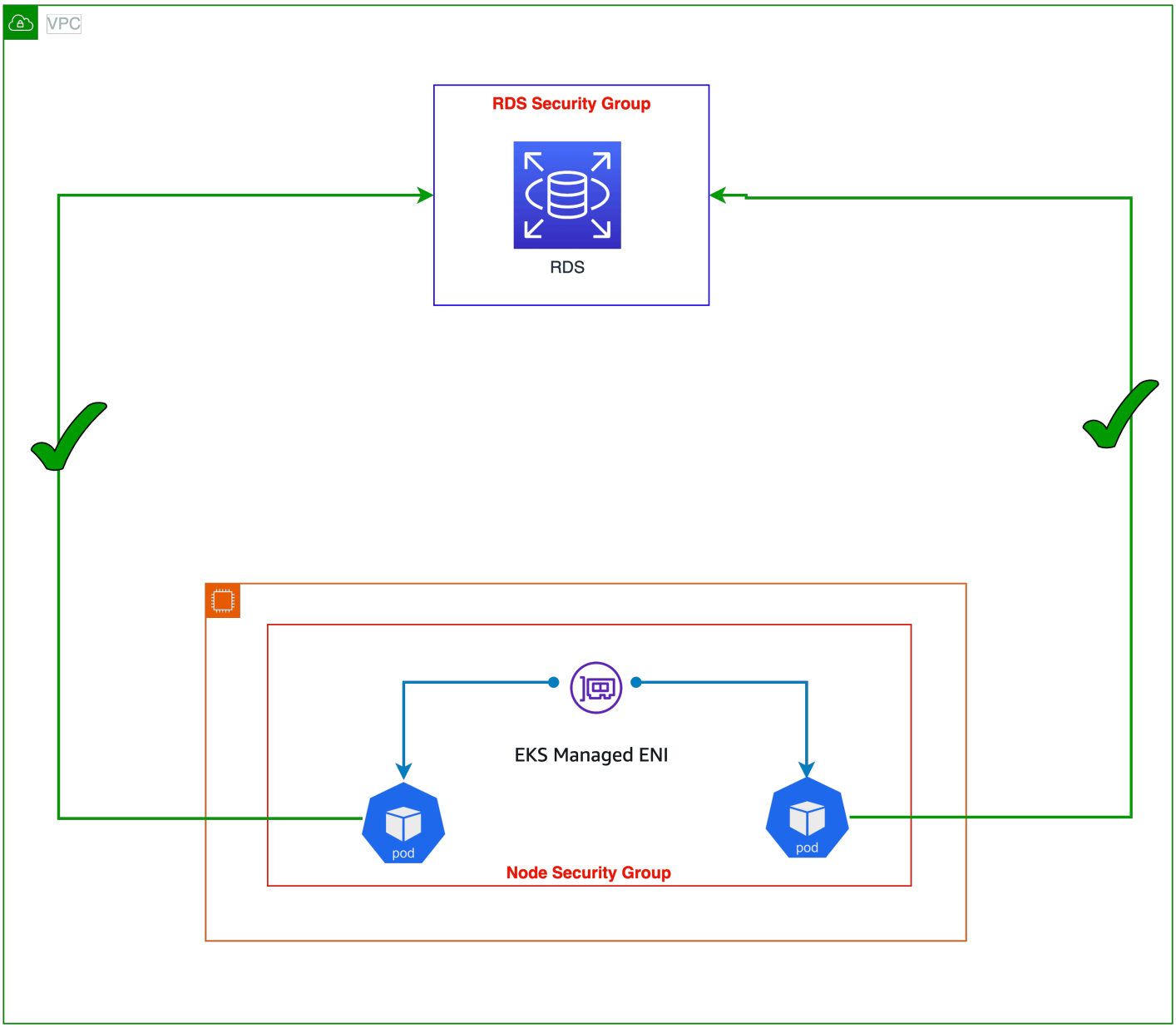
Anda dapat menggunakan panduan debugging kami di sini untuk menyelami masalah yang Anda hadapi dengan delegasi awalan di Windows.

Grup Keamanan Per Pod

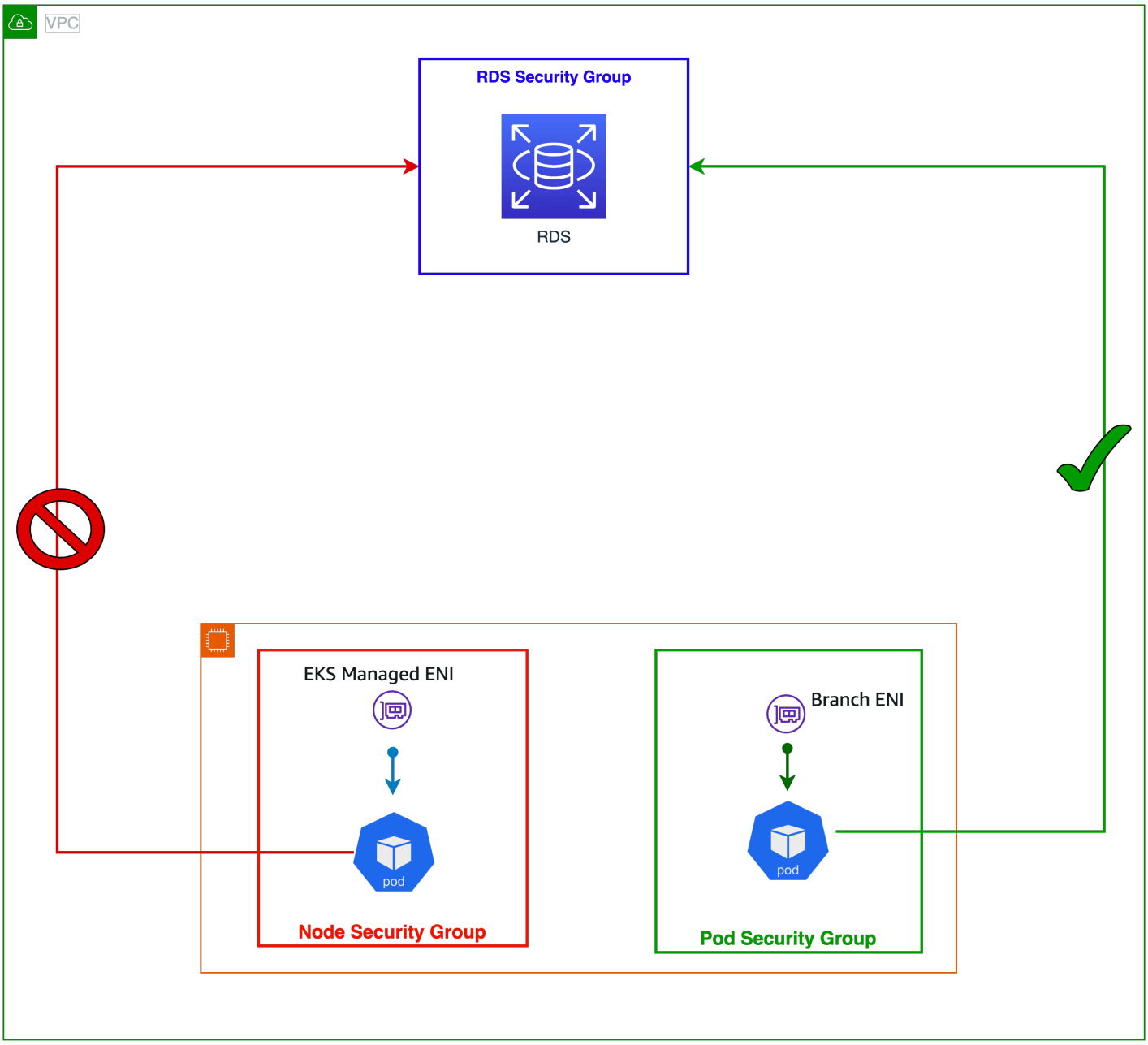
> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Grup keamanan AWS bertindak sebagai firewall virtual untuk instans EC2 untuk mengontrol lalu lintas masuk dan keluar. Secara default, Amazon VPC CNI akan menggunakan grup keamanan yang terkait dengan ENI utama pada node. Lebih khusus lagi, setiap ENI yang terkait dengan instans akan memiliki Grup Keamanan EC2 yang sama. Dengan demikian, setiap Pod pada sebuah node berbagi grup keamanan yang sama dengan node yang dijalankannya.

Seperti yang terlihat pada gambar di bawah ini, semua Pod aplikasi yang beroperasi pada node pekerja akan memiliki akses ke layanan database RDS (mengingat RDS inbound memungkinkan grup keamanan node). Kelompok keamanan terlalu kasar karena mereka berlaku untuk semua Pod yang berjalan pada sebuah node. Grup keamanan untuk Pod menyediakan segmentasi jaringan untuk beban kerja yang merupakan bagian penting dari strategi pertahanan mendalam yang baik.

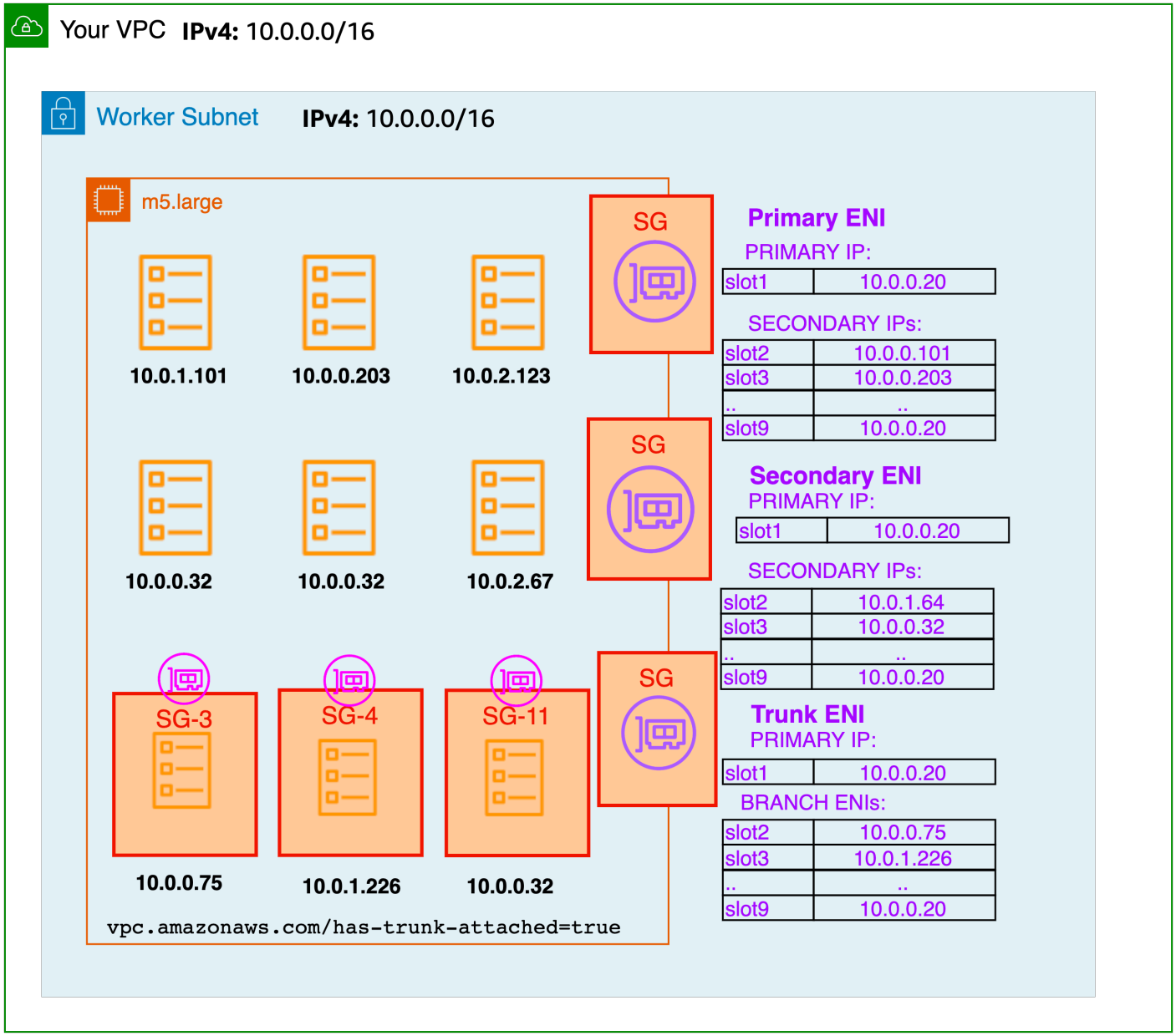


Dengan grup keamanan untuk Pod, Anda dapat meningkatkan efisiensi komputasi dengan menjalankan aplikasi dengan berbagai persyaratan keamanan jaringan pada sumber daya komputasi bersama. Berbagai jenis aturan keamanan, seperti Pod-to-Pod dan layanan Pod-to-External AWS, dapat didefinisikan di satu tempat dengan grup keamanan EC2 dan diterapkan pada beban kerja dengan Kubernetes native. APIs Gambar di bawah ini menunjukkan grup keamanan yang diterapkan pada level Pod dan bagaimana mereka menyederhanakan penerapan aplikasi dan arsitektur node Anda. Pod sekarang dapat mengakses database Amazon RDS.



Anda dapat mengaktifkan grup keamanan untuk Pod dengan menyetel `ENABLE_POD_ENI=true` untuk VPC CNI. Setelah diaktifkan, VPC Resource Controller yang berjalan di bidang kontrol (dikelola oleh EKS) membuat dan melampirkan antarmuka trunk yang disebut "`aws-k8 "`s-trunk-enike node. Antarmuka trunk bertindak sebagai antarmuka jaringan standar yang melekat pada instance. Untuk mengelola antarmuka trunk, Anda harus menambahkan kebijakan `AmazonEKSVPCResourceController` terkelola ke peran klaster yang menyertai klaster Amazon EKS Anda.

Pengontrol juga membuat antarmuka cabang bernama “aws-k8s-branch-eni" dan mengaitkannya dengan antarmuka trunk. Pod diberi grup keamanan menggunakan sumber daya SecurityGroupPolicykustom dan dikaitkan dengan antarmuka cabang. Karena grup keamanan ditentukan dengan antarmuka jaringan, kita sekarang dapat menjadwalkan Pod yang membutuhkan grup keamanan tertentu pada antarmuka jaringan tambahan ini. Tinjau Bagian Panduan Pengguna EKS tentang Grup Keamanan untuk Pod, termasuk prasyarat penerapan.



Kapasitas antarmuka cabang adalah tambahan untuk batas tipe instans yang ada untuk alamat IP sekunder. Pod yang menggunakan grup keamanan tidak diperhitungkan dalam rumus max-pod dan ketika Anda menggunakan grup keamanan untuk pod, Anda perlu mempertimbangkan untuk

meningkatkan nilai max-pod atau baik-baik saja dengan menjalankan pod lebih sedikit daripada yang sebenarnya dapat didukung oleh node.

Sebuah m5.large dapat memiliki hingga 9 antarmuka jaringan cabang dan hingga 27 alamat IP sekunder yang ditetapkan ke antarmuka jaringan standarnya. Seperti yang ditunjukkan pada contoh di bawah ini, max-pod default untuk m5.large adalah 29, dan EKS menghitung Pod yang menggunakan grup keamanan terhadap Pod maksimum. Silakan lihat panduan pengguna EKS untuk petunjuk tentang cara mengubah max-pod untuk node.

Ketika grup keamanan untuk Pod digunakan dalam kombinasi dengan jaringan kustom, grup keamanan yang ditentukan dalam grup keamanan untuk Pod digunakan, bukan grup keamanan yang ditentukan dalam ENIConfig. Akibatnya, ketika jaringan kustom diaktifkan, hati-hati menilai urutan grup keamanan saat menggunakan grup keamanan per Pod.

Rekomendasi

Nonaktifkan TCP Early Demux untuk Liveness Probe

Jika Anda menggunakan liveness atau readiness probe, Anda juga perlu menonaktifkan TCP early demux, sehingga kubelet dapat terhubung ke Pod pada antarmuka jaringan cabang melalui TCP. Ini hanya diperlukan dalam mode ketat. Untuk melakukan ini jalankan perintah berikut:

```
kubectl edit daemonset aws-node -n kube-system
```

Di bawah `initContainer` bagian ini, ubah nilainya `DISABLE_TCP_EARLY_DEMUX` menjadi `true`.

Gunakan Grup Keamanan Untuk Pod untuk memanfaatkan investasi konfigurasi AWS yang ada.

Grup keamanan memudahkan untuk membatasi akses jaringan ke sumber daya VPC, seperti database RDS atau instans EC2. Salah satu keuntungan yang jelas dari grup keamanan per Pod adalah kesempatan untuk menggunakan kembali sumber daya grup keamanan AWS yang ada. Jika Anda menggunakan grup keamanan sebagai firewall jaringan untuk membatasi akses ke layanan AWS Anda, kami mengusulkan untuk menerapkan grup keamanan ke Pod menggunakan ENI cabang. Pertimbangkan untuk menggunakan grup keamanan untuk Pod jika Anda mentransfer aplikasi dari instans EC2 ke EKS dan membatasi akses ke layanan AWS lain dengan grup keamanan.

Konfigurasikan Mode Penegakan Grup Keamanan Pod

Plugin Amazon VPC CNI versi 1.11 menambahkan pengaturan baru bernama `POD_SECURITY_GROUP_ENFORCING_MODE` (“mode penegakan”). Mode penegakan mengontrol grup keamanan mana yang diterapkan ke pod, dan jika sumber NAT diaktifkan. Anda dapat menentukan mode penegakan sebagai ketat atau standar. Strict adalah default, mencerminkan perilaku sebelumnya dari VPC CNI `ENABLE_POD_ENI` dengan disetel ke. `true`

Dalam Mode Ketat, hanya kelompok keamanan ENI cabang yang diberlakukan. Sumber NAT juga dinonaktifkan.

Dalam Mode Standar, grup keamanan yang terkait dengan ENI primer dan ENI cabang (terkait dengan pod) diterapkan. Lalu lintas jaringan harus mematuhi kedua kelompok keamanan.

> **Warning**
>
> Perubahan mode apa pun hanya akan memengaruhi Pod yang baru diluncurkan. Pod yang ada akan menggunakan mode yang dikonfigurasi saat Pod dibuat. Pelanggan perlu mendaur ulang Pod yang ada dengan grup keamanan jika mereka ingin mengubah perilaku lalu lintas.

Mode Penegakan: Gunakan mode Ketat untuk mengisolasi lalu lintas pod dan node:

Secara default, grup keamanan untuk Pod diatur ke “mode ketat.” Gunakan pengaturan ini jika Anda harus benar-benar memisahkan lalu lintas Pod dari sisa lalu lintas node. Dalam mode ketat, sumber NAT dimatikan sehingga grup keamanan keluar ENI cabang dapat digunakan.

> **Warning**
>
> Ketika mode ketat diaktifkan, semua lalu lintas keluar dari pod akan meninggalkan node dan masuk ke jaringan VPC. Lalu lintas antar pod pada node yang sama akan melewati VPC. Ini meningkatkan lalu lintas VPC dan membatasi fitur berbasis node. NodeLocal DNSCache Tidak didukung dengan mode ketat.

Mode Penegakan: Gunakan mode Standar dalam situasi berikut

IP sumber klien terlihat oleh kontainer di Pod

Jika Anda perlu menjaga IP sumber klien tetap terlihat oleh kontainer di dalam Pod, pertimbangkan `POD_SECURITY_GROUP_ENFORCING_MODE` untuk menyetelnya`standard`. Dukungan layanan Kubernetes externalTrafficPolicy =local untuk mendukung pelestarian IP sumber klien (cluster tipe default). Anda sekarang dapat menjalankan layanan Kubernetes dari tipe NodePort dan LoadBalancer menggunakan target instance dengan externalTrafficPolicy set ke Local dalam mode standar. `Local`mempertahankan IP sumber klien dan menghindari lompatan kedua untuk LoadBalancer dan NodePort mengetik Layanan.

Menyebarkan NodeLocal DNSCache

Saat menggunakan grup keamanan untuk Pod, konfigurasikan mode standar untuk mendukung Pod yang digunakan NodeLocal DNSCache. NodeLocal DNSCache meningkatkan kinerja DNS Cluster dengan menjalankan agen caching DNS pada node cluster sebagai file. DaemonSet Ini akan membantu pod yang memiliki persyaratan QPS DNS tertinggi untuk menanyakan Kube-DNS/CoreDNS lokal yang memiliki cache lokal, yang akan meningkatkan latensi.

NodeLocal DNSCache tidak didukung dalam mode ketat karena semua lalu lintas jaringan, bahkan ke node, memasuki VPC.

Mendukung Kebijakan Jaringan Kubernetes

Sebaiknya gunakan mode penegakan standar saat menggunakan kebijakan jaringan dengan Pod yang memiliki grup keamanan terkait.

Kami sangat menyarankan untuk menggunakan grup keamanan untuk Pod untuk membatasi akses tingkat jaringan ke layanan AWS yang bukan merupakan bagian dari klaster. Pertimbangkan kebijakan jaringan untuk membatasi lalu lintas jaringan antara Pod di dalam klaster, yang sering dikenal sebagai East/West lalu lintas.

Identifikasi Ketidakcocokan dengan Grup Keamanan per Pod

Instance berbasis Windows dan non-nitro tidak mendukung grup keamanan untuk Pod. Untuk memanfaatkan grup keamanan dengan Pod, instance harus diberi tag. isTrunkingEnabled Gunakan kebijakan jaringan untuk mengelola akses antar Pod dan bukan grup keamanan jika Pod Anda tidak bergantung pada layanan AWS apa pun di dalam atau di luar VPC Anda.

Gunakan Grup Keamanan per Pod untuk mengontrol lalu lintas ke AWS Services secara efisien

Jika aplikasi yang berjalan di dalam kluster EKS harus berkomunikasi dengan sumber daya lain di dalam VPC, misalnya database RDS, maka pertimbangkan untuk menggunakan SGs Pod. Meskipun

ada mesin kebijakan yang memungkinkan Anda menentukan CIDR atau nama DNS, mereka adalah pilihan yang kurang optimal saat berkomunikasi dengan layanan AWS yang memiliki titik akhir yang berada dalam VPC.

Sebaliknya, kebijakan jaringan Kubernetes menyediakan mekanisme untuk mengendalikan lalu lintas masuk dan keluar baik di dalam maupun di luar klaster. Kebijakan jaringan Kubernetes harus dipertimbangkan jika aplikasi Anda memiliki ketergantungan terbatas pada layanan AWS lainnya. Anda dapat mengonfigurasi kebijakan jaringan yang menentukan aturan keluar berdasarkan rentang CIDR untuk membatasi akses ke layanan AWS sebagai lawan dari semantik asli AWS. SGs Anda dapat menggunakan kebijakan jaringan Kubernetes untuk mengontrol lalu lintas jaringan antara Pod (sering disebut sebagai East/West lalu lintas) dan antara Pod dan layanan eksternal. Kebijakan jaringan Kubernetes diimplementasikan pada level OSI 3 dan 4.

Amazon EKS memungkinkan Anda untuk menggunakan mesin kebijakan jaringan seperti Calico dan Cilium. Secara default, mesin kebijakan jaringan tidak diinstal. Silakan periksa panduan pemasangan masing-masing untuk petunjuk tentang cara mengatur. Untuk informasi selengkapnya tentang cara menggunakan kebijakan jaringan, lihat Praktik terbaik EKS Security. Fitur nama host DNS tersedia di engine kebijakan jaringan versi perusahaan, yang dapat berguna untuk mengontrol lalu lintas antara Kubernetes Services/Pods dan sumber daya yang berjalan di luar AWS. Selain itu, Anda dapat mempertimbangkan dukungan nama host DNS untuk layanan AWS yang tidak mendukung grup keamanan secara default.

Tandai satu Grup Keamanan untuk menggunakan AWS Loadbalancer Controller

Ketika banyak grup keamanan dialokasikan ke Pod, Amazon EKS merekomendasikan untuk menandai satu grup keamanan dengan `kubernetes.io/cluster/$name`dibagikan atau dimiliki. Tag ini memungkinkan AWS Loadbalancer Controller untuk memperbarui aturan grup keamanan untuk merutekan lalu lintas ke Pod. Jika hanya satu grup keamanan yang diberikan ke sebuah Pod, penetapan tag bersifat opsional. Izin yang ditetapkan dalam grup keamanan bersifat aditif, oleh karena itu menandai satu grup keamanan cukup bagi pengontrol loadbalancer untuk menemukan dan merekonsiliasi aturan. Ini juga membantu untuk mematuhi kuota default yang ditentukan oleh grup keamanan.

Konfigurasikan NAT untuk Lalu Lintas Keluar

Sumber NAT dinonaktifkan untuk lalu lintas keluar dari Pod yang ditetapkan grup keamanan. Untuk Pod yang menggunakan grup keamanan yang memerlukan akses node pekerja peluncuran internet pada subnet pribadi yang dikonfigurasi dengan gateway atau instance NAT dan mengaktifkan SNAT eksternal di CNI.

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

Menerapkan Pod dengan Grup Keamanan ke Subnet Pribadi

Pod yang diberi grup keamanan harus dijalankan pada node yang di-deploy ke subnet pribadi. Perhatikan bahwa Pod dengan grup keamanan yang ditetapkan yang disebarkan ke subnet publik tidak akan dapat mengakses internet.

Verifikasi terminationGracePeriodDetik dalam File Spesifikasi Pod

Pastikan itu `terminationGracePeriodSeconds` bukan nol dalam file spesifikasi Pod Anda (default 30 detik). Hal ini penting agar Amazon VPC CNI dapat menghapus jaringan Pod dari node pekerja. Ketika disetel ke nol, plugin CNI tidak menghapus jaringan Pod dari host, dan cabang ENI tidak dibersihkan secara efektif.

Menggunakan Grup Keamanan untuk Pod dengan Fargate

Grup keamanan untuk Pod yang berjalan di Fargate bekerja sangat mirip dengan Pod yang berjalan pada node pekerja EC2. Misalnya, Anda harus membuat grup keamanan sebelum mereferensikannya dalam asosiasi SecurityGroupPolicy Anda dengan Pod Fargate Anda. Secara default, grup keamanan klaster ditetapkan ke semua Pod Fargate ketika Anda tidak secara eksplisit menetapkan a ke Pod Fargate. SecurityGroupPolicy Demi kesederhanaan, Anda mungkin ingin menambahkan grup keamanan klaster ke Pod Fagate SecurityGroupPolicy jika tidak, Anda harus menambahkan aturan grup keamanan minimum ke grup keamanan Anda. Anda dapat menemukan grup keamanan klaster menggunakan API klaster deskripsikan.

```
 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
```

Aturan grup keamanan minimum tercantum di sini. Aturan ini memungkinkan Pod Fargate untuk berkomunikasi dengan layanan in-cluster seperti kube-apiserver, kubelet, dan CoreDNS. Anda juga perlu menambahkan aturan untuk mengizinkan koneksi masuk dan keluar ke dan dari Fargate Pod Anda. Ini akan memungkinkan Pod Anda untuk berkomunikasi dengan Pod atau sumber daya lain di VPC Anda. Selain itu, Anda harus menyertakan aturan untuk Fargate untuk menarik gambar kontainer dari Amazon ECR atau pendaftar kontainer lainnya seperti. DockerHub Untuk informasi selengkapnya, lihat rentang alamat IP AWS di AWS General Reference.

Anda dapat menggunakan perintah di bawah ini untuk menemukan grup keamanan yang diterapkan pada Pod Fargate.

```
kubectl get pod FARGATE_POD -o jsonpath='{.metadata.annotations.fargate\.amazonaws
\.com/pod-sg}{"\n"}'
```

Catat eNIID dari perintah di atas.

```
aws ec2 describe-network-interfaces --network-interface-ids ENI_ID --query
 'NetworkInterfaces[*].Groups[*]'
```

Pod Fargate yang ada harus dihapus dan dibuat ulang agar grup keamanan baru diterapkan. Misalnya, perintah berikut memulai penerapan example-app. Untuk memperbarui pod tertentu, Anda dapat mengubah namespace dan nama deployment pada perintah di bawah ini.

```
kubectl rollout restart -n example-ns deployment example-pod
```

Penyeimbangan Beban

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Load Balancer menerima lalu lintas masuk dan mendistribusikannya ke seluruh target aplikasi yang dituju yang di-host di EKS Cluster. Ini meningkatkan ketahanan aplikasi. Saat diterapkan di Kluster

EKS, pengontrol AWS Load Balancer akan membuat dan mengelola AWS Elastic Load Balancers untuk cluster tersebut. Ketika tipe LoadBalancer Layanan Kubernetes dibuat, AWS Load Balancer Controller akan membuat Network Load Balancer (NLB) yang memuat saldo menerima lalu lintas di Layer 4 model OSI. Saat objek Kubernetes Ingress dibuat, AWS Load Balancer Controller membuat Application Load Balancer (ALB) yang memuat lalu lintas di Layer 7 model OSI.

Memilih Jenis Load Balancer

Portofolio AWS Elastic Load Balancing (ELB) mendukung penyeimbang beban berikut: Application Load Balancers (ALB), Network Load Balancers (NLB), Gateway Load Balancers (GWLB), dan Classic Load Balancers (CLB). Bagian praktik terbaik ini akan fokus pada ALB dan NLB yang merupakan dua yang paling relevan untuk Kluster EKS.

Pertimbangan utama dalam memilih jenis load balancer adalah persyaratan beban kerja.

Untuk informasi selengkapnya dan sebagai referensi untuk semua penyeimbang AWS Load, lihat Perbandingan Produk

Pilih Application Load Balancer (ALB) jika beban kerja Anda adalah HTTP/HTTPS

Jika beban kerja memerlukan penyeimbangan beban pada Layer 7 Model OSI, AWS Load Balancer Controller dapat digunakan untuk menyediakan ALB; kami membahas penyediaan di bagian berikut. ALB dikontrol dan dikonfigurasi oleh sumber daya Ingress yang disebutkan sebelumnya dan merutekan lalu lintas HTTP atau HTTPS ke Pod yang berbeda di dalam klaster. ALB menyediakan pelanggan dengan fleksibilitas untuk mengubah algoritma routing lalu lintas aplikasi; algoritma routing default adalah round robin dengan algoritma routing permintaan yang paling tidak menonjol juga alternatif.

Pilih Network Load Balancer (NLB) jika beban kerja Anda adalah TCP, atau jika beban kerja Anda memerlukan Source IP Preservation of Clients

Network Load Balancer berfungsi pada lapisan keempat (Transport) dari model Open Systems Interconnection (OSI). Ini cocok untuk beban kerja berbasis TCP & UDP. Network Load Balancer juga secara default mempertahankan IP Sumber alamat klien saat menyajikan lalu lintas ke pod.

Pilih Network Load Balancer (NLB) jika beban kerja Anda tidak dapat memanfaatkan DNS

Alasan utama lain untuk menggunakan NLB adalah jika klien Anda tidak dapat menggunakan DNS. Dalam hal ini, NLB mungkin lebih cocok untuk beban kerja Anda karena IPs pada Network Load

Balancer bersifat statis. Meskipun klien disarankan untuk menggunakan DNS saat menyelesaikan Nama Domain ke Alamat IP saat menghubungkan ke Load Balancer, jika aplikasi klien tidak mendukung resolusi DNS dan hanya menerima kode keras IPs maka NLB lebih cocok karena statis dan tetap sama untuk masa IPs pakai NLB.

Penyeimbang Beban Penyedia

Setelah menentukan Load Balancer yang paling cocok untuk beban kerja Anda, pelanggan memiliki sejumlah opsi untuk menyediakan penyeimbang beban.

Menyediakan Load Balancer dengan menerapkan AWS Load Balancer Controller

Ada dua metode utama penyediaan penyeimbang beban dalam Kluster EKS.

- Memanfaatkan Pengontrol Layanan di AWS Cloud Provider (lama)
- Memanfaatkan AWS Load Balancer Controller (disarankan)

Secara default, Kubernetes Service Controller, juga dikenal sebagai in-tree controller, merekonsiliasi jenis sumber daya Layanan Kubernetes. LoadBalancer Kontroler ini dibangun ke dalam komponen AWS Cloud Provider yang berfungsi sebagai Kubernetes Cloud Controller Manager.

Konfigurasi Elastic Load Balancer yang disediakan dikendalikan oleh anotasi yang harus ditambahkan ke manifes Layanan Kubernetes. Anotasi yang digunakan oleh Service Controller dan AWS Load Balancer Controller berbeda.

Pengontrol Layanan adalah warisan dan saat ini hanya menerima perbaikan bug kritis. Saat Anda membuat tipe Layanan Kubernetes LoadBalancer, Service Controller membuat AWS CLB secara default, tetapi juga dapat membuat AWS NLB jika Anda menggunakan anotasi yang tepat. Perlu dicatat bahwa Service Controller tidak mendukung sumber daya Kubernetes Ingress dan juga tidak mendukung. IPv6

Sebaiknya gunakan AWS Load Balancer Controller di kluster EKS Anda untuk mendamaikan resource Layanan Kubernetes dan Ingress. Anda harus menggunakan anotasi yang tepat di Layanan Kubernetes atau manifes Ingress agar AWS Load Balancer Controller memiliki proses rekonsiliasi. (bukan Service Controller)

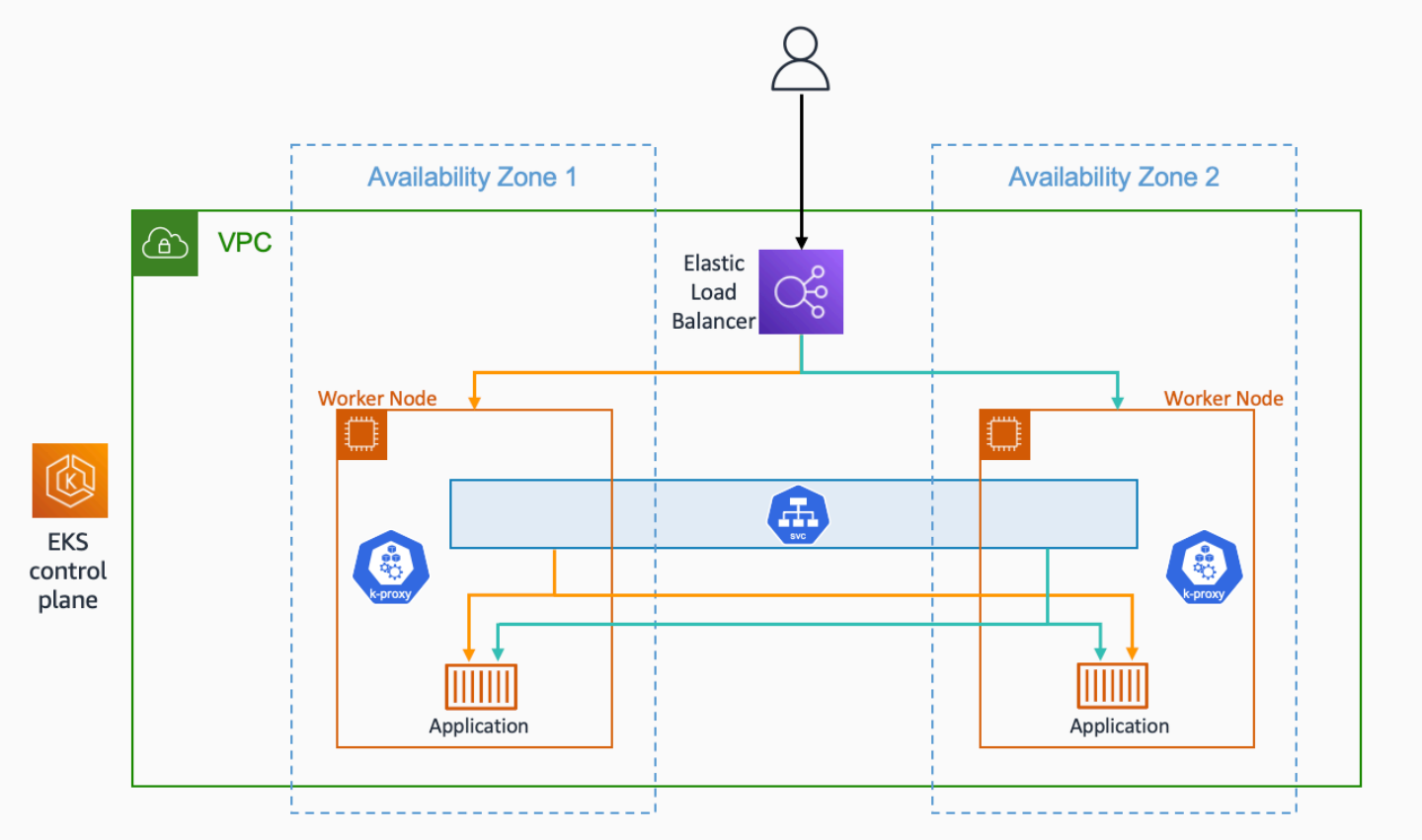
Jika Anda menggunakan Mode Otomatis EKS, AWS Load Balancer Controller disediakan untuk Anda secara otomatis; tidak diperlukan instalasi.

Memilih Jenis Target Load Balancer

Daftarkan Pod sebagai target menggunakan IP Target-Type

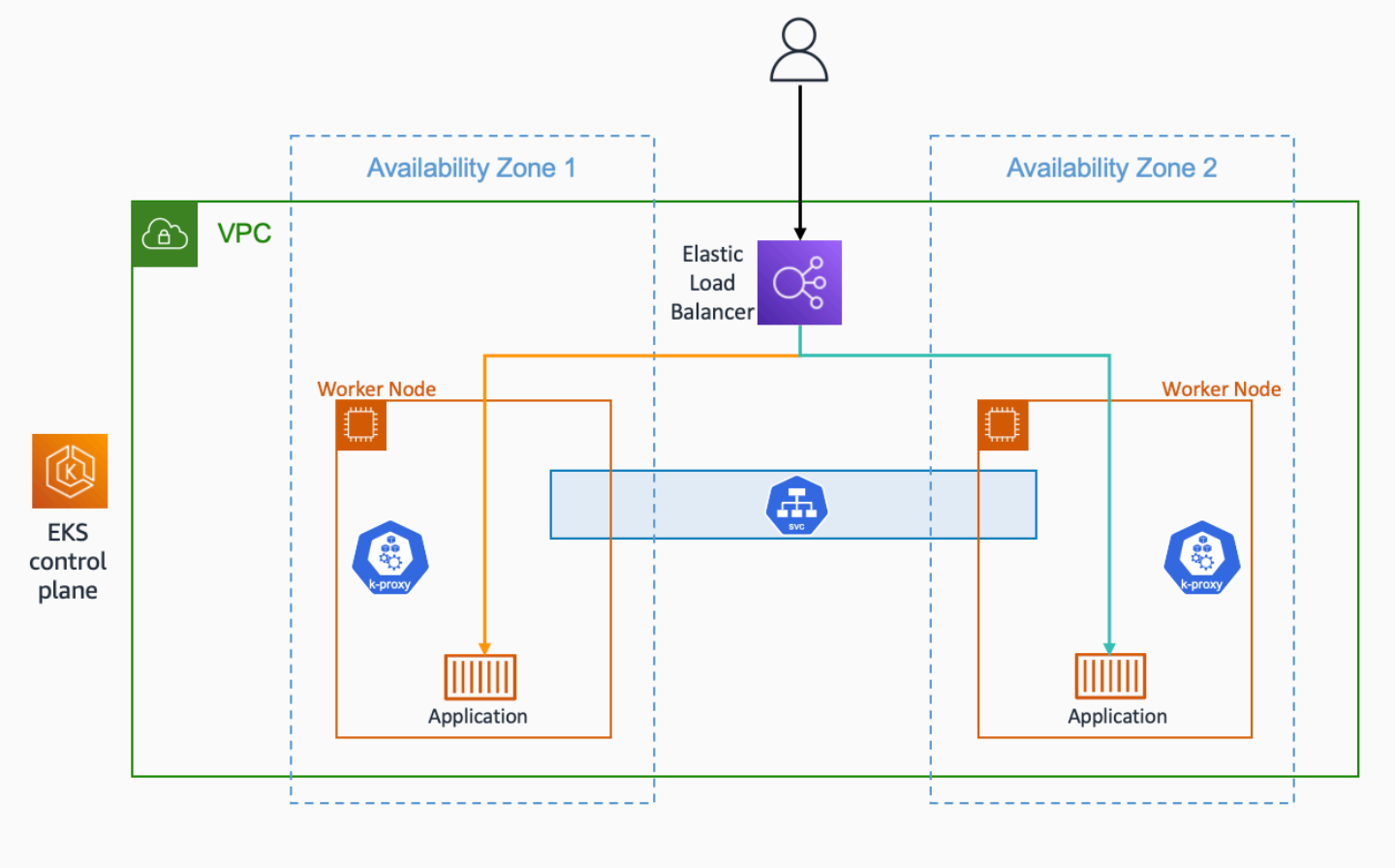
AWS Elastic Load Balancer: Jaringan & Aplikasi, mengirimkan lalu lintas yang diterima ke target terdaftar dalam grup target. Untuk Kluster EKS ada 2 jenis target yang dapat Anda daftarkan di grup target: Instance & IP, tipe target mana yang digunakan memiliki implikasi pada apa yang didaftarkan dan bagaimana lalu lintas dirutekan dari Load Balancer ke pod. Secara default, pengontrol AWS Load Balancer akan mendaftarkan target menggunakan tipe “Instance” dan target ini akan menjadi IP Worker Node dan NodePort implikasinya meliputi:

- Lalu lintas dari Load Balancer akan diteruskan ke Worker Node pada NodePort, ini diproses oleh aturan iptables (dikonfigurasi oleh kube-proxy yang berjalan pada node), dan diteruskan ke Service pada ClusterIp-nya (masih di node), akhirnya Service secara acak memilih pod yang terdaftar padanya dan meneruskan lalu lintas ke sana. Alur ini melibatkan beberapa hop dan latensi tambahan dapat terjadi terutama karena Layanan terkadang akan memilih pod yang berjalan pada node pekerja lain yang mungkin juga berada di AZ lain.
- Karena Load Balancer mendaftarkan Worker Node sebagai targetnya, ini berarti pemeriksaan kesehatan yang dikirim ke target tidak akan langsung diterima oleh pod tetapi oleh Worker Node pada lalu lintas pemeriksaan kesehatan akan mengikuti jalur yang sama seperti yang dijelaskan di atas. NodePort
- Pemantauan dan Pemecahan Masalah lebih kompleks karena lalu lintas yang diteruskan oleh Load Balancer tidak langsung dikirim ke pod dan Anda harus hati-hati mengkorelasikan paket yang diterima di Worker Node ke Service ClusterIP dan akhirnya pod memiliki visibilitas penuh ke jalur paket untuk pemecahan masalah yang tepat. end-to-end



Sebaliknya jika Anda mengonfigurasi tipe target sebagai “IP” seperti yang kami sarankan implikasinya adalah sebagai berikut:

- Lalu lintas dari Load Balancer akan diteruskan langsung ke pod, ini menyederhanakan jalur jaringan karena melewati lompatan ekstra sebelumnya dari Worker Nodes dan Service Cluster IP, ini mengurangi latensi yang seharusnya terjadi jika Service meneruskan lalu lintas ke pod di AZ lain dan terakhir menghapus pemrosesan overhead aturan iptables pada Worker Nodes.
- Pemeriksaan kesehatan Load Balancer langsung diterima dan ditanggapi oleh pod, ini berarti status target “sehat” atau “tidak sehat” adalah representasi langsung dari status kesehatan pod.
- Pemantauan dan Pemecahan Masalah lebih mudah dan alat apa pun yang digunakan untuk menangkap alamat IP paket akan secara langsung mengungkapkan lalu lintas dua arah antara Load Balancer dan pod di bidang sumber dan tujuannya.



Untuk membuat AWS Elastic Load Balancing yang menggunakan Target IP, Anda menambahkan:

- `alb.ingress.kubernetes.io/target-type: ip`anotasi ke manifes Ingress Anda saat mengonfigurasi Kubernetes Ingress (Application Load Balancer)
- `service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip`anotasi ke Manifes Layanan Anda saat mengonfigurasi tipe Layanan Kubernetes Anda (Network Load LoadBalancer Balancer).

Mengkonfigurasi Pemeriksaan Kesehatan Load Balancer

Meskipun Kubernetes menyediakan mekanisme pemeriksaan kesehatannya sendiri (dirinci di bagian selanjutnya), kami merekomendasikan untuk menerapkan pemeriksaan kesehatan ELB sebagai perlindungan pelengkap yang berfungsi di luar bidang kendali Kubernetes. Lapisan independen ini terus memantau aplikasi Anda bahkan selama:

- Kubernetes mengontrol gangguan pesawat
- Penundaan eksekusi probe

- Partisi jaringan antara kubelet dan pod

Untuk beban kerja kritis yang membutuhkan pemulihan akselerasi ketersediaan maksimum selama skenario yang disebutkan di atas, pemeriksaan kesehatan ELB menyediakan jaring pengaman penting yang bekerja bersama — bukan menggantikan — mekanisme asli Kubernetes.

Untuk mengonfigurasi dan menyempurnakan pemeriksaan kesehatan pada ELB Anda, Anda harus menggunakan anotasi di Kubernetes Service atau manifes Ingress yang akan direkonsiliasi oleh Service Controller atau AWS Load Balancer Controller.

Ketersediaan dan Siklus Hidup Pod

Selama upgrade aplikasi Anda harus memastikan bahwa aplikasi Anda selalu tersedia untuk memproses permintaan sehingga pengguna tidak mengalami downtime apapun. Salah satu tantangan umum dalam skenario ini adalah menyinkronkan status ketersediaan beban kerja Anda antara lapisan Kubernetes, dan infrastruktur, misalnya Load Balancer eksternal. Beberapa bagian berikutnya menyoroti praktik terbaik untuk mengatasi skenario tersebut.

> **Note**
>
> Penjelasan di bawah ini didasarkan pada EndpointSliceskarena merupakan pengganti yang direkomendasikan untuk Endpoint di Kubernetes. Perbedaan antara keduanya dapat diabaikan dalam konteks skenario yang dibahas di bawah ini. AWS Load Balancer Controller secara default menggunakan Endpoint, Anda dapat mengaktifkan EndpointSlices dengan mengaktifkan pada pengontrol. enable-endpoint-sliceflag

Gunakan cek kesehatan

Kubernetes secara default menjalankan pemeriksaan kesehatan proses di mana proses kubelet pada node memverifikasi apakah proses utama dari container sedang berjalan atau tidak. Jika tidak maka secara default itu memulai ulang wadah itu. Namun Anda juga dapat mengonfigurasi probe Kubernetes untuk mengidentifikasi kapan proses kontainer berjalan tetapi dalam keadaan kebuntuan, atau apakah aplikasi telah berhasil atau tidak. Probe dapat didasarkan pada mekanisme exec, grpc, HttpGet dan TCPSocket. Berdasarkan jenis dan hasil probe wadah dapat dimulai ulang.

Silakan lihat Pembuatan Pod di bagian Lampiran di bawah ini untuk meninjau kembali urutan kejadian dalam proses pembuatan Pod.

Gunakan probe kesiapan

Secara default ketika semua kontainer dalam sebuah Pod menjalankan kondisi Pod dianggap "Siap". Namun aplikasi mungkin masih belum dapat memproses permintaan klien. Misalnya aplikasi mungkin perlu menarik beberapa data atau konfigurasi dari sumber daya eksternal untuk dapat memproses permintaan. Dalam keadaan seperti itu Anda tidak ingin mematikan aplikasi atau meneruskan permintaan apa pun ke sana. Readiness probe memungkinkan Anda untuk memastikan bahwa Pod tidak dianggap "Ready", yang berarti bahwa Pod tidak akan ditambahkan ke EndpointSlice objek, sampai hasil probe selesai`success`. Di sisi lain jika probe gagal lebih jauh ke bawah garis maka Pod dihapus dari EndpointSlice objek. Anda dapat mengonfigurasi probe kesiapan dalam manifes Pod untuk setiap kontainer. `kubelet`proses pada setiap node menjalankan probe kesiapan terhadap kontainer pada node itu.

Memanfaatkan gerbang kesiapan Pod

Salah satu aspek dari probe kesiapan adalah kenyataan bahwa tidak ada mekanisme umpan balik/pengaruh eksternal di dalamnya, proses kubelet pada node mengeksekusi probe dan mendefinisikan status probe. Hal ini tidak berdampak pada permintaan antara layanan mikro itu sendiri di lapisan Kubernetes (lalu lintas timur barat) karena EndpointSlice Controller selalu memperbarui daftar endpoint (Pod). Mengapa dan kapan Anda membutuhkan mekanisme eksternal?

Ketika Anda mengekspos aplikasi Anda menggunakan Kubernetes Service type Load Balancer atau Kubernetes Ingress (untuk lalu lintas utara - selatan) maka daftar Pod IPs untuk masing-masing Layanan Kubernetes harus disebarkan ke load balancer infrastruktur eksternal sehingga load balancer juga memiliki daftar target yang up to date. AWS Load Balancer Controller menjembatani kesenjangan di sini. Saat Anda menggunakan AWS Load Balancer Controller dan leverage`target group: IP`, seperti `kube-proxy` AWS Load Balancer Controller juga menerima pembaruan (`watch`via) dan kemudian berkomunikasi dengan ELB API untuk mengonfigurasi dan mulai mendaftarkan IP Pod sebagai target di ELB.

Saat Anda melakukan pembaruan bergulir dari Deployment, Pod baru akan dibuat, dan segera setelah kondisi Pod baru "Siap", old/existing Pod akan dihentikan. Selama proses ini, EndpointSlice objek Kubernetes diperbarui lebih cepat daripada waktu yang dibutuhkan ELB untuk mendaftarkan Pod baru sebagai target, lihat registrasi target. Untuk waktu yang singkat Anda dapat memiliki ketidakcocokan status antara lapisan Kubernetes dan lapisan infrastruktur tempat permintaan klien dapat dihapus. Selama periode ini di dalam layer Kubernetes, Pod baru akan siap untuk memproses permintaan tetapi dari sudut pandang ELB mereka tidak.

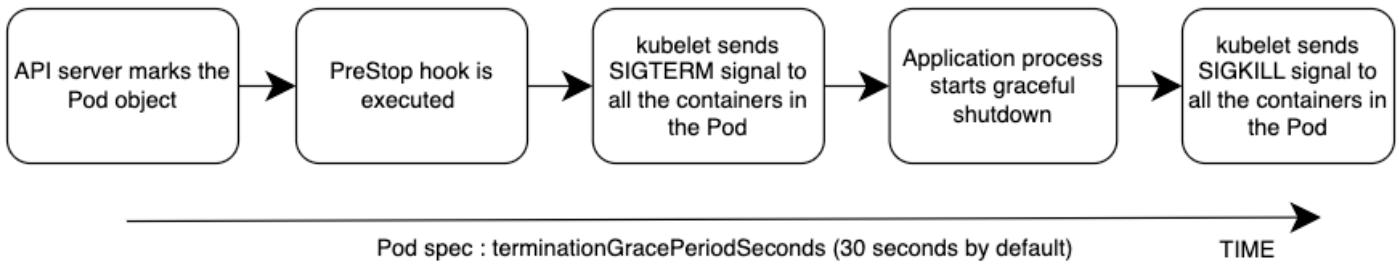
Pod Readiness Gates memungkinkan Anda untuk menentukan persyaratan tambahan yang harus dipenuhi sebelum kondisi Pod dianggap “Siap”. Dalam kasus AWS ELB, AWS Load Balancer Controller memantau status target (Pod) di AWS ELB dan setelah pendaftaran target selesai dan statusnya berubah menjadi “Sehat” maka pengontrol memperbarui kondisi Pod menjadi “Siap”. Dengan pendekatan ini, Anda memengaruhi kondisi Pod berdasarkan status jaringan eksternal, yang merupakan status target di AWS ELB. Pod Readiness Gates sangat penting dalam skenario pembaruan yang bergulir karena memungkinkan Anda mencegah pembaruan bergulir penerapan menghentikan pod lama hingga status target Pod yang baru dibuat berubah menjadi “Sehat” di AWS ELB.

Matikan aplikasi dengan anggun

Aplikasi Anda harus merespons sinyal SIGTERM dengan memulai shutdown yang anggun sehingga klien tidak mengalami downtime apa pun. Artinya, aplikasi Anda harus menjalankan prosedur pembersihan seperti menyimpan data, menutup deskriptor file, menutup koneksi database, menyelesaikan permintaan dalam penerbangan dengan anggun dan keluar tepat waktu untuk memenuhi permintaan penghentian Pod. Anda harus mengatur masa tenggang cukup lama sehingga pembersihan dapat selesai. Untuk mempelajari cara merespons sinyal SIGTERM, Anda dapat merujuk ke sumber daya dari masing-masing bahasa pemrograman yang Anda gunakan untuk aplikasi Anda.

Jika aplikasi Anda tidak dapat dimatikan dengan anggun setelah menerima sinyal SIGTERM atau jika mengabaikan/tidak menerima sinyal, maka Anda dapat memanfaatkan PreStophook untuk memulai shutdown aplikasi yang anggun. Prestop hook dijalankan segera sebelum sinyal SIGTERM dikirim dan dapat melakukan operasi arbitrer tanpa harus mengimplementasikan operasi tersebut dalam kode aplikasi itu sendiri.

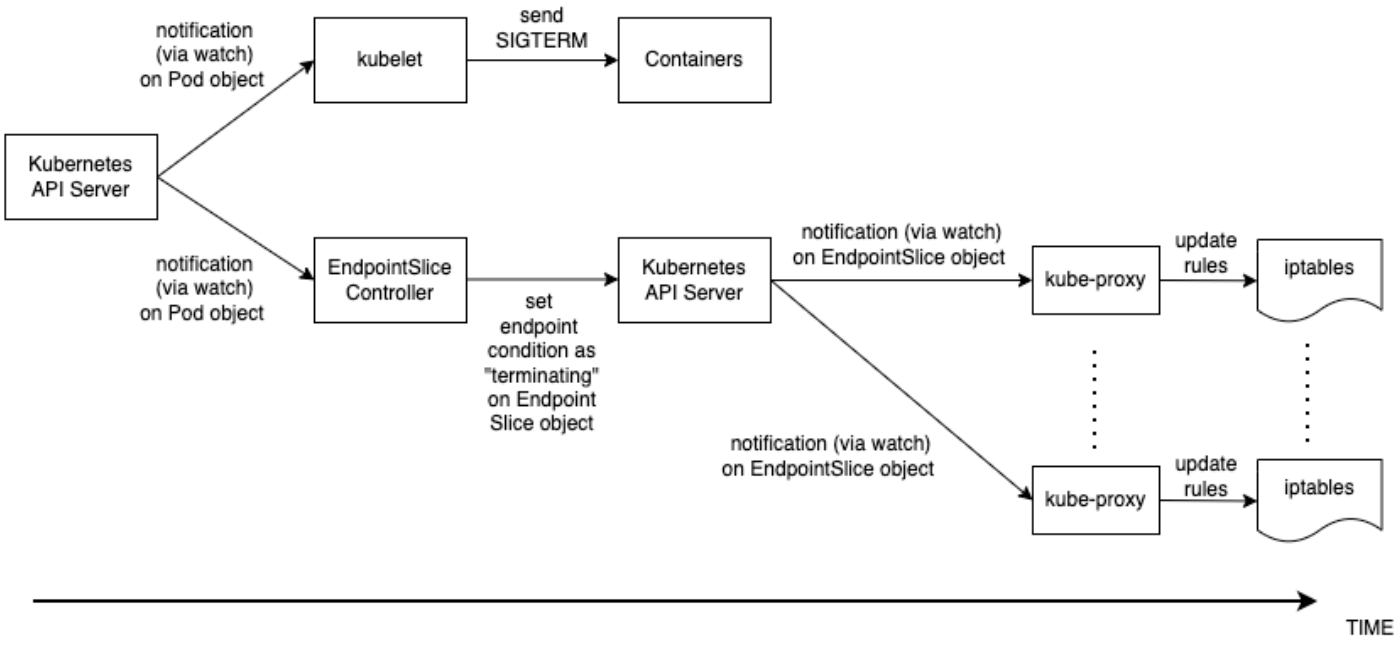
Urutan keseluruhan peristiwa ditunjukkan pada diagram di bawah ini. Catatan: terlepas dari hasil prosedur shutdown aplikasi yang anggun, atau hasil PreStop hook, wadah aplikasi akhirnya dihentikan pada akhir masa tenggang melalui SIGKILL.



Silakan lihat Penghapusan Pod di bagian Lampiran di bawah ini untuk meninjau kembali urutan kejadian dalam proses penghapusan Pod.

Menangani permintaan klien dengan anggun

Urutan kejadian dalam penghapusan Pod berbeda dengan pembuatan Pod. Ketika sebuah Pod dibuat akan `kubelet` memperbarui IP Pod di Kubernetes API dan baru kemudian EndpointSlice objek tersebut diperbarui. Di sisi lain ketika sebuah Pod sedang dihentikan, Kubernetes API akan memberitahukan kubelet dan controller secara bersamaan. EndpointSlice Hati-hati memeriksa diagram berikut yang menunjukkan urutan peristiwa.



Cara status menyebar sepanjang jalan dari server API hingga aturan iptables pada node yang dijelaskan di atas menciptakan kondisi balapan yang menarik. Karena ada kemungkinan besar bahwa container menerima sinyal SIGKILL jauh lebih awal daripada kube-proxy pada setiap node memperbarui aturan iptables lokal. Dalam peristiwa seperti itu, dua skenario yang layak disebutkan adalah:

- Jika aplikasi Anda segera dan terus terang membatalkan permintaan dan koneksi dalam penerbangan setelah menerima SIGTERM yang berarti klien akan melihat kesalahan 50x di semua tempat.
- Bahkan jika aplikasi Anda memastikan bahwa semua permintaan dan koneksi dalam penerbangan diproses sepenuhnya setelah menerima SIGTERM, selama masa tenggang, permintaan klien baru akan tetap dikirim ke wadah aplikasi karena aturan iptables mungkin masih belum diperbarui.

Sampai prosedur pembersihan menutup soket server pada wadah, permintaan baru tersebut akan menghasilkan koneksi baru. Ketika masa tenggang berakhir koneksi tersebut, yang dibuat setelah SIGTERM, pada saat itu dijatuhkan tanpa syarat sejak SIGKILL dikirim.

Menyetel masa tenggang dalam spesifikasi Pod cukup lama dapat mengatasi tantangan ini, tetapi tergantung pada penundaan propagasi dan jumlah permintaan klien yang sebenarnya, sulit untuk mengantisipasi waktu yang dibutuhkan aplikasi untuk menutup koneksi dengan anggun. Oleh karena itu pendekatan yang tidak begitu sempurna tetapi paling layak di sini adalah menggunakan PreStop kait untuk menunda sinyal SIGTERM hingga aturan iptables diperbarui untuk memastikan bahwa tidak ada permintaan klien baru yang dikirim ke aplikasi, hanya koneksi yang ada yang melanjutkan. PreStop hook bisa menjadi handler Exec sederhana seperti. `sleep 10`

Perilaku dan rekomendasi yang disebutkan di atas akan berlaku sama ketika Anda mengekspos aplikasi Anda menggunakan Load Balancer tipe Layanan Kubernetes atau Kubernetes Ingress (untuk lalu lintas utara - selatan) menggunakan AWS Load Balancer Controller dan leverage. `target group: IP` Karena seperti `kube-proxy` AWS Load Balancer Controller juga menerima pembaruan (via watch) pada EndpointSlice objek dan kemudian berkomunikasi dengan ELB API untuk mulai membatalkan pendaftaran IP Pod dari ELB. Namun tergantung pada pemuatan pada Kubernetes API atau ELB API, ini juga dapat memakan waktu dan SIGTERM mungkin sudah dikirim ke aplikasi sejak lama. Setelah ELB mulai membatalkan pendaftaran target, ia berhenti mengirim permintaan ke target tersebut sehingga aplikasi tidak akan menerima permintaan baru dan ELB juga memulai penundaan Deregistrasi yaitu 300 detik secara default. Selama proses deregistrasi, target adalah `draining` di mana pada dasarnya ELB menunggu koneksi dalam penerbangan ke target tersebut mengalir requests/existing . Setelah penundaan deregistrasi berakhir maka target tidak digunakan dan permintaan dalam penerbangan ke target tersebut secara paksa dibatalkan.

Gunakan anggaran gangguan Pod

Konfigurasikan Pod Disruption Budget (PDB) untuk aplikasi Anda. PDBlimits jumlah Pod dari aplikasi yang direplikasi yang turun secara bersamaan dari gangguan sukarela. Ini memastikan bahwa jumlah minimum atau persentase Pod tetap tersedia dalam sebuah StatefulSet atau Deployment. Misalnya, aplikasi berbasis kuorum perlu memastikan bahwa jumlah replika yang berjalan tidak pernah dibawa di bawah angka yang dibutuhkan untuk kuorum. Atau ujung depan web mungkin memastikan bahwa jumlah replika yang melayani beban tidak pernah turun di bawah persentase tertentu dari total. PDB akan melindungi aplikasi terhadap tindakan seperti node yang terkuras, atau versi baru Deployment yang diluncurkan. Perlu diingat bahwa PDB tidak akan melindungi aplikasi terhadap gangguan yang tidak disengaja seperti kegagalan sistem operasi node atau hilangnya konektivitas jaringan. Untuk

informasi lebih lanjut, silakan lihat dokumentasi Menentukan Anggaran Gangguan untuk Aplikasi Anda di Kubernetes.

Referensi

- KubeCon Sesi Eropa 2019 - Siap? Menyelam Jauh ke Pod Readiness Gates untuk Service Health
- Buku - Kubernetes in Action
- AWS Blog - Cara cepat menskalakan aplikasi Anda dengan ALB di EKS (tanpa kehilangan lalu lintas)

Lampiran

Pembuatan Pod

Sangat penting untuk memahami apa urutan peristiwa dalam skenario di mana sebuah Pod di-deploy dan kemudian menjadi healthy/ready untuk menerima dan memproses permintaan klien. Mari kita bicara tentang urutan peristiwa.

1. Sebuah Pod dibuat pada control plane Kubernetes (yaitu dengan perintah kubectl, atau Deployment update, atau scaling action).
2. `kube-scheduler`menugaskan Pod ke sebuah node di cluster.
3. Proses kubelet yang berjalan pada node yang ditetapkan menerima update (via`watch`) dan berkomunikasi dengan runtime container untuk memulai container yang ditentukan dalam spesifikasi Pod.
4. Ketika container mulai berjalan, kubelet akan memperbarui kondisi Pod seperti `Ready` pada objek Pod di Kubernetes API.
5. EndpointSliceController menerima update kondisi Pod (via`watch`) dan menambahkan Pod IP/Port sebagai endpoint baru ke EndpointSliceobjek (list of Pod IPs) dari masing-masing Layanan Kubernetes.
6. proses kube-proxy pada setiap node menerima pembaruan (via`watch`) pada EndpointSlice objek dan kemudian memperbarui aturan iptables pada setiap node, dengan IP/port Pod baru.

Penghapusan Pod

Sama seperti pembuatan Pod, sangat penting untuk memahami apa urutan peristiwa selama penghapusan Pod. Mari kita bicara tentang urutan kejadian.

1. Permintaan penghapusan Pod dikirim ke server API Kubernetes (yaitu dengan `kubectl` perintah, atau pembaruan Deployment, atau tindakan penskalaan).
2. Server API Kubernetes memulai masa tenggang, yaitu 30 detik secara default, dengan menyetel field deletionTimeStamp pada objek Pod. (Masa tenggang dapat dikonfigurasi dalam spesifikasi Pod melalui`terminationGracePeriodSeconds`)
3. `kubelet`Proses yang berjalan pada node menerima update (via watch) pada objek Pod dan mengirimkan sinyal SIGTERM untuk memproses identifier 1 (PID 1) di dalam setiap kontainer dalam Pod tersebut. Kemudian menonton`terminationGracePeriodSeconds`.
4. EndpointSliceController juga menerima update (via`watch`) dari Langkah 2 dan menetapkan kondisi endpoint menjadi "terminating" pada EndpointSliceobjek (list of Pod IPs) dari masing-masing Layanan Kubernetes.
5. Proses kube-proxy pada setiap node menerima update (via`watch`) pada EndpointSlice objek kemudian aturan iptables pada setiap node diperbarui oleh kube-proxy untuk menghentikan penerusan permintaan klien ke Pod.
6. Ketika `terminationGracePeriodSeconds` kadaluarsa maka sinyal SIGKILL `kubelet` mengirimkan ke proses induk dari setiap kontainer di dalam Pod dan secara paksa menghentikannya.
7. TheEndpointSliceController menghapus endpoint dari EndpointSliceobjek.
8. Server API menghapus objek Pod.

Memantau beban kerja EKS untuk masalah kinerja Jaringan

Memantau lalu lintas CoreDNS untuk masalah pelambatan DNS

Menjalankan beban kerja intensif DNS terkadang dapat mengalami kegagalan CoreDNS intermiten karena pelambatan DNS, dan ini dapat memengaruhi aplikasi di mana Anda mungkin mengalami kesalahan sesekali. UnknownHostException

Deployment untuk CoreDNS memiliki kebijakan anti-afinitas yang menginstruksikan scheduler Kubernetes untuk menjalankan instance CoreDNS pada node pekerja terpisah di cluster, yaitu harus menghindari penempatan replika bersama pada node pekerja yang sama. Ini secara efektif mengurangi jumlah kueri DNS per antarmuka jaringan karena lalu lintas dari setiap replika dirutekan melalui ENI yang berbeda. Jika Anda melihat bahwa kueri DNS sedang dibatasi karena batas 1024 paket per detik, Anda dapat 1) mencoba meningkatkan jumlah replika CoreDNS atau 2) implementasi. NodeLocal DNSCache Lihat Monitor CoreDNS Metrics untuk informasi lebih lanjut.

Tantangan

- Packet drop terjadi dalam hitungan detik dan mungkin sulit bagi kita untuk memantau pola-pola ini dengan benar untuk menentukan apakah pelambatan DNS benar-benar terjadi.
- Kueri DNS dibatasi pada tingkat elastic network interface. Jadi, kueri yang dibatasi tidak muncul di pencatatan kueri.
- Log alur tidak menangkap semua lalu lintas IP. Misalnya Lalu lintas yang dihasilkan oleh instans ketika menghubungi server DNS Amazon. Jika Anda menggunakan server DNS Anda sendiri, maka semua lalu lintas ke server DNS tersebut dicatat

Solusi

Cara mudah untuk mengidentifikasi masalah pelambatan DNS di node pekerja adalah dengan menangkap metrik. `linklocal_allowance_exceeded` Linklocal_allowance_exceeded adalah jumlah paket yang dijatuhkan karena PPS lalu lintas ke layanan proxy lokal melebihi maksimum untuk antarmuka jaringan. Hal ini berdampak lalu lintas ke layanan DNS, Layanan Metadata Instans, dan Layanan Amazon Time Sync. Alih-alih melacak acara ini secara real-time, kami juga dapat mengalirkan metrik ini ke Amazon Managed Service untuk Prometheus dan dapat memvisualisasikannya di Grafana yang Dikelola Amazon

Memantau penundaan kueri DNS menggunakan metrik Conntrack

Metrik lain yang dapat membantu dalam memantau pembatasan CoreDNS/penundaan kueri adalah dan. `conntrack_allowance_available` `conntrack_allowance_exceeded` Kegagalan konektivitas yang disebabkan oleh melebihi tunjangan Koneksi yang Dilacak dapat memiliki dampak yang lebih besar daripada yang dihasilkan dari melebihi tunjangan lainnya. Saat mengandalkan TCP untuk mentransfer data, paket yang antri atau turun karena melebihi tunjangan jaringan instans EC2, seperti Bandwidth, PPS, dll., Biasanya ditangani dengan anggun berkat kemampuan kontrol kemacetan TCP. Aliran yang terkena dampak akan diperlambat, dan paket yang hilang akan ditransmisikan kembali. Namun, ketika sebuah instance melebihi tunjangan Connections Tracked, tidak ada koneksi baru yang dapat dibuat sampai beberapa yang sudah ada ditutup untuk memberi ruang bagi koneksi baru.

`conntrack_allowance_available`dan `conntrack_allowance_exceeded` membantu pelanggan dalam memantau tunjangan yang dilacak koneksi yang bervariasi untuk setiap contoh. Metrik kinerja jaringan ini memberi pelanggan visibilitas ke dalam jumlah paket yang diantri atau dijatuhkan ketika tunjangan jaringan instans, seperti Bandwidth Jaringan, Packets-Per-Second (PPS),

Connections Tracked, dan akses layanan link-lokal (Amazon DNS, Instance Meta Data Service, Amazon Time Sync) terlampaui

`conntrack_allowance_available`adalah jumlah koneksi yang dilacak yang dapat dibuat oleh instance sebelum mencapai tunjangan Connections Tracked dari jenis instance tersebut (hanya didukung untuk instance berbasis nitroo). `conntrack_allowance_exceeded`adalah jumlah paket yang dijatuhkan karena pelacakan koneksi melebihi maksimum untuk instance dan koneksi baru tidak dapat dibuat.

Metrik kinerja Jaringan penting lainnya

Metrik kinerja jaringan penting lainnya meliputi:

`bw_in_allowance_exceeded`(nilai ideal metrik harus nol) adalah jumlah paket yang diantrian and/or turun karena bandwidth agregat masuk melebihi maksimum untuk instance

`bw_out_allowance_exceeded`(nilai ideal metrik harus nol) adalah jumlah paket yang diantrian and/or turun karena bandwidth agregat keluar melebihi maksimum untuk instance

`pps_allowance_exceeded`(nilai ideal metrik harus nol) adalah jumlah paket yang diantrian and/or dijatuhkan karena PPS dua arah melebihi maksimum untuk contoh

Menangkap metrik untuk memantau beban kerja untuk masalah kinerja jaringan

Driver Elastic Network Adapter (ENA) menerbitkan metrik kinerja jaringan yang dibahas di atas dari contoh di mana mereka diaktifkan. Semua metrik kinerja jaringan dapat dipublikasikan untuk CloudWatch menggunakan CloudWatch agen. Silakan merujuk ke blog untuk informasi lebih lanjut.

Sekarang mari kita tangkap metrik yang dibahas di atas, simpan di Amazon Managed Service untuk Prometheus dan memvisualisasikan menggunakan Amazon Managed Grafana

Prasyarat

- ethtool - Pastikan node pekerja telah menginstal ethtool
- Ruang kerja AMP yang dikonfigurasi di akun AWS Anda. Untuk petunjuknya, lihat Membuat ruang kerja di Panduan Pengguna AMP.
- Ruang Kerja Grafana yang Dikelola Amazon

Menyebarkan Prometheus Node Exporter

Untuk mengumpulkan metrik ethtool untuk node pekerja, kita perlu menerapkan kolektor yang mengumpulkan metrik dan memaparkannya dalam format Prometheus. Perintah berikut akan menambahkan repositori `prometheus-community` Helm dan menginstal `prometheus-node-exporter` bagan dengan kolektor diaktifkan. `ethtool`

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm upgrade -i prometheus-node-exporter prometheus-community/prometheus-node-exporter
 \
  --set extraArgs[0]="--collector.ethtool" \
  --set extraArgs[1]="--collector.ethtool.device-include=(eth|em|eno|ens|enp)[0-9s]+" \
  --set extraArgs[2]="--collector.ethtool.metrics-include=.*" \
  --set podAnnotations."prometheus\.io/scrape='true'" \
  --set podAnnotations."prometheus\.io/port='9100'"
```

Terapkan kolektor ADOT untuk mengikis metrik ethtool dan simpan di Amazon Managed Service untuk ruang kerja Prometheus

Setiap cluster tempat Anda menginstal AWS Distro for OpenTelemetry (ADOT) harus memiliki peran ini untuk memberikan izin akun layanan AWS Anda untuk menyimpan metrik ke Amazon Managed Service for Prometheus. Ikuti langkah-langkah ini untuk membuat dan mengaitkan peran IAM Anda ke akun layanan Amazon EKS Anda menggunakan 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
```

Mari gunakan kolektor ADOT untuk mengikis metrcis dari eksportir prometheus ethtool dan menyimpannya di Amazon Managed Service untuk Prometheus

Prosedur berikut menggunakan contoh file YAMAL dengan deployment sebagai nilai mode. Ini adalah mode default dan menyebarkan ADOT Collector mirip dengan aplikasi mandiri. Konfigurasi ini menerima metrik OTLP dari aplikasi sampel dan Amazon Managed Service untuk metrik Prometheus yang tergores dari pod di klaster

```
curl -o collector-config-amp.yaml https://raw.githubusercontent.com/aws-observability/
aws-otel-community/master/sample-configs/operator/collector-config-amp.yaml
```

Di collector-config-amp .yaml, ganti yang berikut ini dengan nilai Anda sendiri:

- modus: penyebaran
- ServiceAccount: adot-collector
- titik akhir: <YOUR_REMOTE_WRITE_ENDPOINT>
- wilayah: <YOUR_ >AWS_REGION
- nama: adot-kolektor

```
kubectl apply -f collector-config-amp.yaml
```

Setelah kolektor adot digunakan, metrik akan berhasil disimpan di Amazon Prometheus

Konfigurasikan pengelola peringatan di Amazon Managed Service untuk Prometheus untuk mengirim notifikasi

Anda dapat menggunakan pengelola peringatan di Amazon Managed Service untuk Prometheus untuk menyiapkan aturan peringatan untuk peringatan penting, lalu Anda dapat mengirim pemberitahuan ke topik Amazon SNS. Mari konfigurasikan aturan perekaman dan aturan peringatan untuk memeriksa metrik yang dibahas sejauh ini.

Kami akan menggunakan ACK Controller untuk Amazon Managed Service untuk Prometheus untuk menyediakan aturan peringatan dan perekaman.

Mari gunakan pengontrol ACL untuk Layanan Terkelola Amazon untuk layanan 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
```

Jalankan perintah dan setelah beberapa saat Anda akan melihat pesan berikut:

```
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.
```

Sekarang mari kita membuat file yaml untuk menyediakan definisi manajer peringatan dan grup aturan. Simpan file di bawah ini sebagai `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"
```

Ganti WORKSPACE-ID Anda dengan ID Workspace dari ruang kerja yang Anda gunakan.

Sekarang mari kita konfigurasikan definisi manajer peringatan. Simpan fie di bawah ini sebagai `alertmanager.yaml`

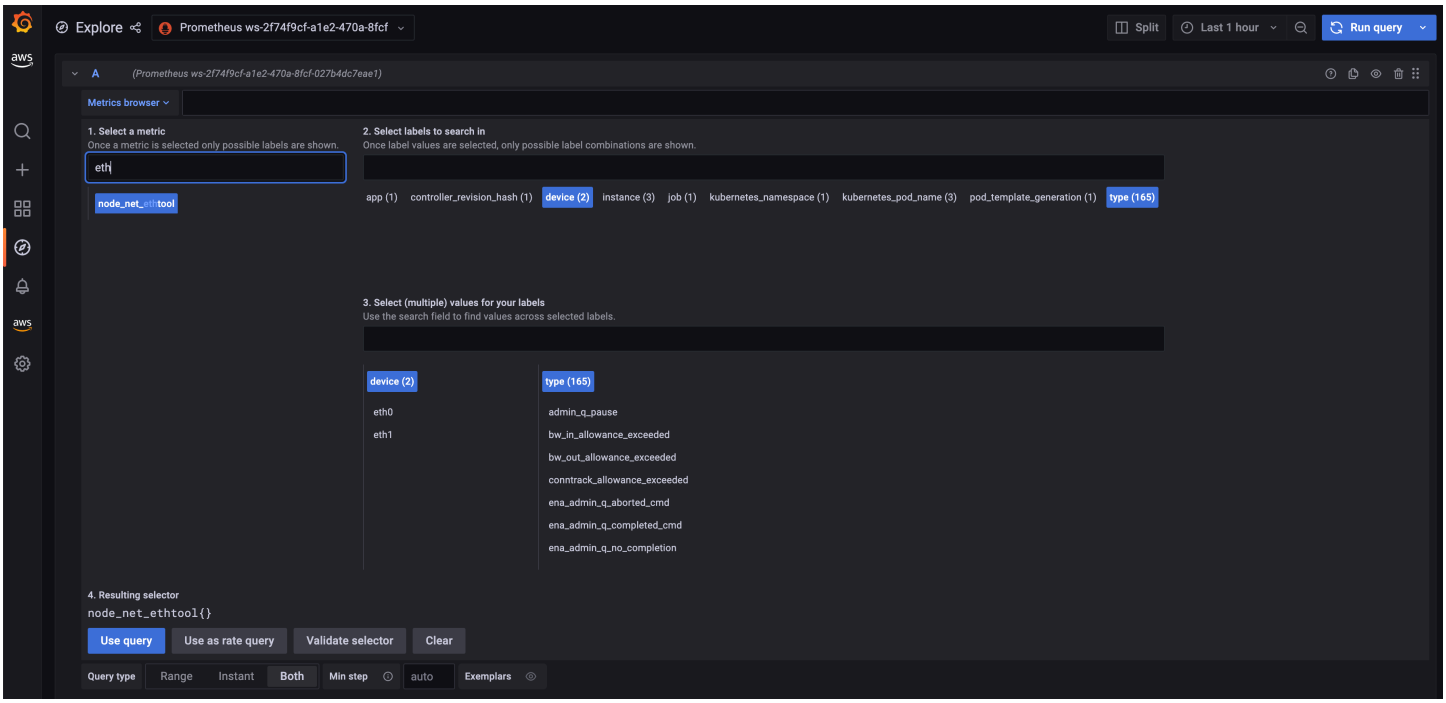
```
apiVersion: prometheusservice.services.k8s.aws/v1alpha1
kind: AlertManagerDefinition
metadata:
  name: alert-manager
spec:
  workspaceID: <Your WORKSPACE-ID >
  configuration: |
    alertmanager_config: |
      route:
         receiver: default_receiver
       receivers:
       - name: default_receiver
          sns_configs:
          - topic_arn: TOPIC-ARN
            sigv4:
              region: REGION
            message: |
              alert_type: {{ .CommonLabels.alertname }}
              event_type: {{ .CommonLabels.event_type }}
```

Ganti WORKSPACE-ID Anda dengan ID Workspace ruang kerja baru, TOPIC-ARN dengan ARN dari topik Amazon Simple Notification Service tempat Anda ingin mengirim peringatan, dan REGION dengan wilayah beban kerja saat ini. Pastikan ruang kerja Anda memiliki izin untuk mengirim pesan ke Amazon SNS.

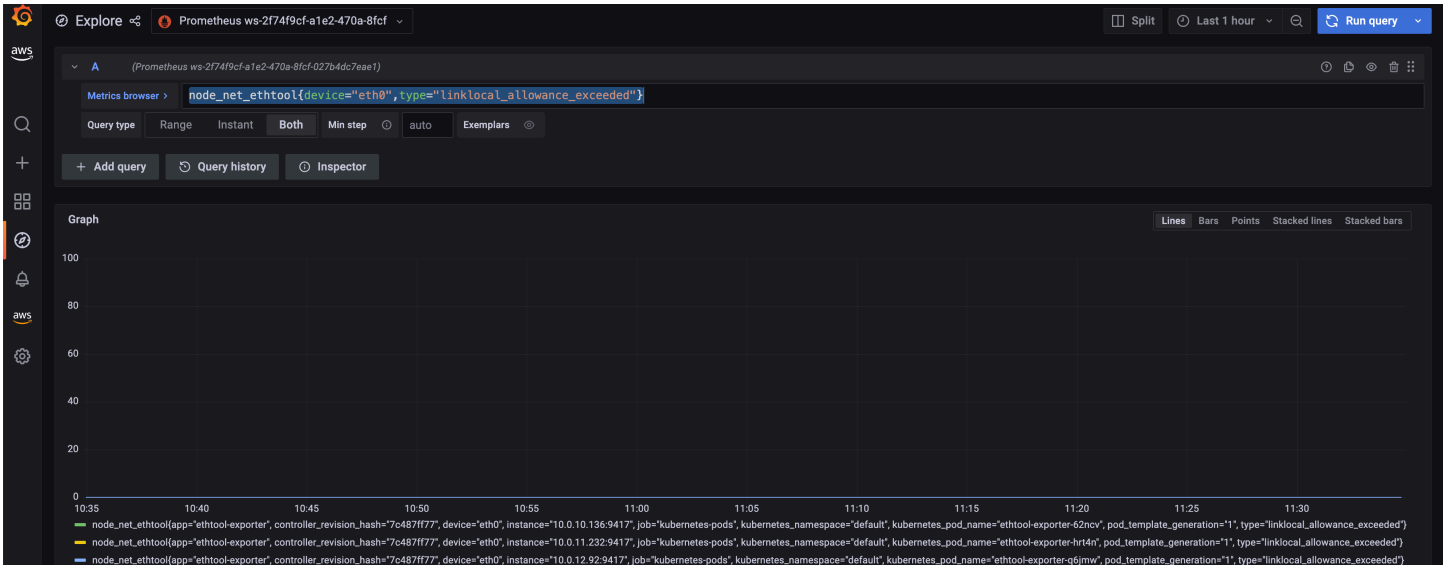
Visualisasikan metrik ethtool di Grafana Terkelola Amazon

Mari kita visualisasikan metrik dalam Grafana Terkelola Amazon dan buat dasbor. Konfigurasikan Layanan Terkelola Amazon untuk Prometheus sebagai sumber data di dalam konsol Grafana Terkelola Amazon. Untuk petunjuk, lihat Menambahkan Amazon Prometheus sebagai sumber data

Mari jelajahi metrik di Grafana Terkelola Amazon sekarang: Klik tombol jelajahi, dan cari ethtool:



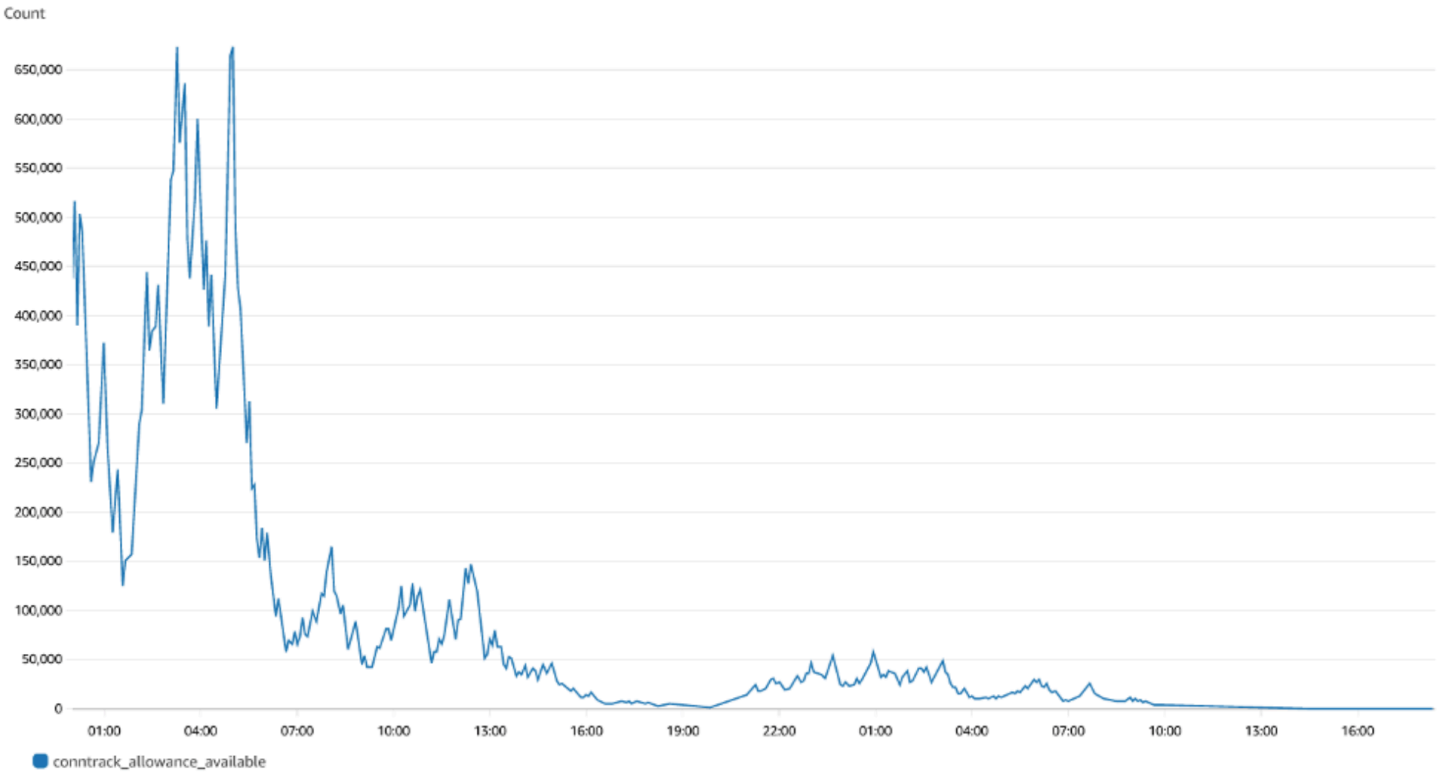
Mari buat dasbor untuk metrik linklocal_allowance_exceeded dengan menggunakan kueri. `rate(node_net_ethtool{device="eth0",type="linklocal_allowance_exceeded"}[30s])` Ini akan menghasilkan dasbor di bawah ini.



Kita dapat dengan jelas melihat bahwa tidak ada paket yang dijatuhkan karena nilainya nol.

Mari buat dasbor untuk metrik conntrack_allowance_exceeded dengan menggunakan kueri. `rate(node_net_ethtool{device="eth0",type="conntrack_allowance_exceeded"}[30s])` Ini akan menghasilkan dasbor di bawah ini.

Metrik `conntrack_allowance_exceeded` dapat divisualisasikan CloudWatch, asalkan Anda menjalankan agen cloudwatch seperti yang dijelaskan di sini. Dasbor yang dihasilkan CloudWatch akan terlihat seperti di bawah ini:



Kita dapat dengan jelas melihat bahwa tidak ada paket yang dijatuhkan karena nilainya nol. Jika Anda menggunakan instans berbasis Nitro, Anda dapat membuat dasbor serupa `conntrack_allowance_available` dan secara proaktif memantau koneksi di instans EC2 Anda. Anda dapat memperluas ini lebih lanjut dengan mengonfigurasi peringatan di Grafana Terkelola Amazon untuk mengirim pemberitahuan ke Slack, SNS, Pagerduty, dll.

Menjalankan kube-proxy dalam Mode IPVS

EKS dalam mode IP Virtual Server (IPVS) memecahkan masalah latensi jaringan yang sering terlihat saat menjalankan cluster besar dengan lebih dari 1.000 layanan dengan kube-proxy berjalan dalam mode iptables lama. Masalah kinerja ini adalah hasil dari pemrosesan berurutan aturan penyaringan paket iptables untuk setiap paket. Masalah latensi ini telah diatasi di nftables, penerus iptables. Namun, pada saat penulisan ini, kube-proxy masih dalam pengembangan untuk memanfaatkan nftables. Untuk mengatasi masalah ini, Anda dapat mengonfigurasi cluster Anda agar berjalan `kube-proxy` dalam mode IPVS.

Ikhtisar

IPVS, yang telah menjadi GA sejak Kubernetes versi 1.11, menggunakan tabel hash daripada pencarian linier untuk memproses paket, memberikan efisiensi untuk cluster dengan ribuan node

dan layanan. IPVS dirancang untuk load balancing, menjadikannya solusi yang cocok untuk masalah kinerja jaringan Kubernetes.

IPVS menawarkan beberapa opsi untuk mendistribusikan lalu lintas ke pod backend. Informasi terperinci untuk setiap opsi dapat ditemukan di dokumentasi resmi Kubernetes, tetapi daftar sederhana ditunjukkan di bawah ini. Round Robin dan Least Connections adalah salah satu pilihan paling populer untuk opsi load balancing IPVS di 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)
```

Implementasi

Hanya beberapa langkah yang diperlukan untuk mengaktifkan IPVS di cluster EKS Anda. Hal pertama yang perlu Anda lakukan adalah memastikan gambar node pekerja EKS Anda memiliki `ipvsadm` paket administrasi Server Virtual Linux diinstal. Untuk menginstal paket ini pada gambar berbasis Fedora, seperti Amazon Linux 2023, Anda dapat menjalankan perintah berikut pada instance node pekerja.

```
sudo dnf install -y ipvsadm
```

Pada gambar berbasis Debian, seperti Ubuntu, perintah instalasi akan terlihat seperti ini.

```
sudo apt-get install ipvsadm
```

Selanjutnya, Anda perlu memuat modul kernel untuk opsi konfigurasi IPVS yang tercantum di atas. Kami merekomendasikan menulis modul-modul ini ke file di dalam `/etc/modules-load.d/` direktori sehingga mereka bertahan dari reboot.

```
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'
```

Anda dapat menjalankan perintah berikut untuk memuat modul-modul ini pada mesin yang sudah berjalan.

```
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**
>
> Sangat disarankan untuk menjalankan langkah-langkah node pekerja ini sebagai bagian dari proses bootstrap node pekerja Anda melalui skrip data pengguna atau dalam skrip build apa pun yang dijalankan untuk membangun node pekerja khusus AMI.

Selanjutnya, Anda akan mengonfigurasi cluster Anda `kube-proxy` DaemonSet untuk berjalan dalam mode IPVS. Ini dilakukan dengan mengatur `kube-proxy mode` ke `ipvs` dan `ipvs scheduler` ke salah satu opsi penyeimbangan beban yang tercantum di atas, misalnya: `rr` untuk Round Robin.

> **Warning**
>
> Ini adalah perubahan yang mengganggu dan harus dilakukan di luar jam kerja. Sebaiknya lakukan perubahan ini selama pembuatan klaster EKS awal untuk meminimalkan dampak.

Anda dapat mengeluarkan perintah AWS CLI untuk mengaktifkan IPVS dengan memperbarui Eks Add-on. `kube-proxy`

```
aws eks update-addon --cluster-name $CLUSTER_NAME --addon-name kube-proxy \
  --configuration-values '{"ipvs": {"scheduler": "rr"}, "mode": "ipvs"}' \
  --resolve-conflicts OVERWRITE
```

Atau Anda dapat melakukan ini dengan memodifikasi `kube-proxy-config` ConfigMap di cluster Anda.

```
kubectl -n kube-system edit cm kube-proxy-config
```

Temukan `scheduler` pengaturan di bawah `ipvs` dan atur nilainya ke salah satu opsi penyeimbangan beban ipvs yang tercantum di atas, misalnya: `rr` untuk Round Robin. Temukan `mode` pengaturan, yang defaultnya`iptables`, dan ubah nilainya menjadi. `ipvs` Hasil dari salah satu opsi harus terlihat mirip dengan konfigurasi di bawah ini.

```
  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
```

Jika node pekerja Anda bergabung ke klaster Anda sebelum melakukan perubahan ini, Anda harus me-restart DaemonSet kube-proxy.

```
kubectl -n kube-system rollout restart ds kube-proxy
```

Validasi

Anda dapat memvalidasi bahwa node cluster dan worker Anda berjalan dalam mode IPVS dengan mengeluarkan perintah berikut pada salah satu node pekerja Anda.

```
sudo ipvsadm -L
```

Minimal, Anda akan melihat hasil yang mirip dengan yang di bawah ini, menampilkan entri untuk layanan Kubernetes API Server di dan layanan CoreDNS `10.100.0.1` di. `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**
>
> Output contoh ini berasal dari kluster EKS dengan rentang alamat IP layanan`10.100.0.0/16`.

Praktik terbaik Skalabilitas EKS

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Panduan ini memberikan saran untuk penskalaan kluster EKS. Tujuan penskalaan cluster EKS adalah untuk memaksimalkan jumlah pekerjaan yang dapat dilakukan oleh satu cluster. Menggunakan kluster EKS tunggal yang besar dapat mengurangi beban operasional dibandingkan dengan menggunakan beberapa cluster, tetapi memiliki trade-off untuk hal-hal seperti penerapan multi-wilayah, isolasi penyewa, dan peningkatan cluster. Dalam dokumen ini kita akan fokus pada bagaimana mencapai skalabilitas maksimum dengan satu cluster.

Cara menggunakan panduan ini

Panduan ini ditujukan untuk pengembang dan administrator yang bertanggung jawab untuk membuat dan mengelola kluster EKS di AWS. Ini berfokus pada beberapa praktik penskalaan Kubernetes generik, tetapi tidak memiliki spesifikasi khusus untuk cluster atau cluster Kubernetes yang dikelola sendiri yang berjalan di luar wilayah AWS dengan EKS Anywhere.

Setiap topik memiliki ikhtisar singkat, diikuti oleh rekomendasi dan praktik terbaik untuk mengoperasikan kluster EKS dalam skala besar. Topik tidak perlu dibaca dalam urutan tertentu dan rekomendasi tidak boleh diterapkan tanpa pengujian dan verifikasi mereka bekerja di cluster Anda.

Memahami dimensi penskalaan

Skalabilitas berbeda dari kinerja dan keandalan, dan ketiganya harus dipertimbangkan saat merencanakan kebutuhan klaster dan beban kerja Anda. Saat skala cluster, mereka perlu dipantau, tetapi panduan ini tidak akan mencakup praktik terbaik pemantauan. EKS dapat menskalakan ke ukuran besar, tetapi Anda perlu merencanakan bagaimana Anda akan menskalakan klaster di atas 300 node atau 5.000 Pod. Ini bukan angka absolut, tetapi mereka berasal dari kolaborasi panduan ini dengan banyak pengguna, insinyur, dan profesional pendukung.

Penskalaan di Kubernetes bersifat multi-dimensi dan tidak ada pengaturan atau rekomendasi khusus yang berfungsi dalam setiap situasi. Area utama di mana kami dapat memberikan panduan untuk penskalaan meliputi:

Kubernetes Control Plane di kluster EKS mencakup semua layanan AWS yang dijalankan dan diskalakan untuk Anda secara otomatis (misalnya server API Kubernetes). Menskalakan Control Plane adalah tanggung jawab AWS, tetapi menggunakan Control Plane secara bertanggung jawab adalah tanggung jawab Anda.

Penskalaan Kubernetes Data Plane berhubungan dengan sumber daya AWS yang diperlukan untuk klaster dan beban kerja Anda, tetapi mereka berada di luar EKS Control Plane. Sumber daya termasuk instans EC2, kubelet, dan penyimpanan semuanya perlu diskalakan sesuai skala klaster Anda.

Layanan cluster adalah pengontrol dan aplikasi Kubernetes yang berjalan di dalam klaster dan menyediakan fungsionalitas untuk klaster dan beban kerja Anda. Ini bisa berupa Pengaya EKS dan juga layanan lain atau bagan Helm yang Anda instal untuk kepatuhan dan integrasi. Layanan ini sering kali bergantung pada beban kerja dan seiring dengan skala beban kerja Anda, layanan klaster Anda perlu menskalakannya.

Beban kerja adalah alasan Anda memiliki cluster dan harus menskalakan secara horizontal dengan cluster. Ada integrasi dan pengaturan yang dimiliki beban kerja di Kubernetes yang dapat membantu skala klaster. Ada juga pertimbangan arsitektur dengan abstraksi Kubernetes seperti ruang nama dan layanan.

Penskalaan ekstra besar

Jika Anda menskalakan satu cluster melebihi 1.000 node atau 50.000 Pod, kami ingin berbicara dengan Anda. Kami merekomendasikan untuk menghubungi tim dukungan atau manajer akun teknis Anda untuk menghubungi spesialis yang dapat membantu Anda merencanakan dan meningkatkan skala di luar informasi yang disediakan dalam panduan ini. Amazon EKS dapat mendukung hingga 100.000 node dalam satu cluster jika Anda dipilih untuk orientasi.

Pesawat Kontrol Kubernetes

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Bidang kontrol Kubernetes terdiri dari Kubernetes API Server, Kubernetes Controller Manager, Scheduler dan komponen lain yang diperlukan agar Kubernetes berfungsi. Batas skalabilitas

komponen-komponen ini berbeda tergantung pada apa yang Anda jalankan di cluster, tetapi area dengan dampak terbesar terhadap penskalaan termasuk versi Kubernetes, pemanfaatan, dan penskalaan Node individual.

Batasi beban kerja dan ledakan simpul

> **Important**
>
> Untuk menghindari mencapai batas API pada bidang kontrol, Anda harus membatasi lonjakan penskalaan yang meningkatkan ukuran cluster dengan persentase dua digit sekaligus (misalnya 1000 node menjadi 1100 node atau 4000 hingga 4.500 pod sekaligus).

Bidang kontrol EKS akan secara otomatis menskalakan saat cluster Anda tumbuh, tetapi ada batasan seberapa cepat skalanya. Saat pertama kali membuat cluster EKS, Control Plane tidak akan segera dapat menskalakan ke ratusan node atau ribuan pod. Untuk membaca lebih lanjut tentang bagaimana EKS telah melakukan peningkatan penskalaan, lihat posting blog ini.

Penskalaan aplikasi besar membutuhkan infrastruktur untuk beradaptasi agar siap sepenuhnya (misalnya penyeimbang beban pemanasan). Untuk mengontrol kecepatan penskalaan, pastikan Anda melakukan penskalaan berdasarkan metrik yang tepat untuk aplikasi Anda. Penskalaan CPU dan memori mungkin tidak secara akurat memprediksi batasan aplikasi Anda dan menggunakan metrik kustom (misalnya permintaan per detik) di Kubernetes Horizontal Pod Autoscaler (HPA) mungkin merupakan opsi penskalaan yang lebih baik.

Untuk menggunakan metrik kustom, lihat contoh di dokumentasi Kubernetes. Jika Anda memiliki kebutuhan penskalaan yang lebih maju atau perlu menskalakan berdasarkan sumber eksternal (misalnya antrian AWS SQS), gunakan KEDA untuk penskalaan beban kerja berbasis peristiwa.

Skala node dan pod dengan aman

Ganti instance yang berjalan lama

Mengganti node secara teratur membuat cluster Anda tetap sehat dengan menghindari penyimpangan konfigurasi dan masalah yang hanya terjadi setelah waktu aktif yang diperpanjang (misalnya kebocoran memori lambat). Penggantian otomatis akan memberi Anda proses dan praktik yang baik untuk peningkatan node dan patch keamanan. Jika setiap node di cluster Anda diganti secara teratur maka ada lebih sedikit kerja keras yang diperlukan untuk mempertahankan proses terpisah untuk pemeliharaan berkelanjutan.

Gunakan pengaturan waktu untuk hidup (TTL) Karpenter untuk mengganti instance setelah dijalankan untuk jangka waktu tertentu. Grup node yang dikelola sendiri dapat menggunakan `max-instance-lifetime` pengaturan untuk siklus node secara otomatis. Grup node terkelola saat ini tidak memiliki fitur ini tetapi Anda dapat melacak permintaan di sini GitHub.

Hapus node yang kurang dimanfaatkan

Anda dapat menghapus node ketika mereka tidak memiliki beban kerja yang berjalan menggunakan ambang batas bawah skala di Kubernetes Cluster Autoscaler dengan `--scale-down-utilization-threshold`atau di Karpenter Anda dapat menggunakan pengaturan penyedia. `ttlSecondsAfterEmpty`

Gunakan anggaran gangguan pod dan shutdown node aman

Menghapus pod dan node dari klaster Kubernetes membutuhkan pengontrol untuk melakukan pembaruan ke beberapa sumber daya (misalnya). EndpointSlices Melakukan hal ini sering atau terlalu cepat dapat menyebabkan pelambatan server API dan pemadaman aplikasi saat perubahan menyebar ke pengontrol. Anggaran Gangguan Pod adalah praktik terbaik untuk memperlambat churn untuk melindungi ketersediaan beban kerja saat node dihapus atau dijadwalkan ulang dalam sebuah klaster.

Gunakan Cloent-Side Cache saat menjalankan Kubectl

Menggunakan perintah kubectl secara tidak efisien dapat menambahkan beban tambahan ke Kubernetes API Server. Anda harus menghindari menjalankan skrip atau otomatisasi yang menggunakan kubectl berulang kali (misalnya dalam loop for) atau menjalankan perintah tanpa cache lokal.

`kubectl`memiliki cache sisi klien yang menyimpan informasi penemuan dari cluster untuk mengurangi jumlah panggilan API yang diperlukan. Cache diaktifkan secara default dan disegarkan setiap 10 menit.

Jika Anda menjalankan kubectl dari container atau tanpa cache sisi klien, Anda mungkin mengalami masalah pembatasan API. Disarankan untuk mempertahankan cache klaster Anda dengan memasang file `--cache-dir` untuk menghindari panggilan API yang tidak biasa.

Nonaktifkan Kompresi kubectl

Menonaktifkan kompresi kubectl di file kubeconfig Anda dapat mengurangi penggunaan API dan CPU klien. Secara default server akan memampatkan data yang dikirim ke klien untuk

mengoptimalkan bandwidth jaringan. Ini menambahkan beban CPU pada klien dan server untuk setiap permintaan dan menonaktifkan kompresi dapat mengurangi overhead dan latensi jika Anda memiliki bandwidth yang memadai. Untuk menonaktifkan kompresi, Anda dapat menggunakan `--disable-compression=true` flag atau set `disable-compression: true` di file kubeconfig Anda.

```
apiVersion: v1
clusters:
- cluster:
    server: serverURL
    disable-compression: true
  name: cluster
```

Shard Cluster Autoscaler

Kubernetes Cluster Autoscaler telah diuji untuk skala hingga 1000 node. Pada cluster besar dengan lebih dari 1000 node, disarankan untuk menjalankan beberapa instance Cluster Autoscaler dalam mode shard. Setiap instance Cluster Autoscaler dikonfigurasi untuk menskalakan satu set grup node. Contoh berikut menunjukkan konfigurasi autoscaling 2 cluster yang dikonfigurasi untuk setiap grup node skala 4.

ClusterAutoscaler-1

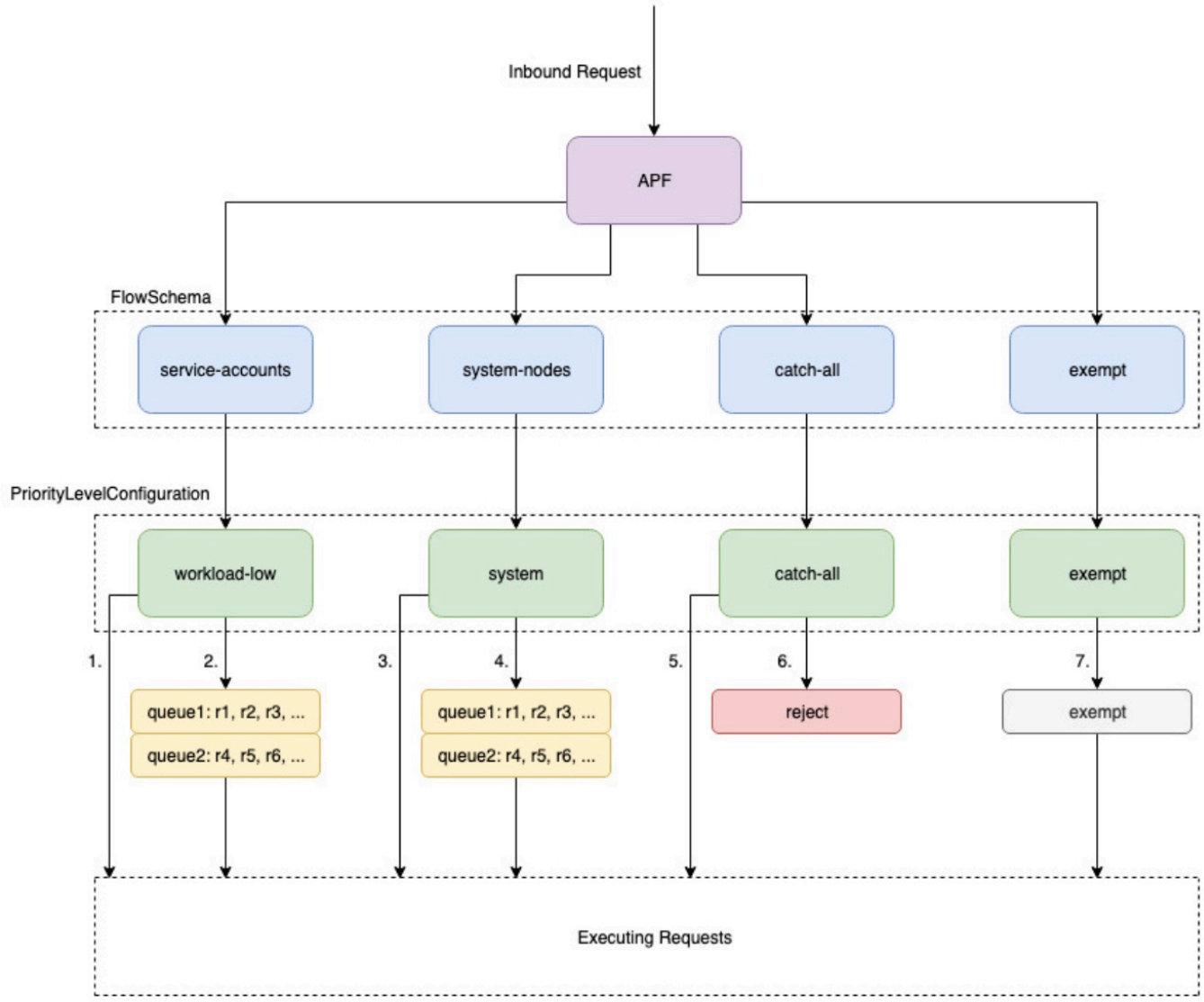
```
autoscalingGroups:
- name: eks-core-node-grp-20220823190924690000000011-80c1660e-030d-476d-cb0d-
d04d585a8fcb
  maxSize: 50
  minSize: 2
- name: eks-data_m1-20220824130553925600000011-5ec167fa-ca93-8ca4-53a5-003e1ed8d306
  maxSize: 450
  minSize: 2
- name: eks-data_m2-20220824130733258600000015-aac167fb-8bf7-429d-d032-e195af4e25f5
  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
```

Prioritas dan Keadilan 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.

Ikhtisar

Untuk melindungi diri dari kelebihan beban selama periode permintaan yang meningkat, Server API membatasi jumlah permintaan dalam penerbangan yang dapat ditulisnya pada waktu tertentu.

Setelah batas ini terlampaui, Server API akan mulai menolak permintaan dan mengembalikan kode respons HTTP 429 untuk “Terlalu Banyak Permintaan” kembali ke klien. Server menjatuhkan permintaan dan meminta klien mencoba lagi nanti lebih disukai daripada tidak memiliki batas sisi server pada jumlah permintaan dan membebani bidang kontrol, yang dapat mengakibatkan penurunan kinerja atau tidak tersedianya.

Mekanisme yang digunakan oleh Kubernetes untuk mengonfigurasi bagaimana permintaan dalam penerbangan ini dibagi di antara berbagai jenis permintaan disebut API Priority and Fairness. Server API mengonfigurasi jumlah total permintaan dalam pesawat yang dapat diterima dengan menjumlahkan nilai yang ditentukan oleh flag dan`--max-requests-inflight`. `--max-mutating-requests-inflight` EKS menggunakan nilai default 400 dan 200 permintaan untuk flag ini, memungkinkan total 600 permintaan dikirim pada waktu tertentu. Namun, karena skala bidang kontrol ke ukuran yang lebih besar sebagai respons terhadap peningkatan pemanfaatan dan churn beban kerja, hal itu juga meningkatkan kuota permintaan dalam penerbangan hingga tahun 2000 (dapat berubah). APF menentukan bagaimana kuota permintaan dalam pesawat ini dibagi lagi di antara jenis permintaan yang berbeda. Perhatikan bahwa bidang kontrol EKS sangat tersedia dengan setidaknya 2 Server API yang terdaftar di setiap cluster. Ini berarti jumlah total permintaan dalam pesawat yang dapat ditangani klaster Anda adalah dua kali (atau lebih tinggi jika diperkecil secara horizontal lebih lanjut) kuota dalam pesawat yang ditetapkan per kube-apiserver. Jumlah ini mencapai beberapa ribu requests/second di kluster EKS terbesar.

Dua jenis objek Kubernetes, dipanggil PriorityLevelConfigurations dan FlowSchemas, mengkonfigurasi bagaimana jumlah total permintaan dibagi antara jenis permintaan yang berbeda. Objek-objek ini dikelola oleh API Server secara otomatis dan EKS menggunakan konfigurasi default dari objek-objek ini untuk versi minor Kubernetes yang diberikan. PriorityLevelConfigurations mewakili sebagian kecil dari jumlah total permintaan yang diizinkan. Misalnya, beban kerja tinggi PriorityLevelConfiguration dialokasikan 98 dari total 600 permintaan. Jumlah permintaan yang dialokasikan untuk semua PriorityLevelConfigurations akan sama dengan 600 (atau sedikit di atas 600 karena Server API akan membulatkan jika level tertentu diberikan sebagian kecil dari permintaan). Untuk memeriksa PriorityLevelConfigurations di cluster Anda dan jumlah permintaan yang dialokasikan untuk masing-masing, Anda dapat menjalankan perintah berikut. Ini adalah default pada 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
```

Jenis objek kedua adalah FlowSchemas. Permintaan API Server dengan sekumpulan properti tertentu diklasifikasikan di bawah yang sama FlowSchema. Properti ini mencakup pengguna yang diautentikasi atau atribut permintaan, seperti grup API, namespace, atau sumber daya. A FlowSchema juga menentukan PriorityLevelConfiguration jenis permintaan yang harus dipetakan. Kedua objek bersama-sama berkata, “Saya ingin jenis permintaan ini dihitung terhadap bagian permintaan dalam penerbangan ini.” Ketika permintaan mengenai Server API, ia akan memeriksa masing-masing FlowSchemas hingga menemukan satu yang cocok dengan semua properti yang diperlukan. Jika beberapa FlowSchemas cocok dengan permintaan, Server API akan memilih FlowSchema dengan prioritas pencocokan terkecil yang ditentukan sebagai properti dalam objek.

Pemetaan FlowSchemas to PriorityLevelConfigurations dapat dilihat menggunakan perintah ini:

```
$ 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 dapat memiliki jenis Antrian, Tolak, atau Dikecualikan. Untuk tipe Antrian dan Tolak, batas diberlakukan pada jumlah maksimum permintaan dalam penerbangan untuk tingkat prioritas tersebut, namun, perilaku berbeda ketika batas tersebut tercapai. Misalnya, workload-high PriorityLevelConfiguration menggunakan tipe Queue dan memiliki 98 permintaan yang tersedia untuk digunakan oleh controller-manager, endpoint-controller, scheduler, eks related controllers dan from pods yang berjalan di namespace kube-system. Karena tipe Antrian digunakan, Server API akan berusaha menyimpan permintaan di memori dan berharap jumlah permintaan dalam pesawat turun di bawah 98 sebelum waktu permintaan ini habis. Jika waktu permintaan yang diberikan habis dalam antrian atau jika terlalu banyak permintaan sudah antri, Server API tidak punya pilihan selain membatalkan permintaan dan mengembalikan klien 429. Perhatikan bahwa antrian dapat mencegah permintaan menerima 429, tetapi disertai dengan tradeoff peningkatan end-to-end latensi pada permintaan.

Sekarang pertimbangkan catch-all FlowSchema yang memetakan ke catch-all PriorityLevelConfiguration dengan tipe Reject. Jika klien mencapai batas 13 permintaan dalam pesawat, Server API tidak akan melakukan antrian dan akan menghapus permintaan secara instan dengan kode respons 429. Terakhir, pemetaan permintaan ke PriorityLevelConfiguration with type Exempt tidak akan pernah menerima 429 dan selalu dikirim segera. Ini digunakan untuk permintaan prioritas tinggi seperti permintaan healthz atau permintaan yang berasal dari grup system:master.

Memantau APF dan Permintaan yang Dijatuhkan

Untuk mengonfirmasi apakah ada permintaan yang dihapus karena APF, metrik API Server untuk `apiserver_flowcontrol_rejected_requests_total` dapat dipantau untuk memeriksa dampak dan. FlowSchemas PriorityLevelConfigurations Misalnya, metrik ini menunjukkan bahwa 100 permintaan dari akun layanan FlowSchema dibatalkan karena waktu permintaan habis dalam antrian rendah beban kerja:

```
% 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
```

Untuk memeriksa seberapa dekat yang PriorityLevelConfiguration diberikan dengan menerima 429 detik atau mengalami peningkatan latensi karena antrian, Anda dapat membandingkan perbedaan antara batas konkurensi dan konkurensi yang digunakan. Dalam contoh ini, kami memiliki buffer 100 permintaan.

```
% 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
```

Untuk memeriksa apakah permintaan tertentu PriorityLevelConfiguration mengalami antrian tetapi belum tentu membatalkan permintaan, metrik untuk `apiserver_flowcontrol_current_inqueue_requests` dapat direferensikan:

```
% 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
```

Metrik Prometheus berguna lainnya meliputi:

- apiserver_flowcontrol_dispatched_requests_total
- apiserver_flowcontrol_request_execution_seconds
- apiserver_flowcontrol_request_wait_duration_seconds

Lihat dokumentasi upstream untuk daftar lengkap metrik APF.

Mencegah Permintaan Jatuh

Cegah 429 detik dengan mengubah beban kerja Anda

Ketika APF membatalkan permintaan karena permintaan yang diberikan PriorityLevelConfiguration melebihi jumlah maksimum permintaan dalam penerbangan yang diizinkan, klien yang terpengaruh

FlowSchemas dapat mengurangi jumlah permintaan yang dieksekusi pada waktu tertentu. Hal ini dapat dicapai dengan mengurangi jumlah total permintaan yang dibuat selama periode di mana 429 terjadi. Perhatikan bahwa permintaan yang berjalan lama seperti panggilan daftar mahal sangat bermasalah karena dihitung sebagai permintaan dalam penerbangan untuk seluruh durasi yang mereka jalankan. Mengurangi jumlah permintaan mahal ini atau mengoptimalkan latensi panggilan daftar ini (misalnya, dengan mengurangi jumlah objek yang diambil per permintaan atau beralih menggunakan permintaan jam tangan) dapat membantu mengurangi total konkurensi yang diperlukan oleh beban kerja yang diberikan.

Cegah 429 detik dengan mengubah pengaturan APF

> ⚠ Warning
>
> Hanya ubah pengaturan APF default jika Anda tahu apa yang Anda lakukan. Setelan APF yang salah konfigurasi dapat mengakibatkan permintaan Server API yang terputus dan gangguan beban kerja yang signifikan.

Satu pendekatan lain untuk mencegah permintaan yang dijatuhkan adalah mengubah default FlowSchemas atau PriorityLevelConfigurations diinstal pada kluster EKS. EKS menginstal pengaturan default upstream untuk FlowSchemas dan PriorityLevelConfigurations untuk versi minor Kubernetes yang diberikan. Server API akan secara otomatis merekonsiliasi objek ini kembali ke defaultnya jika diubah kecuali anotasi berikut pada objek disetel ke false:

```
metadata:
  annotations:
    apf.kubernetes.io/autoupdate-spec: "false"
```

Pada tingkat tinggi, pengaturan APF dapat dimodifikasi menjadi:

- Alokasikan lebih banyak kapasitas dalam penerbangan untuk permintaan yang Anda pedulikan.
- Mengisolasi permintaan yang tidak penting atau mahal yang dapat membuat kapasitas kelaparan untuk jenis permintaan lainnya.

Ini dapat dicapai dengan mengubah default FlowSchemas dan PriorityLevelConfigurations atau dengan membuat objek baru dari jenis ini. Operator dapat meningkatkan nilai assuredConcurrencyShares untuk PriorityLevelConfigurations objek yang relevan untuk meningkatkan fraksi permintaan dalam penerbangan yang dialokasikan. Selain itu, jumlah

permintaan yang dapat diantrian pada waktu tertentu juga dapat ditingkatkan jika aplikasi dapat menangani latensi tambahan yang disebabkan oleh permintaan yang diantrian sebelum dikirim.

Atau, baru FlowSchema dan PriorityLevelConfigurations objek dapat dibuat yang khusus untuk beban kerja pelanggan. Ketahuilah bahwa mengalokasikan lebih banyak assuredConcurrencyShares ke yang sudah ada PriorityLevelConfigurations atau yang baru PriorityLevelConfigurations akan menyebabkan jumlah permintaan yang dapat ditangani oleh bucket lain berkurang karena batas keseluruhan akan tetap menjadi 600 dalam pesawat per Server API.

Saat membuat perubahan pada default APF, metrik ini harus dipantau pada klaster non-produksi untuk memastikan perubahan pengaturan tidak menyebabkan 429 detik yang tidak diinginkan:

1. Metrik untuk `apiserver_flowcontrol_rejected_requests_total` harus dipantau untuk semua FlowSchemas untuk memastikan bahwa tidak ada bucket yang mulai menjatuhkan permintaan.
2. Nilai untuk `apiserver_flowcontrol_nominal_limit_seats` dan `apiserver_flowcontrol_current_executing_seats` harus dibandingkan untuk memastikan bahwa konkurensi yang digunakan tidak berisiko melanggar batas untuk tingkat prioritas tersebut.

Salah satu kasus penggunaan umum untuk mendefinisikan yang baru FlowSchema dan PriorityLevelConfiguration untuk isolasi. Misalkan kita ingin mengisolasi panggilan event list yang sudah berjalan lama dari pod ke bagian permintaannya sendiri. Ini akan mencegah permintaan penting dari pod yang menggunakan akun layanan yang FlowSchema ada menerima 429 detik dan kekurangan kapasitas permintaan. Ingat bahwa jumlah total permintaan dalam pesawat terbatas, namun, contoh ini menunjukkan pengaturan APF dapat dimodifikasi untuk membagi kapasitas permintaan untuk beban kerja yang diberikan dengan lebih baik:

Contoh FlowSchema objek untuk mengisolasi daftar permintaan acara:

```
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
```

- Ini FlowSchema menangkap semua panggilan acara daftar yang dibuat oleh akun layanan di namespace default.
- Prioritas pencocokan 8000 lebih rendah dari nilai 9000 yang digunakan oleh akun layanan yang ada FlowSchema sehingga panggilan acara daftar ini akan cocok dengan -akun daripada akun layanan. list-events-default-service
- Kami menggunakan catch-all PriorityLevelConfiguration untuk mengisolasi permintaan ini. Bucket ini hanya mengizinkan 13 permintaan dalam pesawat untuk digunakan oleh panggilan acara daftar yang sudah berjalan lama ini. Pod akan mulai menerima 429 detik segera setelah mereka mencoba mengeluarkan lebih dari 13 permintaan ini secara bersamaan.

Mengambil sumber daya di server API

Mendapatkan informasi dari server API adalah perilaku yang diharapkan untuk cluster dalam berbagai ukuran. Saat Anda menskalakan jumlah sumber daya di cluster, frekuensi permintaan dan volume data dapat dengan cepat menjadi hambatan bagi bidang kontrol dan akan menyebabkan latensi dan kelambatan API. Tergantung pada tingkat keparahan latensi itu menyebabkan downtime yang tidak terduga jika Anda tidak berhati-hati.

Menyadari apa yang Anda minta dan seberapa sering langkah pertama untuk menghindari jenis masalah ini. Berikut adalah panduan untuk membatasi volume kueri berdasarkan praktik terbaik penskalaan. Saran di bagian ini disediakan agar dimulai dengan opsi yang diketahui memiliki skala terbaik.

Gunakan Informan Bersama

Saat membangun pengontrol dan otomatisasi yang terintegrasi dengan API Kubernetes, Anda sering perlu mendapatkan informasi dari sumber daya Kubernetes. Jika Anda melakukan polling untuk sumber daya ini secara teratur, hal itu dapat menyebabkan beban yang signifikan pada server API.

Menggunakan informan dari pustaka client-go akan memberi Anda manfaat melihat perubahan sumber daya berdasarkan acara alih-alih melakukan polling untuk perubahan. Informan lebih lanjut mengurangi beban dengan menggunakan cache bersama untuk peristiwa dan perubahan sehingga beberapa pengontrol yang menonton sumber daya yang sama tidak menambahkan beban tambahan.

Pengontrol harus menghindari polling sumber daya kluster yang luas tanpa label dan pemilih bidang terutama dalam kelompok besar. Setiap polling yang tidak difilter membutuhkan banyak data yang tidak perlu untuk dikirim dari etcd melalui server API untuk difilter oleh klien. Dengan memfilter berdasarkan label dan ruang nama, Anda dapat mengurangi jumlah pekerjaan yang perlu dilakukan server API untuk memenuhi permintaan dan data yang dikirim ke klien.

Optimalkan penggunaan API Kubernetes

Saat memanggil Kubernetes API dengan pengontrol atau otomatisasi kustom, penting bagi Anda untuk membatasi panggilan hanya pada sumber daya yang Anda butuhkan. Tanpa batas Anda dapat menyebabkan beban yang tidak dibutuhkan pada server API dan etcd.

Disarankan agar Anda menggunakan argumen arloji bila memungkinkan. Tanpa argumen, perilaku default adalah mencantumkan objek. Untuk menggunakan watch alih-alih list, Anda dapat menambahkan `?watch=true` ke akhir permintaan API Anda. Misalnya, untuk mendapatkan semua pod di namespace default dengan menggunakan jam tangan:

```
/api/v1/namespaces/default/pods?watch=true
```

Jika Anda mencantumkan objek, Anda harus membatasi ruang lingkup dari apa yang Anda cantumkan dan jumlah data yang dikembalikan. Anda dapat membatasi data yang dikembalikan dengan menambahkan `limit=500` argumen ke permintaan. `fieldSelector`Argumen dan `/namespace/` jalur dapat berguna untuk memastikan daftar Anda memiliki cakupan yang sempit sesuai kebutuhan. Misalnya, untuk mencantumkan hanya pod yang sedang berjalan di namespace default, gunakan jalur dan argumen API berikut.

```
/api/v1/namespaces/default/pods?fieldSelector=status.phase=Running&limit=500
```

Atau daftar semua pod yang berjalan dengan:

```
/api/v1/pods?fieldSelector=status.phase=Running&limit=500
```

Opsi lain untuk membatasi panggilan jam tangan atau objek terdaftar adalah dengan menggunakan `resourceVersions`yang dapat Anda baca di dokumentasi Kubernetes. Tanpa `resourceVersion` argumen, Anda akan menerima versi terbaru yang tersedia yang memerlukan pembacaan kuorum etcd yang merupakan bacaan paling mahal dan paling lambat untuk database. ResourceVersion bergantung pada sumber daya apa yang Anda coba kueri dan dapat ditemukan di lapangan. `metadata.resourseVersion` Ini juga disarankan jika menggunakan panggilan jam tangan dan bukan hanya daftar panggilan

Ada khusus yang `resourceVersion=0` tersedia yang akan mengembalikan hasil dari cache server API. Ini dapat mengurangi beban etcd tetapi tidak mendukung pagination.

```
/api/v1/namespaces/default/pods?resourceVersion=0
```

Disarankan untuk menggunakan jam tangan dengan set ResourceVersion untuk menjadi nilai terbaru yang diketahui yang diterima dari daftar atau jam tangan sebelumnya. Ini ditangani secara otomatis di client-go. Tetapi disarankan untuk memeriksanya kembali jika Anda menggunakan klien k8s dalam bahasa lain.

```
/api/v1/namespaces/default/pods?watch=true&resourceVersion=362812295
```

Jika Anda memanggil API tanpa argumen apa pun, itu akan menjadi sumber daya yang paling intensif untuk server API dan etcd. Panggilan ini akan mendapatkan semua pod di semua ruang nama tanpa pagination atau membatasi cakupan dan memerlukan kuorum yang dibaca dari etcd.

```
/api/v1/pods
```

Mencegah ternak yang DaemonSet bergemuruh

A DaemonSet memastikan bahwa semua (atau beberapa) node menjalankan salinan pod. Saat node bergabung dengan cluster, daemonset-controller membuat pod untuk node tersebut. Saat node meninggalkan cluster, pod tersebut adalah sampah yang dikumpulkan. Menghapus a DaemonSet akan membersihkan pod yang dibuatnya.

Beberapa kegunaan khas a DaemonSet adalah:

- Menjalankan daemon penyimpanan cluster di setiap node
- Menjalankan daemon koleksi log di setiap node
- Menjalankan daemon pemantauan node di setiap node

Pada cluster dengan ribuan node, membuat yang baru DaemonSet, memperbarui DaemonSet, atau meningkatkan jumlah node dapat menghasilkan beban tinggi yang ditempatkan pada bidang kontrol. Jika DaemonSet pod mengeluarkan permintaan server API yang mahal saat memulai pod, mereka dapat menyebabkan penggunaan sumber daya yang tinggi pada bidang kontrol dari sejumlah besar permintaan bersamaan.

Dalam operasi normal, Anda dapat menggunakan a `RollingUpdate` untuk memastikan peluncuran pod baru secara bertahap. DaemonSet Dengan strategi `RollingUpdate` pembaruan, setelah Anda memperbarui DaemonSet template, pengontrol membunuh DaemonSet pod lama dan membuat DaemonSet pod baru secara otomatis dengan cara yang terkontrol. Paling banyak satu pod DaemonSet akan berjalan di setiap node selama seluruh proses pembaruan. Anda dapat melakukan peluncuran bertahap dengan mengatur `maxUnavailable` ke 1, `maxSurge` ke 0, dan `minReadySeconds` 60. Jika Anda tidak menentukan strategi pembaruan, Kubernetes akan default membuat a `RollingUpdate` dengan `maxUnavailable` sebagai 1, sebagai 0, dan `maxSurge` `minReadySeconds` sebagai 0.

```
minReadySeconds: 60
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 0
    maxUnavailable: 1
```

A `RollingUpdate` memastikan peluncuran bertahap DaemonSet Pod baru jika DaemonSet sudah dibuat dan memiliki jumlah `Ready` Pod yang diharapkan di semua node. Masalah kawanan yang bergemuruh dapat terjadi dalam kondisi tertentu yang tidak tercakup oleh strategi. `RollingUpdate`

Cegah ternak yang bergemuruh saat penciptaan DaemonSet

Secara default, terlepas dari `RollingUpdate` konfigurasinya, daemonset-controller di dalamnya kube-controller-manager akan membuat pod untuk semua node yang cocok secara bersamaan saat Anda membuat yang baru. DaemonSet Untuk memaksa peluncuran pod secara bertahap setelah Anda membuat DaemonSet, Anda dapat menggunakan salah satu atau. `NodeSelector` `NodeAffinity` Ini akan membuat sebuah DaemonSet yang cocok dengan nol node dan kemudian

Anda dapat secara bertahap memperbarui node untuk membuatnya memenuhi syarat untuk menjalankan pod dari DaemonSet pada tingkat yang terkontrol. Anda dapat mengikuti pendekatan ini:

- Tambahkan label ke semua node untuk`run-daemonset=false`.

```
kubectl label nodes --all run-daemonset=false
```

- Buat pengaturan Anda DaemonSet dengan `NodeAffinity` pengaturan untuk mencocokkan node apa pun tanpa `run-daemonset=false` label. Awalnya, ini akan mengakibatkan Anda tidak DaemonSet memiliki pod yang sesuai.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: run-daemonset
          operator: NotIn
          values:
          - "false"
```

- Hapus `run-daemonset=false` label dari node Anda pada tingkat yang terkontrol. Anda dapat menggunakan skrip bash ini sebagai contoh:

```
#!/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
```

- Secara opsional, hapus `NodeAffinity` pengaturan dari DaemonSet objek Anda. Perhatikan bahwa ini juga akan memicu `RollingUpdate` dan secara bertahap mengganti semua DaemonSet pod yang ada karena DaemonSet template berubah.

Mencegah kawanan gemuruh pada skala node

Sama halnya dengan DaemonSet pembuatan, membuat node baru dengan kecepatan cepat dapat menghasilkan sejumlah besar DaemonSet pod yang dimulai secara bersamaan. Anda harus membuat node baru pada tingkat yang terkontrol sehingga controller membuat DaemonSet pod pada tingkat yang sama. Jika ini tidak memungkinkan, Anda dapat membuat node baru awalnya tidak memenuhi syarat untuk yang ada DaemonSet dengan menggunakan`NodeAffinity`. Selanjutnya, Anda dapat menambahkan label ke node baru secara bertahap sehingga daemonset-controller membuat pod pada tingkat yang terkontrol. Anda dapat mengikuti pendekatan ini:

- Tambahkan label ke semua node yang ada untuk `run-daemonset=true`

```
kubectl label nodes --all run-daemonset=true
```

- Perbarui Anda DaemonSet dengan `NodeAffinity` pengaturan untuk mencocokkan node apa pun dengan `run-daemonset=true` label. Perhatikan bahwa ini juga akan memicu `RollingUpdate` dan secara bertahap mengganti semua DaemonSet pod yang ada karena DaemonSet template berubah. Anda harus menunggu `RollingUpdate` sampai selesai sebelum maju ke langkah berikutnya.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: run-daemonset
          operator: In
          values:
          - "true"
```

- Buat node baru di cluster Anda. Perhatikan bahwa node ini tidak akan memiliki `run-daemonset=true` label sehingga tidak DaemonSet akan cocok dengan node tersebut.
- Tambahkan `run-daemonset=true` label ke node baru Anda (yang saat ini tidak memiliki `run-daemonset` label) pada tingkat yang terkontrol. Anda dapat menggunakan skrip bash ini sebagai contoh:

```
#!/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
```

- Secara opsional, hapus `NodeAffinity` pengaturan dari DaemonSet objek Anda dan hapus `run-daemonset` label dari semua node.

Cegah kawanan gemuruh pada pembaruan DaemonSet

`RollingUpdate`Kebijakan hanya akan menghormati `maxUnavailable` pengaturan untuk DaemonSet pod yang ada`Ready`. Jika A hanya DaemonSet memiliki `NotReady` pod atau sebagian besar `NotReady` pod dan Anda memperbarui templatnya, daemonset-controller akan membuat pod baru secara bersamaan untuk pod apa pun. `NotReady` Hal ini dapat mengakibatkan masalah kawanan yang menggelegar jika ada sejumlah besar `NotReady` pod, misalnya jika pod terus-menerus crash looping atau gagal menarik gambar.

Untuk memaksa peluncuran pod secara bertahap saat Anda memperbarui DaemonSet dan ada `NotReady` pod, Anda dapat mengubah sementara strategi pembaruan dari ke DaemonSet . `RollingUpdate` `OnDelete` Dengan`OnDelete`, setelah Anda memperbarui DaemonSet template, controller membuat pod baru setelah Anda secara manual menghapus yang lama sehingga Anda dapat mengontrol peluncuran pod baru. Anda dapat mengikuti pendekatan ini:

- Periksa apakah Anda memiliki `NotReady` pod di DaemonSet.
- Jika tidak, Anda dapat memperbarui DaemonSet template dengan aman dan `RollingUpdate` strategi akan memastikan peluncuran bertahap.
- Jika ya, Anda harus memperbarui terlebih dahulu DaemonSet untuk menggunakan `OnDelete` strategi.

```
updateStrategy:
  type: OnDelete
```

- Selanjutnya, perbarui DaemonSet template Anda dengan perubahan yang diperlukan.
- Setelah pembaruan ini, Anda dapat menghapus DaemonSet pod lama dengan mengeluarkan permintaan delete pod pada tingkat yang terkontrol. Anda dapat menggunakan skrip bash ini sebagai contoh di mana namanya fluentd-elasticsearch di DaemonSet 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
```

- Akhirnya, Anda dapat memperbarui DaemonSet kembali ke `RollingUpdate` strategi sebelumnya.

Pesawat Data Kubernetes

Memilih jenis instans EC2 mungkin merupakan salah satu keputusan tersulit yang dihadapi pelanggan karena dalam kelompok dengan banyak beban kerja. Tidak ada solusi one-size-fits semua. Berikut adalah beberapa tips untuk membantu Anda menghindari jebakan umum dengan penskalaan komputasi.

Penskalaan otomatis node

Kami menyarankan Anda menggunakan penskalaan otomatis node yang mengurangi kerja keras dan terintegrasi secara mendalam dengan Kubernetes. Grup node terkelola dan Karpenter direkomendasikan untuk klaster skala besar.

Grup node terkelola akan memberi Anda fleksibilitas grup Penskalaan Otomatis Amazon EC2 dengan manfaat tambahan untuk peningkatan dan konfigurasi terkelola. Ini dapat diskalakan dengan Kubernetes Cluster Autoscaler dan merupakan opsi umum untuk cluster yang memiliki berbagai kebutuhan komputasi.

Karpenter adalah open source, workload-native node autoscaler yang dibuat oleh AWS. Ini menskalakan node dalam cluster berdasarkan persyaratan beban kerja untuk sumber daya (misalnya

GPU) dan taint dan toleransi (misalnya penyebaran zona) tanpa mengelola grup node. Node dibuat langsung dari EC2 yang menghindari kuota grup node default—450 node per grup—dan memberikan fleksibilitas pemilihan instance yang lebih besar dengan overhead operasional yang lebih sedikit. Kami merekomendasikan pelanggan menggunakan Karpenter jika memungkinkan.

Gunakan banyak jenis instans EC2 yang berbeda

Setiap wilayah AWS memiliki jumlah instans yang tersedia terbatas per jenis instans. Jika Anda membuat klaster yang hanya menggunakan satu jenis instance dan menskalakan jumlah node di luar kapasitas wilayah, Anda akan menerima kesalahan bahwa tidak ada instance yang tersedia. Untuk menghindari masalah ini, Anda tidak boleh secara sewenang-wenang membatasi jenis instance yang dapat digunakan di cluster Anda.

Karpenter akan menggunakan serangkaian luas jenis instance yang kompatibel secara default dan akan memilih instance pada waktu penyediaan berdasarkan persyaratan beban kerja yang tertunda, ketersediaan, dan biaya. Anda dapat memperluas daftar jenis instance yang digunakan dalam `karpenter.k8s.aws/instance-category` kunci. NodePools

Kubernetes Cluster Autoscaler membutuhkan kelompok node untuk memiliki ukuran yang sama sehingga mereka dapat secara konsisten diskalakan. Anda harus membuat beberapa grup berdasarkan CPU dan ukuran memori dan menskalakannya secara independen. Gunakan ec2-instance-selector untuk mengidentifikasi instance yang berukuran sama untuk grup node Anda.

```
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
```

Lebih suka node yang lebih besar untuk mengurangi beban server API

Ketika memutuskan jenis instance apa yang akan digunakan, lebih sedikit, node besar akan menempatkan lebih sedikit beban pada Kubernetes Control Plane karena akan ada lebih sedikit kubelet dan berjalan. DaemonSets Namun, node besar mungkin tidak digunakan sepenuhnya seperti node yang lebih kecil. Ukuran node harus dievaluasi berdasarkan ketersediaan beban kerja dan persyaratan skala Anda.

Sebuah cluster dengan tiga instance u-24tb1.metal (memori 24 TB dan 448 core) memiliki 3 kubelet, dan akan dibatasi hingga 110 pod per node secara default. Jika pod Anda masing-masing menggunakan 4 core maka ini mungkin diharapkan (4 core x 110 = 440 core/node). Dengan cluster 3 node, kemampuan Anda untuk menangani insiden instance akan rendah karena 1 pemadaman instance dapat memengaruhi 1/3 cluster. Anda harus menentukan persyaratan node dan penyebaran pod di beban kerja Anda sehingga scheduler Kubernetes dapat menempatkan beban kerja dengan benar.

Beban kerja harus menentukan sumber daya yang mereka butuhkan dan ketersediaan yang diperlukan melalui taints, toleransi, dan. PodTopologySpread Mereka harus memilih node terbesar yang dapat dimanfaatkan sepenuhnya dan memenuhi tujuan ketersediaan untuk mengurangi beban pesawat kontrol, menurunkan operasi, dan mengurangi biaya.

Penjadwal Kubernetes akan secara otomatis mencoba menyebarkan beban kerja di seluruh zona ketersediaan dan host jika sumber daya tersedia. Jika tidak ada kapasitas yang tersedia, Kubernetes Cluster Autoscaler akan mencoba menambahkan node di setiap Availability Zone secara merata. Karpenter akan mencoba menambahkan node secepat dan semurah mungkin kecuali beban kerja menentukan persyaratan lain.

Untuk memaksa beban kerja menyebar dengan penjadwal dan node baru yang akan dibuat di seluruh zona ketersediaan, Anda harus menggunakan: 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
```

Gunakan ukuran node yang serupa untuk kinerja beban kerja yang konsisten

Beban kerja harus menentukan ukuran node apa yang mereka butuhkan untuk dijalankan untuk memungkinkan kinerja yang konsisten dan penskalaan yang dapat diprediksi. Beban kerja yang meminta CPU 500m akan berkinerja berbeda pada instance dengan 4 core vs satu dengan 16 core. Hindari tipe instance yang menggunakan burstable CPUs seperti instance seri T.

Untuk memastikan beban kerja Anda mendapatkan kinerja yang konsisten, beban kerja dapat menggunakan label Karpenter yang didukung untuk menargetkan ukuran instance tertentu.

```
kind: deployment
...
spec:
  template:
    spec:
    containers:
    nodeSelector:
      karpenter.k8s.aws/instance-size: 8xlarge
```

Beban kerja yang dijadwalkan dalam klaster dengan Kubernetes Cluster Autoscaler harus cocok dengan pemilih node dengan grup node berdasarkan pencocokan label.

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/nodegroup
            operator: In
            values:
            - 8-core-node-group    # match your node group name
```

Gunakan sumber daya komputasi secara efisien

Sumber daya komputasi mencakup instans EC2 dan zona ketersediaan. Menggunakan sumber daya komputasi secara efektif akan meningkatkan skalabilitas, ketersediaan, kinerja, dan mengurangi total biaya Anda. Penggunaan sumber daya yang efisien sangat sulit diprediksi dalam lingkungan penskalaan otomatis dengan banyak aplikasi. Karpenter diciptakan untuk menyediakan contoh sesuai permintaan berdasarkan kebutuhan beban kerja untuk memaksimalkan pemanfaatan dan fleksibilitas.

Karpenter memungkinkan beban kerja untuk mendeklarasikan jenis sumber daya komputasi yang dibutuhkannya tanpa terlebih dahulu membuat grup node atau mengonfigurasi noda label untuk node tertentu. Lihat praktik terbaik Karpenter untuk informasi lebih lanjut. Pertimbangkan untuk mengaktifkan konsolidasi di penyedia Karpenter Anda untuk mengganti node yang sedang digunakan.

Otomatiskan pembaruan Amazon Machine Image (AMI)

Menjaga agar komponen node pekerja tetap up to date akan memastikan Anda memiliki patch keamanan terbaru dan fitur yang kompatibel dengan Kubernetes API. Memperbarui kubelet adalah komponen terpenting untuk fungsionalitas Kubernetes, tetapi mengotomatisasi OS, kernel, dan patch aplikasi yang diinstal secara lokal akan mengurangi pemeliharaan saat Anda menskalakan.

Disarankan agar Anda menggunakan Amazon EKS terbaru yang dioptimalkan Amazon Linux 2 atau Amazon EKS Bottlerocket AMI yang dioptimalkan untuk gambar simpul Anda. Karpenter akan secara otomatis menggunakan AMI terbaru yang tersedia untuk menyediakan node baru di cluster. Grup node terkelola akan memperbarui AMI selama pembaruan grup node tetapi tidak akan memperbarui ID AMI pada waktu penyediaan node.

Untuk Grup Node Terkelola, Anda perlu memperbarui templat peluncuran Auto Scaling Group (ASG) dengan AMI baru IDs saat tersedia untuk rilis patch. Versi minor AMI (misalnya 1.23.5 hingga 1.24.3) akan tersedia di konsol EKS dan API sebagai peningkatan untuk grup node. Versi rilis patch (misalnya 1.23.5 hingga 1.23.6) tidak akan disajikan sebagai peningkatan untuk grup node. Jika Anda ingin memperbarui grup node Anda dengan rilis patch AMI, Anda perlu membuat versi template peluncuran baru dan membiarkan grup node mengganti instance dengan rilis AMI baru.

Anda dapat menemukan AMI terbaru yang tersedia dari halaman ini atau menggunakan 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
```

Gunakan beberapa volume EBS untuk wadah

Volume EBS memiliki kuota input/output (I/O) berdasarkan jenis volume (misalnya gp3) dan ukuran disk. Jika aplikasi Anda berbagi satu volume root EBS dengan host, ini dapat menghabiskan kuota disk untuk seluruh host dan menyebabkan aplikasi lain menunggu kapasitas yang tersedia. Aplikasi menulis ke disk jika mereka menulis file ke partisi overlay mereka, memasang volume lokal dari host, dan juga ketika mereka log ke standar out (STDOUT) tergantung pada agen logging yang digunakan.

Untuk menghindari I/O kehabisan disk, Anda harus memasang volume kedua ke folder status kontainer (mis. /run/containerd), gunakan volume EBS terpisah untuk penyimpanan beban kerja, dan nonaktifkan logging lokal yang tidak perlu.

Untuk memasang volume kedua ke instans EC2 Anda menggunakan eksctl, Anda dapat menggunakan grup node dengan konfigurasi ini:

```
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"
```

Jika Anda menggunakan terraform untuk menyediakan grup node Anda, silakan lihat contoh di EKS Blueprints for terraform. Jika Anda menggunakan Karpenter untuk menyediakan node, Anda dapat menggunakan `blockDeviceMappings`dengan data pengguna node untuk menambahkan volume tambahan.

Untuk memasang volume EBS langsung ke pod Anda, Anda harus menggunakan driver AWS EBS CSI dan menggunakan volume dengan kelas penyimpanan.

```
---
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
```

Hindari instance dengan batas lampiran EBS rendah jika beban kerja menggunakan volume EBS

EBS adalah salah satu cara termudah agar beban kerja memiliki penyimpanan persisten, tetapi juga dilengkapi dengan keterbatasan skalabilitas. Setiap jenis instans memiliki jumlah maksimum

volume EBS yang dapat dilampirkan. Beban kerja perlu mendeklarasikan tipe instance apa yang harus mereka jalankan dan membatasi jumlah replika pada satu instance dengan taints Kubernetes.

Nonaktifkan logging yang tidak perlu ke disk

Hindari logging lokal yang tidak perlu dengan tidak menjalankan aplikasi Anda dengan debug logging dalam produksi dan menonaktifkan logging yang sering membaca dan menulis ke disk. Journald adalah layanan logging lokal yang menyimpan buffer log di memori dan flushes ke disk secara berkala. Journald lebih disukai daripada syslog yang mencatat setiap baris langsung ke disk. Menonaktifkan syslog juga menurunkan jumlah total penyimpanan yang Anda butuhkan dan menghindari perlunya aturan rotasi log yang rumit. Untuk menonaktifkan syslog, Anda dapat menambahkan cuplikan berikut ke konfigurasi cloud-init Anda:

```
runcmd:
  - [ systemctl, disable, --now, syslog.service ]
```

Contoh tambalan di tempat ketika kecepatan pembaruan OS adalah suatu keharusan

> **Important**
>
> Contoh penambalan di tempat hanya boleh dilakukan bila diperlukan. Amazon merekomendasikan untuk memperlakukan infrastruktur sebagai pembaruan yang tidak dapat diubah dan diuji secara menyeluruh yang dipromosikan melalui lingkungan yang lebih rendah dengan cara yang sama seperti aplikasi. Bagian ini berlaku ketika itu tidak memungkinkan.

Dibutuhkan beberapa detik untuk menginstal paket pada host Linux yang ada tanpa mengganggu beban kerja kontainer. Paket dapat diinstal dan divalidasi tanpa menutup, menguras, atau mengganti instance.

Untuk mengganti instance, Anda harus terlebih dahulu membuat, memvalidasi, dan mendistribusikan yang baru AMIs. Instance perlu memiliki pengganti yang dibuat, dan instance lama perlu ditutup dan dikeringkan. Kemudian beban kerja perlu dibuat pada instance baru, diverifikasi, dan diulang untuk semua instance yang perlu ditambal. Dibutuhkan berjam-jam, berhari-hari, atau berminggu-minggu untuk mengganti instance dengan aman tanpa mengganggu beban kerja.

Amazon merekomendasikan penggunaan infrastruktur yang tidak dapat diubah yang dibangun, diuji, dan dipromosikan dari sistem deklaratif otomatis, tetapi jika Anda memiliki persyaratan untuk menambal sistem dengan cepat maka Anda perlu menambal sistem di tempat dan menggantinya saat baru AMIs tersedia. Karena perbedaan waktu yang besar antara menambal dan mengganti sistem, kami sarankan menggunakan AWS Systems Manager Patch Manager untuk mengotomatiskan patch node saat diperlukan untuk melakukannya.

Menambal node akan memungkinkan Anda untuk dengan cepat meluncurkan pembaruan keamanan dan mengganti instance pada jadwal reguler setelah AMI Anda diperbarui. Jika Anda menggunakan sistem operasi dengan sistem file root read-only seperti Flatcar Container Linux atau Bottlerocket OS, kami sarankan menggunakan operator pembaruan yang bekerja dengan sistem operasi tersebut. Operator pembaruan Flatcar Linux dan operator pembaruan Bottlerocket akan me-reboot instance untuk menjaga node tetap up to date secara otomatis.

Layanan Cluster

Layanan cluster berjalan di dalam kluster EKS, tetapi itu bukan beban kerja pengguna. Jika Anda memiliki server Linux, Anda sering perlu menjalankan layanan seperti NTP, syslog, dan runtime container untuk mendukung beban kerja Anda. Layanan cluster serupa, layanan pendukung yang membantu Anda mengotomatiskan dan mengoperasikan klaster Anda. Di Kubernetes ini biasanya dijalankan di namespace kube-system dan beberapa dijalankan sebagai. DaemonSets

Layanan cluster diharapkan memiliki up-time yang tinggi dan seringkali kritis selama pemadaman dan untuk pemecahan masalah. Jika layanan cluster inti tidak tersedia, Anda mungkin kehilangan akses ke data yang dapat membantu memulihkan atau mencegah pemadaman (misalnya pemanfaatan disk yang tinggi). Mereka harus berjalan pada instance komputasi khusus seperti grup node terpisah atau AWS Fargate. Ini akan memastikan bahwa layanan klaster tidak terpengaruh pada instance bersama oleh beban kerja yang mungkin meningkatkan atau menggunakan lebih banyak sumber daya.

Skala CoreDNS

Scaling CoreDNS memiliki dua mekanisme utama. Mengurangi jumlah panggilan ke layanan CoreDNS dan meningkatkan jumlah replika.

Kurangi kueri eksternal dengan menurunkan ndots

Pengaturan ndots menentukan berapa banyak periode (alias “titik”) dalam nama domain dianggap cukup untuk menghindari kueri DNS. Jika aplikasi Anda memiliki setelan ndots 5 (default) dan Anda

meminta sumber daya dari domain eksternal seperti api.example.com (2 titik) maka CoreDNS akan ditanyakan untuk setiap domain pencarian yang ditentukan dalam /etc/resolv.conf untuk domain yang lebih spesifik. Secara default domain berikut akan dicari sebelum membuat permintaan eksternal.

```
api.example.<namespace>.svc.cluster.local
api.example.svc.cluster.local
api.example.cluster.local
api.example.<region>.compute.internal
```

`region`Nilai `namespace` dan akan diganti dengan namespace beban kerja dan wilayah komputasi Anda. Anda mungkin memiliki domain pencarian tambahan berdasarkan pengaturan klaster Anda.

Anda dapat mengurangi jumlah permintaan ke CoreDNS dengan menurunkan opsi ndots beban kerja Anda atau sepenuhnya memenuhi syarat permintaan domain Anda dengan menyertakan trailing. (misalnya`api.example.com.`). Jika beban kerja Anda terhubung ke layanan eksternal melalui DNS, kami sarankan untuk menyetel ndots ke 2 sehingga beban kerja tidak membuat kueri DNS cluster yang tidak perlu di dalam cluster. Anda dapat mengatur server DNS dan domain pencarian yang berbeda jika beban kerja tidak memerlukan akses ke layanan di dalam klaster.

```
spec:
  dnsPolicy: "None"
  dnsConfig:
    options:
      - name: ndots
        value: "2"
      - name: edns0
```

Jika Anda menurunkan ndots ke nilai yang terlalu rendah atau domain yang Anda sambungkan tidak menyertakan spesifisitas yang cukup (termasuk trailing.) maka ada kemungkinan pencarian DNS akan gagal. Pastikan Anda menguji bagaimana pengaturan ini akan memengaruhi beban kerja Anda.

Skala CoreDNS Secara Horizontal

Instance CoreDNS dapat diskalakan dengan menambahkan replika tambahan ke penerapan. Disarankan Anda menggunakan NodeLocal DNS atau autoscaler proporsional cluster untuk menskalakan CoreDNS.

NodeLocal DNS akan membutuhkan jalankan satu instance per node—sebagai DaemonSet —yang membutuhkan lebih banyak sumber daya komputasi di cluster, tetapi akan menghindari permintaan DNS yang gagal dan mengurangi waktu respons untuk kueri DNS di cluster. Autoscaler proporsional

cluster akan menskalakan CoreDNS berdasarkan jumlah node atau core dalam cluster. Ini bukan korelasi langsung dengan permintaan kueri, tetapi dapat berguna tergantung pada beban kerja dan ukuran cluster Anda. Skala proporsional default adalah menambahkan replika tambahan untuk setiap 256 core atau 16 node di cluster — mana yang terjadi lebih dulu.

Jika menggunakan add-on CoreDNS EKS, pertimbangkan untuk mengaktifkan opsi penskalaan otomatis. Autoscaler CoreDNS secara dinamis menyesuaikan jumlah replika CoreDNS dengan memantau jumlah node dan inti CPU, menggunakan rumus yang mengambil maksimum (node-16) atau (inti CPU 256), segera ditingkatkan saat diperlukan dan turun secara bertahap untuk menjaga stabilitas.

Skala Server Metrik Kubernetes Secara Vertikal

Kubernetes Metrics Server mendukung penskalaan horizontal dan vertikal. Dengan menskalakan Server Metrik secara horizontal, ini akan sangat tersedia, tetapi tidak akan menskalakan secara horizontal untuk menangani lebih banyak metrik cluster. Anda perlu menskalakan Server Metrik secara vertikal berdasarkan rekomendasinya saat node dan metrik yang dikumpulkan ditambahkan ke cluster.

Server Metrik menyimpan data yang dikumpulkan, dikumpulkan, dan disajikan dalam memori. Seiring pertumbuhan cluster, jumlah data yang disimpan Metrics Server meningkat. Dalam cluster besar Metrics Server akan membutuhkan lebih banyak sumber daya komputasi daripada memori dan reservasi CPU yang ditentukan dalam instalasi default. Anda dapat menggunakan Vertical Pod Autoscaler (VPA) atau Addon Resizer untuk menskalakan Server Metrik. Addon Resizer menskalakan secara vertikal sebanding dengan node pekerja dan skala VPA berdasarkan penggunaan CPU dan memori.

Durasi lameduck CoreDNS

Pod menggunakan `kube-dns` Service untuk resolusi nama. Kubernetes menggunakan NAT tujuan (DNAT) untuk mengarahkan `kube-dns` lalu lintas dari node ke pod backend CoreDNS. Saat Anda menskalakan Deployment CoreDNS`kube-proxy`, memperbarui aturan dan rantai iptables pada node untuk mengarahkan lalu lintas DNS ke pod CoreDNS. Menyebarkan titik akhir baru saat Anda meningkatkan dan menghapus aturan saat Anda menurunkan CoreDNS dapat memakan waktu antara 1 hingga 10 detik tergantung pada ukuran cluster.

Penundaan propagasi ini dapat menyebabkan kegagalan pencarian DNS ketika pod CoreDNS dihentikan namun aturan iptables node belum diperbarui. Dalam skenario ini, node dapat terus mengirim kueri DNS ke Pod CoreDNS yang dihentikan.

Anda dapat mengurangi kegagalan pencarian DNS dengan menyetel durasi lameduck di pod CoreDNS Anda. Saat dalam mode lameduck, CoreDNS akan terus menanggapi permintaan dalam penerbangan. Menyetel durasi lameduck akan menunda proses shutdown CoreDNS, memungkinkan node waktu yang mereka butuhkan untuk memperbarui aturan dan rantai iptables mereka.

Sebaiknya atur durasi lameduck CoreDNS menjadi 30 detik.

Penyelidikan kesiapan CoreDNS

Kami merekomendasikan penggunaan `/ready` bukan `/health` untuk probe kesiapan CoreDNS.

Sejalan dengan rekomendasi sebelumnya untuk menyetel durasi lameduck menjadi 30 detik, memberikan waktu yang cukup untuk aturan iptables node diperbarui sebelum penghentian pod, menggunakan alih-alih `/ready` untuk probe kesiapan CoreDNS memastikan bahwa pod CoreDNS sepenuhnya siap saat startup untuk `/health` segera menanggapi permintaan DNS.

```
readinessProbe:
  httpGet:
    path: /ready
    port: 8181
    scheme: HTTP
```

Untuk informasi lebih lanjut tentang plugin CoreDNS Ready, silakan merujuk ke https://coredns. io/plugins/ready/

Agen penebangan dan pemantauan

Agen logging dan monitoring dapat menambahkan beban signifikan ke bidang kontrol klaster Anda karena agen menanyakan server API untuk memperkaya log dan metrik dengan metadata beban kerja. Agen pada node hanya memiliki akses ke sumber daya node lokal untuk melihat hal-hal seperti wadah dan nama proses. Menanyakan server API, ia dapat menambahkan lebih banyak detail seperti nama dan label penerapan Kubernetes. Ini bisa sangat membantu untuk pemecahan masalah tetapi merugikan penskalaan.

Karena ada begitu banyak opsi berbeda untuk pencatatan dan pemantauan, kami tidak dapat menunjukkan contoh untuk setiap penyedia. Dengan fluentbit, kami merekomendasikan untuk mengaktifkan Use_Kubelet untuk mengambil metadata dari kubelet lokal alih-alih Kubernetes API Server dan disetel ke nomor yang mengurangi panggilan berulang saat data dapat di-cache (misalnya `Kube_Meta_Cache_TTL` 60).

Pemantauan penskalaan dan pencatatan memiliki dua opsi umum:

- Nonaktifkan integrasi
- Pengambilan sampel dan penyaringan

Menonaktifkan integrasi seringkali bukan pilihan karena Anda kehilangan metadata log. Ini menghilangkan masalah penskalaan API, tetapi akan menimbulkan masalah lain dengan tidak memiliki metadata yang diperlukan saat diperlukan.

Pengambilan sampel dan penyaringan mengurangi jumlah metrik dan log yang dikumpulkan. Ini akan menurunkan jumlah permintaan ke Kubernetes API, dan ini akan mengurangi jumlah penyimpanan yang dibutuhkan untuk metrik dan log yang dikumpulkan. Mengurangi biaya penyimpanan akan menurunkan biaya untuk keseluruhan sistem.

Kemampuan untuk mengkonfigurasi sampling tergantung pada perangkat lunak agen dan dapat diimplementasikan pada berbagai titik konsumsi. Penting untuk menambahkan sampling sedekat mungkin dengan agen karena kemungkinan di situlah panggilan server API terjadi. Hubungi penyedia Anda untuk mengetahui lebih lanjut tentang dukungan pengambilan sampel.

Jika Anda menggunakan CloudWatch dan CloudWatch Log, Anda dapat menambahkan pemfilteran agen menggunakan pola yang dijelaskan dalam dokumentasi.

Untuk menghindari kehilangan log dan metrik, Anda harus mengirim data ke sistem yang dapat menyangga data jika terjadi pemadaman pada titik akhir penerima. Dengan fluentbit, Anda dapat menggunakan Amazon Kinesis Data Firehose untuk menyimpan data sementara yang dapat mengurangi kemungkinan kelebihan beban lokasi penyimpanan data akhir Anda.

Beban kerja

Beban kerja berdampak pada seberapa besar skala klaster Anda. Beban kerja yang menggunakan Kubernetes APIs akan membatasi jumlah total beban kerja yang dapat Anda miliki dalam satu cluster, tetapi ada beberapa default yang dapat Anda ubah untuk membantu mengurangi beban.

Beban kerja di klaster Kubernetes memiliki akses ke fitur-fitur yang terintegrasi dengan Kubernetes API (misalnya Secrets and ServiceAccounts), tetapi fitur-fitur ini tidak selalu diperlukan dan harus dinonaktifkan jika tidak digunakan. Membatasi akses beban kerja dan ketergantungan pada bidang kontrol Kubernetes akan meningkatkan jumlah beban kerja yang dapat Anda jalankan di klaster dan meningkatkan keamanan klaster Anda dengan menghapus akses yang tidak perlu ke beban kerja

dan menerapkan praktik hak istimewa yang paling sedikit. Silakan baca praktik terbaik keamanan untuk informasi lebih lanjut.

Gunakan IPv6 untuk jaringan pod

Anda tidak dapat mentransisikan VPC dari IPv4 ke IPv6 pengaktifan IPv6 sebelum menyediakan klaster adalah penting. Jika Anda mengaktifkan IPv6 dalam VPC, itu tidak berarti Anda harus menggunakannya dan jika pod dan layanan Anda digunakan, Anda masih dapat IPv6 merutekan lalu lintas ke dan dari IPv4 alamat. Silakan lihat praktik terbaik jaringan EKS untuk informasi lebih lanjut.

Menggunakan IPv6 di klaster Anda menghindari beberapa batas penskalaan klaster dan beban kerja yang paling umum. IPv6 menghindari kehabisan alamat IP di mana pod dan node tidak dapat dibuat karena tidak ada alamat IP yang tersedia. Ini juga memiliki peningkatan kinerja per node karena pod menerima alamat IP lebih cepat dengan mengurangi jumlah lampiran ENI per node. Anda dapat mencapai kinerja node yang serupa dengan menggunakan mode IPv4 awalan di VPC CNI, tetapi Anda masih perlu memastikan bahwa Anda memiliki cukup alamat IP yang tersedia di VPC.

Batasi jumlah layanan per namespace

Jumlah maksimum layanan di ruang nama adalah 5.000 dan jumlah maksimum layanan dalam sebuah cluster adalah 10.000. Untuk membantu mengatur beban kerja dan layanan, meningkatkan kinerja, dan untuk menghindari dampak cascading untuk sumber daya cakupan namespace, kami sarankan untuk membatasi jumlah layanan per namespace menjadi 500.

Jumlah aturan tabel IP yang dibuat per node dengan kube-proxy bertambah dengan jumlah total layanan di cluster. Menghasilkan ribuan aturan tabel IP dan paket routing melalui aturan tersebut memiliki dampak kinerja pada node dan menambahkan latensi jaringan.

Buat ruang nama Kubernetes yang mencakup satu lingkungan aplikasi selama jumlah layanan per namespace di bawah 500. Ini akan membuat penemuan layanan cukup kecil untuk menghindari batas penemuan layanan dan juga dapat membantu Anda menghindari tabrakan penamaan layanan. Lingkungan aplikasi (misalnya dev, test, prod) harus menggunakan cluster EKS terpisah alih-alih ruang nama.

Memahami Kuota Elastic Load Balancer

Saat membuat layanan Anda, pertimbangkan jenis load balancing yang akan Anda gunakan (misalnya Network Load Balancer (NLB) atau Application Load Balancer (ALB)). Setiap jenis penyeimbang beban menyediakan fungsionalitas yang berbeda dan memiliki kuota yang berbeda.

Beberapa kuota default dapat disesuaikan, tetapi ada beberapa kuota maksimum yang tidak dapat diubah. Untuk melihat kuota dan penggunaan akun Anda, lihat dasbor Service Quotas di konsol AWS.

Misalnya, target ALB default adalah 1000. Jika Anda memiliki layanan dengan lebih dari 1000 titik akhir, Anda perlu menambah kuota atau membagi layanan menjadi beberapa ALBs atau menggunakan Kubernetes Ingress. Target NLB default adalah 3000, tetapi terbatas pada 500 target per AZ. Jika klaster Anda menjalankan lebih dari 500 pod untuk layanan NLB, Anda harus menggunakan beberapa AZs atau meminta peningkatan batas kuota.

Alternatif untuk menggunakan penyeimbang beban yang digabungkan ke layanan adalah dengan menggunakan pengontrol masuk. Pengontrol AWS Load Balancer dapat membuat sumber daya ingress, tetapi Anda dapat mempertimbangkan ALBs untuk menjalankan pengontrol khusus di klaster Anda. Pengontrol ingress in-cluster memungkinkan Anda mengekspos beberapa layanan Kubernetes dari penyeimbang beban tunggal dengan menjalankan proxy terbalik di dalam klaster Anda. Pengontrol memiliki fitur yang berbeda seperti dukungan untuk API Gateway yang mungkin memiliki manfaat tergantung pada berapa banyak dan seberapa besar beban kerja Anda.

Gunakan Route 53, Global Accelerator, atau CloudFront

Agar layanan menggunakan beberapa penyeimbang beban tersedia sebagai satu titik akhir, Anda perlu menggunakan Amazon, AWS Global Accelerator CloudFront, atau Amazon Route 53 untuk mengekspos semua penyeimbang beban sebagai satu titik akhir yang menghadap pelanggan. Setiap opsi memiliki manfaat yang berbeda dan dapat digunakan secara terpisah atau bersama-sama tergantung pada kebutuhan Anda.

Route 53 dapat mengekspos beberapa penyeimbang beban dengan nama umum dan dapat mengirim lalu lintas ke masing-masing berdasarkan bobot yang ditetapkan. Anda dapat membaca lebih lanjut tentang bobot DNS dalam dokumentasi dan Anda dapat membaca cara mengimplementasikannya dengan pengontrol DNS eksternal Kubernetes dalam dokumentasi AWS Load Balancer Controller.

Global Accelerator dapat merutekan beban kerja ke wilayah terdekat berdasarkan alamat IP permintaan. Ini mungkin berguna untuk beban kerja yang diterapkan ke beberapa wilayah, tetapi tidak meningkatkan perutean ke satu cluster di satu wilayah. Menggunakan Route 53 dalam kombinasi dengan Global Accelerator memiliki manfaat tambahan seperti pemeriksaan kesehatan dan failover otomatis jika AZ tidak tersedia. Anda dapat melihat contoh penggunaan Global Accelerator dengan Route 53 di posting blog ini.

CloudFront dapat digunakan dengan Route 53 dan Global Accelerator atau dengan sendirinya untuk merutekan lalu lintas ke beberapa tujuan. CloudFront menyimpan aset yang dilayani dari sumber asal yang dapat mengurangi persyaratan bandwidth tergantung pada apa yang Anda layani.

Gunakan EndpointSlices alih-alih Titik Akhir

Saat menemukan pod yang cocok dengan label layanan, Anda harus menggunakan EndpointSlicesalih-alih Titik Akhir. Endpoint adalah cara sederhana untuk mengekspos layanan dalam skala kecil tetapi layanan besar yang secara otomatis menskalakan atau memiliki pembaruan menyebabkan banyak lalu lintas di bidang kontrol Kubernetes. EndpointSlices memiliki pengelompokan otomatis yang memungkinkan hal-hal seperti petunjuk sadar topologi.

Tidak semua pengontrol menggunakan secara EndpointSlices default. Anda harus memverifikasi pengaturan pengontrol Anda dan mengaktifkannya jika diperlukan. Untuk AWS Load Balancer Controller, Anda harus mengaktifkan flag `--enable-endpoint-slices` opsional untuk digunakan. EndpointSlices

Gunakan rahasia abadi dan eksternal jika memungkinkan

Kubelet menyimpan cache kunci dan nilai saat ini untuk Secrets yang digunakan dalam volume untuk pod pada node tersebut. Kubelet mengatur arloji pada Rahasia untuk mendeteksi perubahan. Seiring dengan skala klaster, semakin banyak jam tangan dapat berdampak negatif pada kinerja server API.

Ada dua strategi untuk mengurangi jumlah jam tangan di Rahasia:

- Untuk aplikasi yang tidak memerlukan akses ke sumber daya Kubernetes, Anda dapat menonaktifkan rahasia akun layanan pemasangan otomatis dengan menyetel Token: false automountServiceAccount
- Jika rahasia aplikasi Anda statis dan tidak akan dimodifikasi di masa depan, tandai rahasianya sebagai tidak dapat diubah. Kubelet tidak memelihara API watch untuk rahasia yang tidak dapat diubah.

Untuk menonaktifkan pemasangan akun layanan secara otomatis ke pod, Anda dapat menggunakan pengaturan berikut di beban kerja Anda. Anda dapat mengganti pengaturan ini jika beban kerja tertentu memerlukan akun layanan.

```
apiVersion: v1
kind: ServiceAccount
```

```
metadata:
  name: app
automountServiceAccountToken: true
```

Pantau jumlah rahasia di cluster sebelum melebihi batas 10.000. Anda dapat melihat jumlah total rahasia dalam sebuah cluster dengan perintah berikut. Anda harus memantau batas ini melalui alat pemantauan cluster Anda.

```
kubectl get secrets -A | wc -l
```

Anda harus mengatur pemantauan untuk mengingatkan admin klaster sebelum batas ini tercapai. Pertimbangkan untuk menggunakan opsi manajemen rahasia eksternal seperti AWS Key Management Service (AWS KMS) atau Hashicorp Vault dengan driver Secrets Store CSI.

Batasi riwayat Penerapan

Pod bisa lambat saat membuat, memperbarui, atau menghapus karena objek lama masih dilacak di cluster. Anda dapat mengurangi `revisionHistoryLimit` penerapan menjadi pembersihan yang lebih lama ReplicaSets yang akan menurunkan jumlah total objek yang dilacak oleh Kubernetes Controller Manager. Batas riwayat default untuk Deployment di 10.

Jika klaster Anda membuat banyak objek pekerjaan melalui CronJobs atau mekanisme lain, Anda harus menggunakan `ttlSecondsAfterFinished`pengaturan untuk secara otomatis membersihkan pod lama di cluster. Ini akan menghapus pekerjaan yang berhasil dieksekusi dari riwayat pekerjaan setelah jangka waktu tertentu.

Nonaktifkan secara enableServiceLinks default

Ketika sebuah Pod berjalan pada Node, kubelet menambahkan satu set variabel lingkungan untuk setiap Service yang aktif. Proses Linux memiliki ukuran maksimum untuk lingkungan mereka yang dapat dicapai jika Anda memiliki terlalu banyak layanan di namespace Anda. Jumlah layanan per namespace tidak boleh melebihi 5.000. Setelah ini, jumlah variabel lingkungan layanan melebihi batas shell, menyebabkan Pod mogok saat startup.

Ada alasan lain Pod tidak boleh menggunakan variabel lingkungan layanan untuk penemuan layanan. Bentrokan nama variabel lingkungan, nama layanan bocor, dan ukuran lingkungan total adalah beberapa. Anda harus menggunakan CoreDNS untuk menemukan titik akhir layanan.

Batasi webhook penerimaan dinamis per sumber daya

Dynamic Admission Webhooks mencakup webhook masuk dan webhook yang bermutasi. Mereka adalah endpoint API yang bukan bagian dari Kubernetes Control Plane yang dipanggil secara berurutan ketika sebuah resource dikirim ke Kubernetes API. Setiap webhook memiliki batas waktu default 10 detik dan dapat meningkatkan jumlah waktu yang dibutuhkan permintaan API jika Anda memiliki beberapa webhook atau salah satu dari mereka batas waktu.

Pastikan webhook Anda sangat tersedia—terutama selama insiden AZ—dan FailurePolicy disetel dengan benar untuk menolak sumber daya atau mengabaikan kegagalan. Jangan panggil webhook saat tidak diperlukan dengan mengizinkan perintah --dry-run kubectl untuk melewati webhook.

```
apiVersion: admission.k8s.io/v1
kind: AdmissionReview
request:
  dryRun: False
```

Webhook yang bermutasi dapat memodifikasi sumber daya secara berurutan. Jika Anda memiliki 5 webhook yang bermutasi dan menyebarkan 50 sumber daya, etcd akan menyimpan semua versi setiap sumber daya hingga pemadatan berjalan—setiap 5 menit—untuk menghapus versi lama sumber daya yang dimodifikasi. Dalam skenario ini ketika etcd menghapus sumber daya yang digantikan, akan ada 200 versi sumber daya yang dihapus dari etcd dan tergantung pada ukuran sumber daya dapat menggunakan ruang yang cukup besar pada host etcd hingga defragmentasi berjalan setiap 15 menit.

Defragmentasi ini dapat menyebabkan jeda di etcd yang dapat memiliki pengaruh lain pada API dan pengontrol Kubernetes. Anda harus menghindari modifikasi sumber daya besar yang sering atau memodifikasi ratusan sumber daya secara berurutan.

Bandingkan beban kerja di beberapa cluster

Jika Anda memiliki dua cluster yang seharusnya memiliki kinerja serupa tetapi tidak, coba bandingkan metrik untuk mengidentifikasi alasannya.

Misalnya, membandingkan latensi cluster adalah masalah umum. Hal ini biasanya disebabkan oleh perbedaan volume permintaan API. Anda dapat menjalankan CloudWatch LogInsight kueri berikut untuk memahami perbedaannya.

```
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
```

Anda dapat menambahkan filter tambahan untuk mempersempitnya. misalnya berfokus pada semua permintaan daftar dari`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
```

Teori Penskalaan Kubernetes

Node vs Tingkat Churn

Seringkali ketika kita membahas skalabilitas Kubernetes, kita melakukannya dalam hal berapa banyak node yang ada dalam satu cluster. Menariknya, ini jarang merupakan metrik yang paling berguna untuk memahami skalabilitas. Misalnya, cluster 5.000 node dengan jumlah pod yang besar tetapi tetap tidak akan memberikan banyak tekanan pada bidang kontrol setelah penyiapan awal. Namun, jika kita mengambil cluster 1.000 node dan mencoba menciptakan 10.000 pekerjaan berumur pendek dalam waktu kurang dari satu menit, itu akan memberikan banyak tekanan berkelanjutan pada bidang kontrol.

Cukup menggunakan jumlah node untuk memahami penskalaan bisa menyesatkan. Lebih baik berpikir dalam hal tingkat perubahan yang terjadi dalam periode waktu tertentu (mari kita gunakan interval 5 menit untuk diskusi ini, karena inilah yang biasanya digunakan kueri Prometheus secara default). Mari kita jelajahi mengapa membingkai masalah dalam hal tingkat perubahan dapat memberi kita gambaran yang lebih baik tentang apa yang harus disetel untuk mencapai skala yang kita inginkan.

Berpikir dalam Pertanyaan Per Detik

Kubernetes memiliki sejumlah mekanisme perlindungan untuk setiap komponen - Kubelet, Scheduler, Kube Controller Manager, dan API server - untuk mencegah kewalahan link berikutnya dalam rantai

Kubernetes. Misalnya, Kubelet memiliki flag untuk membatasi panggilan ke server API pada tingkat tertentu. Mekanisme perlindungan ini umumnya, tetapi tidak selalu, dinyatakan dalam hal kueri yang diizinkan berdasarkan per detik atau QPS.

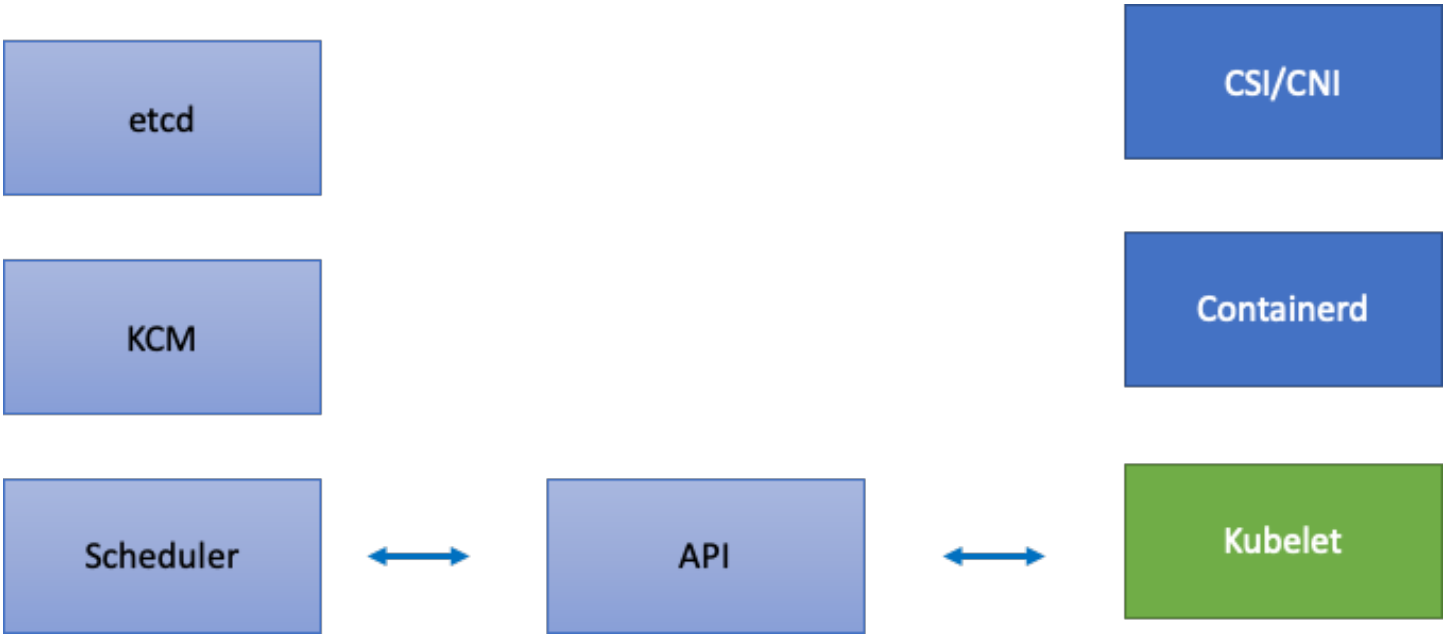
Perhatian besar harus diberikan saat mengubah pengaturan QPS ini. Menghapus satu kemacetan, seperti kueri per detik pada Kubelet akan berdampak pada komponen downstream lainnya. Ini dapat dan akan membanjiri sistem di atas tingkat tertentu, sehingga memahami dan memantau setiap bagian dari rantai layanan adalah kunci untuk berhasil menskalakan beban kerja di Kubernetes.

> **Note**
>
> Server API memiliki sistem yang lebih kompleks dengan pengenalan API Priority and Fairness yang akan kita bahas secara terpisah.

> **Note**
>
> Perhatian, beberapa metrik tampak cocok tetapi sebenarnya mengukur sesuatu yang lain. Sebagai contoh, hanya `kubelet_http_inflight_requests` berkaitan dengan server metrik di Kubelet, bukan jumlah permintaan dari Kubelet ke permintaan apiserver. Ini dapat menyebabkan kita salah mengkonfigurasi flag QPS di Kubelet. Kueri pada log audit untuk Kubelet tertentu akan menjadi cara yang lebih andal untuk memeriksa metrik.

Penskalaan Komponen Terdistribusi

Karena EKS adalah layanan terkelola, mari kita bagi komponen Kubernetes menjadi dua kategori: komponen yang dikelola AWS yang meliputi etcd, Kube Controller Manager, dan Scheduler (di bagian kiri diagram), dan komponen yang dapat dikonfigurasi pelanggan seperti Kubelet, Container Runtime, dan berbagai operator yang memanggil APIs AWS seperti driver Networking and Storage (di bagian kanan diagram). Kami meninggalkan server API di tengah meskipun AWS dikelola, karena pengaturan untuk Prioritas dan Keadilan API dapat dikonfigurasi oleh pelanggan.



Kemacetan Hulu dan Hilir

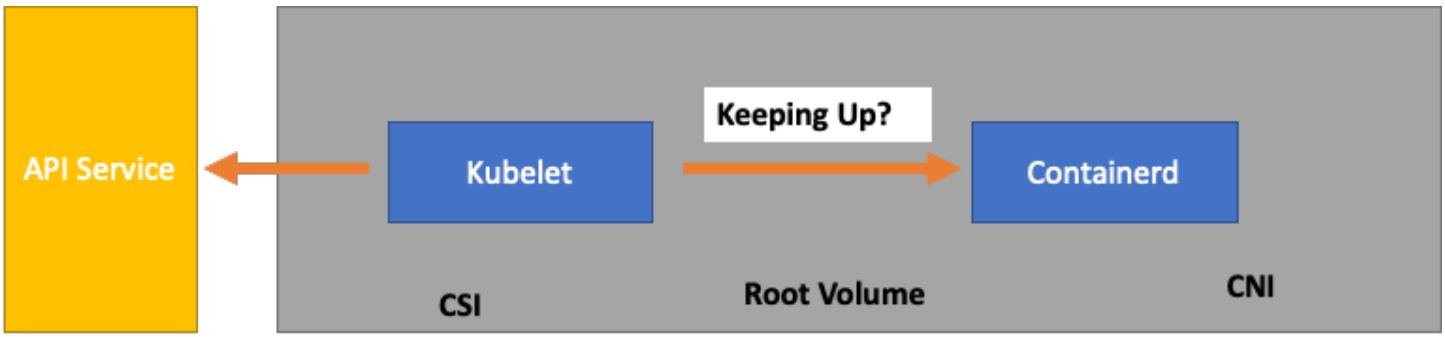
Saat kami memantau setiap layanan, penting untuk melihat metrik di kedua arah untuk mencari kemacetan. Mari kita belajar bagaimana melakukan ini dengan menggunakan Kubelet sebagai contoh. Kubelet berbicara baik ke server API dan runtime container; bagaimana dan apa yang perlu kita pantau untuk mendeteksi apakah salah satu komponen mengalami masalah?

Berapa banyak Pod per Node

Ketika kita melihat angka penskalaan, seperti berapa banyak Pod yang dapat dijalankan pada sebuah node, kita dapat mengambil 110 pod per node yang didukung upstream pada nilai nominal.

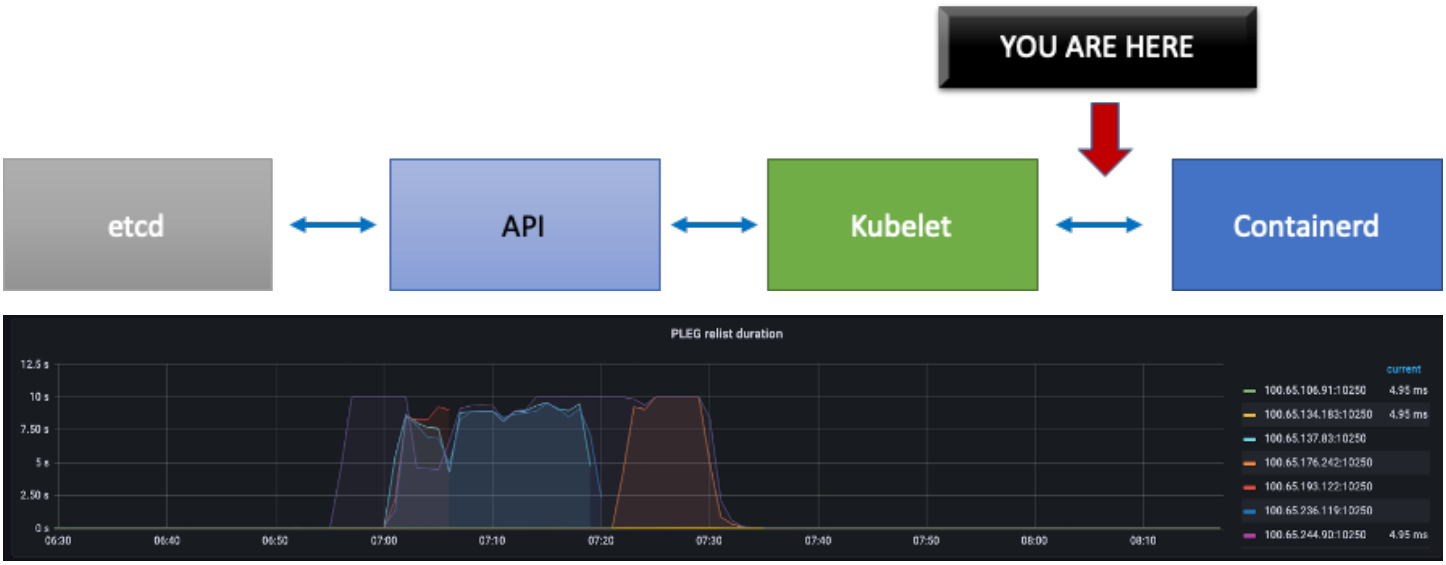
> **Note**
>
> https://kubernetes. io/docs/setup/best-practices/cluster-besar/

Namun, beban kerja Anda kemungkinan lebih kompleks daripada yang diuji dalam uji skalabilitas di Upstream. Untuk memastikan kita dapat melayani jumlah pod yang ingin kita jalankan dalam produksi, mari kita pastikan bahwa Kubelet “mengikuti” runtime Containerd.



Untuk menyederhanakan, Kubelet mendapatkan status pod dari runtime container (dalam kasus kami Containerd). Bagaimana jika kita memiliki terlalu banyak pod yang mengubah status terlalu cepat? Jika tingkat perubahan terlalu tinggi, permintaan [ke runtime kontainer] dapat batas waktu.

> **Note**
>
> Kubernetes terus berkembang, subsistem ini saat ini sedang mengalami perubahan. https://github.com/kubernetes/peningkatan/masalah/3386



Pada grafik di atas, kita melihat garis datar yang menunjukkan bahwa kita baru saja mencapai nilai batas waktu untuk metrik durasi pembuatan peristiwa siklus hidup pod. Jika Anda ingin melihat ini di cluster Anda sendiri, Anda dapat menggunakan sintaks promQL berikut.

```
increase(kubelet_pleg_relist_duration_seconds_bucket{instance="$instance"}
[$__rate_interval])
```

Jika kita menyaksikan perilaku batas waktu ini, kita tahu kita mendorong node melebihi batas yang mampu dilakukannya. Kita perlu memperbaiki penyebab batas waktu sebelum melanjutkan lebih jauh. Ini dapat dicapai dengan mengurangi jumlah pod per node, atau mencari kesalahan yang mungkin menyebabkan volume percobaan ulang yang tinggi (sehingga mempengaruhi laju churn). Take-away yang penting adalah bahwa metrik adalah cara terbaik untuk memahami apakah sebuah node mampu menangani churn rate dari pod yang ditetapkan vs. menggunakan nomor tetap.

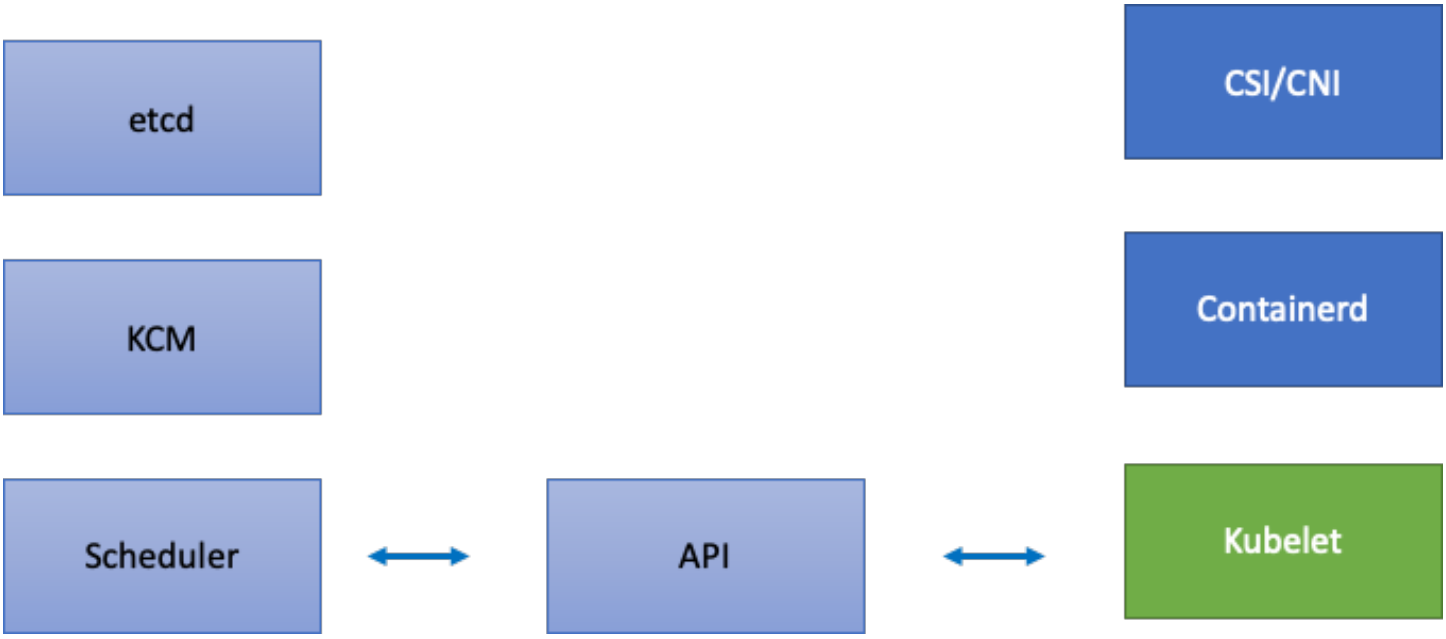
Skala menurut Metrik

Meskipun konsep menggunakan metrik untuk mengoptimalkan sistem sudah lama, konsep ini sering diabaikan saat orang memulai perjalanan Kubernetes mereka. Alih-alih berfokus pada angka tertentu (yaitu 110 pod per node), kami memfokuskan upaya kami untuk menemukan metrik yang membantu kami menemukan kemacetan di sistem kami. Memahami ambang batas yang tepat untuk metrik ini dapat memberi kami tingkat kepercayaan yang tinggi, sistem kami dikonfigurasi secara optimal.

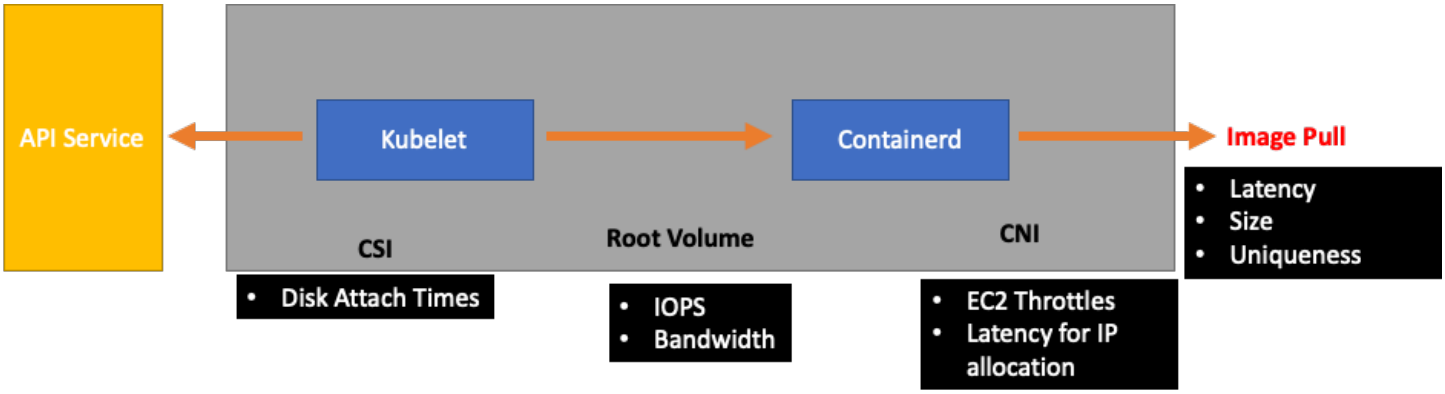
Dampak Perubahan

Pola umum yang bisa membuat kita mendapat masalah adalah berfokus pada metrik pertama atau kesalahan log yang terlihat mencurigakan. Ketika kami melihat bahwa Kubelet sudah habis waktu sebelumnya, kami dapat mencoba hal-hal acak, seperti meningkatkan tarif per detik yang diizinkan untuk dikirim oleh Kubelet, dll. Namun, adalah bijaksana untuk melihat keseluruhan gambaran dari segala sesuatu di hilir kesalahan yang kita temukan terlebih dahulu. Buat setiap perubahan dengan tujuan dan didukung oleh data.

Hilir Kubelet akan menjadi runtime Containerd (kesalahan pod), DaemonSets seperti driver penyimpanan (CSI) dan driver jaringan (CNI) yang berbicara dengan EC2 API, dll.



Mari kita lanjutkan contoh Kubelet kita sebelumnya yang tidak mengikuti runtime. Ada sejumlah titik di mana kita bisa mengemas node dengan sangat padat sehingga memicu kesalahan.



Saat merancang ukuran node yang tepat untuk beban kerja kami, ini adalah easy-to-overlook sinyal yang mungkin memberikan tekanan yang tidak perlu pada sistem sehingga membatasi skala dan kinerja kami.

Biaya Kesalahan yang Tidak Perlu

Pengontrol Kubernetes unggul dalam mencoba lagi ketika kondisi kesalahan muncul, namun ini membutuhkan biaya. Percobaan ulang ini dapat meningkatkan tekanan pada komponen seperti Kube Controller Manager. Ini adalah penyewa penting pengujian skala untuk memantau kesalahan tersebut.

Ketika lebih sedikit kesalahan yang terjadi, lebih mudah menemukan masalah dalam sistem. Dengan secara berkala memastikan bahwa klaster kami bebas dari kesalahan sebelum operasi besar (seperti peningkatan), kami dapat menyederhanakan log pemecahan masalah ketika peristiwa yang tidak terduga terjadi.

Memperluas Pandangan Kami

Dalam cluster skala besar dengan 1.000 node, kami tidak ingin mencari kemacetan satu per satu. Dalam promQL kita dapat menemukan nilai tertinggi dalam kumpulan data menggunakan fungsi yang disebut topk; K menjadi variabel kita menempatkan jumlah item yang kita inginkan. Di sini kita menggunakan tiga node untuk mendapatkan ide apakah semua Kubelet di cluster sudah jenuh. Kami telah melihat latensi hingga saat ini, sekarang mari kita lihat apakah Kubelet membuang peristiwa.

```
topk(3, increase(kubelet_pleg_discard_events{}[$__rate_interval]))
```

Memecah pernyataan ini.

- Kami menggunakan variabel Grafana `$__rate_interval` untuk memastikannya mendapatkan empat sampel yang dibutuhkannya. Ini melewati topik kompleks dalam pemantauan dengan variabel sederhana.
- `topk`beri kami hanya hasil teratas dan angka 3 membatasi hasil tersebut menjadi tiga. Ini adalah fungsi yang berguna untuk metrik luas cluster.
- `{}`beri tahu kami tidak ada filter, biasanya Anda akan memasukkan nama pekerjaan apa pun aturan pengikisan, namun karena nama-nama ini bervariasi, kami akan membiarkannya kosong.

Memisahkan Masalah Menjadi Setengah

Untuk mengatasi kemacetan dalam sistem, kami akan mengambil pendekatan menemukan metrik yang menunjukkan kepada kami ada masalah di hulu atau hilir karena ini memungkinkan kami untuk membagi masalah menjadi dua. Ini juga akan menjadi prinsip inti bagaimana kami menampilkan data metrik kami.

Tempat yang baik untuk memulai dengan proses ini adalah server API, karena memungkinkan kita untuk melihat apakah ada masalah dengan aplikasi klien atau Control Plane.

Kontrol Pemantauan Pesawat

Server API

Saat melihat server API kami, penting untuk diingat bahwa salah satu fungsinya adalah membatasi permintaan masuk untuk mencegah overloading bidang kontrol. Apa yang tampak seperti hambatan di tingkat server API mungkin sebenarnya melindunginya dari masalah yang lebih serius. Kita perlu memperhitungkan pro dan kontra untuk meningkatkan volume permintaan yang bergerak melalui sistem. Untuk menentukan apakah nilai server API harus ditingkatkan, berikut adalah contoh kecil dari hal-hal yang perlu kita perhatikan:

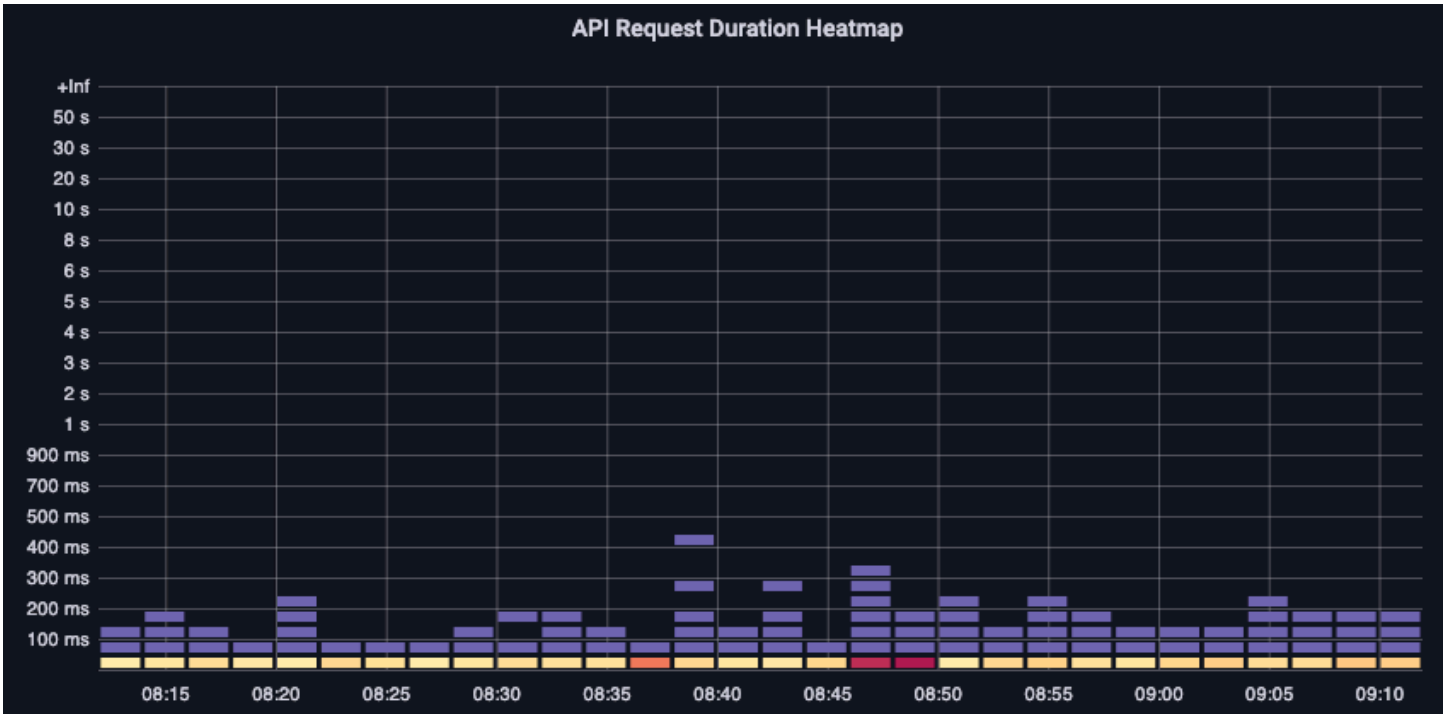
1. Apa latensi permintaan yang bergerak melalui sistem?
2. Apakah latensi itu server API itu sendiri, atau sesuatu yang “hilir” seperti etcd?
3. Apakah kedalaman antrian server API merupakan faktor dalam latensi ini?
4. Apakah antrian API Priority and Fairness (APF) diatur dengan benar untuk pola panggilan API yang kita inginkan?

Dimana masalahnya?

Untuk memulai, kita dapat menggunakan metrik untuk latensi API untuk memberi kita wawasan tentang berapa lama waktu yang dibutuhkan server API untuk permintaan layanan. Mari kita gunakan peta panas PromQL dan Grafana di bawah ini untuk menampilkan data ini.

```
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**
>
> Untuk penulisan mendalam tentang cara memantau server API dengan dasbor API yang digunakan dalam artikel ini, silakan lihat blog berikut



Permintaan ini semua berada di bawah tanda satu detik, yang merupakan indikasi yang baik bahwa pesawat kontrol menangani permintaan secara tepat waktu. Tetapi bagaimana jika bukan itu masalahnya?

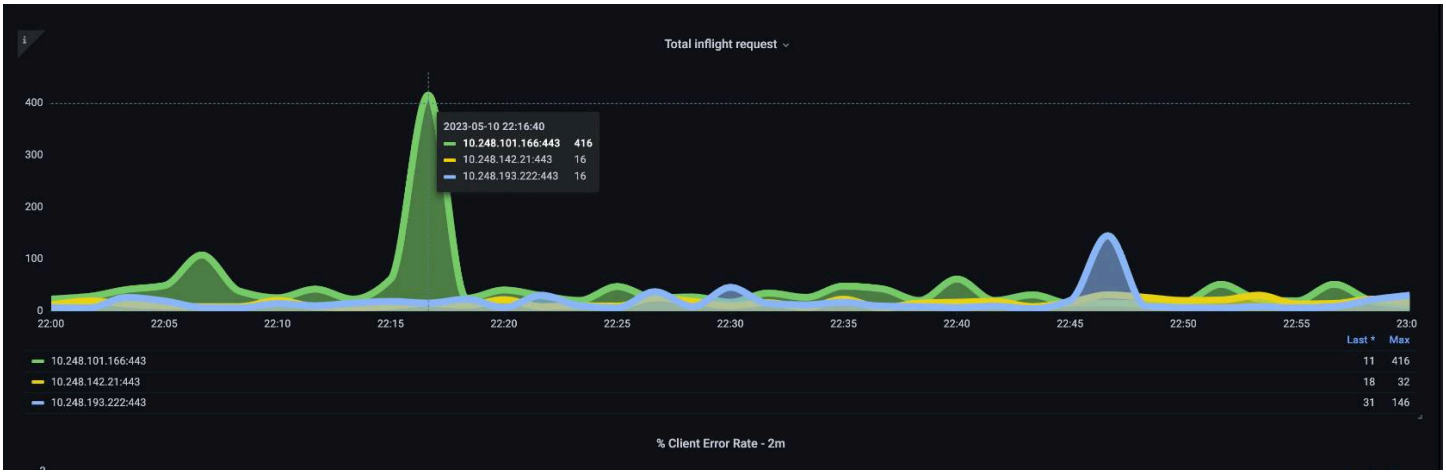
Format yang kita gunakan dalam API Request Duration di atas adalah heatmap. Apa yang baik tentang format heatmap, adalah bahwa ia memberi tahu kita nilai batas waktu untuk API secara default (60 detik). Namun, yang benar-benar perlu kita ketahui adalah pada ambang batas berapa nilai ini harus menjadi perhatian sebelum kita mencapai ambang batas waktu. Untuk pedoman kasar tentang ambang batas yang dapat diterima, kita dapat menggunakan SLO Kubernetes hulu, yang dapat ditemukan di sini

> **Note**
>
> Perhatikan fungsi maksimal pada pernyataan ini? Saat menggunakan metrik yang menggabungkan beberapa server (secara default dua server API di EKS), penting untuk tidak merata-ratakan server tersebut bersama-sama.

Pola lalu lintas asimetris

Bagaimana jika satu server API [pod] dimuat dengan ringan, dan yang lainnya dimuat dengan berat? Jika kita merata-ratakan kedua angka itu bersama-sama, kita mungkin salah menafsirkan apa yang

terjadi. Misalnya, di sini kita memiliki tiga server API tetapi semua beban ada di salah satu server API ini. Sebagai aturan, apa pun yang memiliki banyak server seperti etcd dan server API harus dipecah saat menginvestasikan skala dan masalah kinerja.



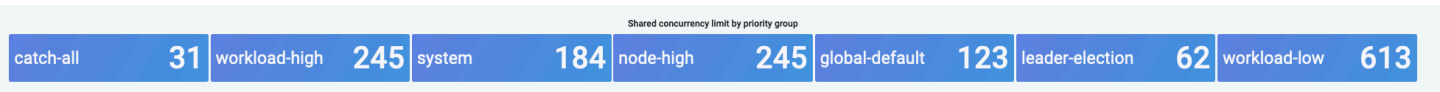
Dengan perpindahan ke Prioritas dan Keadilan API, jumlah total permintaan pada sistem hanyalah satu faktor yang harus diperiksa untuk melihat apakah server API kelebihan langganan. Karena sistem sekarang bekerja dari serangkaian antrian, kita harus melihat apakah ada antrian ini penuh dan apakah lalu lintas untuk antrian itu semakin menurun.

Mari kita lihat antrian ini dengan kueri berikut:

```
max without(instance)(apiserver_flowcontrol_nominal_limit_seats{})
```

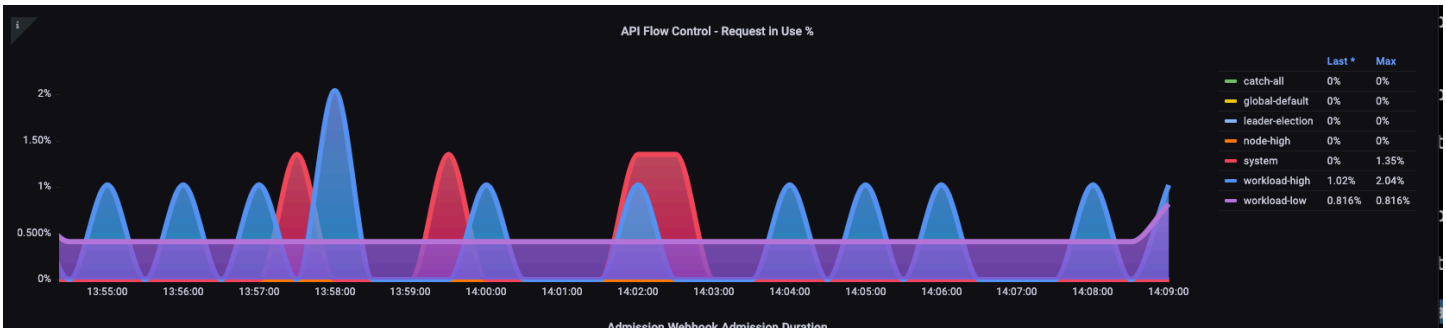
> **Note**
>
> Untuk informasi selengkapnya tentang cara kerja API A&F, silakan lihat panduan praktik terbaik berikut

Di sini kita melihat tujuh grup prioritas berbeda yang datang secara default di cluster



Selanjutnya kita ingin melihat berapa persentase dari kelompok prioritas yang digunakan, sehingga kita dapat memahami apakah tingkat prioritas tertentu sedang jenuh. Permintaan pelambatan di tingkat beban kerja rendah mungkin diinginkan, namun penurunan tingkat pemilihan pemimpin tidak akan terjadi.

Sistem API Priority and Fairness (APF) memiliki sejumlah opsi kompleks, beberapa opsi tersebut dapat memiliki konsekuensi yang tidak diinginkan. Masalah umum yang kita lihat di lapangan adalah meningkatkan kedalaman antrian ke titik itu mulai menambahkan latensi yang tidak perlu. Kita dapat memantau masalah ini dengan menggunakan `apiserver_flowcontrol_current_inqueue_request` metrik. Kami dapat memeriksa tetes menggunakan`apiserver_flowcontrol_rejected_requests_total`. Metrik ini akan menjadi nilai bukan nol jika ada bucket yang melebihi konkurensinya.



Meningkatkan kedalaman antrian dapat membuat Server API menjadi sumber latensi yang signifikan dan harus dilakukan dengan hati-hati. Kami merekomendasikan bersikap bijaksana dengan jumlah antrian yang dibuat. Misalnya, jumlah saham pada sistem EKS adalah 600, jika kita membuat terlalu banyak antrian, ini dapat mengurangi saham dalam antrian penting yang membutuhkan throughput seperti antrian pemimpin-pemilihan atau antrian sistem. Membuat terlalu banyak antrian tambahan dapat membuatnya lebih sulit untuk mengukur antrian tesis dengan benar.

Untuk fokus pada perubahan berdampak sederhana yang dapat Anda lakukan di APF, kami cukup mengambil saham dari ember yang kurang dimanfaatkan dan meningkatkan ukuran ember yang dapat digunakan secara maksimal. Dengan secara cerdas mendistribusikan kembali saham di antara ember ini, Anda dapat membuat penurunan lebih kecil kemungkinannya.

Untuk informasi selengkapnya, kunjungi setelan Prioritas dan Keadilan API di Panduan Praktik Terbaik EKS.

Latensi API vs. etcd

Bagaimana kita bisa menggunakan server API untuk menentukan apakah ada masalah dengan server API, atau masalah yang ada upstream/downstream pada server API, atau kombinasi keduanya. metrics/logs Untuk memahami hal ini dengan lebih baik, mari kita lihat bagaimana API Server dan etcd dapat dihubungkan, dan betapa mudahnya memecahkan masalah sistem yang salah.

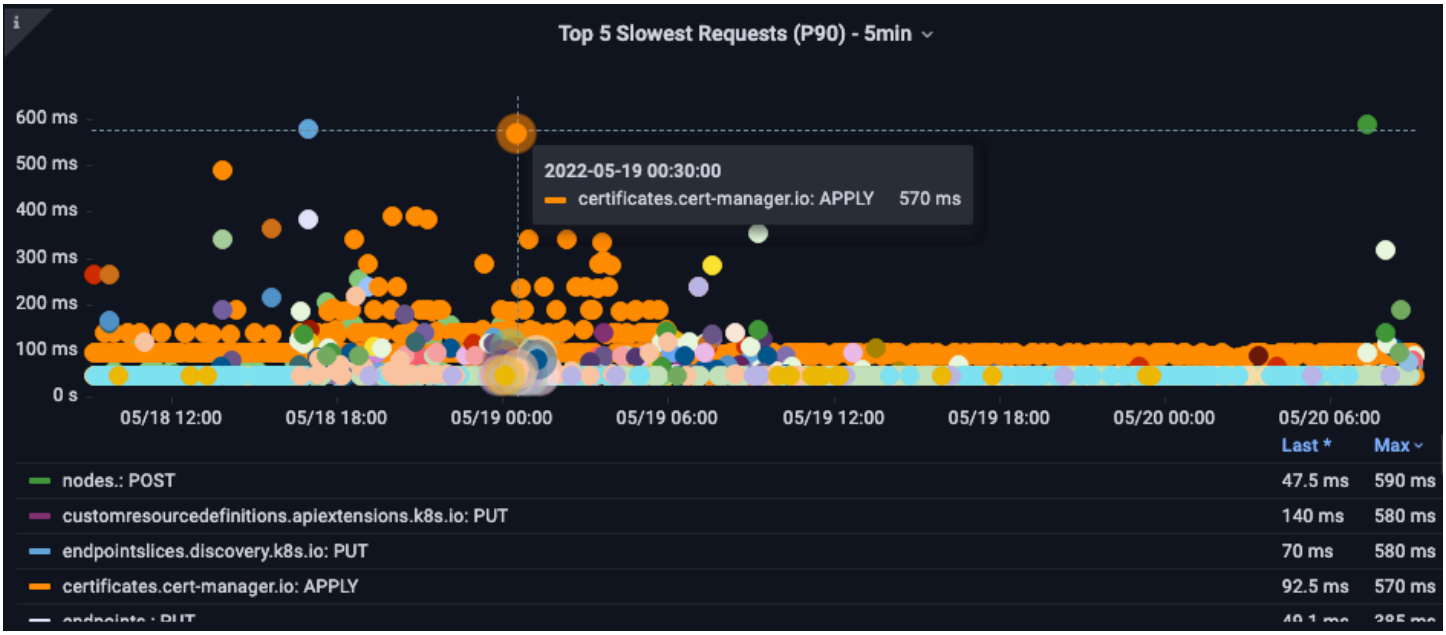
Pada bagan di bawah ini kita melihat latensi server API, tetapi kita juga melihat banyak latensi ini berkorelasi dengan server etcd karena bilah dalam grafik yang menunjukkan sebagian besar latensi pada tingkat etcd. Jika ada 15 detik latensi etcd pada saat yang sama ada 20 detik latensi server API, maka sebagian besar latensi sebenarnya pada level etcd.

Dengan melihat seluruh aliran, kami melihat bahwa adalah bijaksana untuk tidak hanya fokus pada Server API, tetapi juga mencari sinyal yang menunjukkan bahwa etcd berada di bawah tekanan (yaitu penghitung penerapan lambat meningkat). Mampu dengan cepat pindah ke area masalah yang tepat hanya dengan sekilas adalah apa yang membuat dasbor kuat.

> **Note**
>
> Dasbor di bagian dapat ditemukan di Troubleshooting- https://github.com/RiskyAdventure/ - troubleshooter.json Dashboards/blob/main/api

Bidang kontrol vs masalah sisi Klien

Dalam bagan ini kami mencari panggilan API yang membutuhkan waktu paling lama untuk diselesaikan untuk periode itu. Dalam hal ini kita melihat sumber daya khusus (CRD) memanggil fungsi APPLY yang merupakan panggilan paling laten selama kerangka waktu 05:40.



Berbekal data ini, kita dapat menggunakan Ad-Hoc PromQL atau kueri CloudWatch Insights untuk menarik permintaan LIST dari log audit selama jangka waktu tersebut untuk melihat aplikasi mana ini.

Menemukan sumbernya dengan CloudWatch

Metrik paling baik digunakan untuk menemukan area masalah yang ingin kita lihat dan mempersempit kerangka waktu dan parameter pencarian masalah. Setelah kami memiliki data ini, kami ingin beralih ke log untuk waktu dan kesalahan yang lebih rinci. Untuk melakukan ini, kami akan mengubah log kami menjadi metrik menggunakan Wawasan CloudWatch Log.

Misalnya, untuk menyelidiki masalah di atas, kita akan menggunakan query CloudWatch Logs Insights berikut untuk menarik UserAgent dan RequestURI sehingga kita dapat menentukan aplikasi mana yang menyebabkan latensi ini.

> **Note**
>
> Hitungan yang sesuai perlu digunakan agar tidak menarik List/Resync perilaku normal pada Jam Tangan.

```
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
```

Dengan menggunakan kueri ini, kami menemukan dua agen berbeda yang menjalankan sejumlah besar operasi daftar latensi tinggi. Splunk dan CloudWatch agen. Berbekal data, kita dapat membuat keputusan untuk menghapus, memperbarui, atau mengganti pengontrol ini dengan proyek lain.

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**
>
> Untuk detail lebih lanjut tentang hal ini, silakan lihat blog berikut

Penjadwal

Karena instance bidang kontrol EKS dijalankan di akun AWS terpisah, kami tidak akan dapat mengikis komponen tersebut untuk metrik (Server API menjadi pengecualian). Namun, karena kami memiliki akses ke log audit untuk komponen ini, kami dapat mengubah log tersebut menjadi metrik untuk melihat apakah ada sub-sistem yang menyebabkan kemacetan penskalaan. Mari kita gunakan CloudWatch Logs Insights untuk melihat berapa banyak pod yang tidak terjadwal dalam antrean scheduler.

Pod tak terjadwal di log scheduler

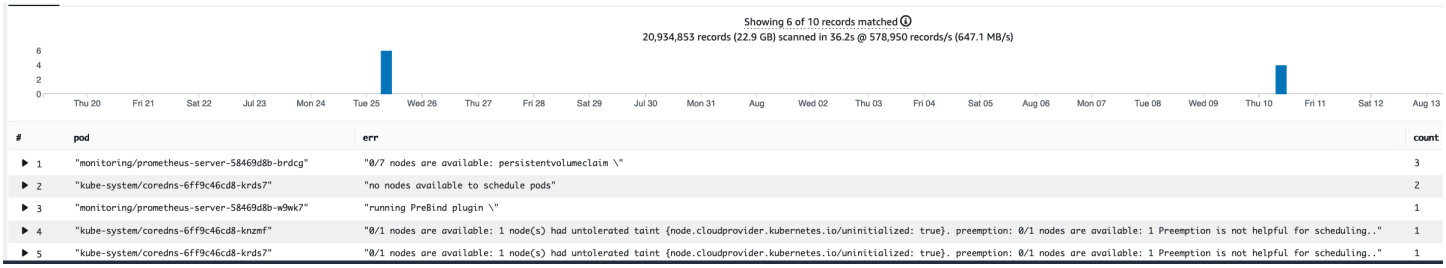
Jika kami memiliki akses untuk mengikis metrik penjadwal secara langsung pada Kubernetes yang dikelola sendiri (seperti Kops), kami akan menggunakan PromQL berikut untuk memahami backlog penjadwal.

```
max without(instance)(scheduler_pending_pods)
```

Karena kami tidak memiliki akses ke metrik di atas di EKS, kami akan menggunakan kueri CloudWatch Logs Insights di bawah ini untuk melihat backlog dengan memeriksa berapa banyak pod yang tidak dapat di-unscheduled selama jangka waktu tertentu. Kemudian kita bisa menyelam lebih jauh ke dalam pesan pada kerangka waktu puncak untuk memahami sifat kemacetan. Misalnya, node tidak berputar cukup cepat, atau pembatas laju di penjadwal itu sendiri.

```
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
```

Di sini kita melihat kesalahan dari penjadwal yang mengatakan pod tidak digunakan karena PVC penyimpanan tidak tersedia.



> **Note**
>
> Pencatatan audit harus diaktifkan pada bidang kontrol untuk mengaktifkan fungsi ini. Ini juga merupakan praktik terbaik untuk membatasi retensi log agar tidak menaikkan biaya dari waktu ke waktu yang tidak perlu. Contoh untuk mengaktifkan semua fungsi logging menggunakan alat EKSCTL di bawah ini.

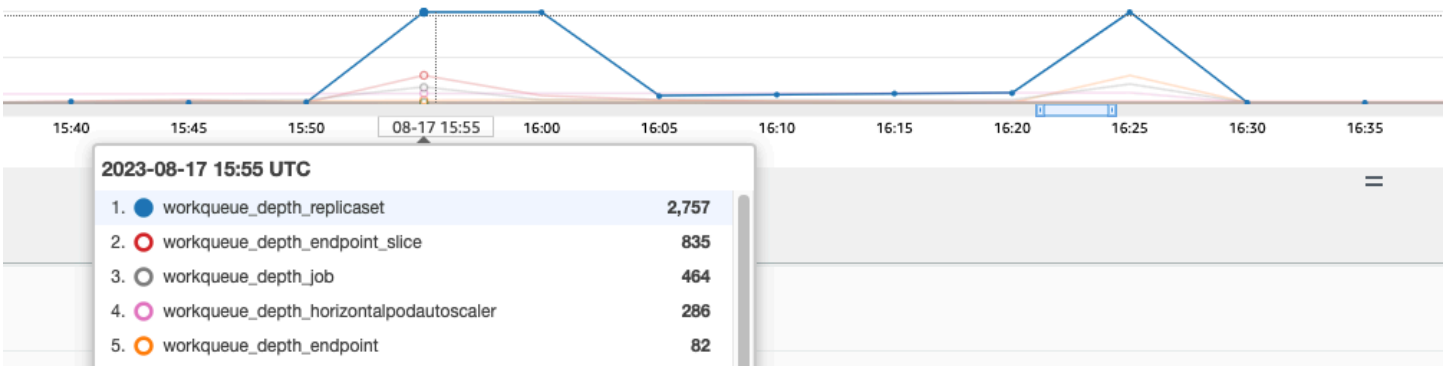
```
cloudWatch:
  clusterLogging:
    enableTypes: ["*"]
    logRetentionInDays: 10
```

Manajer Pengontrol Kube

Kube Controller Manager, seperti semua pengendali lainnya, memiliki batasan berapa banyak operasi yang dapat dilakukan sekaligus. Mari kita tinjau apa beberapa flag tersebut dengan melihat konfigurasi KOPS di mana kita dapat mengatur parameter ini.

```
kubeControllerManager:
  concurrentEndpointSyncs: 5
  concurrentReplicasetSyncs: 5
  concurrentNamespaceSyncs: 10
  concurrentServiceaccountTokenSyncs: 5
  concurrentServiceSyncs: 5
  concurrentResourceQuotaSyncs: 5
  concurrentGcSyncs: 20
  kubeAPIBurst: 30
  kubeAPIQPS: "20"
```

Pengontrol ini memiliki antrian yang terisi selama waktu churn tinggi pada sebuah cluster. Dalam hal ini kita melihat replicaset set controller memiliki backlog besar dalam antreannya.



Kami memiliki dua cara berbeda untuk mengatasi situasi seperti itu. Jika menjalankan dikelola sendiri, kami cukup meningkatkan goroutine bersamaan, namun ini akan berdampak pada etcd dengan memproses lebih banyak data di KCM. Pilihan lainnya adalah mengurangi jumlah objek replicaset yang digunakan `.spec.revisionHistoryLimit` pada penerapan untuk mengurangi jumlah objek replicaset yang dapat kita kembalikan, sehingga mengurangi tekanan pada pengontrol ini.

```
spec:
  revisionHistoryLimit: 2
```

Fitur Kubernetes lainnya dapat disetel atau dimatikan untuk mengurangi tekanan pada sistem churn rate tinggi. Misalnya, jika aplikasi di pod kita tidak perlu berbicara dengan API k8s secara langsung, maka mematikan rahasia yang diproyeksikan ke dalam pod tersebut akan mengurangi beban. ServiceaccountTokenSyncs Ini adalah cara yang lebih diinginkan untuk mengatasi masalah seperti itu jika memungkinkan.

```
kind: Pod
spec:
  automountServiceAccountToken: false
```

Dalam sistem di mana kita tidak bisa mendapatkan akses ke metrik, kita dapat melihat kembali log untuk mendeteksi pertentangan. Jika kami ingin melihat jumlah permintaan yang sedang diproses pada per pengontrol atau tingkat agregat, kami akan menggunakan Kueri Wawasan CloudWatch Log berikut.

Total Volume yang Diproses oleh 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
```

Kunci utama di sini adalah ketika melihat masalah skalabilitas, untuk melihat setiap langkah di jalur (API, penjadwal, KCM, dll) sebelum pindah ke fase pemecahan masalah terperinci. Seringkali dalam produksi Anda akan menemukan bahwa dibutuhkan penyesuaian pada lebih dari satu bagian Kubernetes untuk memungkinkan sistem bekerja dengan kinerja terbaiknya. Sangat mudah untuk secara tidak sengaja memecahkan masalah apa yang hanya gejala (seperti batas waktu simpul) dari leher botol yang jauh lebih besar.

ETCD

etcd menggunakan file yang dipetakan memori untuk menyimpan pasangan nilai kunci secara efisien. Ada mekanisme perlindungan untuk mengatur ukuran ruang memori ini tersedia ditetapkan umumnya pada batas 2, 4, dan 8GB. Lebih sedikit objek dalam database berarti lebih sedikit pembersihan etcd yang perlu dilakukan ketika objek diperbarui dan versi yang lebih lama perlu dibersihkan. Proses pembersihan versi lama dari suatu objek disebut sebagai pemadatan. Setelah sejumlah operasi pemadatan, ada proses selanjutnya yang memulihkan ruang ruang yang dapat digunakan yang disebut defragging yang terjadi di atas ambang batas tertentu atau pada jadwal waktu yang tetap.

Ada beberapa item terkait pengguna yang dapat kita lakukan untuk membatasi jumlah objek di Kubernetes dan dengan demikian mengurangi dampak dari proses pemadatan dan de-fragmentasi. Misalnya, Helm tetap tinggi`revisionHistoryLimit`. Ini membuat objek yang lebih tua seperti ReplicaSets pada sistem untuk dapat melakukan rollback. Dengan menetapkan batas riwayat menjadi 2 kita dapat mengurangi jumlah objek (seperti ReplicaSets) dari sepuluh menjadi dua yang pada gilirannya akan mengurangi beban pada sistem.

```
apiVersion: apps/v1
kind: Deployment
spec:
  revisionHistoryLimit: 2
```

Dari sudut pandang pemantauan, jika lonjakan latensi sistem terjadi dalam pola set yang dipisahkan oleh jam, memeriksa untuk melihat apakah proses defragmentasi ini adalah sumbernya dapat membantu. Kita bisa melihat ini dengan menggunakan CloudWatch Log.

Jika Anda ingin melihat start/end waktu defrag gunakan kueri berikut:

```
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

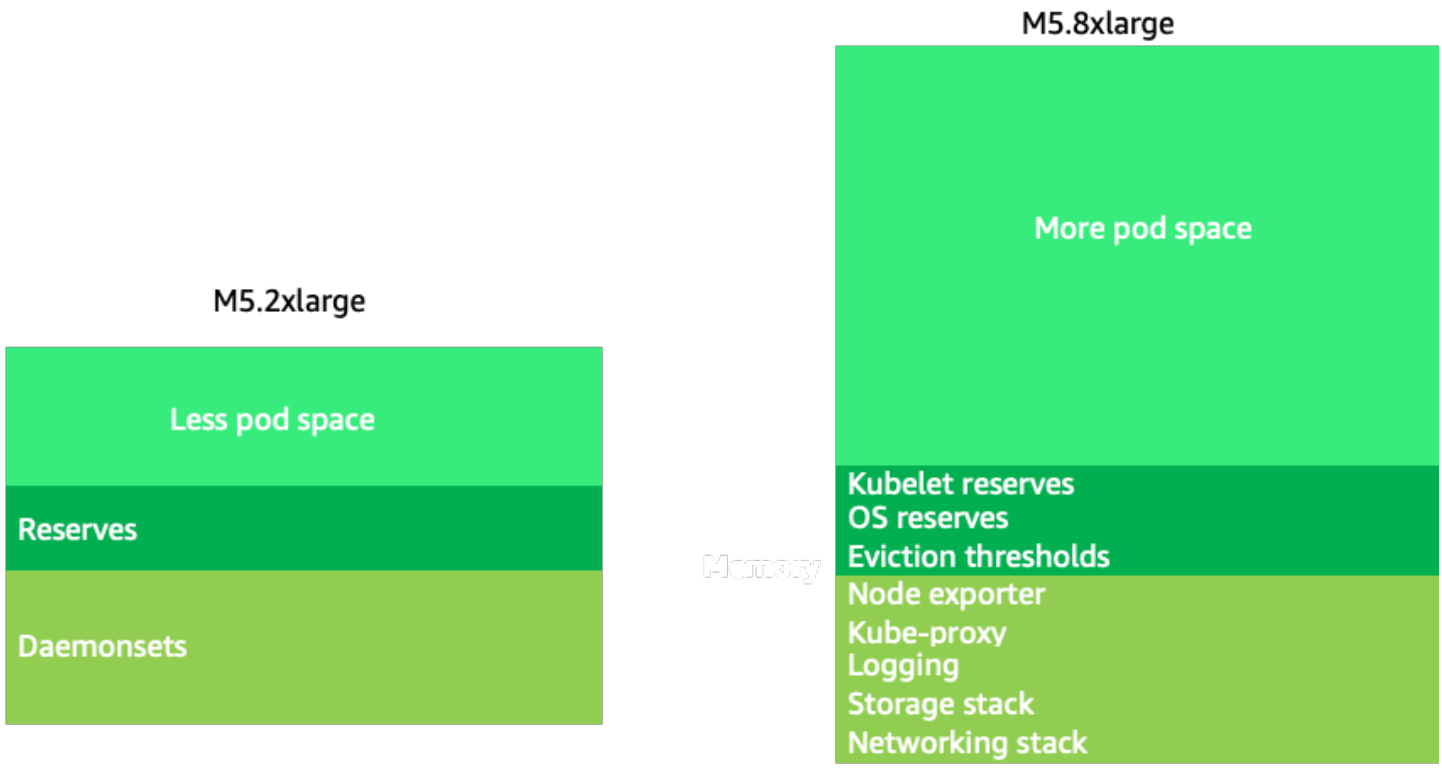
Node dan Efisiensi Beban Kerja

Menjadi efisien dengan beban kerja dan node kami mengurangi complexity/cost sekaligus meningkatkan kinerja dan skala. Ada banyak faktor yang perlu dipertimbangkan ketika merencanakan efisiensi ini, dan paling mudah untuk berpikir dalam hal trade off vs. satu pengaturan praktik terbaik untuk setiap fitur. Mari kita jelajahi pengorbanan ini secara mendalam di bagian berikut.

Pemilihan Node

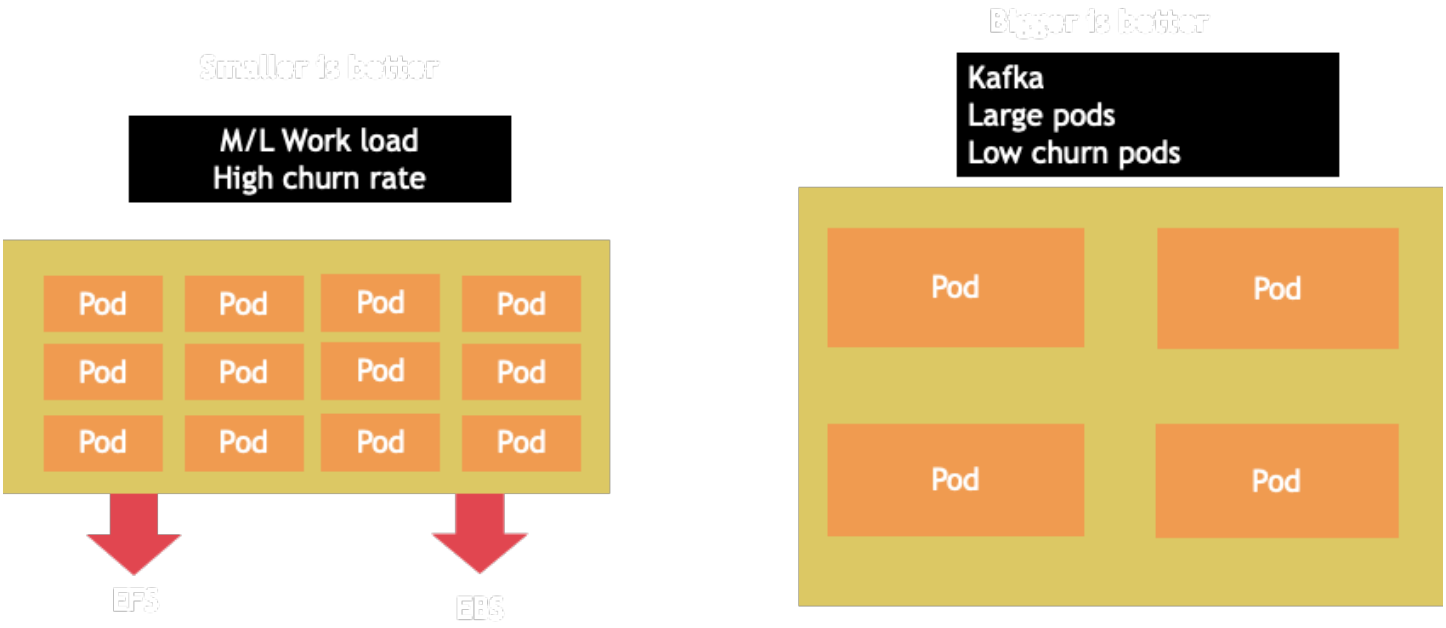
Menggunakan ukuran node yang sedikit lebih besar (4-12xlarge) meningkatkan ruang yang tersedia yang kita miliki untuk menjalankan pod karena fakta itu mengurangi persentase node yang digunakan untuk “overhead” seperti DaemonSetsdan Cadangan untuk komponen sistem. Pada diagram di bawah ini kita melihat perbedaan antara ruang yang dapat digunakan pada sistem 2xlarge vs 8xlarge dengan hanya jumlah moderat. DaemonSets

> **Note**
>
> Karena k8s menskalakan secara horizontal sebagai aturan umum, untuk sebagian besar aplikasi, tidak masuk akal untuk mengambil dampak kinerja node ukuran NUMA, sehingga rekomendasi rentang di bawah ukuran node tersebut.



Ukuran node besar memungkinkan kita untuk memiliki persentase yang lebih tinggi dari ruang yang dapat digunakan per node. Namun, model ini dapat dibawa ke ekstrem dengan mengemas node dengan begitu banyak pod sehingga menyebabkan kesalahan atau menjenuhkan node. Memantau saturasi node adalah kunci untuk berhasil menggunakan ukuran node yang lebih besar.

Pemilihan simpul jarang merupakan one-size-fits-all proposisi. Seringkali yang terbaik adalah membagi beban kerja dengan churn rate yang berbeda secara dramatis ke dalam grup node yang berbeda. Beban kerja batch kecil dengan churn rate tinggi akan paling baik dilayani oleh keluarga instance 4xlarge, sedangkan aplikasi skala besar seperti Kafka yang membutuhkan 8 vCPU dan memiliki churn rate rendah akan lebih baik dilayani oleh keluarga 12xlarge.



> **Note**
>
> Faktor lain yang perlu dipertimbangkan dengan ukuran node yang sangat besar adalah karena CGROUPS tidak menyembunyikan jumlah total vCPU dari aplikasi container. Runtime dinamis seringkali dapat memunculkan sejumlah utas OS yang tidak disengaja, menciptakan latensi yang sulit dipecahkan. Untuk aplikasi ini CPU pinning dianjurkan. Untuk eksplorasi topik yang lebih dalam, silakan lihat https://www.youtube.com video/tonton berikut? v = _ NqtfDy KAqg

Pengepakan Node Bin

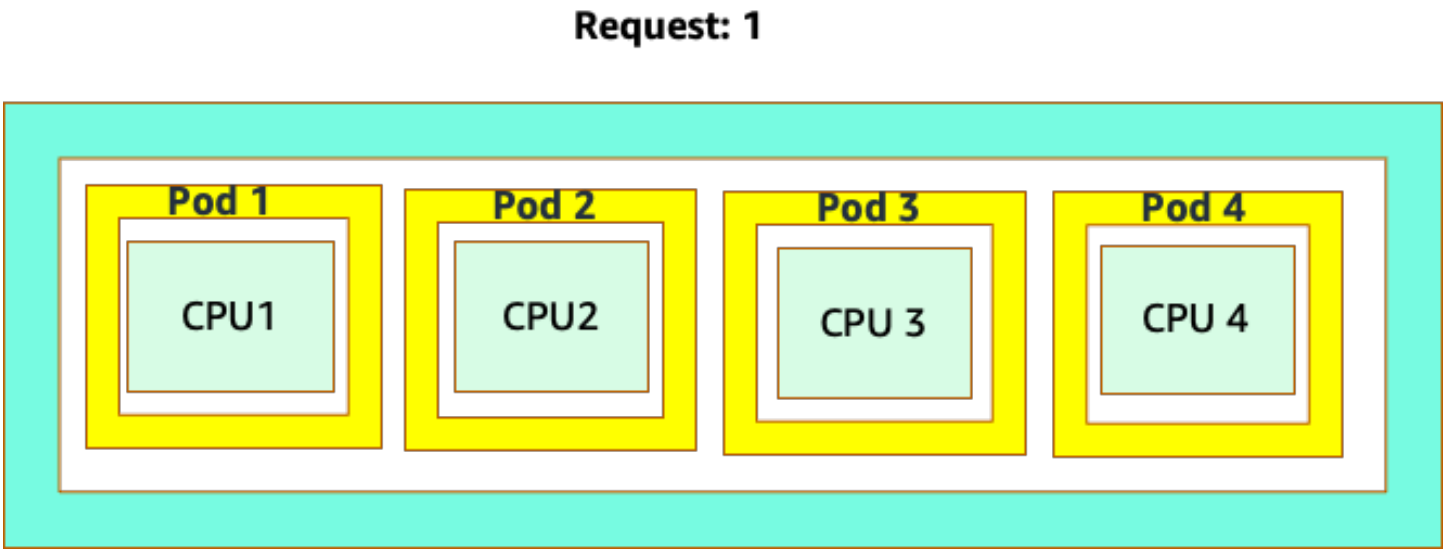
Aturan Kubernetes vs Linux

Ada dua set aturan yang perlu kita perhatikan ketika berhadapan dengan beban kerja di Kubernetes. Aturan Kubernetes Scheduler, yang menggunakan nilai permintaan untuk menjadwalkan pod pada sebuah node, dan kemudian apa yang terjadi setelah pod dijadwalkan, yang merupakan ranah Linux, bukan Kubernetes.

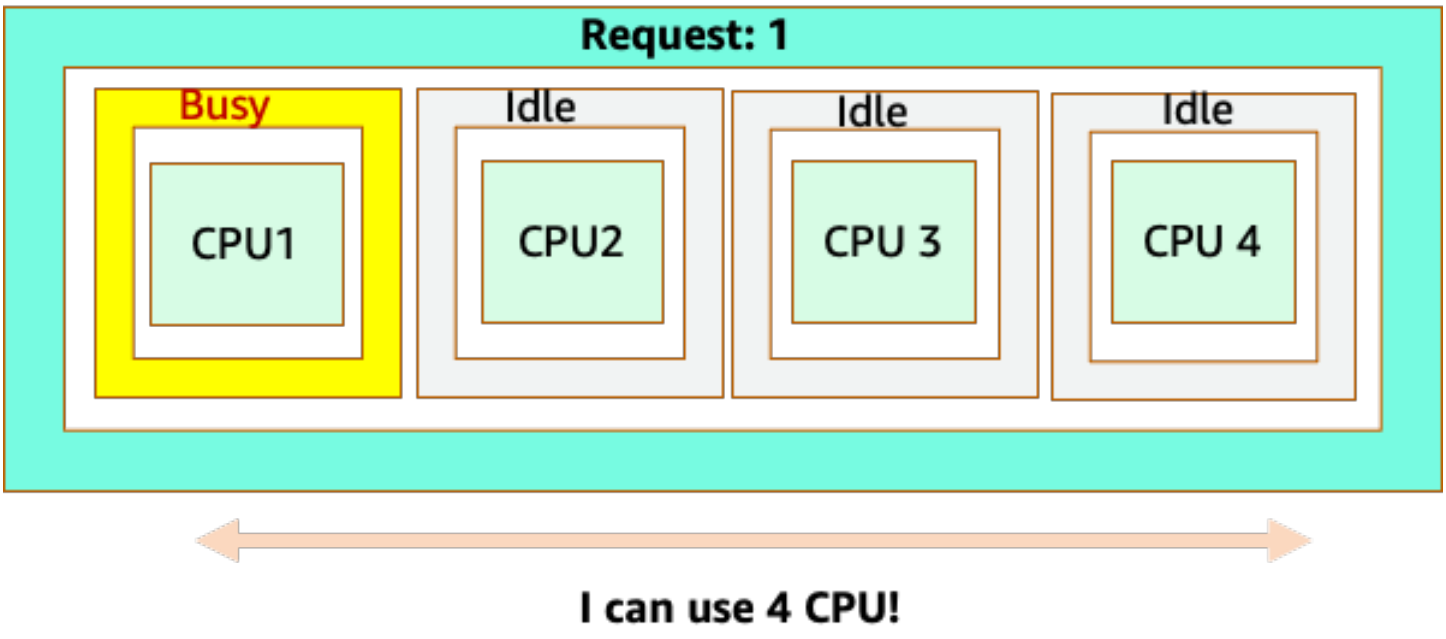
Setelah penjadwal Kubernetes selesai, seperangkat aturan baru mengambil alih, Linux Completely Fair Scheduler (CFS). Kuncinya adalah Linux CFS tidak memiliki konsep inti. Kami akan membahas mengapa berpikir dalam inti dapat menyebabkan masalah besar dengan mengoptimalkan beban kerja untuk skala.

Berpikir di Cores

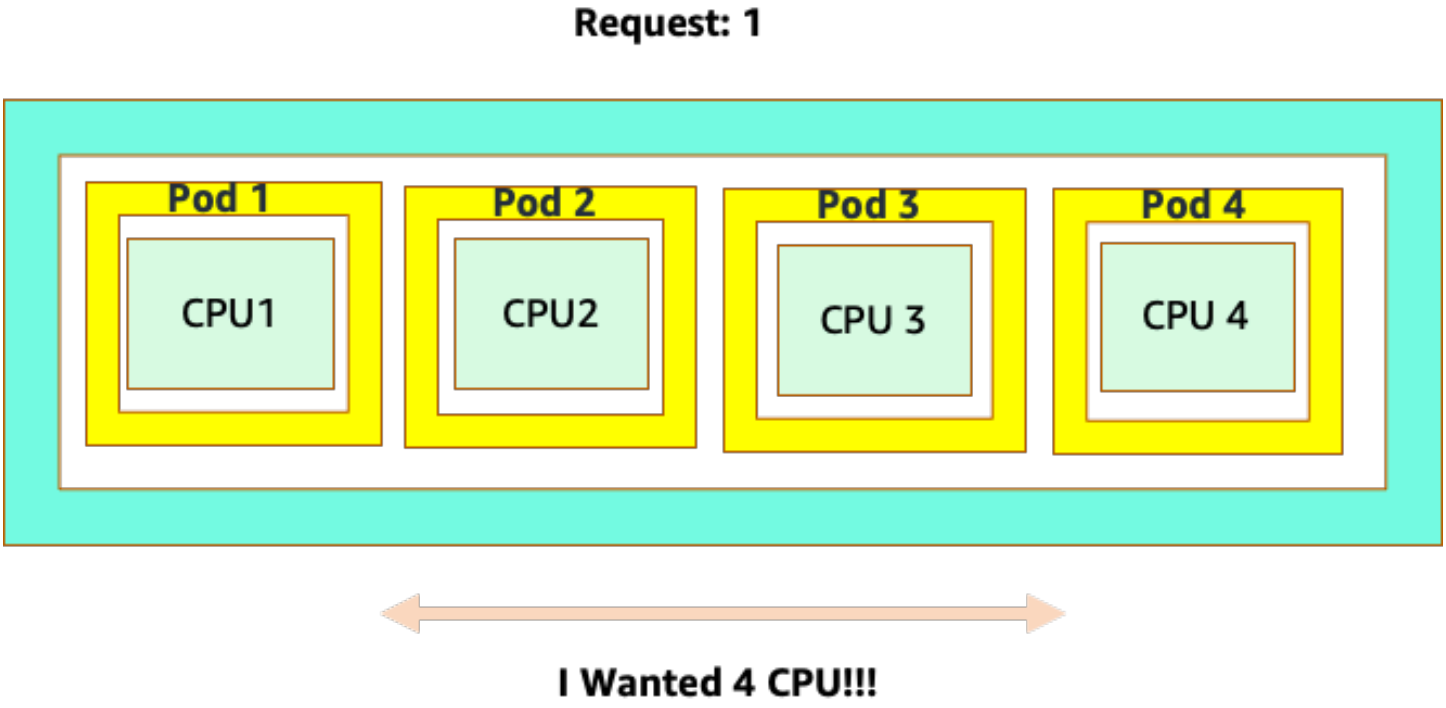
Kebingungan dimulai karena penjadwal Kubernetes memang memiliki konsep inti. Dari perspektif penjadwal Kubernetes jika kita melihat sebuah node dengan 4 pod NGINX, masing-masing dengan permintaan satu set inti, node akan terlihat seperti ini.



Namun, mari kita lakukan eksperimen pemikiran tentang betapa berbedanya ini terlihat dari perspektif CFS Linux. Hal yang paling penting untuk diingat ketika menggunakan sistem CFS Linux adalah: container sibuk (CGROUPS) adalah satu-satunya wadah yang diperhitungkan dalam sistem berbagi. Dalam hal ini, hanya wadah pertama yang sibuk sehingga diizinkan untuk menggunakan semua 4 core pada node.



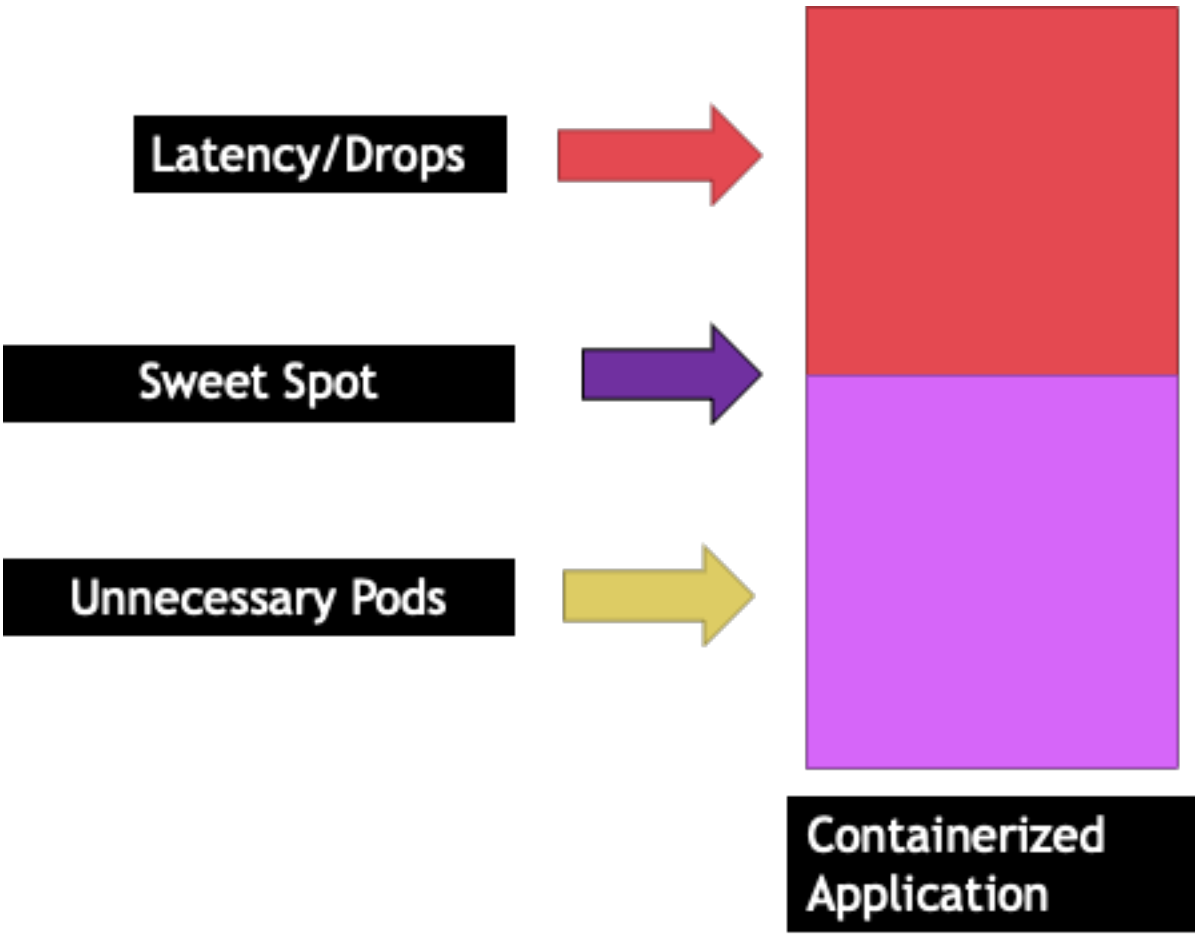
Mengapa ini penting? Katakanlah kami menjalankan pengujian kinerja kami di cluster pengembangan di mana aplikasi NGINX adalah satu-satunya wadah yang sibuk di node itu. Ketika kita memindahkan aplikasi ke produksi, hal berikut akan terjadi: aplikasi NGINX menginginkan 4 vCPU sumber daya, karena semua pod lain di node sibuk, kinerja aplikasi kita dibatasi.



Situasi ini akan membuat kami menambahkan lebih banyak wadah secara tidak perlu karena kami tidak mengizinkan skala aplikasi kami ke “`sweet spot"` mereka. Mari kita jelajahi konsep penting ini `"sweet spot"` dengan sedikit lebih detail.

Aplikasi ukuran yang tepat

Setiap aplikasi memiliki titik tertentu di mana tidak dapat mengambil lalu lintas lagi. Melangkah di atas titik ini dapat meningkatkan waktu pemrosesan dan bahkan menurunkan lalu lintas ketika didorong jauh melampaui titik ini. Ini dikenal sebagai titik saturasi aplikasi. Untuk menghindari masalah penskalaan, kita harus mencoba menskalakan aplikasi sebelum mencapai titik saturasinya. Mari kita sebut titik ini sweet spot.



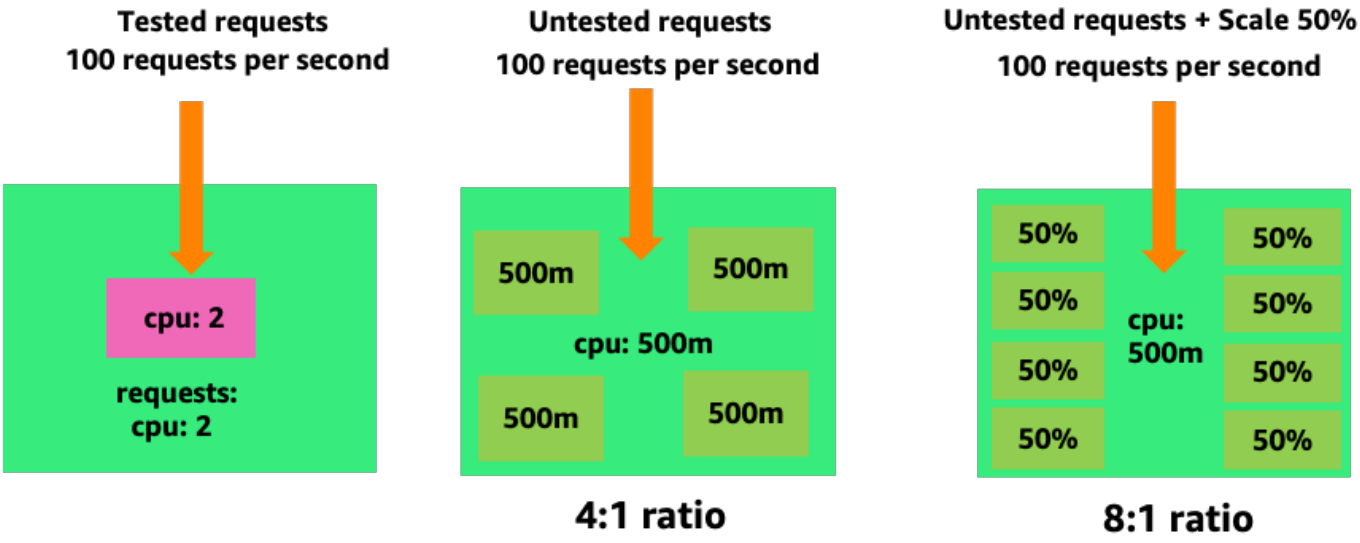
Kita perlu menguji setiap aplikasi kita untuk memahami sweet spotnya. Tidak akan ada panduan universal di sini karena setiap aplikasi berbeda. Selama pengujian ini kami mencoba memahami metrik terbaik yang menunjukkan titik saturasi aplikasi kami. Seringkali, metrik pemanfaatan digunakan untuk menunjukkan aplikasi jenuh tetapi ini dapat dengan cepat menyebabkan masalah penskalaan (Kami akan mengeksplorasi topik ini secara rinci di bagian selanjutnya). Setelah kita memiliki “`sweet spot"` ini, kita dapat menggunakannya untuk meningkatkan skala beban kerja kita secara efisien.

Sebaliknya, apa yang akan terjadi jika kita meningkatkan jauh sebelum sweet spot dan membuat pod yang tidak perlu? Mari kita jelajahi itu di bagian selanjutnya.

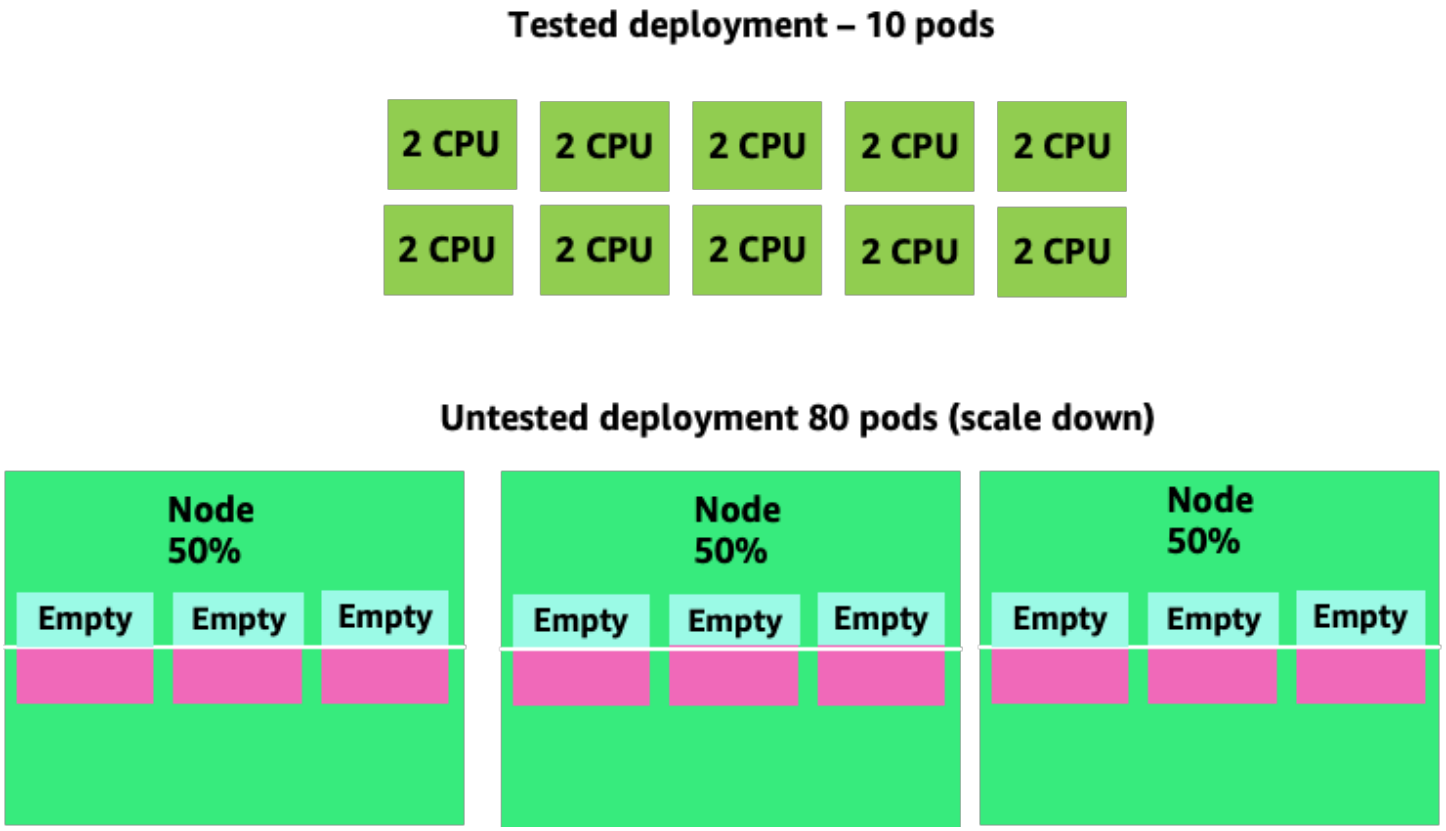
Pod terbentang

Untuk melihat bagaimana membuat pod yang tidak perlu dapat dengan cepat lepas kendali, mari kita lihat contoh pertama di sebelah kiri. Skala vertikal yang benar dari wadah ini membutuhkan CPUs pemanfaatan sekitar dua v saat menangani 100 permintaan per detik. Namun, jika kita kurang memberikan nilai permintaan dengan menyetel permintaan ke setengah inti, kita sekarang

membutuhkan 4 pod untuk setiap satu pod yang sebenarnya kita butuhkan. Memperburuk masalah ini lebih lanjut, jika HPA kami ditetapkan pada default CPU 50%, pod tersebut akan menskalakan setengah kosong, menciptakan rasio 8:1.

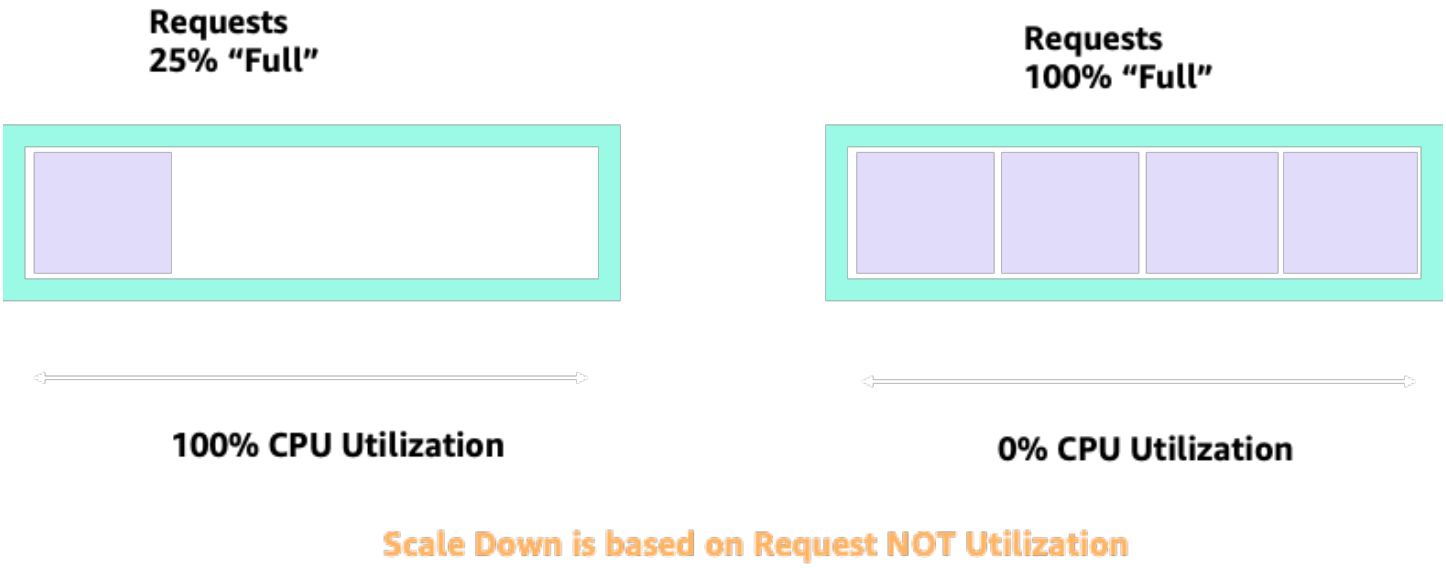


Menskalakan masalah ini, kita dapat dengan cepat melihat bagaimana ini bisa lepas kendali. Penyebaran sepuluh pod yang sweet spotnya disetel secara tidak benar dapat dengan cepat berputar ke 80 pod dan infrastruktur tambahan yang diperlukan untuk menjalankannya.



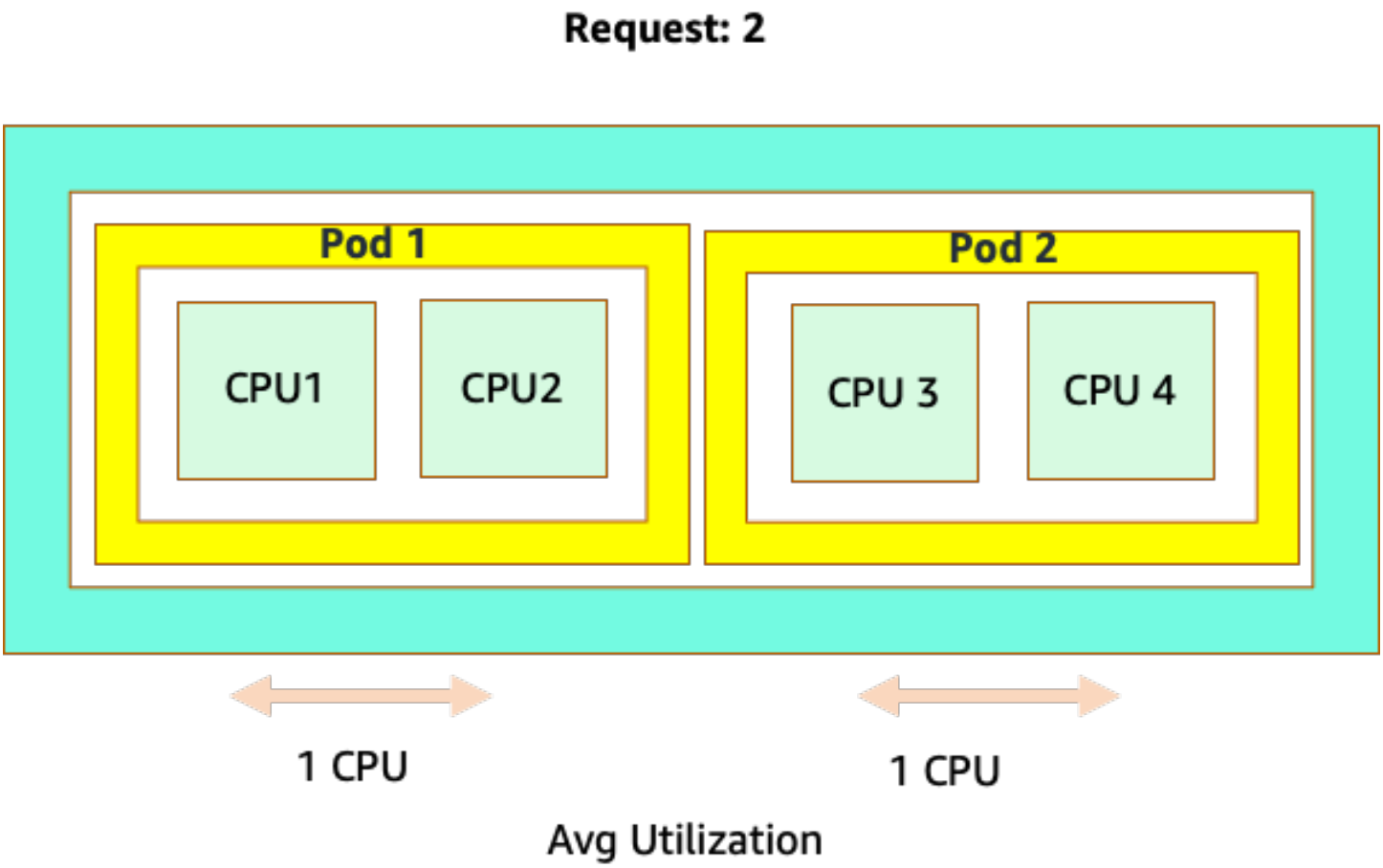
Sekarang setelah kita memahami dampak dari tidak mengizinkan aplikasi beroperasi di sweet spot mereka, mari kita kembali ke level node dan bertanya mengapa perbedaan antara penjadwal Kubernetes dan Linux CFS ini begitu penting?

Saat menskalakan naik dan turun dengan HPA, kita dapat memiliki skenario di mana kita memiliki banyak ruang untuk mengalokasikan lebih banyak pod. Ini akan menjadi keputusan yang buruk karena node yang digambarkan di sebelah kiri sudah pada pemanfaatan CPU 100%. Dalam skenario yang tidak realistis tetapi secara teoritis mungkin, kita dapat memiliki ekstrem lain di mana node kita sepenuhnya penuh, namun pemanfaatan CPU kita nol.

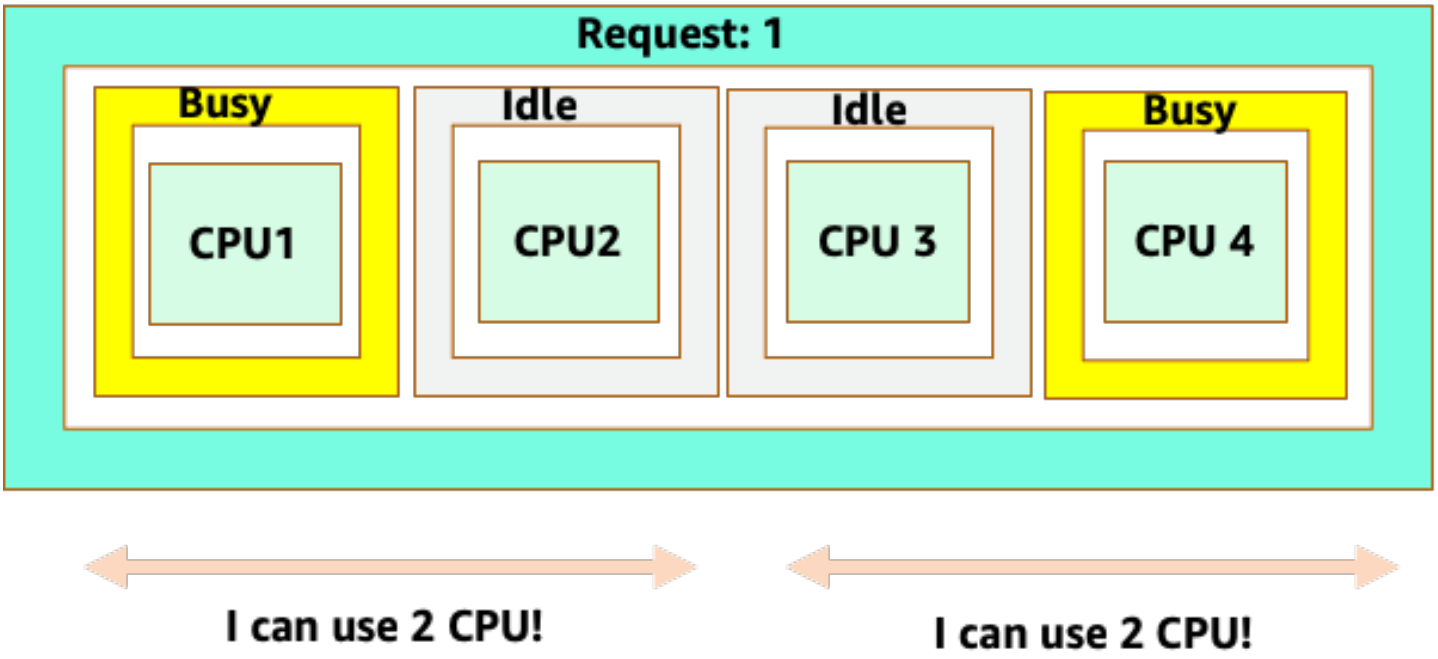


Pengaturan Permintaan

Akan tergoda untuk menetapkan permintaan pada nilai “sweet spot” untuk aplikasi itu, namun ini akan menyebabkan inefisiensi seperti yang digambarkan dalam diagram di bawah ini. Di sini kita telah menetapkan nilai permintaan ke 2 vCPU, namun penggunaan rata-rata pod ini hanya berjalan 1 CPU sebagian besar waktu. Pengaturan ini akan menyebabkan kami membuang 50% siklus CPU kami, yang tidak dapat diterima.



Ini membawa kita ke jawaban kompleks untuk masalah. Pemanfaatan kontainer tidak dapat dipikirkan dalam ruang hampa; seseorang harus memperhitungkan aplikasi lain yang berjalan pada node. Dalam contoh berikut, wadah yang bersifat bursty dicampur dengan dua wadah pemanfaatan CPU rendah yang mungkin dibatasi memori. Dengan cara ini kami mengizinkan kontainer untuk mencapai sweet spot mereka tanpa membebani node.



Konsep penting yang harus diambil dari semua ini adalah bahwa menggunakan konsep penjadwal inti Kubernetes untuk memahami kinerja container Linux dapat menyebabkan pengambilan keputusan yang buruk karena tidak terkait.

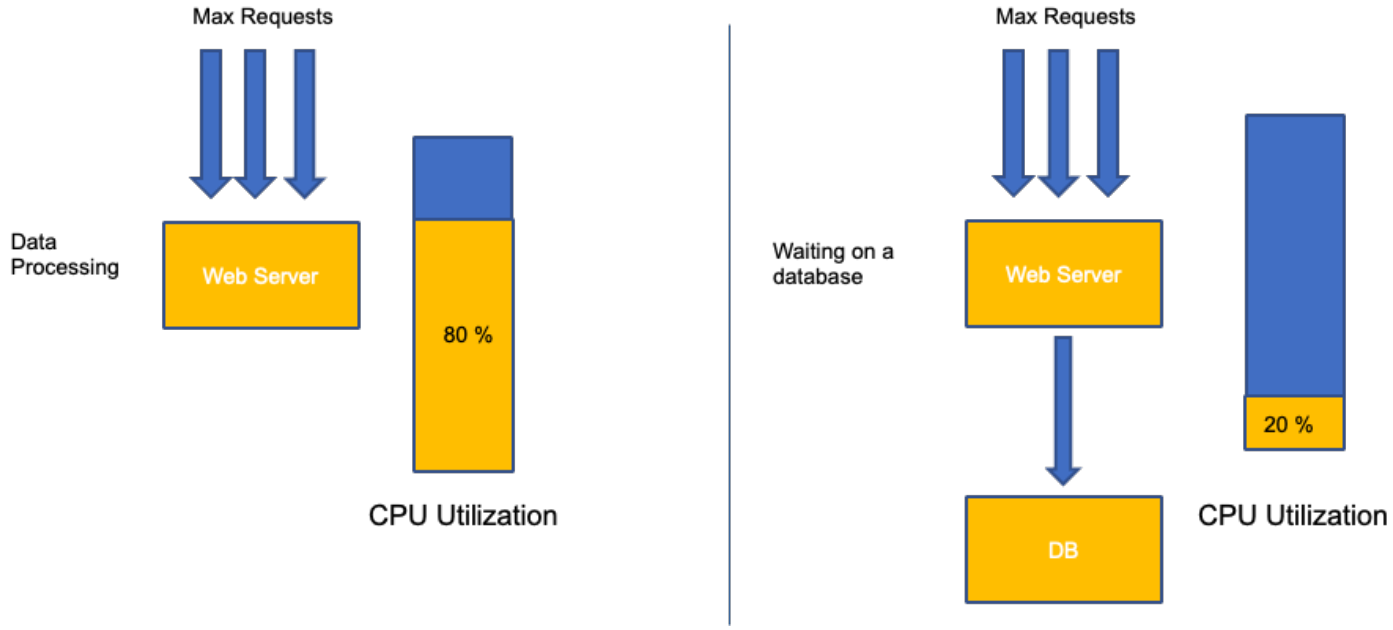
> **Note**
>
> Linux CFS memiliki poin kuatnya. Ini terutama berlaku untuk beban kerja I/O berbasis. Namun, jika aplikasi Anda menggunakan inti penuh tanpa sespan, dan tidak memiliki I/O persyaratan, penyematan CPU dapat menghilangkan banyak kerumitan dari proses ini dan didorong dengan peringatan tersebut.

Pemanfaatan vs Saturasi

Kesalahan umum dalam penskalaan aplikasi hanya menggunakan pemanfaatan CPU untuk metrik penskalaan Anda. Dalam aplikasi yang kompleks, ini hampir selalu merupakan indikator yang buruk bahwa aplikasi sebenarnya jenuh dengan permintaan. Dalam contoh di sebelah kiri, kita melihat semua permintaan kita benar-benar mengenai server web, sehingga pemanfaatan CPU melacak dengan baik dengan saturasi.

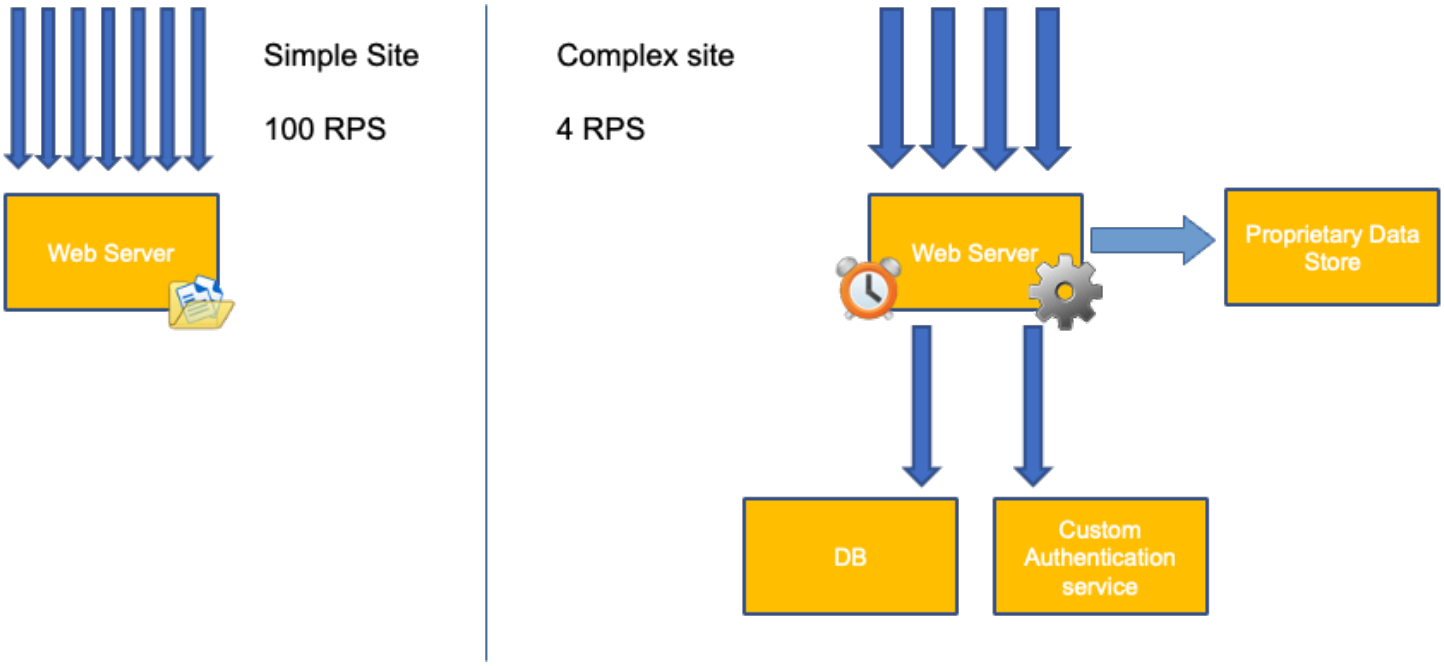
Dalam aplikasi dunia nyata, kemungkinan beberapa permintaan tersebut akan dilayani oleh lapisan database atau lapisan otentikasi, dll. Dalam kasus yang lebih umum ini, perhatikan CPU tidak

melacak dengan saturasi karena permintaan sedang dilayani oleh entitas lain. Dalam hal ini CPU adalah indikator yang sangat buruk untuk saturasi.



Menggunakan metrik yang salah dalam kinerja aplikasi adalah alasan nomor satu untuk penskalaan yang tidak perlu dan tidak dapat diprediksi di Kubernetes. Perhatian besar harus diambil dalam memilih metrik saturasi yang tepat untuk jenis aplikasi yang Anda gunakan. Penting untuk dicatat bahwa tidak ada satu ukuran cocok untuk semua rekomendasi yang dapat diberikan. Bergantung pada bahasa yang digunakan dan jenis aplikasi yang dimaksud, ada beragam metrik untuk saturasi.

Kita mungkin berpikir masalah ini hanya dengan Pemanfaatan CPU, namun metrik umum lainnya seperti permintaan per detik juga dapat jatuh ke dalam masalah yang sama persis seperti yang dibahas di atas. Perhatikan permintaan juga dapat pergi ke lapisan DB, lapisan autentikasi, tidak langsung dilayani oleh server web kami, sehingga metrik yang buruk untuk saturasi sebenarnya dari server web itu sendiri.



Sayangnya tidak ada jawaban yang mudah dalam memilih metrik saturasi yang tepat. Berikut adalah beberapa pedoman yang perlu dipertimbangkan:

- Pahami runtime bahasa Anda - bahasa dengan beberapa utas OS akan bereaksi berbeda dari aplikasi berulir tunggal, sehingga memengaruhi node secara berbeda.
- Pahami skala vertikal yang benar - berapa banyak buffer yang Anda inginkan dalam skala vertikal aplikasi Anda sebelum menskalakan pod baru?
- Metrik apa yang benar-benar mencerminkan saturasi aplikasi Anda - Metrik saturasi untuk Produser Kafka akan sangat berbeda dari aplikasi web yang kompleks.
- Bagaimana semua aplikasi lain pada node saling mempengaruhi - Kinerja aplikasi tidak dilakukan dalam ruang hampa beban kerja lain pada node memiliki dampak besar.

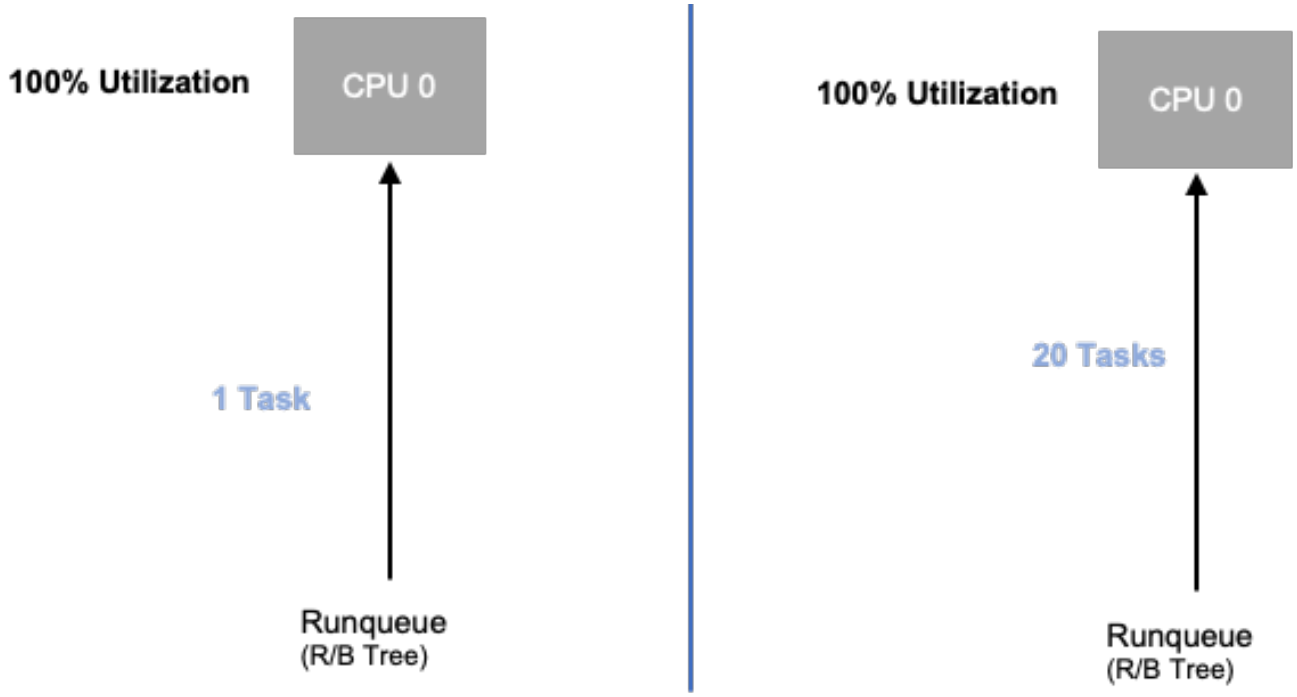
Untuk menutup bagian ini, akan mudah untuk mengabaikan hal di atas sebagai terlalu rumit dan tidak perlu. Seringkali kita mengalami masalah tetapi kita tidak menyadari sifat sebenarnya dari masalah tersebut karena kita melihat metrik yang salah. Pada bagian selanjutnya kita akan melihat bagaimana itu bisa terjadi.

Saturasi Node

Sekarang kita telah mengeksplorasi saturasi aplikasi, mari kita lihat konsep yang sama ini dari sudut pandang simpul. Mari kita ambil dua CPUs yang 100% digunakan untuk melihat perbedaan antara pemanfaatan vs saturasi.

VCPU di sebelah kiri 100% digunakan, namun tidak ada tugas lain yang menunggu untuk dijalankan pada vCPU ini, jadi dalam arti teoritis murni, ini cukup efisien. Sementara itu, kami memiliki 20 aplikasi ulir tunggal yang menunggu untuk diproses oleh vCPU pada contoh kedua. Semua 20 aplikasi sekarang akan mengalami beberapa jenis latensi saat mereka menunggu giliran untuk diproses oleh vCPU. Dengan kata lain, vCPU di sebelah kanan jenuh.

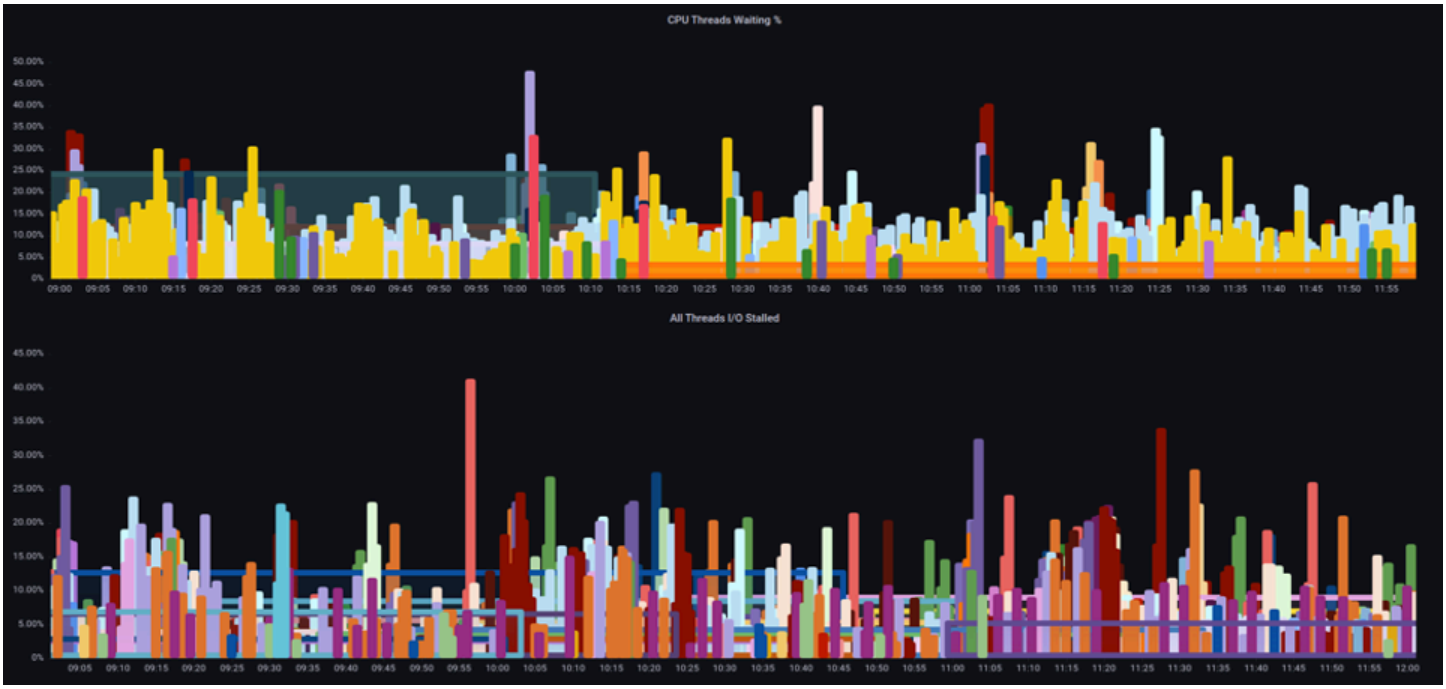
Kita tidak hanya tidak akan melihat masalah ini jika kita hanya melihat pemanfaatan, tetapi kita mungkin menghubungkan latensi ini dengan sesuatu yang tidak terkait seperti jaringan yang akan membawa kita ke jalan yang salah.



Penting untuk melihat metrik saturasi, bukan hanya metrik pemanfaatan saat meningkatkan jumlah total pod yang berjalan pada sebuah node pada waktu tertentu karena kita dapat dengan mudah melewatkan fakta bahwa kita memiliki node yang terlalu jenuh. Untuk tugas ini kita dapat menggunakan metrik informasi pressure stall seperti yang terlihat pada grafik di bawah ini.

PromQL - I/O Terhenti

```
topk(3, ((irate(node_pressure_io_stalled_seconds_total[1m])) * 100))
```



> **Note**
>
> Untuk informasi lebih lanjut tentang metrik tekanan stall, lihat https:// facebookmicrosites.github. io/psi/docs/overview*

Dengan metrik ini, kami dapat mengetahui apakah utas menunggu di CPU, atau bahkan jika setiap utas di kotak macet menunggu sumber daya seperti memori atau I/O. For example, we could see what percentage every thread on the instance was stalled waiting on I/O selama 1 menit.

```
topk(3, ((irate(node_pressure_io_stalled_seconds_total[1m])) * 100))
```

Dengan menggunakan metrik ini, kita dapat melihat pada bagan di atas setiap utas pada kotak terhenti 45% dari waktu menunggu I/O pada tanda air tinggi, yang berarti kita membuang semua siklus CPU pada menit itu. Memahami bahwa ini terjadi dapat membantu kami merebut kembali sejumlah besar waktu vCPU, sehingga membuat penskalaan lebih efisien.

HPA V2

Disarankan untuk menggunakan versi autoscaling/v2 dari HPA API. Versi lama dari HPA API bisa terjebak dalam penskalaan dalam kasus tepi tertentu. Itu juga terbatas pada pod yang hanya berlipat

ganda selama setiap langkah penskalaan, yang menciptakan masalah untuk penerapan kecil yang perlu diskalakan dengan cepat.

AutoScaling/v2 memungkinkan kami lebih banyak fleksibilitas untuk memasukkan beberapa kriteria untuk diskalakan dan memungkinkan kami banyak fleksibilitas saat menggunakan metrik khusus dan eksternal (metrik non K8s).

Sebagai contoh, kita dapat menskalakan pada nilai tertinggi dari tiga nilai (lihat di bawah). Kami menskalakan jika rata-rata pemanfaatan semua pod lebih dari 50%, jika metrik kustom paket per detik dari ingress melebihi rata-rata 1.000, atau objek ingress melebihi 10K permintaan per detik.

> **Note**
>
> Ini hanya untuk menunjukkan fleksibilitas API auto-scaling, kami merekomendasikan terhadap aturan yang terlalu rumit yang sulit dipecahkan dalam produksi.

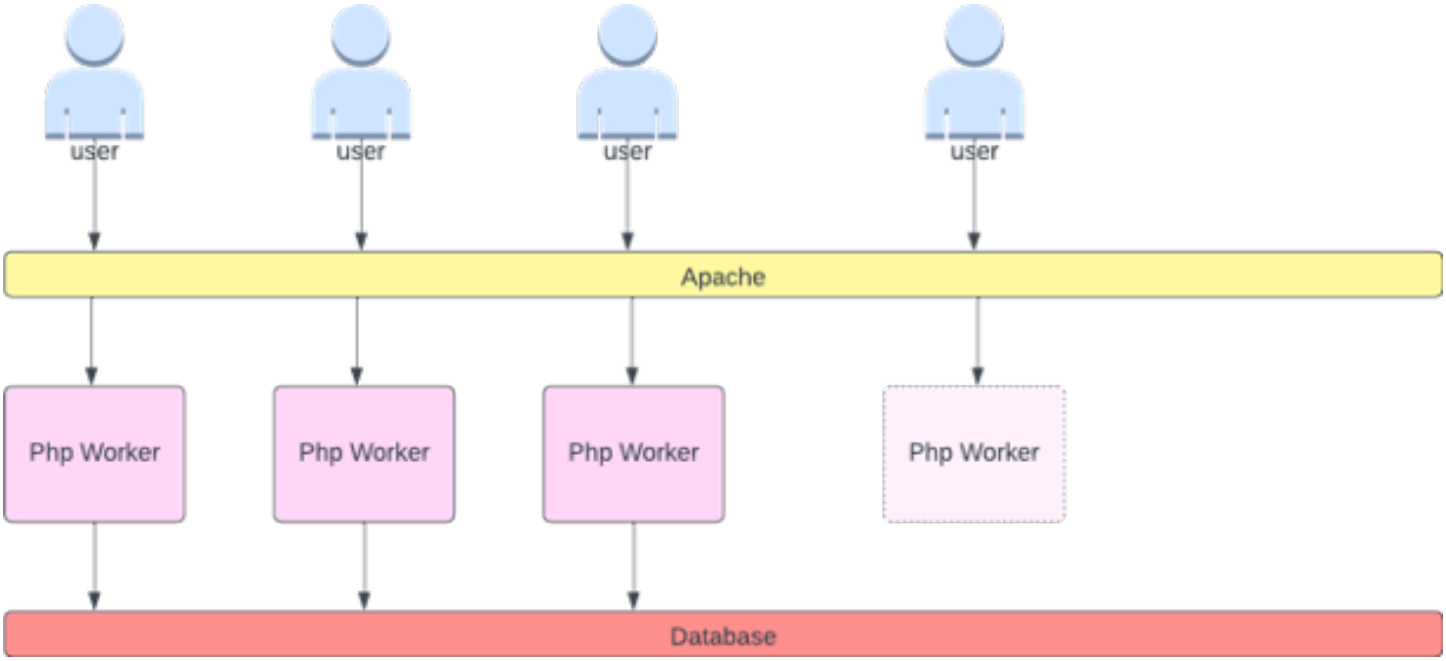
```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: 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
```

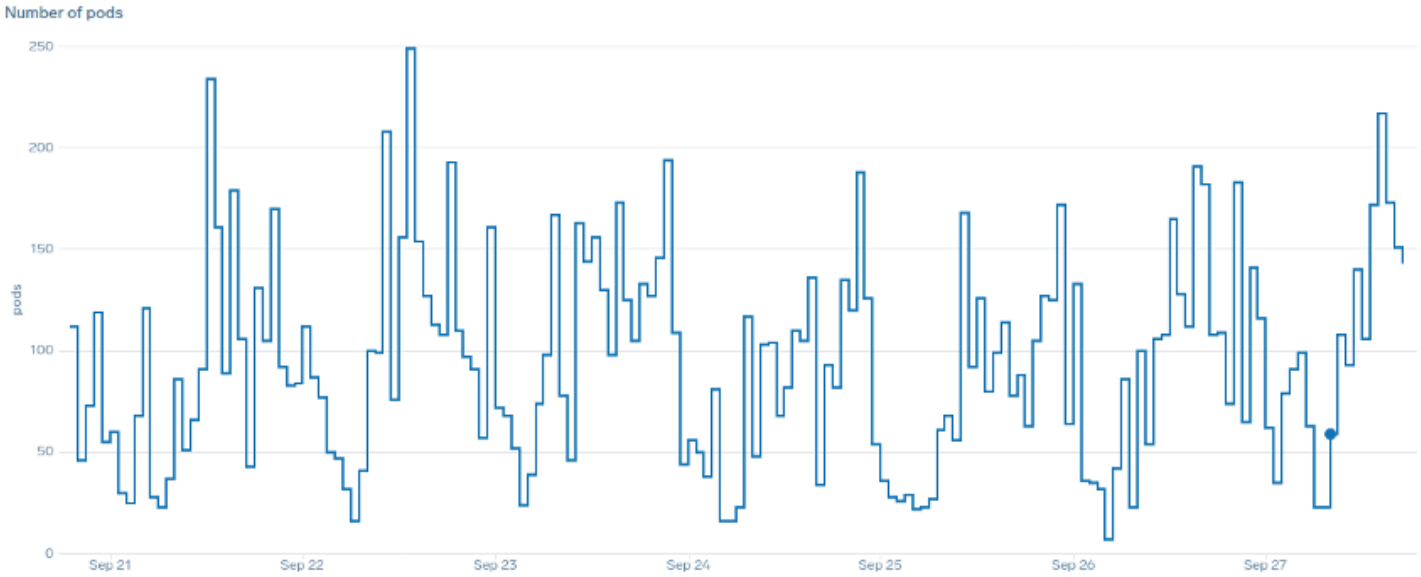
Namun, kami mempelajari bahaya menggunakan metrik tersebut untuk aplikasi web yang kompleks. Dalam hal ini kami akan lebih baik dilayani dengan menggunakan metrik khusus atau eksternal yang secara akurat mencerminkan kejenuhan aplikasi kami vs. pemanfaatan. HPAv2 memungkinkan untuk ini dengan memiliki kemampuan untuk menskalakan sesuai dengan metrik apa pun, namun kami masih perlu menemukan dan mengekspor metrik itu ke Kubernetes untuk digunakan.

Misalnya, kita dapat melihat jumlah antrian thread aktif di Apache. Ini sering membuat profil penskalaan yang “lebih halus” (lebih lanjut tentang istilah itu segera). Jika utas aktif, tidak masalah jika utas itu menunggu di lapisan basis data atau melayani permintaan secara lokal, jika semua utas aplikasi digunakan, itu merupakan indikasi bagus bahwa aplikasi sudah jenuh.

Kita dapat menggunakan kehabisan utas ini sebagai sinyal untuk membuat pod baru dengan kumpulan utas yang tersedia sepenuhnya. Ini juga memberi kita kendali atas seberapa besar buffer yang kita inginkan dalam aplikasi untuk diserap selama masa lalu lintas padat. Misalnya, jika kami memiliki kumpulan utas total 10, penskalaan pada 4 utas yang digunakan vs. 8 utas yang digunakan akan berdampak besar pada buffer yang kami miliki saat menskalakan aplikasi. Pengaturan 4 akan masuk akal untuk aplikasi yang perlu skala cepat di bawah beban berat, di mana pengaturan 8 akan lebih efisien dengan sumber daya kami jika kami memiliki banyak waktu untuk skala karena jumlah permintaan meningkat perlahan vs. tajam dari waktu ke waktu.

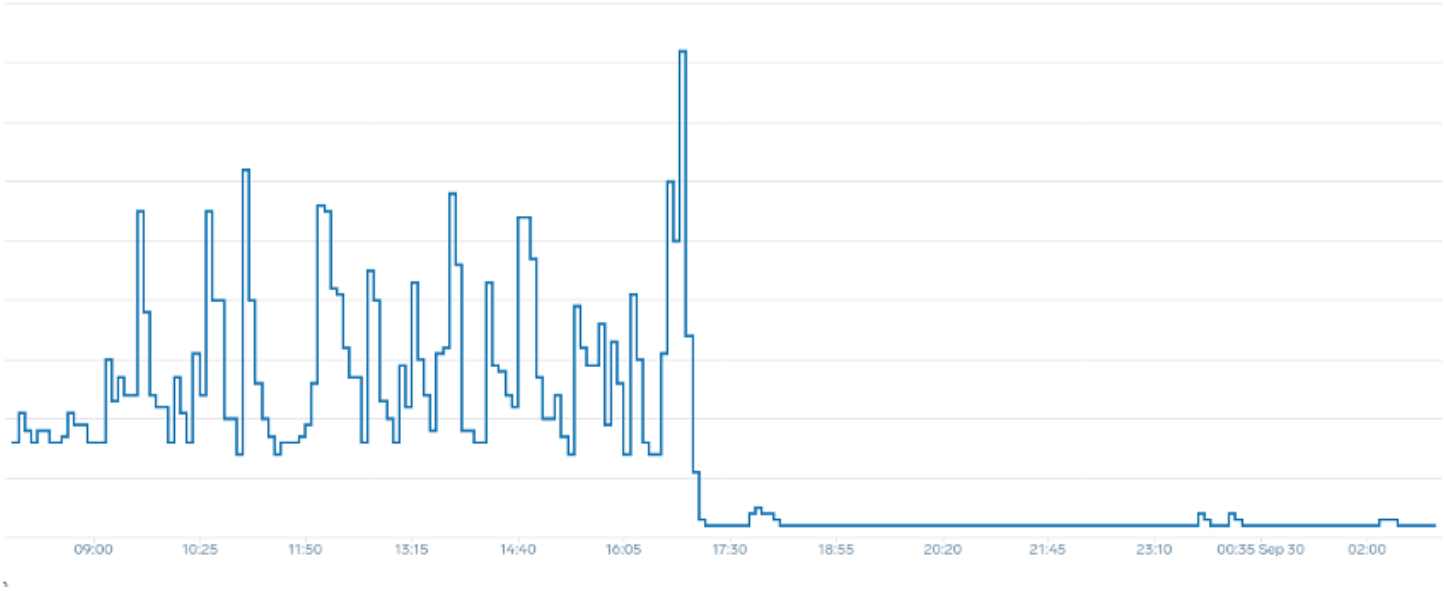


Apa yang kita maksud dengan istilah “halus” dalam hal penskalaan? Perhatikan bagan di bawah ini di mana kita menggunakan CPU sebagai metrik. Pod dalam penerapan ini melonjak dalam waktu singkat dari 50 pod, hingga 250 pod hanya untuk segera menurunkan skala lagi. Ini adalah penskalaan yang sangat tidak efisien adalah penyebab utama churn pada cluster.



Perhatikan bagaimana setelah kita mengubah ke metrik yang mencerminkan sweet spot yang benar dari aplikasi kita (bagian tengah bagan), kita dapat menskalakan dengan lancar. Penskalaan kami sekarang efisien, dan pod kami diizinkan untuk sepenuhnya menskalakan dengan headroom yang kami sediakan dengan menyesuaikan pengaturan permintaan. Sekarang sekelompok kecil pod

melakukan pekerjaan yang dilakukan ratusan pod sebelumnya. Data dunia nyata menunjukkan bahwa ini adalah faktor nomor satu dalam skalabilitas cluster Kubernetes.

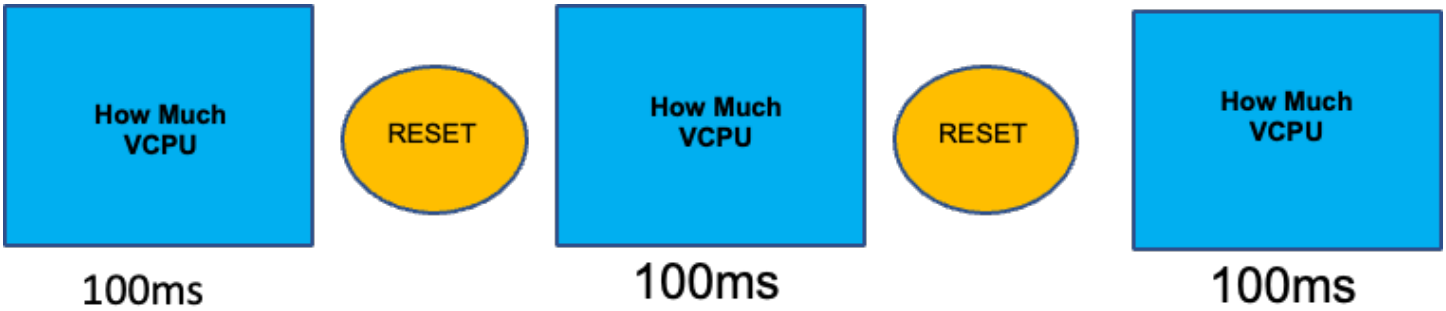


Kesimpulan utamanya adalah pemanfaatan CPU hanya satu dimensi dari kinerja aplikasi dan node. Menggunakan pemanfaatan CPU sebagai indikator kesehatan tunggal untuk node dan aplikasi kami menciptakan masalah dalam penskalaan, kinerja, dan biaya yang semuanya merupakan konsep yang terkait erat. Semakin berkinerja aplikasi dan node, semakin sedikit yang Anda butuhkan untuk menskalakan, yang pada gilirannya menurunkan biaya Anda.

Menemukan dan menggunakan metrik saturasi yang benar untuk menskalakan aplikasi khusus Anda juga memungkinkan Anda untuk memantau dan alarm tentang kemacetan sebenarnya untuk aplikasi itu. Jika langkah kritis ini dilewati, laporan masalah kinerja akan sulit, jika bukan tidak mungkin, untuk dipahami.

Mengatur Batas CPU

Untuk melengkapi bagian ini tentang topik yang disalahpahami, kami akan membahas batas CPU. Singkatnya, batas adalah metadata yang terkait dengan wadah yang memiliki penghitung yang mengatur ulang setiap 100 ms. Ini membantu Linux melacak berapa banyak sumber daya CPU yang digunakan di seluruh node oleh wadah tertentu dalam periode waktu 100 ms.



Kesalahan umum dengan menetapkan batas adalah dengan asumsi bahwa aplikasi berulir tunggal dan hanya berjalan pada vCPU “`ditugaskan”. Pada bagian di atas kami mengetahui bahwa CFS tidak menetapkan inti, dan pada kenyataannya wadah yang menjalankan kumpulan utas besar akan menjadwalkan semua vCPU yang tersedia di kotak.

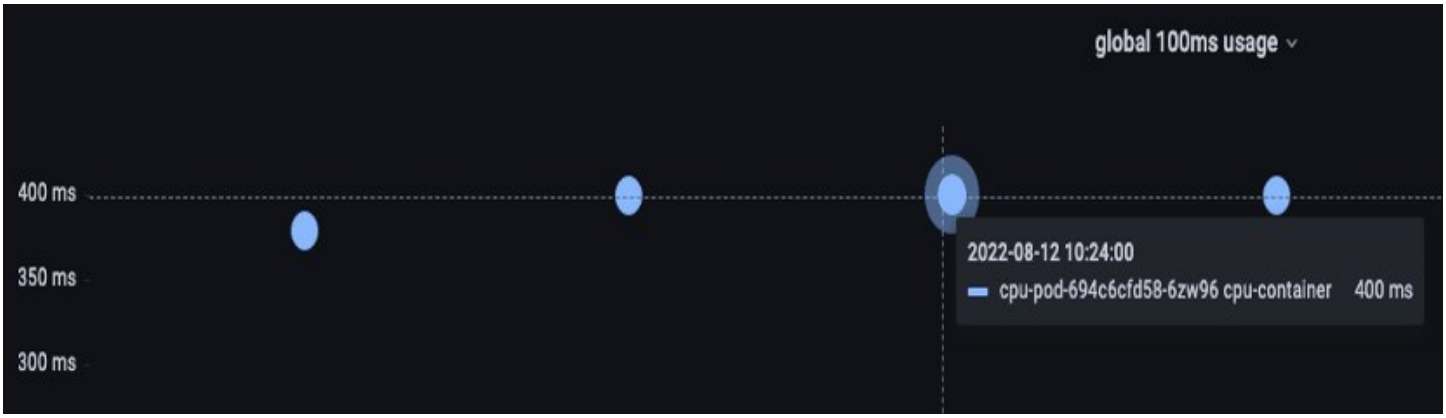
Jika 64 utas OS berjalan di 64 core yang tersedia (dari perspektif node Linux), kami akan membuat tagihan total waktu CPU yang digunakan dalam periode 100ms cukup besar setelah waktu berjalan pada semua 64 core tersebut ditambahkan. Karena ini mungkin hanya terjadi selama proses pengumpulan sampah, sangat mudah untuk melewatkan sesuatu seperti ini. Inilah sebabnya mengapa perlu menggunakan metrik untuk memastikan kami memiliki penggunaan yang benar dari waktu ke waktu sebelum mencoba menetapkan batas.

Untungnya, kami memiliki cara untuk melihat dengan tepat berapa banyak vCPU yang digunakan oleh semua utas dalam suatu aplikasi. Kami akan menggunakan metrik `container_cpu_usage_seconds_total` untuk tujuan ini.

Karena logika pelambatan terjadi setiap 100ms dan metrik ini adalah metrik per detik, kami akan mempromQL untuk mencocokkan periode 100ms ini. Jika Anda ingin menyelam jauh ke dalam karya pernyataan PromQL ini, silakan lihat blog berikut.

Kueri PromQL:

```
topk(3, max by (pod, container)(rate(container_cpu_usage_seconds_total{image!="",
 instance="$instance"}[$__rate_interval]))) / 10
```



Begitu kita merasa memiliki nilai yang tepat, kita dapat membatasi produksi. Kemudian menjadi perlu untuk melihat apakah aplikasi kita sedang dibatasi karena sesuatu yang tidak terduga. Kita bisa melakukan ini dengan melihat `container_cpu_throttled_seconds_total`

```
topk(3, max by (pod, container)(rate(container_cpu_cfs_throttled_seconds_total{image!
=``""``, instance=``"$instance"``}[$__rate_interval]))) / 10
```

Memori

Alokasi memori adalah contoh lain di mana mudah untuk membingungkan perilaku penjadwalan Kubernetes untuk perilaku Linux. CGroup Ini adalah topik yang lebih bernuansa karena ada perubahan besar dalam cara CGroup v2 menangani memori di Linux dan Kubernetes telah mengubah sintaksnya untuk mencerminkan hal ini; baca blog ini untuk detail lebih lanjut.

Tidak seperti permintaan CPU, permintaan memori tidak digunakan setelah proses penjadwalan selesai. Ini karena kita tidak dapat mengompres memori di CGroup v1 dengan cara yang sama dengan CPU. Itu membuat kita hanya memiliki batas memori, yang dirancang untuk bertindak sebagai brankas yang gagal untuk kebocoran memori dengan menghentikan pod sepenuhnya. Ini

adalah proposisi gaya semua atau tidak sama sekali, namun kami sekarang telah diberi cara baru untuk mengatasi masalah ini.

Pertama, penting untuk dipahami bahwa pengaturan jumlah memori yang tepat untuk wadah tidaklah mudah seperti yang terlihat. Sistem file di Linux akan menggunakan memori sebagai cache untuk meningkatkan kinerja. Cache ini akan tumbuh seiring waktu, dan mungkin sulit untuk mengetahui berapa banyak memori yang bagus untuk dimiliki untuk cache tetapi dapat direklamasi tanpa dampak signifikan terhadap kinerja aplikasi. Hal ini sering mengakibatkan salah menafsirkan penggunaan memori.

Memiliki kemampuan untuk “mengompres” memori adalah salah satu driver utama di belakang CGroup v2. Untuk sejarah lebih lanjut tentang mengapa CGroup V2 diperlukan, silakan lihat presentasi Chris Down di LISA21 mana dia membahas mengapa tidak dapat mengatur memori minimum dengan benar adalah salah satu alasan yang mendorongnya untuk membuat CGroup v2 dan metrik penghentian tekanan.

Untungnya, Kubernetes sekarang memiliki konsep dan di bawah. `memory.min` `memory.high` `requests.memory` Ini memberi kami opsi untuk melepaskan memori cache ini secara agresif untuk digunakan wadah lain. Setelah wadah mencapai batas tinggi memori, kernel dapat secara agresif merebut kembali memori kontainer itu hingga nilai yang ditetapkan pada. `memory.min` Dengan demikian memberi kita lebih banyak fleksibilitas ketika sebuah node berada di bawah tekanan memori.

Pertanyaan kuncinya adalah, nilai apa yang `memory.min` harus ditetapkan? Di sinilah metrik penghentian tekanan memori ikut bermain. Kita dapat menggunakan metrik ini untuk mendeteksi memori “meronta-ronta” pada tingkat kontainer. Kemudian kita dapat menggunakan pengontrol seperti fbtax untuk mendeteksi nilai yang benar `memory.min` dengan mencari thrashing memori ini, dan secara dinamis mengatur nilai ke pengaturan ini. `memory.min`

Ringkasan

Untuk meringkas bagian ini, mudah untuk menggabungkan konsep-konsep berikut:

- Pemanfaatan dan Saturasi
- Aturan kinerja Linux dengan logika Penjadwal Kubernetes

Perhatian besar harus diberikan untuk menjaga konsep-konsep ini terpisah. Kinerja dan skala terkait pada tingkat yang dalam. Penskalaan yang tidak perlu menciptakan masalah kinerja, yang pada gilirannya menciptakan masalah penskalaan.

Kubernetes Hulu SLOs

Amazon EKS menjalankan kode yang sama dengan rilis Kubernetes hulu dan memastikan bahwa kluster EKS beroperasi dalam yang SLOs ditentukan oleh komunitas Kubernetes. Kubernetes Scalability Special Interest Group (SIG) mendefinisikan tujuan skalabilitas dan menyelidiki kemacetan dalam kinerja melalui dan. SLIs SLOs

SLIs adalah bagaimana kami mengukur sistem seperti metrik atau ukuran yang dapat digunakan untuk menentukan seberapa “baik” sistem berjalan, misalnya meminta latensi atau hitungan. SLOs tentukan nilai yang diharapkan saat sistem berjalan “baik”, misalnya latensi permintaan tetap kurang dari 3 detik. Kubernetes SLOs dan SLIs fokus pada kinerja komponen Kubernetes dan sepenuhnya independen dari Amazon EKS Service SLAs yang berfokus pada ketersediaan endpoint kluster EKS.

Kubernetes memiliki sejumlah fitur yang memungkinkan pengguna untuk memperluas sistem dengan add-on atau driver khusus, seperti driver CSI, webhook masuk, dan auto-scaler. Ekstensi ini dapat secara drastis memengaruhi kinerja klaster Kubernetes dengan berbagai cara, yaitu webhook penerimaan dengan `failurePolicy=Ignore` dapat menambahkan latensi ke permintaan API K8s jika target webhook tidak tersedia. SIG Kubernetes Scalability mendefinisikan skalabilitas menggunakan kerangka kerja “you promise, we promise”:

> Jika Anda berjanji untuk: - mengonfigurasi cluster Anda dengan benar - gunakan fitur ekstensibilitas “secara wajar” - simpan beban di cluster dalam batas yang disarankan
>
> maka kami berjanji bahwa skala cluster Anda, yaitu: - semua SLOs puas.

Kubernetes SLOs

Kubernetes SLOs tidak memperhitungkan semua plugin dan batasan eksternal yang dapat memengaruhi klaster, seperti penskalaan node pekerja atau webhook masuk. Ini SLOs berfokus pada komponen Kubernetes dan memastikan bahwa tindakan dan sumber daya Kubernetes beroperasi sesuai harapan. SLOs Bantuan pengembang Kubernetes memastikan bahwa perubahan pada kode Kubernetes tidak menurunkan kinerja untuk seluruh sistem.

Kuberntes Scalability SIG mendefinisikan SLO/ resmi berikut. SLIs Tim Amazon EKS secara teratur menjalankan tes skalabilitas pada kluster EKS SLOs/SLIs untuk memantau penurunan kinerja saat perubahan dibuat dan versi baru dirilis.

Tujuan	Definisi	SLO
Latensi permintaan API (bermutasi)	Latensi pemrosesan panggilan API yang bermutasi untuk objek tunggal untuk setiap pasangan (sumber daya, kata kerja), diukur sebagai persentil ke-99 selama 5 menit terakhir	Dalam instalasi Kubernetes default, untuk setiap pasangan (sumber daya, kata kerja), tidak termasuk sumber daya virtual dan agregat dan Definisi Sumber Daya Kustom, persentil ke-99 per hari klaster <= 1s
Latensi permintaan API (hanya-baca)	Latensi pemrosesan panggilan API hanya-baca non-strea ming untuk setiap pasangan (sumber daya, ruang lingkup), diukur sebagai persentil ke-99 selama 5 menit terakhir	Dalam instalasi Kubernetes default, untuk setiap pasangan (sumber daya, cakupan), tidak termasuk sumber daya virtual dan agregat dan Definisi Sumber Daya Kustom, persentil ke-99 per hari klaster: (a) <= 1 detik jika (b) <= 30 detik sebalikny a (jika atau) `scope=res ource scope=namespace scope=cluster`
Latensi startup pod	Latensi startup pod stateless yang dapat dijadwalkan, tidak termasuk waktu untuk menarik gambar dan menjalankan kontainer init, diukur dari stempel waktu pembuatan pod hingga saat semua kontainer nya dilaporkan sebagai dimulai dan diamati melalui jam tangan, diukur sebagai persentil ke-99 selama 5 menit terakhir	Dalam instalasi Kubernetes default, persentil ke-99 per hari cluster <= 5s

Latensi Permintaan API

`kube-apiserver`Telah `--request-timeout` didefinisikan sebagai secara `1m0s` default, yang berarti permintaan dapat berjalan hingga satu menit (60 detik) sebelum waktunya habis dan dibatalkan. Yang SLOs didefinisikan untuk Latensi dipecah oleh jenis permintaan yang sedang dibuat, yang dapat bermutasi atau hanya-baca:

Bermutasi

Mutasi permintaan di Kubernetes membuat perubahan pada sumber daya, seperti kreasi, penghapusan, atau pembaruan. Permintaan ini mahal karena perubahan tersebut harus ditulis ke backend etcd sebelum objek yang diperbarui dikembalikan. Etcd adalah penyimpanan nilai kunci terdistribusi yang digunakan untuk semua data klaster Kubernetes.

Latensi ini diukur sebagai persentil ke-99 selama 5 menit untuk pasangan (resource, verb) sumber daya Kubernetes, misalnya ini akan mengukur latensi untuk permintaan Create Pod dan Update Node request. Latensi permintaan harus <= 1 detik untuk memenuhi SLO.

Hanya baca

Permintaan hanya-baca mengambil satu sumber daya (seperti Dapatkan Pod X) atau koleksi (seperti "Dapatkan semua Pod dari Namespace X"). `kube-apiserver`Mempertahankan cache objek, sehingga sumber daya yang diminta dapat dikembalikan dari cache atau mereka mungkin perlu diambil dari etcd terlebih dahulu. Latensi ini juga diukur dengan persentil ke-99 selama 5 menit, namun permintaan hanya-baca dapat memiliki cakupan terpisah. SLO mendefinisikan dua tujuan yang berbeda:

- Untuk permintaan yang dibuat untuk satu sumber daya (yaitu`kubectl get pod -n mynamespace my-controller-xxx`), latensi permintaan harus tetap <= 1 detik.
- Untuk permintaan yang dibuat untuk beberapa sumber daya dalam namespace atau cluster (misalnya,`kubectl get pods -A`) latensi harus tetap <= 30 detik

SLO memiliki nilai target yang berbeda untuk cakupan permintaan yang berbeda karena permintaan yang dibuat untuk daftar sumber daya Kubernetes mengharapkan detail semua objek dalam permintaan dikembalikan dalam SLO. Pada kelompok besar, atau koleksi sumber daya yang besar, ini dapat menghasilkan ukuran respons yang besar yang dapat memakan waktu untuk kembali. Misalnya, dalam klaster yang menjalankan puluhan ribu Pod dengan masing-masing Pod kira-kira 1 KiB saat dikodekan di JSON, mengembalikan semua Pod di cluster akan terdiri dari 10MB atau lebih.

Klien Kubernetes dapat membantu mengurangi ukuran respons ini menggunakan APIList Chunking untuk mengambil koleksi sumber daya yang besar.

Latensi Startup Pod

SLO ini terutama berkaitan dengan waktu yang dibutuhkan dari pembuatan Pod hingga saat kontainer di Pod itu benar-benar memulai eksekusi. Untuk mengukur ini perbedaan dari stempel waktu pembuatan yang direkam pada Pod, dan ketika WATCH pada Pod tersebut melaporkan kontainer telah dimulai dihitung (tidak termasuk waktu untuk penarikan image kontainer dan eksekusi kontainer init). Untuk memenuhi SLO, persentil ke-99 per hari klaster Latensi Startup Pod ini harus tetap <=5 detik.

Perhatikan bahwa SLO ini mengasumsikan bahwa node pekerja sudah ada di klaster ini dalam keadaan siap untuk Pod yang akan dijadwalkan. SLO ini tidak memperhitungkan penarikan gambar atau eksekusi kontainer init, dan juga membatasi pengujian ke “pod stateless” yang tidak memanfaatkan plugin penyimpanan persisten.

Metrik Kubernetes SLI

Kubernetes juga meningkatkan Observabilitas di sekitar dengan SLIs menambahkan metrik Prometheus ke komponen Kubernetes yang melacak ini dari waktu ke waktu. SLIs Menggunakan Prometheus Query Language (PromQL) kita dapat membangun kueri yang menampilkan kinerja SLI dari waktu ke waktu di alat seperti dasbor Prometheus atau Grafana, di bawah ini adalah beberapa contoh di atas. SLOs

Latensi Permintaan Server API

Metrik	Definisi
apiserver_request_sli_duration_seconds	Distribusi latensi respons (tidak termasuk durasi webhook dan waktu tunggu antrian prioritas & keadilan) dalam hitungan detik untuk setiap kata kerja, grup, versi, sumber daya, subsumber daya, ruang lingkup, dan komponen.
apiserver_request_duration_seconds	Distribusi latensi respons dalam hitungan detik untuk setiap kata kerja, nilai dry run, grup,

Metrik	Definisi
	versi, sumber daya, subsumber daya, ruang lingkup, dan komponen.

> **Note**
>
> `apiserver_request_sli_duration_seconds`Metrik tersedia mulai dari Kubernetes 1.27.

Anda dapat menggunakan metrik ini untuk menyelidiki waktu respons API Server dan jika ada hambatan dalam komponen Kubernetes atau plugin/komponen lainnya. Kueri di bawah ini didasarkan pada dasbor SLO komunitas.

API Request latency SLI (mutating) - kali ini tidak termasuk eksekusi webhook atau waktu menunggu dalam antrian. `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`

Latensi Permintaan API Total (bermutasi) - ini adalah total waktu yang dibutuhkan permintaan di server API, kali ini mungkin lebih lama dari waktu SLI karena mencakup eksekusi webhook dan waktu tunggu Prioritas dan Keadilan 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`

Dalam kueri ini kami mengecualikan permintaan API streaming yang tidak segera kembali, seperti `kubectl port-forward` atau `kubectl exec` request (`subresource!~"proxy|attach|log|exec|portforward"`), dan kami memfilter hanya untuk kata kerja Kubernetes yang memodifikasi objek (). `verb=~"CREATE|DELETE|PATCH|POST|PUT"` Kami kemudian menghitung persentil ke-99 dari latensi itu selama 5 menit terakhir.

Kita dapat menggunakan kueri serupa untuk permintaan API hanya baca, kita cukup memodifikasi kata kerja yang kita filter untuk menyertakan tindakan `LIST` Read only dan. `GET` Ada juga ambang batas SLO yang berbeda tergantung pada ruang lingkup permintaan, yaitu mendapatkan satu sumber daya atau mencantumkan sejumlah sumber daya.

API Request latency SLI (read-only) - kali ini tidak termasuk eksekusi webhook atau waktu menunggu dalam antrian. Untuk sumber daya tunggal (cakupan = sumber daya, ambang batas = 1s) `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"GET", scope=~"resource"}[5m])) by (resource, subresource, verb, scope, le))`

Untuk kumpulan sumber daya di namespace yang sama (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))`

Untuk kumpulan sumber daya di seluruh cluster (scope=cluster, threshold = 30s) `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"LIST", scope=~"cluster"}[5m])) by (resource, subresource, verb, scope, le))`

Latensi Permintaan API Total (hanya-baca) - ini adalah total waktu yang dibutuhkan permintaan di server API, kali ini mungkin lebih lama dari waktu SLI karena termasuk eksekusi webhook dan waktu tunggu. Untuk sumber daya tunggal (cakupan = sumber daya, ambang batas = 1s) `histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_bucket{verb=~"GET", scope=~"resource"}[5m])) by (resource, subresource, verb, scope, le))`

Untuk kumpulan sumber daya di namespace yang sama (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))`

Untuk kumpulan sumber daya di seluruh cluster (scope=cluster, threshold = 30s) `histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_bucket{verb=~"LIST", scope=~"cluster"}[5m])) by (resource, subresource, verb, scope, le))`

Metrik SLI memberikan wawasan tentang kinerja komponen Kubernetes dengan mengecualikan waktu yang dihabiskan permintaan untuk menunggu dalam antrean Prioritas dan Keadilan API, bekerja melalui webhook masuk, atau ekstensi Kubernetes lainnya. Metrik total memberikan tampilan yang lebih holistik karena mencerminkan waktu aplikasi Anda menunggu respons dari server API. Membandingkan metrik ini dapat memberikan wawasan tentang di mana penundaan dalam pemrosesan permintaan diperkenalkan.

Latensi Startup Pod

Metrik	Definisi
kubelet_pod_start_sli_duration_seconds	Durasi dalam hitungan detik untuk memulai pod, tidak termasuk waktu untuk menarik gambar dan menjalankan kontainer init, diukur dari stempel waktu pembuatan pod hingga saat semua kontainernya dilaporkan seperti yang dimulai dan diamati melalui jam tangan
kubelet_pod_start_duration_seconds	Durasi dalam hitungan detik dari kubelet melihat pod untuk pertama kalinya hingga pod mulai berjalan. Ini tidak termasuk waktu untuk menjadwalkan pod atau skala kapasitas node pekerja.

> **Note**
>
> `kubelet_pod_start_sli_duration_seconds`tersedia mulai Kubernetes 1.27.

Mirip dengan kueri di atas, Anda dapat menggunakan metrik ini untuk mendapatkan wawasan tentang berapa lama penskalaan node, penarikan gambar, dan container init menunda peluncuran pod dibandingkan dengan tindakan Kubelet.

Latensi startup Pod SLI - ini adalah waktu dari pod yang dibuat hingga saat container aplikasi dilaporkan sedang berjalan. Ini termasuk waktu yang diperlukan untuk kapasitas node pekerja untuk tersedia dan pod dijadwalkan, tetapi ini tidak termasuk waktu yang diperlukan untuk menarik gambar atau untuk menjalankan container init. `histogram_quantile(0.99, sum(rate(kubelet_pod_start_sli_duration_seconds_bucket[5m])) by (le))`

Latensi startup Pod Total - ini adalah waktu yang dibutuhkan kubelet untuk memulai pod untuk pertama kalinya. Ini diukur dari saat kubelet menerima pod melalui WATCH, yang tidak menyertakan waktu untuk penskalaan atau penjadwalan node pekerja. Ini termasuk waktu untuk menarik gambar dan wadah init untuk dijalankan. `histogram_quantile(0.99, sum(rate(kubelet_pod_start_duration_seconds_bucket[5m])) by (le))`

SLOs di Cluster Anda

Jika Anda mengumpulkan metrik Prometheus dari sumber daya Kubernetes di kluster EKS Anda, Anda dapat memperoleh wawasan yang lebih dalam tentang kinerja komponen bidang kontrol Kubernetes.

Repo perf-tests mencakup dasbor Grafana yang menampilkan latensi dan metrik kinerja penting untuk cluster selama pengujian. Konfigurasi perf-tests memanfaatkan proyek open source yang dikonfigurasi untuk mengumpulkan metrik Kubernetes kube-prometheus-stack, tetapi Anda juga dapat menggunakan Amazon Managed Prometheus dan Amazon Managed Grafana.

Jika Anda menggunakan `kube-prometheus-stack` atau solusi Prometheus serupa, Anda dapat menginstal dasbor yang sama untuk mengamati di cluster Anda secara real SLOs time.

1. Pertama-tama Anda harus menginstal Aturan Prometheus yang digunakan di dasbor dengan. `kubectl apply -f prometheus-rules.yaml` Anda dapat mengunduh salinan aturan di sini: https://github.com/kubernetes/ perf- tests/blob/master/clusterloader2/pkg/prometheus/manifests/ prometheus -rules.yaml
   a. Pastikan untuk memeriksa namespace dalam file yang cocok dengan lingkungan Anda
   b. Verifikasi bahwa label cocok dengan nilai `prometheus.prometheusSpec.ruleSelector` helm jika Anda menggunakan `kube-prometheus-stack`
2. Anda kemudian dapat menginstal dasbor di Grafana. Dasbor json dan skrip python untuk menghasilkannya tersedia di sini: perf- https://github.com/kubernetes/ tests/tree/master/ clusterloader2/pkg/prometheus/manifests/dashboards
   a. `slo.jsondasbor` menampilkan kinerja cluster dalam kaitannya dengan Kubernetes SLOs

Pertimbangkan bahwa SLOs mereka berfokus pada kinerja komponen Kubernetes di klaster Anda, tetapi ada metrik tambahan yang dapat Anda tinjau yang memberikan perspektif atau wawasan berbeda ke klaster Anda. Proyek komunitas Kubernetes seperti K ube-state-metrics dapat membantu Anda menganalisis tren di klaster dengan cepat. Sebagian besar plugin dan driver umum dari komunitas Kubernetes juga memancarkan metrik Prometheus, memungkinkan Anda untuk menyelidiki hal-hal seperti autoscaler atau penjadwal kustom.

Panduan Praktik Terbaik Observabilitas memiliki contoh metrik Kubernetes lainnya yang dapat Anda gunakan untuk mendapatkan wawasan lebih lanjut.

Batas yang Diketahui dan Service Quotas

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Amazon EKS dapat digunakan untuk berbagai beban kerja dan dapat berinteraksi dengan berbagai layanan AWS, dan kami telah melihat beban kerja pelanggan menghadapi rentang kuota layanan AWS yang serupa dan masalah lain yang menghambat skalabilitas.

Akun AWS Anda memiliki kuota default (batas atas jumlah setiap sumber daya AWS yang dapat diminta tim Anda). Setiap layanan AWS menentukan kuota mereka sendiri, dan kuota umumnya khusus wilayah. Anda dapat meminta kenaikan untuk beberapa kuota (batas lunak), dan kuota lainnya tidak dapat ditingkatkan (batas keras). Anda harus mempertimbangkan nilai-nilai ini saat merancang aplikasi Anda. Pertimbangkan untuk meninjau batas layanan ini secara berkala dan menggabungkannya selama dalam desain aplikasi Anda.

Anda dapat meninjau penggunaan di akun Anda dan membuka permintaan peningkatan kuota di konsol AWS Service Quotas, atau menggunakan AWS CLI. Lihat dokumentasi AWS dari Layanan AWS masing-masing untuk detail selengkapnya tentang Service Quotas dan pembatasan atau pemberitahuan lebih lanjut tentang peningkatannya.

> **Note**
>
> Amazon EKS Service Quotas mencantumkan kuota layanan dan memiliki tautan untuk meminta peningkatan jika tersedia.

Quotas Layanan AWS Lainnya

Kami telah melihat pelanggan EKS terpengaruh oleh kuota yang tercantum di bawah ini untuk layanan AWS lainnya. Beberapa di antaranya mungkin hanya berlaku untuk kasus penggunaan atau konfigurasi tertentu, namun Anda dapat mempertimbangkan apakah solusi Anda akan menemukan salah satu dari ini saat skalanya. Kuota diatur oleh Layanan dan setiap Kuota memiliki ID dalam format L-XXXXXXXX yang dapat Anda gunakan untuk mencarinya di konsol AWS Service Quotas

Layanan	Kuota (L-XXxxx)	Dampak	ID (L-XXxxx)	default
IAM	Tabel Per Akun	Dapat membatasi jumlah cluster atau peran IRSA dalam akun.	L- FE177 D64	1.000
IAM	OpenId menghubungkan penyedia per akun	Dapat membatasi jumlah Cluster per akun, OpenID Connect digunakan oleh IRSA	L-858F3967	100
IAM	Panjang kebijakan kepercayaan peran	Dapat membatasi jumlah cluster yang dikaitkan dengan peran IAM untuk IRSA	L-C07B4B0D	2,048
VPC	Grup keamanan per antarmuka jaringan	Dapat membatasi kontrol atau konektivitas jaringan untuk cluster Anda	L-2 AFB9258	5
VPC	IPv4 Blok CIDR untuk VPC	Dapat membatasi jumlah Node Pekerja EKS	L-83 CA0 A9D	5
VPC	Rute per tabel perutean	Dapat membatasi kontrol atau	L-93826ACB	50

Layanan	Kuota (L-XXxxx)	Dampak	ID (L-XXxxx)	default
		konektivitas jaringan untuk cluster Anda		
VPC	Koneksi peering VPC aktif per VPC	Dapat membatasi kontrol atau konektivitas jaringan untuk cluster Anda	L-7E9ECCDB	50
VPC	Aturan masuk atau keluar per grup keamanan.	Dapat membatasi kontrol atau konektivitas jaringan untuk cluster Anda, beberapa pengontrol di EKS membuat aturan baru	L-0 F EA8095	50
VPC	VPCs per Wilayah	Dapat membatasi jumlah Cluster per akun atau kontrol atau konektivitas jaringan untuk cluster Anda	L-F678F1CE	5

Layanan	Kuota (L-XXxxx)	Dampak	ID (L-XXxxx)	default
VPC	Gateway internet per Wilayah	Dapat membatasi jumlah Cluster per akun atau kontrol atau konektivitas jaringan untuk cluster Anda	L-A4707A72	5
VPC	Antarmuka jaringan per Wilayah	Dapat membatasi jumlah node EKS Worker, atau scaling/update aktivitas bidang kontrol EKS Dampak.	L- DF5 E4 CA3	5.000
VPC	Penggunaan Alamat Jaringan	Dapat membatasi jumlah Cluster per akun atau kontrol atau konektivitas jaringan untuk cluster Anda	L- BB24 F6E5	64.000
VPC	Penggunaan Alamat Jaringan Peered	Dapat membatasi jumlah Cluster per akun atau kontrol atau konektivitas jaringan untuk cluster Anda	L- CD17 FD4 B	128.000

Layanan	Kuota (L-XXxxx)	Dampak	ID (L-XXxxx)	default
ELB	Listener per Penyeimbang Beban Jaringan	Dapat membatasi kontrol masuknya lalu lintas ke cluster.	L-57A373D6	50
ELB	Grup Target per Wilayah	Dapat membatasi kontrol masuknya lalu lintas ke cluster.	L-B22855CB	3.000
ELB	Target per Application Load Balancer	Dapat membatasi kontrol masuknya lalu lintas ke cluster.	L-7E6692B2	1.000
ELB	Target per Penyeimbang Beban Jaringan	Dapat membatasi kontrol masuknya lalu lintas ke cluster.	L- EEF1 AD04	3.000
ELB	Subnet per Availability Zone per Penyeimbang Beban Jaringan	Dapat membatasi kontrol masuknya lalu lintas ke cluster.	L-B211E961	500
ELB	Target per Grup Target per Wilayah	Dapat membatasi kontrol masuknya lalu lintas ke cluster.	L-A0D0B863	1.000

Layanan	Kuota (L-XXxxx)	Dampak	ID (L-XXxxx)	default
ELB	Application Load Balancer per Wilayah	Dapat membatasi kontrol masuknya lalu lintas ke cluster.	L-53 B97 DA6	50
ELB	Classic Load Balancer per Wilayah	Dapat membatasi kontrol masuknya lalu lintas ke cluster.	L-E9E9831D	20
ELB	Penyeimbang Beban Jaringan per Wilayah	Dapat membatasi kontrol masuknya lalu lintas ke cluster.	L-69A177A2	50
EC2	Menjalankan instance Standar Sesuai Permintaan (A, C, D, H, I, M, R, T, Z) (sebagai jumlah vCPU maksimum)	Dapat membatasi jumlah Node Pekerja EKS	L-1216C47A	5
EC2	Semua Permintaan Instans Spot Standar (A, C, D, H, I, M, R, T, Z) (sebagai jumlah vCPU maksimum)	Dapat membatasi jumlah Node Pekerja EKS	L-34B43A08	5

Layanan	Kuota (L-XXxxx)	Dampak	ID (L-XXxxx)	default
EC2	EC2-VPC Elastis IPs	Dapat membatasi jumlah NAT GWs (dan dengan demikian VPCs), yang dapat membatasi jumlah cluster di suatu wilayah	L-0263D0A3	5
EBS	Snapshot per Wilayah	Dapat membatasi strategi pencadangan untuk beban kerja stateful	L-309 BACF6	100.000
EBS	Penyimpanan untuk volume SSD Tujuan Umum (gp3) dalam TiB	Dapat membatasi jumlah Node Pekerja EKS, atau Persisten tVolume penyimpanan	L-7A658B76	50
EBS	Penyimpanan untuk volume SSD Tujuan Umum (gp2) dalam TiB	Dapat membatasi jumlah Node Pekerja EKS, atau Persisten tVolume penyimpanan	L-D18 D FCD1	50

Layanan	Kuota (L-XXxxx)	Dampak	ID (L-XXxxx)	default
ECR	Repositori terdaftar	Dapat membatasi jumlah beban kerja di cluster Anda	L- CFEB8 E8D	100.000
ECR	Gambar per repositori	Dapat membatasi jumlah beban kerja di cluster Anda	L-03A36 CE1	20.000
SecretsManager	Rahasia per Wilayah	Dapat membatasi jumlah beban kerja di cluster Anda	L-2F66C23C	500.000

Pelambatan Permintaan AWS

Layanan AWS juga menerapkan pembatasan permintaan untuk memastikan bahwa layanan tersebut tetap berkinerja dan tersedia untuk semua pelanggan. Mirip dengan Service Quotas, setiap layanan AWS mempertahankan ambang batas pembatasan permintaannya sendiri. Pertimbangkan untuk meninjau dokumentasi AWS Service masing-masing jika beban kerja Anda perlu mengeluarkan sejumlah besar panggilan API dengan cepat atau jika Anda melihat kesalahan pembatasan permintaan dalam aplikasi Anda.

Permintaan API EC2 seputar penyediaan antarmuka jaringan EC2 atau alamat IP dapat mengalami pembatasan permintaan dalam cluster besar atau ketika klaster menskalakan secara drastis. Tabel di bawah ini menunjukkan beberapa tindakan API yang telah kami lihat pelanggan mengalami pembatasan permintaan. Anda dapat meninjau default batas tarif EC2 dan langkah-langkah untuk meminta kenaikan batas tarif dalam dokumentasi EC2 pada Rate Throttling.

Tindakan Mutasi	Tindakan hanya-baca
AssignPrivateIpAddresses	DescribeDhcpOptions
AttachNetworkInterface	DescribeInstances
CreateNetworkInterface	DescribeNetworkInterfaces
DeleteNetworkInterface	DescribeSecurityGroups
DeleteTags	DescribeTags
DetachNetworkInterface	DescribeVpcs
ModifyNetworkInterfaceAttribute	DescribeVolumes
UnassignPrivateIpAddresses	

Batas Diketahui Lainnya

- Route 53 juga memiliki batas tarif yang cukup rendah yaitu 5 permintaan per detik ke Route 53 API. Jika Anda memiliki sejumlah besar domain untuk diperbarui dengan proyek seperti DNS Eksternal, Anda mungkin melihat pembatasan tingkat dan penundaan dalam memperbarui domain.
  - Beberapa jenis instans Nitro memiliki batas lampiran volume 28 yang dibagi antara volume Amazon EBS, antarmuka jaringan, dan volume penyimpanan NVMe instans. Jika beban kerja Anda meningkatkan banyak volume EBS, Anda mungkin menemukan batasan kepadatan pod yang dapat Anda capai dengan jenis instance ini
  - Ada jumlah maksimum koneksi yang dapat dilacak per instance Ec2. Jika beban kerja Anda menangani sejumlah besar koneksi, Anda mungkin melihat kegagalan atau kesalahan komunikasi karena maksimum ini telah terpukul. Anda dapat menggunakan metrik kinerja `conntrack_allowance_available` dan `conntrack_allowance_exceeded` jaringan untuk memantau jumlah koneksi yang dilacak pada node pekerja EKS Anda.
  - Di lingkungan EKS, batas penyimpanan etcd adalah 8 GiB sesuai panduan hulu. Harap pantau metrik `apiserver_storage_size_bytes` untuk melacak ukuran db etcd. Anda dapat merujuk ke aturan peringatan `etcdBackendQuotaLowSpace` dan `etcdExcessiveDatabaseGrowth` mengatur pemantauan ini.

Praktik Terbaik untuk Upgrade Cluster

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Panduan ini menunjukkan kepada administrator klaster cara merencanakan dan menjalankan strategi peningkatan Amazon EKS mereka. Ini juga menjelaskan cara memutakhirkan node yang dikelola sendiri, grup node terkelola, node Karpenter, dan node Fargate. Ini tidak termasuk panduan tentang EKS Anywhere, Kubernetes yang dikelola sendiri, AWS Outposts, atau AWS Local Zones.

Ikhtisar

Versi Kubernetes mencakup bidang kontrol dan bidang data. Untuk memastikan kelancaran operasi, baik bidang kontrol dan bidang data harus menjalankan versi minor Kubernetes yang sama, seperti 1.24. Saat AWS mengelola dan memutakhirkan bidang kontrol, memperbarui node pekerja di bidang data adalah tanggung jawab Anda.

- Control plane — Versi control plane ditentukan oleh server API Kubernetes. Di kluster Amazon EKS, AWS menangani pengelolaan komponen ini. Upgrade control plane dapat dimulai melalui AWS API.
- Data plane — Versi data plane dikaitkan dengan versi Kubelet yang berjalan pada node individual Anda. Dimungkinkan untuk memiliki node di cluster yang sama yang menjalankan versi yang berbeda. Anda dapat memeriksa versi semua node dengan menjalankan`kubectl get nodes`.

Sebelum Upgrade

Jika Anda berencana untuk meningkatkan versi Kubernetes Anda di Amazon EKS, ada beberapa kebijakan, alat, dan prosedur penting yang harus Anda lakukan sebelum memulai peningkatan.

- Memahami Kebijakan Penghentian — Dapatkan pemahaman mendalam tentang cara kerja kebijakan penghentian Kubernetes. Waspadai perubahan yang akan datang yang dapat memengaruhi aplikasi Anda yang ada. Versi Kubernetes yang lebih baru sering menghapus fitur tertentu APIs , berpotensi menyebabkan masalah untuk menjalankan aplikasi.

- Tinjau Log Perubahan Kubernetes — Tinjau log perubahan Kubernetes secara menyeluruh bersama versi Amazon EKS Kubernetes untuk memahami dampak apa pun yang mungkin terjadi pada klaster Anda, seperti melanggar perubahan yang dapat memengaruhi beban kerja Anda.
- Menilai Kompatibilitas Add-On Cluster — Amazon EKS tidak secara otomatis memperbarui add-on ketika versi baru dirilis atau setelah Anda memperbarui klaster Anda ke versi minor Kubernetes yang baru. Tinjau Memperbarui add-on untuk memahami kompatibilitas add-on cluster yang ada dengan versi cluster yang ingin Anda tingkatkan.
- Aktifkan Pencatatan Pesawat Kontrol — Aktifkan pencatatan bidang kontrol untuk menangkap log, kesalahan, atau masalah yang dapat muncul selama proses peningkatan. Pertimbangkan untuk meninjau log ini untuk anomali apa pun. Uji peningkatan klaster di lingkungan non-produksi, atau integrasikan pengujian otomatis ke dalam alur kerja integrasi berkelanjutan Anda untuk menilai kompatibilitas versi dengan aplikasi, pengontrol, dan integrasi kustom Anda.
- Jelajahi eksctl untuk Manajemen Cluster — Pertimbangkan untuk menggunakan eksctl untuk mengelola kluster EKS Anda. Ini memberi Anda kemampuan untuk memperbarui bidang kontrol, mengelola add-on, dan menangani pembaruan out-of-the-box node pekerja.
- Pilih Mode Otomatis atau Grup Node Terkelola — Merampingkan dan mengotomatiskan peningkatan node pekerja dengan menggunakan Mode Otomatis atau grup node terkelola EKS. Opsi ini menyederhanakan proses dan mengurangi intervensi manual.
- Manfaatkan kubectl Convert Plugin — Manfaatkan plugin kubectl convert untuk memfasilitasi konversi file manifes Kubernetes di antara versi API yang berbeda. Ini dapat membantu memastikan bahwa konfigurasi Anda tetap kompatibel dengan versi Kubernetes yang baru.

Pertahankan cluster Anda up-to-date

Tetap mengikuti pembaruan Kubernetes sangat penting untuk lingkungan EKS yang aman dan efisien, yang mencerminkan model tanggung jawab bersama di Amazon EKS. Dengan mengintegrasikan strategi ini ke dalam alur kerja operasional Anda, Anda memposisikan diri Anda untuk mempertahankan up-to-date, mengamankan klaster yang memanfaatkan sepenuhnya fitur dan peningkatan terbaru. Taktik:

- Kebijakan Versi yang Didukung - Sejajar dengan komunitas Kubernetes, Amazon EKS biasanya menyediakan tiga versi Kubernetes yang aktif. Versi minor Kubernetes berada di bawah dukungan standar di Amazon EKS selama 14 bulan pertama setelah dirilis. Setelah versi melewati akhir tanggal dukungan standar, ia memasuki dukungan diperpanjang untuk 12 bulan ke depan.

Pemberitahuan penghentian dikeluarkan setidaknya 60 hari sebelum versi mencapai akhir tanggal dukungan standar. Untuk detail selengkapnya, lihat dokumen Siklus Hidup Versi EKS.

- Kebijakan Upgrade Otomatis - Kami sangat menyarankan untuk tetap sinkron dengan pembaruan Kubernetes di kluster EKS Anda. Cluster yang berjalan pada versi Kubernetes yang telah menyelesaikan siklus hidup 26 bulan (14 bulan dukungan standar ditambah 12 bulan dukungan diperpanjang) akan otomatis ditingkatkan ke versi berikutnya. Perhatikan bahwa Anda dapat menonaktifkan dukungan yang diperluas. Kegagalan untuk memutakhirkan secara proaktif sebelum versi end-of-life memicu peningkatan otomatis, yang dapat mengganggu beban kerja dan sistem Anda. Untuk informasi tambahan, lihat Versi EKS FAQs.
- Buat Runbook Upgrade - Buat proses yang terdokumentasi dengan baik untuk mengelola peningkatan. Sebagai bagian dari pendekatan proaktif Anda, kembangkan runbook dan alat khusus yang disesuaikan dengan proses peningkatan Anda. Ini tidak hanya meningkatkan kesiapan Anda tetapi juga menyederhanakan transisi yang kompleks. Jadikan praktik standar untuk meningkatkan cluster Anda setidaknya setahun sekali. Praktik ini menyelaraskan Anda dengan kemajuan teknologi yang sedang berlangsung, sehingga meningkatkan efisiensi dan keamanan lingkungan Anda.

Tinjau kalender rilis EKS

Tinjau kalender rilis EKS Kubernetes untuk mempelajari kapan versi baru akan datang, dan kapan dukungan untuk versi tertentu berakhir. Umumnya, EKS merilis tiga versi minor Kubernetes setiap tahunnya, dan setiap versi minor didukung selama sekitar 14 bulan.

Selain itu, tinjau informasi rilis Kubernetes hulu.

Memahami bagaimana model tanggung jawab bersama berlaku untuk peningkatan klaster

Anda bertanggung jawab untuk memulai upgrade untuk kedua bidang kontrol cluster serta bidang data. Pelajari cara memulai pemutakhiran. Saat Anda memulai pemutakhiran klaster, AWS mengelola pemutakhiran bidang kontrol cluster. Anda bertanggung jawab untuk meningkatkan pesawat data, termasuk pod dan addon Fargate. Anda harus memvalidasi dan merencanakan peningkatan untuk beban kerja yang berjalan di klaster Anda untuk memastikan ketersediaan dan operasinya tidak terpengaruh setelah peningkatan klaster

Tingkatkan cluster di tempat

EKS mendukung strategi peningkatan cluster di tempat. Ini mempertahankan sumber daya klaster, dan menjaga konfigurasi klaster tetap konsisten (misalnya, titik akhir API, OIDC, ENIs, penyeimbang beban). Ini tidak terlalu mengganggu bagi pengguna klaster, dan akan menggunakan beban kerja dan sumber daya yang ada di klaster tanpa mengharuskan Anda menerapkan ulang beban kerja atau memigrasikan sumber daya eksternal (misalnya, DNS, penyimpanan).

Saat melakukan peningkatan cluster di tempat, penting untuk dicatat bahwa hanya satu peningkatan versi minor yang dapat dijalankan pada satu waktu (misalnya, dari 1,24 hingga 1,25).

Ini berarti bahwa jika Anda perlu memperbarui beberapa versi, serangkaian peningkatan berurutan akan diperlukan. Merencanakan peningkatan berurutan lebih rumit, dan memiliki risiko downtime yang lebih tinggi. Dalam situasi ini, lihatthe section called “Evaluasi Blue/Green Cluster sebagai alternatif untuk peningkatan klaster di tempat”.

Tingkatkan bidang kontrol dan bidang data Anda secara berurutan

Untuk memutakhirkan cluster, Anda perlu melakukan tindakan berikut:

1. Tinjau catatan rilis Kubernetes dan EKS.
2. Ambil cadangan dari cluster. (opsional)
3. Mengidentifikasi dan memulihkan penggunaan API yang tidak digunakan lagi dan dihapus di beban kerja Anda.
4. Pastikan Managed Node Groups, jika digunakan, berada pada versi Kubernetes yang sama dengan control plane. Grup node dan node terkelola EKS yang dibuat oleh EKS Fargate Profiles mendukung 2 versi minor yang miring antara bidang kontrol dan bidang data untuk Kubernetes versi 1.27 ke bawah. Mulai 1,28 ke atas, grup node dan node yang dikelola EKS yang dibuat oleh EKS Fargate Profiles mendukung 3 versi minor miring antara bidang kontrol dan bidang data. Misalnya, jika versi bidang kontrol EKS Anda adalah 1,28, Anda dapat menggunakan versi kubelet dengan aman setua 1,25. Jika versi EKS Anda adalah 1.27, versi kubelet tertua yang dapat Anda gunakan adalah 1.25.
5. Tingkatkan bidang kontrol cluster menggunakan konsol AWS atau cli.
6. Tinjau kompatibilitas add-on. Tingkatkan add-on Kubernetes dan pengontrol kustom Anda, sesuai kebutuhan.
7. Perbarui kubectl.

8. Tingkatkan bidang data cluster. Tingkatkan node Anda ke versi minor Kubernetes yang sama dengan klaster yang telah Anda upgrade.

> **Tip**
>
> Jika cluster Anda dibuat menggunakan Mode Otomatis EKS, Anda tidak perlu memutakhirkan bidang data cluster Anda. Setelah memutakhirkan bidang kontrol Anda, Mode Otomatis EKS akan mulai memperbarui node terkelola secara bertahap sambil menghormati semua anggaran gangguan pod. Pastikan untuk memantau pembaruan ini untuk memverifikasi kepatuhan terhadap persyaratan operasional Anda.

Gunakan Dokumentasi EKS untuk membuat daftar periksa pemutakhiran

Dokumentasi versi EKS Kubernetes mencakup daftar perubahan terperinci untuk setiap versi. Buat daftar periksa untuk setiap peningkatan.

Untuk panduan peningkatan versi EKS tertentu, tinjau dokumentasi untuk perubahan dan pertimbangan penting untuk setiap versi.

- Tinjau catatan rilis untuk versi Kubernetes pada dukungan standar
- Tinjau catatan rilis untuk versi Kubernetes pada dukungan yang diperluas

Upgrade add-on dan komponen menggunakan Kubernetes API

Sebelum Anda meng-upgrade klaster, Anda harus memahami versi komponen Kubernetes yang Anda gunakan. Inventarisasi komponen klaster, dan identifikasi komponen yang menggunakan API Kubernetes secara langsung. Ini termasuk komponen kluster penting seperti agen pemantauan dan logging, autoscaler cluster, driver penyimpanan kontainer (misalnya EBS CSI, EFS CSI), pengontrol ingress, dan beban kerja atau add-on lainnya yang bergantung pada Kubernetes API secara langsung.

> **Tip**
>
> Komponen cluster kritis sering dipasang di `*-system` namespace

```
kubectl get ns | grep '-system'
```

Setelah Anda mengidentifikasi komponen yang mengandalkan API Kubernetes, periksa dokumentasi mereka untuk kompatibilitas versi dan persyaratan peningkatan. Misalnya, lihat dokumentasi AWS Load Balancer Controller untuk kompatibilitas versi. Beberapa komponen mungkin perlu ditingkatkan atau konfigurasi diubah sebelum melanjutkan dengan upgrade cluster. Beberapa komponen penting yang harus diperiksa termasuk CoreDNS, kube-proxy, VPC CNI, dan driver penyimpanan.

Cluster sering berisi banyak beban kerja yang menggunakan API Kubernetes dan diperlukan untuk fungsionalitas beban kerja seperti pengontrol ingress, sistem pengiriman berkelanjutan, dan alat pemantauan. Saat Anda memutakhirkan kluster EKS, Anda juga harus meningkatkan add-on dan alat pihak ketiga untuk memastikannya kompatibel.

Lihat contoh add-on umum berikut dan dokumentasi pemutakhiran yang relevan:

- Amazon VPC CNI: Untuk versi yang direkomendasikan dari add-on Amazon VPC CNI untuk setiap versi cluster, lihat Memperbarui plugin Amazon VPC CNI untuk add-on yang dikelola sendiri Kubernetes. Ketika diinstal sebagai Amazon EKS Add-on, itu hanya dapat ditingkatkan satu versi minor pada satu waktu.
- kube-proxy: Lihat Memperbarui add-on yang dikelola sendiri kube-proxy Kubernetes.
- CoreDNS: Lihat Memperbarui add-on yang dikelola sendiri CoreDNS.
- AWS Load Balancer Controller: AWS Load Balancer Controller harus kompatibel dengan versi EKS yang telah Anda gunakan. Lihat panduan instalasi untuk informasi lebih lanjut.
- Driver Amazon Elastic Block Store (Amazon EBS) Container Storage Interface (CSI): Untuk informasi instalasi dan peningkatan, lihat Mengelola driver Amazon EBS CSI sebagai add-on Amazon EKS.
- Driver Amazon Elastic File System (Amazon EFS) Container Storage Interface (CSI): Untuk informasi penginstalan dan peningkatan, lihat driver Amazon EFS CSI.
- Kubernetes Metrics Server: Untuk informasi selengkapnya, lihat metrics-server di. GitHub
- Kubernetes Cluster Autoscaler: Untuk memutakhirkan versi Kubernetes Cluster Autoscaler, ubah versi gambar dalam penerapan. Cluster Autoscaler digabungkan erat dengan penjadwal Kubernetes. Anda akan selalu perlu untuk meng-upgrade ketika Anda meng-upgrade cluster. Tinjau GitHub rilis untuk menemukan alamat rilis terbaru yang sesuai dengan versi minor Kubernetes Anda.
- Karpenter: Untuk informasi instalasi dan peningkatan, lihat dokumentasi Karpenter.

> **Tip**
>
> Anda tidak perlu memutakhirkan kemampuan Amazon EKS Auto Mode secara manual, termasuk kemampuan penskalaan otomatis komputasi, penyimpanan blok, dan penyeimbangan beban.

Verifikasi persyaratan EKS dasar sebelum meningkatkan

AWS memerlukan sumber daya tertentu di akun Anda untuk menyelesaikan proses peningkatan. Jika sumber daya ini tidak ada, klaster tidak dapat ditingkatkan. Upgrade pesawat kontrol membutuhkan sumber daya berikut:

1. Alamat IP yang tersedia: Amazon EKS memerlukan hingga lima alamat IP yang tersedia dari subnet yang Anda tentukan saat membuat cluster untuk memperbarui cluster. Jika tidak, perbarui konfigurasi klaster Anda untuk menyertakan subnet klaster baru sebelum melakukan pembaruan versi.
2. Peran EKS IAM: Peran IAM bidang kontrol masih ada di akun dengan izin yang diperlukan.
3. Jika kluster Anda mengaktifkan enkripsi rahasia, pastikan peran IAM cluster memiliki izin untuk menggunakan kunci AWS Key Management Service (AWS KMS).

Verifikasi alamat IP yang tersedia

Untuk memperbarui klaster, Amazon EKS memerlukan hingga lima alamat IP yang tersedia dari subnet yang Anda tentukan saat membuat klaster.

Untuk memverifikasi bahwa subnet Anda memiliki alamat IP yang cukup untuk meng-upgrade cluster, Anda dapat menjalankan perintah berikut:

```
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 |
+--------------------------+-------------+-------+
```

Pembantu Metrik CNI VPC dapat digunakan untuk membuat dasbor CloudWatch metrik VPC. Amazon EKS merekomendasikan untuk memperbarui subnet cluster menggunakan "UpdateClusterConfiguration" API sebelum memulai upgrade versi Kubernetes jika Anda kehabisan alamat IP di subnet yang awalnya ditentukan selama pembuatan klaster. Harap verifikasi bahwa subnet baru Anda akan diberikan:

- milik set yang sama AZs yang dipilih selama pembuatan cluster.
- milik VPC yang sama yang disediakan selama pembuatan cluster

Harap pertimbangkan untuk mengaitkan blok CIDR tambahan jika alamat IP di blok CIDR VPC yang ada habis. AWS memungkinkan asosiasi blok CIDR tambahan dengan VPC klaster Anda yang ada, yang secara efektif memperluas kumpulan alamat IP Anda. Ekspansi ini dapat dicapai dengan memperkenalkan rentang IP pribadi tambahan (RFC 1918) atau, jika perlu, rentang IP publik (non-RFC 1918). Anda harus menambahkan blok CIDR VPC baru dan mengizinkan penyegaran VPC selesai sebelum Amazon EKS dapat menggunakan CIDR baru. Setelah itu, Anda dapat memperbarui subnet berdasarkan blok CIDR yang baru disiapkan ke VPC.

Verifikasi peran EKS IAM

Untuk memverifikasi bahwa peran IAM tersedia dan memiliki kebijakan peran asumsi yang benar di akun Anda, Anda dapat menjalankan perintah berikut:

```
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"
        }
    ]
}
```

Migrasi ke Pengaya EKS

Amazon EKS secara otomatis menginstal add-on seperti plugin Amazon VPC CNI untuk Kubernetes, `kube-proxy` dan CoreDNS untuk setiap cluster. Add-on dapat dikelola sendiri, atau diinstal sebagai Amazon EKS Add-on. Amazon EKS Add-ons adalah cara alternatif untuk mengelola add-on menggunakan EKS API.

Anda dapat menggunakan Amazon EKS Add-on untuk memperbarui versi dengan satu perintah. Sebagai Contoh:

```
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
```

Periksa apakah Anda memiliki Pengaya EKS dengan:

```
aws eks list-addons --cluster-name <cluster name>
```

> **Warning**
>
> Pengaya EKS tidak ditingkatkan secara otomatis selama upgrade pesawat kontrol. Anda harus memulai pembaruan add-on EKS, dan memilih versi yang diinginkan.

- Anda bertanggung jawab untuk memilih versi yang kompatibel dari semua versi yang tersedia. Tinjau panduan tentang kompatibilitas versi add-on.
- Amazon EKS Add-on hanya dapat ditingkatkan satu versi minor pada satu waktu.

Pelajari lebih lanjut tentang komponen apa saja yang tersedia sebagai Pengaya EKS, dan cara memulainya.

Pelajari cara menyediakan konfigurasi kustom ke Add-on EKS.

Mengidentifikasi dan memulihkan penggunaan API yang dihapus sebelum memutakhirkan bidang kontrol

Anda harus mengidentifikasi penggunaan API yang dihapus APIs sebelum memutakhirkan bidang kontrol EKS Anda. Untuk melakukan itu, kami sarankan menggunakan alat yang dapat memeriksa cluster yang sedang berjalan atau file manifes Kubernetes statis yang dirender.

Menjalankan pemeriksaan terhadap file manifes statis umumnya lebih akurat. Jika dijalankan terhadap klaster langsung, alat ini dapat mengembalikan positif palsu.

Kubernetes API yang sudah usang tidak berarti API telah dihapus. Anda harus memeriksa Kebijakan Penghentian Kubernetes untuk memahami bagaimana penghapusan API memengaruhi beban kerja Anda.

Wawasan Cluster

Cluster Insights adalah fitur yang memberikan temuan tentang masalah yang dapat memengaruhi kemampuan untuk meningkatkan kluster EKS ke versi Kubernetes yang lebih baru. Temuan ini dikuratori dan dikelola oleh Amazon EKS dan menawarkan rekomendasi tentang cara memperbaikinya. Dengan memanfaatkan Cluster Insights, Anda dapat meminimalkan upaya yang dihabiskan untuk meningkatkan ke versi Kubernetes yang lebih baru.

Untuk melihat wawasan cluster EKS, Anda dapat menjalankan perintah:

```
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."
        }
    ]
}
```

Untuk keluaran yang lebih deskriptif tentang wawasan yang diterima, Anda dapat menjalankan perintah:

```
aws eks describe-insight --region <region-code> --id <insight-id> --cluster-name <my-
cluster>
```

Anda juga memiliki opsi untuk melihat wawasan di Konsol Amazon EKS. Setelah memilih klaster Anda dari daftar klaster, temuan wawasan terletak di bawah `Upgrade Insights` tab.

Jika Anda menemukan wawasan cluster dengan`"status": ERROR`, Anda harus mengatasi masalah sebelum melakukan upgrade cluster. Jalankan `aws eks describe-insight` perintah yang akan membagikan saran remediasi berikut:

Sumber daya yang terpengaruh:

```
"resources": [
      {
        "insightStatus": {
          "status": "ERROR"
        },
        "kubernetesResourceUri": "/apis/policy/v1beta1/podsecuritypolicies/null"
      }
]
```

APIs usang:

```
"deprecationDetails": [
      {
        "usage": "/apis/flowcontrol.apiserver.k8s.io/v1beta2/flowschemas",
        "replacedWith": "/apis/flowcontrol.apiserver.k8s.io/v1beta3/flowschemas",
        "stopServingVersion": "1.29",
```

```
        "clientStats": [],
        "startServingReplacementVersion": "1.26"
      }
]
```

Tindakan yang disarankan untuk diambil:

```
"recommendation": "Update manifests and API clients to use newer Kubernetes APIs if
 applicable before upgrading to Kubernetes v1.26."
```

Memanfaatkan wawasan cluster melalui Konsol EKS atau CLI membantu mempercepat proses peningkatan versi kluster EKS yang berhasil. Pelajari lebih lanjut dengan sumber daya berikut: * Dokumen EKS Resmi * Blog peluncuran Cluster Insights.

K ube-no-trouble

K ube-no-trouble adalah utilitas baris perintah open source dengan perintah`kubent`. Ketika Anda menjalankan `kubent` tanpa argumen apa pun, itu akan menggunakan KubeConfig konteks Anda saat ini dan memindai cluster dan mencetak laporan dengan apa yang APIs akan ditinggalkan dan dihapus.

```
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)
```

Hal ini juga dapat digunakan untuk memindai file manifes statis dan paket helm. Disarankan untuk dijalankan `kubent` sebagai bagian dari proses integrasi berkelanjutan (CI) untuk mengidentifikasi masalah sebelum manifes digunakan. Manifestasi pemindaian juga lebih akurat daripada memindai cluster langsung.

Kube-no-trouble menyediakan contoh Akun Layanan dan Peran dengan izin yang sesuai untuk memindai klaster.

Pluto

Pilihan lain adalah pluto yang mirip dengan `kubent` karena mendukung pemindaian cluster langsung, file manifes, bagan helm dan memiliki GitHub Tindakan yang dapat Anda sertakan dalam proses CI Anda.

```
pluto detect-all-in-cluster

NAME             KIND                VERSION          REPLACEMENT   REMOVED
 DEPRECATED   REPL AVAIL
eks.privileged   PodSecurityPolicy   policy/v1beta1                 false     true
    true
```

Sumber daya

Untuk memverifikasi bahwa klaster Anda tidak menggunakan usang APIs sebelum pemutakhiran, Anda harus memantau:

- metrik `apiserver_requested_deprecated_apis` sejak 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
```

- peristiwa di log audit dengan `k8s.io/deprecated` disetel ke`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
```

Yang akan menampilkan baris jika usang APIs sedang digunakan:

```
{
    "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\"}}"
            },
[...]
```

Perbarui beban kerja Kubernetes. Gunakan kubectl-convert untuk memperbarui manifes

Setelah Anda mengidentifikasi beban kerja dan manifes apa yang perlu diperbarui, Anda mungkin perlu mengubah jenis sumber daya dalam file manifes Anda (misalnya PodSecurityPolicies ke PodSecurityStandards). Ini akan membutuhkan pembaruan spesifikasi sumber daya dan penelitian tambahan tergantung pada sumber daya apa yang sedang diganti.

Jika jenis sumber daya tetap sama tetapi versi API perlu diperbarui, Anda dapat menggunakan `kubectl-convert` perintah untuk mengonversi file manifes secara otomatis. Misalnya, untuk mengonversi Deployment yang lebih lama menjadi`apps/v1`. Untuk informasi selengkapnya, lihat Menginstal kubectl convert plugin di situs web Kubernetes.

```
kubectl-convert -f <file> --output-version <group>/<version>
```

topologySpreadConstraints Konfigurasikan PodDisruptionBudgets dan untuk memastikan ketersediaan beban kerja Anda saat bidang data ditingkatkan

Pastikan beban kerja Anda sesuai PodDisruptionBudgetsdan topologySpreadConstraintsuntuk memastikan ketersediaan beban kerja Anda saat bidang data ditingkatkan. Tidak setiap beban kerja membutuhkan tingkat ketersediaan yang sama sehingga Anda perlu memvalidasi skala dan persyaratan beban kerja Anda.

Pastikan beban kerja tersebar di beberapa Availability Zone dan pada beberapa host dengan spread topologi akan memberikan tingkat kepercayaan yang lebih tinggi bahwa beban kerja akan bermigrasi ke bidang data baru secara otomatis tanpa insiden.

Berikut adalah contoh beban kerja yang akan selalu memiliki 80% replika yang tersedia dan menyebarkan replika di seluruh zona dan host

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: myapp
spec:
  minAvailable: "80%"
  selector:
    matchLabels:
      app: myapp
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 10
  selector:
```

```
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - image: public.ecr.aws/eks-distro/kubernetes/pause:3.2
        name: myapp
        resources:
          requests:
            cpu: "1"
            memory: 256M
      topologySpreadConstraints:
      - labelSelector:
          matchLabels:
            app: host-zone-spread
        maxSkew: 2
        topologyKey: kubernetes.io/hostname
        whenUnsatisfiable: DoNotSchedule
      - labelSelector:
          matchLabels:
            app: host-zone-spread
        maxSkew: 2
        topologyKey: topology.kubernetes.io/zone
        whenUnsatisfiable: DoNotSchedule
```

AWS Resilience Hub telah menambahkan Amazon Elastic Kubernetes Service (Amazon EKS) sebagai sumber daya yang didukung. Resilience Hub menyediakan satu tempat untuk menentukan, memvalidasi, dan melacak ketahanan aplikasi Anda sehingga Anda dapat menghindari downtime yang tidak perlu yang disebabkan oleh gangguan perangkat lunak, infrastruktur, atau operasional.

Gunakan Grup Node Terkelola atau Karpenter untuk menyederhanakan peningkatan pesawat data

Grup Node Terkelola dan Karpenter keduanya menyederhanakan peningkatan node, tetapi mereka mengambil pendekatan yang berbeda.

Grup node terkelola mengotomatiskan penyediaan dan manajemen siklus hidup node. Ini berarti Anda dapat membuat, memperbarui, atau menghentikan node secara otomatis dengan satu operasi.

Dalam konfigurasi default, Karpenter secara otomatis membuat node baru menggunakan EKS Optimized AMI terbaru yang kompatibel. Saat EKS merilis EKS Optimized yang diperbarui AMIs atau cluster ditingkatkan, Karpenter akan secara otomatis mulai menggunakan gambar-gambar ini. Karpenter juga mengimplementasikan Node Expiry untuk memperbarui node.

Karpenter dapat dikonfigurasi untuk menggunakan kustom. AMIs Jika Anda menggunakan custom AMIs dengan Karpenter, Anda bertanggung jawab atas versi kubelet.

Konfirmasikan kompatibilitas versi dengan node yang ada dan bidang kontrol

Sebelum melanjutkan dengan upgrade Kubernetes di Amazon EKS, penting untuk memastikan kompatibilitas antara grup node terkelola, node yang dikelola sendiri, dan bidang kontrol. Kompatibilitas ditentukan oleh versi Kubernetes yang Anda gunakan, dan itu bervariasi berdasarkan skenario yang berbeda. Taktik:

- Kubernetes v1.28+ — * * Mulai dari Kubernetes versi 1.28 dan seterusnya, ada kebijakan versi yang lebih lunak untuk komponen inti. Secara khusus, kemiringan yang didukung antara server API Kubernetes dan kubelet telah diperluas oleh satu versi minor, dari n-2 ke n-3. Misalnya, jika versi bidang kontrol EKS Anda adalah 1,28, Anda dapat menggunakan versi kubelet dengan aman setua 1,25. Versi miring ini didukung di seluruh AWS Fargate, grup node terkelola, dan node yang dikelola sendiri. Kami sangat menyarankan untuk menyimpan versi Amazon Machine Image (AMI) Anda up-to-date untuk alasan keamanan. Versi kubelet yang lebih lama mungkin menimbulkan risiko keamanan karena potensi Kerentanan Umum dan Eksposur (CVEs), yang bisa lebih besar daripada manfaat menggunakan versi kubelet yang lebih lama.
- Kubernetes < v1.28 - Jika Anda menggunakan versi yang lebih lama dari v1.28, kemiringan yang didukung antara server API dan kubelet adalah n-2. Misalnya, jika versi EKS Anda adalah 1.27, versi kubelet tertua yang dapat Anda gunakan adalah 1.25. Kemiringan versi ini berlaku di seluruh AWS Fargate, grup node terkelola, dan node yang dikelola sendiri.

Aktifkan kedaluwarsa node untuk node terkelola Karpenter

Salah satu cara Karpenter mengimplementasikan upgrade node adalah menggunakan konsep node expiry. Ini mengurangi perencanaan yang diperlukan untuk peningkatan node. Saat Anda menetapkan nilai untuk ttlSecondsUntilExpired di penyedia Anda, ini mengaktifkan kedaluwarsa node. Setelah node mencapai usia yang ditentukan dalam hitungan detik, node tersebut terkuras

dan dihapus dengan aman. Ini benar bahkan jika sedang digunakan, memungkinkan Anda untuk mengganti node dengan instance upgrade yang baru disediakan. Ketika sebuah node diganti, Karpenter menggunakan EKS terbaru yang dioptimalkan. AMIs Untuk informasi lebih lanjut, lihat Deprovisioning di situs web Karpenter.

Karpenter tidak secara otomatis menambahkan jitter ke nilai ini. Untuk mencegah gangguan beban kerja yang berlebihan, tentukan anggaran gangguan pod, seperti yang ditunjukkan dalam dokumentasi Kubernetes.

Jika Anda mengonfigurasi ttlSecondsUntilKedaluwarsa pada penyedia, ini berlaku untuk node yang ada yang terkait dengan penyedia.

Gunakan fitur Drift untuk node terkelola Karpenter

Fitur Drift Karpenter dapat secara otomatis meningkatkan node yang disediakan Karpenter agar tetap sinkron dengan bidang kontrol EKS. Karpenter Drift saat ini perlu diaktifkan menggunakan gerbang fitur. Konfigurasi default Karpenter menggunakan AMI terbaru yang dioptimalkan EKS untuk versi mayor dan minor yang sama dengan bidang kontrol kluster EKS.

Setelah upgrade EKS Cluster selesai, fitur Drift Karpenter akan mendeteksi bahwa node yang disediakan Karpenter menggunakan EKS yang dioptimalkan AMIs untuk versi cluster sebelumnya, dan secara otomatis mengikat, menguras, dan mengganti node tersebut. Untuk mendukung pod pindah ke node baru, ikuti praktik terbaik Kubernetes dengan menetapkan kuota sumber daya pod yang sesuai, dan menggunakan anggaran gangguan pod (PDB). Deprovisioning Karpenter akan melakukan pra-putaran node pengganti berdasarkan permintaan sumber daya pod, dan akan menghormati saat melakukan deprovisioning node. PDBs

Gunakan eksctl untuk mengotomatiskan peningkatan untuk grup node yang dikelola sendiri

Grup node yang dikelola sendiri adalah instans EC2 yang digunakan di akun Anda dan dilampirkan ke cluster di luar layanan EKS. Ini biasanya digunakan dan dikelola oleh beberapa bentuk perkakas otomatisasi. Untuk memutakhirkan grup node yang dikelola sendiri, Anda harus merujuk ke dokumentasi alat Anda.

Misalnya, eksctl mendukung penghapusan dan pengeringan node yang dikelola sendiri.

Beberapa alat umum meliputi:

- eksctl
- KOP
- Cetak Biru EKS

Backup cluster sebelum upgrade

Versi baru Kubernetes memperkenalkan perubahan signifikan pada klaster Amazon EKS Anda. Setelah Anda memutakhirkan klaster, Anda tidak dapat menurunkannya.

Velero adalah alat sumber terbuka yang didukung komunitas yang dapat digunakan untuk mengambil cadangan cluster yang ada dan menerapkan cadangan ke cluster baru.

Perhatikan bahwa Anda hanya dapat membuat cluster baru untuk versi Kubernetes yang saat ini didukung oleh EKS. Jika versi yang sedang dijalankan klaster Anda masih didukung dan pemutakhiran gagal, Anda dapat membuat klaster baru dengan versi asli dan memulihkan bidang data. Perhatikan bahwa sumber daya AWS, termasuk IAM, tidak disertakan dalam cadangan oleh Velero. Sumber daya ini perlu diciptakan kembali.

Mulai ulang penerapan Fargate setelah memutakhirkan bidang kontrol

Untuk memutakhirkan node bidang data Fargate, Anda perlu menerapkan kembali beban kerja. Anda dapat mengidentifikasi beban kerja mana yang berjalan pada node fargate dengan mencantumkan semua pod dengan opsi tersebut. `-o wide` Setiap nama node yang dimulai dengan `fargate-` perlu digunakan kembali di cluster.

Evaluasi Blue/Green Cluster sebagai alternatif untuk peningkatan klaster di tempat

Beberapa pelanggan lebih suka melakukan strategi blue/green upgrade. Ini dapat memiliki manfaat, tetapi juga termasuk kerugian yang harus dipertimbangkan.

Manfaatnya meliputi:

- Kemungkinan untuk mengubah beberapa versi EKS sekaligus (misalnya 1.23 ke 1.25)
- Mampu beralih kembali ke cluster lama

- Membuat cluster baru yang dapat dikelola dengan sistem yang lebih baru (misalnya terraform)
- Beban kerja dapat dimigrasikan secara individual

Beberapa kelemahan meliputi:

- Titik akhir API dan perubahan OIDC yang memerlukan pembaruan konsumen (misalnya kubectl dan CI/CD)
- Membutuhkan 2 cluster untuk dijalankan secara paralel selama migrasi, yang bisa mahal dan membatasi kapasitas wilayah
- Diperlukan lebih banyak koordinasi jika beban kerja saling bergantung satu sama lain untuk dimigrasikan bersama
- Load balancer dan DNS eksternal tidak dapat dengan mudah menjangkau beberapa cluster

Meskipun strategi ini mungkin dilakukan, ini lebih mahal daripada peningkatan di tempat dan membutuhkan lebih banyak waktu untuk koordinasi dan migrasi beban kerja. Ini mungkin diperlukan dalam beberapa situasi dan harus direncanakan dengan hati-hati.

Dengan otomatisasi tingkat tinggi dan sistem deklaratif seperti GitOps, ini mungkin lebih mudah dilakukan. Anda perlu mengambil tindakan pencegahan tambahan untuk beban kerja stateful sehingga data dicadangkan dan dimigrasikan ke cluster baru.

Tinjau posting blog ini untuk informasi lebih lanjut:

- Peningkatan klaster Kubernetes: strategi penerapan biru-hijau
- Biru/Hijau atau Canary Amazon EKS mengelompokkan migrasi untuk beban kerja ArgoCD tanpa kewarganegaraan

Lacak perubahan besar yang direncanakan dalam proyek Kubernetes — Pikirkan ke depan

Jangan hanya melihat versi berikutnya. Tinjau versi baru Kubernetes saat dirilis, dan identifikasi perubahan besar. Misalnya, beberapa aplikasi langsung menggunakan docker API, dan dukungan untuk Container Runtime Interface (CRI) untuk Docker (juga dikenal sebagai Dockershim) telah dihapus di Kubernetes. `1.24` Perubahan semacam ini membutuhkan lebih banyak waktu untuk dipersiapkan.

Tinjau semua perubahan terdokumentasi untuk versi yang Anda upgrade, dan perhatikan setiap langkah pemutakhiran yang diperlukan. Juga, perhatikan persyaratan atau prosedur apa pun yang khusus untuk klaster terkelola Amazon EKS.

- Changelog Kubernetes

Panduan Khusus tentang Penghapusan Fitur

Penghapusan Dockershim di 1.25 - Gunakan Detektor untuk Docker Socket (DDS)

EKS Optimized AMI untuk 1.25 tidak lagi menyertakan dukungan untuk Dockershim. Jika Anda memiliki ketergantungan pada Dockershim, misalnya Anda memasang soket Docker, Anda harus menghapus dependensi tersebut sebelum memutakhirkan node pekerja Anda ke 1,25.

Temukan contoh di mana Anda memiliki ketergantungan pada soket Docker sebelum memutakhirkan ke 1,25. Sebaiknya gunakan Detector for Docker Socket (DDS), sebuah plugin kubectl. .

Penghapusan PodSecurityPolicy in 1.25 - Migrasi ke Standar Keamanan Pod atau solusi policy-as-code

`PodSecurityPolicy`tidak digunakan lagi di Kubernetes 1.21, dan telah dihapus di Kubernetes 1.25. Jika Anda menggunakan PodSecurityPolicy klaster Anda, maka Anda harus bermigrasi ke Kubernetes Pod Security Standards (PSS) bawaan atau ke policy-as-code solusi sebelum mengupgrade klaster Anda ke versi 1.25 untuk menghindari gangguan pada beban kerja Anda.

AWS menerbitkan FAQ terperinci dalam dokumentasi EKS.

Tinjau praktik terbaik Pod Security Standards (PSS) dan Pod Security Admission (PSA).

Tinjau posting blog PodSecurityPolicy Deprecation di situs web Kubernetes.

Pengakhiran Driver Penyimpanan In-Tree di 1.23 - Migrasi ke Driver Container Storage Interface (CSI)

Container Storage Interface (CSI) dirancang untuk membantu Kubernetes mengganti mekanisme driver penyimpanan in-tree yang ada. Fitur migrasi antarmuka penyimpanan kontainer (CSI) Amazon EBS diaktifkan secara default di Amazon EKS `1.23` dan kluster yang lebih baru. Jika Anda memiliki

pod yang berjalan pada versi `1.22` atau klaster sebelumnya, maka Anda harus menginstal driver Amazon EBS CSI sebelum memperbarui klaster Anda ke versi `1.23` untuk menghindari gangguan layanan.

Tinjau pertanyaan umum migrasi Amazon EBS CSI.

Sumber Daya Tambahan

ClowdHaus Panduan Peningkatan EKS

ClowdHaus Panduan Peningkatan EKS adalah CLI untuk membantu meningkatkan kluster Amazon EKS. Ini dapat menganalisis cluster untuk setiap masalah potensial untuk diperbaiki sebelum upgrade.

GoNoGo

GoNoGoadalah alat tahap alfa untuk menentukan kepercayaan peningkatan add-on cluster Anda.

Praktik Terbaik untuk Optimalisasi Biaya

Optimalisasi Biaya adalah mencapai hasil bisnis Anda pada titik harga terendah. Dengan mengikuti dokumentasi dalam panduan ini, Anda akan mengoptimalkan beban kerja Amazon EKS Anda.

Pedoman Umum

Di cloud, ada sejumlah pedoman umum yang dapat membantu Anda mencapai optimalisasi biaya layanan mikro Anda:

- Pastikan beban kerja yang berjalan di Amazon EKS tidak tergantung pada jenis infrastruktur khusus untuk menjalankan kontainer Anda, ini akan memberikan fleksibilitas yang lebih besar sehubungan dengan menjalankannya pada jenis infrastruktur yang paling murah. Saat menggunakan Amazon EKS dengan EC2, mungkin ada pengecualian ketika kami memiliki beban kerja yang memerlukan tipe EC2 Instance tertentu seperti memerlukan GPU atau jenis instans lainnya, karena sifat beban kerjanya.
- Pilih instans kontainer yang diprofilkan secara optimal — buat profil lingkungan produksi atau pra-produksi Anda dan pantau metrik penting seperti CPU dan memori, menggunakan layanan seperti Amazon Container CloudWatch Insights untuk Amazon EKS atau alat pihak ketiga yang tersedia di ekosistem Kubernetes. Ini akan memastikan bahwa kami dapat mengalokasikan jumlah sumber daya yang tepat dan menghindari pemborosan sumber daya.
- Manfaatkan berbagai opsi pembelian yang tersedia di AWS untuk menjalankan EKS EC2, misalnya On-Demand, Spot, dan Savings Plan.

Praktik Terbaik Optimalisasi Biaya EKS

Ada tiga area praktik terbaik umum untuk pengoptimalan biaya di cloud:

- Sumber daya hemat biaya (Auto Scaling, Down Scaling, Kebijakan, dan Opsi Pembelian)
- Kesadaran pengeluaran (Menggunakan AWS dan alat pihak ketiga)
- Mengoptimalkan dari waktu ke waktu (Ukuran yang tepat)

Seperti halnya panduan apa pun, ada trade-off. Pastikan Anda bekerja dengan organisasi Anda untuk memahami prioritas beban kerja ini dan praktik terbaik mana yang paling penting.

Cara menggunakan panduan ini

Panduan ini ditujukan untuk tim devops yang bertanggung jawab untuk mengimplementasikan dan mengelola kluster EKS dan beban kerja yang mereka dukung. Panduan ini diatur ke dalam area praktik terbaik yang berbeda untuk konsumsi yang lebih mudah. Setiap topik memiliki daftar rekomendasi, alat untuk digunakan, dan praktik terbaik untuk optimalisasi biaya kluster EKS Anda. Topik tidak perlu dibaca dalam urutan tertentu.

Layanan AWS Utama dan fitur Kubernetes

Optimalisasi biaya didukung oleh layanan dan fitur AWS berikut:

- EC2 Jenis instans, Savings Plan (dan Instans Cadangan) dan Instans Spot, dengan harga berbeda.
- Auto Scaling bersama dengan kebijakan Auto Scaling asli Kubernetes. Pertimbangkan Savings Plan (Previous Reserved Instances) untuk beban kerja yang dapat diprediksi. Gunakan penyimpanan data terkelola seperti EBS dan EFS, untuk elastisitas dan daya tahan data aplikasi.
- Dasbor konsol Billing and Cost Management bersama AWS Cost Explorer memberikan gambaran umum tentang penggunaan AWS Anda. Gunakan AWS Organizations untuk detail penagihan terperinci. Rincian beberapa alat pihak ketiga juga telah dibagikan.
- Amazon CloudWatch Container Metrics menyediakan metrik seputar penggunaan sumber daya oleh kluster EKS. Selain dasbor Kubernetes, ada beberapa alat di ekosistem Kubernetes yang dapat digunakan untuk mengurangi pemborosan.

Panduan ini mencakup serangkaian rekomendasi yang dapat Anda gunakan untuk meningkatkan optimalisasi biaya kluster Amazon EKS Anda.

Umpan Balik

Panduan ini sedang GitHub dirilis untuk mengumpulkan umpan balik langsung dan saran dari komunitas EKS/Kubernetes yang lebih luas. Jika Anda memiliki praktik terbaik yang menurut Anda harus kami sertakan dalam panduan ini, harap ajukan masalah atau kirimkan PR di GitHub repositori. Tujuan kami adalah untuk memperbarui panduan secara berkala karena fitur baru ditambahkan ke layanan atau ketika praktik terbaik baru berkembang.

Kerangka Optimalisasi Biaya

AWS Cloud Economics adalah disiplin yang membantu pelanggan meningkatkan efisiensi dan mengurangi biaya mereka melalui adopsi teknologi komputasi modern seperti Amazon EKS. Disiplin merekomendasikan mengikuti metodologi yang disebut “Cloud Financial Management (CFM) framework” yang terdiri dari 4 pilar:

Pilar See: Pengukuran dan akuntabilitas

Pilar See adalah serangkaian aktivitas dan teknologi dasar yang menentukan cara mengukur, memantau, dan menciptakan akuntabilitas untuk pengeluaran cloud. Ini sering disebut sebagai “Observabilitas”, “Instrumentasi”, atau “Telemetri”. Kemampuan dan keterbatasan infrastruktur “Observabilitas” menentukan apa yang dapat dioptimalkan. Mendapatkan gambaran yang jelas tentang biaya Anda adalah langkah pertama yang penting dalam pengoptimalan biaya karena Anda perlu tahu dari mana Anda memulai. Jenis visibilitas ini juga akan memandu jenis kegiatan yang perlu Anda lakukan untuk lebih mengoptimalkan lingkungan Anda.

Berikut adalah ikhtisar singkat tentang praktik terbaik kami untuk pilar See:

- Tentukan dan pertahankan strategi penandaan untuk beban kerja Anda.
  - Menggunakan Instance Tagging, menandai kluster EKS memungkinkan Anda melihat biaya cluster individual dan mengalokasikannya dalam Laporan Biaya & Penggunaan Anda.
- Menetapkan pelaporan dan pemantauan penggunaan EKS dengan menggunakan teknologi seperti Kubecost.

  - Aktifkan Dasbor Cloud Intelligence, dengan memiliki sumber daya yang ditandai dengan benar dan menggunakan visualisasi, Anda dapat mengukur dan memperkirakan biaya.
- Alokasikan biaya cloud ke aplikasi, Lines of Business (LoBs), dan aliran pendapatan.
- Mendefinisikan, mengukur, dan beredar efficiency/value KPIs dengan pemangku kepentingan bisnis. Misalnya, buat KPI “satuan metrik” yang mengukur biaya per transaksi, misalnya layanan berbagi perjalanan mungkin memiliki KPI untuk “biaya per perjalanan”.

Untuk detail lebih lanjut tentang teknologi dan aktivitas yang direkomendasikan yang terkait dengan pilar ini, silakan lihat bagian Optimalisasi Biaya - Pengamatan dari panduan ini.

Pilar Hemat: Optimalisasi biaya

Pilar ini didasarkan pada teknologi dan kemampuan yang dikembangkan di pilar “Lihat”. Kegiatan berikut biasanya termasuk dalam pilar ini:

- Identifikasi dan hilangkan limbah di lingkungan Anda.
- Arsitek dan desain untuk efisiensi biaya.
- Pilih opsi pembelian terbaik, misalnya instans sesuai permintaan vs instans Spot.
- Beradaptasi seiring berkembangnya layanan: seiring berkembangnya layanan AWS, cara untuk menggunakan layanan tersebut secara efisien dapat berubah. Bersedia beradaptasi untuk memperhitungkan perubahan ini.

Karena kegiatan ini operasional, mereka sangat tergantung pada karakteristik lingkungan Anda. Tanyakan pada diri sendiri, apa pendorong utama biaya? Nilai bisnis apa yang disediakan oleh lingkungan Anda yang berbeda? Opsi pembelian dan pilihan infrastruktur apa, misalnya tipe keluarga instance, yang paling cocok untuk setiap lingkungan?

Di bawah ini adalah daftar prioritas driver biaya paling umum untuk kluster EKS:

1. Biaya komputasi: Menggabungkan beberapa jenis keluarga instans, opsi pembelian, dan skalabilitas penyeimbangan dengan ketersediaan memerlukan pertimbangan yang cermat. Untuk informasi lebih lanjut, lihat rekomendasi di bagian Optimasi Biaya - Komputasi pada panduan ini.
2. Biaya jaringan: menggunakan 3 AZs untuk kluster EKS berpotensi meningkatkan biaya lalu lintas antar-AZ. Untuk rekomendasi kami tentang cara menyeimbangkan persyaratan HA dengan menjaga biaya lalu lintas jaringan tetap rendah, silakan lihat bagian Pengoptimalan Biaya - Jaringan dari panduan ini.

3. Biaya penyimpanan: Tergantung pada stateful/stateless sifat beban kerja di kluster EKS, dan bagaimana berbagai jenis penyimpanan digunakan, penyimpanan dapat dianggap sebagai bagian dari beban kerja. Untuk pertimbangan yang berkaitan dengan biaya penyimpanan EKS, silakan baca bagian Pengoptimalan Biaya - Penyimpanan dari panduan ini.

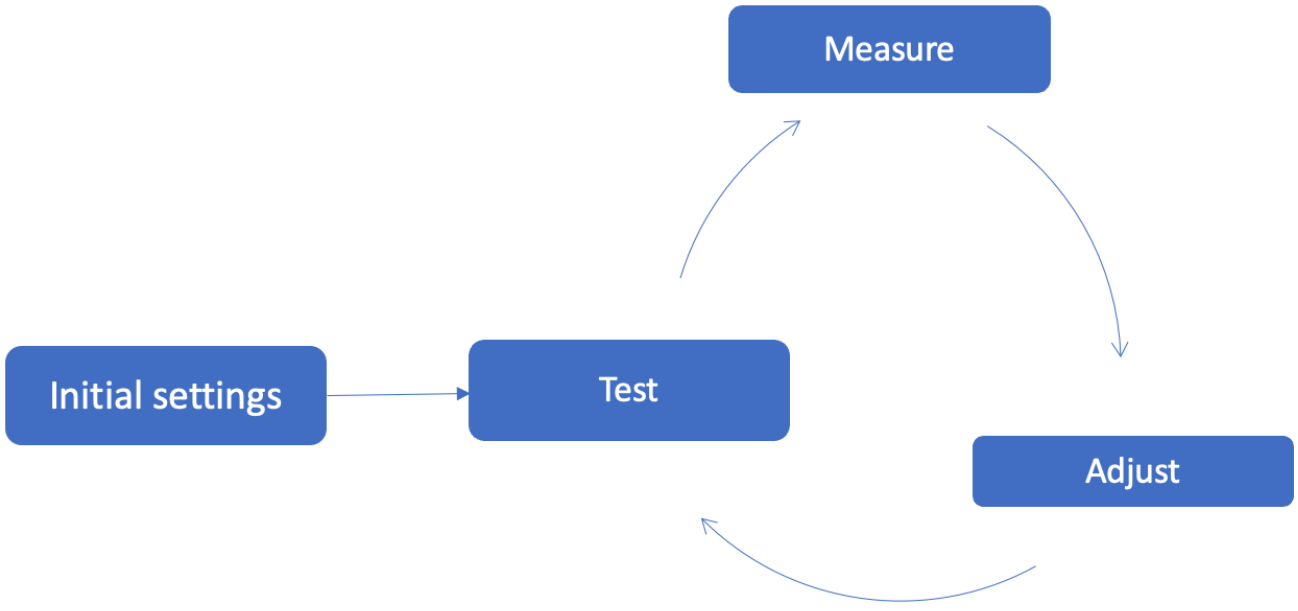
Pilar Rencana: Perencanaan dan peramalan

Setelah rekomendasi di pilar See diimplementasikan, cluster dioptimalkan secara berkelanjutan. Karena pengalaman diperoleh dalam mengoperasikan cluster secara efisien, kegiatan perencanaan dan peramalan dapat berfokus pada:

- Penganggaran dan perkiraan biaya cloud secara dinamis.
- Mengukur nilai bisnis yang disampaikan oleh layanan kontainer EKS.
- Mengintegrasikan manajemen biaya klaster EKS dengan perencanaan manajemen keuangan TI.

Pilar Run

Optimalisasi biaya adalah proses yang berkelanjutan dan melibatkan flywheel peningkatan inkremental:



Mengamankan sponsor eksekutif untuk jenis kegiatan ini sangat penting untuk mengintegrasikan optimasi klaster EKS ke dalam upaya "" organisasi. FinOps Hal ini memungkinkan penyelarasan pemangku kepentingan melalui pemahaman bersama tentang biaya klaster EKS, implementasi pagar biaya kluster EKS, dan memastikan bahwa perkakas, otomatisasi, dan aktivitas berkembang sesuai dengan kebutuhan organisasi.

Referensi

- AWS Cloud Economics, Manajemen Keuangan Cloud

Kesadaran pengeluaran

Kesadaran pengeluaran adalah memahami siapa, di mana dan apa yang menyebabkan pengeluaran di cluster EKS Anda. Mendapatkan gambaran yang akurat tentang data ini akan membantu meningkatkan kesadaran akan pengeluaran Anda dan menyoroti area untuk diperbaiki.

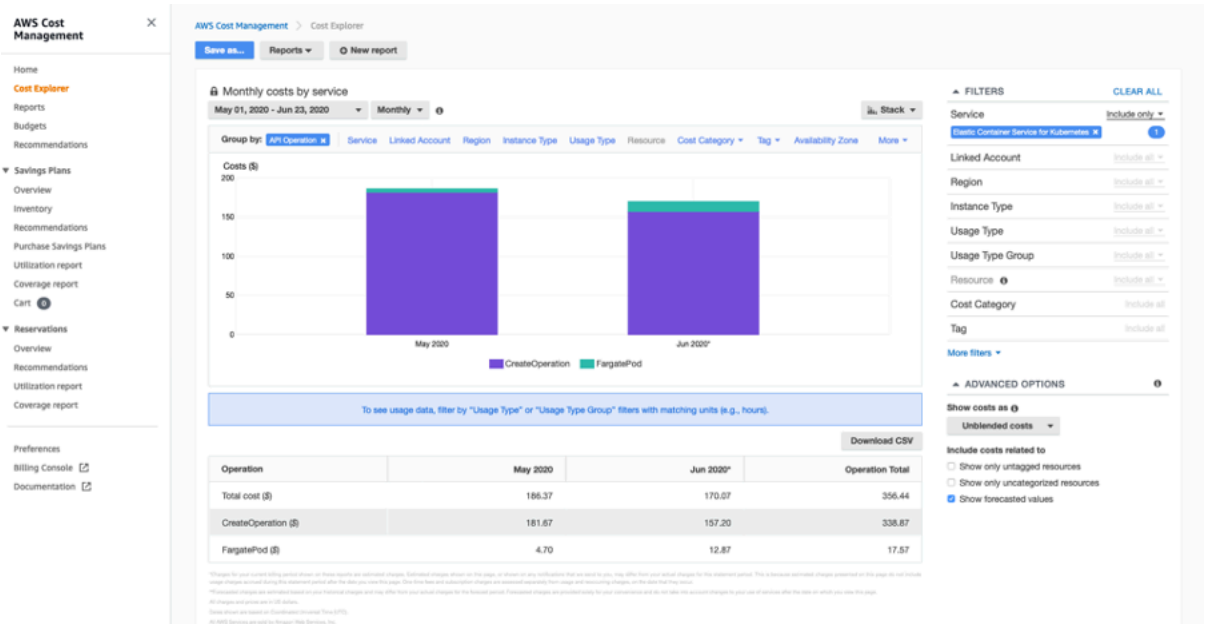
Rekomendasi

Gunakan Cost Explorer

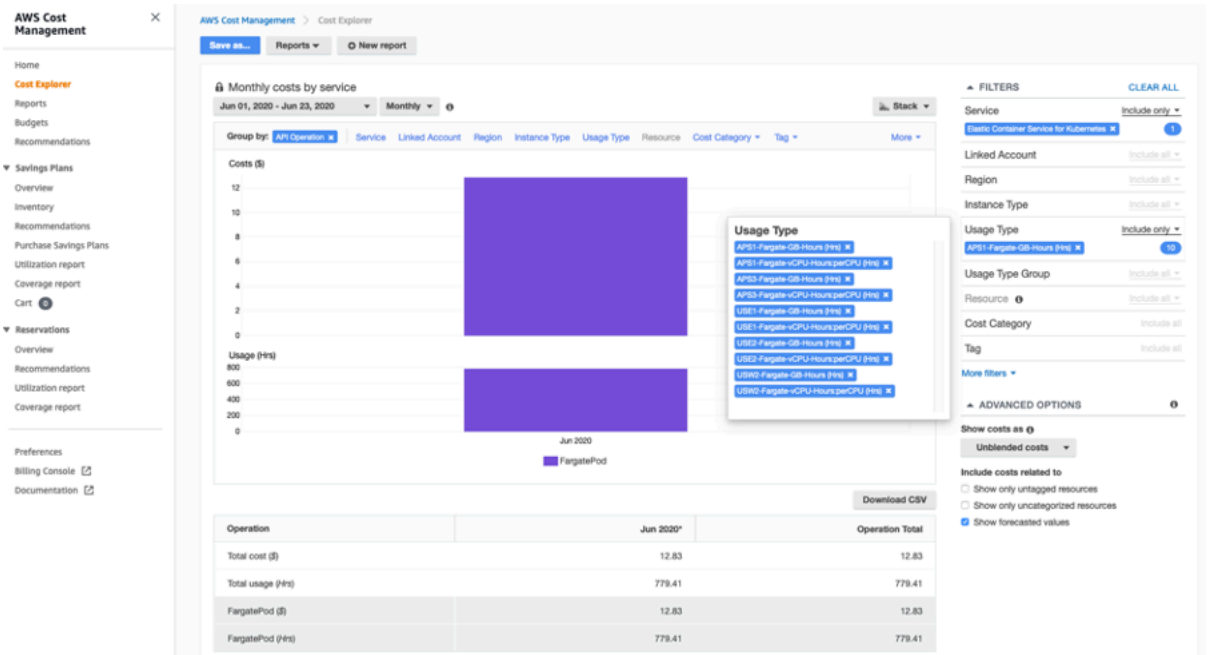
AWS Cost Explorer memiliki easy-to-use antarmuka yang memungkinkan Anda memvisualisasikan, memahami, dan mengelola biaya dan penggunaan AWS Anda dari waktu ke waktu. Anda dapat menganalisis data biaya dan penggunaan, di berbagai tingkatan menggunakan filter yang tersedia di Cost Explorer.

Biaya EKS Control Plane dan EKS Fargate

Dengan menggunakan filter, kita dapat menanyakan biaya yang dikeluarkan untuk biaya EKS di Control Plane dan Fargate Pod seperti yang ditunjukkan pada diagram di bawah ini:



Dengan menggunakan filter, kita dapat menanyakan biaya agregat yang dikeluarkan untuk Pod Fargate di seluruh wilayah di EKS - yang mencakup jam VCPU per CPU dan GB Jam seperti yang ditunjukkan pada diagram di bawah ini:



Penandaan Sumber Daya

Amazon EKS mendukung penambahan tag AWS ke kluster Amazon EKS Anda. Hal ini memudahkan untuk mengontrol akses ke EKS API untuk mengelola cluster Anda. Tag yang ditambahkan ke kluster EKS khusus untuk sumber daya kluster AWS EKS, tag tersebut tidak menyebar ke sumber daya

AWS lain yang digunakan oleh cluster seperti EC2 instance atau penyeimbang beban. Saat ini, penandaan cluster didukung untuk semua cluster EKS baru dan yang sudah ada melalui AWS API, Console, dan. SDKs

AWS Fargate adalah teknologi yang menyediakan kapasitas komputasi sesuai permintaan dan ukuran yang tepat untuk kontainer. Sebelum Anda dapat menjadwalkan pod di Fargate di klaster Anda, Anda harus menentukan setidaknya satu profil Fargate yang menentukan pod mana yang harus menggunakan Fargate ketika mereka diluncurkan.

Menambahkan dan mencantumkan tag ke kluster 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"
    }
}
```

Setelah Anda mengaktifkan tag alokasi biaya di AWS Cost Explorer, AWS menggunakan tag alokasi biaya untuk mengatur biaya sumber daya pada laporan alokasi biaya, untuk memudahkan Anda mengkategorikan dan melacak biaya AWS Anda.

Tag tidak memiliki arti semantik untuk Amazon EKS dan ditafsirkan secara ketat sebagai serangkaian karakter. Misalnya, Anda dapat menentukan satu set tag untuk kluster Amazon EKS Anda untuk membantu Anda melacak setiap pemilik klaster dan tingkat tumpukan.

Gunakan AWS Trusted Advisor

AWS Trusted Advisor menawarkan serangkaian pemeriksaan dan rekomendasi praktik terbaik di lima kategori: pengoptimalan biaya; keamanan; toleransi kesalahan; kinerja; dan batas layanan.

Untuk Optimalisasi Biaya, Trusted Advisor membantu menghilangkan sumber daya yang tidak digunakan dan tidak digunakan dan merekomendasikan untuk membuat komitmen terhadap kapasitas cadangan. Item tindakan utama yang akan membantu Amazon EKS akan berada di sekitar EC2 instance yang digunakan rendah, alamat IP Elastis yang tidak terkait, Penyeimbang Beban

Idle, volume EBS yang kurang dimanfaatkan, antara lain. Daftar lengkap pemeriksaan disediakan di https://aws.amazon.com/premiumsupport/ technology/trusted-advisor/best -practice-checklist/.

Trusted Advisor juga menyediakan rekomendasi Savings Plans dan Reserved EC2 Instances untuk instans dan Fargate yang memungkinkan Anda untuk berkomitmen pada jumlah penggunaan yang konsisten dengan imbalan tarif diskon.

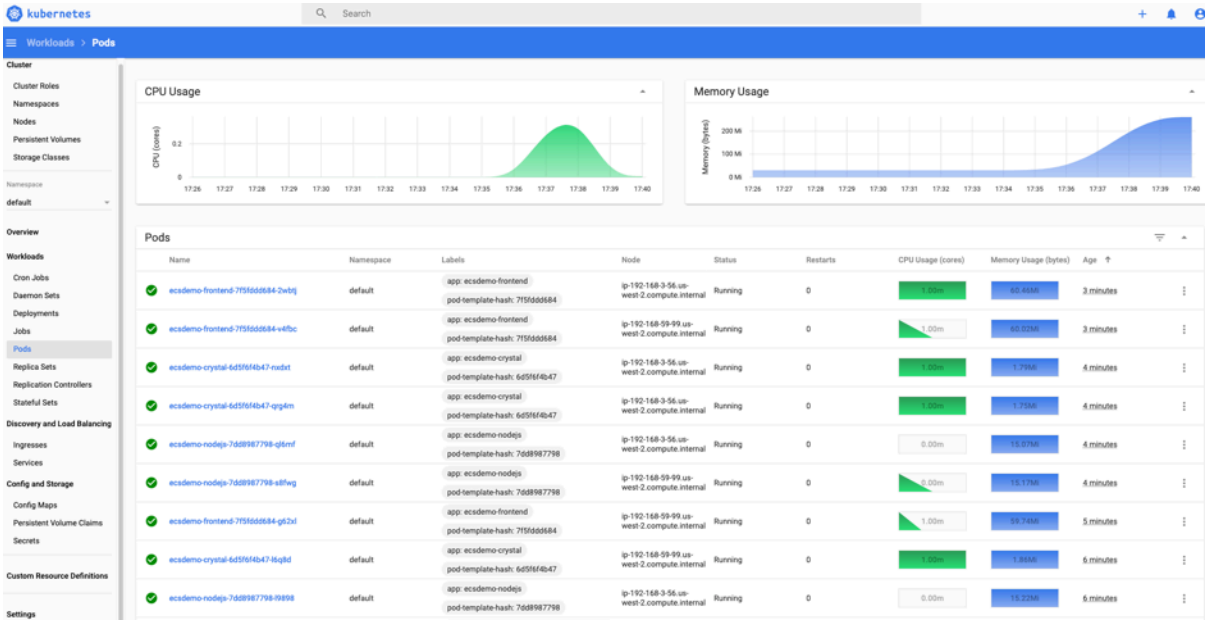
> **Note**
>
> Rekomendasi dari Trusted Advisor adalah rekomendasi generik dan tidak spesifik untuk EKS.

Gunakan dasbor Kubernetes

Dasbor Kubernetes

Dasbor Kubernetes adalah UI berbasis web untuk klaster Kubernetes, yang menyediakan informasi tentang klaster Kubernetes termasuk penggunaan sumber daya pada tingkat cluster, node, dan pod. Penerapan dasbor Kubernetes pada kluster Amazon EKS dijelaskan dalam dokumentasi Amazon EKS.

Dashboard menyediakan rincian penggunaan sumber daya untuk setiap node dan pod, serta metadata rinci tentang pod, layanan, Deployment, dan objek Kubernetes lainnya. Informasi terkonsolidasi ini memberikan visibilitas ke lingkungan Kubernetes Anda.



kubectl top dan jelaskan perintah

Melihat metrik penggunaan sumber daya dengan perintah kubectl top dan kubectl describe. kubectl top akan menampilkan penggunaan CPU dan memori saat ini untuk pod atau node di seluruh klaster Anda, atau untuk pod atau node tertentu. Perintah kubectl describe akan memberikan informasi yang lebih rinci tentang node atau pod tertentu.

```
$ kubectl top pods
$ kubectl top nodes
$ kubectl top pod pod-name --namespace mynamespace --containers
```

Dengan menggunakan perintah atas, output akan menampilkan jumlah total CPU (dalam inti) dan memori (dalam MiB) yang digunakan node, dan persentase kapasitas node yang dapat dialokasikan oleh angka-angka tersebut. Anda kemudian dapat menelusuri ke level berikutnya, level kontainer di dalam pod dengan menambahkan flag --containers.

```
$ kubectl describe node <node>
$ kubectl describe pod <pod>
```

kubectl describe mengembalikan persen dari total kapasitas yang tersedia yang diwakili oleh setiap permintaan atau batas sumber daya.

kubectl top dan jelaskan, lacak pemanfaatan dan ketersediaan sumber daya penting seperti CPU, memori, dan penyimpanan di seluruh pod, node, dan container kubernetes. Kesadaran ini akan membantu dalam memahami penggunaan sumber daya dan membantu dalam mengendalikan biaya.

Gunakan CloudWatch Wawasan Kontainer

Gunakan CloudWatch Wawasan Kontainer untuk mengumpulkan, menggabungkan, dan meringkas metrik dan log dari aplikasi dan layanan mikro dalam kontainer Anda. Container Insights tersedia untuk Amazon Elastic Kubernetes EC2 Service di, dan platform Kubernetes di Amazon. EC2 Metrik tersebut mencakup pemanfaatan sumber daya seperti CPU, memori, disk, dan jaringan.

Pemasangan wawasan diberikan dalam dokumentasi.

CloudWatch membuat metrik agregat di tingkat cluster, node, pod, task, dan service sebagai CloudWatch metrik.

Kueri berikut menunjukkan daftar node, diurutkan berdasarkan pemanfaatan CPU node rata-rata

```
STATS avg(node_cpu_utilization) as avg_node_cpu_utilization by NodeName
| SORT avg_node_cpu_utilization DESC
```

Penggunaan CPU dengan nama Container

```
stats pct(container_cpu_usage_total, 50) as CPUPercMedian by kubernetes.container_name
| filter Type="Container"
```

Penggunaan disk dengan nama Container

```
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
```

Lebih banyak contoh kueri diberikan dalam dokumen Container Insights

Kesadaran ini akan membantu dalam memahami penggunaan sumber daya dan membantu dalam mengendalikan biaya.

Menggunakan Kubecost untuk kesadaran dan bimbingan pengeluaran

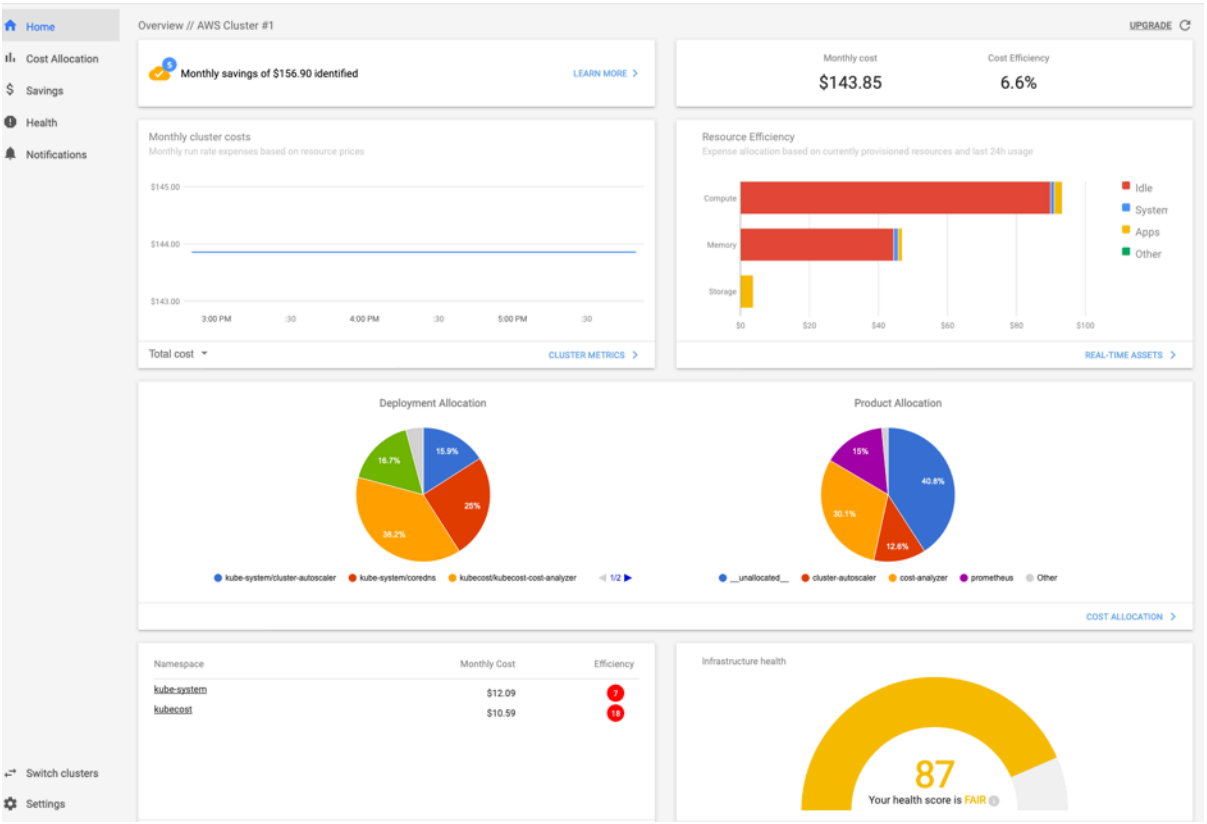
Alat pihak ketiga seperti kubecost juga dapat digunakan di Amazon EKS untuk mendapatkan visibilitas terhadap biaya menjalankan klaster Kubernetes Anda. Silakan merujuk ke blog AWS ini untuk melacak biaya menggunakan Kubecost

Menerapkan kubecost menggunakan 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
```

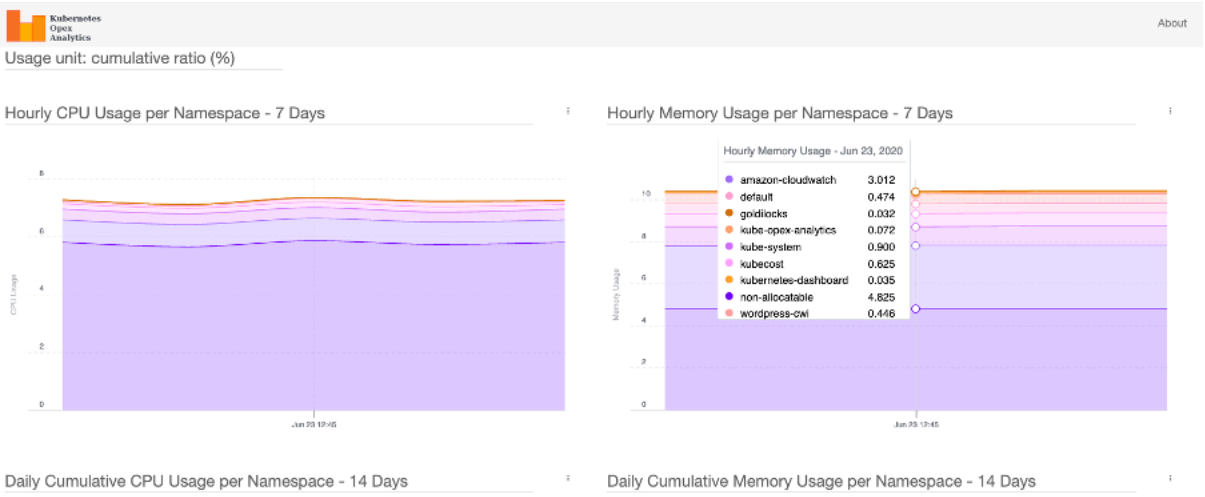
Dasbor Kubecost -



Gunakan Alat Analisis Alokasi Biaya dan Perencanaan Kapasitas Kubernetes

Kubernetes Opex Analytics adalah alat untuk membantu organisasi melacak sumber daya yang dikonsumsi oleh klaster Kubernetes mereka untuk mencegah pembayaran lebih. Untuk

melakukannya, laporan penggunaan jangka pendek (7 hari), menengah (14 hari) dan jangka panjang (12 bulan) yang menunjukkan wawasan yang relevan tentang jumlah sumber daya yang dihabiskan setiap proyek dari waktu ke waktu.



Yotascale

Yotascale membantu mengalokasikan biaya Kubernetes secara akurat. Fitur Alokasi Biaya Yotascale Kubernetes menggunakan data biaya aktual, yang mencakup diskon Instans Cadangan dan harga instans spot alih-alih estimasi tarif pasar umum, untuk menginformasikan total jejak biaya Kubernetes

Rincian lebih lanjut dapat ditemukan di situs web mereka.

Penasihat Alcide

Alcide adalah Mitra Teknologi Lanjutan AWS Partner Network (APN). Alcide Advisor membantu memastikan konfigurasi klaster, node, dan pod Amazon EKS Anda disetel agar berjalan sesuai dengan praktik terbaik keamanan dan pedoman internal. Alcide Advisor adalah layanan tanpa agen untuk audit dan kepatuhan Kubernetes yang dibangun untuk memastikan aliran tanpa gesekan dan aman DevSecOps dengan memperkuat tahap pengembangan sebelum pindah ke produksi.

Detail lebih lanjut dapat ditemukan di posting blog ini.

Alat-alat lainnya

Pengumpulan Sampah Kubernetes

Peran pengumpul sampah Kubernetes adalah menghapus benda-benda tertentu yang pernah memiliki pemilik, tetapi tidak lagi memiliki pemilik.

Hitungan Fargate

Fargatecount adalah alat yang berguna, yang memungkinkan pelanggan AWS melacak, dengan CloudWatch metrik khusus, jumlah total pod EKS yang telah digunakan di Fargate di wilayah tertentu dari akun tertentu. Ini membantu melacak semua pod Fargate yang berjalan di cluster EKS.

Popeye - Pembersih Cluster Kubernetes

Popeye - Kubernetes Cluster Sanitizer adalah utilitas yang memindai klaster Kubernetes secara langsung dan melaporkan potensi masalah dengan sumber daya dan konfigurasi yang diterapkan. Ini membersihkan cluster Anda berdasarkan apa yang digunakan dan bukan apa yang ada di disk. Dengan memindai klaster Anda, ia mendeteksi kesalahan konfigurasi dan membantu Anda memastikan bahwa praktik terbaik sudah ada

Sumber daya

Lihat sumber daya berikut untuk mempelajari lebih lanjut tentang praktik terbaik untuk pengoptimalan biaya.

Dokumentasi dan Blog

- Amazon EKS mendukung penandaan

Alat

- Apa itu AWS Billing and Cost Management?
- CloudWatch Wawasan Kontainer Amazon
- Cara melacak biaya di klaster Amazon EKS multi-tenant menggunakan Kubecost
- Kubecost
- Kube Oppsview
- Kubernetes Opex Analitik

Komputasi dan Penskalaan Otomatis

Sebagai pengembang, Anda akan membuat perkiraan tentang kebutuhan sumber daya aplikasi Anda, misalnya CPU dan memori, tetapi jika Anda tidak terus-menerus menyesuaikannya, mereka mungkin menjadi usang yang dapat meningkatkan biaya dan memperburuk kinerja dan keandalan.

Menyesuaikan kebutuhan sumber daya aplikasi secara terus-menerus lebih penting daripada memperbaikinya pertama kali.

Praktik terbaik yang disebutkan di bawah ini akan membantu Anda membangun dan mengoperasikan beban kerja sadar biaya yang mencapai hasil bisnis sambil meminimalkan biaya dan memungkinkan organisasi Anda memaksimalkan laba atas investasinya. Urutan tingkat tinggi yang penting untuk mengoptimalkan biaya komputasi cluster Anda adalah:

1. Beban kerja ukuran kanan
2. Kurangi kapasitas yang tidak terpakai
3. Optimalkan jenis kapasitas komputasi (misalnya Spot) dan akselerator (mis.) GPUs

Ukuran beban kerja Anda dengan benar

Di sebagian besar kluster EKS, sebagian besar biaya berasal dari EC2 instance yang digunakan untuk menjalankan beban kerja kontainer Anda. Anda tidak akan dapat mengukur sumber daya komputasi Anda dengan benar tanpa memahami persyaratan beban kerja Anda. Inilah sebabnya mengapa penting bagi Anda untuk menggunakan permintaan dan batasan yang sesuai dan membuat penyesuaian pada pengaturan tersebut seperlunya. Selain itu, dependensi, seperti ukuran instans dan pemilihan penyimpanan, dapat mempengaruhi kinerja beban kerja yang dapat memiliki berbagai konsekuensi yang tidak diinginkan pada biaya dan keandalan.

Permintaan harus selaras dengan pemanfaatan yang sebenarnya. Jika permintaan kontainer terlalu tinggi akan ada kapasitas yang tidak terpakai yang merupakan faktor besar dalam total biaya cluster. Setiap kontainer dalam sebuah pod, misalnya application dan sidecars, harus memiliki permintaan dan batasannya sendiri yang ditetapkan untuk memastikan batas agregat pod seakurat mungkin.

Gunakan alat seperti Goldilocks, KRR, dan Kubecost yang memperkirakan permintaan sumber daya dan batasan untuk kontainer Anda. Bergantung pada sifat aplikasi, performance/cost persyaratan, dan kompleksitasnya, Anda perlu mengevaluasi metrik mana yang terbaik untuk diskalakan, pada titik mana kinerja aplikasi Anda menurun (titik saturasi), dan cara mengubah permintaan dan batasan yang sesuai. Silakan merujuk ke Ukuran aplikasi yang tepat untuk panduan lebih lanjut tentang topik ini.

Sebaiknya gunakan Horizontal Pod Autoscaler (HPA) untuk mengontrol berapa banyak replika aplikasi Anda yang harus dijalankan, Vertical Pod Autoscaler (VPA) untuk menyesuaikan berapa banyak permintaan dan membatasi kebutuhan aplikasi Anda per replika, dan autoscaler node seperti Karpenter atau Cluster Autoscaler untuk terus menyesuaikan jumlah total node di cluster Anda.

Teknik pengoptimalan biaya menggunakan Karpenter dan Cluster Autoscaler didokumentasikan di bagian selanjutnya dari dokumen ini.

Vertical Pod Autoscaler dapat menyesuaikan permintaan dan batasan yang ditetapkan ke kontainer sehingga beban kerja berjalan secara optimal. Anda harus menjalankan VPA dalam mode audit sehingga tidak secara otomatis membuat perubahan dan memulai ulang pod Anda. Ini akan menyarankan perubahan berdasarkan metrik yang diamati. Dengan perubahan apa pun yang memengaruhi beban kerja produksi, Anda harus meninjau dan menguji perubahan tersebut terlebih dahulu di lingkungan non-produksi karena hal ini dapat berdampak pada keandalan dan kinerja aplikasi Anda.

Kurangi konsumsi

Cara terbaik untuk menghemat uang adalah dengan menyediakan lebih sedikit sumber daya. Salah satu cara untuk melakukannya adalah dengan menyesuaikan beban kerja berdasarkan kebutuhan mereka saat ini. Anda harus memulai upaya pengoptimalan biaya dengan memastikan beban kerja Anda menentukan persyaratan dan skalanya secara dinamis. Ini akan membutuhkan metrik dari aplikasi Anda dan pengaturan konfigurasi seperti `PodDisruptionBudgets`dan Pod Readiness Gates untuk memastikan aplikasi Anda dapat dengan aman menskalakan naik dan turun secara dinamis. Penting untuk dipertimbangkan bahwa restriktif PodDisruptionBudgets dapat mencegah Cluster Autoscaler dan Karpenter mengurangi Node, karena keduanya menghormati Cluster Autoscaler dan Karpenter. PodDisruptionBudgets Nilai 'minAvailable' dalam PodDisruptionBudget harus selalu lebih rendah dari jumlah pod dalam penerapan dan Anda harus menyimpan buffer yang baik di antara keduanya misalnya Dalam penerapan 6 pod di mana Anda ingin minimal 4 pod berjalan setiap saat, atur 'minAvailable' di Anda ke 4. PodDisruptionBidget Ini akan memungkinkan Cluster Autoscaler dan Karpenter untuk dengan aman mengalirkan dan mengusir pod dari node yang kurang dimanfaatkan selama acara scale-down Node. Silakan merujuk ke dokumen FAQ Cluster Autoscaler.

Horizontal Pod Autoscaler adalah autoscaler beban kerja fleksibel yang dapat menyesuaikan berapa banyak replika yang diperlukan untuk memenuhi persyaratan kinerja dan keandalan aplikasi Anda. Ini memiliki model yang fleksibel untuk menentukan kapan harus naik dan turun berdasarkan berbagai metrik seperti CPU, memori, atau metrik khusus misalnya kedalaman antrian, jumlah koneksi ke pod, dll.

Kubernetes Metrics Server memungkinkan penskalaan sebagai respons terhadap metrik bawaan seperti penggunaan CPU dan memori, tetapi jika Anda ingin menskalakan berdasarkan metrik lain, seperti kedalaman antrian Amazon CloudWatch atau SQS, Anda harus mempertimbangkan proyek

penskalaan otomatis berbasis peristiwa seperti KEDA. Silakan lihat posting blog ini tentang cara menggunakan KEDA dengan CloudWatch metrik. Jika Anda tidak yakin, metrik mana yang akan dipantau dan diskalakan berdasarkan, lihat praktik terbaik tentang metrik pemantauan yang penting.

Mengurangi konsumsi beban kerja menciptakan kelebihan kapasitas dalam sebuah cluster dan dengan konfigurasi penskalaan otomatis yang tepat memungkinkan Anda untuk menurunkan skala node secara otomatis dan mengurangi total pengeluaran Anda. Kami menyarankan Anda untuk tidak mencoba mengoptimalkan kapasitas komputasi secara manual. Penjadwal Kubernetes dan autoscaler node dirancang untuk menangani proses ini untuk Anda.

Kurangi kapasitas yang tidak terpakai

Setelah Anda menentukan ukuran yang benar untuk aplikasi, mengurangi permintaan berlebih, Anda dapat mulai mengurangi kapasitas komputasi yang disediakan. Anda harus dapat melakukan ini secara dinamis jika Anda telah meluangkan waktu untuk mengukur beban kerja Anda dengan benar dari bagian di atas. Ada dua autoscaler node utama yang digunakan dengan Kubernetes di AWS.

Karpenter dan Cluster Autoscaler

Baik Karpenter dan Kubernetes Cluster Autoscaler akan menskalakan jumlah node di klaster Anda saat pod dibuat atau dihapus dan persyaratan komputasi berubah. Tujuan utama keduanya adalah sama, tetapi Karpenter mengambil pendekatan yang berbeda untuk penyediaan manajemen node dan de-provisioning yang dapat membantu mengurangi biaya dan mengoptimalkan penggunaan luas cluster.

Ketika cluster bertambah besar dan variasi beban kerja meningkat, menjadi lebih sulit untuk melakukan pra-konfigurasi grup dan instance node. Sama seperti permintaan beban kerja, penting untuk menetapkan garis dasar awal dan terus menyesuaikan sesuai kebutuhan.

Jika Anda menggunakan Cluster Autoscaler, itu akan menghormati nilai “minimum” dan “maksimum” dari setiap grup Auto Scaling (ASG) dan hanya menyesuaikan nilai “yang diinginkan”. Penting untuk memperhatikan saat menyetel nilai-nilai ini untuk ASG yang mendasarinya karena Cluster Autoscaler tidak akan dapat menurunkan ASG di luar jumlah “minimum”. Tetapkan jumlah “yang diinginkan” sebagai jumlah node yang Anda butuhkan selama jam kerja normal dan “minimum” sebagai jumlah node yang Anda butuhkan selama jam kerja di luar bisnis. Silakan merujuk ke dokumen FAQ Cluster Autoscaler.

Expander Prioritas Cluster Autoscaler

Kubernetes Cluster Autoscaler bekerja dengan menskalakan kelompok node — disebut grup node — naik dan turun saat aplikasi menskalakan naik dan turun. Jika Anda tidak menskalakan beban kerja secara dinamis maka Cluster Autoscaler tidak akan membantu Anda menghemat uang. Cluster Autoscaler memerlukan admin cluster untuk membuat grup node sebelumnya agar beban kerja dapat dikonsumsi. Grup node perlu dikonfigurasi untuk menggunakan instance yang memiliki "profil" yang sama, yaitu kira-kira jumlah CPU dan memori yang sama.

Anda dapat memiliki beberapa grup node dan Cluster Autoscaler dapat dikonfigurasi untuk menetapkan tingkat penskalaan prioritas dan setiap grup node dapat berisi node berukuran berbeda. Grup node dapat memiliki tipe kapasitas yang berbeda dan expander prioritas dapat digunakan untuk menskalakan grup yang lebih murah terlebih dahulu.

Di bawah ini adalah contoh cuplikan konfigurasi klaster yang menggunakan a `ConfigMap` untuk memprioritaskan kapasitas cadangan sebelum menggunakan instance sesuai permintaan. Anda dapat menggunakan teknik yang sama untuk memprioritaskan Graviton atau Spot Instances di atas jenis lainnya.

```
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.*
```

Menggunakan grup node dapat membantu sumber daya komputasi yang mendasarinya melakukan hal yang diharapkan secara default, misalnya menyebarkan node AZs, tetapi tidak semua beban kerja memiliki persyaratan atau harapan yang sama dan lebih baik membiarkan aplikasi mendeklarasikan persyaratannya secara eksplisit. Untuk informasi selengkapnya tentang Cluster Autoscaler, silakan lihat bagian praktik terbaik.

Descheduler

Cluster Autoscaler dapat menambah dan menghapus kapasitas node dari cluster berdasarkan pod baru yang perlu dijadwalkan atau node yang kurang dimanfaatkan. Itu tidak mengambil tampilan menyeluruh dari penempatan pod setelah dijadwalkan ke sebuah node. Jika Anda menggunakan Cluster Autoscaler, Anda juga harus melihat descheduler Kubernetes untuk menghindari pemborosan kapasitas di cluster Anda.

Jika Anda memiliki 10 node dalam sebuah cluster dan setiap node 60% digunakan, Anda tidak menggunakan 40% dari kapasitas yang disediakan di cluster. Dengan Cluster Autoscaler Anda dapat mengatur ambang batas pemanfaatan per node menjadi 60%, tetapi itu hanya akan mencoba menurunkan satu node setelah pemanfaatan turun di bawah 60%.

Dengan descheduler dapat melihat kapasitas dan pemanfaatan cluster setelah pod dijadwalkan atau node telah ditambahkan ke cluster. Ini mencoba untuk menjaga kapasitas total cluster di atas ambang batas yang ditentukan. Hal ini juga dapat menghapus pod berdasarkan node taints atau node baru yang bergabung dengan cluster untuk memastikan pod berjalan di lingkungan komputasi optimal mereka. Perhatikan bahwa, descheduler tidak menjadwalkan penggantian pod yang diusir tetapi bergantung pada penjadwal default untuk itu.

Konsolidasi Karpenter

Karpenter mengambil pendekatan “tanpa kelompok” untuk manajemen simpul. Pendekatan ini lebih fleksibel untuk jenis beban kerja yang berbeda dan membutuhkan konfigurasi awal yang lebih sedikit untuk administrator klaster. Alih-alih menentukan grup sebelumnya dan menskalakan setiap grup sesuai kebutuhan beban kerja, Karpenter menggunakan penyedia dan templat simpul untuk menentukan secara luas jenis instance apa yang dapat dibuat dan EC2 pengaturan tentang instance saat dibuat.

Bin packing adalah praktik memanfaatkan lebih banyak sumber daya instans dengan mengemas lebih banyak beban kerja ke instance yang lebih sedikit dan berukuran optimal. Meskipun ini membantu mengurangi biaya komputasi Anda dengan hanya menyediakan sumber daya yang digunakan beban kerja Anda, ini memiliki trade-off. Diperlukan waktu lebih lama untuk memulai beban kerja baru karena kapasitas harus ditambahkan ke cluster, terutama selama peristiwa penskalaan besar. Pertimbangkan keseimbangan antara optimalisasi biaya, kinerja, dan ketersediaan saat menyiapkan pengepakan tempat sampah.

Karpenter dapat terus memantau dan binpack untuk meningkatkan pemanfaatan sumber daya instans dan menurunkan biaya komputasi Anda. Karpenter juga dapat memilih node pekerja yang lebih hemat biaya untuk beban kerja Anda. Ini dapat dicapai dengan mengaktifkan flag “konsolidasi” ke true di penyedia (contoh cuplikan kode di bawah). Contoh di bawah ini menunjukkan contoh penyedia yang memungkinkan konsolidasi. Pada saat menulis panduan ini, Karpenter tidak akan mengganti instance Spot yang sedang berjalan dengan instans Spot yang lebih murah. Untuk detail lebih lanjut tentang konsolidasi Karpenter, lihat blog ini.

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
  name: enable-binpacking
spec:
  consolidation:
    enabled: true
```

Untuk beban kerja yang mungkin tidak dapat diinterupsi, misalnya pekerjaan batch yang berjalan lama tanpa checkpointing, pertimbangkan untuk membuat anotasi pod dengan anotasi. `do-not-evict` Dengan memilih pod keluar dari penggusuran, Anda memberi tahu Karpenter bahwa itu tidak boleh secara sukarela menghapus node yang berisi pod ini. Namun, jika sebuah `do-not-evict` pod ditambahkan ke node saat node terkuras, pod yang tersisa akan tetap mengusir, tetapi pod itu akan memblokir penghentian sampai dihapus. Dalam kedua kasus tersebut, node akan ditutup untuk mencegah pekerjaan tambahan dijadwalkan pada node. Di bawah ini adalah contoh yang menunjukkan cara mengatur anotasi:

```
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
```

Hapus node yang kurang dimanfaatkan dengan menyesuaikan parameter Cluster Autoscaler

Pemanfaatan node didefinisikan sebagai jumlah sumber daya yang diminta dibagi dengan kapasitas. Secara default `scale-down-utilization-threshold` diatur ke 50%. Parameter ini dapat digunakan bersama dengan dan`scale-down-unneeded-time`, yang menentukan berapa lama node tidak diperlukan sebelum memenuhi syarat untuk menurunkan skala - defaultnya adalah 10 menit. Pod yang masih berjalan pada node yang diperkecil akan dijadwalkan pada node lain oleh kube-scheduler. Menyesuaikan pengaturan ini dapat membantu menghapus node yang kurang dimanfaatkan, tetapi penting Anda menguji nilai-nilai ini terlebih dahulu sehingga Anda tidak memaksa cluster untuk menurunkan skala sebelum waktunya.

Anda dapat mencegah terjadinya penurunan skala dengan memastikan bahwa pod yang mahal untuk diusir dilindungi oleh label yang dikenali oleh Cluster Autoscaler. Untuk melakukan ini, pastikan bahwa pod yang mahal untuk diusir memiliki anotasi. `cluster-autoscaler.kubernetes.io/safe-to-evict=false` Di bawah ini adalah contoh yaml untuk mengatur anotasi:

```
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
```

Tag node dengan Cluster Autoscaler dan Karpenter

Tag sumber daya AWS digunakan untuk mengatur sumber daya Anda, dan untuk melacak biaya AWS Anda pada tingkat terperinci. Mereka tidak berkorelasi langsung dengan label Kubernetes untuk pelacakan biaya. Disarankan untuk memulai dengan pelabelan sumber daya Kubernetes dan menggunakan alat seperti Kubecost untuk mendapatkan pelaporan biaya infrastruktur berdasarkan label Kubernetes pada pod, namespace, dll.

Node pekerja harus memiliki tag untuk menampilkan informasi penagihan di AWS Cost Explorer. Dengan Cluster Autoscaler, beri tag node pekerja Anda di dalam grup node terkelola menggunakan template peluncuran. Untuk grup node yang dikelola sendiri, beri tag instance Anda menggunakan grup penskalaan EC2 otomatis. Untuk instance yang disediakan oleh Karpenter, beri tag menggunakan spec.tags di template node.

Cluster multi-penyewa

Saat mengerjakan cluster yang dibagikan oleh tim yang berbeda, Anda mungkin tidak memiliki visibilitas ke beban kerja lain yang berjalan pada node yang sama. Sementara permintaan sumber daya dapat membantu mengisolasi beberapa masalah “tetangga yang bising”, seperti berbagi CPU, mereka mungkin tidak mengisolasi semua batas sumber daya seperti kelelahan disk. I/O Tidak setiap sumber daya habis pakai oleh beban kerja dapat diisolasi atau dibatasi. Beban kerja yang mengkonsumsi sumber daya bersama pada tingkat yang lebih tinggi daripada beban kerja lainnya harus diisolasi melalui noda dan toleransi node. Teknik canggih lainnya untuk beban kerja semacam itu adalah penyematan CPU yang memastikan CPU eksklusif alih-alih CPU bersama untuk wadah.

Mengisolasi beban kerja pada tingkat node bisa lebih mahal, tetapi dimungkinkan untuk menjadwalkan BestEffortpekerjaan atau memanfaatkan penghematan tambahan dengan menggunakan Instans Cadangan, prosesor Graviton, atau Spot.

Cluster bersama mungkin juga memiliki batasan sumber daya tingkat klaster seperti kehabisan IP, batas layanan Kubernetes, atau permintaan penskalaan API. Anda harus meninjau panduan praktik terbaik skalabilitas untuk memastikan klaster Anda menghindari batasan ini.

Anda dapat mengisolasi sumber daya di namespace atau tingkat penyedia Karpenter. Resource Quotas menyediakan cara untuk menetapkan batasan berapa banyak beban kerja sumber daya dalam namespace yang dapat dikonsumsi. Ini bisa menjadi rel penjaga awal yang baik tetapi harus terus dievaluasi untuk memastikan itu tidak secara artifitis membatasi beban kerja dari penskalaan.

Penyedia Karpenter dapat menetapkan batasan pada beberapa sumber daya habis pakai dalam sebuah cluster (misalnya CPU, GPU), tetapi Anda perlu mengonfigurasi aplikasi penyewa untuk menggunakan penyedia yang sesuai. Ini dapat mencegah satu penyedia membuat terlalu banyak node dalam sebuah cluster, tetapi harus terus dievaluasi untuk memastikan batas tidak disetel terlalu rendah dan pada gilirannya, mencegah beban kerja dari penskalaan.

Penskalaan Otomatis Terjadwal

Anda mungkin perlu menurunkan cluster Anda pada akhir pekan dan jam kerja. Ini sangat relevan untuk cluster pengujian dan non-produksi di mana Anda ingin menurunkan skala ke nol ketika mereka tidak digunakan. Solusi seperti cluster-turndown dapat menurunkan replika menjadi nol berdasarkan jadwal cron. Anda juga dapat mencapai ini dengan Karpenter, yang diuraikan dalam blog AWS berikut.

Optimalkan jenis kapasitas komputasi

Setelah mengoptimalkan total kapasitas komputasi di cluster Anda dan menggunakan kemasan bin, Anda harus melihat jenis komputasi apa yang telah Anda sediakan di cluster Anda dan bagaimana Anda membayar sumber daya tersebut. AWS memiliki paket Tabungan Komputasi yang dapat mengurangi biaya komputasi Anda yang akan kami kategorikan ke dalam jenis kapasitas berikut:

- Spot
- Savings Plans
- Sesuai Permintaan
- Fargate

Setiap jenis kapasitas memiliki trade-off yang berbeda untuk overhead manajemen, ketersediaan, dan komitmen jangka panjang dan Anda perlu memutuskan mana yang tepat untuk lingkungan Anda. Tidak ada lingkungan yang harus bergantung pada satu jenis kapasitas dan Anda dapat mencampur beberapa jenis run dalam satu cluster untuk mengoptimalkan persyaratan dan biaya beban kerja tertentu.

Spot

Jenis kapasitas spot menyediakan EC2 contoh dari kapasitas cadangan di Availability Zone. Spot menawarkan diskon terbesar — hingga 90% — tetapi contoh tersebut dapat terganggu ketika dibutuhkan di tempat lain. Selain itu, mungkin tidak selalu ada kapasitas untuk menyediakan instans Spot baru dan instans Spot yang ada dapat direklamasi dengan pemberitahuan interupsi 2 menit.

Jika aplikasi Anda memiliki proses startup atau shutdown yang lama, instans Spot mungkin bukan pilihan terbaik.

Komputasi spot harus menggunakan berbagai jenis instans untuk mengurangi kemungkinan tidak memiliki kapasitas spot yang tersedia. Interupsi instance perlu ditangani untuk mematikan node dengan aman. Node yang disediakan dengan Karpenter atau bagian dari Grup Node Terkelola secara otomatis mendukung notifikasi interupsi instans. Jika Anda menggunakan node yang dikelola sendiri, Anda harus menjalankan penangan terminasi node secara terpisah untuk mematikan instance spot dengan anggun.

Dimungkinkan untuk menyeimbangkan instans spot dan on-demand dalam satu cluster. Dengan Karpenter Anda dapat membuat penyedia tertimbang untuk mencapai keseimbangan berbagai jenis kapasitas. Dengan Cluster Autoscaler Anda dapat membuat grup node campuran dengan instans spot dan on-demand atau cadangan.

Berikut adalah contoh penggunaan Karpenter untuk memprioritaskan instans Spot sebelum instans On-Demand. Saat membuat penyedia, Anda dapat menentukan Spot, Sesuai Permintaan, atau keduanya (seperti yang ditunjukkan di bawah). Ketika Anda menentukan keduanya, dan jika pod tidak secara eksplisit menentukan apakah perlu menggunakan Spot atau On-Demand, maka Karpenter memprioritaskan Spot saat menyediakan node dengan strategi alokasi. 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, Instans Cadangan, dan AWS EDP

Anda dapat mengurangi pengeluaran komputasi Anda dengan menggunakan rencana tabungan komputasi. Paket tabungan menawarkan pengurangan harga untuk komitmen penggunaan komputasi 1 atau 3 tahun. Penggunaan dapat berlaku untuk EC2 instance di kluster EKS tetapi juga berlaku untuk penggunaan komputasi apa pun seperti Lambda dan Fargate. Dengan rencana tabungan Anda dapat mengurangi biaya dan masih memilih jenis EC2 instans apa pun selama periode komitmen Anda.

Paket penghematan komputasi dapat mengurangi EC2 biaya Anda hingga 66% tanpa memerlukan komitmen pada jenis instans, keluarga, atau wilayah yang ingin Anda gunakan. Tabungan secara otomatis diterapkan ke instans saat Anda menggunakannya.

EC2 Instance Savings Plans menyediakan penghematan hingga 72% pada komputasi dengan komitmen penggunaan di wilayah dan EC2 keluarga tertentu, misalnya instans dari keluarga C. Anda dapat mengalihkan penggunaan ke AZ mana pun di dalam wilayah, menggunakan generasi keluarga instans apa pun, misalnya c5 atau c6, dan menggunakan ukuran instance apa pun dalam keluarga. Diskon akan secara otomatis diterapkan untuk setiap contoh di akun Anda yang sesuai dengan kriteria paket tabungan.

Instans Cadangan mirip dengan EC2 Instans Savings Plan tetapi juga menjamin kapasitas di Availability Zone atau Region dan mengurangi biaya—hingga 72% — dibandingkan instans sesuai permintaan. Setelah Anda menghitung berapa banyak kapasitas cadangan yang Anda perlukan, Anda dapat memilih berapa lama Anda ingin memesannya (1 tahun atau 3 tahun). Diskon akan diterapkan secara otomatis saat Anda menjalankan EC2 instans tersebut di akun Anda.

Pelanggan juga memiliki opsi untuk mendaftar dalam Perjanjian Perusahaan dengan AWS. Perjanjian Perusahaan memberi pelanggan pilihan untuk menyesuaikan perjanjian yang paling sesuai dengan kebutuhan mereka. Pelanggan dapat menikmati diskon harga berdasarkan AWS EDP (Enterprise Discount Program). Untuk informasi tambahan tentang Perjanjian Perusahaan, silakan hubungi perwakilan penjualan AWS Anda.

Sesuai Permintaan

EC2 Instans On-Demand memiliki manfaat ketersediaan tanpa gangguan — dibandingkan dengan spot — dan tidak ada komitmen jangka panjang — dibandingkan dengan rencana tabungan. Jika Anda ingin mengurangi biaya dalam klaster, Anda harus mengurangi penggunaan EC2 instans sesuai permintaan.

Setelah mengoptimalkan persyaratan beban kerja Anda, Anda harus dapat menghitung kapasitas minimum dan maksimum untuk cluster Anda. Jumlah ini dapat berubah dari waktu ke waktu tetapi jarang turun. Pertimbangkan untuk menggunakan Savings Plan untuk segala sesuatu di bawah minimum, dan temukan kapasitas yang tidak akan memengaruhi ketersediaan aplikasi Anda. Hal lain yang mungkin tidak terus digunakan atau diperlukan untuk ketersediaan dapat digunakan sesuai permintaan.

Seperti disebutkan di bagian ini, cara terbaik untuk mengurangi penggunaan Anda adalah dengan mengkonsumsi lebih sedikit sumber daya dan memanfaatkan sumber daya yang Anda berikan semaksimal mungkin. Dengan Cluster Autoscaler Anda dapat menghapus node yang kurang

dimanfaatkan dengan pengaturan. `scale-down-utilization-threshold` Dengan Karpenter disarankan untuk mengaktifkan konsolidasi.

Untuk mengidentifikasi jenis EC2 instance secara manual yang dapat digunakan dengan beban kerja Anda, Anda harus menggunakan ec2-instance-selector yang dapat menampilkan instance yang tersedia di setiap wilayah serta instance yang kompatibel dengan EKS. Contoh penggunaan untuk instance dengan arsitektur proses x86, memori 4 Gb, 2 v CPUs dan tersedia di wilayah 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
```

Untuk lingkungan non-produksi, Anda dapat secara otomatis memiliki klaster yang diperkecil selama jam yang tidak digunakan seperti malam dan akhir pekan. Proyek kubecost cluster-turndown adalah contoh dari controller yang dapat secara otomatis menurunkan skala klaster Anda berdasarkan jadwal yang ditetapkan.

Fargate menghitung

Fargate compute adalah opsi komputasi yang dikelola sepenuhnya untuk kluster EKS. Ini menyediakan isolasi pod dengan menjadwalkan satu pod per node dalam cluster Kubernetes. Ini memungkinkan Anda untuk mengukur node komputasi Anda ke persyaratan CPU dan RAM dari beban kerja Anda untuk mengontrol penggunaan beban kerja secara ketat dalam sebuah cluster.

Fargate dapat menskalakan beban kerja sekecil 0,25 vCPU dengan memori 0,5 GB dan sebesar 16 vCPU dengan memori 120 GB. Ada batasan berapa banyak variasi ukuran pod yang tersedia dan Anda perlu memahami bagaimana beban kerja Anda paling sesuai dengan konfigurasi Fargate. Misalnya, jika beban kerja Anda membutuhkan 1 vCPU dengan memori 0,5 GB, pod Fargate terkecil adalah 1 vCPU dengan memori 2 GB.

Sementara Fargate memiliki banyak manfaat seperti tidak ada EC2 instance atau manajemen sistem operasi, mungkin memerlukan kapasitas komputasi yang lebih besar daripada EC2 instance

tradisional karena fakta bahwa setiap pod yang di-deploy diisolasi sebagai node terpisah di cluster. Ini membutuhkan lebih banyak duplikasi untuk hal-hal seperti Kubelet, agen logging, dan apa pun yang biasanya DaemonSets Anda terapkan ke node. DaemonSets tidak didukung di Fargate dan mereka perlu diubah menjadi pod “`sidecars"` dan dijalankan di samping aplikasi.

Fargate tidak dapat mengambil manfaat dari pengepakan bin atau CPU melalui penyediaan karena batas untuk beban kerja adalah simpul yang tidak dapat dibobol atau dibagikan di antara beban kerja. Fargate akan menghemat waktu manajemen EC2 instans yang memiliki biaya, tetapi biaya CPU dan memori mungkin lebih mahal daripada jenis EC2 kapasitas lainnya. Pod Fargate dapat memanfaatkan rencana penghematan komputasi untuk mengurangi biaya sesuai permintaan.

Optimalkan Penggunaan Komputasi

Cara lain untuk menghemat uang pada infrastruktur komputasi Anda adalah dengan menggunakan komputasi yang lebih efisien untuk beban kerja. Ini bisa berasal dari komputasi tujuan umum yang lebih berkinerja seperti prosesor Graviton yang hingga 20% lebih murah dan 60% lebih hemat energi daripada x86 — atau akselerator khusus beban kerja seperti dan. GPUs FPGAs Anda perlu membangun kontainer yang dapat berjalan pada arsitektur arm dan menyiapkan node dengan akselerator yang tepat untuk beban kerja Anda.

EKS memiliki kemampuan untuk menjalankan cluster dengan arsitektur campuran (misalnya amd64 dan arm64) dan jika kontainer Anda dikompilasi untuk beberapa arsitektur, Anda dapat memanfaatkan prosesor Graviton dengan Karpenter dengan mengizinkan kedua arsitektur di penyedia Anda. Namun, untuk menjaga kinerja yang konsisten, Anda disarankan untuk menyimpan setiap beban kerja pada arsitektur komputasi tunggal dan hanya menggunakan arsitektur yang berbeda jika tidak ada kapasitas tambahan yang tersedia.

Penyedia dapat dikonfigurasi dengan beberapa arsitektur dan beban kerja juga dapat meminta arsitektur tertentu dalam spesifikasi beban kerja mereka.

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
  name: default
spec:
  requirements:
  - key: "kubernetes.io/arch"
    operator: In
    values: ["arm64", "amd64"]
```

Dengan Cluster Autoscaler Anda perlu membuat grup node untuk instance Graviton dan mengatur toleransi node pada beban kerja Anda untuk memanfaatkan kapasitas baru.

GPUs dan FPGAs dapat sangat meningkatkan kinerja untuk beban kerja Anda, tetapi beban kerja perlu dioptimalkan untuk menggunakan akselerator. Banyak jenis beban kerja untuk pembelajaran mesin dan kecerdasan buatan dapat digunakan GPUs untuk komputasi dan instance dapat ditambahkan ke cluster dan dipasang ke beban kerja menggunakan permintaan sumber daya.

```
spec:
  template:
    spec:
    - containers:
      ...
      resources:
          limits:
            nvidia.com/gpu: "1"
```

Beberapa perangkat keras GPU dapat dibagi di beberapa beban kerja sehingga satu GPU dapat disediakan dan digunakan. Untuk melihat cara mengonfigurasi berbagi GPU beban kerja, lihat plugin perangkat GPU virtual untuk informasi selengkapnya. Anda juga dapat merujuk ke blog berikut:

- Berbagi GPU di Amazon EKS dengan pengiris waktu NVIDIA dan instans yang dipercepat EC2
- Memaksimalkan pemanfaatan GPU dengan NVIDIA Multi-Instance GPU (MIG) di Amazon EKS: Menjalankan lebih banyak pod per GPU untuk meningkatkan kinerja

Optimalisasi Biaya - Jaringan

Sistem arsitektur untuk ketersediaan tinggi (HA) adalah praktik terbaik untuk mencapai ketahanan dan toleransi kesalahan. Dalam praktiknya, ini berarti menyebarkan beban kerja Anda dan infrastruktur yang mendasarinya di beberapa Availability Zone (AZs) di Wilayah AWS tertentu. Memastikan karakteristik ini tersedia untuk lingkungan Amazon EKS Anda akan meningkatkan keandalan keseluruhan sistem Anda. Sehubungan dengan ini, lingkungan EKS Anda kemungkinan juga akan terdiri dari berbagai konstruksi (yaitu VPCs), komponen (yaitu ELBs), dan integrasi (yaitu ECR dan pendaftar kontainer lainnya).

Kombinasi sistem yang sangat tersedia dan komponen spesifik kasus penggunaan lainnya dapat memainkan peran penting dalam bagaimana data ditransfer dan diproses. Ini pada gilirannya akan berdampak pada biaya yang dikeluarkan karena transfer dan pemrosesan data.

Praktik yang dirinci di bawah ini akan membantu Anda merancang dan mengoptimalkan lingkungan EKS Anda untuk mencapai efektivitas biaya untuk berbagai domain dan kasus penggunaan.

Komunikasi Pod ke Pod

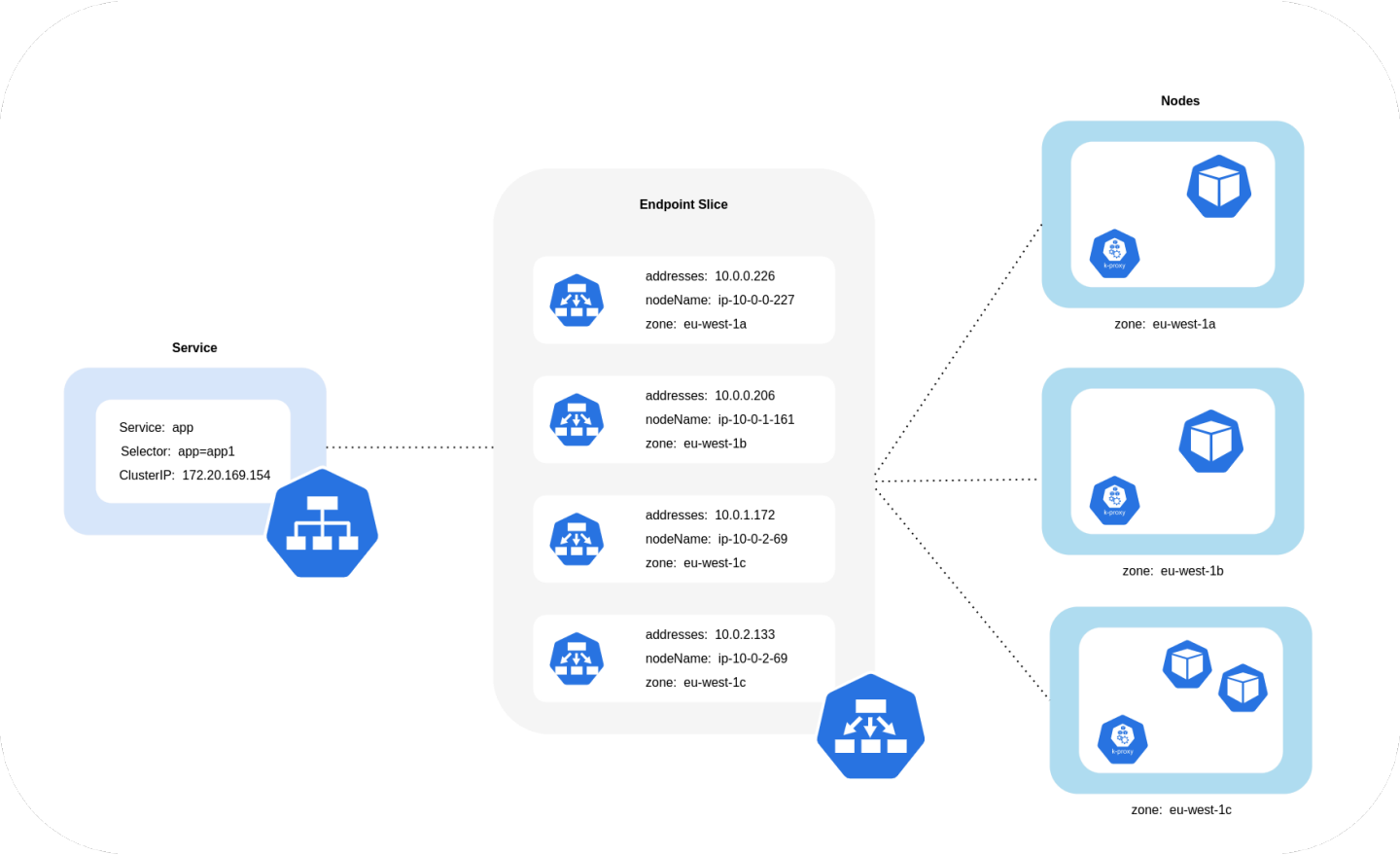
Bergantung pada penyiapan Anda, komunikasi jaringan dan transfer data antar Pod dapat berdampak signifikan terhadap keseluruhan biaya menjalankan beban kerja Amazon EKS. Bagian ini akan mencakup berbagai konsep dan pendekatan untuk mengurangi biaya yang terkait dengan komunikasi antar-pod, sambil mempertimbangkan arsitektur, kinerja aplikasi, dan ketahanan yang sangat tersedia (HA).

Membatasi Lalu Lintas ke Availability Zone

Proyek Kubernetes sejak awal mulai mengembangkan konstruksi sadar topologi termasuk label seperti kubernetes. io/hostname, topology.kubernetes.io/region, and topology.kubernetes.io/ zoneditugaskan ke node untuk mengaktifkan fitur seperti distribusi beban kerja di seluruh domain kegagalan dan penyedia volume sadar topologi. Setelah lulus di Kubernetes 1.17, label juga dimanfaatkan untuk mengaktifkan kemampuan routing yang sadar topologi untuk komunikasi Pod ke Pod.

Di bawah ini adalah beberapa strategi tentang cara mengontrol jumlah lalu lintas lintas AZ antar Pod di klaster EKS Anda untuk mengurangi biaya dan meminimalkan latensi.

Jika Anda ingin visibilitas granular ke dalam jumlah lalu lintas lintas AZ antar Pod di klaster Anda (seperti jumlah data yang ditransfer dalam byte), lihat posting ini.



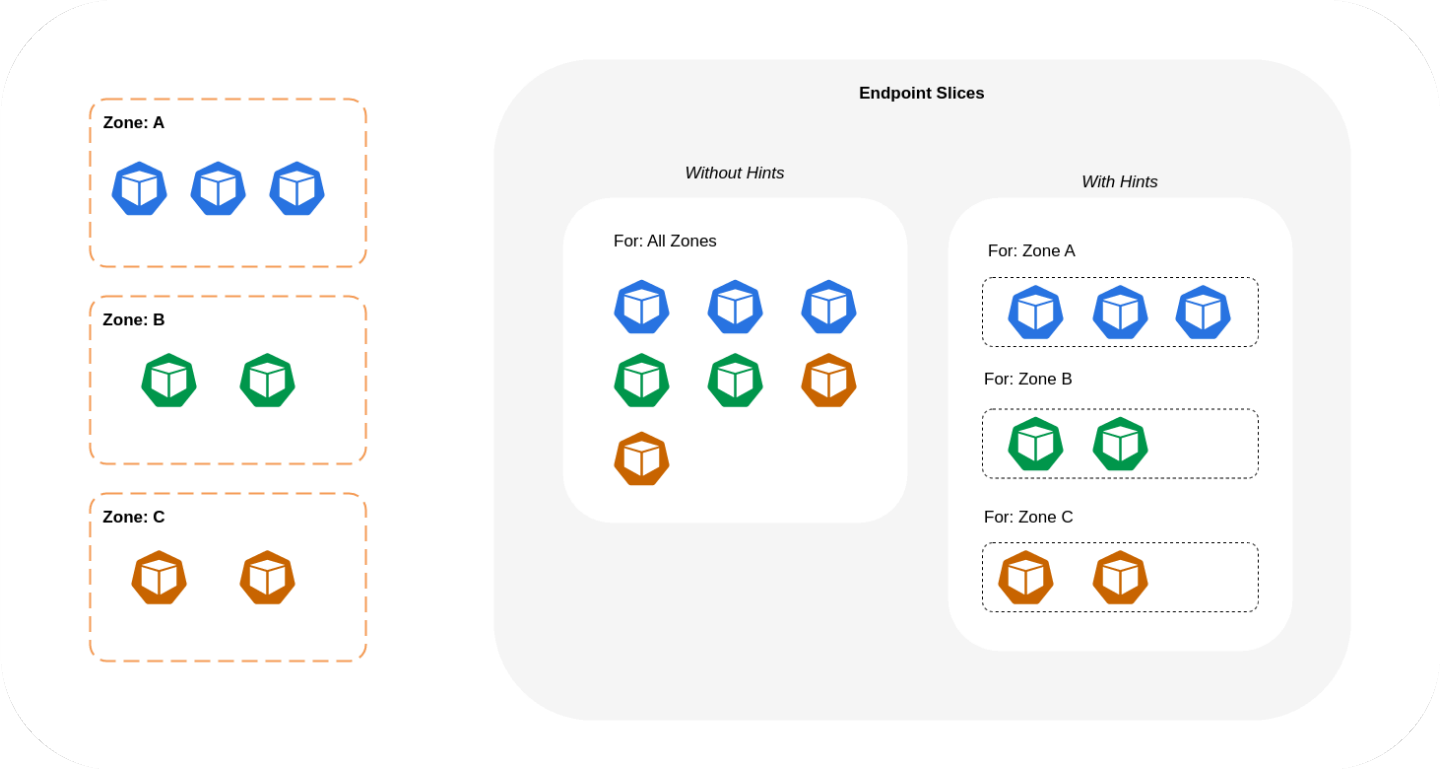
Seperti yang digambarkan diagram di atas, Layanan adalah lapisan abstraksi jaringan stabil yang menerima lalu lintas yang ditujukan untuk Pod Anda. Ketika Layanan dibuat, beberapa EndpointSlices dibuat. Masing-masing EndpointSlice memiliki daftar titik akhir yang berisi subset alamat Pod bersama dengan node yang mereka jalankan dan informasi topologi tambahan. Saat menggunakan Amazon VPC CNI, kube-proxy, daemonset yang berjalan di setiap node, mempertahankan aturan jaringan untuk mengaktifkan komunikasi Pod dan penemuan Layanan (berbasis EBPF CNIs alternatif mungkin tidak menggunakan kube-proxy tetapi memberikan perilaku yang setara). Ini memenuhi peran perutean internal, tetapi melakukannya berdasarkan apa yang dikonsumsi dari yang dibuat. EndpointSlices

Di EKS, kube-proxy terutama menggunakan aturan NAT iptables (atau IPVS, NFTablessebagai alternatif) untuk distribusi lalu lintas di semua pod di cluster, terlepas dari penempatan node atau AZ mereka. Distribusi default ini dapat menyebabkan perutean lalu lintas lintas AZ, berpotensi menyebabkan peningkatan latensi untuk aplikasi sensitif dan biaya transfer data antar-AZ dalam penerapan besar.

Menggunakan Topology Aware Routing (sebelumnya dikenal sebagai Topology Aware Hints)

Ketika routing sadar topologi diaktifkan dan diimplementasikan pada Layanan Kubernetes, EndpointSlice pengontrol akan mengalokasikan titik akhir secara proporsional ke zona berbeda yang tersebar di kluster Anda. Untuk masing-masing titik akhir tersebut, EndpointSlice pengontrol juga akan menetapkan petunjuk untuk zona tersebut. Petunjuk menjelaskan zona mana titik akhir harus melayani lalu lintas. `kube-proxy`kemudian akan merutekan lalu lintas dari zona ke titik akhir berdasarkan petunjuk yang diterapkan.

Diagram di bawah ini menunjukkan bagaimana EndpointSlices dengan petunjuk diatur sedemikian rupa sehingga `kube-proxy` dapat mengetahui tujuan apa yang harus mereka tuju berdasarkan titik asal zona mereka. Tanpa petunjuk, tidak ada alokasi atau organisasi seperti itu dan lalu lintas akan diproksi ke tujuan zona yang berbeda terlepas dari mana asalnya.



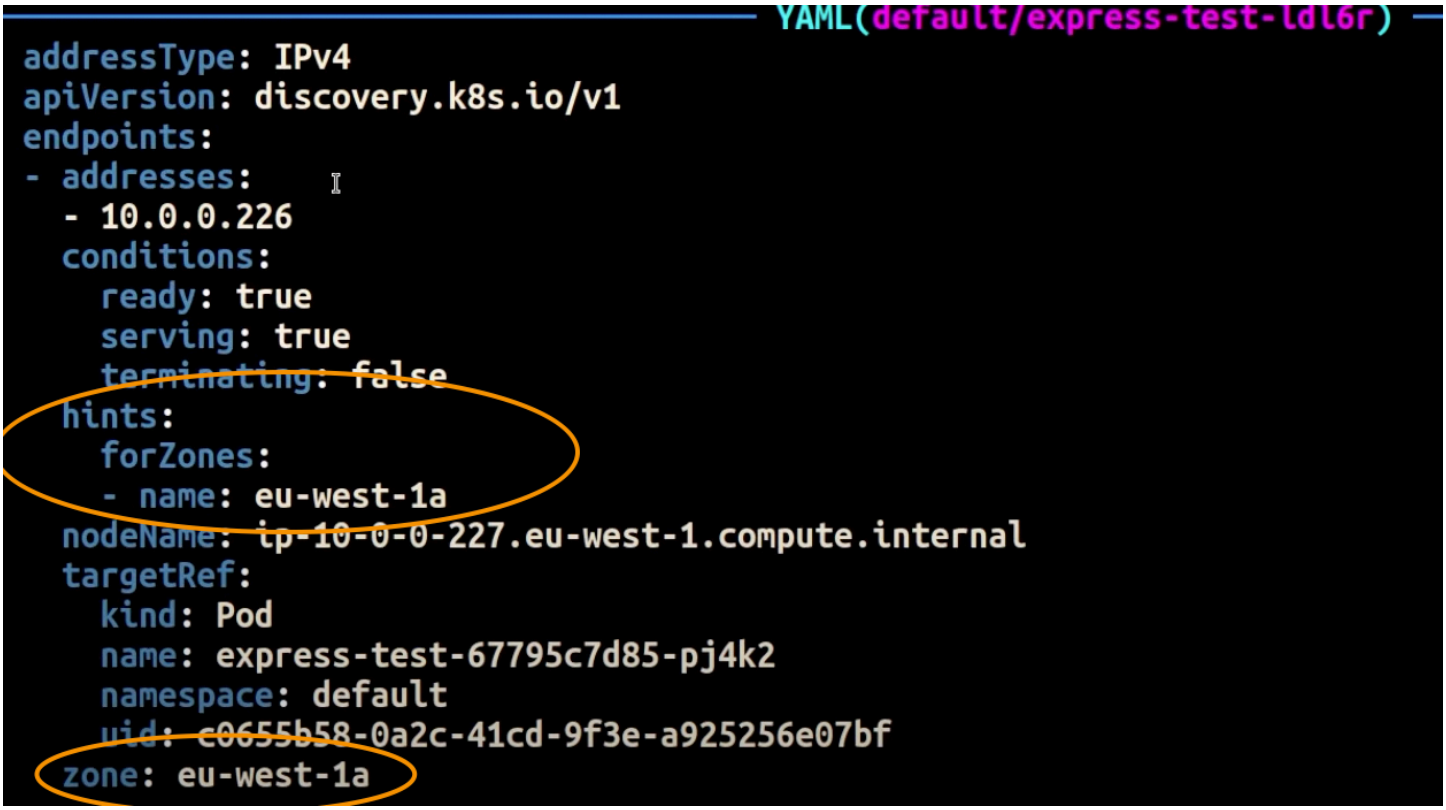
Dalam beberapa kasus, EndpointSlice pengontrol dapat menerapkan petunjuk untuk zona yang berbeda, yang berarti titik akhir dapat berakhir melayani lalu lintas yang berasal dari zona yang berbeda. Alasan untuk ini adalah untuk mencoba dan mempertahankan distribusi lalu lintas yang merata antara titik akhir di zona yang berbeda.

Di bawah ini adalah cuplikan kode tentang cara mengaktifkan perutean sadar topologi untuk Layanan.

```
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
```

Tangkapan layar di bawah ini menunjukkan hasil dari EndpointSlice controller yang telah berhasil menerapkan petunjuk ke titik akhir untuk replika Pod yang berjalan di AZ. `eu-west-1a`



> **Note**
>
> Penting untuk dicatat bahwa perutean sadar topologi masih dalam versi beta. Fitur ini bekerja lebih dapat diprediksi dengan beban kerja yang didistribusikan secara merata di seluruh

topologi cluster, karena pengontrol mengalokasikan titik akhir secara proporsional di seluruh zona tetapi dapat melewati penetapan petunjuk ketika sumber daya node di zona terlalu tidak seimbang untuk menghindari kelebihan beban yang berlebihan. Oleh karena itu, sangat disarankan untuk menggunakannya bersamaan dengan kendala penjadwalan yang meningkatkan ketersediaan aplikasi seperti kendala penyebaran topologi pod. Perhatikan bahwa petunjuk juga tidak dapat ditetapkan saat kapasitas berfluktuasi di seluruh zona, seperti saat menggunakan Instans Spot Amazon EC2, karena interupsi atau penggantian tidak terdeteksi secara real-time saat menghitung distribusi proporsional.

Menggunakan Distribusi Lalu Lintas

Diperkenalkan di Kubernetes 1.30 dan tersedia secara umum di 1.33, Traffic Distribution menawarkan alternatif yang lebih sederhana untuk Topology Aware Routing untuk preferensi lalu lintas zona yang sama. Sementara Topology Aware Routing mencoba menggunakan pendekatan cerdas untuk perutean lalu lintas untuk menghindari titik akhir yang berlebihan, hal itu menghasilkan perilaku yang tidak dapat diprediksi. Distribusi Lalu Lintas memprioritaskan prediktabilitas sebagai gantinya. PreferClose Opsi mengarahkan kube-proxy untuk membuat aturan yang merutekan lalu lintas ke titik akhir zona yang sama terlebih dahulu berdasarkan petunjuk zona yang ditetapkan oleh Controller. EndpointSlice Ketika tidak ada titik akhir zona yang sama yang tersedia, itu akan kembali mendistribusikan lalu lintas di seluruh titik akhir klaster mana pun untuk Layanan. Fitur ini dirancang untuk beban kerja yang menerima tradeoff dari pengoptimalan untuk kedekatan daripada upaya pemerataan beban yang diberikan oleh Topology Aware Routing.

Di bawah ini adalah cuplikan kode tentang cara mengaktifkan distribusi lalu lintas untuk Layanan.

```
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
```

Saat mengaktifkan Distribusi Lalu Lintas, tantangan umum muncul: titik akhir dalam satu AZ dapat menjadi kelebihan beban jika sebagian besar lalu lintas berasal dari zona yang sama. Kelebihan beban ini dapat menimbulkan masalah signifikan:

- Satu Horizontal Pod Autoscaler (HPA) yang mengelola penerapan Multi-AZ dapat merespons dengan menskalakan pod di berbagai jenis. AZs Namun, tindakan ini gagal untuk secara efektif mengatasi peningkatan beban di zona yang terkena dampak.
- Situasi ini pada gilirannya dapat menyebabkan inefisiensi sumber daya. Ketika autoscaler cluster seperti Karpenter mendeteksi penskalaan pod secara berbeda AZs, mereka dapat menyediakan node tambahan di tempat yang tidak terpengaruh, menghasilkan alokasi sumber daya yang tidak AZs perlu.

Untuk mengatasi tantangan ini:

- Buat penerapan terpisah per zona yang akan memiliki skala sendiri HPAs untuk diskalakan secara independen satu sama lain.
- Manfaatkan Batasan Penyebaran Topologi untuk memastikan distribusi beban kerja di seluruh klaster, yang membantu mencegah kelebihan titik akhir di zona lalu lintas tinggi.

Menggunakan Autoscaler: Menyediakan Node ke AZ Tertentu

Kami sangat menyarankan untuk menjalankan beban kerja Anda di lingkungan yang sangat tersedia di beberapa AZs tempat. Ini meningkatkan keandalan aplikasi Anda, terutama ketika ada insiden masalah dengan AZ. Jika Anda bersedia mengorbankan keandalan demi mengurangi biaya terkait jaringan mereka, Anda dapat membatasi node Anda ke satu AZ.

Untuk menjalankan semua Pod Anda di AZ yang sama, berikan node worker di AZ yang sama atau jadwalkan Pod pada node pekerja yang berjalan pada AZ yang sama. Untuk menyediakan node dalam satu AZ, tentukan grup node dengan subnet milik AZ yang sama dengan Cluster Autoscaler (CA). Untuk Karpenter, gunakan `topology.kubernetes.io/zone` dan tentukan AZ tempat Anda ingin membuat node pekerja. Misalnya, cuplikan penyedia Karpenter di bawah ini menyediakan node di us-west-2a AZ.

Karpenter

```
apiVersion: karpenter.sh/v1
```

```
kind: Provisioner
metadata:
name: single-az
spec:
  requirements:

* key: "topology.kubernetes.io/zone"`
operator: In
values: ["us-west-2a"]
```

Cluster Autoscaler (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
...
```

Menggunakan Pod Assignment dan Node Affinity

Atau, jika Anda memiliki node pekerja yang berjalan dalam beberapa AZs, setiap node akan memiliki label topology.kubernetes.io/zone dengan nilai AZ-nya (seperti us-west-2a atau us-west-2b). Anda dapat menggunakan `nodeSelector` atau `nodeAffinity` menjadwalkan Pod ke node dalam satu AZ. Misalnya, file manifes berikut akan menjadwalkan Pod di dalam sebuah node yang berjalan di 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
```

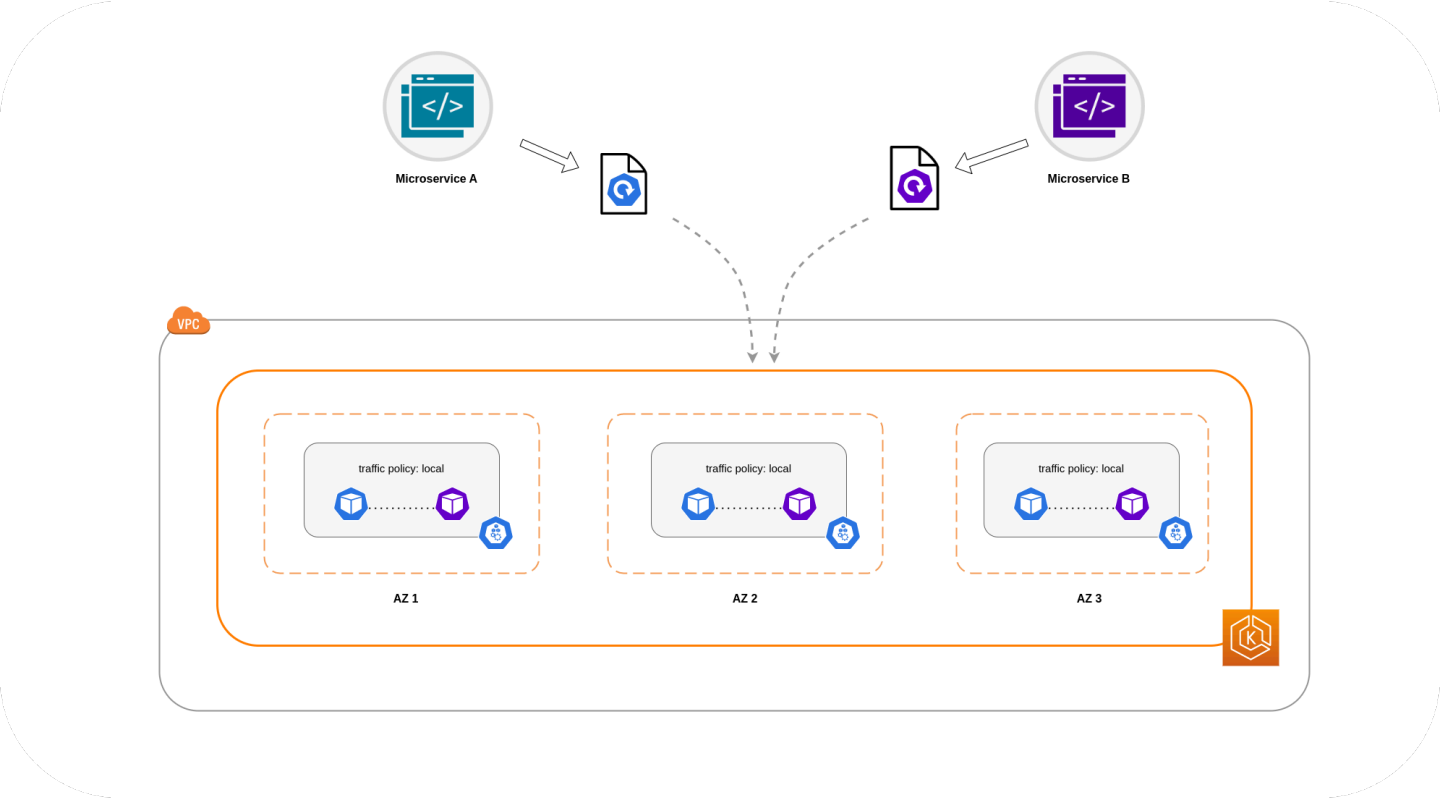
Membatasi Lalu Lintas ke Node

Ada kasus di mana membatasi lalu lintas di tingkat zona tidak cukup. Selain mengurangi biaya, Anda mungkin memiliki persyaratan tambahan untuk mengurangi latensi jaringan antara aplikasi tertentu yang sering melakukan komunikasi antar. Untuk mencapai kinerja jaringan yang optimal dan mengurangi biaya, Anda memerlukan cara untuk membatasi lalu lintas ke node tertentu. Misalnya, Microservice A harus selalu berbicara dengan Microservice B di Node 1, bahkan dalam pengaturan (HA) yang sangat tersedia. Memiliki Microservice A pada Node 1 berbicara dengan Microservice B pada Node 2 mungkin berdampak negatif pada kinerja yang diinginkan untuk aplikasi seperti ini, terutama jika Node 2 berada di AZ terpisah sama sekali.

Menggunakan Kebijakan Lalu Lintas Internal Layanan

Untuk membatasi lalu lintas jaringan Pod ke sebuah node, Anda dapat menggunakan kebijakan lalu lintas internal Layanan. Secara default, lalu lintas yang dikirim ke Layanan beban kerja akan didistribusikan secara acak di berbagai titik akhir yang dihasilkan. Jadi dalam arsitektur HA, itu berarti lalu lintas dari Microservice A dapat pergi ke replika Microservice B apa pun pada node tertentu di seluruh yang berbeda. AZs Namun, dengan kebijakan lalu lintas internal Layanan yang disetel ke`Local`, lalu lintas akan dibatasi ke titik akhir pada node tempat lalu lintas berasal. Kebijakan ini menentukan penggunaan eksklusif titik akhir node-lokal. Implikasinya, biaya terkait lalu lintas jaringan Anda untuk beban kerja itu akan lebih rendah daripada jika distribusinya luas. Selain itu, latensi akan lebih rendah, membuat aplikasi Anda lebih berkinerja.

> **Note**
>
> Penting untuk dicatat bahwa fitur ini tidak dapat digabungkan dengan perutean sadar topologi di Kubernetes.



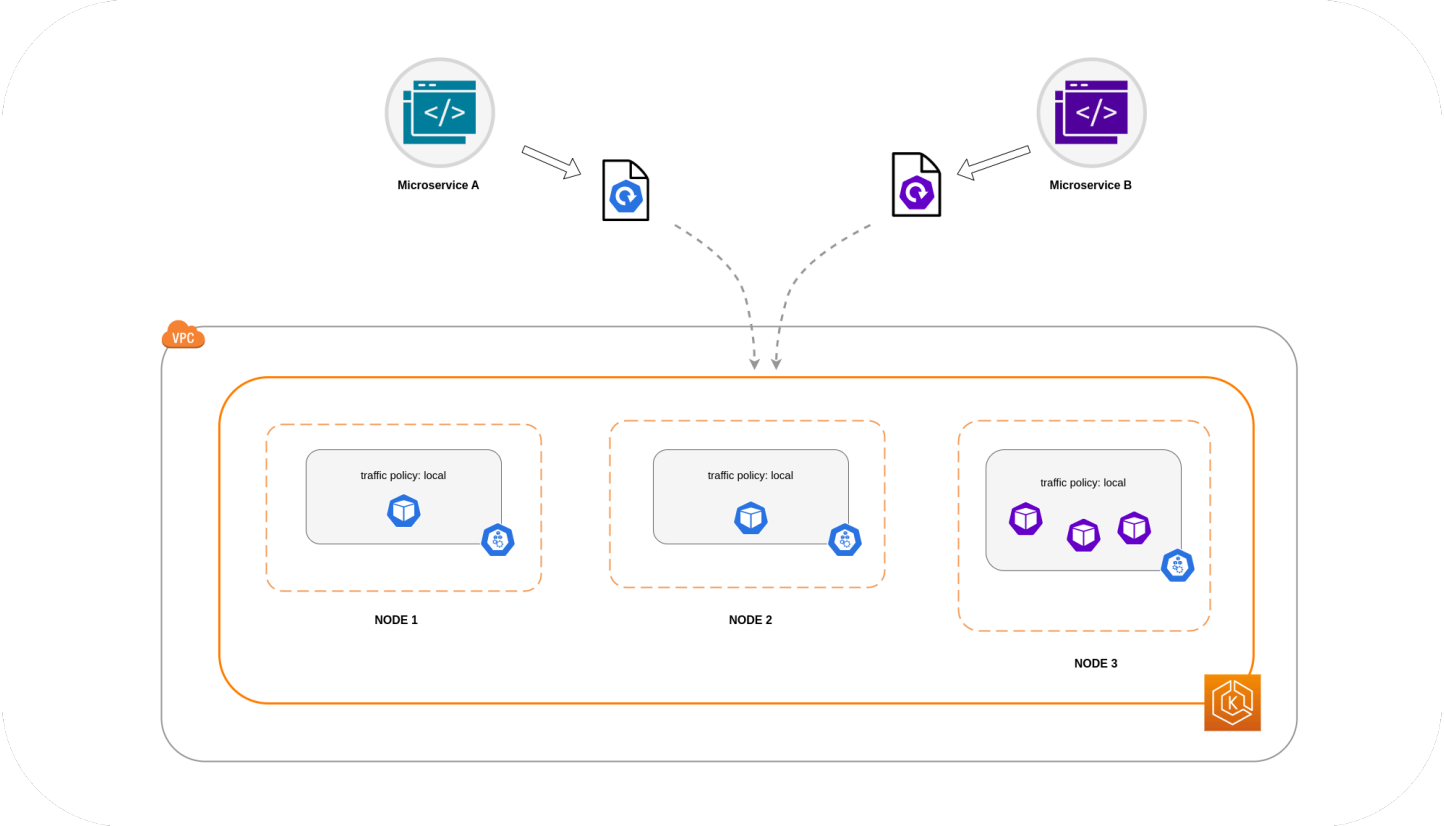
Di bawah ini adalah cuplikan kode tentang cara mengatur kebijakan lalu lintas internal untuk Layanan.

```
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
```

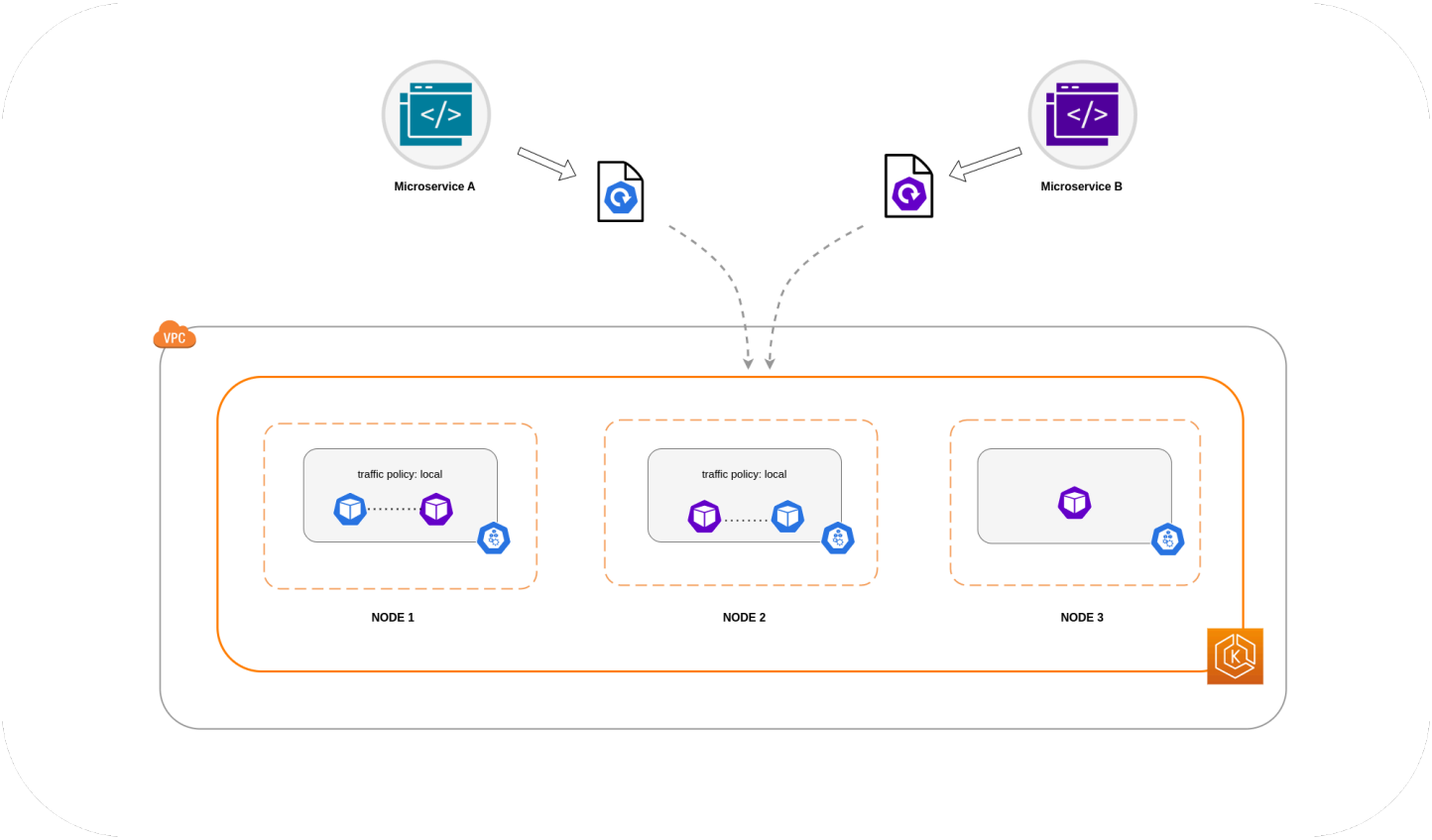
Untuk menghindari perilaku tak terduga dari aplikasi Anda karena penurunan lalu lintas, Anda harus mempertimbangkan pendekatan berikut:

- Jalankan replika yang cukup untuk setiap Pod yang berkomunikasi
- Memiliki penyebaran Pod yang relatif merata menggunakan kendala penyebaran topologi
- Memanfaatkan aturan pod-affinity untuk co-lokasi Pod yang berkomunikasi

Dalam contoh ini, Anda memiliki 2 replika Microservice A dan 3 replika Microservice B. Jika Microservice A memiliki replika yang tersebar antara Nodes 1 dan 2, dan Microservice B memiliki semua 3 replika pada Node 3, maka mereka tidak akan dapat berkomunikasi karena kebijakan lalu lintas internal. `Local` Ketika tidak ada titik akhir node-lokal yang tersedia, lalu lintas dijatuhkan.



Jika Microservice B memang memiliki 2 dari 3 replika pada Nodes 1 dan 2, maka akan ada komunikasi antara aplikasi peer. Tetapi Anda masih akan memiliki replika Microservice B yang terisolasi tanpa replika rekan untuk berkomunikasi.



> **Note**
>
> Dalam beberapa skenario, replika terisolasi seperti yang digambarkan dalam diagram di atas mungkin tidak menjadi perhatian jika masih melayani tujuan (seperti melayani permintaan dari lalu lintas masuk eksternal).

Menggunakan Kebijakan Lalu Lintas Internal Layanan dengan Kendala Penyebaran Topologi

Menggunakan kebijakan lalu lintas internal bersama dengan kendala penyebaran topologi dapat berguna untuk memastikan bahwa Anda memiliki jumlah replika yang tepat untuk mengkomunikasikan layanan mikro pada node yang berbeda.

```
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
```

Menggunakan Kebijakan Lalu Lintas Internal Layanan dengan Aturan Afinitas Pod

Pendekatan lain adalah dengan menggunakan aturan afinitas Pod saat menggunakan kebijakan lalu lintas internal Layanan. Dengan afinitas Pod, Anda dapat memengaruhi scheduler untuk menemukan Pod tertentu karena sering berkomunikasi. Dengan menerapkan batasan penjadwalan yang ketat (`requiredDuringSchedulingIgnoredDuringExecution`) pada Pod tertentu, ini akan memberi Anda hasil yang lebih baik untuk co-location Pod ketika Scheduler menempatkan Pod pada node.

```
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"
```

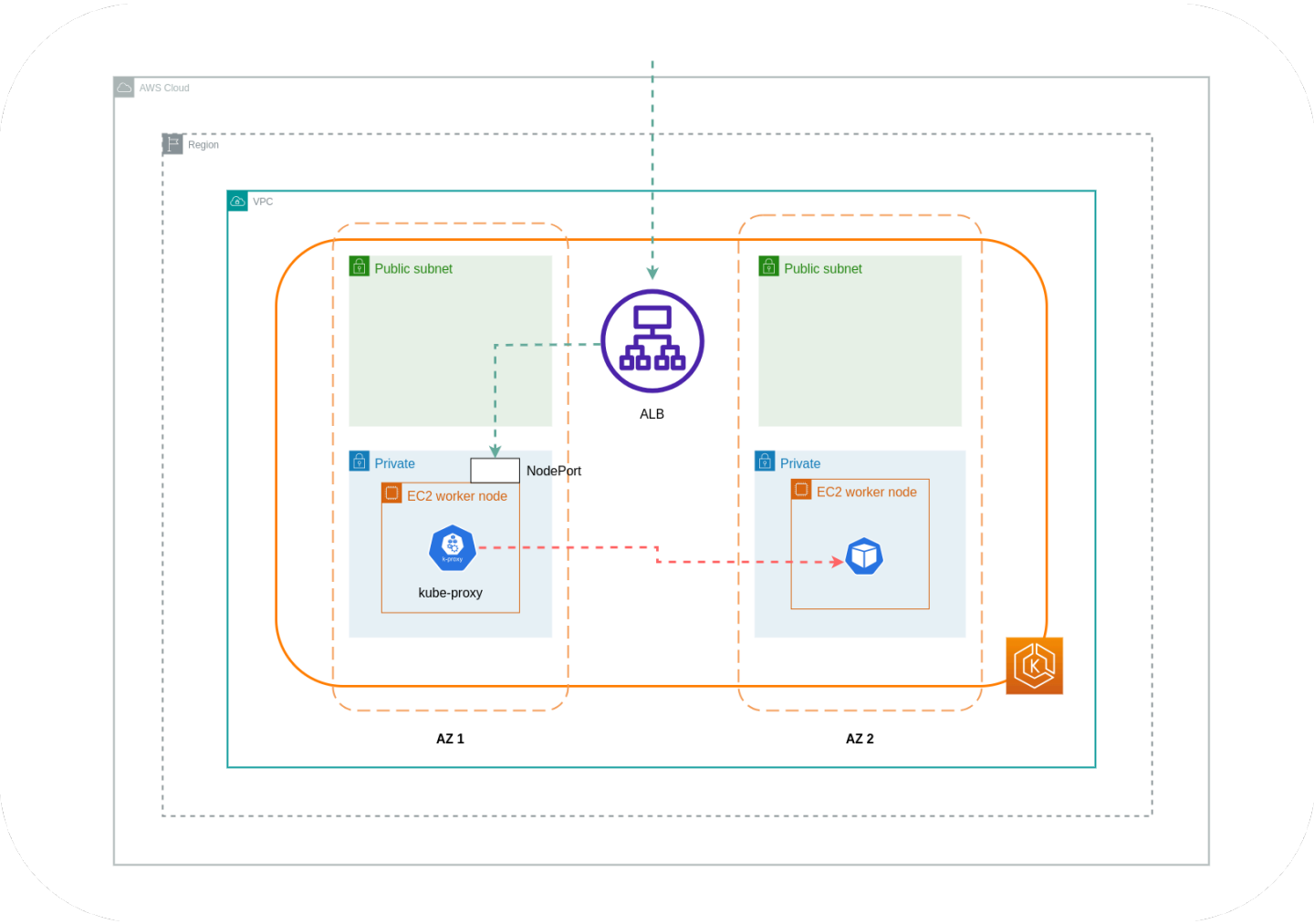
Load Balancer ke Komunikasi Pod

Beban kerja EKS biasanya digawangi oleh penyeimbang beban yang mendistribusikan lalu lintas ke Pod yang relevan di klaster EKS Anda. Arsitektur Anda dapat terdiri dari penyeimbang beban menghadap and/or eksternal internal. Bergantung pada arsitektur dan konfigurasi lalu lintas jaringan Anda, komunikasi antara load balancer dan Pod dapat memberikan kontribusi yang signifikan terhadap biaya transfer data.

Anda dapat menggunakan AWS Load Balancer Controller untuk mengelola pembuatan sumber daya ELB (ALB dan NLB) secara otomatis. Biaya transfer data yang Anda keluarkan dalam pengaturan tersebut akan tergantung pada jalur yang diambil oleh lalu lintas jaringan. AWS Load Balancer Controller mendukung dua mode lalu lintas jaringan, mode instans, dan mode ip.

Saat menggunakan mode instance, a NodePort akan dibuka pada setiap node di cluster EKS Anda. Penyeimbang beban kemudian akan mem-proxy lalu lintas secara merata di seluruh node. Jika sebuah node memiliki Pod tujuan yang berjalan di atasnya, maka tidak akan ada biaya transfer data yang dikeluarkan. Namun, jika Pod tujuan berada pada node terpisah dan di AZ yang berbeda dari yang NodePort menerima lalu lintas, maka akan ada tambahan network hop dari kube-proxy ke Pod tujuan. Dalam skenario seperti itu, akan ada biaya transfer data lintas-AZ. Karena distribusi lalu lintas yang merata di seluruh node, sangat mungkin bahwa akan ada biaya transfer data tambahan yang terkait dengan hop lalu lintas jaringan lintas zona dari kube-proxy ke Pod tujuan yang relevan.

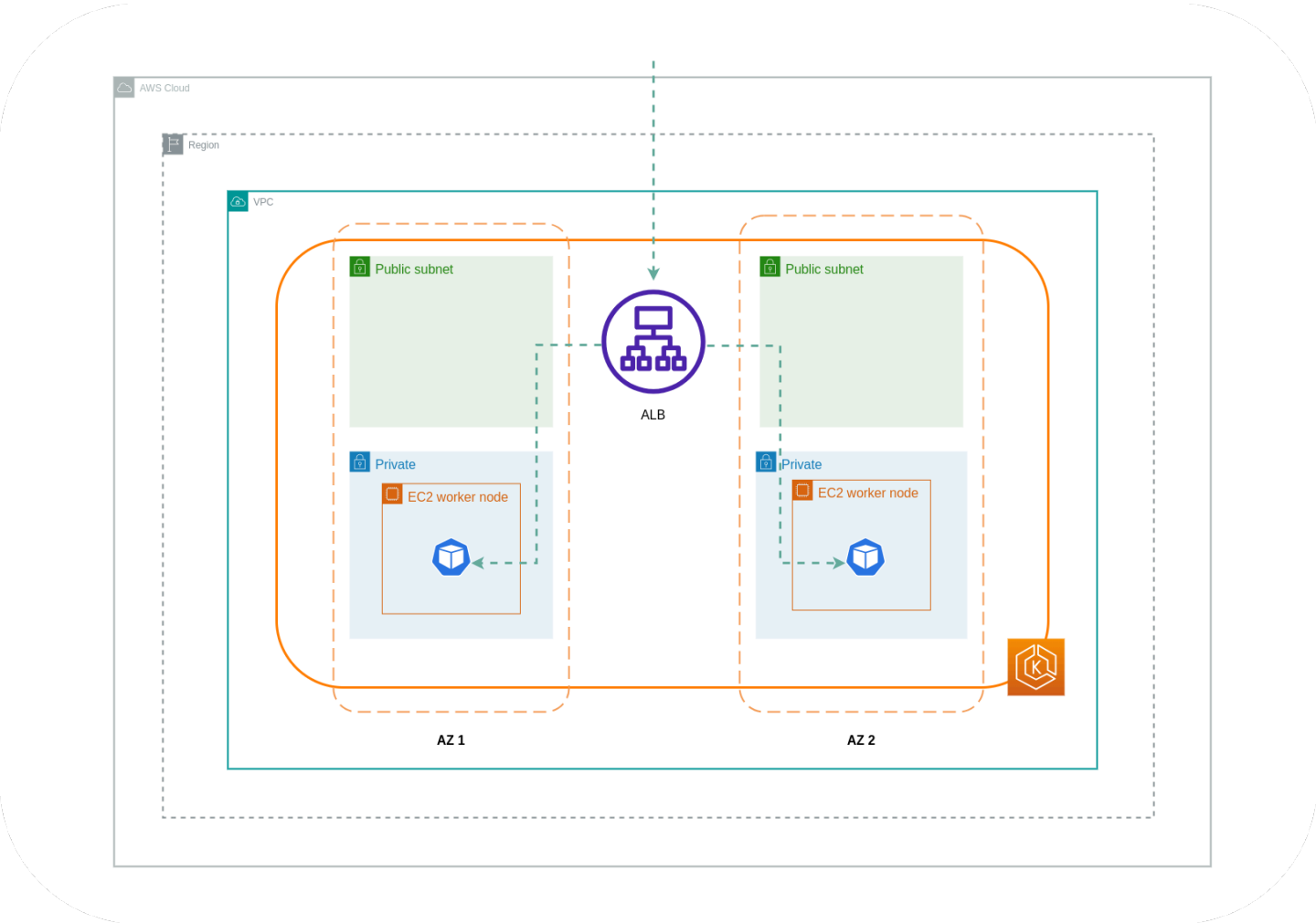
Diagram di bawah ini menggambarkan jalur jaringan untuk lalu lintas yang mengalir dari load balancer ke NodePort, dan selanjutnya dari Pod `kube-proxy` ke tujuan pada node terpisah di AZ yang berbeda. Ini adalah contoh pengaturan mode instance.



Saat menggunakan mode ip, lalu lintas jaringan diproksi dari load balancer langsung ke Pod tujuan. Akibatnya, tidak ada biaya transfer data yang terlibat dalam pendekatan ini.

> **Note**
>
> Disarankan agar Anda mengatur penyeimbang beban Anda ke mode lalu lintas ip untuk mengurangi biaya transfer data. Untuk pengaturan ini, penting juga untuk memastikan bahwa penyeimbang beban Anda diterapkan di semua subnet di VPC Anda.

Diagram di bawah ini menggambarkan jalur jaringan untuk lalu lintas yang mengalir dari load balancer ke Pod dalam mode ip jaringan.



Transfer Data dari Registri Kontainer

Amazon ECR

Transfer data ke registri pribadi Amazon ECR gratis. Transfer data dalam wilayah tidak dikenakan biaya, tetapi transfer data ke internet dan lintas wilayah akan dikenakan tarif Transfer Data Internet di kedua sisi transfer.

Anda harus menggunakan fitur replikasi gambar ECRs bawaan untuk mereplikasi gambar kontainer yang relevan ke wilayah yang sama dengan beban kerja Anda. Dengan cara ini replikasi akan diisi sekali, dan semua tarikan gambar wilayah (intra-wilayah) yang sama akan gratis.

Anda dapat mengurangi biaya transfer data yang terkait dengan pengambilan gambar dari ECR (transfer data keluar) dengan menggunakan Titik Akhir VPC Antarmuka untuk terhubung ke repositori ECR di wilayah. Pendekatan alternatif untuk menghubungkan ke titik akhir AWS publik ECR (melalui Gateway NAT dan Internet Gateway) akan menimbulkan biaya pemrosesan dan transfer data yang

lebih tinggi. Bagian selanjutnya akan mencakup pengurangan biaya transfer data antara beban kerja Anda dan AWS Services secara lebih rinci.

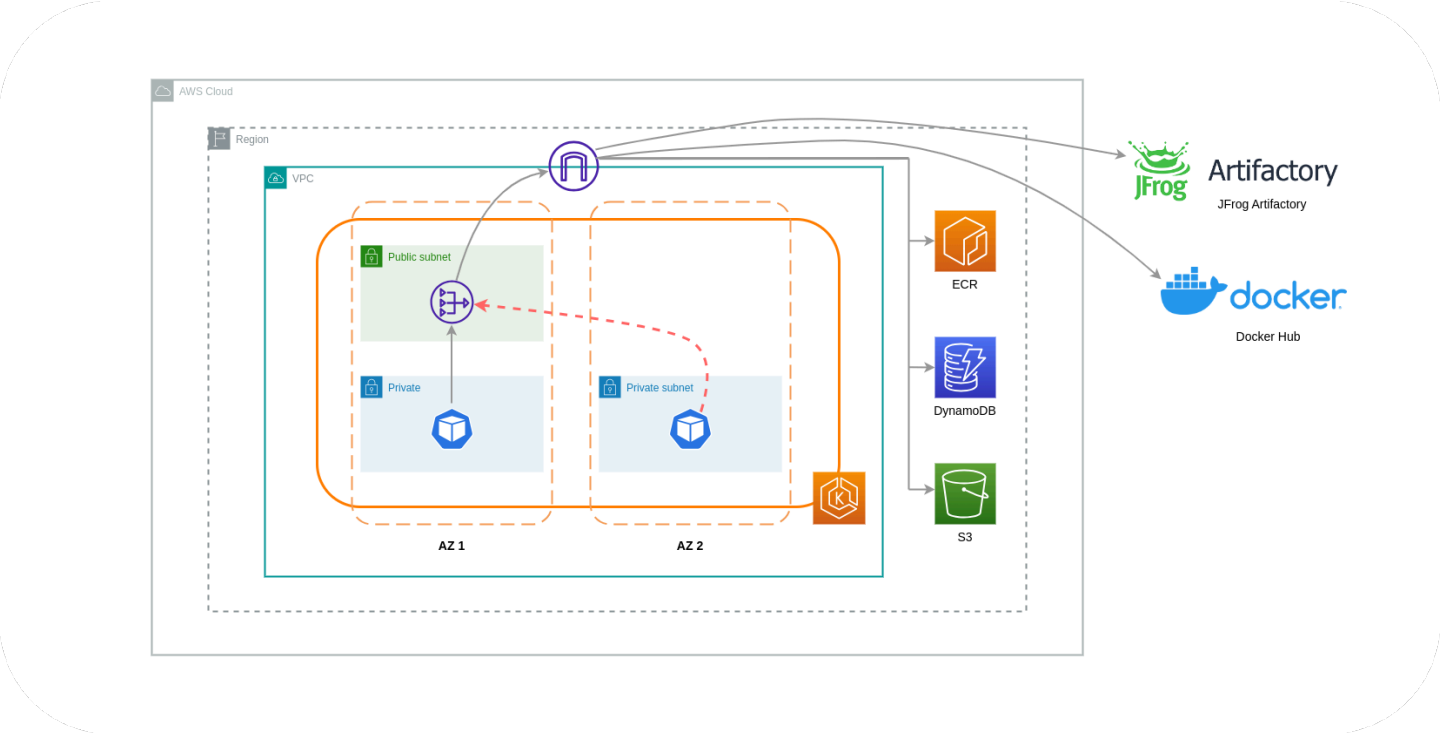
Jika Anda menjalankan beban kerja dengan gambar yang sangat besar, Anda dapat membuat Amazon Machine Images (AMIs) kustom Anda sendiri dengan gambar kontainer pra-cache. Ini dapat mengurangi waktu penarikan gambar awal dan potensi biaya transfer data dari registri kontainer ke node pekerja EKS.

Transfer Data ke Layanan Internet & AWS

Merupakan praktik umum untuk mengintegrasikan beban kerja Kubernetes dengan layanan AWS lainnya atau alat dan platform pihak ketiga melalui Internet. Infrastruktur jaringan yang mendasari yang digunakan untuk mengarahkan lalu lintas ke dan dari tujuan yang relevan dapat memengaruhi biaya yang dikeluarkan dalam proses transfer data.

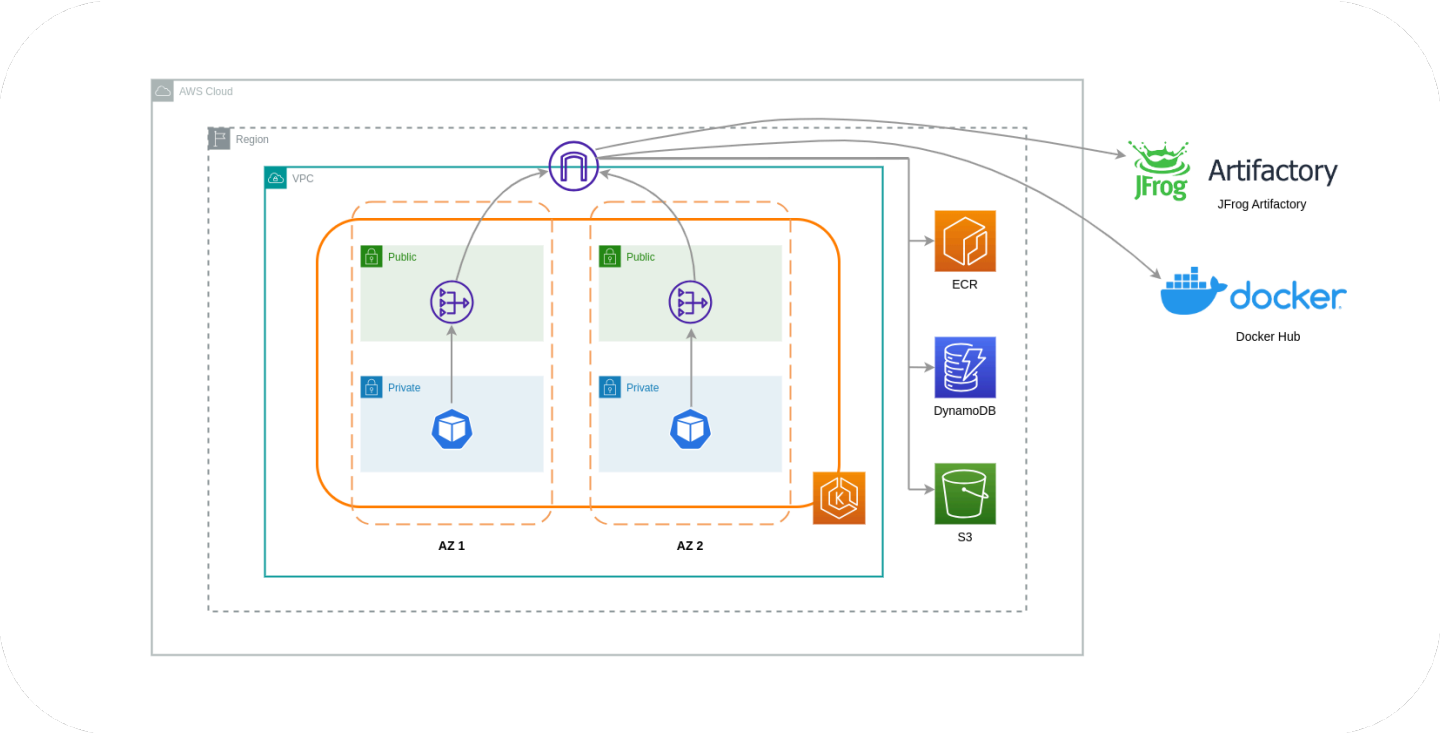
Menggunakan NAT Gateways

NAT Gateways adalah komponen jaringan yang melakukan terjemahan alamat jaringan (NAT). Diagram di bawah ini menggambarkan Pod dalam klaster EKS yang berkomunikasi dengan layanan AWS lainnya (Amazon ECR, DynamoDB, dan S3), dan platform pihak ketiga. Dalam contoh ini, Pod berjalan di subnet pribadi secara terpisah AZs. Untuk mengirim dan menerima lalu lintas dari Internet, NAT Gateway dikerahkan ke subnet publik dari satu AZ, memungkinkan sumber daya apa pun dengan alamat IP pribadi untuk berbagi satu alamat IP publik untuk mengakses Internet. NAT Gateway ini pada gilirannya berkomunikasi dengan komponen Internet Gateway, memungkinkan paket dikirim ke tujuan akhir mereka.



Saat menggunakan NAT Gateways untuk kasus penggunaan seperti itu, Anda dapat meminimalkan biaya transfer data dengan menerapkan NAT Gateway di setiap AZ. Dengan cara ini, lalu lintas yang diarahkan ke Internet akan melalui NAT Gateway di AZ yang sama, menghindari transfer data antar-AZ. Namun, meskipun Anda akan menghemat biaya transfer data antar-AZ, implikasi dari pengaturan ini adalah Anda akan dikenakan biaya tambahan NAT Gateway dalam arsitektur Anda.

Pendekatan yang direkomendasikan ini digambarkan dalam diagram di bawah ini.



Menggunakan VPC Endpoint

Untuk lebih mengurangi biaya dalam arsitektur tersebut, Anda harus menggunakan VPC Endpoint untuk membangun konektivitas antara beban kerja Anda dan layanan AWS. Titik Akhir VPC memungkinkan Anda mengakses layanan AWS dari dalam VPC tanpa data/network paket yang melintasi Internet. Semua lalu lintas bersifat internal dan tetap berada dalam jaringan AWS. Ada dua jenis VPC Endpoint: Interface VPC Endpoints (didukung oleh banyak layanan AWS) dan Gateway VPC Endpoints (hanya didukung oleh S3 dan DynamoDB).

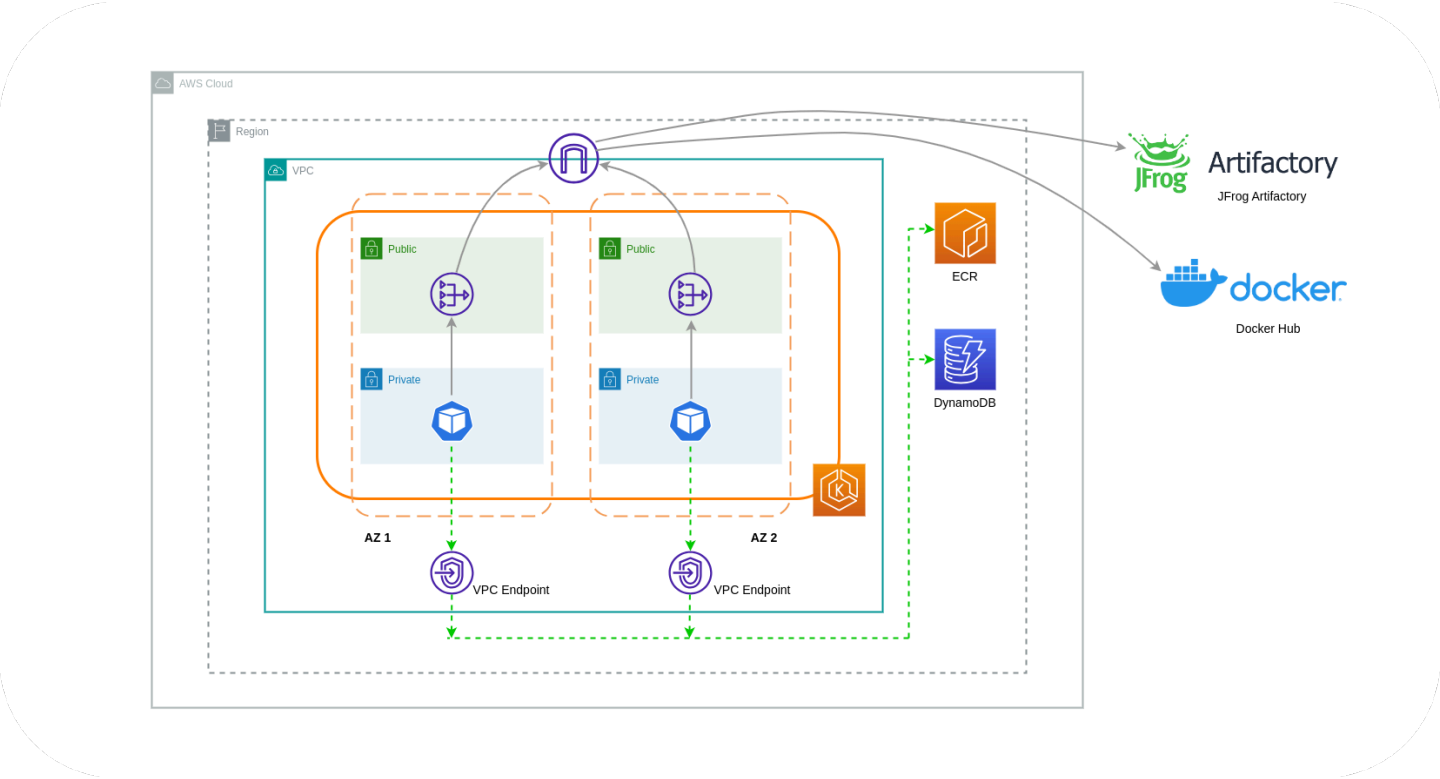
Titik Akhir VPC Gerbang

Tidak ada biaya per jam atau transfer data yang terkait dengan Titik Akhir VPC Gateway. Saat menggunakan Gateway VPC Endpoints, penting untuk dicatat bahwa titik tersebut tidak dapat diperpanjang melintasi batas VPC. Mereka tidak dapat digunakan dalam peering VPC, jaringan VPN, atau melalui Direct Connect.

Antarmuka VPC Endpoint

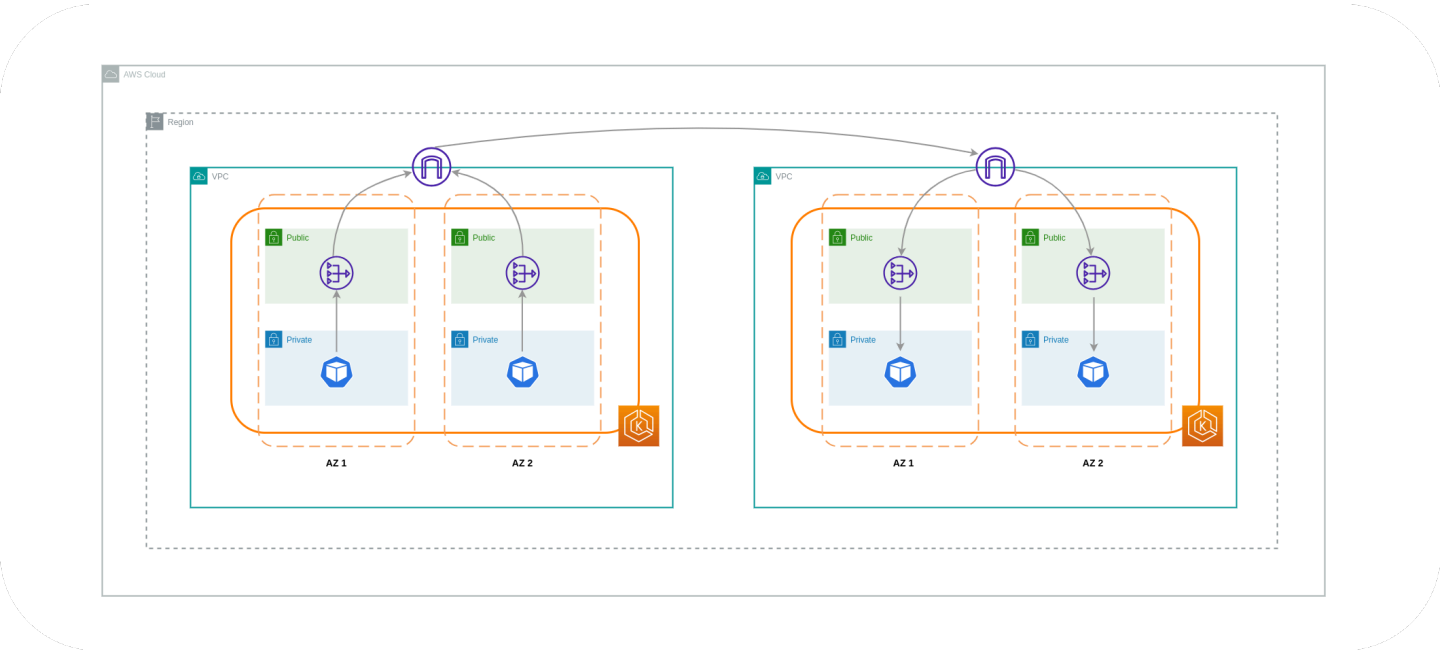
Titik Akhir VPC memiliki biaya per jam dan memiliki biaya tambahan yang terkait dengan pemrosesan data melalui ENI yang mendasarinya. Perhatikan bahwa transfer data antar-AZ tidak dikenakan biaya.

Diagram di bawah ini menunjukkan Pod yang berkomunikasi dengan layanan AWS melalui Titik Akhir VPC.



Transfer Data antara VPCs

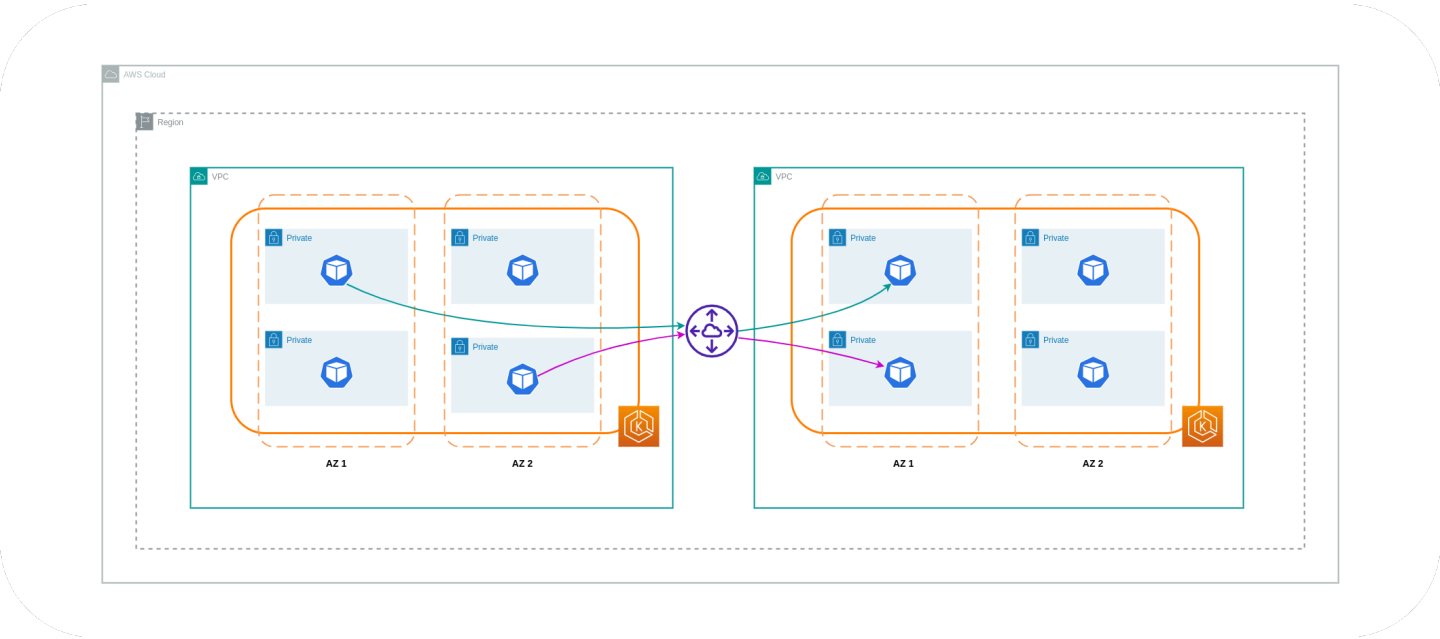
Dalam beberapa kasus, Anda mungkin memiliki beban kerja di VPC yang berbeda (dalam wilayah AWS yang sama) yang perlu berkomunikasi satu sama lain. Hal ini dapat dicapai dengan memungkinkan lalu lintas untuk melintasi internet publik melalui Internet Gateways yang melekat pada VPC masing-masing. Komunikasi tersebut dapat diaktifkan dengan menerapkan komponen infrastruktur seperti instans EC2, NAT Gateways atau instans NAT di subnet publik. Namun, pengaturan termasuk komponen-komponen ini akan dikenakan biaya untuk processing/transferring data masuk dan keluar dari. VPCs Jika lalu lintas ke dan dari VPCs yang terpisah bergerak melintasi AZs, maka akan ada biaya tambahan dalam transfer data. Diagram di bawah ini menggambarkan pengaturan yang menggunakan NAT Gateways dan Internet Gateways untuk membangun komunikasi antara beban kerja yang berbeda. VPCs



Koneksi Peering VPC

Untuk mengurangi biaya untuk kasus penggunaan seperti itu, Anda dapat menggunakan VPC Peering. Dengan koneksi VPC Peering, tidak ada biaya transfer data untuk lalu lintas jaringan yang tetap berada dalam AZ yang sama. Jika lalu lintas melintas AZs, akan ada biaya yang dikeluarkan. Meskipun demikian, pendekatan VPC Peering direkomendasikan untuk komunikasi yang hemat biaya antara beban kerja secara terpisah dalam wilayah AWS yang sama. VPCs Namun, penting untuk dicatat bahwa peering VPC terutama efektif untuk konektivitas VPC 1:1 karena tidak memungkinkan jaringan transitif.

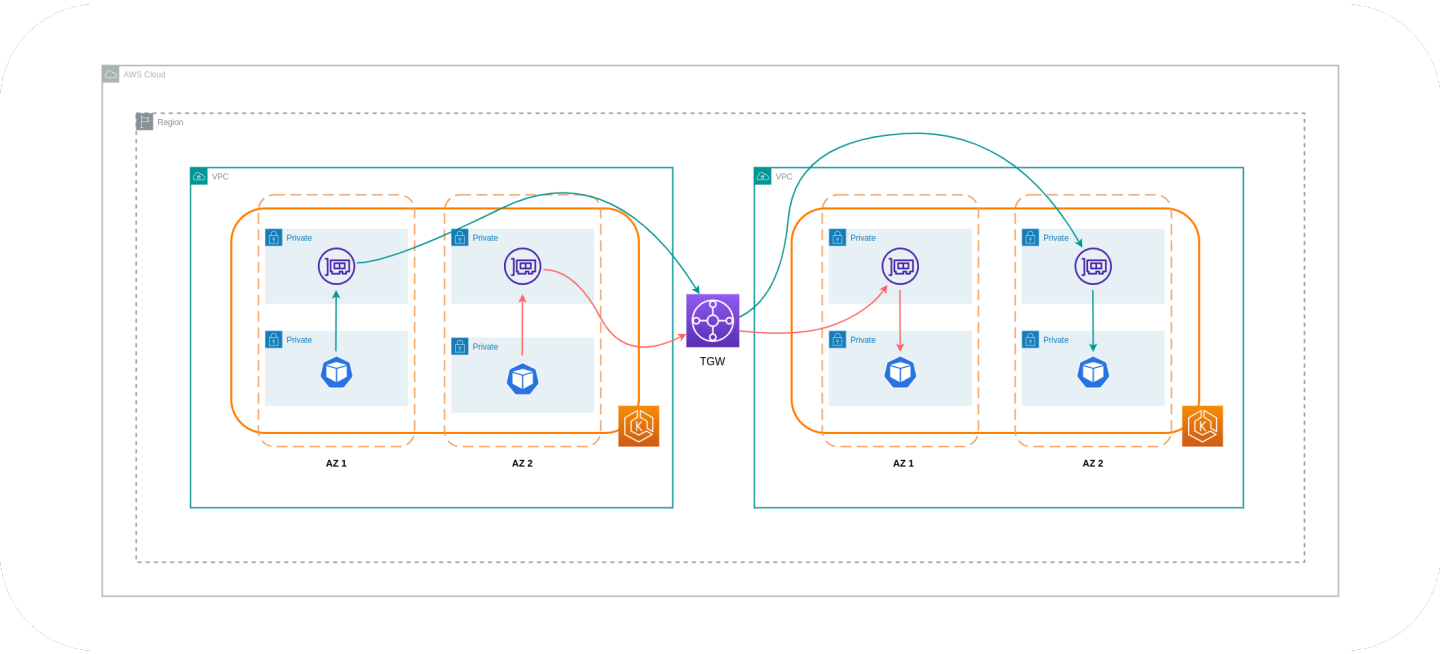
Diagram di bawah ini adalah representasi tingkat tinggi dari komunikasi beban kerja melalui koneksi peering VPC.



Koneksi Jaringan Transitif

Seperti yang ditunjukkan di bagian sebelumnya, koneksi VPC Peering tidak memungkinkan konektivitas jaringan transitif. Jika Anda ingin menghubungkan 3 atau lebih VPCs dengan persyaratan jaringan transitif, maka Anda harus menggunakan Transit Gateway (TGW). Ini akan memungkinkan Anda untuk mengatasi batasan VPC Peering atau overhead operasional apa pun yang terkait dengan memiliki beberapa koneksi VPC Peering antara beberapa. VPCs Anda ditagih setiap jam dan untuk data yang dikirim ke TGW. Tidak ada biaya tujuan yang terkait dengan lalu lintas antar-AZ yang mengalir melalui TGW.

Diagram di bawah ini menunjukkan lalu lintas antar-AZ yang mengalir melalui TGW antara beban kerja yang berbeda VPCs tetapi dalam wilayah AWS yang sama.



Menggunakan Service Mesh

Service mesh menawarkan kemampuan jaringan yang kuat yang dapat digunakan untuk mengurangi biaya terkait jaringan di lingkungan kluster EKS Anda. Namun, Anda harus mempertimbangkan dengan cermat tugas operasional dan kompleksitas yang akan diperkenalkan oleh mesh layanan ke lingkungan Anda jika Anda mengadopsinya.

Membatasi Lalu Lintas ke Availability Zone

Menggunakan Distribusi Tertimbang Lokalitas Istio

Istio memungkinkan Anda menerapkan kebijakan jaringan ke lalu lintas setelah perutean terjadi. Ini dilakukan dengan menggunakan Aturan Tujuan seperti distribusi tertimbang lokalitas. Dengan menggunakan fitur ini, Anda dapat mengontrol bobot (dinyatakan sebagai persentase) lalu lintas yang dapat pergi ke tujuan tertentu berdasarkan asalnya. Sumber lalu lintas ini dapat berasal dari penyeimbang beban eksternal (atau menghadap publik) atau Pod di dalam cluster itu sendiri. Ketika semua titik akhir Pod tersedia, lokalitas akan dipilih berdasarkan algoritma load balancing round-robin tertimbang. Jika titik akhir tertentu tidak sehat atau tidak tersedia, bobot lokalitas akan disesuaikan secara otomatis untuk mencerminkan perubahan ini di titik akhir yang tersedia.

> **Note**
>
> Sebelum menerapkan distribusi tertimbang lokalitas, Anda harus mulai dengan memahami pola lalu lintas jaringan Anda dan implikasi yang mungkin dimiliki kebijakan Aturan Tujuan

terhadap perilaku aplikasi Anda. Karena itu, penting untuk memiliki mekanisme penelusuran terdistribusi dengan alat seperti AWS X-Ray atau Jaeger.

Aturan Destinasi Istio yang dijelaskan di atas juga dapat diterapkan untuk mengelola lalu lintas dari penyeimbang beban ke Pod di klaster EKS Anda. Aturan distribusi tertimbang lokalitas dapat diterapkan pada Layanan yang menerima lalu lintas dari penyeimbang beban yang sangat tersedia (khususnya Ingress Gateway). Aturan-aturan ini memungkinkan Anda untuk mengontrol berapa banyak lalu lintas yang berjalan berdasarkan asal zonalnya - penyeimbang beban dalam kasus ini. Jika dikonfigurasi dengan benar, lalu lintas lintas zona keluar yang lebih sedikit akan terjadi dibandingkan dengan penyeimbang beban yang mendistribusikan lalu lintas secara merata atau acak ke replika Pod yang berbeda. AZs

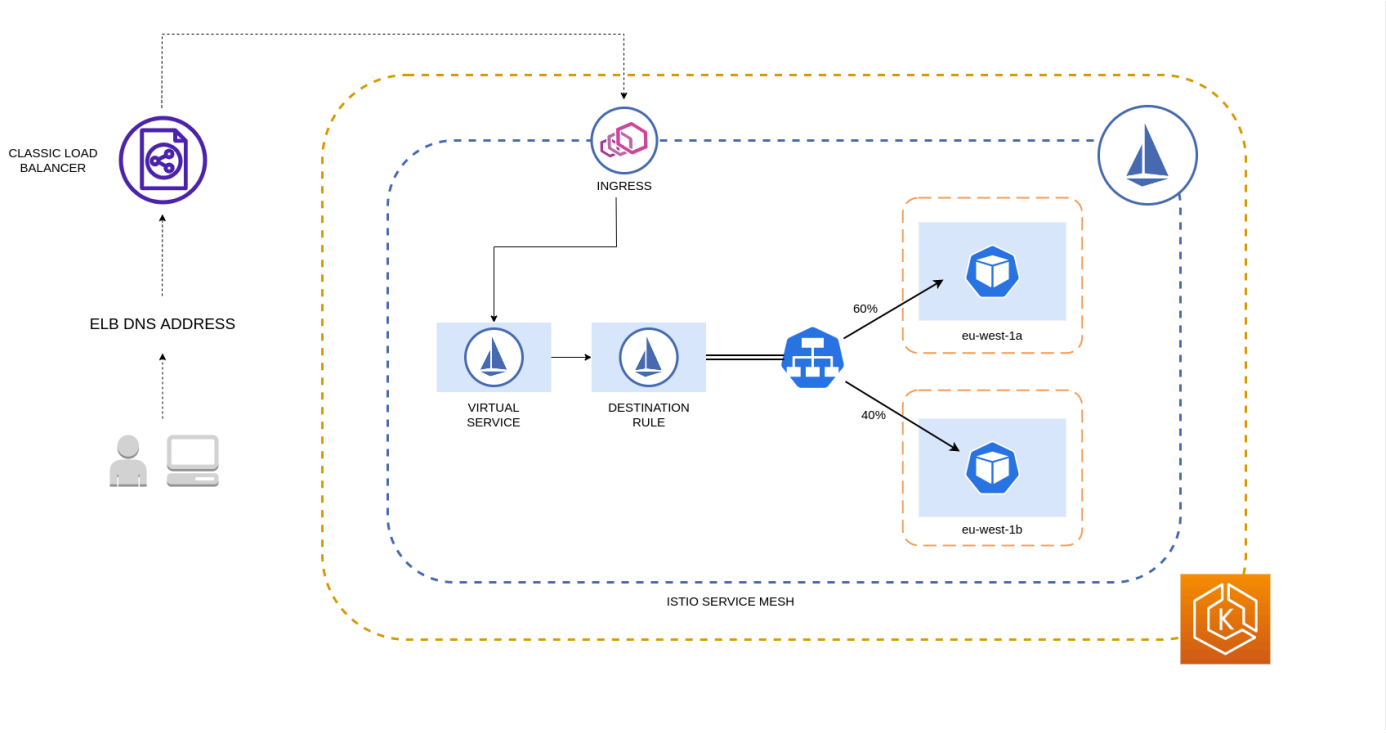
Di bawah ini adalah contoh blok kode sumber daya Aturan Tujuan di Istio. Seperti dapat dilihat di bawah, sumber daya ini menentukan konfigurasi tertimbang untuk lalu lintas masuk dari 3 berbeda AZs di wilayah tersebut. `eu-west-1` Konfigurasi ini menyatakan bahwa sebagian besar lalu lintas masuk (70% dalam hal ini) dari AZ tertentu harus diproksi ke tujuan di AZ yang sama dari mana asalnya.

```
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**
>
> Berat minimum yang dapat didistribusikan tujuan adalah 1%. Alasan untuk ini adalah untuk mempertahankan wilayah dan zona failover jika titik akhir di tujuan utama menjadi tidak sehat atau tidak tersedia.

Diagram di bawah ini menggambarkan skenario di mana ada penyeimbang beban yang sangat tersedia di wilayah eu-west-1 dan distribusi tertimbang lokalitas diterapkan. Kebijakan Aturan Tujuan untuk diagram ini dikonfigurasi untuk mengirim 60% lalu lintas yang berasal dari eu-west-1a ke Pod di AZ yang sama, sedangkan 40% lalu lintas dari eu-west-1a harus masuk ke Pod di eu-west-1b.



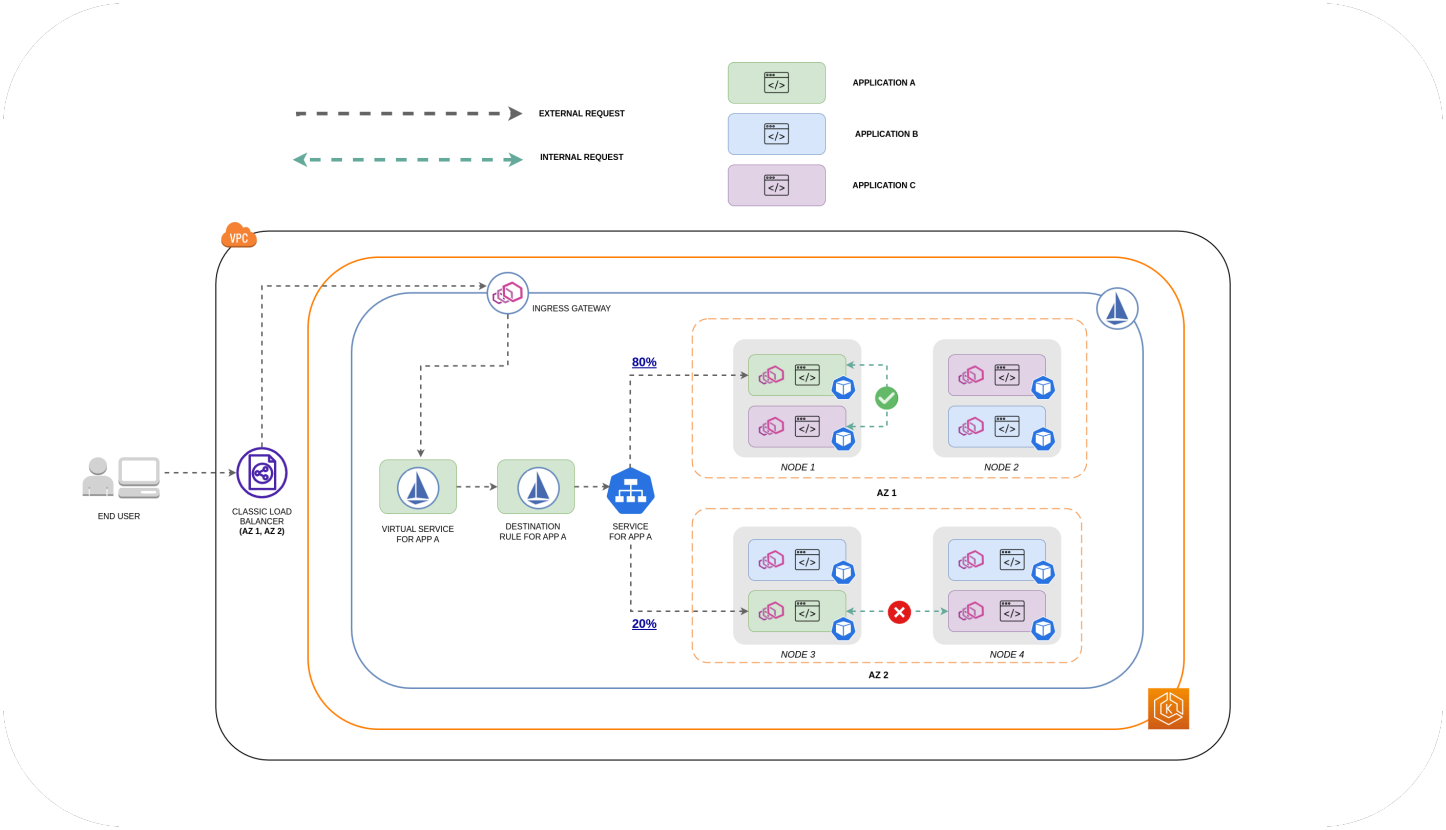
Membatasi Lalu Lintas ke Availability Zone dan Node

Menggunakan Kebijakan Lalu Lintas Internal Layanan dengan Istio

Untuk mengurangi biaya jaringan yang terkait dengan lalu lintas masuk eksternal dan lalu lintas internal antar Pod, Anda dapat menggabungkan Aturan Tujuan Istio dan kebijakan lalu lintas internal Layanan Kubernetes. Cara menggabungkan aturan tujuan Istio dengan kebijakan lalu lintas internal layanan akan sangat bergantung pada 3 hal:

- Peran layanan mikro
- Pola lalu lintas jaringan di seluruh layanan mikro
- Bagaimana layanan mikro harus diterapkan di seluruh topologi klaster Kubernetes

Diagram di bawah ini menunjukkan seperti apa aliran jaringan dalam kasus permintaan bersarang dan bagaimana kebijakan yang disebutkan di atas akan mengontrol lalu lintas.



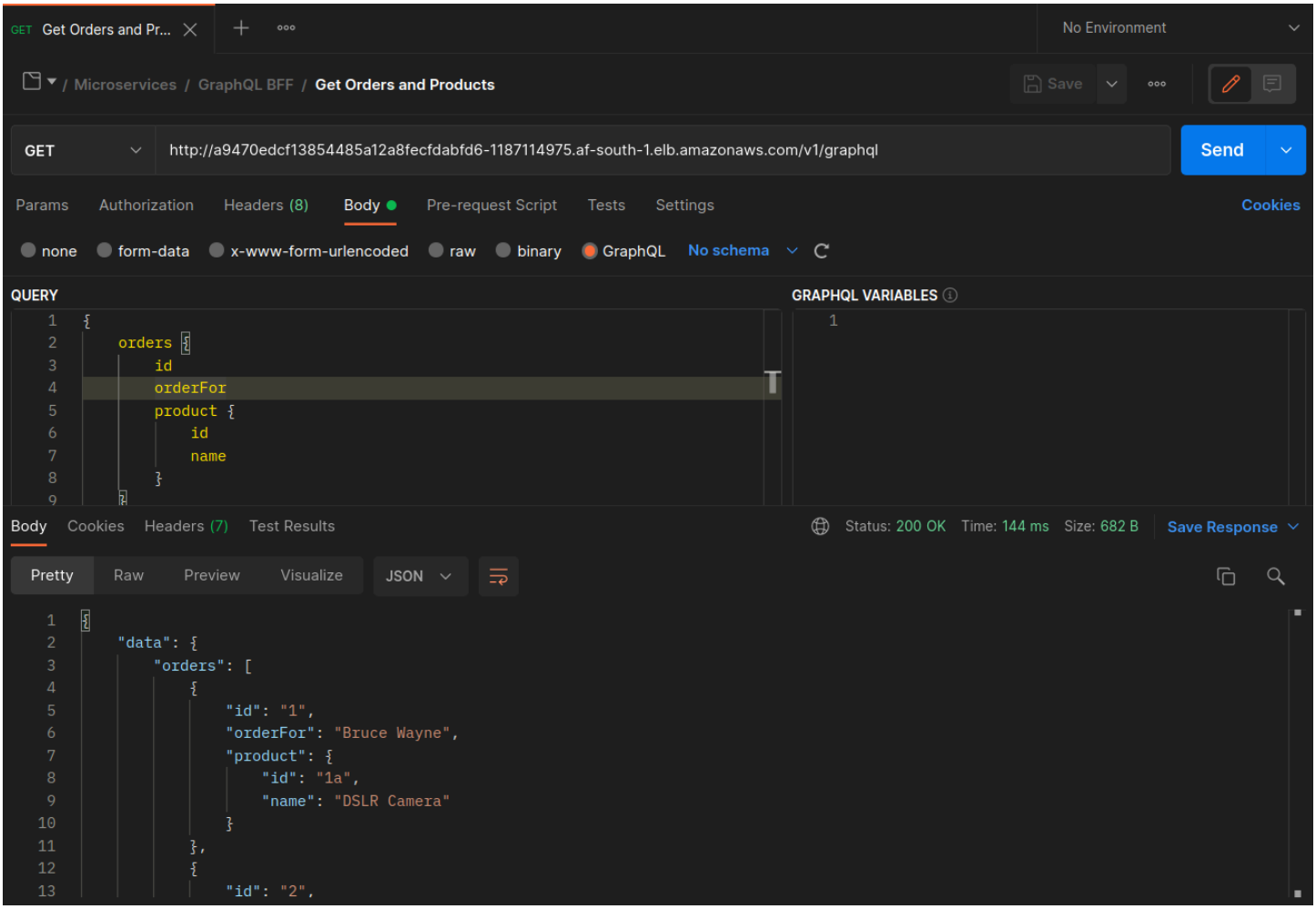
1. Pengguna akhir membuat permintaan ke APP A, yang pada gilirannya membuat permintaan bersarang ke APP C. Permintaan ini pertama kali dikirim ke penyeimbang beban yang sangat tersedia, yang memiliki instance di AZ 1 dan AZ 2 seperti yang ditunjukkan diagram di atas.

2. Permintaan masuk eksternal kemudian diarahkan ke tujuan yang benar oleh Layanan Virtual Istio.
3. Setelah permintaan dialihkan, Aturan Destinasi Istio mengontrol berapa banyak lalu lintas yang masuk ke masing-masing AZs berdasarkan dari mana asalnya (AZ 1 atau AZ 2).
4. Lalu lintas kemudian masuk ke Layanan untuk APP A, dan kemudian diproksi ke titik akhir Pod masing-masing. Seperti yang ditunjukkan pada diagram, 80% dari lalu lintas masuk dikirim ke titik akhir Pod di AZ 1, dan 20% dari lalu lintas masuk dikirim ke AZ 2.
5. APP A kemudian membuat permintaan internal ke APP C. Layanan APP C memiliki kebijakan lalu lintas internal yang diaktifkan (`internalTrafficPolicy`: Local`).
6. Permintaan internal dari APP A (pada NODE 1) ke APP C berhasil karena titik akhir node-lokal yang tersedia untuk APP C.
7. Permintaan internal dari APP A (pada NODE 3) ke APP C gagal karena tidak ada titik akhir node-lokal yang tersedia untuk APP C. Seperti yang ditunjukkan diagram, APP C tidak memiliki replika pada NODE 3. * *

Tangkapan layar di bawah ini diambil dari contoh langsung dari pendekatan ini. Kumpulan tangkapan layar pertama menunjukkan permintaan eksternal yang berhasil ke `graphql` dan permintaan bersarang yang berhasil dari `graphql` ke `orders` replika yang ditempatkan bersama pada node. `ip-10-0-0-151.af-south-1.compute.internal`

test-results.txt

```
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-0-147.af-south-1.compute.i
NAME                        READY   STATUS    RESTARTS   AGE   IP          NODE
products-8567b458c8-brgdp   2/2     Running   0          64m   10.0.0.83   ip-10-0-0-147.af-south-1.com
--------------------------------------------------------------------------
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-1-153.af-south-1.compute.i
NAME                        READY   STATUS    RESTARTS   AGE   IP          NODE
products-8567b458c8-5shq8   2/2     Running   0          65m   10.0.1.4    ip-10-0-1-153.af-south-1.comp
--------------------------------------------------------------------------
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-1-68.af-south-1.compute.in
NAME                      READY   STATUS    RESTARTS   AGE   IP           NODE
orders-684bfbd6d9-5dxcf   2/2     Running   0          22m   10.0.1.145   ip-10-0-1-68.af-south-1.compu
--------------------------------------------------------------------------
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-0-151.af-south-1.compute.i
NAME                      READY   STATUS    RESTARTS   AGE   IP           NODE
graphql-8f877c686-k7k9b   2/2     Running   0          18m   10.0.0.239   ip-10-0-0-151.af-south-1.comp
orders-684bfbd6d9-rc2zp   2/2     Running   0          18m   10.0.0.182   ip-10-0-0-151.af-south-1.comp
```



Dengan Istio, Anda dapat memverifikasi dan mengekspor statistik kluster dan titik akhir hulu yang diketahui oleh proxy Anda. Ini dapat membantu memberikan gambaran aliran jaringan serta pangsa distribusi di antara layanan beban kerja. Melanjutkan dengan contoh yang sama, `orders` titik akhir yang diketahui `graphql` proxy dapat diperoleh dengan menggunakan perintah berikut:

```
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**
```

```
...
```

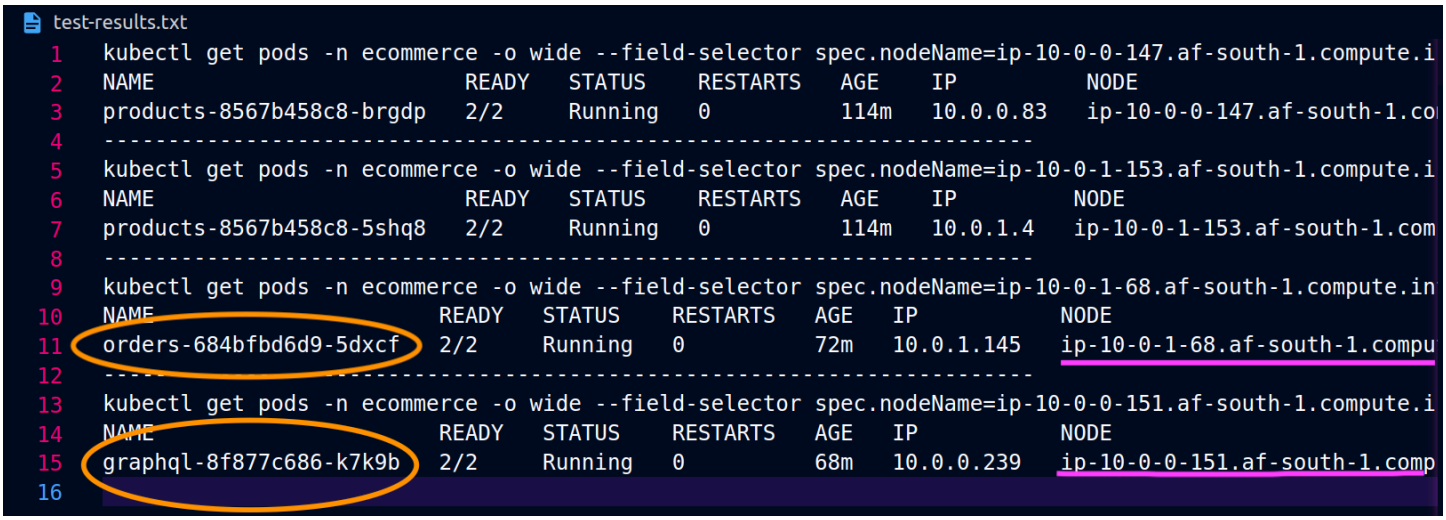
Dalam hal ini, `graphql` proxy hanya mengetahui `orders` titik akhir untuk replika yang berbagi node dengannya. Jika Anda menghapus `internalTrafficPolicy: Local` pengaturan dari Layanan pesanan, dan menjalankan kembali perintah seperti di atas, maka hasilnya akan mengembalikan semua titik akhir replika yang tersebar di node yang berbeda. Selanjutnya, dengan memeriksa titik akhir masing-masing, Anda akan melihat pangsa yang relatif merata dalam distribusi jaringan. `rq_total` Akibatnya, jika titik akhir dikaitkan dengan layanan hulu yang berjalan berbeda AZs, maka distribusi jaringan ini di seluruh zona akan menghasilkan biaya yang lebih tinggi.

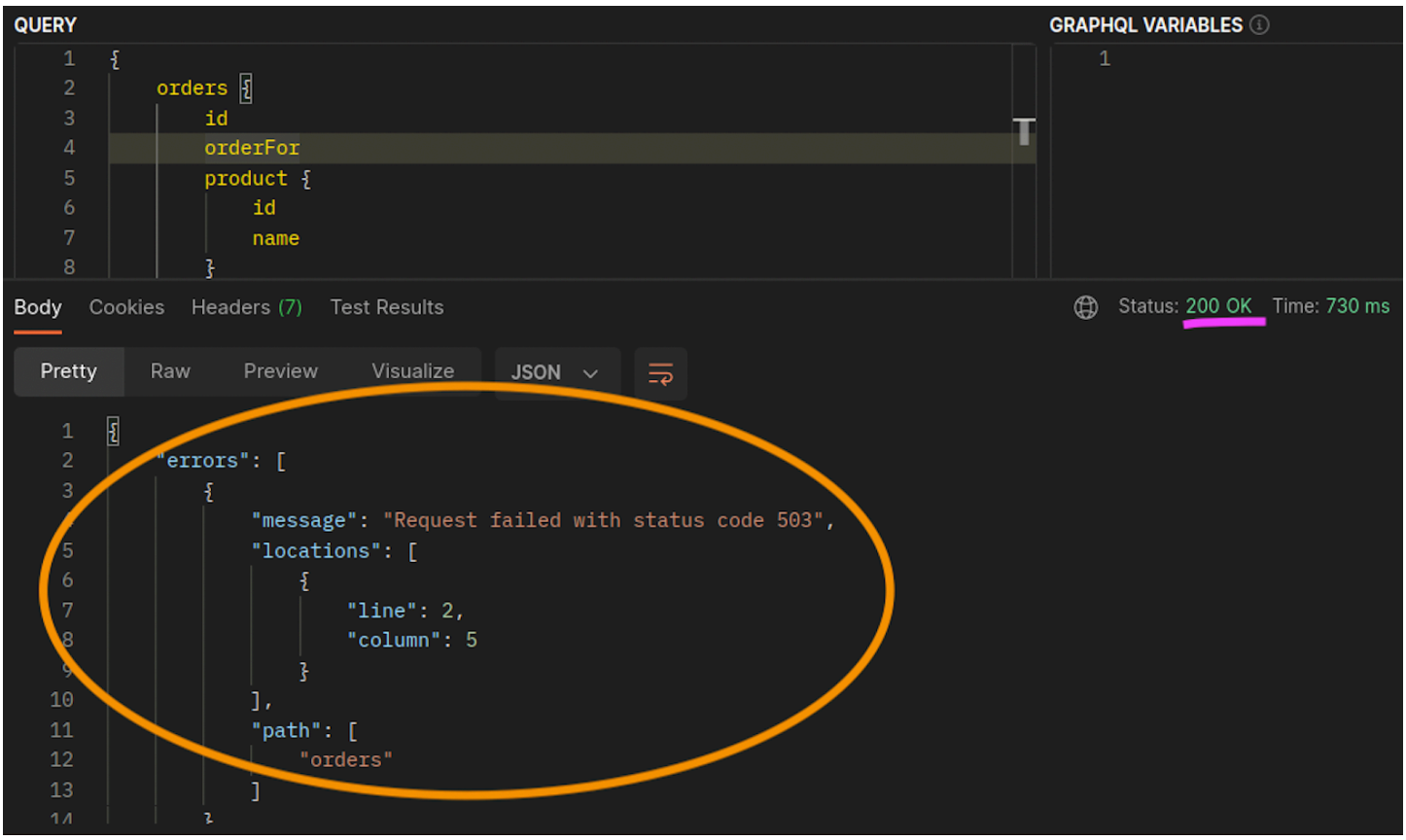
Seperti yang disebutkan di bagian sebelumnya di atas, Anda dapat menempatkan Pod yang sering berkomunikasi dengan memanfaatkan pod-affinity.

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

Ketika `graphql` dan `orders` replika tidak hidup berdampingan pada node (`ip-10-0-0-151.af-south-1.compute.internal`) yang sama, permintaan pertama berhasil seperti yang dicatat oleh

screenshot di Postman `200 response code` di bawah ini, sedangkan permintaan bersarang kedua dari `graphql` gagal `orders` dengan a. `graphql` `503 response code`





Sumber Daya Tambahan

- Mengatasi latensi dan biaya transfer data di EKS menggunakan Istio

- Menjelajahi efek Petunjuk Sadar Topologi pada lalu lintas jaringan di Amazon Elastic Kubernetes Service
- Mendapatkan visibilitas ke pod Amazon EKS Cross-AZ Anda ke byte jaringan pod
- Optimalkan Lalu Lintas AZ dengan Istio
- Optimalkan Lalu Lintas AZ dengan Perutean Sadar Topologi
- Optimalkan Biaya & Kinerja Kubernetes dengan Kebijakan Lalu Lintas Internal Layanan
- Optimalkan Biaya & Kinerja Kubernetes dengan Istio dan Kebijakan Lalu Lintas Internal Layanan
- Ikhtisar Biaya Transfer Data untuk Arsitektur Umum
- Memahami biaya transfer data untuk layanan kontainer AWS

Penyimpanan

Gambaran Umum

Ada skenario di mana Anda mungkin ingin menjalankan aplikasi yang perlu menyimpan data untuk jangka pendek atau jangka panjang. Untuk kasus penggunaan seperti itu, volume dapat ditentukan dan dipasang oleh Pod sehingga kontainernya dapat memanfaatkan mekanisme penyimpanan yang berbeda. Kubernetes mendukung berbagai jenis volume untuk penyimpanan sementara dan persisten. Pilihan penyimpanan sangat tergantung pada persyaratan aplikasi. Untuk setiap pendekatan, ada implikasi biaya, dan praktik yang dirinci di bawah ini yang akan membantu Anda mencapai efisiensi biaya untuk beban kerja yang membutuhkan beberapa bentuk penyimpanan di lingkungan EKS Anda.

Volume Ephemeral

Volume fana adalah untuk aplikasi yang memerlukan volume lokal sementara tetapi tidak memerlukan data untuk bertahan setelah restart. Contohnya termasuk persyaratan untuk ruang awal, caching, dan data input hanya-baca seperti data konfigurasi dan rahasia. Anda dapat menemukan detail lebih lanjut tentang volume fana Kubernetes di sini. Sebagian besar volume fana (misalnya EmptyDir, ConfigMap, DownwardApi, secret, hostpath) didukung oleh perangkat yang dapat ditulis secara lokal (biasanya disk root) atau RAM, jadi penting untuk memilih volume host yang paling hemat biaya dan berkinerja.

Menggunakan Volume EBS

Sebaiknya mulai dengan gp3 sebagai volume root host. Ini adalah volume SSD tujuan umum terbaru yang ditawarkan oleh Amazon EBS dan juga menawarkan harga yang lebih rendah (hingga 20%) per GB dibandingkan dengan volume gp2.

Menggunakan Toko EC2 Instans Amazon

Toko EC2 instans Amazon menyediakan penyimpanan tingkat blok sementara untuk instans Anda EC2 . Penyimpanan yang disediakan oleh toko EC2 instans dapat diakses melalui disk yang secara fisik terpasang ke host. Tidak seperti Amazon EBS, Anda hanya dapat melampirkan volume penyimpanan instans saat instance diluncurkan, dan volume ini hanya ada selama masa pakai instance. Mereka tidak dapat dilepaskan dan dilampirkan kembali ke instance lain. Anda dapat mempelajari lebih lanjut tentang toko EC2 instans Amazon di sini. Tidak ada biaya tambahan yang terkait dengan volume penyimpanan instans. Hal ini membuat mereka (volume penyimpanan instance) lebih hemat biaya daripada EC2 instance umum dengan volume EBS yang besar.

Untuk menggunakan volume penyimpanan lokal di Kubernetes, Anda harus mempartisi, mengonfigurasi, dan memformat disk menggunakan EC2 data pengguna Amazon sehingga volume dapat dipasang sebagai spesifikasi pod. HostPath Atau, Anda dapat memanfaatkan Penyedia Statis Volume Persisten Lokal untuk menyederhanakan manajemen penyimpanan lokal. Penyedia statis Volume Persisten Lokal memungkinkan Anda mengakses volume penyimpanan instans lokal melalui antarmuka Kubernetes PersistentVolumeClaim (PVC) standar. Selanjutnya, ia akan menyediakan PersistentVolumes (PVs) yang berisi informasi afinitas node untuk menjadwalkan Pod ke node yang benar. Meskipun menggunakan Kubernetes PersistentVolumes, volume penyimpanan EC2 instance bersifat fana. Data yang ditulis ke disk fana hanya tersedia selama masa hidup instance. Ketika instance dihentikan, begitu juga datanya. Silakan merujuk ke blog ini untuk lebih jelasnya.

Perlu diingat bahwa saat menggunakan volume penyimpanan EC2 instans Amazon, total batas IOPS dibagikan dengan host dan mengikat Pod ke host tertentu. Anda harus meninjau persyaratan beban kerja Anda secara menyeluruh sebelum mengadopsi volume penyimpanan EC2 instans Amazon.

Volume Persisten

Kubernetes biasanya dikaitkan dengan menjalankan aplikasi stateless. Namun, ada skenario di mana Anda mungkin ingin menjalankan layanan mikro yang perlu menyimpan data atau informasi persisten dari satu permintaan ke permintaan berikutnya. Database adalah contoh umum untuk kasus penggunaan semacam itu. Namun, Pod, dan wadah atau proses di dalamnya, bersifat fana. Untuk mempertahankan data di luar masa pakai Pod, Anda dapat menggunakan PVs untuk

menentukan akses ke penyimpanan di lokasi tertentu yang independen dari Pod. Biaya yang terkait dengan PVs sangat tergantung pada jenis penyimpanan yang digunakan dan bagaimana aplikasi mengkonsumsinya.

Ada berbagai jenis opsi penyimpanan yang mendukung Kubernetes di PVs Amazon EKS yang tercantum di sini. Opsi penyimpanan yang tercakup di bawah ini adalah Amazon EBS, Amazon EFS, Amazon FSx untuk Lustre, Amazon FSx untuk ONTAP. NetApp

Volume Amazon Elastic Block Store (EBS)

Volume Amazon EBS dapat digunakan sebagai Kubernetes PVs untuk menyediakan volume penyimpanan tingkat blok. Ini sangat cocok untuk database yang mengandalkan pembacaan & penulisan acak dan aplikasi intensif throughput yang melakukan pembacaan dan penulisan yang panjang dan berkelanjutan. Driver Amazon Elastic Block Store Container Storage Interface (CSI) memungkinkan kluster Amazon EKS mengelola siklus hidup volume Amazon EBS untuk volume persisten. Container Storage Interface memungkinkan dan memfasilitasi interaksi antara Kubernetes dan sistem penyimpanan. Ketika driver CSI di-deploy ke kluster EKS Anda, Anda dapat mengakses kemampuannya melalui sumber daya penyimpanan Kubernetes asli seperti Persistent Volume (PVs), Persistent Volume Claims () dan Storage Classes (PVCs). SCs Tautan ini memberikan contoh praktis tentang cara berinteraksi dengan volume Amazon EBS dengan driver Amazon EBS CSI.

Memilih volume yang tepat

Kami merekomendasikan penggunaan penyimpanan blok generasi terbaru (gp3) karena memberikan keseimbangan yang tepat antara harga dan kinerja. Ini juga memungkinkan Anda untuk menskalakan IOPS volume dan throughput secara independen dari ukuran volume tanpa perlu menyediakan kapasitas penyimpanan blok tambahan. Jika saat ini Anda menggunakan volume gp2, kami sangat menyarankan untuk bermigrasi ke volume gp3. Posting blog Migrasi kluster Amazon EKS dari gp2 ke gp3 volume EBS menjelaskan cara bermigrasi dari gp2di gp3 di kluster Amazon EKS dengan pencadangan dan pemulihan dengan menggunakan fitur CSI Volume Snapshots, yang memerlukan waktu henti aplikasi.

Amazon EBS memungkinkan perubahan karakteristik volume seperti ukuran volume, IOPS, dan throughput online. Memanfaatkan fitur ini seseorang dapat bermigrasi dari gp2 di gp3 tanpa downtime aplikasi menggunakan anotasi PVC seperti yang dijelaskan di blog ini, yang memerlukan driver EBS CSI v1.19.0+, atau dimulai dengan Amazon EKS v1.31 dan driver EBS CSI 1.35 dengan menggunakan API seperti yang dijelaskan di sini. VolumeAttributesClass

Jika Anda memiliki aplikasi yang membutuhkan kinerja lebih tinggi dan membutuhkan volume yang lebih besar dari yang dapat didukung oleh satu volume gp3, Anda harus mempertimbangkan untuk

menggunakan io2 block express. Jenis penyimpanan ini sangat ideal untuk penyebaran terbesar, paling I/O intensif, dan misi kritis Anda seperti SAP HANA atau database besar lainnya dengan persyaratan latensi rendah. Perlu diingat bahwa kinerja EBS instans dibatasi oleh batas kinerja instans, jadi tidak semua instance mendukung volume ekspres blok io2. Anda dapat memeriksa jenis instans yang didukung dan pertimbangan lain di dokumen ini.

Volume gp3 tunggal dapat mendukung hingga 16.000 IOPS maks, 1.000 throughput maks, MiB/s maks 16TiB. Volume SSD IOPS Provisioned generasi terbaru yang menyediakan hingga 256.000 IOPS, 4.000 MiB/s, throughput, dan 64TiB.

Di antara opsi-opsi ini, Anda sebaiknya menyesuaikan kinerja dan biaya penyimpanan Anda dengan kebutuhan aplikasi Anda.

Memantau dan mengoptimalkan dari waktu ke waktu

Penting untuk memahami kinerja dasar aplikasi Anda dan memantaunya untuk volume yang dipilih untuk memeriksa apakah itu memenuhi kebutuhan Anda requirements/expectations atau jika terlalu banyak disediakan (misalnya skenario di mana IOPS yang disediakan tidak sepenuhnya digunakan).

Alih-alih mengalokasikan volume besar dari awal, Anda dapat secara bertahap meningkatkan ukuran volume saat Anda mengumpulkan data. Anda dapat mengukur ulang volume secara dinamis menggunakan fitur pengubahan ukuran volume di driver Amazon Elastic Block Store CSI (). aws-ebs-csi-driver Perlu diingat bahwa Anda hanya dapat meningkatkan ukuran volume EBS.

Untuk mengidentifikasi dan menghapus volume EBS yang menggantung, Anda dapat menggunakan kategori pengoptimalan biaya penasihat AWS tepercaya. Fitur ini membantu Anda mengidentifikasi volume atau volume yang tidak terikat dengan aktivitas tulis yang sangat rendah untuk jangka waktu tertentu. Ada alat cloud-native open-source, read-only yang disebut Popeye yang memindai klaster Kubernetes langsung dan melaporkan potensi masalah dengan sumber daya dan konfigurasi yang diterapkan. Misalnya, dapat memindai yang tidak digunakan PVs dan PVCs dan memeriksa apakah mereka terikat atau apakah ada kesalahan pemasangan volume.

Untuk penyelaman mendalam tentang pemantauan, silakan merujuk ke panduan observabilitas pengoptimalan biaya EKS.

Satu opsi lain yang dapat Anda pertimbangkan adalah rekomendasi volume AWS Compute Optimizer Amazon EBS. Alat ini secara otomatis mengidentifikasi konfigurasi volume optimal dan tingkat kinerja yang benar yang diperlukan. Misalnya, dapat digunakan untuk pengaturan optimal yang berkaitan dengan IOPS yang disediakan, ukuran volume, dan jenis volume EBS berdasarkan pemanfaatan

maksimum selama 14 hari terakhir. Ini juga mengukur potensi penghematan biaya bulanan yang berasal dari rekomendasinya. Anda dapat meninjau blog ini untuk lebih jelasnya.

Kebijakan retensi cadangan

Anda dapat mencadangkan data pada volume Amazon EBS Anda dengan mengambil point-in-time snapshot. Driver Amazon EBS CSI mendukung snapshot volume. Anda dapat mempelajari cara membuat snapshot dan memulihkan EBS PV menggunakan langkah-langkah yang diuraikan di sini.

Snapshot berikutnya adalah backup tambahan, yang berarti bahwa hanya blok pada perangkat yang telah berubah setelah snapshot terbaru Anda disimpan. Hal ini meminimalkan waktu yang diperlukan untuk membuat snapshot dan menghemat biaya penyimpanan dengan tidak menduplikasi data. Namun, bertambahnya jumlah snapshot EBS lama tanpa kebijakan retensi yang tepat dapat menyebabkan biaya tak terduga saat beroperasi dalam skala besar. Jika Anda langsung mencadangkan volume Amazon EBS melalui AWS API, Anda dapat memanfaatkan Amazon Data Lifecycle Manager (DLM) yang menyediakan solusi manajemen siklus hidup berbasis kebijakan otomatis untuk Amazon Elastic Block Store (EBS) Snapshots dan Amazon Machine Images () yang didukung EBS. AMIs Konsol membuatnya lebih mudah untuk mengotomatiskan pembuatan, retensi, dan penghapusan Snapshots EBS dan. AMIs

> **Note**
>
> Saat ini tidak ada cara untuk menggunakan Amazon DLM melalui driver Amazon EBS CSI.

Di lingkungan Kubernetes, Anda dapat memanfaatkan alat sumber terbuka yang disebut Velero untuk membuat cadangan Volume Persisten EBS Anda. Anda dapat mengatur tanda TTL saat menjadwalkan pekerjaan untuk kedaluwarsa cadangan. Berikut adalah panduan dari Velero sebagai contoh.

Amazon Elastic File System (EFS)

Amazon Elastic File System (EFS) adalah sistem file tanpa server dan sepenuhnya elastis yang memungkinkan Anda berbagi data file menggunakan antarmuka sistem file standar dan semantik sistem file untuk spektrum beban kerja dan aplikasi yang luas. Contoh beban kerja dan aplikasi termasuk Wordpress dan Drupal, alat pengembang seperti JIRA dan Git, dan sistem notebook bersama seperti Jupyter serta direktori rumah.

Salah satu manfaat utama Amazon EFS adalah dapat dipasang oleh beberapa kontainer yang tersebar di beberapa node dan beberapa zona ketersediaan. Manfaat lain adalah Anda hanya

membayar untuk penyimpanan yang Anda gunakan. Sistem file EFS akan secara otomatis tumbuh dan menyusut saat Anda menambah dan menghapus file yang menghilangkan kebutuhan untuk perencanaan kapasitas.

Untuk menggunakan Amazon EFS di Kubernetes, Anda perlu menggunakan Driver Amazon Elastic File System Container Storage Interface (CSI),. aws-efs-csi-driver Saat ini, pengemudi dapat secara dinamis membuat titik akses. Namun, sistem file Amazon EFS harus disediakan terlebih dahulu dan disediakan sebagai input ke parameter kelas penyimpanan Kubernetes.

Memilih kelas penyimpanan EFS yang tepat

Amazon EFS menawarkan empat kelas penyimpanan.

Dua kelas penyimpanan standar:

- Amazon EFS Standar
- Amazon EFS Standar-Jarang Akses (EFS Standard-IA)

Dua kelas penyimpanan satu zona:

- Amazon EFS Satu Zona
- Amazon EFS Satu Zona Akses Jarang (EFS One Zone-IA)

Kelas penyimpanan Infrequent Access (IA) dioptimalkan biaya untuk file yang tidak diakses setiap hari. Dengan manajemen siklus hidup Amazon EFS, Anda dapat memindahkan file yang belum diakses selama durasi kebijakan siklus hidup (7, 14, 30, 60, atau 90 hari) ke kelas penyimpanan IA yang dapat mengurangi biaya penyimpanan hingga 92 persen dibandingkan dengan kelas penyimpanan EFS Standard dan EFS One Zone masing-masing.

Dengan EFS Intelligent-Tiering, manajemen siklus hidup memantau pola akses sistem file Anda dan secara otomatis memindahkan file ke kelas penyimpanan yang paling optimal.

> **Note**
>
> aws-efs-csi-driver saat ini tidak memiliki kontrol untuk mengubah kelas penyimpanan, manajemen siklus hidup, atau Intelligent-Tiering. Itu harus diatur secara manual di konsol AWS atau melalui EFS APIs.

> **Note**
>
> aws-efs-csi-driver tidak kompatibel dengan gambar kontainer berbasis Windows.

> **Note**
>
> Ada masalah memori yang diketahui ketika vol-metrics-opt-in(untuk memancarkan metrik volume) diaktifkan karena DiskUsagefungsi yang mengkonsumsi sejumlah memori yang sebanding dengan ukuran sistem file Anda. Saat ini, kami sarankan untuk menonaktifkan opsi vol-metrics-opt-in `-- `pada sistem file besar untuk menghindari terlalu banyak memori. Berikut adalah tautan masalah github untuk lebih jelasnya.

Amazon FSx untuk Lustre

Lustre adalah sistem file paralel berkinerja tinggi yang biasa digunakan dalam beban kerja yang membutuhkan throughput hingga ratusan dan sub-milidetik latensi per operasi GB/s . Ini digunakan untuk skenario seperti pelatihan pembelajaran mesin, pemodelan keuangan, HPC, dan pemrosesan video. Amazon FSx for Lustre menyediakan penyimpanan bersama yang dikelola sepenuhnya dengan skalabilitas dan kinerja, terintegrasi dengan mulus dengan Amazon S3.

Anda dapat menggunakan volume penyimpanan persisten Kubernetes yang didukung oleh FSx for Lustre menggunakan driver for Lustre CSI dari Amazon FSx EKS atau cluster Kubernetes yang dikelola sendiri di AWS. Lihat dokumentasi Amazon EKS untuk detail dan contoh selengkapnya.

Tautan ke Amazon S3

Disarankan untuk menautkan repositori data jangka panjang yang sangat tahan lama yang berada di Amazon S3 dengan sistem file for Lustre Anda FSx . Setelah ditautkan, kumpulan data besar dimuat dengan lambat sesuai kebutuhan dari Amazon S3 hingga untuk sistem file Lustre. FSx Anda juga dapat menjalankan analisis dan hasil Anda kembali ke S3, dan kemudian menghapus sistem file Lustre Anda.

Memilih opsi penyebaran dan penyimpanan yang tepat

FSx untuk Lustre menyediakan opsi penerapan yang berbeda. Opsi pertama disebut scratch dan tidak mereplikasi data, sedangkan opsi kedua disebut persisten yang, seperti namanya, mempertahankan data.

Opsi pertama (awal) dapat digunakan untuk mengurangi biaya pemrosesan data jangka pendek sementara. Opsi penerapan persisten dirancang untuk penyimpanan jangka panjang yang secara otomatis mereplikasi data dalam AWS Availability Zone. Ini juga mendukung penyimpanan SSD dan HDD.

Anda dapat mengonfigurasi jenis penerapan yang diinginkan di bawah parameter di Kubernetes FSx for lustre filesystem. StorageClass Berikut adalah link yang menyediakan contoh template.

> **Note**
>
> Untuk beban kerja atau beban kerja yang sensitif terhadap latensi yang membutuhkan tingkat IOP/throughput tertinggi, Anda harus memilih penyimpanan SSD. Untuk beban kerja yang berfokus pada throughput yang tidak sensitif terhadap latensi, Anda harus memilih penyimpanan HDD.

Aktifkan kompresi data

Anda juga dapat mengaktifkan kompresi data pada sistem file Anda dengan menentukan "LZ4" sebagai Jenis Kompresi Data. Setelah diaktifkan, semua file yang baru ditulis akan secara otomatis dikompresi FSx untuk Lustre sebelum ditulis ke disk dan tidak dikompresi saat dibaca. LZ4 algoritma kompresi data lossless sehingga data asli dapat sepenuhnya direkonstruksi dari data terkompresi.

Anda dapat mengonfigurasi tipe kompresi data seperti LZ4 di bawah parameter di Kubernetes FSx for lustre filesystem. StorageClass Kompresi dinonaktifkan ketika nilai diatur ke NONE, yang merupakan default. Tautan ini menyediakan contoh templat.

> **Note**
>
> Amazon FSx for Lustre tidak kompatibel dengan gambar kontainer berbasis Windows.

Amazon FSx untuk NetApp ONTAP

Amazon FSx untuk NetApp ONTAP adalah penyimpanan bersama yang dikelola sepenuhnya yang dibangun di atas sistem NetApp file ONTAP. FSx untuk ONTAP menyediakan penyimpanan file bersama yang kaya fitur, cepat, dan fleksibel yang dapat diakses secara luas dari instans komputasi Linux, Windows, dan macOS yang berjalan di AWS atau di tempat.

Amazon FSx untuk NetApp ONTAP mendukung dua tingkatan penyimpanan: 1/tier primer dan 2/ kapasitas pool tier.

Tingkat utama adalah tingkat berbasis SSD berkinerja tinggi yang disediakan untuk data aktif dan sensitif latensi. Tingkat kolam kapasitas yang sepenuhnya elastis dioptimalkan biaya untuk data yang jarang diakses, secara otomatis diskalakan saat data berjenjang, dan menawarkan kapasitas petabyte yang hampir tidak terbatas. Anda dapat mengaktifkan kompresi dan deduplikasi data pada penyimpanan kolam kapasitas dan selanjutnya mengurangi jumlah kapasitas penyimpanan yang dikonsumsi data Anda. NetApp FabricPool Fitur asli berbasis kebijakan terus memantau pola akses data, secara otomatis mentransfer data dua arah antara tingkatan penyimpanan untuk mengoptimalkan kinerja dan biaya.

NetAppAstra Trident menyediakan orkestrasi penyimpanan dinamis menggunakan driver CSI yang memungkinkan kluster Amazon EKS mengelola siklus hidup PVs volume persisten yang didukung oleh Amazon untuk sistem file ONTAP. FSx NetApp Untuk memulai, lihat Menggunakan Astra Trident dengan Amazon FSx untuk NetApp ONTAP di dokumentasi Astra Trident.

Pertimbangan lainnya

Minimalkan ukuran gambar kontainer

Setelah kontainer diterapkan, gambar kontainer di-cache pada host sebagai beberapa lapisan. Dengan mengurangi ukuran gambar, jumlah penyimpanan yang dibutuhkan pada host dapat dikurangi.

Dengan menggunakan gambar dasar ramping seperti gambar awal atau gambar kontainer distroless (yang hanya berisi aplikasi Anda dan dependensi runtime-nya) dari awal, Anda dapat mengurangi biaya penyimpanan selain manfaat tambahan lainnya seperti mengurangi luas permukaan serangan dan waktu tarik gambar yang lebih pendek.

Anda juga harus mempertimbangkan untuk menggunakan alat open source, seperti Slim.ai yang menyediakan cara mudah dan aman untuk membuat gambar minimal.

Beberapa lapisan paket, alat, dependensi aplikasi, pustaka dapat dengan mudah membengkak ukuran gambar wadah. Dengan menggunakan build multi-tahap, Anda dapat menyalin artefak secara selektif dari satu tahap ke tahap lainnya, tidak termasuk semua yang tidak diperlukan dari gambar akhir. Anda dapat memeriksa lebih banyak praktik terbaik pembuatan gambar di sini.

Hal lain yang perlu dipertimbangkan adalah berapa lama untuk mempertahankan gambar yang di-cache. Anda mungkin ingin membersihkan gambar basi dari cache gambar ketika sejumlah disk

digunakan. Melakukannya akan membantu memastikan Anda memiliki cukup ruang untuk operasi host. Secara default, kubelet melakukan pengumpulan sampah pada gambar yang tidak digunakan setiap lima menit dan pada wadah yang tidak digunakan setiap menit.

Untuk mengonfigurasi opsi untuk penampung yang tidak terpakai dan pengumpulan sampah gambar, atur kubelet menggunakan file konfigurasi dan ubah parameter yang terkait dengan pengumpulan sampah menggunakan tipe sumber daya. *`KubeletConfiguration`*

Anda dapat mempelajarinya lebih lanjut di dokumentasi Kubernetes.

Observabilitas

Pengantar

Alat observabilitas membantu Anda mendeteksi, memulihkan, dan menyelidiki beban kerja Anda secara efisien. Biaya data telemetri meningkat secara alami seiring dengan meningkatnya penggunaan EKS. Terkadang, mungkin sulit untuk menyeimbangkan kebutuhan operasional Anda dan mengukur apa yang penting bagi bisnis Anda dan menjaga biaya pengamatan tetap terkendali. Panduan ini berfokus pada strategi pengoptimalan biaya untuk tiga pilar pengamatan: log, metrik, dan jejak. Masing-masing praktik terbaik ini dapat diterapkan secara independen agar sesuai dengan tujuan pengoptimalan organisasi Anda.

Pencatatan log

Logging memainkan peran penting dalam memantau dan memecahkan masalah aplikasi di cluster Anda. Ada beberapa strategi yang dapat digunakan untuk mengoptimalkan biaya logging. Strategi praktik terbaik yang tercantum di bawah ini termasuk memeriksa kebijakan penyimpanan log Anda untuk menerapkan kontrol terperinci tentang berapa lama data log disimpan, mengirim data log ke opsi penyimpanan yang berbeda berdasarkan kepentingan, dan memanfaatkan penyaringan log untuk mempersempit jenis pesan log yang disimpan. Mengelola telemetri log secara efisien dapat menghemat biaya untuk lingkungan Anda.

Pesawat Kontrol EKS

Optimalkan Log Pesawat Kontrol Anda

Bidang kontrol Kubernetes adalah sekumpulan komponen yang mengelola cluster dan komponen ini mengirim berbagai jenis informasi sebagai aliran log ke grup log di Amazon. CloudWatch Meskipun

ada manfaat untuk mengaktifkan semua jenis log bidang kontrol, Anda harus mengetahui informasi di setiap log dan biaya terkait untuk menyimpan semua telemetri log. Anda dikenakan biaya untuk konsumsi data CloudWatch Log standar dan biaya penyimpanan untuk log yang dikirim ke Amazon CloudWatch Logs dari cluster Anda. Sebelum mengaktifkannya, evaluasi apakah setiap aliran log diperlukan.

Misalnya, di klaster non-produksi, aktifkan jenis log tertentu secara selektif, seperti log server api, hanya untuk analisis dan nonaktifkan sesudahnya. Tetapi untuk cluster produksi, di mana Anda mungkin tidak dapat mereproduksi peristiwa, dan menyelesaikan masalah memerlukan lebih banyak informasi log, maka Anda dapat mengaktifkan semua jenis log. Rincian implementasi pengoptimalan biaya bidang kontrol lebih lanjut ada di posting blog ini.

Streaming Log ke S3

Praktik terbaik pengoptimalan biaya lainnya adalah streaming log pesawat kontrol ke S3 melalui langganan CloudWatch Log. Memanfaatkan langganan CloudWatch Log memungkinkan Anda meneruskan log secara selektif ke S3 yang menyediakan penyimpanan jangka panjang yang lebih hemat biaya dibandingkan dengan menyimpan log tanpa batas waktu. CloudWatch Misalnya, untuk kluster produksi, Anda dapat membuat grup log kritis dan memanfaatkan langganan untuk mengalirkan log ini ke S3 setelah 15 hari. Ini akan memastikan Anda memiliki akses cepat ke log untuk analisis tetapi juga menghemat biaya dengan memindahkan log ke penyimpanan yang lebih hemat biaya.

> **Important**
>
> Pada 9/5/2023 log EKS diklasifikasikan sebagai Vded Logs di Amazon Logs. CloudWatch Vending Logs adalah log layanan AWS khusus yang diterbitkan secara native oleh layanan AWS atas nama pelanggan dan tersedia dengan harga diskon volume. Silakan kunjungi halaman CloudWatch harga Amazon untuk mempelajari lebih lanjut tentang harga Vending Logs.

Pesawat Data EKS

Retensi Log

Kebijakan penyimpanan default Amazon CloudWatch adalah menyimpan log tanpa batas waktu dan tidak pernah kedaluwarsa, menimbulkan biaya penyimpanan yang berlaku untuk wilayah AWS Anda.

Untuk mengurangi biaya penyimpanan, Anda dapat menyesuaikan kebijakan penyimpanan untuk setiap grup log berdasarkan persyaratan beban kerja Anda.

Dalam lingkungan pengembangan, periode retensi yang panjang mungkin tidak diperlukan. Namun di lingkungan produksi, Anda dapat menetapkan kebijakan retensi yang lebih lama untuk memenuhi persyaratan pemecahan masalah, kepatuhan, dan perencanaan kapasitas. Misalnya, jika Anda menjalankan aplikasi e-commerce selama musim liburan puncak, sistem berada di bawah beban yang lebih berat dan masalah dapat muncul yang mungkin tidak segera terlihat, Anda akan ingin mengatur retensi log yang lebih lama untuk pemecahan masalah terperinci dan analisis posting acara.

Anda dapat mengonfigurasi periode retensi Anda di CloudWatch konsol AWS atau AWS API dengan durasi dari 1 hari hingga 10 tahun berdasarkan setiap grup log. Memiliki periode retensi yang fleksibel dapat menghemat biaya penyimpanan log, sekaligus mempertahankan log penting.

Opsi Penyimpanan Log

Penyimpanan adalah pendorong besar biaya observabilitas oleh karena itu sangat penting untuk mengoptimalkan strategi penyimpanan log Anda. Strategi Anda harus selaras dengan persyaratan beban kerja Anda sambil mempertahankan kinerja dan skalabilitas. Salah satu strategi untuk mengurangi biaya penyimpanan log adalah dengan memanfaatkan bucket AWS S3 dan tingkatan penyimpanannya yang berbeda.

Teruskan log langsung ke S3

Pertimbangkan untuk meneruskan log yang kurang penting, seperti lingkungan pengembangan, langsung ke S3 alih-alih Cloudwatch. Ini dapat berdampak langsung pada biaya penyimpanan log. Salah satu opsi adalah meneruskan log langsung ke S3 menggunakan Fluentbit. Anda menentukan ini di `[OUTPUT]` bagian, tujuan di mana FluentBit mentransmisikan log kontainer untuk retensi. Tinjau parameter konfigurasi tambahan di sini.

```
[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
```

Teruskan log CloudWatch hanya untuk analisis jangka pendek

Untuk log yang lebih penting, seperti lingkungan produksi di mana Anda mungkin perlu melakukan analisis segera pada data, pertimbangkan untuk meneruskan log ke. CloudWatch Anda menentukan ini di `[OUTPUT]` bagian, tujuan di mana FluentBit mentransmisikan log kontainer untuk retensi. Tinjau parameter konfigurasi tambahan di sini.

```
[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
```

Namun, ini tidak akan berdampak instan pada penghematan biaya Anda. Untuk penghematan tambahan, Anda harus mengekspor log ini ke Amazon S3.

Ekspor ke Amazon S3 dari CloudWatch

Untuk menyimpan CloudWatch log Amazon dalam jangka panjang, sebaiknya ekspor CloudWatch log Amazon EKS Anda ke Amazon Simple Storage Service (Amazon S3). Anda dapat meneruskan log ke bucket Amazon S3 dengan membuat tugas ekspor melalui Konsol atau API. Setelah Anda melakukannya, Amazon S3 menghadirkan banyak opsi untuk lebih mengurangi biaya. Anda dapat menentukan aturan Siklus Hidup Amazon S3 Anda sendiri untuk memindahkan log Anda ke kelas penyimpanan yang sesuai dengan kebutuhan Anda, atau memanfaatkan kelas penyimpanan Amazon S3 Intelligent-Tiering agar AWS memindahkan data secara otomatis ke penyimpanan jangka panjang berdasarkan pola penggunaan Anda. Silakan merujuk ke blog ini untuk lebih jelasnya. Misalnya, untuk lingkungan produksi Anda, log berada CloudWatch selama lebih dari 30 hari kemudian diekspor ke bucket Amazon S3. Anda kemudian dapat menggunakan Amazon Athena untuk menanyakan data di bucket Amazon S3 jika Anda perlu merujuk kembali ke log di lain waktu.

Kurangi Level Log

Berlatih pencatatan selektif untuk aplikasi Anda. Baik aplikasi dan node Anda mengeluarkan log secara default. Untuk log aplikasi Anda, sesuaikan level log agar selaras dengan kekritisan beban kerja dan lingkungan. Misalnya, aplikasi java di bawah ini mengeluarkan `INFO` log yang merupakan konfigurasi aplikasi default yang khas dan tergantung pada kode dapat menghasilkan volume data log yang tinggi.

```
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!");    } }
```

Dalam lingkungan pengembangan, ubah level log Anda menjadi`DEBUG`, karena ini dapat membantu Anda men-debug masalah atau menangkap masalah potensial sebelum masuk ke produksi.

```
      log.setLevel(Level.DEBUG);
```

Dalam lingkungan produksi, pertimbangkan untuk memodifikasi level log Anda ke `ERROR` atau`FATAL`. Ini akan menampilkan log hanya ketika aplikasi Anda memiliki kesalahan, mengurangi output log dan membantu Anda fokus pada data penting tentang status aplikasi Anda.

```
      log.setLevel(Level.ERROR);
```

Anda dapat menyempurnakan berbagai level log komponen Kubernetes. Misalnya, jika Anda menggunakan Bottlerocket sebagai sistem operasi EKS Node Anda, ada pengaturan konfigurasi yang memungkinkan Anda untuk menyesuaikan tingkat log proses kubelet. Cuplikan pengaturan konfigurasi ini ada di bawah ini. Perhatikan tingkat log default 2 yang menyesuaikan verbositas proses logging. `kubelet`

```
[settings.kubernetes]
log-level = "2"
image-gc-high-threshold-percent = "85"
image-gc-low-threshold-percent = "80"
```

Untuk lingkungan pengembangan, Anda dapat mengatur level log lebih besar dari 2 untuk melihat peristiwa tambahan, ini bagus untuk debugging. Untuk lingkungan produksi, Anda dapat mengatur level ke 0 untuk hanya melihat peristiwa penting.

Filter Leverage

Saat menggunakan konfigurasi EKS Fluentbit default untuk mengirim log kontainer ke Cloudwatch, FluentBit menangkap dan mengirim SEMUA log kontainer aplikasi yang diperkaya dengan metadata Kubernetes ke Cloudwatch seperti yang ditunjukkan pada blok konfigurasi di bawah ini. `[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}
```

`[INPUT]`Bagian di atas menelan semua log kontainer. Ini dapat menghasilkan sejumlah besar data yang mungkin tidak diperlukan. Memfilter data ini dapat mengurangi jumlah data log yang dikirim CloudWatch sehingga mengurangi biaya Anda. Anda dapat menerapkan filter ke log Anda sebelum output ke CloudWatch. Fluentbit mendefinisikan ini di bagian. `[FILTER]` Misalnya, memfilter metadata Kubernetes agar tidak ditambahkan ke peristiwa log dapat mengurangi volume log Anda.

```
    [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.
```

Metrik

Metrik memberikan informasi berharga mengenai kinerja sistem Anda. Dengan mengkonsolidasikan semua metrik sumber daya terkait sistem atau yang tersedia di lokasi terpusat, Anda memperoleh kemampuan untuk membandingkan dan menganalisis data kinerja. Pendekatan terpusat ini memungkinkan Anda untuk membuat keputusan strategis yang lebih tepat, seperti meningkatkan atau mengurangi sumber daya. Selain itu, metrik memainkan peran penting dalam menilai kesehatan sumber daya, memungkinkan Anda untuk mengambil tindakan proaktif bila diperlukan. Umumnya skala biaya observabilitas dengan pengumpulan dan retensi data telemetri. Di bawah ini adalah beberapa strategi yang dapat Anda terapkan untuk mengurangi biaya telemetri metrik: mengumpulkan hanya metrik yang penting, mengurangi kardinalitas data telemetri Anda, dan menyempurnakan perincian pengumpulan data telemetri Anda.

Pantau apa yang penting dan kumpulkan hanya apa yang Anda butuhkan

Strategi pengurangan biaya pertama adalah mengurangi jumlah metrik yang Anda kumpulkan dan pada gilirannya, mengurangi biaya retensi.

1. Mulailah dengan bekerja mundur dari persyaratan Anda dan/atau pemangku kepentingan Anda untuk menentukan metrik yang paling penting. Metrik sukses berbeda untuk semua orang! Ketahui seperti apa penampilan yang baik dan ukur untuk itu.
2. Pertimbangkan untuk menyelam jauh ke dalam beban kerja yang Anda dukung dan identifikasi Indikator Kinerja Utama (KPIs) alias 'Sinyal Emas'. Ini harus selaras dengan persyaratan bisnis dan pemangku kepentingan. Menghitung SLIs, SLOs, dan SLAs menggunakan Amazon CloudWatch dan Metric Math sangat penting untuk mengelola keandalan layanan. Ikuti praktik terbaik yang diuraikan dalam panduan ini untuk memantau dan mempertahankan kinerja lingkungan EKS Anda secara efektif.
3. Kemudian lanjutkan melalui berbagai lapisan infrastruktur untuk menghubungkan dan menghubungkan kluster EKS, node, dan metrik infrastruktur tambahan dengan beban kerja

Anda. KPIs Simpan metrik bisnis dan metrik operasional Anda dalam sistem di mana Anda dapat menghubungkannya bersama-sama dan menarik kesimpulan berdasarkan dampak yang diamati pada keduanya.

4. EKS mengekspos metrik dari bidang kontrol, cluster, pod kube-state-metrics, dan node. Relevansi semua metrik ini tergantung pada kebutuhan Anda, namun kemungkinan Anda tidak memerlukan setiap metrik tunggal di lapisan yang berbeda. Anda dapat menggunakan panduan metrik penting EKS ini sebagai dasar untuk memantau kesehatan keseluruhan cluster EKS dan beban kerja Anda.

Berikut adalah contoh konfigurasi scrape prometheus di mana kita menggunakan `relabel_config` untuk menyimpan hanya metrik kubelet dan untuk menghapus semua metrik container. `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
```

Kurangi kardinalitas jika berlaku

Kardinalitas mengacu pada keunikan nilai data dalam kombinasi dengan dimensinya (misalnya label prometheus) untuk kumpulan metrik tertentu. Metrik kardinalitas tinggi memiliki banyak dimensi dan setiap kombinasi metrik dimensi memiliki keunikan yang lebih tinggi. Kardinalitas yang lebih tinggi menghasilkan ukuran data telemetri metrik yang lebih besar dan kebutuhan penyimpanan yang meningkatkan biaya.

Dalam contoh kardinalitas tinggi di bawah ini, kita melihat bahwa Metrik, Latensi, memiliki Dimensi, RequesTid, CustomerID, dan Layanan dan setiap Dimensi memiliki banyak nilai unik. Kardinalitas adalah ukuran kombinasi jumlah nilai yang mungkin per Dimensi. Dalam Prometheus, setiap set dimensi/label unik dianggap sebagai metrik baru, oleh karena itu kardinalitas tinggi berarti lebih banyak metrik.

Di lingkungan EKS dengan banyak metrik dan dimensi/label per metrik (Cluster, Namespace, Service, Pod, Container, dll), kardinalitas cenderung tumbuh. Untuk mengoptimalkan biaya, pertimbangkan kardinalitas metrik yang Anda kumpulkan dengan hati-hati. Misalnya, jika Anda menggabungkan metrik tertentu untuk visualisasi di tingkat cluster, maka Anda dapat menjatuhkan label tambahan yang berada di lapisan bawah seperti label namespace.

Untuk mengidentifikasi metrik kardinalitas tinggi di prometheus, Anda dapat menjalankan kueri PROMQL berikut untuk menentukan target scrape mana yang memiliki jumlah metrik (kardinalitas) tertinggi:

```
topk_max(5, max_over_time(scrape_samples_scraped[1h]))
```

dan kueri PROMQL berikut dapat membantu Anda menentukan target scrape mana yang memiliki tingkat churn metrik tertinggi (berapa banyak seri metrik baru yang dibuat dalam gesekan tertentu):

```
topk_max(5, max_over_time(scrape_series_added[1h]))
```

Jika Anda menggunakan grafana, Anda dapat menggunakan Mimirtool Grafana Lab untuk menganalisis dasbor grafana dan aturan prometheus Anda untuk mengidentifikasi metrik kardinalitas tinggi yang tidak digunakan. Ikuti panduan ini tentang cara menggunakan `mimirtool analyze prometheus` perintah `mimirtool analyze` dan untuk mengidentifikasi metrik aktif yang tidak direferensikan di dasbor Anda.

Pertimbangkan granularitas metrik

Mengumpulkan metrik pada granularitas yang lebih tinggi seperti setiap detik vs setiap menit dapat berdampak besar pada seberapa banyak telemetri dikumpulkan dan disimpan yang meningkatkan biaya. Tentukan interval pengumpulan gesekan atau metrik yang masuk akal yang menyeimbangkan antara granularitas yang cukup untuk melihat masalah sementara dan cukup rendah agar hemat biaya. Kurangi granularitas untuk metrik yang digunakan untuk perencanaan kapasitas dan analisis jendela waktu yang lebih besar.

Di bawah ini adalah cuplikan dari konfigurasi AWS Distro for Opentelemetry (ADOT) EKS Addon Collector default.

> **Important**
>
> interval gesekan prometheus global diatur ke 15 detik. Interval gesekan ini dapat ditingkatkan sehingga terjadi penurunan jumlah data metrik yang dikumpulkan di 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
```

Pelacakan

Biaya utama yang terkait dengan penelusuran berasal dari generasi penyimpanan jejak. Dengan penelusuran, tujuannya adalah untuk mengumpulkan data yang cukup untuk mendiagnosis dan memahami aspek kinerja. Namun, karena biaya jejak X-Ray didasarkan pada data yang diteruskan ke X-Ray, menghapus jejak setelah diteruskan tidak akan mengurangi biaya Anda. Mari kita tinjau cara untuk menurunkan biaya penelusuran sambil mempertahankan data agar Anda dapat melakukan analisis yang tepat.

Terapkan aturan Sampling

Tingkat pengambilan sampel X-Ray konservatif secara default. Tentukan aturan pengambilan sampel di mana Anda dapat mengontrol jumlah data yang Anda kumpulkan. Ini akan meningkatkan efisiensi kinerja sekaligus mengurangi biaya. Dengan mengurangi laju pengambilan sampel, Anda dapat mengumpulkan jejak dari permintaan hanya apa yang dibutuhkan beban kerja Anda sambil mempertahankan struktur biaya yang lebih rendah.

Misalnya, Anda memiliki aplikasi java yang ingin Anda debug jejak semua permintaan untuk 1 rute bermasalah.

Konfigurasikan melalui SDK untuk memuat aturan pengambilan sampel dari dokumen 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
  }
}
```

Melalui Konsol

Terapkan Tail Sampling dengan AWS Distro for OpenTelemetry (ADOT)

ADOT Tail Sampling memungkinkan Anda untuk mengontrol volume jejak yang tertelan dalam layanan. Namun, Tail Sampling memungkinkan Anda untuk menentukan kebijakan pengambilan sampel setelah semua rentang dalam permintaan selesai, bukan di awal. Ini selanjutnya membatasi jumlah data mentah yang ditransfer CloudWatch, sehingga mengurangi biaya.

Misalnya, jika Anda mengambil sampel 1% lalu lintas ke halaman arahan dan 10% permintaan ke halaman pembayaran, ini mungkin meninggalkan Anda dengan 300 jejak untuk jangka waktu 30 menit. Dengan aturan ADOT Tail Sampling yang menyaring kesalahan tertentu, Anda dapat dibiarkan dengan 200 jejak yang mengurangi jumlah jejak yang disimpan.

```
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]
```

Manfaatkan opsi Penyimpanan Amazon S3

Anda harus memanfaatkan bucket AWS S3 dan kelas penyimpanannya yang berbeda untuk menyimpan jejak. Ekspor jejak ke S3 sebelum periode retensi berakhir. Gunakan aturan Siklus Hidup Amazon S3 untuk memindahkan data jejak ke kelas penyimpanan yang memenuhi persyaratan Anda.

Misalnya, jika Anda memiliki jejak yang berusia 90 hari, Amazon S3 Intelligent-Tiering dapat secara otomatis memindahkan data ke penyimpanan jangka panjang berdasarkan pola penggunaan Anda. Anda dapat menggunakan Amazon Athena untuk menanyakan data di Amazon S3 jika Anda perlu merujuk kembali ke jejak di lain waktu. Ini selanjutnya dapat mengurangi biaya Anda untuk penelusuran terdistribusi.

Sumber Daya Tambahan:

- Panduan Praktik Terbaik Observabilitas

- Koleksi Metrik Praktik Terbaik
- AWS re:invent 2022 - Praktik terbaik observabilitas di Amazon () COP343
- AWS re:invent 2022 - Observabilitas: Praktik terbaik untuk aplikasi modern () COP344

Praktik Terbaik untuk Windows

Panduan ini memberikan saran tentang menjalankan wadah dan node windows.

Topik

- Amazon EKS mengoptimalkan manajemen AMI Windows
- Konfigurasikan GMSA untuk Pod dan kontainer Windows
- Pengerasan node pekerja Windows
- Pemindaian gambar kontainer
- Versi dan Lisensi Windows Server
- Pencatatan log
- Memantau
- Jaringan Windows
- Menghindari kesalahan OOM
- Menambal Server dan Wadah Windows
- Menjalankan beban kerja Heterogen
- Konteks Keamanan Pod
- Opsi penyimpanan persisten
- Pengerasan gambar wadah Windows

Amazon EKS mengoptimalkan manajemen AMI Windows

Windows Amazon EKS AMIs yang dioptimalkan dibangun di atas Windows Server 2019 dan Windows Server 2022. Mereka dikonfigurasi untuk berfungsi sebagai gambar dasar untuk node Amazon EKS. Secara default, AMIs termasuk komponen-komponen berikut:

- kubelet
- kube-proxy
- AWS IAM Authenticator untuk Kubernetes
- csi-proxy
- containerd

Anda dapat mengambil ID Amazon Machine Image (AMI) secara terprogram untuk Amazon EKS yang dioptimalkan AMIs dengan menanyakan AWS Systems Manager Parameter Store API. Parameter ini menghilangkan kebutuhan bagi Anda untuk secara manual mencari AMI Amazon EKS yang dioptimalkan IDs. Untuk informasi selengkapnya tentang Systems Manager Parameter Store API, lihat GetParameter. Akun pengguna Anda harus memiliki izin ssm: GetParameter IAM untuk mengambil metadata AMI Amazon EKS yang dioptimalkan.

Contoh berikut mengambil ID AMI untuk AMI Amazon EKS terbaru yang dioptimalkan untuk Windows Server 2019 LTSC Core. Nomor versi yang tercantum dalam nama AMI berhubungan dengan build Kubernetes yang sesuai yang disiapkan untuknya.

```
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
```

Contoh output:

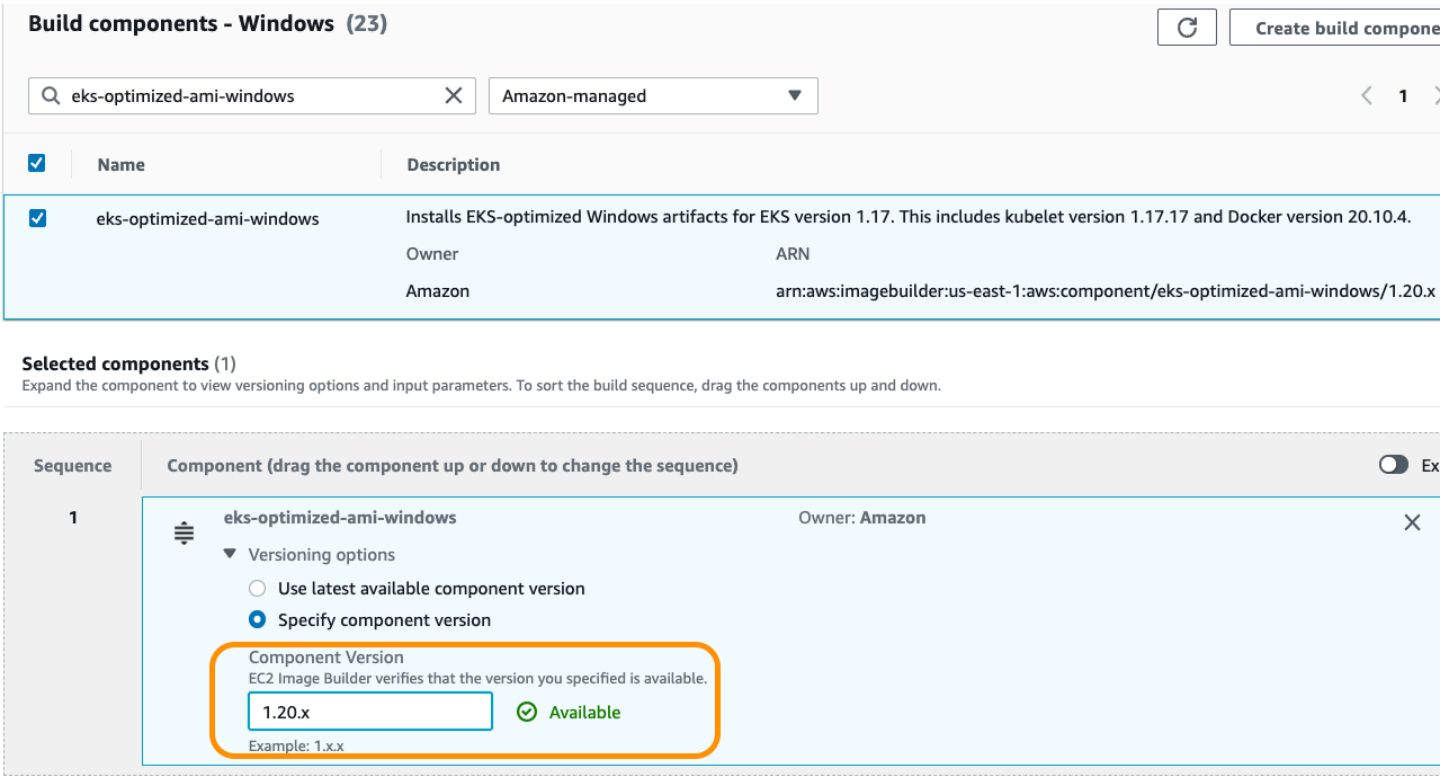
```
ami-09770b3eec4552d4e
```

Mengelola Amazon EKS Anda sendiri yang dioptimalkan Windows AMI

Langkah penting menuju lingkungan produksi adalah mempertahankan versi Windows AMI dan kubelet yang dioptimalkan Amazon EKS yang sama di seluruh kluster Amazon EKS.

Menggunakan versi yang sama di seluruh kluster Amazon EKS mengurangi waktu selama pemecahan masalah dan meningkatkan konsistensi klaster. Amazon EC2 Image Builder membantu membuat dan memelihara Windows yang dioptimalkan Amazon EKS khusus AMIs untuk digunakan di seluruh kluster Amazon EKS.

Gunakan Amazon EC2 Image Builder untuk memilih antara versi Windows Server, tanggal rilis AWS Windows Server AMI, versi build and/or OS. Langkah membangun komponen, memungkinkan Anda untuk memilih antara Artefak Windows yang Dioptimalkan EKS yang ada serta versi kubelet. Untuk informasi lebih lanjut: https://docs.aws.amazon.com/eks/ latest/userguide/eks - custom-ami-windows .html



CATATAN: Sebelum memilih gambar dasar, lihat bagian Versi dan Lisensi Windows Server untuk detail penting yang berkaitan dengan pembaruan saluran rilis.

Mengkonfigurasi peluncuran yang lebih cepat untuk EKS khusus yang dioptimalkan AMIs

Saat menggunakan AMI yang dioptimalkan Windows Amazon EKS khusus, node pekerja Windows dapat diluncurkan hingga 65% lebih cepat dengan mengaktifkan fitur Peluncuran Cepat. Fitur ini mempertahankan serangkaian snapshot yang telah disediakan sebelumnya yang memiliki spesialisasi Sysprep, langkah-langkah Windows Out of Box Experience (OOBE) dan reboot yang diperlukan sudah selesai. Snapshot ini kemudian digunakan pada peluncuran berikutnya, mengurangi waktu untuk scale-out atau mengganti node. Peluncuran Cepat hanya dapat diaktifkan untuk AMIs Anda sendiri melalui konsol EC2 atau di AWS CLI dan jumlah snapshot yang dipertahankan dapat dikonfigurasi.

CATATAN: Peluncuran Cepat tidak kompatibel dengan AMI yang dioptimalkan EKS yang disediakan Amazon default, buat AMI khusus seperti di atas sebelum mencoba mengaktifkannya.

Untuk informasi lebih lanjut: AWS Windows AMIs - Konfigurasikan AMI Anda untuk peluncuran yang lebih cepat

Caching lapisan dasar Windows pada kustom AMIs

Gambar kontainer Windows lebih besar dari rekan-rekan Linux mereka. Jika Anda menjalankan aplikasi.NET Framework berbasis container, ukuran gambar rata-rata adalah sekitar 8.24GB. Selama penjadwalan pod, image kontainer harus sepenuhnya ditarik dan diekstraksi dalam disk sebelum pod mencapai status Running.

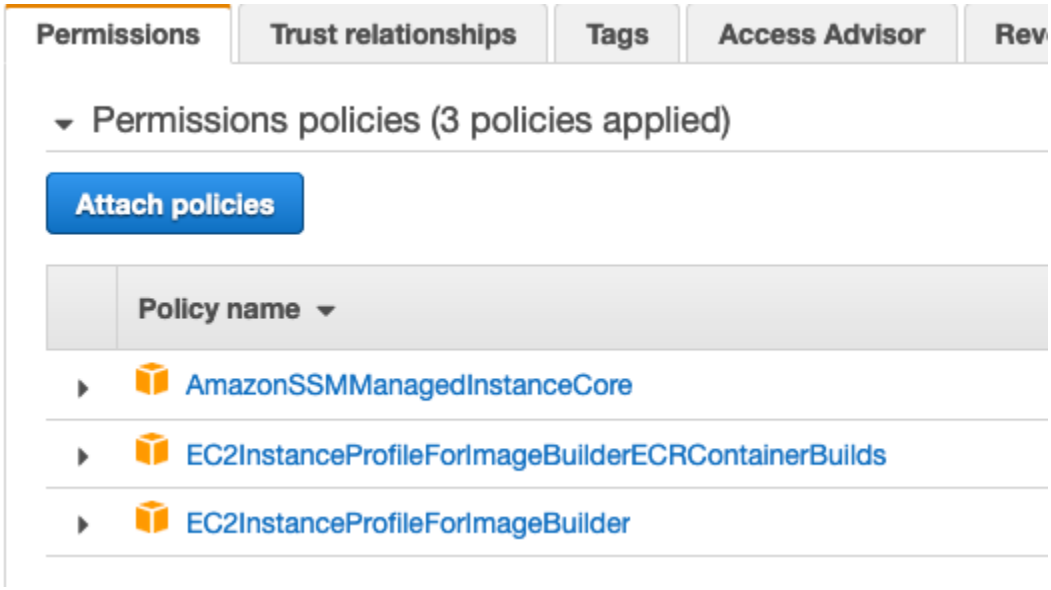
Selama proses ini, runtime container (containerd) menarik dan mengekstrak seluruh image kontainer dalam disk. Operasi pull adalah proses paralel, artinya runtime container menarik layer image container secara paralel. Sebaliknya, operasi ekstraksi terjadi dalam proses berurutan, dan I/O intensif. Karena itu, image container dapat memakan waktu lebih dari 8 menit untuk sepenuhnya diekstraksi dan siap digunakan oleh runtime container (containerd), dan sebagai hasilnya, waktu startup pod dapat memakan waktu beberapa menit.

Seperti disebutkan dalam topik Patching Windows Server dan Container, ada opsi untuk membangun AMI khusus dengan EKS. Selama persiapan AMI, Anda dapat menambahkan komponen pembuat Gambar EC2 tambahan untuk menarik semua gambar kontainer Windows yang diperlukan secara lokal dan kemudian menghasilkan AMI. Strategi ini akan secara drastis mengurangi waktu pod mencapai status Running.

Di Amazon EC2 Image Builder, buat komponen untuk mengunduh gambar yang diperlukan dan melampirkannya ke resep Gambar. Contoh berikut menarik gambar tertentu dari repositori 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
```

Untuk memastikan komponen berikut berfungsi seperti yang diharapkan, periksa apakah peran IAM yang digunakan oleh EC2 Image builder (EC2InstanceProfileForImageBuilder) memiliki kebijakan terlampir:



Posting blog

Dalam posting blog berikut, Anda akan menemukan langkah demi langkah tentang cara menerapkan strategi caching untuk kustom Amazon EKS Windows AMIs:

Mempercepat waktu peluncuran kontainer Windows dengan pembuat Gambar EC2 dan strategi cache gambar

Konfigurasikan GMSA untuk Pod dan kontainer Windows

Apa itu akun GMSA

Aplikasi berbasis Windows seperti aplikasi.NET sering menggunakan Active Directory sebagai penyedia identitas, menyediakan authorization/authentication menggunakan protokol NTLM atau Kerberos.

Server aplikasi untuk menukar tiket Kerberos dengan Active Directory harus bergabung dengan domain. Wadah Windows tidak mendukung gabungan domain dan tidak masuk akal karena wadah adalah sumber daya sementara, menciptakan beban pada kumpulan RID Direktori Aktif.

Namun, administrator dapat memanfaatkan akun GMSA Active Directory untuk menegosiasikan otentikasi Windows untuk sumber daya seperti wadah Windows, NLB, dan server farm.

Wadah Windows dan kasus penggunaan GMSA

Aplikasi yang memanfaatkan otentikasi Windows, dan berjalan sebagai wadah Windows, mendapat manfaat dari GMSA karena Windows Node digunakan untuk menukar tiket Kerberos atas nama kontainer.Ada dua opsi yang tersedia untuk mengatur node pekerja Windows untuk mendukung integrasi GMSA:

1 - Node pekerja Windows yang bergabung dengan domain

Dalam pengaturan ini, node pekerja Windows bergabung dengan domain di domain Active Directory, dan akun AD Computer dari node pekerja Windows digunakan untuk mengautentikasi terhadap Active Directory dan mengambil identitas GMSA yang akan digunakan dengan pod.

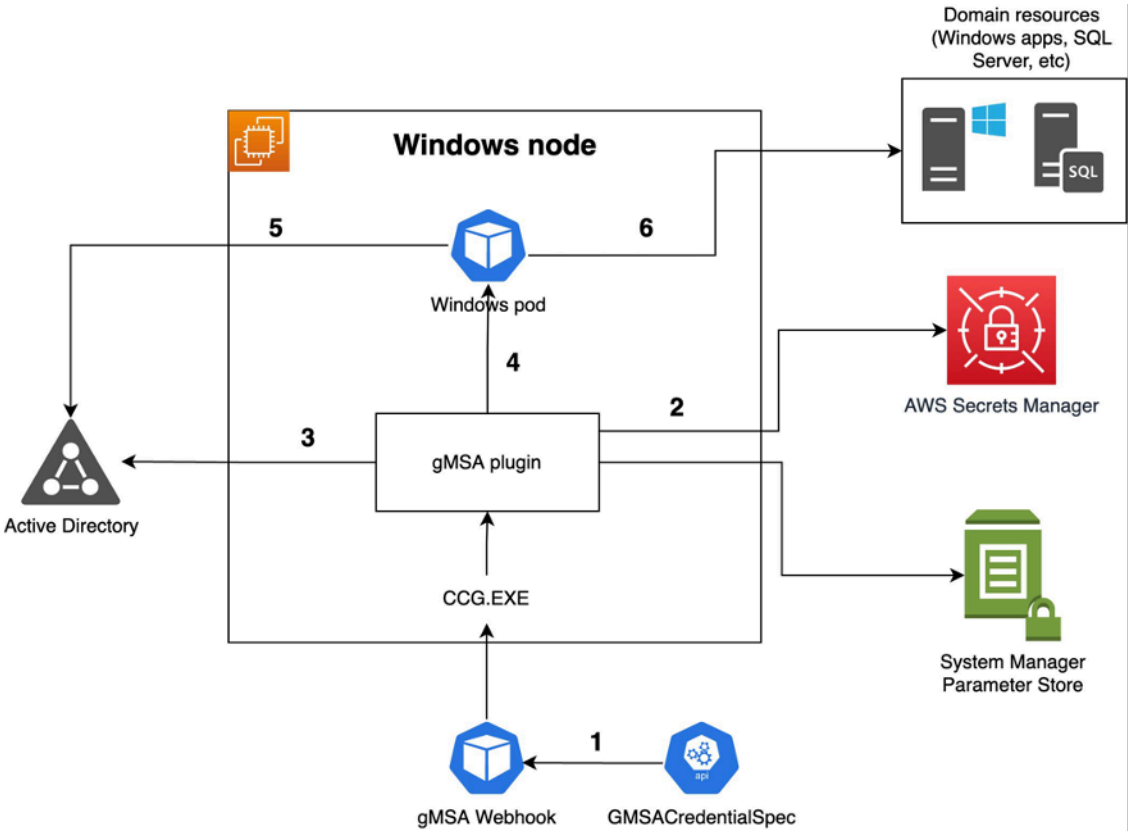
Dalam pendekatan domain-joined, Anda dapat dengan mudah mengelola dan mengeraskan node pekerja Windows Anda menggunakan Active Directory yang ada GPOs; namun, ini menghasilkan overhead operasional tambahan dan penundaan selama node pekerja Windows bergabung di cluster Kubernetes, karena memerlukan reboot tambahan selama startup node dan pembersihan garasi Active Directory setelah cluster Kubernetes mengakhiri node.

Dalam posting blog berikut, Anda akan menemukan detail step-by-step tentang cara menerapkan pendekatan node pekerja Windows yang bergabung dengan Domain:

Otentikasi Windows di Amazon EKS Pod Windows

2 - Node pekerja Windows tanpa domain

Dalam pengaturan ini, node pekerja Windows tidak bergabung dalam domain Active Directory, dan identitas “portabel” (pengguna/kata sandi) digunakan untuk mengautentikasi terhadap Active Directory dan mengambil identitas GMSA yang akan digunakan dengan pod.



Identitas portabel adalah pengguna Active Directory; identitas (pengguna/kata sandi) disimpan di AWS Secrets Manager atau AWS System Manager Parameter Store, dan plugin yang dikembangkan AWS bernama ccg_plugin akan digunakan untuk mengambil identitas ini dari AWS Secrets Manager atau AWS System Manager Parameter Store dan meneruskannya ke containerd untuk mengambil identitas GMSA dan membuatnya tersedia untuk pod.

Dalam pendekatan tanpa domain ini, Anda bisa mendapatkan keuntungan dari tidak memiliki interaksi Active Directory selama startup node pekerja Windows saat menggunakan GMSA dan mengurangi overhead operasional untuk administrator Active Directory.

Dalam posting blog berikut, Anda akan menemukan detail step-by-step tentang cara menerapkan pendekatan node pekerja Windows Domainless:

Otentikasi Windows Tanpa Domain untuk Pod Windows Amazon EKS

Catatan penting

Meskipun pod dapat menggunakan akun GMSA, perlu juga mengatur aplikasi atau layanan yang sesuai untuk mendukung otentikasi Windows, misalnya, untuk mengatur Microsoft IIS untuk mendukung otentikasi Windows, Anda harus menyiapkannya melalui 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'
```

Pengerasan node pekerja Windows

OS Hardening adalah kombinasi dari konfigurasi OS, menambal, dan menghapus paket perangkat lunak yang tidak perlu, yang bertujuan untuk mengunci sistem dan mengurangi permukaan serangan. Ini adalah praktik terbaik untuk mempersiapkan EKS Optimized Windows AMI Anda sendiri dengan konfigurasi pengerasan yang diperlukan oleh perusahaan Anda.

AWS menyediakan EKS Optimized Windows AMI baru setiap bulan yang berisi Patch Keamanan Windows Server terbaru. Namun, masih menjadi tanggung jawab pengguna untuk mengeraskan AMI mereka dengan menerapkan konfigurasi OS yang diperlukan terlepas dari apakah mereka menggunakan grup node yang dikelola sendiri atau dikelola.

Microsoft menawarkan berbagai alat seperti Microsoft Security Compliance Toolkit dan Security Baselines yang membantu Anda mencapai pengerasan berdasarkan kebutuhan kebijakan keamanan Anda. Tolok Ukur CIS juga tersedia dan harus diimplementasikan di atas Amazon EKS Dioptimalkan Windows AMI untuk lingkungan produksi.

Mengurangi permukaan serangan dengan Windows Server Core

Windows Server Core adalah opsi instalasi minimal yang tersedia sebagai bagian dari EKS Optimized Windows AMI. Menyebarkan Windows Server Core memiliki beberapa manfaat. Pertama, ia memiliki jejak disk yang relatif kecil, menjadi 6GB pada Server Core terhadap 10GB pada Windows Server dengan pengalaman Desktop. Kedua, ia memiliki permukaan serangan yang lebih kecil karena basis kode yang lebih kecil dan tersedia APIs.

AWS memberi pelanggan Windows Amazon EKS Optimized baru AMIs setiap bulan, berisi patch keamanan Microsoft terbaru, terlepas dari versi yang didukung Amazon EKS. Sebagai praktik terbaik, node pekerja Windows harus diganti dengan yang baru berdasarkan AMI terbaru yang dioptimalkan

Amazon EKS. Setiap node yang berjalan selama lebih dari 45 hari tanpa pembaruan di tempat atau penggantian node tidak memiliki praktik terbaik keamanan.

Menghindari koneksi RDP

Remote Desktop Protocol (RDP) adalah protokol koneksi yang dikembangkan oleh Microsoft untuk menyediakan pengguna dengan antarmuka grafis untuk terhubung ke komputer Windows lain melalui jaringan.

Sebagai praktik terbaik, Anda harus memperlakukan node pekerja Windows Anda seolah-olah mereka adalah host sementara. Itu berarti tidak ada koneksi manajemen, tidak ada pembaruan, dan tidak ada pemecahan masalah. Setiap modifikasi dan pembaruan harus diimplementasikan sebagai AMI kustom baru dan diganti dengan memperbarui grup Auto Scaling. Lihat Menambal Server dan Kontainer Windows dan pengelolaan AMI Windows yang dioptimalkan Amazon EKS.

Nonaktifkan koneksi RDP pada node Windows selama penyebaran dengan meneruskan nilai false pada properti ssh, seperti contoh di bawah ini:

```
nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
```

Jika akses ke node Windows diperlukan, gunakan AWS System Manager Session Manager untuk membuat PowerShell sesi aman melalui AWS Console dan agen SSM. Untuk melihat cara menerapkan solusi, tonton Akses Instans Windows dengan Aman Menggunakan AWS Systems Manager Session Manager

Untuk menggunakan Manajer Sesi Manajer Sistem, kebijakan IAM tambahan harus diterapkan pada peran IAM yang digunakan untuk meluncurkan node pekerja Windows. Di bawah ini adalah contoh di mana Amazon SSMManaged InstanceCore ditentukan dalam manifes `eksctl` cluster:

```
 nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
```

```
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
  iam:
    attachPolicyARNs:
      - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
      - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
      - arn:aws:iam::aws:policy/ElasticLoadBalancingFullAccess
      - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
      - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
```

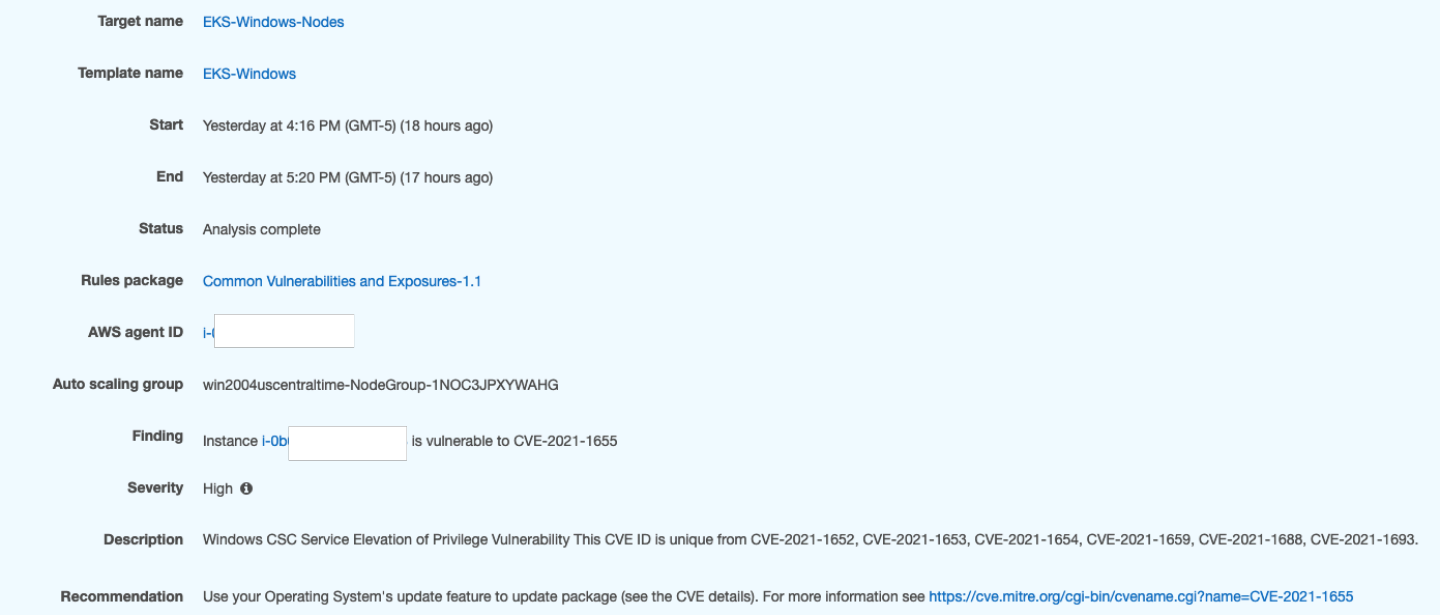
Amazon Inspector

> Amazon Inspector adalah layanan penilaian keamanan otomatis yang membantu meningkatkan keamanan dan kepatuhan aplikasi yang diterapkan di AWS. Amazon Inspector secara otomatis menilai aplikasi untuk eksposur, kerentanan, dan penyimpangan dari praktik terbaik. Setelah melakukan penilaian, Amazon Inspector menghasilkan daftar detail temuan keamanan yang diprioritaskan berdasarkan tingkat keparahan. Temuan ini dapat ditinjau secara langsung atau sebagai bagian dari laporan penilaian terperinci yang tersedia melalui konsol Amazon Inspector atau API.

Amazon Inspector dapat digunakan untuk menjalankan penilaian CIS Benchmark pada node pekerja Windows dan dapat diinstal pada Windows Server Core dengan melakukan tugas-tugas berikut:

1. Unduh file.exe berikut: .exe https://inspector-agent.amazonaws.com/windows/ installer/latest/ AWSAgentInstall
2. Transfer agen ke node pekerja Windows.
3. Jalankan perintah berikut PowerShell untuk menginstal Amazon Inspector Agent: `. \AWSAgentInstall.exe /install`

Di bawah ini adalah ouput setelah lari pertama. Seperti yang Anda lihat, itu menghasilkan temuan berdasarkan database CVE. Anda dapat menggunakan ini untuk mengeraskan node Worker atau membuat AMI berdasarkan konfigurasi yang diperkeras.



Untuk informasi selengkapnya tentang Amazon Inspector, termasuk cara menginstal agen Amazon Inspector, menyiapkan penilaian CIS Benchmark, dan membuat laporan, tonton video Meningkatkan keamanan dan kepatuhan Windows Workloads with Amazon Inspector.

Amazon GuardDuty

> Amazon GuardDuty adalah layanan deteksi ancaman yang terus memantau aktivitas berbahaya dan perilaku tidak sah untuk melindungi akun AWS, beban kerja, dan data yang disimpan di Amazon S3 Anda. Dengan cloud, pengumpulan dan agregasi aktivitas akun dan jaringan disederhanakan, tetapi dapat memakan waktu bagi tim keamanan untuk terus menganalisis data log peristiwa untuk potensi ancaman.

Dengan menggunakan Amazon, GuardDuty Anda memiliki visilitiby pada tindakan berbahaya terhadap node pekerja Windows, seperti serangan RDP brute force dan Port Probe.

Tonton Deteksi Ancaman untuk Beban Kerja Windows menggunakan GuardDuty video Amazon untuk mempelajari cara menerapkan dan menjalankan Tolok Ukur CIS pada EKS Windows AMI yang Dioptimalkan

Keamanan di Amazon EC2 untuk Windows

Baca tentang praktik terbaik Keamanan untuk instans Windows Amazon EC2 untuk menerapkan kontrol keamanan di setiap lapisan.

Pemindaian gambar kontainer

Pemindaian Gambar adalah fitur penilaian kerentanan otomatis yang membantu meningkatkan keamanan gambar kontainer aplikasi Anda dengan memindai mereka untuk berbagai kerentanan sistem operasi.

Saat ini, Amazon Elastic Container Registry (ECR) hanya dapat memindai image container Linux untuk mencari kerentanan. Namun; ada alat pihak ketiga yang dapat diintegrasikan dengan CI/CD pipeline yang ada untuk pemindaian gambar kontainer Windows.

- Jangkar
- PaloAlto Awan Prisma
- Trend Micro - Pemeriksaan Cerdas Keamanan Dalam

Untuk mempelajari lebih lanjut tentang cara mengintegrasikan solusi ini dengan Amazon Elastic Container Repository (ECR), periksa:

- Jangkar, memindai gambar di Amazon Elastic Container Registry (ECR)
- PaloAlto, memindai gambar di Amazon Elastic Container Registry (ECR)
- TrendMicro, memindai gambar di Amazon Elastic Container Registry (ECR)

Versi dan Lisensi Windows Server

Versi Windows Server

AMI Windows Amazon EKS yang Dioptimalkan didasarkan pada edisi Pusat Data Windows Server 2019 dan 2022 di Saluran Layanan Jangka Panjang (LTSC). Versi Datacenter tidak memiliki batasan pada jumlah kontainer yang berjalan pada node pekerja. Untuk informasi lebih lanjut: https://docs.microsoft.com/en-us/ virtualization/windowscontainers/about/faq

Saluran Servis Jangka Panjang (LTSC)

Sebelumnya disebut “Long-Term Servicing Branch”, ini adalah model rilis yang sudah Anda kenal, di mana versi utama baru Windows Server dirilis setiap 2-3 tahun. Pengguna berhak atas 5 tahun dukungan arus utama dan 5 tahun dukungan diperpanjang.

Lisensi

Saat meluncurkan instans Amazon EC2 dengan AMI berbasis Windows Server, Amazon menanggung biaya lisensi dan kepatuhan lisensi untuk Anda.

Pencatatan log

Aplikasi kontainer biasanya mengarahkan log aplikasi ke STDOUT. Runtime kontainer menjebak log ini dan melakukan sesuatu dengannya - biasanya menulis ke file. Di mana file-file ini disimpan tergantung pada runtime dan konfigurasi kontainer.

Salah satu perbedaan mendasar dengan pod Windows adalah mereka tidak menghasilkan STDOUT. Anda dapat menjalankan LogMonitoruntuk mengambil ETW (Event Tracing for Windows), Windows Event Logs dan log khusus aplikasi lainnya dari menjalankan wadah Windows dan pipa output log yang diformat ke STDOUT. Log ini kemudian dapat dialirkan menggunakan fluent-bit atau fluentd ke tujuan yang Anda inginkan seperti Amazon. CloudWatch

Mekanisme pengumpulan Log mengambil STDOUT/STDERR log dari pod Kubernetes. A DaemonSetadalah cara umum untuk mengumpulkan log dari wadah. Ini memberi Anda kemampuan untuk mengelola log routing/filtering/enrichment secara independen dari aplikasi. Fluentd DaemonSet dapat digunakan untuk mengalirkan log ini dan log yang dihasilkan aplikasi lainnya ke agregator log yang diinginkan.

Informasi lebih rinci tentang streaming log dari beban kerja Windows ke CloudWatch dijelaskan di sini

Rekomendasi Pencatatan

Praktik terbaik pencatatan umum tidak berbeda saat mengoperasikan beban kerja Windows di Kubernetes.

- Selalu log entri log terstruktur (JSON/SYSLOG) yang membuat penanganan entri log lebih mudah karena ada banyak parser pra-tertulis untuk format terstruktur tersebut.
- Sentralisasi log - wadah logging khusus dapat digunakan secara khusus untuk mengumpulkan dan meneruskan pesan log dari semua kontainer ke tujuan
- Pertahankan verbositas log kecuali saat men-debug. Verbositas menempatkan banyak stres pada infrastruktur penebangan dan peristiwa penting dapat hilang dalam kebisingan.
- Selalu catat informasi aplikasi bersama dengan id transaksi/permintaan untuk keterlacakan. Objek Kubernetes tidak membawa nama aplikasi, jadi misalnya nama pod `windows-twryrqyw`

mungkin tidak memiliki arti apa pun saat men-debug log. Ini membantu dengan ketertelusuran dan pemecahan masalah aplikasi dengan log agregat Anda.

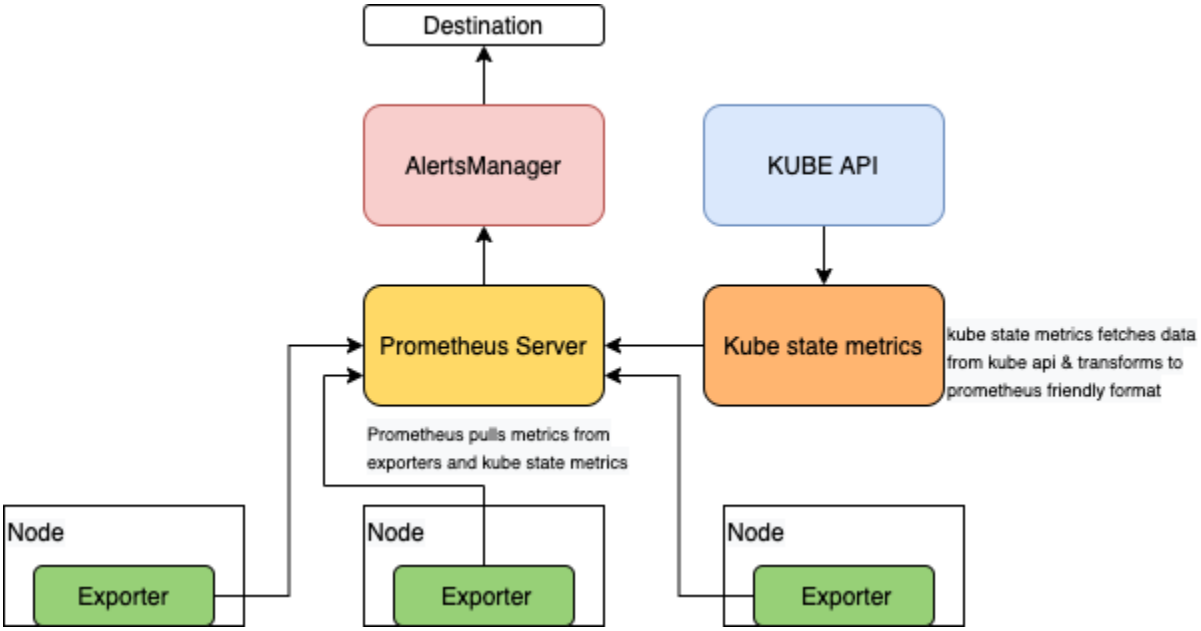
Bagaimana Anda menghasilkan transaction/correlation id ini tergantung pada konstruksi pemrograman. Tetapi pola yang sangat umum adalah menggunakan Aspect/Interceptor logging, yang dapat menggunakan MDC (Mapped diagnostic context) untuk menyuntikkan transaction/correlation id unik ke setiap permintaan yang masuk, seperti:

```
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();
    }
}
```

Memantau

Prometheus, proyek CNCF yang lulus sejauh ini merupakan sistem pemantauan paling populer dengan integrasi asli ke Kubernetes. Prometheus mengumpulkan metrik di sekitar kontainer, pod, node, dan cluster. Selain itu, Prometheus AlertsManager memanfaatkan yang memungkinkan Anda memprogram peringatan untuk memperingatkan Anda jika ada sesuatu di cluster Anda yang salah. Prometheus menyimpan data metrik sebagai data deret waktu yang diidentifikasi dengan nama metrik dan pasangan. key/value Prometheus menyertakan away to query menggunakan bahasa yang disebut PromQL, yang merupakan kependekan dari Prometheus Query Language.

Arsitektur tingkat tinggi dari koleksi metrik Prometheus ditunjukkan di bawah ini:



Prometheus menggunakan mekanisme tarik dan mengorek metrik dari target menggunakan eksportir dan dari Kubernetes API menggunakan metrik status kube. Ini berarti aplikasi dan layanan harus mengekspos titik akhir HTTP (S) yang berisi metrik berformat Prometheus. Prometheus kemudian akan, sesuai konfigurasinya, secara berkala menarik metrik dari titik akhir HTTP (S) ini.

Eksportir memungkinkan Anda menggunakan metrik pihak ketiga sebagai metrik berformat Prometheus. Eksportir Prometheus biasanya digunakan pada setiap node. Untuk daftar eksportir lengkap, silakan merujuk ke eksportir Prometheus. Sementara pengekspor node cocok untuk mengekspor perangkat keras host dan metrik OS untuk node linux, itu tidak akan berfungsi untuk node Windows.

Dalam cluster EKS node campuran dengan node Windows saat Anda menggunakan bagan helm Prometheus yang stabil, Anda akan melihat pod yang gagal pada node Windows, karena eksportir ini tidak ditujukan untuk Windows. Anda harus memperlakukan kumpulan pekerja Windows secara terpisah dan sebagai gantinya menginstal eksportir Windows pada grup node pekerja Windows.

Untuk mengatur pemantauan Prometheus untuk node Windows, Anda perlu mengunduh dan menginstal eksportir WMI di server Windows itu sendiri dan kemudian mengatur target di dalam konfigurasi scrape file konfigurasi Prometheus. Halaman rilis menyediakan semua installer.msi yang tersedia, dengan set fitur masing-masing dan perbaikan bug. Pemasang akan mengatur windows_exporter sebagai layanan Windows, serta membuat pengecualian di firewall Windows. Jika penginstal dijalankan tanpa parameter apa pun, eksportir akan berjalan dengan pengaturan default untuk kolektor, port, dll yang diaktifkan.

Anda dapat memeriksa bagian praktik terbaik penjadwalan dari panduan ini yang menyarankan penggunaan taints/tolerations atau RuntimeClass untuk menyebarkan eksportir node secara selektif hanya ke node linux, sedangkan eksportir Windows diinstal pada node Windows saat Anda mem-bootstrap node atau menggunakan alat manajemen konfigurasi pilihan Anda (contoh chef, Ansible, SSM dll).

Perhatikan bahwa, tidak seperti node linux di mana pengekspor node diinstal sebagai daemonset, pada node Windows eksportir WMI diinstal pada host itu sendiri. Eksportir akan mengekspor metrik seperti penggunaan CPU, memori dan I/O penggunaan disk dan juga dapat digunakan untuk memantau situs dan aplikasi IIS, antarmuka jaringan dan layanan.

Windows_exporter akan mengekspos semua metrik dari kolektor yang diaktifkan secara default. Ini adalah cara yang disarankan untuk mengumpulkan metrik untuk menghindari kesalahan. Namun, untuk penggunaan lanjutan windows_exporter dapat diteruskan daftar opsional kolektor untuk memfilter metrik. Parameter collect [], dalam konfigurasi Prometheus memungkinkan Anda melakukannya.

Langkah-langkah penginstalan default untuk Windows termasuk mengunduh dan memulai eksportir sebagai layanan selama proses bootstrap dengan argumen, seperti kolektor yang ingin Anda filter.

```
> 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"
```

Secara default, metrik dapat dikikis di titik akhir /metrics pada port 9182. Pada titik ini, Prometheus dapat menggunakan metrik dengan menambahkan scrape_config berikut ke konfigurasi 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', ...]
```

Konfigurasi Prometheus dimuat ulang menggunakan

```
> ps aux | grep prometheus
> kill HUP <PID>
```

Cara yang lebih baik dan direkomendasikan untuk menambahkan target adalah dengan menggunakan Definisi Sumber Daya Kustom yang disebut ServiceMonitor, yang datang sebagai bagian dari operator Prometheus] yang memberikan definisi untuk ServiceMonitor Objek dan pengontrol yang akan mengaktifkan ServiceMonitors kita mendefinisikan dan secara otomatis membangun konfigurasi Prometheus yang diperlukan.

The ServiceMonitor, yang secara deklaratif menentukan bagaimana kelompok layanan Kubernetes harus dipantau, digunakan untuk mendefinisikan aplikasi yang ingin Anda kikis metrik dari dalam Kubernetes. Di dalam ServiceMonitor kita menentukan label Kubernetes yang dapat digunakan operator untuk mengidentifikasi Layanan Kubernetes yang pada gilirannya mengidentifikasi Pod, yang ingin kita pantau.

Untuk memanfaatkan ServiceMonitor, buat objek Endpoint yang menunjuk ke target Windows tertentu, layanan tanpa kepala, dan ServiceMontor untuk node 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
```

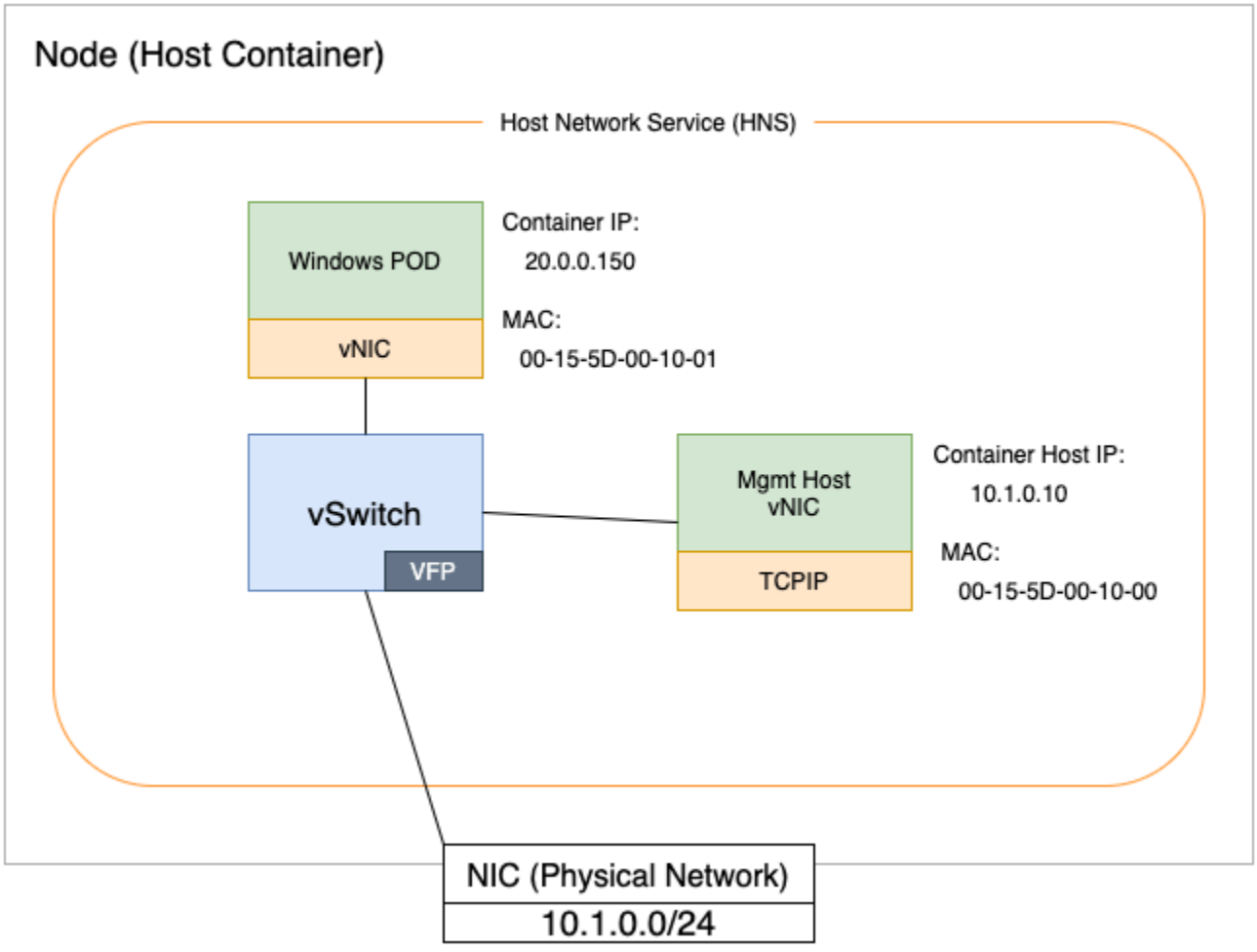
```
    port: 9182
    protocol: TCP

---
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
```

Untuk detail lebih lanjut tentang operator dan penggunaan ServiceMonitor, periksa dokumentasi operator resmi. Perhatikan bahwa Prometheus memang mendukung penemuan target dinamis menggunakan banyak opsi penemuan layanan.

Jaringan Windows

Ikhtisar Jaringan Kontainer Windows

Kontainer Windows pada dasarnya berbeda dari wadah Linux. Kontainer Linux menggunakan konstruksi Linux seperti namespace, sistem file union, dan cgroups. Pada Windows, konstruksi tersebut diabstraksikan dari containerd oleh Host Compute Service (HCS). HCS bertindak sebagai lapisan API yang berada di atas implementasi container pada Windows. Windows container juga memanfaatkan Host Network Service (HNS) yang mendefinisikan topologi jaringan pada sebuah node.



Dari perspektif jaringan, HCS dan HNS membuat wadah Windows berfungsi seperti mesin virtual. Misalnya, setiap kontainer memiliki adaptor jaringan virtual (vNIC) yang terhubung ke sakelar virtual Hyper-V (vSwitch) seperti yang ditunjukkan pada diagram di atas.

Manajemen Alamat IP

Sebuah node di Amazon EKS menggunakan Elastic Network Interface (ENI) untuk terhubung ke jaringan AWS VPC. Saat ini, hanya satu ENI per node pekerja Windows yang didukung. Manajemen alamat IP untuk node Windows dilakukan oleh VPC Resource Controller yang berjalan di bidang kontrol. Rincian lebih lanjut tentang alur kerja untuk manajemen alamat IP node Windows dapat ditemukan di sini.

Jumlah pod yang dapat didukung oleh node pekerja Windows ditentukan oleh ukuran node dan jumlah IPv4 alamat yang tersedia. Anda dapat menghitung IPv4 alamat yang tersedia pada node seperti di bawah ini:

- Secara default, hanya IPv4 alamat sekunder yang ditetapkan ke ENI. Dalam kasus seperti itu:

  ```
  Total IPv4 addresses available for Pods = Number of supported IPv4 addresses in the
   primary interface - 1
  ```

  Kami mengurangi satu dari jumlah total karena satu IPv4 alamat akan digunakan sebagai alamat utama ENI dan karenanya tidak dapat dialokasikan ke Pod.
- Jika cluster telah dikonfigurasi untuk kepadatan pod yang tinggi dengan mengaktifkan fitur delegasi awalan maka-

  ```
  Total IPv4 addresses available for Pods = (Number of supported IPv4 addresses in the
   primary interface - 1) * 16
  ```

  Di sini, alih-alih mengalokasikan IPv4 alamat sekunder, VPC Resource Controller akan mengalokasikan `/28 prefixes` dan oleh karena itu, jumlah keseluruhan IPv4 alamat yang tersedia akan ditingkatkan 16 kali.

Dengan menggunakan rumus di atas, kita dapat menghitung pod maks untuk pekerja Windows yang mengangguk berdasarkan instance m5.large seperti di bawah ini:

- Secara default, saat berjalan dalam mode IP sekunder-

```
10 secondary IPv4 addresses per ENI - 1 = 9 available IPv4 addresses
```

- Saat menggunakan `prefix delegation` -

```
(10 secondary IPv4 addresses per ENI - 1) * 16 = 144 available IPv4 addresses
```

Untuk informasi selengkapnya tentang berapa banyak alamat IP yang dapat didukung oleh tipe instans, lihat alamat IP per antarmuka jaringan per jenis instans.

Pertimbangan utama lainnya adalah arus lalu lintas jaringan. Dengan Windows ada risiko kelelahan port pada node dengan lebih dari 100 layanan. Ketika kondisi ini muncul, node akan mulai melempar kesalahan dengan pesan berikut:

“Pembuatan kebijakan gagal: hcnCreateLoad Penyeimbang gagal di Win32: Port yang ditentukan sudah ada.”

Untuk mengatasi masalah ini, kami memanfaatkan Direct Server Return (DSR). DSR adalah implementasi distribusi beban jaringan asimetris. Dengan kata lain, lalu lintas permintaan dan respons menggunakan jalur jaringan yang berbeda. Fitur ini mempercepat komunikasi antar pod dan mengurangi risiko kelelahan port. Oleh karena itu kami merekomendasikan untuk mengaktifkan DSR pada node Windows.

DSR diaktifkan secara default di Windows Server SAC EKS Dioptimalkan. AMIs Untuk Windows Server 2019 LTSC EKS Optimized AMIs, Anda harus mengaktifkannya selama penyediaan instance menggunakan skrip di bawah ini dan dengan menggunakan Windows Server 2019 Full atau Core sebagai AmiFamily di NodeGroup. `eksctl` Lihat eksctl custom AMI untuk informasi tambahan.

```
nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
```

Untuk memanfaatkan DSR di Windows Server 2019 dan yang lebih baru, Anda harus menentukan flag kube-proxy berikut selama startup instance. Anda dapat melakukan ini dengan menyesuaikan skrip userdata yang terkait dengan grup node yang dikelola sendiri Launch Template.

```
<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>
```

Pengaktifan DSR dapat diverifikasi mengikuti instruksi di blog Microsoft Networking dan Windows Containers di 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}
```

Jika menjaga IPv4 alamat yang tersedia dan meminimalkan pemborosan sangat penting untuk subnet Anda, umumnya disarankan untuk menghindari penggunaan mode delegasi awalan seperti

yang disebutkan dalam Mode Awalan untuk Windows - Kapan harus dihindari. Jika menggunakan delegasi awalan masih diinginkan, Anda dapat mengambil langkah-langkah untuk mengoptimalkan penggunaan IPv4 alamat di subnet Anda. Lihat Mengonfigurasi Parameter untuk Delegasi Awalan untuk petunjuk terperinci tentang cara menyempurnakan permintaan IPv4 alamat dan proses alokasi. Menyesuaikan konfigurasi ini dapat membantu Anda mencapai keseimbangan antara menyimpan IPv4 alamat dan manfaat kepadatan pod dari delegasi awalan.

Saat menggunakan pengaturan default untuk menetapkan IPv4 alamat sekunder, saat ini tidak ada konfigurasi yang didukung untuk memanipulasi cara VPC Resource Controller meminta dan mengalokasikan alamat. IPv4 Lebih khusus lagi, `minimum-ip-target` dan hanya `warm-ip-target` didukung untuk mode delegasi awalan. Perhatikan juga bahwa dalam mode IP sekunder, tergantung pada alamat IP yang tersedia pada antarmuka, VPC Resource Controller biasanya akan mengalokasikan 3 IPv4 alamat yang tidak digunakan pada node atas nama Anda untuk menjaga IPs kehangatan untuk waktu startup pod yang lebih cepat. Jika Anda ingin meminimalkan pemborosan IP dari alamat IP hangat yang tidak digunakan, Anda dapat bertujuan untuk menjadwalkan lebih banyak pod pada node Windows tertentu sehingga Anda menggunakan kapasitas alamat IP ENI sebanyak mungkin. Lebih eksplisit, Anda dapat menghindari warm yang tidak digunakan IPs jika semua alamat IP pada ENI sudah digunakan oleh node dan menjalankan pod. Solusi lain untuk membantu Anda menyelesaikan kendala dengan ketersediaan alamat IP di subnet Anda adalah dengan mengeksplorasi peningkatan ukuran subnet Anda atau memisahkan node Windows Anda menjadi subnet khusus mereka sendiri.

Selain itu, penting untuk IPv6 dicatat bahwa tidak didukung pada node Windows saat ini.

Opsi Antarmuka Jaringan Kontainer (CNI)

AWSVPC CNI adalah plugin CNI de facto untuk node pekerja Windows dan Linux. Sementara AWSVPC CNI memenuhi kebutuhan banyak pelanggan, masih ada saat-saat ketika Anda perlu mempertimbangkan alternatif seperti jaringan overlay untuk menghindari kelelahan IP. Dalam kasus ini, Calico CNI dapat digunakan sebagai pengganti CNI. AWSVPC Project Calico adalah perangkat lunak open source yang dikembangkan oleh Tigera. Perangkat lunak itu termasuk CNI yang bekerja dengan EKS. Petunjuk untuk menginstal Calico CNI di EKS dapat ditemukan di halaman instalasi Project Calico EKS.

Kebijakan Jaringan

Merupakan praktik terbaik untuk mengubah dari mode default komunikasi terbuka antar pod di klaster Kubernetes Anda menjadi membatasi akses berdasarkan kebijakan jaringan. Project Calico open source memiliki dukungan kuat untuk kebijakan jaringan yang bekerja dengan node Linux dan

Windows. Fitur ini terpisah dan tidak tergantung pada penggunaan Calico CNI. Oleh karena itu kami merekomendasikan menginstal Calico dan menggunakannya untuk manajemen kebijakan jaringan.

Petunjuk untuk menginstal Calico di EKS dapat ditemukan di halaman Menginstal Calico di Amazon EKS.

Selain itu, saran yang diberikan dalam Panduan Praktik Terbaik Amazon EKS untuk Keamanan - Bagian Jaringan berlaku sama untuk klaster EKS dengan node pekerja Windows, namun, beberapa fitur seperti “Grup Keamanan untuk Pod” tidak didukung oleh Windows saat ini.

Menghindari kesalahan OOM

Windows tidak memiliki pembunuh out-of-memory proses seperti Linux. Windows selalu memperlakukan semua alokasi memori mode pengguna sebagai virtual, dan file halaman wajib. Efek bersihnya adalah bahwa Windows tidak akan menjangkau kondisi memori dengan cara yang sama seperti Linux. Proses akan halaman ke disk alih-alih tunduk pada penghentian out of memory (OOM). Jika memori terlalu banyak disediakan dan semua memori fisik habis, maka paging dapat memperlambat kinerja.

Sistem reservasi dan memori kubelet

Berbeda dari Linux di mana `--kubelet-reserve` menangkap reservasi sumber daya untuk daemon sistem kubernetes seperti kubelet, runtime container, dll; dan `--system-reserve` menangkap reservasi sumber daya untuk daemon sistem OS seperti sshd, udev, dll. Pada Windows flag ini tidak menangkap dan menetapkan batas memori pada kubelet atau proses yang berjalan pada node.

Namun, Anda dapat menggabungkan flag ini NodeAllocatableuntuk mengelola pengurangan Kapasitas pada node dengan batas sumber daya memori manifes Pod untuk mengontrol alokasi memori per pod. Dengan menggunakan strategi ini Anda memiliki kontrol alokasi memori yang lebih baik serta mekanisme untuk meminimalkan out-of-memory (OOM) pada node Windows.

Pada node Windows, praktik terbaik adalah memesan setidaknya 2GB memori untuk OS dan proses. Gunakan `--kubelet-reserve` and/or `--system-reserve` untuk mengurangi NodeAllocatable.

Mengikuti dokumentasi node Windows yang dikelola sendiri Amazon EKS, gunakan CloudFormation template untuk meluncurkan grup node Windows baru dengan penyesuaian ke konfigurasi kubelet. Ini CloudFormation memiliki elemen `BootstrapArguments` yang disebut yang sama dengan`KubeletExtraArgs`. Gunakan dengan tanda dan nilai berikut:

```
--kube-reserved memory=0.5Gi,ephemeral-storage=1Gi --system-
reserved memory=1.5Gi,ephemeral-storage=1Gi --eviction-hard
 memory.available<200Mi,nodefs.available<10%"
```

Jika eksctl adalah alat penerapan, periksa dokumentasi berikut untuk menyesuaikan konfigurasi kubelet https://eksctl. io/usage/customizing-kubelet/

Persyaratan memori wadah Windows

Sesuai dokumentasi Microsoft, image dasar Windows Server untuk NANO membutuhkan setidaknya 30MB, sedangkan Server Core membutuhkan 45MB. Angka-angka ini bertambah saat Anda menambahkan komponen Windows seperti .NET Framework, Layanan Web sebagai IIS dan aplikasi.

Penting bagi Anda untuk mengetahui jumlah minimum memori yang diperlukan oleh image container Windows Anda, yaitu image dasar ditambah lapisan aplikasinya, dan mengaturnya sebagai container resources/requests dalam spesifikasi pod. Anda juga harus menetapkan batas untuk menghindari pod mengkonsumsi semua memori node yang tersedia jika terjadi masalah aplikasi.

Pada contoh di bawah ini, ketika penjadwal Kubernetes mencoba menempatkan sebuah pod pada sebuah node, permintaan pod digunakan untuk menentukan node mana yang memiliki sumber daya yang cukup untuk penjadwalan.

```
spec:
 - name: iis
   image: mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019
   resources:
     limits:
       cpu: 1
       memory: 800Mi
     requests:
       cpu: .1
       memory: 128Mi
```

Kesimpulan

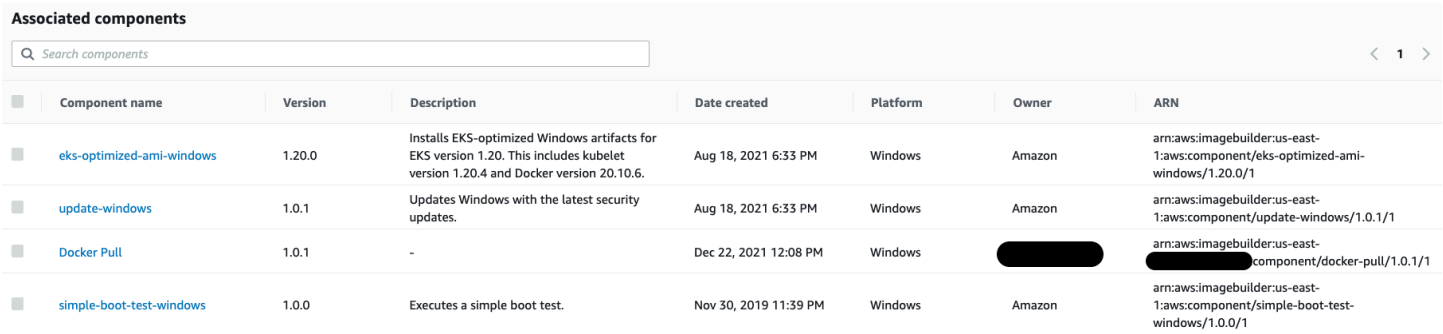
Menggunakan pendekatan ini meminimalkan risiko kelelahan memori tetapi tidak mencegahnya terjadi. Menggunakan Amazon CloudWatch Metrics, Anda dapat mengatur peringatan dan remediasi jika terjadi kelelahan memori.

Menambal Server dan Wadah Windows

Menambal Windows Server adalah tugas manajemen standar untuk Administrator Windows. Ini dapat dicapai dengan menggunakan alat yang berbeda seperti Amazon System Manager - Patch Manager, WSUS, System Center Configuration Manager, dan banyak lainnya. Namun, node Windows di cluster Amazon EKS tidak boleh diperlakukan sebagai server Windows biasa. Mereka harus diperlakukan sebagai server yang tidak dapat diubah. Sederhananya, hindari memperbarui node yang ada, cukup luncurkan yang baru berdasarkan AMI baru yang diperbarui.

Menggunakan EC2 Image Builder Anda dapat AMIs mengotomatiskan build, dengan membuat resep dan menambahkan komponen.

Contoh berikut menunjukkan komponen, yang dapat merupakan komponen yang sudah ada sebelumnya yang dibuat oleh AWS (dikelola Amazon) serta komponen yang Anda buat (Dimiliki oleh saya). Perhatikan baik-baik komponen yang dikelola Amazon yang disebut update-windows, ini memperbarui Windows Server sebelum menghasilkan AMI melalui pipeline EC2 Image Builder.



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 Image Builder memungkinkan Anda membuat AMI berbasis Amazon Managed AMIs Public dan menyesuaikannya untuk memenuhi kebutuhan bisnis Anda. Anda kemudian dapat mengaitkannya AMIs dengan Template Peluncuran yang memungkinkan Anda menautkan AMI baru ke Grup Auto Scaling yang dibuat oleh EKS Nodegroup. Setelah itu selesai, Anda dapat mulai menghentikan Node Windows yang ada dan yang baru akan diluncurkan berdasarkan AMI yang baru diperbarui.

Mendorong dan menarik gambar Windows

Amazon menerbitkan EKS yang dioptimalkan AMIs yang mencakup dua gambar kontainer Windows yang di-cache.

```
mcr.microsoft.com/windows/servercore
mcr.microsoft.com/windows/nanoserver
```

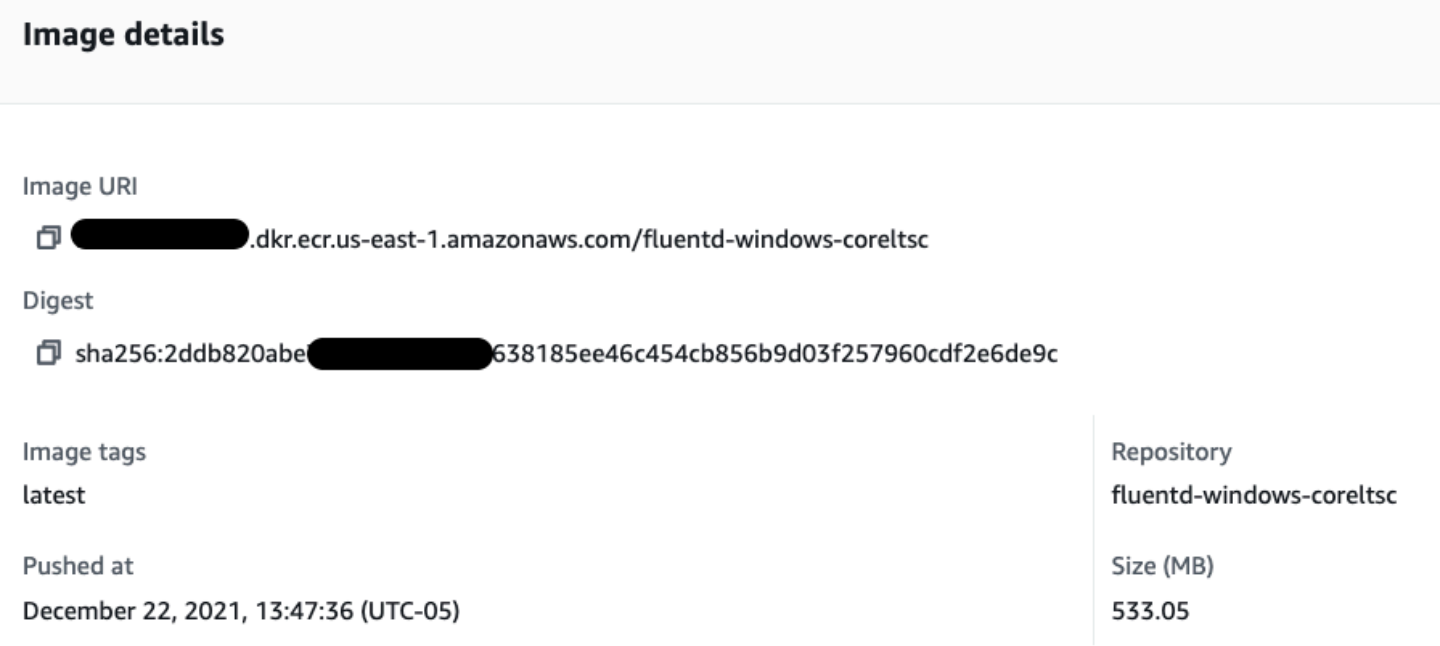
```
REPOSITORY                                                      TAG                     IMAGE ID      CREATED         SIZE
mcr.microsoft.com/windows/servercore                            ltsc2019                617dd5ead5b8  6 weeks ago     5.74GB
mcr.microsoft.com/windows/nanoserver                            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
```

Gambar cache diperbarui mengikuti pembaruan pada OS utama. Ketika Microsoft merilis pembaruan Windows baru yang secara langsung memengaruhi gambar dasar wadah Windows, pembaruan akan diluncurkan sebagai Pembaruan Windows biasa pada OS utama. Menjaga lingkungan up-to-date menawarkan lingkungan yang lebih aman di tingkat Node dan Container.

Ukuran gambar kontainer Windows memengaruhi push/pull operasi yang dapat menyebabkan waktu startup kontainer lambat. Caching gambar kontainer Windows memungkinkan I/O operasi mahal (ekstraksi file) terjadi pada pembuatan build AMI alih-alih peluncuran kontainer. Akibatnya, semua lapisan gambar yang diperlukan akan diekstraksi pada AMI dan akan siap digunakan, mempercepat waktu peluncuran wadah Windows dan dapat mulai menerima lalu lintas. Selama operasi push, hanya layer yang menyusun gambar Anda yang diunggah ke repositori.

Contoh berikut menunjukkan bahwa pada Amazon ECR gambar fluentd-windows-sac2004 hanya memiliki 390.18MB. Ini adalah jumlah unggahan yang terjadi selama operasi push.

Contoh berikut menunjukkan gambar ltsc Windows lancar yang didorong ke repositori Amazon ECR. Ukuran lapisan yang disimpan dalam ECR adalah 533.05MB.



Output di bawah dari`docker image ls`, ukuran fluentd v1.14-windows-ltsc2019-1 adalah 6.96GB pada disk, tetapi itu tidak berarti itu mengunduh dan mengekstrak jumlah data tersebut.

Dalam praktiknya, selama operasi tarik hanya 533.05MB terkompresi yang akan diunduh dan diekstraksi.

```
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
```

Kolom ukuran menunjukkan ukuran keseluruhan gambar, 6.96GB. Memecahnya:

- Gambar Dasar LTSC Windows Server Core 2019 = 5.74GB
- Gambar Dasar Tidak Terkompresi Lancar = 6.96GB
- Perbedaan pada disk = 1.2GB
- Gambar akhir terkompresi Fluentd ECR = 533.05MB

Gambar dasar sudah ada di disk lokal, sehingga jumlah total pada disk menjadi tambahan 1.2GB. Lain kali Anda melihat jumlah GBs di kolom ukuran, jangan terlalu khawatir, kemungkinan lebih dari 70% sudah ada di disk sebagai gambar wadah yang di-cache.

Referensi

Mempercepat waktu peluncuran kontainer Windows dengan pembuat Gambar EC2 dan strategi cache gambar

Menjalankan beban kerja Heterogen

Kubernetes memiliki dukungan untuk cluster heterogen di mana Anda dapat memiliki campuran node Linux dan Windows di cluster yang sama. Di dalam cluster itu, Anda dapat memiliki campuran Pod yang berjalan di Linux dan Pod yang berjalan di Windows. Anda bahkan dapat menjalankan beberapa versi Windows di cluster yang sama. Namun, ada beberapa faktor (seperti yang disebutkan di bawah) yang perlu diperhitungkan saat membuat keputusan ini.

Menetapkan PODs ke Node Praktik terbaik

Untuk menjaga beban kerja Linux dan Windows pada node khusus OS masing-masing, Anda perlu menggunakan beberapa kombinasi pemilih node dan noda/toleransi. Tujuan utama penjadwalan beban kerja di lingkungan yang heterogen adalah untuk menghindari kerusakan kompatibilitas untuk beban kerja Linux yang ada.

Memastikan beban kerja khusus OS mendarat di host kontainer yang sesuai

Pengguna dapat memastikan wadah Windows dapat dijadwalkan pada host yang sesuai menggunakan NodeSelectors. Semua node Kubernetes saat ini memiliki label default berikut:

```
kubernetes.io/os = [windows|linux]
kubernetes.io/arch = [amd64|arm64|...]
```

Jika spesifikasi Pod tidak menyertakan NodeSelector seperti`"kubernetes.io/os": windows`, Pod dapat dijadwalkan pada host mana pun, Windows atau Linux. Ini bisa menjadi masalah karena wadah Windows hanya dapat berjalan di Windows dan wadah Linux hanya dapat berjalan di Linux.

Di lingkungan Enterprise, tidak jarang memiliki sejumlah besar penerapan yang sudah ada sebelumnya untuk wadah Linux, serta ekosistem off-the-shelf konfigurasi, seperti bagan Helm. Dalam situasi ini, Anda mungkin ragu untuk membuat perubahan pada NodeSelectors penerapan. Alternatifnya adalah dengan menggunakan Taints.

Misalnya: `--register-with-taints='os=windows:NoSchedule'`

Jika Anda menggunakan EKS, eksctl menawarkan cara untuk menerapkan taints melalui ClusterConfig:

```
NodeGroups:
  - name: windows-ng
    amiFamily: WindowsServer2022FullContainer
    ...
    labels:
      nodeclass: windows2022
    taints:
      os: "windows:NoSchedule"
```

Menambahkan taint ke semua node Windows, scheduler tidak akan menjadwalkan pod pada node tersebut kecuali mereka mentolerir taint. Contoh manifes Pod:

```
nodeSelector:
    kubernetes.io/os: windows
tolerations:
    - key: "os"
      operator: "Equal"
      value: "windows"
      effect: "NoSchedule"
```

Menangani beberapa build Windows di cluster yang sama

Image dasar kontainer Windows yang digunakan oleh setiap pod harus cocok dengan versi build kernel yang sama dengan node. Jika Anda ingin menggunakan beberapa build Windows Server di cluster yang sama, maka Anda harus menetapkan label node tambahan, NodeSelectors atau memanfaatkan label yang disebut windows-build.

Kubernetes 1.17 secara otomatis menambahkan label baru node.kubernetes.io/windows-build untuk menyederhanakan pengelolaan beberapa build Windows di cluster yang sama. Jika Anda menjalankan versi yang lebih lama, maka disarankan untuk menambahkan label ini secara manual ke node Windows.

Label ini mencerminkan nomor mayor, minor, dan build Windows yang perlu dicocokkan untuk kompatibilitas. Di bawah ini adalah nilai yang digunakan hari ini untuk setiap versi Windows Server.

Penting untuk dicatat bahwa Windows Server pindah ke Long-Term Servicing Channel (LTSC) sebagai saluran rilis utama. Saluran Semi-Tahunan Windows Server (SAC) dihentikan pada 9 Agustus 2022. Tidak akan ada rilis SAC masa depan dari Windows Server.

Nama Produk	Membangun Nomor (s)
Server penuh 2022 LTSC	10.0.20348
Inti server 2019 LTSC	10.0.17763

Dimungkinkan untuk memeriksa versi build OS melalui perintah berikut:

```
kubectl get nodes -o wide
```

Output KERNEL-VERSION cocok dengan versi build OS 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
```

Contoh di bawah ini menerapkan NodeSelector tambahan ke manifes pod agar sesuai dengan versi Windows-build yang benar saat menjalankan versi OS grup node Windows yang berbeda.

```
nodeSelector:
    kubernetes.io/os: windows
    node.kubernetes.io/windows-build: '10.0.20348'
tolerations:
    - key: "os"
    operator: "Equal"
    value: "windows"
    effect: "NoSchedule"
```

Penyederhanaan NodeSelector dan Tolerasi dalam manifes Pod menggunakan RuntimeClass

Anda juga dapat memanfaatkan RuntimeClass untuk menyederhanakan proses penggunaan noda dan toleransi. Ini dapat dicapai dengan membuat RuntimeClass objek yang digunakan untuk merangkum noda dan toleransi ini.

Buat RuntimeClass dengan menjalankan manifes berikut:

```
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"
```

Setelah Runtimeclass dibuat, tetapkan sebagai Spec pada manifes 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
```

Dukungan Grup Node Terkelola

Untuk membantu pelanggan menjalankan aplikasi Windows mereka dengan cara yang lebih efisien, AWS meluncurkan dukungan untuk dukungan Amazon EKS Managed Node Group (MNG) untuk container Windows pada 15 Desember 2022. Untuk membantu menyelaraskan tim operasi, Windows MNGs diaktifkan menggunakan alur kerja dan alat yang sama seperti Linux. MNGs Versi keluarga AMI (Amazon Machine Image) lengkap dan inti dari Windows Server 2019 dan 2022 didukung.

Keluarga AMI berikut didukung untuk Grup Node Terkelola (MNG) s.

Keluarga AMI
Windows_Core_2019_x86_64
Windows_Full_2019_x86_64
Windows_Core_2022_X86_64
Windows_Full_2022_x86_64

Dokumentasi tambahan

Dokumentasi Resmi AWS: https://docs.aws.amazon.com/eks/ latest/userguide/windows -support.html

Untuk lebih memahami cara kerja Pod Networking (CNI), periksa tautan berikut: -networking.html https://docs.aws.amazon.com/eks/ latest/userguide/pod

Blog AWS tentang Menerapkan Grup Node Terkelola untuk Windows di EKS: https:// aws.amazon.com/blogs/ kontainer/ deploying-amazon-eks-windows -/managed-node-groups

Konteks Keamanan Pod

Pod Security Policies (PSP) dan Pod Security Standards (PSS) adalah dua cara utama untuk menegakkan keamanan di Kubernetes. Perhatikan bahwa PodSecurityPolicy Kubernetes v1.21 sudah usang, dan akan dihapus di v1.25 dan Pod Security Standard (PSS) adalah pendekatan yang direkomendasikan Kubernetes untuk menegakkan keamanan ke depan.

Pod Security Policy (PSP) adalah solusi native di Kubernetes untuk mengimplementasikan kebijakan keamanan. PSP adalah sumber daya tingkat cluster yang mengontrol aspek keamanan yang sensitif dari spesifikasi Pod. Dengan menggunakan Kebijakan Keamanan Pod, Anda dapat menentukan serangkaian kondisi yang harus dipenuhi oleh Pod agar dapat diterima oleh klaster. Fitur PSP telah tersedia sejak awal Kubernetes dan dirancang untuk memblokir pod yang salah konfigurasi agar tidak dibuat pada klaster tertentu.

Untuk informasi lebih lanjut tentang Kebijakan Keamanan Pod, silakan lihat dokumentasi Kubernetes. Menurut kebijakan penghentian Kubernetes, versi lama akan berhenti mendapatkan dukungan sembilan bulan setelah penghentian fitur.

Di sisi lain, Pod Security Standards (PSS) yang merupakan pendekatan keamanan yang direkomendasikan dan biasanya diimplementasikan menggunakan Konteks Keamanan didefinisikan sebagai bagian dari spesifikasi Pod dan container dalam manifes Pod. PSS adalah standar resmi yang telah ditetapkan oleh tim proyek Kubernetes untuk menangani praktik terbaik terkait keamanan untuk Pod. Ini mendefinisikan kebijakan seperti baseline (minimal restriktif, default), istimewa (tidak terbatas) dan terbatas (paling membatasi).

Kami merekomendasikan memulai dengan profil dasar. Profil dasar PSS memberikan keseimbangan yang solid antara keamanan dan potensi gesekan, membutuhkan daftar pengecualian minimal, ini berfungsi sebagai titik awal yang baik untuk keamanan beban kerja. Jika saat ini Anda menggunakan, PSPs kami sarankan beralih ke PSS. Detail lebih lanjut tentang kebijakan PSS dapat ditemukan di dokumentasi Kubernetes. Kebijakan ini dapat ditegakkan dengan beberapa alat termasuk dari OPA dan Kyverno. Misalnya, Kyverno menyediakan koleksi lengkap kebijakan PSS di sini.

Pengaturan konteks keamanan memungkinkan seseorang untuk memberikan hak istimewa untuk memilih proses, menggunakan profil program untuk membatasi kemampuan untuk program individu, memungkinkan eskalasi hak istimewa, memfilter panggilan sistem, antara lain.

Pod Windows di Kubernetes memiliki beberapa keterbatasan dan pembeda dari beban kerja berbasis Linux standar dalam hal konteks keamanan.

Windows menggunakan objek Job per kontainer dengan filter namespace sistem untuk memuat semua proses dalam wadah dan memberikan isolasi logis dari host. Tidak ada cara untuk menjalankan wadah Windows tanpa pemfilteran namespace di tempatnya. Ini berarti bahwa hak istimewa sistem tidak dapat ditegaskan dalam konteks host, dan dengan demikian wadah istimewa tidak tersedia di Windows.

Berikut ini `windowsOptions` adalah satu-satunya opsi Konteks Keamanan Windows yang didokumentasikan sementara sisanya adalah opsi Konteks Keamanan umum

Untuk daftar atribut konteks keamanan yang didukung di Windows vs linux, silakan lihat dokumentasi resmi di sini.

Pengaturan khusus Pod diterapkan ke semua kontainer. Jika tidak ditentukan, opsi dari PodSecurityContext akan digunakan. Jika diatur dalam keduanya SecurityContext dan PodSecurityContext, nilai yang ditentukan dalam SecurityContext diutamakan.

Misalnya, pengaturan runAsUser Nama untuk Pod dan kontainer yang merupakan opsi Windows adalah setara kasar dengan runAsUser pengaturan khusus Linux dan dalam manifes berikut, konteks keamanan khusus pod diterapkan ke semua kontainer

```
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
```

Sedangkan di bawah ini, konteks keamanan tingkat kontainer mengesampingkan konteks keamanan tingkat 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
```

Contoh nilai yang dapat diterima untuk bidang runAsUser Nama: ContainerAdministrator, ContainerUser, NT AUTHORITY\ NETWORK SERVICE, NT AUTHORITY\ LOCAL SERVICE

Umumnya merupakan ide yang baik untuk menjalankan container Anda dengan ContainerUser pod Windows. Pengguna tidak dibagi antara penampung dan host tetapi ContainerAdministrator memang memiliki hak istimewa tambahan di dalam wadah. Perhatikan bahwa, ada batasan nama pengguna yang harus diperhatikan.

Contoh yang baik tentang kapan harus digunakan ContainerAdministrator adalah mengatur PATH. Anda dapat menggunakan direktif USER untuk melakukannya, seperti:

```
USER ContainerAdministrator
RUN setx /M PATH "%PATH%;C:/your/path"
USER ContainerUser
```

Perhatikan juga bahwa, rahasia ditulis dalam teks yang jelas pada volume node (dibandingkan dengan tmpfs/in-memory di linux). Ini berarti Anda harus melakukan dua hal

- Gunakan file ACLs untuk mengamankan lokasi file rahasia
- Gunakan enkripsi tingkat volume menggunakan BitLocker

Opsi penyimpanan persisten

Apa itu plugin in-tree vs. out-of-tree volume?

Sebelum pengenalan Container Storage Interface (CSI), semua plugin volume di-tree yang berarti mereka dibangun, ditautkan, dikompilasi, dan dikirimkan dengan binari inti Kubernetes dan memperluas inti Kubernetes API. Ini berarti bahwa menambahkan sistem penyimpanan baru ke Kubernetes (plugin volume) memerlukan pemeriksaan kode ke dalam repositori kode Kubernetes inti.

Out-of-tree plugin volume dikembangkan secara independen dari basis kode Kubernetes, dan digunakan (diinstal) pada klaster Kubernetes sebagai ekstensi. Ini memberi vendor kemampuan untuk memperbarui driver out-of-band, yaitu secara terpisah dari siklus rilis Kubernetes. Hal ini sebagian besar dimungkinkan karena Kubernetes telah menciptakan antarmuka penyimpanan atau CSI yang menyediakan vendor cara standar untuk berinteraksi dengan k8s.

Anda dapat memeriksa lebih lanjut tentang kelas penyimpanan Amazon Elastic Kubernetes Services (EKS) dan Driver CSI di.html https://docs.aws.amazon.com/eks/ latest/userguide/storage

Plugin Volume In-tree untuk Windows

Volume Kubernetes memungkinkan aplikasi, dengan persyaratan persistensi data, untuk diterapkan di Kubernetes. Pengelolaan volume persisten terdiri dari kontainer provisioning/de-provisioning/ resizing of volumes, attaching/detaching a volume to/from a Kubernetes node, and mounting/ dismounting a volume to/from individu dalam sebuah pod. Kode untuk mengimplementasikan tindakan manajemen volume ini untuk back-end atau protokol penyimpanan tertentu dikirimkan dalam bentuk plugin volume Kubernetes (In-tree Volume Plugins). Di Amazon Elastic Kubernetes Services (EKS) kelas berikut dari plugin volume Kubernetes didukung pada Windows:

Plugin Volume Dalam Pohon: Simpan awsElasticBlock

Untuk menggunakan plugin volume In-tree pada node Windows, perlu untuk membuat tambahan StorageClass untuk menggunakan NTFS sebagai FSType. Pada EKS, default StorageClass menggunakan ext4 sebagai fSType default.

A StorageClass menyediakan cara bagi administrator untuk menggambarkan “kelas” penyimpanan yang mereka tawarkan. Kelas yang berbeda mungkin memetakan ke quality-of-service level, kebijakan cadangan, atau kebijakan arbitrer yang ditentukan oleh administrator klaster. Kubernetes tidak beropini tentang apa yang diwakili kelas. Konsep ini kadang-kadang disebut “profil” dalam sistem penyimpanan lain.

Anda dapat memeriksanya dengan menjalankan perintah berikut:

```
kubectl describe storageclass gp2
```

Output:

```
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>
```

Untuk membuat yang baru StorageClass untuk mendukung NTFS, gunakan manifes berikut:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gp2-windows
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  fsType: ntfs
volumeBindingMode: WaitForFirstConsumer
```

Buat StorageClass dengan menjalankan perintah berikut:

```
kubectl apply -f NTFSStorageClass.yaml
```

Langkah selanjutnya adalah membuat Persistent Volume Claim (PVC).

A PersistentVolume (PV) adalah bagian penyimpanan dalam cluster yang telah disediakan oleh administrator atau secara dinamis disediakan menggunakan PVC. Ini adalah sumber daya di cluster seperti node adalah sumber daya cluster. Objek API ini menangkap detail implementasi penyimpanan, baik itu NFS, iSCSI, atau sistem penyimpanan. cloud-provider-specific

A PersistentVolumeClaim (PVC) adalah permintaan penyimpanan oleh pengguna. Klaim dapat meminta ukuran dan mode akses tertentu (misalnya, mereka dapat dipasang ReadWriteOnce, ReadOnlyMany atau ReadWriteMany).

Pengguna membutuhkan atribut PersistentVolumes yang berbeda, seperti kinerja, untuk kasus penggunaan yang berbeda. Administrator cluster harus dapat menawarkan variasi PersistentVolumes yang berbeda dalam lebih dari sekadar ukuran dan mode akses, tanpa mengekspos pengguna pada detail tentang bagaimana volume tersebut diimplementasikan. Untuk kebutuhan ini, ada StorageClass sumber daya.

Dalam contoh di bawah ini, PVC telah dibuat di dalam jendela namespace.

```
apiVersion: v1
```

```
kind: PersistentVolumeClaim
metadata:
  name: ebs-windows-pv-claim
  namespace: windows
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gp2-windows
  resources:
    requests:
      storage: 1Gi
```

Buat PVC dengan menjalankan perintah berikut:

```
kubectl apply -f persistent-volume-claim.yaml
```

Manifes berikut membuat Pod Windows, mengatur VolumeMount as `C:\Data` dan menggunakan PVC sebagai penyimpanan terpasang`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'
```

Uji hasilnya dengan mengakses pod Windows melalui PowerShell:

```
kubectl exec -it podname powershell -n windows
```

Di dalam Pod Windows, jalankan: `ls`

Output:

```
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
```

Direktori data disediakan oleh volume EBS.

Out-of-tree untuk Windows

Kode yang terkait dengan plugin CSI dikirimkan sebagai out-of-tree skrip dan binari yang biasanya didistribusikan sebagai image kontainer dan digunakan menggunakan konstruksi Kubernetes standar seperti dan. DaemonSets StatefulSets Plugin CSI menangani berbagai tindakan manajemen volume di Kubernetes. Plugin CSI biasanya terdiri dari plugin node (yang berjalan pada setiap node sebagai a DaemonSet) dan plugin controller.

Plugin node CSI (terutama yang terkait dengan volume persisten yang diekspos sebagai perangkat blok atau melalui sistem file bersama) perlu melakukan berbagai operasi istimewa seperti pemindaian perangkat disk, pemasangan sistem file, dll. Operasi ini berbeda untuk setiap sistem operasi host. Untuk node pekerja Linux, plugin node CSI kontainer biasanya digunakan sebagai wadah istimewa. Untuk node pekerja Windows, operasi istimewa untuk plugin node CSI kontainer didukung menggunakan csi-proxy, biner mandiri yang dikelola komunitas yang perlu diinstal sebelumnya pada setiap node Windows.

Amazon EKS Dioptimalkan Windows AMI menyertakan CSI-Proxy mulai dari April 2022. Pelanggan dapat menggunakan Driver SMB CSI pada node Windows untuk mengakses Amazon untuk Windows File Server, Amazon FSx FSx untuk NetApp ONTAP SMB Shares, and/or AWS Storage Gateway — File Gateway.

Blog berikut memiliki detail implementasi tentang cara mengatur Driver SMB CSI untuk menggunakan Amazon FSx untuk Windows File Server sebagai penyimpanan persisten untuk Pod Windows.

Amazon FSx untuk Server File Windows

Pilihannya adalah menggunakan Amazon FSx untuk Windows File Server melalui fitur SMB yang disebut SMB Global Mapping yang memungkinkan untuk memasang berbagi SMB di host, lalu meneruskan direktori pada bagian itu ke dalam wadah. Wadah tidak perlu dikonfigurasi dengan server, berbagi, nama pengguna, atau kata sandi tertentu - itu semua ditangani di host sebagai gantinya. Wadah akan bekerja sama seperti jika memiliki penyimpanan lokal.

> Pemetaan Global SMB transparan bagi orkestrator, dan dipasang di mana dapat menyiratkan masalah HostPath yang aman.

Pada contoh di bawah ini, jalurnya `G:\Directory\app-state` adalah berbagi SMB di 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
```

Blog berikut memiliki detail implementasi tentang cara mengatur Amazon FSx untuk Windows File Server sebagai penyimpanan persisten untuk Pod Windows.

Pengerasan gambar wadah Windows

Apakah Anda mengeraskan gambar wadah Windows Anda? Selama bertahun-tahun, saya telah bekerja dengan pelanggan secara global untuk membantu mereka memigrasikan beban kerja lama ke wadah, terutama beban kerja Windows. Dengan pengalaman lebih dari 20 tahun, saya telah melihat organisasi mendedikasikan upaya dan sumber daya yang besar untuk memperkuat Server Windows mereka, menerapkan semuanya mulai dari Tolok Ukur CIS hingga perlindungan antivirus runtime untuk melindungi data sensitif.

Namun, tren yang mengkhawatirkan telah muncul. Karena mesin virtual yang sangat aman ini dimodernisasi ke dalam wadah, banyak praktik pengerasan kritis diabaikan. Praktik terbaik keamanan

Windows, dari gambar dasar (OS) hingga layanan web seperti IIS, sering diabaikan, dengan sebagian besar fokus ditempatkan hanya pada pengamanan host kontainer. Sangat penting untuk menyadari bahwa sementara kontainer beroperasi di ruang nama yang terisolasi, mereka masih berbagi primitif kernel dengan host. Penyerang biasanya lebih tertarik pada gerakan lateral daripada menargetkan host kontainer secara langsung, memungkinkan mereka untuk mengeksploitasi pengaturan keamanan kontainer yang lemah dan mengakses data sensitif.

Tujuan dari dokumentasi ini adalah untuk menyoroti beberapa pengaturan keamanan penting yang harus Anda terapkan secara khusus untuk `ASP.NET` situs web hosting kontainer Windows di IIS. Kami akan fokus pada empat bidang utama:

- Kebijakan keamanan akun
- Kebijakan audit
- Praktik terbaik keamanan IIS
- Prinsip hak istimewa yang paling kecil

Kami akan mulai dengan menyelidiki mengapa masing-masing konfigurasi keamanan ini sangat penting untuk melindungi wadah Windows Anda, memeriksa risiko spesifik yang mereka kurangi dan manfaat keamanan yang mereka berikan. Selanjutnya, kita akan menelusuri cuplikan kode yang menunjukkan cara mengimplementasikan konfigurasi ini dengan benar di Dockerfile Anda, memastikan container Anda diperkeras terhadap potensi ancaman. Terakhir, kami akan memecah setiap pengaturan secara rinci, menawarkan penjelasan komprehensif tentang fungsinya, dampaknya pada keamanan kontainer, dan bagaimana kontribusinya untuk melindungi aplikasi Anda. Pendekatan ini tidak hanya akan menunjukkan kepada Anda bagaimana menerapkan praktik terbaik ini, tetapi juga memberi Anda wawasan untuk memahami mengapa mereka penting untuk mempertahankan postur keamanan yang kuat di lingkungan dalam peti kemas.

1. Konfigurasikan Kebijakan Akun (Kata Sandi atau Lockout) menggunakan Kebijakan Keamanan Lokal dan Registri

Windows Server Core adalah opsi instalasi minimal yang tersedia sebagai bagian dari [EKS Optimized Windows AMI] (https://docs.aws.amazon.com/eks/latest/userguide/eks- optimized-windows-ami .html). Mengkonfigurasi Kebijakan Akun (Kata Sandi atau Penguncian) menggunakan Kebijakan Keamanan Lokal dan Registri memperkuat keamanan sistem dengan menerapkan aturan kata sandi dan penguncian yang kuat. Kebijakan ini mengharuskan pengguna untuk membuat kata sandi yang kuat dengan panjang dan kompleksitas minimum yang ditentukan, melindungi terhadap serangan umum terkait kata sandi.

Dengan menetapkan usia kata sandi maksimum, pengguna diminta untuk memperbarui kata sandi mereka secara teratur, mengurangi kemungkinan kredensitas yang disusupi. Kebijakan penguncian menambahkan lapisan perlindungan tambahan dengan mengunci akun sementara setelah sejumlah upaya login gagal, membantu mencegah serangan brute-force. Mengkonfigurasi pengaturan ini melalui Windows Registry memungkinkan administrator untuk menegakkan langkah-langkah keamanan ini di tingkat sistem, memastikan keseragaman dan kepatuhan di seluruh organisasi. Menerapkan Kebijakan Akun ini di Windows Container sangat penting untuk menjaga konsistensi keamanan, meskipun kontainer sering bersifat sementara dan ditujukan untuk beban kerja yang terisolasi:

Konsistensi Keamanan

- Kepatuhan: Menegakkan kebijakan kata sandi yang konsisten dan aturan penguncian dalam wadah membantu menjaga kepatuhan keamanan, terutama di lingkungan yang memerlukan kontrol akses yang ketat (misalnya, kepatuhan terhadap peraturan seperti HIPAA, PCI-DSS).
- Kontainer Hardened: Menerapkan pengaturan ini memastikan bahwa wadah Windows Anda dikeraskan terhadap akses tidak sah atau serangan berbasis kata sandi, menyelaraskan postur keamanan wadah Anda dengan kebijakan keamanan sistem yang lebih luas.

Perlindungan Terhadap Serangan Brute Force

- Penguncian Akun: Pengaturan ini membantu mempertahankan diri dari upaya login brute force dengan mengunci akun setelah sejumlah upaya login gagal. Ini mencegah penyerang mencoba kata sandi dalam jumlah yang tidak terbatas.
- Kompleksitas Kata Sandi: Membutuhkan kata sandi yang kompleks dengan panjang yang cukup mengurangi kemungkinan kata sandi yang lemah dieksploitasi, bahkan di lingkungan kontainer yang terisolasi.

Skenario Multi-Pengguna

- Jika aplikasi kontainer Anda dirancang untuk menangani beberapa pengguna atau memerlukan otentikasi pengguna, menerapkan kebijakan kata sandi memastikan bahwa akun pengguna dalam wadah mematuhi aturan keamanan yang ketat, sehingga membatasi akses hanya ke pengguna yang berwenang.

Kontainer Windows yang Persisten

- Sementara kontainer umumnya dianggap fana, kontainer Windows tertentu dapat menjalankan layanan jangka panjang atau menangani manajemen pengguna, sehingga penting untuk menegakkan kebijakan keamanan yang tepat mirip dengan server Windows biasa.

Konsistensi dalam Lingkungan Hibrida

- Jika Anda menjalankan mesin virtual dan kontainer di infrastruktur Anda, menerapkan kebijakan keamanan yang sama (misalnya, password/lockout kebijakan) di semua lingkungan memastikan standar keamanan yang seragam, menyederhanakan tata kelola dan manajemen.

Singkatnya, menerapkan kebijakan akun ini dalam wadah Windows memastikan bahwa kontainer Anda bukan titik lemah dalam strategi keamanan Anda, melindungi terhadap serangan kata sandi dan menegakkan konsistensi di seluruh lingkungan Anda.

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
```

Penjelasan:

Bagian ini mengonfigurasi kebijakan akun untuk pengaturan kata sandi dan penguncian melalui Windows Registry. Kebijakan ini membantu menegakkan keamanan dengan mengontrol persyaratan kata sandi dan ambang batas penguncian akun.

1. MinimumPasswordLength (MINPWLEN) = 14 Pengaturan ini mendefinisikan jumlah minimum karakter untuk kata sandi. Rentangnya adalah 0-14 karakter; defaultnya adalah enam karakter.
2. MaximumPasswordAge (MAXPWAGE) = 60 Pengaturan ini mendefinisikan jumlah hari maksimum bahwa kata sandi valid. Tidak ada batasan yang ditentukan dengan menggunakan UNLIMITED. / MAXPWAGE tidak bisa kurang dari /MINPWAGE. Kisarannya adalah 1-999; defaultnya adalah 90 hari
3. Lockout Threshold (LOCKOUTTHRESHOLD) = 5 Pengaturan ini menentukan ambang batas untuk upaya login yang gagal. Setelah 5 upaya yang salah, akun akan dikunci.

Pengaturan ini membantu meningkatkan keamanan kata sandi dan mencegah serangan brute force dengan menegakkan kebijakan kata sandi yang kuat dan mengunci akun setelah sejumlah upaya login gagal.

2. Kebijakan audit

Kebijakan Audit penting untuk Windows Container karena mereka memberikan visibilitas penting ke dalam peristiwa keamanan, seperti upaya login dan penggunaan hak istimewa, membantu mendeteksi akses yang tidak sah, memantau aktivitas pengguna, dan memastikan kepatuhan dengan standar peraturan. Bahkan dalam sifat kontainer yang fana, log audit sangat penting untuk penyelidikan insiden, deteksi ancaman proaktif, dan mempertahankan postur keamanan yang konsisten di seluruh lingkungan dalam peti kemas.

Pemantauan dan Kepatuhan Keamanan:

- Lacak Aktivitas Pengguna: Kebijakan audit memungkinkan administrator untuk memantau aktivitas pengguna, seperti upaya masuk dan penggunaan hak istimewa, di dalam wadah. Ini sangat penting untuk mendeteksi akses yang tidak sah atau perilaku mencurigakan.
- Kepatuhan Peraturan: Banyak organisasi diharuskan mencatat peristiwa keamanan untuk kepatuhan terhadap peraturan seperti HIPAA, PCI-DSS, dan GDPR. Mengaktifkan kebijakan audit dalam kontainer memastikan Anda memenuhi persyaratan ini, bahkan di lingkungan kontainer.

Investigasi Insiden:

- Forensik dan Analisis: Jika aplikasi atau layanan dalam peti kemas dikompromikan, log audit dapat memberikan wawasan berharga untuk analisis pasca-insiden. Mereka membantu tim keamanan melacak tindakan yang diambil oleh penyerang atau mengidentifikasi bagaimana pelanggaran terjadi.
- Deteksi Real-time: Log audit memungkinkan administrator menyiapkan peringatan waktu nyata untuk peristiwa penting (misalnya, upaya login yang gagal, eskalasi hak istimewa). Pemantauan proaktif ini membantu mendeteksi serangan lebih awal dan memungkinkan waktu respons yang lebih cepat.

Konsistensi Lintas Lingkungan:

- Postur Keamanan Seragam: Dengan menerapkan kebijakan audit dalam kontainer melalui registri, Anda memastikan praktik keamanan yang konsisten di lingkungan kontainer dan non-kontainer. Ini menghindari kontainer menjadi titik buta untuk pemantauan keamanan.
- Visibilitas di Lingkungan Hybrid: Untuk organisasi yang menjalankan server dan wadah Windows tradisional, kebijakan audit memberikan visibilitas dan kontrol serupa di semua platform, membuat manajemen lebih mudah dan lebih efektif.

Melacak Operasi Istimewa:

- Hak Istimewa Penggunaan Audit: Dalam lingkungan kontainer di mana aplikasi berjalan dengan hak istimewa yang tinggi atau di mana tugas administratif dilakukan, mengaudit operasi istimewa memastikan akuntabilitas. Anda dapat mencatat siapa yang mengakses sumber daya sensitif atau melakukan tugas-tugas penting di dalam wadah.
- Mencegah Penyalahgunaan Hak Istimewa: Dengan memantau penggunaan hak istimewa, Anda dapat mendeteksi ketika pengguna yang tidak sah mencoba meningkatkan hak istimewa mereka atau mengakses area terlarang di dalam wadah, yang membantu mencegah serangan internal atau eksternal.

Mendeteksi Upaya Akses Tidak Sah:

- Upaya Logon Gagal: Mengaktifkan kebijakan audit untuk upaya login yang gagal membantu mengidentifikasi serangan brute-force atau upaya tidak sah untuk mengakses aplikasi kontainer. Ini memberikan visibilitas ke siapa yang mencoba mendapatkan akses ke sistem dan seberapa sering.
- Pemantauan Penguncian Akun: Mengaudit peristiwa penguncian akun memungkinkan administrator mendeteksi dan menyelidiki kemungkinan penguncian yang disebabkan oleh aktivitas mencurigakan atau berbahaya.

Keamanan Persisten Bahkan di Lingkungan Fana:

- Ephemeral Yet Secure: Meskipun kontainer bersifat sementara, artinya kontainer dapat dihapus dan dibuat ulang sering, audit masih memainkan peran kunci dalam memastikan bahwa peristiwa keamanan ditangkap saat penampung sedang berjalan. Ini memastikan bahwa peristiwa keamanan penting dicatat selama siklus hidup kontainer.

Logging Terpusat:

- Meneruskan Log ke Sistem Terpusat: Kontainer dapat diintegrasikan dengan sistem logging terpusat (misalnya, ELK stack, CloudWatch AWS) untuk menangkap log audit dari beberapa instance kontainer. Ini memungkinkan analisis dan korelasi peristiwa keamanan yang lebih baik di seluruh infrastruktur Anda.

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
```

Penjelasan:

Bagian ini mengonfigurasi kebijakan audit dengan menggunakan modifikasi registri. Kebijakan audit mengontrol peristiwa keamanan apa yang dicatat oleh Windows, yang membantu dalam memantau dan mendeteksi upaya akses yang tidak sah.

1. SCENoApplyLegacyAuditPolicy = 0 Ini menonaktifkan format kebijakan audit lama, memungkinkan kebijakan audit yang lebih terperinci yang diperkenalkan di versi Windows yang lebih baru. Ini penting untuk konfigurasi audit modern.
2. Subkategori Auditpol: “Logon” Pengaturan ini memungkinkan audit untuk peristiwa login sukses dan gagal. Nilai 3 berarti bahwa Windows akan mencatat upaya masuk yang berhasil dan gagal. Ini membantu dalam memantau siapa yang mengakses sistem dan menangkap upaya login yang gagal.

Kebijakan audit ini sangat penting untuk pemantauan dan kepatuhan keamanan, karena mereka menyediakan log terperinci dari peristiwa keamanan penting seperti upaya login dan penggunaan operasi istimewa.

3. Praktik terbaik Keamanan IIS untuk wadah Windows

Menerapkan praktik terbaik IIS di Windows Container penting karena beberapa alasan, memastikan bahwa aplikasi Anda aman, berkinerja tinggi, dan dapat diskalakan. Meskipun kontainer menyediakan isolasi dan lingkungan yang ringan, mereka masih memerlukan konfigurasi yang tepat untuk menghindari kerentanan dan masalah operasional. Inilah mengapa mengikuti praktik terbaik untuk IIS di Windows Containers sangat penting:

Keamanan

- Mencegah Kerentanan Umum: IIS sering menjadi target serangan seperti cross-site scripting (XSS), clickjacking, dan pengungkapan informasi. Menerapkan header keamanan (misalnya, X-Content-Type-Options, X-Frame-Options, dan Strict-Transport-Security) membantu melindungi aplikasi Anda dari ancaman ini.
- Isolasi Tidak Cukup: Kontainer terisolasi, tetapi instance IIS yang salah dikonfigurasi dapat mengekspos informasi sensitif, seperti detail versi server, daftar direktori, atau komunikasi yang tidak terenkripsi. Dengan menonaktifkan fitur seperti penelusuran direktori dan menghapus header versi IIS, Anda meminimalkan permukaan serangan.
- Enkripsi dan HTTPS: Praktik terbaik, seperti menegakkan koneksi khusus HTTP, memastikan bahwa data dalam perjalanan dienkripsi, melindungi informasi sensitif agar tidak dicegat.

Performa

- Penggunaan Sumber Daya yang Efisien: Praktik terbaik IIS seperti mengaktifkan kompresi dinamis dan statis mengurangi penggunaan bandwidth dan meningkatkan waktu muat. Pengoptimalan ini sangat penting dalam lingkungan kontainer, di mana sumber daya dibagikan di seluruh kontainer dan sistem host.
- Logging yang Dioptimalkan: Mengkonfigurasi logging dengan benar (misalnya, termasuk X-Forwarded-For header) memastikan bahwa Anda dapat melacak aktivitas klien sambil meminimalkan overhead logging yang tidak perlu. Ini membantu Anda mengumpulkan data yang relevan untuk pemecahan masalah tanpa menurunkan kinerja.

Skalabilitas dan Pemeliharaan

- Konsistensi di Seluruh Lingkungan: Dengan mengikuti praktik terbaik, Anda memastikan bahwa konfigurasi IIS Anda konsisten di beberapa instance kontainer. Ini menyederhanakan penskalaan

dan memastikan bahwa ketika kontainer baru diterapkan, mereka mematuhi pedoman keamanan dan kinerja yang sama.

- Konfigurasi Otomatis: Praktik terbaik di Dockerfiles, seperti mengatur izin folder dan menonaktifkan fitur yang tidak perlu, memastikan bahwa setiap wadah baru dikonfigurasi secara otomatis dengan benar. Ini mengurangi intervensi manual dan menurunkan risiko kesalahan manusia.

Kepatuhan

- Memenuhi Persyaratan Peraturan: Banyak industri memiliki persyaratan peraturan yang ketat (misalnya, PCI-DSS, HIPAA) yang mengamanatkan langkah-langkah keamanan tertentu, seperti komunikasi terenkripsi (HTTPS) dan pencatatan permintaan klien. Mengikuti praktik terbaik IIS dalam wadah membantu memastikan kepatuhan terhadap standar ini.
- Auditabilitas: Menerapkan kebijakan audit dan pencatatan aman memungkinkan penelusuran peristiwa, yang sangat penting dalam audit. Misalnya, mencatat X-Forwarded-For header memastikan bahwa alamat IP klien direkam dengan benar dalam arsitektur berbasis proxy.

Meminimalkan Risiko di Lingkungan Bersama

- Menghindari Kesalahan Konfigurasi: Kontainer berbagi kernel host, dan sementara mereka diisolasi satu sama lain, instance IIS yang dikonfigurasi dengan buruk dapat mengekspos kerentanan atau membuat kemacetan kinerja. Praktik terbaik memastikan bahwa setiap instans IIS berjalan secara optimal, mengurangi risiko masalah lintas kontainer.
- Akses Hak Istimewa Paling Sedikit: Menyetel izin yang tepat untuk folder dan file di dalam wadah (misalnya, menggunakan Set-Acl in PowerShell) memastikan bahwa pengguna dan proses dalam wadah hanya memiliki akses yang diperlukan, mengurangi risiko eskalasi hak istimewa atau gangguan data.

Ketahanan di Lingkungan Fana

- Sifat Kontainer Ephemeral: Wadah sering berumur pendek dan sering dibangun kembali. Menerapkan praktik terbaik IIS memastikan bahwa setiap kontainer dikonfigurasi dengan aman dan konsisten, terlepas dari berapa kali penampung tersebut dipindahkan. Ini mencegah kesalahan konfigurasi diperkenalkan dari waktu ke waktu.
- Mengurangi Potensi Kesalahan Konfigurasi: Dengan menerapkan praktik terbaik secara otomatis (misalnya, menonaktifkan protokol atau header yang lemah), risiko kesalahan konfigurasi selama restart atau pembaruan kontainer diminimalkan.

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"
```

Penjelasan:

Perintah ini mengkonfigurasi IIS untuk log X-Forwarded-For header, yang biasanya digunakan untuk menangkap alamat IP klien asli ketika permintaan melewati proxy atau load balancer. Secara default, IIS hanya mencatat alamat IP penyeimbang beban atau proxy terbalik, jadi menambahkan bidang log khusus ini membantu melacak IP klien yang sebenarnya untuk audit keamanan, analitik, dan pemecahan masalah.

1. X-Forwarded-For header yang biasanya digunakan untuk menangkap alamat IP klien asli ketika permintaan melewati proxy atau load balancer. Secara default, IIS hanya mencatat alamat IP

penyeimbang beban atau proxy terbalik, jadi menambahkan bidang log khusus ini membantu melacak IP klien yang sebenarnya untuk audit keamanan, analitik, dan pemecahan masalah.

2. Strict-Transport-Security (HSTS) Memastikan browser hanya berkomunikasi melalui HTTPS. Max-age=31536000 menetapkan bahwa kebijakan ini diberlakukan selama 1 tahun, dan menerapkan kebijakan untuk semua subdomain. includeSubDomains
3. X-Content-Type-Options Mencegah browser dari “Mime-sniffing” respons dari Content-Type yang dideklarasikan. Ini membantu mencegah beberapa jenis serangan.
4. Perlindungan X-XSS Memungkinkan perlindungan Cross-Site Scripting (XSS) di browser.
5. X-Frame-Options Mencegah halaman disematkan di iframe, melindungi dari serangan clickjacking.
6. Nonaktifkan pengungkapan versi IIS Perintah ini menonaktifkan header Server dalam respons HTTP, yang secara default mengungkapkan versi IIS yang digunakan. Menyembunyikan informasi ini membantu mengurangi risiko penyerang mengidentifikasi dan menargetkan kerentanan khusus untuk versi IIS.
7. Aktifkan koneksi khusus HTTP Bagian ini (dikomentari keluar) memberlakukan koneksi HTTPS dan menonaktifkan HTTP. Jika tidak dikomentari, Dockerfile akan mengonfigurasi IIS untuk mendengarkan hanya pada port 443 (HTTPS) dan menghapus pengikatan HTTP default pada port 80. Ini berguna saat mengakhiri SSL di dalam wadah dan memastikan bahwa semua lalu lintas dienkripsi.
8. Nonaktifkan Penjelajahan Direktori Mencegah IIS menampilkan daftar direktori saat tidak ada dokumen default. Ini menghindari mengekspos struktur file internal kepada pengguna.
9. Pass Through Custom Error Pages Memastikan bahwa jika aplikasi memiliki penanganan kesalahan sendiri, IIS akan membiarkan halaman kesalahan aplikasi melewati alih-alih menampilkan halaman kesalahan IIS default.
10. Mode Kesalahan Terperinci Mengonfigurasi IIS untuk menampilkan pesan kesalahan terperinci hanya untuk permintaan lokal, membantu pengembang mendiagnosis masalah tanpa mengekspos informasi sensitif kepada pengguna eksternal.
11. Pastikan Izin Folder yang Tepat Blok ini mengonfigurasi izin folder untuk root web IIS (C:\inetpub \wwwroot). Ini menetapkan izin Baca dan Jalankan untuk grup IIS_IUSRS dan Pengguna, memastikan bahwa pengguna ini dapat mengakses folder tetapi tidak memodifikasi file. Mengatur izin yang benar meminimalkan risiko akses tidak sah atau merusak file yang dihosting oleh server web.

Mengikuti praktik terbaik IIS di Windows Containers memastikan bahwa aplikasi kontainer Anda aman, berkinerja tinggi, dan dapat diskalakan. Praktik ini membantu mencegah kerentanan,

mengoptimalkan penggunaan sumber daya, memastikan kepatuhan, dan menjaga konsistensi di seluruh instance container. Meskipun kontainer dirancang untuk diisolasi, konfigurasi yang tepat diperlukan untuk meminimalkan risiko dan memastikan keandalan aplikasi Anda di lingkungan produksi.

4. Prinsip Keistimewaan Terkecil

Prinsip Keistimewaan Terkecil (PolP) sangat penting untuk wadah Windows karena beberapa alasan penting, terutama dalam meningkatkan keamanan dan meminimalkan risiko dalam lingkungan kontainer. Prinsip ini menentukan bahwa sistem atau aplikasi harus beroperasi dengan tingkat izin minimum yang diperlukan untuk berfungsi dengan baik. Inilah mengapa penting dalam wadah Windows:

Meminimalkan Permukaan Serangan

- Container sering menjalankan aplikasi yang berinteraksi dengan berbagai komponen sistem, dan semakin banyak hak istimewa yang dimiliki aplikasi, semakin luas aksesnya ke komponen tersebut. Dengan membatasi izin kontainer hanya untuk apa yang diperlukan, PolP secara signifikan mengurangi permukaan serangan, sehingga lebih sulit bagi penyerang untuk mengeksploitasi kontainer jika menjadi terganggu.

Membatasi Dampak Kontainer yang Terganggu

- Jika wadah Windows dikompromikan, menjalankan aplikasi dengan hak istimewa yang berlebihan (misalnya, Administrator atau akses tingkat root) dapat memungkinkan penyerang untuk mendapatkan kontrol atas file sistem penting atau meningkatkan hak istimewa di seluruh host kontainer. Dengan menegakkan PolP, bahkan jika kontainer dilanggar, penyerang dibatasi dalam apa yang dapat mereka lakukan, mencegah eskalasi lebih lanjut dan akses ke sumber daya sensitif atau wadah lainnya.

Perlindungan di Lingkungan Multitenant

- Di lingkungan cloud atau perusahaan, beberapa kontainer dapat berjalan pada infrastruktur fisik atau virtual yang sama. PolP memastikan bahwa kontainer yang dikompromikan tidak memiliki kemampuan untuk mengakses sumber daya atau data milik penyewa lain. Isolasi ini sangat penting untuk menjaga keamanan di lingkungan bersama dan multitenant, melindungi dari pergerakan lateral antar kontainer.

Mengurangi Eskalasi Hak Istimewa

- Kontainer yang berjalan dengan hak istimewa tinggi dapat digunakan oleh penyerang untuk meningkatkan hak istimewa dalam sistem. PolP mengurangi risiko ini dengan membatasi akses kontainer ke sumber daya sistem, sehingga mencegah tindakan yang tidak sah atau eskalasi hak istimewa di luar lingkungan wadah.

Kepatuhan dan Audit

- Banyak standar peraturan dan kerangka kerja keamanan (misalnya, PCI DSS, HIPAA, GDPR) mengharuskan sistem untuk mematuhi PolP untuk membatasi akses ke data sensitif. Menjalankan wadah Windows dengan hak istimewa terbatas membantu organisasi mematuhi peraturan ini dan memastikan bahwa aplikasi hanya diberikan akses ke sumber daya yang mereka butuhkan secara khusus.

Mengurangi Risiko Kesalahan Konfigurasi

- Ketika kontainer berjalan dengan hak istimewa yang tidak perlu, bahkan kesalahan konfigurasi kecil dapat menyebabkan kerentanan keamanan yang parah. Misalnya, jika kontainer yang berjalan sebagai Administrator secara tidak sengaja terpapar ke internet, penyerang bisa mendapatkan kendali atas sistem. PolP membantu mencegah risiko tersebut dengan default ke hak istimewa terbatas, membuat kesalahan konfigurasi tidak terlalu berbahaya.

Postur Keamanan Kontainer yang Ditingkatkan

- Dengan mengikuti PolP, wadah lebih baik diisolasi dari sistem host yang mendasarinya dan dari satu sama lain. Ini memastikan bahwa aplikasi kontainer cenderung tidak mengakses atau memodifikasi file sistem atau proses di luar ruang lingkup yang ditentukan, menjaga integritas sistem operasi host dan beban kerja lainnya.

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
```

Penjelasan:

Pada bagian ini, ContainerUser perintah USER menentukan bahwa aplikasi di dalam wadah Windows harus berjalan di bawah ContainerUser akun alih-alih akun Administrator default.

Inilah mengapa ini penting, terutama di lingkungan multitenant:

1. Prinsip Hak Istimewa Terkecil: ContainerUser Akun adalah pengguna non-administratif dengan hak istimewa terbatas. Menjalankan aplikasi di bawah akun ini mematuhi prinsip hak istimewa paling sedikit, yang membantu meminimalkan risiko eksploitasi. Jika penyerang mengkompromikan aplikasi, mereka akan memiliki akses terbatas ke sistem, mengurangi potensi kerusakan.
2. Keamanan yang Ditingkatkan: Dalam lingkungan multitenant, kontainer dapat berbagi infrastruktur dasar yang sama. Menjalankan sebagai ContainerUser memastikan bahwa meskipun satu kontainer dikompromikan, itu tidak akan memiliki hak administratif untuk mengakses atau memodifikasi file sistem penting atau wadah lainnya. Ini mengurangi permukaan serangan secara signifikan.
3. Menghindari Akses Root: Secara default, kontainer mungkin berjalan dengan izin yang lebih tinggi (mirip dengan akses root di wadah Linux), yang bisa berbahaya jika dieksploitasi. Menggunakan ContainerUser memastikan bahwa aplikasi tidak berjalan dengan hak administratif yang tidak perlu, sehingga lebih sulit bagi penyerang untuk meningkatkan hak istimewa.
4. Praktik Terbaik untuk Lingkungan Multitenant: Dalam lingkungan di mana beberapa pengguna atau organisasi berbagi infrastruktur yang sama (seperti di cloud), keamanan sangat penting. Menjalankan aplikasi dengan izin terbatas mencegah aplikasi satu penyewa memengaruhi orang lain, melindungi data dan sumber daya sensitif di seluruh platform.

ContainerUserPerintah USER memastikan bahwa aplikasi berjalan dengan hak istimewa minimal, meningkatkan keamanan di lingkungan multitenant dengan membatasi kerusakan yang dapat dilakukan jika kontainer dikompromikan. Ini adalah praktik terbaik untuk mencegah akses tidak sah atau eskalasi hak istimewa di lingkungan peti kemas.

Prinsip Keistimewaan Terkecil sangat penting untuk wadah Windows karena membatasi potensi dampak pelanggaran keamanan, mengurangi permukaan serangan, dan mencegah akses tidak sah ke komponen sistem penting. Dengan menjalankan aplikasi kontainer dengan hanya izin yang diperlukan, organisasi dapat secara signifikan meningkatkan keamanan dan stabilitas lingkungan kontainer mereka, terutama dalam infrastruktur multitenant dan bersama.

Pikiran Akhir: Mengapa Mengamankan Wadah Windows Anda Harus Dimiliki di Lanskap Ancaman Hari Ini

Di dunia digital yang berkembang pesat saat ini, di mana ancaman menjadi lebih canggih dan berlimpah, mengamankan wadah Windows Anda bukan hanya rekomendasi, itu adalah kebutuhan mutlak. Kontainer menyediakan cara yang ringan dan fleksibel untuk mengemas dan menyebarkan aplikasi, tetapi mereka tidak kebal terhadap kerentanan keamanan. Karena semakin banyak bisnis mengadopsi kontainer untuk merampingkan infrastruktur mereka, mereka juga menjadi target potensial untuk serangan siber jika tidak diamankan dengan benar.

Internet dibanjiri berbagai ancaman — mulai dari aktor jahat yang menargetkan kerentanan yang belum ditambal hingga pemindaian bot otomatis untuk kesalahan konfigurasi. Tanpa langkah-langkah keamanan yang tepat, kontainer dapat dieksploitasi untuk mengekspos data sensitif, meningkatkan hak istimewa, atau berfungsi sebagai titik masuk untuk serangan yang dapat membahayakan infrastruktur Anda yang lebih luas. Ini membuat keamanan kontainer sama pentingnya dengan mengamankan bagian lain dari lingkungan Anda.

Saat menggunakan wadah Windows, banyak praktik terbaik keamanan tradisional masih berlaku. Menerapkan kebijakan akun yang kuat, mengamankan konfigurasi IIS, menegakkan HTTPS, menggunakan aturan firewall yang ketat, dan menerapkan akses hak istimewa paling sedikit ke file penting adalah semua langkah utama yang memastikan wadah tetap tangguh terhadap serangan. Selain itu, audit dan pencatatan rutin memberikan visibilitas tentang apa yang terjadi di dalam wadah, memungkinkan Anda untuk menangkap aktivitas yang mencurigakan sebelum berubah menjadi insiden besar-besaran.

Mengamankan wadah Windows juga sejalan dengan persyaratan peraturan yang mengamanatkan perlindungan data sensitif dan memastikan integritas aplikasi. Karena arsitektur cloud-native dan containerized menjadi lebih umum, memastikan keamanan di setiap lapisan, mulai dari image dasar hingga container yang sedang berjalan, akan membantu melindungi operasi Anda dan menjaga kepercayaan pelanggan.

Singkatnya, munculnya aplikasi dalam peti kemas, ditambah dengan meningkatnya jumlah ancaman dunia maya, menjadikan keamanan kontainer sebagai aspek yang tidak dapat dinegosiasikan dari manajemen infrastruktur modern. Dengan mengikuti praktik terbaik dan terus memantau kerentanan, bisnis dapat menikmati kelincahan dan efisiensi wadah Windows tanpa mengorbankan keamanan. Dalam lingkungan yang kaya ancaman ini, mengamankan wadah Windows Anda bukan hanya pilihan — itu harus dimiliki.

Praktik Terbaik untuk Penerapan Hybrid

Panduan ini memberikan panduan tentang menjalankan penerapan di lingkungan on-premise atau edge dengan EKS Hybrid Nodes atau EKS Anywhere.

Saat ini kami telah menerbitkan panduan untuk topik-topik berikut:

- Praktik Terbaik untuk Node Hibrid EKS dan pemutusan jaringan

EKS Hybrid Node dan pemutusan jaringan

Arsitektur EKS Hybrid Nodes dapat menjadi hal baru bagi pelanggan yang terbiasa menjalankan cluster Kubernetes lokal sepenuhnya di pusat data atau lokasi edge mereka sendiri. Dengan EKS Hybrid Nodes, control plane Kubernetes berjalan di Wilayah AWS dan hanya node yang berjalan di lokasi, menghasilkan arsitektur klaster Kubernetes yang “diregangkan” atau “diperpanjang”.

Ini mengarah ke pertanyaan umum, “Apa yang terjadi jika node saya terputus dari bidang kontrol Kubernetes?”

Dalam panduan ini, kami menjawab pertanyaan itu melalui tinjauan topik-topik berikut. Disarankan untuk memvalidasi stabilitas dan keandalan aplikasi Anda melalui pemutusan jaringan karena setiap aplikasi dapat berperilaku berbeda berdasarkan dependensi, konfigurasi, dan lingkungannya. Lihat eks-hybrid-examples GitHub aws-samples/repo untuk penyiapan pengujian, prosedur, dan hasil yang dapat Anda referensikan untuk menguji pemutusan jaringan dengan EKS Hybrid Nodes dan aplikasi Anda sendiri. GitHub Repo juga berisi detail tambahan dari tes yang digunakan untuk memvalidasi perilaku yang dijelaskan dalam panduan ini.

- Praktik terbaik untuk stabilitas melalui pemutusan jaringan
- Perilaku failover pod Kubernetes melalui pemutusan jaringan
- Lalu lintas jaringan aplikasi melalui pemutusan jaringan
- Kredensi host melalui pemutusan jaringan

Praktik terbaik untuk stabilitas melalui pemutusan jaringan

Jaringan yang sangat tersedia

Pendekatan terbaik untuk menghindari pemutusan jaringan antara node hybrid dan bidang kontrol Kubernetes adalah dengan menggunakan koneksi yang berlebihan dan tangguh dari lingkungan lokal Anda ke dan dari AWS. Lihat AWS Direct Connect Resiliency Toolkit dan dokumentasi AWS Site-to-Site VPN untuk informasi selengkapnya tentang merancang jaringan hybrid yang sangat tersedia dengan solusi tersebut.

Aplikasi yang sangat tersedia

Saat merancang aplikasi, pertimbangkan domain kegagalan Anda dan efek dari berbagai jenis pemadaman. Kubernetes menyediakan mekanisme bawaan untuk menyebarkan dan memelihara replika aplikasi di seluruh domain node, zona, dan regional. Penggunaan mekanisme ini bergantung pada arsitektur aplikasi, lingkungan, dan persyaratan ketersediaan Anda. Misalnya, aplikasi stateless sering dapat digunakan dengan beberapa replika dan dapat berpindah melintasi host arbitrer dan kapasitas infrastruktur, dan Anda dapat menggunakan pemilih node dan batasan penyebaran topologi untuk menjalankan instance aplikasi di berbagai domain. Untuk detail teknik tingkat aplikasi untuk membangun aplikasi tangguh di Kubernetes, lihat Panduan Praktik Terbaik EKS.

Kubernetes mengevaluasi informasi zonal untuk node yang terputus dari bidang kontrol Kubernetes saat menentukan apakah akan memindahkan pod ke node lain. Jika semua node di zona tidak dapat dijangkau, Kubernetes membatalkan penggusuran pod untuk node di zona tersebut. Sebagai praktik terbaik, jika Anda memiliki penyebaran dengan node yang berjalan di beberapa pusat data atau lokasi fisik, tetapkan zona ke setiap node berdasarkan pusat data atau lokasi fisiknya. Saat Anda menjalankan EKS dengan node di cloud, label zona ini diterapkan secara otomatis oleh AWS cloud-controller-manager. Namun, a tidak cloud-controller-manager digunakan dengan node hybrid, sehingga Anda dapat meneruskan informasi ini melalui konfigurasi kubelet Anda. Contoh cara mengkonfigurasi zona dalam konfigurasi node Anda untuk node hybrid ditunjukkan di bawah ini. Konfigurasi diteruskan ketika Anda menghubungkan node hybrid Anda ke cluster Anda dengan node hybrid CLI ()`nodeadm`. Untuk informasi lebih lanjut tentang `topology.kubernetes.io/zone` label, lihat dokumentasi Kubernetes. Untuk informasi lebih lanjut tentang CLI node hybrid, lihat referensi nodeadm Hybrid Nodes.

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
```

```
  cluster:
    name: my-cluster
    region: my-region
  kubelet:
    flags:
       - --node-labels=topology.kubernetes.io/zone=dc1
  hybrid:
    ...
```

Pemantauan jaringan

Jika Anda menggunakan AWS Direct Connect atau AWS Site-to-Site VPN untuk konektivitas hybrid, Anda dapat memanfaatkan CloudWatch alarm, log, dan metrik untuk mengamati status koneksi hybrid dan mendiagnosis masalah. Untuk informasi selengkapnya, lihat Memantau sumber daya AWS Direct Connect dan Memantau koneksi AWS Site-to-Site VPN.

Disarankan untuk membuat alarm untuk `NodeNotReady` peristiwa yang dilaporkan oleh node-lifecycle-controller berjalan pada bidang kontrol EKS, yang menandakan bahwa node hibrida mungkin mengalami pemutusan jaringan. Anda dapat membuat alarm ini dengan mengaktifkan pencatatan bidang kontrol EKS untuk Controller Manager dan membuat Filter Metrik CloudWatch untuk pesan “Merekam pesan peristiwa perubahan status untuk node” dengan status= “”. NodeNotReady Setelah membuat Filter Metrik, Anda dapat membuat alarm untuk filter ini berdasarkan ambang batas yang Anda inginkan. Untuk informasi selengkapnya, lihat Mengkhawatirkan log dalam dokumentasi. CloudWatch

Anda dapat menggunakan metrik bawaan Transit Gateway (TGW) dan Virtual Private Gateway (VGW) untuk mengamati lalu lintas jaringan masuk dan keluar dari TGW atau VGW Anda. Anda dapat membuat alarm untuk metrik ini untuk mendeteksi skenario di mana lalu lintas jaringan turun di bawah level normal, yang menunjukkan potensi masalah jaringan antara node hibrida dan bidang kontrol EKS. Metrik TGW dan VGW dijelaskan dalam tabel berikut.

Gateway	Metrik	Deskripsi
Transit Gateway	BytesIn	Byte yang diterima oleh TGW dari lampiran (bidang kontrol EKS ke node hibrida)
Transit Gateway	BytesOut	Byte dikirim dari TGW ke lampiran (node hibrida ke bidang kontrol EKS)

Gateway	Metrik	Deskripsi
Gerbang Pribadi Virtual	TunnelDataIn	Byte yang dikirim dari sisi AWS koneksi melalui terowongan VPN ke gateway pelanggan (bidang kontrol EKS ke node hibrida)
Gerbang Pribadi Virtual	TunnelDataOut	Byte yang diterima di sisi AWS koneksi melalui terowongan VPN dari gateway pelanggan (node hybrid ke bidang kontrol EKS)

Anda juga dapat menggunakan CloudWatch Network Monitor untuk mendapatkan wawasan yang lebih dalam tentang koneksi hybrid Anda guna mengurangi waktu pemulihan dan menentukan apakah masalah jaringan berasal dari AWS atau lingkungan Anda. CloudWatch Network Monitor dapat digunakan untuk memvisualisasikan kehilangan paket dan latensi dalam koneksi jaringan hybrid Anda, mengatur peringatan dan ambang batas, dan kemudian mengambil tindakan untuk meningkatkan kinerja jaringan Anda. Untuk informasi selengkapnya, lihat Menggunakan Monitor CloudWatch Jaringan Amazon.

EKS menawarkan beberapa opsi untuk memantau kesehatan cluster dan aplikasi Anda. Untuk kesehatan klaster, Anda dapat menggunakan dasbor observabilitas di konsol EKS untuk mendeteksi, memecahkan masalah, dan memperbaiki masalah dengan cepat. Anda juga dapat menggunakan Amazon Managed Service untuk Prometheus, AWS Distro for Open Telemetry (ADOT), dan untuk pemantauan klaster, aplikasi CloudWatch , dan infrastruktur. Untuk informasi selengkapnya tentang opsi observabilitas EKS, lihat Memantau kinerja klaster Anda dan melihat log.

Pemecahan masalah lokal

Untuk mempersiapkan pemutusan jaringan antara node hibrida dan bidang kontrol EKS, Anda dapat menyiapkan backend pemantauan dan pencatatan sekunder untuk mempertahankan observabilitas aplikasi saat layanan AWS regional tidak dapat dijangkau. Misalnya, Anda dapat mengonfigurasi kolektor AWS Distro for Open Telemetry (ADOT) untuk mengirim metrik dan log ke beberapa backend. Anda juga dapat menggunakan alat lokal, seperti `crictl` CLI, untuk berinteraksi secara lokal dengan pod dan kontainer sebagai pengganti `kubectl` atau klien lain yang kompatibel dengan API Kubernetes yang biasanya menanyakan endpoint server API Kubernetes. Untuk informasi lebih lanjut tentang`crictl`, lihat `crictl`dokumentasi di cri-tools GitHub. Beberapa `crictl` perintah yang berguna tercantum di bawah ini.

Daftar pod yang berjalan di host:

```
crictl pods
```

Daftar kontainer yang berjalan di host:

```
crictl ps
```

Daftar gambar yang berjalan di host:

```
crictl images
```

Dapatkan log wadah yang berjalan di host:

```
crictl logs CONTAINER_NAME
```

Dapatkan statistik pod yang berjalan di host:

```
crictl statsp
```

Lalu lintas jaringan aplikasi

Saat menggunakan node hybrid, penting untuk mempertimbangkan dan memahami arus jaringan lalu lintas aplikasi Anda dan teknologi yang Anda gunakan untuk mengekspos aplikasi Anda secara eksternal ke cluster Anda. Teknologi yang berbeda untuk penyeimbangan beban aplikasi dan masuknya berperilaku berbeda selama pemutusan jaringan. Misalnya, jika Anda menggunakan kemampuan BGP Control Plane Cilium untuk load balancing aplikasi, sesi BGP untuk pod dan layanan Anda mungkin tidak aktif selama pemutusan jaringan. Ini terjadi karena fungsi speaker BGP terintegrasi dengan agen Cilium, dan agen Cilium akan terus restart ketika terputus dari bidang kontrol Kubernetes. Alasan restart adalah karena pemeriksaan kesehatan Cilium gagal karena kesehatannya digabungkan dengan akses ke bidang kontrol Kubernetes (lihat CFP: #31702 dengan peningkatan keikutsertaan di Cilium v1.17). Demikian pula, jika Anda menggunakan Application Load Balancers (ALB) atau Network Load Balancers (NLB) untuk lalu lintas aplikasi yang berasal dari AWS Region, lalu lintas tersebut mungkin akan turun sementara jika lingkungan lokal Anda kehilangan konektivitas ke Wilayah AWS. Disarankan untuk memvalidasi bahwa teknologi yang Anda gunakan untuk load balancing dan ingress tetap stabil selama pemutusan jaringan sebelum diterapkan ke produksi. Contoh di aws-samples/ eks-hybrid-examples GitHub repo menggunakan MetalLB untuk penyeimbangan beban dalam mode L2, yang tetap stabil selama pemutusan jaringan antara node hibrida dan bidang kontrol EKS.

Tinjau dependensi pada layanan AWS jarak jauh

Saat menggunakan node hybrid, perhatikan dependensi yang Anda ambil pada layanan AWS regional yang berada di luar lingkungan lokal atau edge Anda. Contohnya termasuk mengakses Amazon S3 atau Amazon RDS untuk data aplikasi, menggunakan Amazon Managed Service untuk Prometheus atau untuk metrik dan log, menggunakan Application and Network Load Balancers untuk CloudWatch lalu lintas yang berasal dari Wilayah, dan menarik kontainer dari Amazon Elastic Container Registry. Layanan ini tidak akan dapat diakses selama pemutusan jaringan antara lingkungan lokal Anda dan AWS. Jika lingkungan lokal Anda rentan terhadap pemutusan jaringan dengan AWS, tinjau penggunaan layanan AWS Anda dan pastikan bahwa kehilangan koneksi ke layanan tersebut tidak mengganggu stabilitas statis aplikasi Anda.

Tune perilaku failover pod Kubernetes

Ada opsi untuk menyetel perilaku failover pod selama pemutusan jaringan untuk aplikasi yang tidak portabel di seluruh host, atau untuk lingkungan terbatas sumber daya yang tidak memiliki kapasitas cadangan untuk failover pod. Secara umum, penting untuk mempertimbangkan persyaratan sumber daya aplikasi Anda dan memiliki kapasitas yang cukup untuk satu atau lebih contoh aplikasi gagal ke host yang berbeda jika node gagal.

- Opsi 1 - Gunakan DaemonSets: Opsi ini berlaku untuk aplikasi yang dapat dan harus berjalan di semua node di cluster. DaemonSets secara otomatis dikonfigurasi untuk mentolerir taint yang tidak dapat dijangkau, yang membuat DaemonSet pod terikat ke node mereka melalui pemutusan jaringan.
- Opsi 2 - Tune `tolerationSeconds` for unreachable taint: Anda dapat menyetel jumlah waktu pod Anda tetap terikat ke node selama pemutusan jaringan. Lakukan ini dengan mengonfigurasi pod aplikasi untuk mentolerir taint yang tidak dapat dijangkau dengan `NoExecute` efek selama durasi yang Anda tentukan (`tolerationSeconds`dalam spesifikasi aplikasi). Dengan opsi ini, ketika ada pemutusan jaringan, pod aplikasi Anda tetap terikat ke node hingga `tolerationSeconds` kedaluwarsa. Pertimbangkan hal ini dengan cermat, karena peningkatan `tolerationSeconds` untuk noda yang tidak dapat dijangkau dengan `NoExecute` berarti bahwa pod yang berjalan pada host yang tidak dapat dijangkau mungkin membutuhkan waktu lebih lama untuk pindah ke host lain yang sehat dan dapat dijangkau.
- Opsi 3: Pengontrol kustom: Anda dapat membuat dan menjalankan pengontrol kustom (atau perangkat lunak lain) yang memantau Kubernetes untuk noda yang tidak dapat dijangkau dengan efeknya. `NoExecute` Ketika noda ini terdeteksi, pengontrol khusus dapat memeriksa metrik khusus aplikasi untuk menilai kesehatan aplikasi. Jika aplikasi sehat, pengontrol kustom dapat

menghapus taint yang tidak dapat dijangkau, mencegah penggusuran pod dari node selama pemutusan jaringan.

Contoh cara mengonfigurasi Deployment dengan `tolerationSeconds` untuk taint yang tidak dapat dijangkau ditunjukkan di bawah ini. Dalam contoh, `tolerationSeconds` diatur ke `1800` (30 menit), yang berarti pod yang berjalan pada node yang tidak dapat dijangkau hanya akan diusir jika pemutusan jaringan berlangsung lebih dari 30 menit.

```
apiVersion: apps/v1
kind: Deployment
metadata:
...
spec:
...
      tolerations:
      - key: "node.kubernetes.io/unreachable"
        operator: "Exists"
        effect: "NoExecute"
        tolerationSeconds: 1800
```

Kubernetes pod failover melalui pemutusan jaringan

Kita mulai dengan meninjau konsep kunci, komponen, dan pengaturan yang mempengaruhi bagaimana Kubernetes berperilaku selama pemutusan jaringan antara node dan bidang kontrol Kubernetes. EKS adalah kesesuaian Kubernetes hulu, sehingga semua konsep, komponen, dan pengaturan Kubernetes yang dijelaskan di sini berlaku untuk penerapan EKS dan EKS Hybrid Nodes.

Ada perbaikan yang telah dilakukan pada EKS secara khusus untuk meningkatkan perilaku failover pod selama pemutusan jaringan, untuk informasi selengkapnya lihat GitHub masalah #131294 dan #131481 di repositori Kubernetes hulu.

Konsep

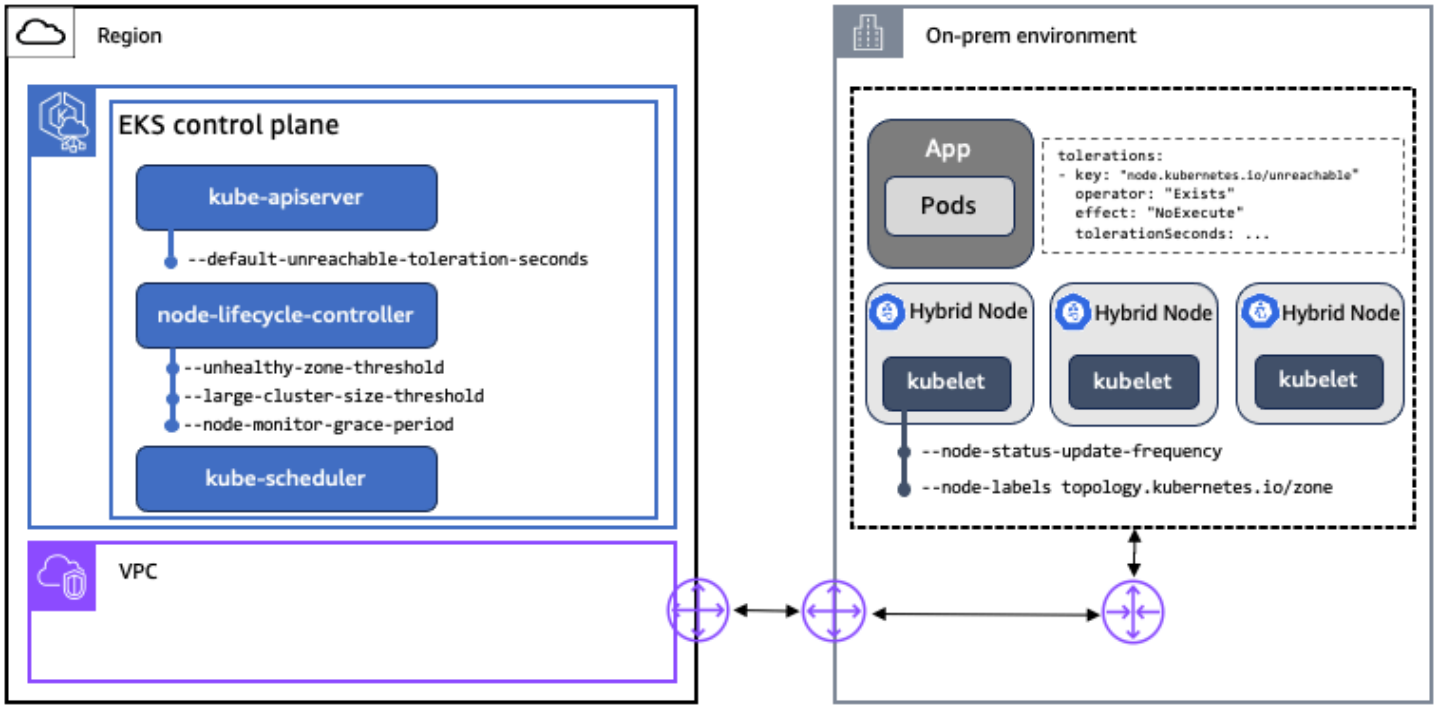
Taints and Tolerations: Taints dan tolerations digunakan di Kubernetes untuk mengontrol penjadwalan pod ke node. Taints ditetapkan oleh node-lifecycle-controller untuk menunjukkan bahwa node tidak memenuhi syarat untuk penjadwalan atau bahwa pod pada node tersebut harus diusir. Ketika node tidak dapat dijangkau karena pemutusan jaringan, node akan diterapkan node.kubernetes. node-lifecycle-controller io/unreachable taint with a NoSchedule effect, and with a NoExecute effect if certain conditions are met. The node.kubernetes.io/unreachabletaint sesuai

dengan NodeCondition Ready being Unknown. Pengguna dapat menentukan toleransi untuk noda di tingkat aplikasi di. PodSpec

- NoSchedule: Tidak ada Pod baru yang dijadwalkan pada node yang tercemar kecuali mereka memiliki toleransi yang cocok. Pod yang sudah berjalan di node tidak diusir.
- NoExecute: Pod yang tidak mentolerir noda segera diusir. Pod yang mentolerir taint (tanpa menentukan TolerationSeconds) tetap terikat selamanya. Pod yang mentolerir taint dengan TolerationSeconds tertentu tetap terikat untuk waktu yang ditentukan. Setelah waktu itu berlalu, pengontrol siklus hidup node mengeluarkan Pod dari node.

Node Leases: Kubernetes menggunakan Lease API untuk mengkomunikasikan detak jantung node kubelet ke server API Kubernetes. Untuk setiap node, ada objek Lease dengan nama yang cocok. Secara internal, setiap detak jantung kubelet memperbarui bidang spec.renewTime dari objek Lease. Bidang kontrol Kubernetes menggunakan stempel waktu bidang ini untuk menentukan ketersediaan node. Jika node terputus dari bidang kontrol Kubernetes, mereka tidak dapat memperbarui Spec.renewTime untuk Lease mereka, dan control plane menafsirkannya sebagai Ready being Unknown. NodeCondition

Komponen-komponen



Komponen	Sub-komponen	Deskripsi
Pesawat kontrol Kubernetes	kube-api-server	Server API adalah komponen inti dari bidang kontrol Kubernetes yang mengekspos API Kubernetes.
Pesawat kontrol Kubernetes	node-lifecycle-controller	Salah satu pengontrol yang kube-controller-manager dijalankan. Ini bertanggung jawab untuk mendeteksi dan menanggapi masalah node.
Pesawat kontrol Kubernetes	penjadwal kube	Komponen control plane yang mengawasi Pod yang baru dibuat tanpa node yang ditetapkan, dan memilih node untuk dijalankan.
Node Kubernetes	kubelet	Agen yang berjalan pada setiap node di cluster. Kubelet mengawasi PodSpecs dan memastikan bahwa wadah yang dijelaskan di dalamnya berjalan dan sehat. PodSpecs

Pengaturan konfigurasi

Kompor	Pengaturan	Deskripsi	K8s standar	EKS standar	Dapat dikonfigurasi di EKS
kube-api-server	default-unreachable-toleration-seconds	`tolerationSeconds` Menunjukkan toleransi untuk `unreachable:NoExecute` itu ditambahkan secara default ke setiap pod yang belum memiliki toleransi seperti itu.	300	300	Tidak
node-lifecycle-controller	node-monitor-grace-period	Jumlah waktu node dapat menjadi tidak responsif sebelum ditandai tidak sehat. Harus N kali lebih banyak dari kubeletnode`StatusUpdateFrequency` , di mana N adalah jumlah	40	40	Tidak

Kompor	Pengaturan	Deskripsi	K8s standar	EKS standar	Dapat dikonfigurasi di EKS
		percobaan ulang yang diizinkan untuk kubelet untuk memposting status node.			
node-life cycle-con troller	large-cluster-size-threshold	Jumlah node di mana cluster node-life cycle-controller memperlakukan cluster sebagai besar untuk logika penggusur an. `--secondary-node-eviction-rate` diganti menjadi 0 untuk cluster dengan ukuran ini atau lebih kecil.	50	100.000	Tidak
node-life cycle-con troller	unhealthy-zone-threshold	Persentase node di zona yang harus Tidak Siap untuk zona itu diperlakukan sebagai tidak sehat.	55%	55%	Tidak
kubelet	node-stat us-update-frequency	Seberapa sering kubelet mempostin g status node ke bidang kontrol. Harus kompatibel dengan `nodeMonit orGracePeriod` in node-lifecycle-con troller.	10	10	Ya
kubelet	label simpul	Label untuk ditambahkan saat mendaftar kan node di cluster. Label `topology. kubernetes.io/zone` dapat ditentukan dengan node hybrid untuk mengelompokkan node ke dalam zona.	Tidak ada	Tidak ada	Ya

Kubernetes pod failover melalui pemutusan jaringan

Perilaku yang dijelaskan di sini mengasumsikan pod berjalan sebagai Deployment Kubernetes dengan pengaturan default, dan EKS digunakan sebagai penyedia Kubernetes. Perilaku aktual mungkin berbeda berdasarkan lingkungan Anda, jenis pemutusan jaringan, aplikasi, dependensi, dan

konfigurasi cluster. Konten dalam panduan ini divalidasi menggunakan aplikasi tertentu, konfigurasi cluster, dan subset plugin. Sangat disarankan untuk menguji perilaku di lingkungan Anda sendiri dan dengan aplikasi Anda sendiri sebelum pindah ke produksi.

Ketika ada pemutusan jaringan antara node dan bidang kontrol Kubernetes, kubelet pada setiap node yang terputus tidak dapat berkomunikasi dengan bidang kontrol Kubernetes. Akibatnya, kubelet tidak dapat mengusir pod pada node tersebut sampai koneksi dipulihkan. Ini berarti bahwa pod yang berjalan pada node tersebut sebelum pemutusan jaringan terus berjalan selama pemutusan, dengan asumsi tidak ada kegagalan lain yang menyebabkannya dimatikan. Singkatnya, Anda dapat mencapai stabilitas statis selama pemutusan jaringan antara node dan bidang kontrol Kubernetes, tetapi Anda tidak dapat melakukan operasi mutasi pada node atau beban kerja Anda sampai koneksi dipulihkan.

Ada lima skenario utama yang menghasilkan perilaku failover pod yang berbeda berdasarkan sifat pemutusan jaringan. Dalam semua skenario, cluster menjadi sehat kembali tanpa intervensi operator setelah node terhubung kembali ke bidang kontrol Kubernetes. Skenario di bawah menguraikan hasil yang diharapkan berdasarkan pengamatan kami, tetapi hasil ini mungkin tidak berlaku untuk semua kemungkinan aplikasi dan konfigurasi cluster.

Skenario 1: Gangguan cluster penuh

Hasil yang diharapkan: Pod pada node yang tidak dapat dijangkau tidak diusir dan terus berjalan pada node tersebut.

Gangguan cluster penuh berarti semua node di cluster terputus dari bidang kontrol Kubernetes. Dalam skenario ini, node-lifecycle-controller on the control plane mendeteksi bahwa semua node di cluster tidak dapat dijangkau dan membatalkan penggusuran pod apa pun.

Administrator cluster akan melihat semua node dengan status `Not Ready` selama pemutusan. Status Pod tidak berubah, dan tidak ada pod baru yang dijadwalkan pada node manapun selama pemutusan dan penyambungan ulang berikutnya.

Skenario 2: Gangguan zona penuh

Hasil yang diharapkan: Pod pada node yang tidak dapat dijangkau tidak diusir dan terus berjalan pada node tersebut.

Gangguan zona penuh berarti semua node di zona terputus dari bidang kontrol Kubernetes. Dalam skenario ini, node-lifecycle-controller on the control plane mendeteksi bahwa semua node di zona tidak dapat dijangkau dan membatalkan penggusuran pod apa pun.

Administrator cluster akan melihat semua node dengan status `Not Ready` selama pemutusan. Status Pod tidak berubah, dan tidak ada pod baru yang dijadwalkan pada node manapun selama pemutusan dan penyambungan ulang berikutnya.

Skenario 3: Gangguan zona mayoritas

Hasil yang diharapkan: Pod pada node yang tidak dapat dijangkau tidak diusir dan terus berjalan pada node tersebut.

Gangguan zona mayoritas berarti bahwa sebagian besar node di zona tertentu terputus dari bidang kontrol Kubernetes. Zona di Kubernetes didefinisikan oleh node dengan label yang sama. `topology.kubernetes.io/zone` Jika tidak ada zona yang didefinisikan dalam cluster, gangguan mayoritas berarti mayoritas node di seluruh cluster terputus. Secara default, mayoritas didefinisikan oleh node-lifecycle-controller'sun`healthy-zone-threshold`, yang disetel ke 55% di Kubernetes dan EKS. Karena `large-cluster-size-threshold` disetel ke 100.000 di EKS, jika 55% atau lebih dari node di zona tidak dapat dijangkau, penggusuran pod dibatalkan (mengingat bahwa sebagian besar cluster jauh lebih kecil dari 100.000 node).

Administrator klaster akan melihat mayoritas node di zona dengan status `Not Ready` selama pemutusan, tetapi status pod tidak akan berubah, dan mereka tidak akan dijadwalkan ulang pada node lain.

Perhatikan bahwa perilaku di atas hanya berlaku untuk cluster yang lebih besar dari tiga node. Dalam klaster yang terdiri dari tiga node atau kurang, pod pada node yang tidak dapat dijangkau dijadwalkan untuk penggusuran, dan pod baru dijadwalkan pada node yang sehat.

Selama pengujian, kami kadang-kadang mengamati bahwa pod diusir dari tepat satu node yang tidak dapat dijangkau selama pemutusan jaringan, bahkan ketika sebagian besar node zona tidak dapat dijangkau. Kami masih menyelidiki kemungkinan kondisi balapan di Kubernetes node-lifecycle-controller sebagai penyebab perilaku ini.

Skenario 4: Gangguan zona minoritas

Hasil yang diharapkan: Pod dikeluarkan dari node yang tidak dapat dijangkau, dan pod baru dijadwalkan pada node yang tersedia dan memenuhi syarat.

Gangguan minoritas berarti bahwa persentase node yang lebih kecil di zona terputus dari bidang kontrol Kubernetes. Jika tidak ada zona yang didefinisikan dalam cluster, gangguan minoritas berarti minoritas node di seluruh cluster terputus. Sebagaimana dinyatakan, minoritas didefinisikan oleh

`unhealthy-zone-threshold` pengaturan node-lifecycle-controller, yaitu 55% secara default. Dalam skenario ini, jika pemutusan jaringan berlangsung lebih lama dari `default-unreachable-toleration-seconds` (5 menit) dan `node-monitor-grace-period` (40 detik), dan kurang dari 55% node di zona tidak dapat dijangkau, pod baru dijadwalkan pada node sehat sementara pod pada node yang tidak dapat dijangkau ditandai untuk penggusuran.

Administrator klaster akan melihat pod baru yang dibuat pada node yang sehat, dan pod pada node yang terputus akan ditampilkan sebagai. `Terminating` Ingatlah bahwa, meskipun Pod pada node yang terputus memiliki `Terminating` status, mereka tidak sepenuhnya diusir sampai node terhubung kembali ke bidang kontrol Kubernetes.

Skenario 5: Node restart selama gangguan jaringan

Hasil yang diharapkan: Pod pada node yang tidak dapat dijangkau tidak dimulai sampai node terhubung kembali ke bidang kontrol Kubernetes. Failover Pod mengikuti logika yang dijelaskan dalam Skenario 1-3, tergantung pada jumlah node yang tidak dapat dijangkau.

Node restart selama gangguan jaringan berarti bahwa kegagalan lain (seperti siklus daya, out-of-memory peristiwa, atau masalah lain) terjadi pada node pada saat yang sama dengan pemutusan jaringan. Pod yang berjalan pada node tersebut ketika pemutusan jaringan dimulai tidak secara otomatis dimulai ulang selama pemutusan sambungan jika kubelet juga telah dimulai ulang. Kubelet menanyakan server API Kubernetes selama startup untuk mempelajari pod mana yang harus dijalankan. Jika kubelet tidak dapat mencapai server API karena pemutusan jaringan, kubelet tidak dapat mengambil informasi yang diperlukan untuk memulai pod.

Dalam skenario ini, alat pemecahan masalah lokal seperti `crictl` CLI tidak dapat digunakan untuk memulai pod secara manual sebagai ukuran “break-glass”. Kubernetes biasanya menghapus pod yang gagal dan membuat yang baru daripada memulai ulang pod yang ada (lihat #10213 di repo containerd untuk detailnya). GitHub Pod statis adalah satu-satunya objek beban kerja Kubernetes yang dikendalikan oleh kubelet dan dapat di-restart selama skenario ini. Namun, umumnya tidak disarankan untuk menggunakan pod statis untuk penerapan aplikasi. Sebagai gantinya, gunakan beberapa replika di host yang berbeda untuk memastikan ketersediaan aplikasi jika terjadi beberapa kegagalan simultan, seperti kegagalan node ditambah pemutusan jaringan antara node Anda dan bidang kontrol Kubernetes.

Lalu lintas jaringan aplikasi melalui pemutusan jaringan

Topik pada halaman ini terkait dengan jaringan klaster Kubernetes dan lalu lintas aplikasi selama pemutusan jaringan antara node dan bidang kontrol Kubernetes.

Cilium

Cilium memiliki beberapa mode untuk IP address management (IPAM), enkapsulasi, load balancing, dan cluster routing. Mode yang divalidasi dalam panduan ini menggunakan Cluster Scope IPAM, overlay VXLAN, load balancing BGP, dan kube-proxy. Cilium juga digunakan tanpa load balancing BGP, menggantinya dengan load balancing MetalLB L2.

Basis pemasangan Cilium terdiri dari operator Cilium dan agen Cilium. Operator Cilium berjalan sebagai Deployment dan mendaftarkan Cilium Custom Resource Definitions (CRDs), mengelola IPAM, dan menyinkronkan objek cluster dengan server API Kubernetes di antara kemampuan lainnya. Agen Cilium berjalan pada setiap node sebagai DaemonSet dan mengelola program eBPF untuk mengontrol aturan jaringan untuk beban kerja yang berjalan di cluster.

Umumnya, routing in-cluster yang dikonfigurasi oleh Cilium tetap tersedia dan di tempat selama pemutusan jaringan, yang dapat dikonfirmasi dengan mengamati arus lalu lintas in-cluster dan aturan tabel IP (iptables) untuk jaringan 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
...
```

Namun, selama pemutusan jaringan, operator Cilium dan agen Cilium memulai ulang karena penggabungan pemeriksaan kesehatan mereka dengan kesehatan koneksi dengan server API Kubernetes. Diharapkan untuk melihat hal berikut di log operator Cilium dan agen Cilium selama pemutusan jaringan. Selama pemutusan jaringan, Anda dapat menggunakan alat seperti `crictl` CLI untuk mengamati restart komponen ini termasuk lognya.

```
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
```

Jika Anda menggunakan kemampuan BGP Control Plane Cilium untuk load balancing aplikasi, sesi BGP untuk pod dan layanan Anda mungkin tidak aktif selama pemutusan jaringan karena fungsi speaker BGP terintegrasi dengan agen Cilium, dan agen Cilium akan terus restart ketika terputus dari bidang kontrol Kubernetes. Untuk informasi lebih lanjut, lihat Panduan Operasi Pesawat Kontrol Cilium BGP di dokumentasi Cilium. Selain itu, jika Anda mengalami kegagalan simultan selama pemutusan jaringan seperti siklus daya atau reboot mesin, rute Cilium tidak akan dipertahankan melalui tindakan ini, meskipun rute dibuat ulang ketika node terhubung kembali ke bidang kontrol Kubernetes dan Cilium dimulai lagi.

Calico

Segera hadir

MetalB

MetalLB memiliki dua mode untuk penyeimbangan beban: mode L2 dan mode BGP. Referensikan dokumentasi MetalLB untuk detail tentang cara kerja mode load balancing ini dan keterbatasannya. Validasi untuk panduan ini menggunakan MetalLB dalam mode L2, di mana satu mesin di klaster mengambil kepemilikan Layanan Kubernetes, dan menggunakan ARP IPv4 untuk membuat alamat IP penyeimbang beban dapat dijangkau di jaringan lokal. Saat menjalankan MetalLB ada pengontrol yang bertanggung jawab atas penugasan IP dan speaker yang berjalan pada setiap node yang bertanggung jawab untuk layanan iklan dengan alamat IP yang ditetapkan. Pengontrol MetalLB berjalan sebagai Deployment dan speaker MetalLB berjalan sebagai file. DaemonSet Selama pemutusan jaringan, pengontrol dan speaker MetalLB gagal menonton server API Kubernetes untuk sumber daya klaster tetapi terus berjalan. Yang terpenting, Layanan yang menggunakan MetalLB untuk konektivitas eksternal tetap tersedia dan dapat diakses selama pemutusan jaringan.

kube-proxy

Dalam kluster EKS, kube-proxy berjalan sebagai a DaemonSet pada setiap node dan bertanggung jawab untuk mengelola aturan jaringan untuk mengaktifkan komunikasi antara layanan dan pod dengan menerjemahkan alamat IP layanan ke alamat IP dari pod yang mendasarinya. Aturan IP tables (iptables) yang dikonfigurasi oleh kube-proxy dipertahankan selama pemutusan jaringan dan routing in-cluster terus berfungsi dan pod kube-proxy terus berjalan.

Anda dapat mengamati aturan kube-proxy dengan perintah iptables berikut. Perintah pertama menunjukkan paket melalui `PREROUTING` rantai diarahkan ke `KUBE-SERVICES` rantai.

```
iptables -t nat -L PREROUTING
```

```
Chain PREROUTING (policy ACCEPT)
target         prot opt source      destination
KUBE-SERVICES  all  --  anywhere    anywhere      /* kubernetes service portals */
```

Memeriksa `KUBE-SERVICES` rantai kita dapat melihat aturan untuk berbagai layanan 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 */
```

Memeriksa rantai layanan frontend untuk aplikasi, kita dapat melihat alamat IP pod mendukung layanan.

```
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 */
```

Pesan log kube-proxy berikut diharapkan selama pemutusan jaringan saat mencoba menonton server API Kubernetes untuk pembaruan ke sumber daya node dan titik akhir.

```
"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

Secara default, pod di kluster EKS menggunakan alamat IP cluster CoreDNS sebagai server nama untuk kueri DNS in-cluster. Di kluster EKS, CoreDNS berjalan sebagai Deployment pada node. Dengan node hybrid, pod dapat terus berkomunikasi dengan CoreDNS selama pemutusan jaringan ketika ada replika CoreDNS yang berjalan secara lokal pada node hybrid. Jika Anda memiliki kluster EKS dengan node di cloud dan node hybrid di lingkungan lokal Anda, disarankan untuk memiliki setidaknya satu replika CoreDNS di setiap lingkungan. CoreDNS terus melayani kueri DNS untuk

catatan yang dibuat sebelum pemutusan jaringan dan terus berjalan melalui koneksi ulang jaringan untuk stabilitas statis.

Pesan log CoreDNS berikut diharapkan selama pemutusan jaringan saat mencoba membuat daftar objek dari server 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
```

Kredensi host melalui pemutusan jaringan

EKS Hybrid Nodes terintegrasi dengan aktivasi hybrid AWS Systems Manager (SSM) dan AWS IAM Roles Anywhere untuk kredensyal IAM sementara yang digunakan untuk mengautentikasi node dengan bidang kontrol EKS. Peran SSM dan IAM Anywhere secara otomatis menyegarkan kredensyal sementara yang mereka kelola di host lokal. Disarankan untuk menggunakan penyedia kredensyal tunggal di seluruh node hibrida di cluster Anda—baik aktivasi hibrida SSM atau Peran IAM Di Mana Saja, tetapi tidak keduanya.

Aktivasi hibrida SSM

Kredensyal sementara yang disediakan oleh SSM berlaku selama satu jam. Anda tidak dapat mengubah durasi validitas kredensyal saat menggunakan SSM sebagai penyedia kredensi Anda. Kredensyal sementara secara otomatis diputar oleh SSM sebelum kedaluwarsa, dan rotasi tidak mempengaruhi status node atau aplikasi Anda. Namun, ketika ada pemutusan jaringan antara agen SSM dan titik akhir Regional SSM, SSM tidak dapat menyegarkan kredensialnya, dan kredensialnya mungkin kedaluwarsa.

SSM menggunakan backoff eksponensial untuk percobaan ulang penyegaran kredenal jika tidak dapat terhubung ke titik akhir Regional SSM. Dalam versi agen SSM `3.3.808.0` dan yang lebih baru (dirilis Agustus 2024), backoff eksponensial dibatasi pada 30 menit. Bergantung pada durasi pemutusan jaringan Anda, SSM mungkin memerlukan waktu hingga 30 menit untuk menyegarkan kredensialnya, dan node hibrida tidak akan terhubung kembali ke bidang kontrol EKS sampai

kredensialnya diperbarui. Dalam skenario ini, Anda dapat memulai ulang agen SSM untuk memaksa penyegaran kredenal. Sebagai efek samping dari perilaku penyegaran kredenal SSM saat ini, node mungkin terhubung kembali pada waktu yang berbeda tergantung pada kapan agen SSM pada setiap node berhasil menyegarkan kredensialnya. Karena itu, Anda mungkin melihat pod failover dari node yang belum terhubung kembali ke node yang sudah terhubung kembali.

Dapatkan versi agen SSM. Anda juga dapat memeriksa bagian Fleet Manager di konsol SSM:

```
# AL2023, RHEL
yum info amazon-ssm-agent
# Ubuntu
snap list amazon-ssm-agent
```

Mulai ulang agen SSM:

```
# AL2023, RHEL
systemctl restart amazon-ssm-agent
# Ubuntu
systemctl restart snap.amazon-ssm-agent.amazon-ssm-agent
```

Lihat log agen SSM:

```
tail -f /var/log/amazon/ssm/amazon-ssm-agent.log
```

Pesan log yang diharapkan selama pemutusan jaringan:

```
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

Kredensyal sementara yang disediakan oleh IAM Roles Anywhere berlaku selama satu jam secara default. Anda dapat mengonfigurasi durasi validitas kredensyal dengan IAM Roles Anywhere melalui `durationSeconds`bidang di profil IAM Roles Anywhere Anda. Durasi validitas kredenal maksimum adalah 12 jam. `MaxSessionDuration`Pengaturan pada peran IAM Hybrid Nodes Anda harus lebih besar dari `durationSeconds` pengaturan pada profil IAM Roles Anywhere Anda.

Saat menggunakan IAM Roles Anywhere sebagai penyedia kredensi untuk node hybrid Anda, koneksi ulang ke bidang kontrol EKS setelah pemutusan jaringan biasanya terjadi dalam hitungan detik setelah restorasi jaringan, karena kubelet memanggil `aws_signing_helper credential-process` untuk mendapatkan kredensyal sesuai permintaan. Meskipun tidak terkait langsung dengan node hibrida atau pemutusan jaringan, Anda dapat mengonfigurasi notifikasi dan peringatan untuk kedaluwarsa sertifikat saat menggunakan Peran IAM Di Mana Saja. Untuk informasi selengkapnya, lihat Menyesuaikan setelan notifikasi di Peran IAM Di Mana Saja.

Praktik Terbaik untuk Menjalankan Beban AI/ML Kerja

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Menerapkan praktik terbaik saat menjalankan AI/ML beban kerja di EKS dapat memastikan bahwa beban kerja tersebut berkinerja baik, hemat biaya, tangguh, dan sumber daya dengan benar. Praktik terbaik dibagi menjadi beberapa bagian umum berikut: Compute, Networking, Storage, Observability, dan Performance.

Umpan Balik

Panduan ini sedang GitHub dirilis untuk mengumpulkan umpan balik langsung dan saran dari EKS/Kubernetes komunitas yang lebih luas. Jika Anda memiliki praktik terbaik yang menurut Anda harus kami sertakan dalam panduan ini, harap ajukan masalah atau kirimkan PR di GitHub repositori. Tujuan kami adalah memperbarui panduan secara berkala karena fitur baru ditambahkan ke layanan atau ketika praktik terbaik baru berkembang.

Komputasi dan Penskalaan Otomatis

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Optimalisasi Sumber Daya GPU dan Manajemen Biaya

Jadwalkan beban kerja dengan persyaratan GPU menggunakan label Terkenal

Untuk AI/ML beban kerja yang sensitif terhadap karakteristik GPU yang berbeda (misalnya GPU, memori GPU), kami merekomendasikan untuk menentukan persyaratan GPU menggunakan label penjadwalan yang diketahui yang didukung oleh tipe node yang digunakan dengan Karpenter dan grup node terkelola. Gagal mendefinisikan ini dapat mengakibatkan pod dijadwalkan pada instance

dengan sumber daya GPU yang tidak memadai, menyebabkan kegagalan atau penurunan kinerja. Kami merekomendasikan penggunaan afinitas nodeSelector atau Node untuk menentukan node mana yang harus dijalankan pod dan menyetel sumber daya komputasi (CPU, memori, GPUs dll) di bagian sumber daya pod.

Contoh

Misalnya, menggunakan pemilih node nama GPU saat menggunakan 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
```

Gunakan Plugin Perangkat Kubernetes untuk mengekspos GPUs

Untuk mengekspos GPUs pada node, driver GPU NVIDIA harus diinstal pada sistem operasi node dan runtime kontainer yang dikonfigurasi untuk memungkinkan penjadwal Kubernetes menetapkan pod ke node yang tersedia. GPUs Proses penyiapan untuk Plugin Perangkat NVIDIA Kubernetes bergantung pada AMI Akselerasi EKS yang Anda gunakan:

- Bottlerocket Accelerated AMI: AMI ini mencakup driver GPU NVIDIA dan Plugin Perangkat NVIDIA Kubernetes sudah diinstal sebelumnya dan siap digunakan, memungkinkan dukungan GPU di luar kotak. Tidak diperlukan konfigurasi tambahan untuk mengekspos GPUs ke penjadwal Kubernetes.
- AL2023 AMI yang Dipercepat: AMI ini menyertakan driver GPU NVIDIA tetapi Plugin Perangkat NVIDIA Kubernetes belum diinstal sebelumnya. Anda harus menginstal dan mengkonfigurasi plugin perangkat secara terpisah, biasanya melalui file DaemonSet. Perhatikan bahwa jika Anda menggunakan eksctl untuk membuat cluster Anda dan menentukan jenis instans GPU (misalnya,`g5.xlarge`) di Anda ClusterConfig, secara otomatis `eksctl` akan memilih AMI yang sesuai dan menginstal Plugin Perangkat NVIDIA Kubernetes. Untuk mempelajari lebih lanjut, lihat Dukungan GPU dalam dokumentasi eksctl.

Jika Anda memutuskan untuk menggunakan operator EKS Accelerated AMIs dan NVIDIA GPU untuk mengelola komponen seperti plugin perangkat NVIDIA Kubernetes sebagai gantinya, perhatikan untuk menonaktifkan manajemen driver GPU NVIDIA dan toolkit NVIDIA Container sesuai dengan Driver GPU NVIDIA Pra-Instalasi dan dokumentasi NVIDIA Container Toolkit NVIDIA.

Untuk memverifikasi bahwa Plugin Perangkat NVIDIA aktif dan GPUs terpapar dengan benar, jalankan:

```
kubectl describe node | grep nvidia.com/gpu
```

Perintah ini memeriksa apakah `nvidia.com/gpu` sumber daya berada dalam kapasitas node dan sumber daya yang dapat dialokasikan. Misalnya, node dengan satu GPU harus ditampilkan`nvidia.com/gpu: 1`. Lihat Panduan Penjadwalan GPU Kubernetes untuk informasi selengkapnya.

Gunakan banyak jenis instans EC2 yang berbeda

Menggunakan sebanyak mungkin jenis instans EC2 yang berbeda adalah praktik terbaik yang penting untuk skalabilitas di Amazon EKS, seperti yang diuraikan di bagian ini. the section called “Bidang Data” Rekomendasi ini juga berlaku untuk instance dengan perangkat keras yang dipercepat (misalnya, GPUs). Jika Anda membuat klaster yang hanya menggunakan satu jenis instans dan mencoba menskalakan jumlah node di luar kapasitas wilayah, Anda mungkin menerima kesalahan kapasitas yang tidak mencukupi (ICE), yang menunjukkan bahwa tidak ada instance yang tersedia. Penting untuk memahami karakteristik unik dari AI/ML beban kerja Anda sebelum melakukan diversifikasi secara sewenang-wenang. Tinjau jenis instans yang tersedia menggunakan alat EC2 Instance Type Explorer untuk membuat daftar jenis instans yang sesuai dengan persyaratan komputasi spesifik Anda, dan hindari membatasi secara sewenang-wenang jenis instance yang dapat digunakan di klaster Anda.

Instans komputasi yang dipercepat ditawarkan dalam model pembelian yang berbeda agar sesuai dengan beban kerja jangka pendek, jangka menengah, dan kondisi tunak. Untuk beban kerja jangka pendek, fleksibel, dan toleran kesalahan, di mana Anda ingin menghindari reservasi, lihat contoh Spot. Blok Kapasitas, Instans Sesuai Permintaan, dan Paket Penghematan memungkinkan Anda menyediakan instans komputasi yang dipercepat untuk durasi beban kerja jangka menengah dan panjang. Untuk meningkatkan peluang berhasil mengakses kapasitas yang diperlukan dalam opsi pembelian pilihan Anda, disarankan untuk menggunakan beragam daftar jenis instans dan zona ketersediaan. Atau, jika Anda menemukan ICEs model pembelian tertentu, coba lagi menggunakan model yang berbeda.

Contoh Contoh berikut menunjukkan cara mengaktifkan Karpenter NodePool untuk menyediakan instance G dan P yang lebih besar dari generasi 3 (misalnya, p3). Untuk mempelajari lebih lanjut, lihat Skalabilitas bagian.

```
- 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
```

Untuk detail tentang penggunaan instans Spot untuk GPU, lihat “Pertimbangkan untuk menggunakan Instans Spot Amazon EC2 GPUs untuk dengan Karpenter” di bawah ini.

Pertimbangkan untuk menggunakan Instans Spot Amazon EC2 untuk Karpenter GPUs

Instans Spot Amazon EC2 memungkinkan Anda memanfaatkan kapasitas EC2 yang tidak terpakai di AWS cloud dan tersedia dengan diskon hingga 90% dibandingkan dengan harga Sesuai Permintaan. Instans Spot Amazon EC2 dapat terganggu dengan pemberitahuan dua menit saat EC2 membutuhkan kapasitas kembali. Untuk informasi selengkapnya, lihat Instans Spot di Panduan Pengguna Amazon EC2. Amazon EC2 Spot dapat menjadi pilihan tepat untuk beban kerja yang toleran terhadap kesalahan, tanpa kewarganegaraan, dan fleksibel (tipe waktu dan instans). Untuk mempelajari lebih lanjut tentang kapan menggunakan instans Spot, lihat Praktik Terbaik Instans Spot EC2. Anda juga dapat menggunakan Instans Spot untuk AI/ML beban kerja jika mereka ramah tempat.

Kasus penggunaan

Beban kerja yang ramah tempat dapat berupa data besar, beban kerja kontainer, CI/CD, server web stateless, komputasi kinerja tinggi (HPC), dan beban kerja rendering. Instans Spot tidak cocok untuk beban kerja yang tidak fleksibel, stateful, tidak toleran kesalahan, atau digabungkan erat antara node instance (misalnya, beban kerja dengan proses paralel yang sangat bergantung satu sama lain untuk komputasi, membutuhkan komunikasi antar-simpul yang konstan, seperti aplikasi komputasi kinerja tinggi berbasis MPI seperti dinamika fluida komputasi atau database terdistribusi dengan saling ketergantungan yang kompleks). Berikut adalah kasus penggunaan khusus yang kami rekomendasikan (tanpa urutan tertentu):

- Inferensi online real-time: Gunakan instans Spot untuk penskalaan yang dioptimalkan biaya untuk beban kerja inferensi waktu nyata Anda, selama beban kerja Anda ramah tempat. Dengan kata lain, waktu inferensi kurang dari dua menit, aplikasi toleran terhadap kesalahan terhadap interupsi,

dan dapat berjalan pada jenis instance yang berbeda. Pastikan ketersediaan tinggi melalui keragaman instans (misalnya, di beberapa jenis instans dan Availability Zone) atau reservasi, sambil menerapkan toleransi kesalahan tingkat aplikasi untuk menangani potensi gangguan Spot.

- Penyetelan parameter hiper: Gunakan instans Spot untuk menjalankan pekerjaan penyetelan eksplorasi secara oportunistik, karena interupsi dapat ditoleransi tanpa kerugian yang signifikan, terutama untuk eksperimen berdurasi pendek.
- Augmentasi data: Gunakan instans Spot untuk melakukan prapemrosesan data dan tugas augmentasi yang dapat dimulai ulang dari pos pemeriksaan jika terputus, menjadikannya ideal untuk ketersediaan variabel Spot.
- Model fine-tuning: Gunakan instans Spot untuk fine-tuning dengan mekanisme checkpointing yang kuat untuk melanjutkan dari status tersimpan terakhir, meminimalkan dampak interupsi instans.
- Inferensi Batch: Gunakan instans Spot untuk memproses sejumlah besar permintaan inferensi offline non-real-time dengan cara tertentu, di mana pekerjaan dapat dijeda dan dilanjutkan, menawarkan penyelarasan terbaik dengan penghematan biaya Spot dan menangani potensi gangguan melalui percobaan ulang atau diversifikasi.
- Subset pelatihan oportunistik: Gunakan instance Spot untuk beban kerja pelatihan marjinal atau eksperimental (misalnya, model yang lebih kecil di bawah 10 juta parameter), di mana interupsi dapat diterima dan pengoptimalan efisiensi seperti diversifikasi di seluruh jenis instans atau wilayah dapat diterapkan—meskipun tidak direkomendasikan untuk pelatihan skala produksi karena potensi gangguan.

Pertimbangan-pertimbangan

Untuk menggunakan Instans Spot untuk beban kerja yang dipercepat di Amazon EKS, ada sejumlah pertimbangan utama (tanpa urutan tertentu):

- Gunakan Karpenter untuk mengelola instans Spot dengan konsolidasi lanjutan diaktifkan. Dengan menentukan karpenter.sh/capacity-type sebagai “spot” di Karpenter Anda, Karpenter NodePool akan menyediakan instance Spot secara default tanpa konfigurasi tambahan. Namun, untuk mengaktifkan Spot-to-Spot konsolidasi lanjutan, yang menggantikan node Spot yang kurang dimanfaatkan dengan alternatif Spot dengan harga lebih rendah, Anda perlu mengaktifkan gerbang SpotToSpotConsolidation fitur dengan menyetel --feature-gates SpotToSpotConsolidation =true dalam argumen pengontrol Karpenter atau melalui variabel lingkungan FEATURE_GATES. Karpenter menggunakan strategi price-capacity-optimizedalokasi untuk menyediakan instans EC2. Berdasarkan NodePool persyaratan dan batasan pod, Karpenter bin-pack pod yang tidak dapat dijadwalkan dan mengirimkan beragam jenis instance ke Amazon EC2 Fleet API. Anda dapat

menggunakan alat EC2 Instance Type Explorer untuk membuat daftar jenis instans yang sesuai dengan persyaratan komputasi spesifik Anda.

- Pastikan beban kerja tidak memiliki kewarganegaraan, toleransi kesalahan, dan fleksibel. Beban kerja harus tanpa kewarganegaraan, toleran terhadap kesalahan, dan fleksibel dalam hal ukuran. instance/GPU Hal ini memungkinkan dimulainya kembali secara mulus setelah interupsi Spot, dan fleksibilitas instans memungkinkan Anda untuk tetap berada di Spot lebih lama. Aktifkan penanganan interupsi Spot di Karpenter dengan mengonfigurasi nilai Settings.InterruptionQueue Helm dengan nama antrian AWS SQS untuk menangkap peristiwa interupsi Spot. Misalnya, saat menginstal melalui Helm, gunakan --set “settings.interruptionQueue=$ {CLUSTER_NAME}”. Untuk melihat contoh, lihat panduan Memulai dengan Karpenter. Ketika Karpenter melihat peristiwa interupsi Spot, ia secara otomatis mengikat, menodai, menguras, dan mengakhiri node sebelum peristiwa interupsi untuk memaksimalkan masa tenggang terminasi pod. Pada saat yang sama, Karpenter akan segera memulai node baru sehingga dapat siap sesegera mungkin.
- Hindari terlalu membatasi pemilihan jenis instance. Anda harus menghindari membatasi jenis instance sebanyak mungkin. Dengan tidak membatasi jenis instans, ada kemungkinan lebih tinggi untuk memperoleh kapasitas Spot pada skala besar dengan frekuensi interupsi Instans Spot yang lebih rendah dengan biaya lebih rendah. Misalnya, hindari membatasi tipe tertentu (misalnya, g5.xlarge). Pertimbangkan untuk menentukan serangkaian kategori dan generasi instance yang beragam menggunakan kunci seperti 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/MLbeban kerja memerlukan jumlah akselerator tertentu atau terbatas tetapi fleksibel antar wilayah, Anda dapat menggunakan Skor Penempatan Spot untuk mengidentifikasi wilayah optimal secara dinamis untuk menerapkan beban kerja Anda sebelum diluncurkan.
- Perluas NodePool persyaratan untuk memasukkan sejumlah besar keluarga instans EC2 serupa. Setiap kumpulan Instans Spot terdiri dari kapasitas instans EC2 yang tidak digunakan untuk jenis instans tertentu di Availability Zone (AZ) tertentu. Ketika Karpenter mencoba untuk menyediakan node baru, ia memilih jenis instance yang cocok dengan persyaratan. NodePool Jika tidak ada jenis instans yang kompatibel yang memiliki kapasitas Spot di AZ mana pun, maka penyediaan gagal. Untuk menghindari masalah ini, izinkan instance g-series yang lebih luas (generasi 4 atau lebih tinggi) dari NVIDIA di seluruh ukuran dan Availability Zones (AZs), sambil mempertimbangkan kebutuhan perangkat keras seperti memori GPU atau Ray Tracing. Karena instance dapat dari berbagai jenis, Anda perlu memastikan bahwa beban kerja Anda dapat berjalan pada setiap jenis, dan kinerja yang Anda dapatkan memenuhi kebutuhan Anda.
- Manfaatkan semua zona ketersediaan di suatu wilayah. Kapasitas yang tersedia bervariasi menurut Availability Zone (AZ), jenis instans tertentu mungkin tidak tersedia di satu AZ tetapi

berlimpah di AZ lainnya. Setiap kombinasi unik dari tipe instans dan Availability Zone merupakan kumpulan kapasitas Spot yang terpisah. Dengan meminta kapasitas di semua wilayah AZs dalam NodePool persyaratan Karpenter Anda, Anda secara efektif mencari lebih banyak kumpulan sekaligus. Ini memaksimalkan jumlah kumpulan kapasitas Spot dan oleh karena itu meningkatkan kemungkinan memperoleh kapasitas Spot. Untuk mencapai ini, dalam NodePool konfigurasi Anda, hilangkan kunci topology.kubernetes.io/zone sepenuhnya untuk memungkinkan Karpenter memilih dari semua yang tersedia AZs di wilayah, atau daftar AZs secara eksplisit menggunakan operator: In dan berikan nilainya (misalnya, us-west-2a).

- Pertimbangkan untuk menggunakan Spot Placement Score (SPS) untuk mendapatkan visibilitas ke kemungkinan berhasil mengakses kapasitas yang diperlukan menggunakan instans Spot. Spot Placement Score (SPS) adalah alat yang memberikan skor untuk membantu Anda menilai seberapa besar kemungkinan permintaan Spot berhasil. Saat menggunakan SPS, pertama-tama Anda menentukan persyaratan komputasi untuk Instans Spot, lalu Amazon EC2 mengembalikan 10 Wilayah atau Availability Zone (AZs) teratas tempat permintaan Spot Anda kemungkinan berhasil. Wilayah dan Zona Ketersediaan diberi skor pada skala 1 hingga 10. Skor 10 menunjukkan bahwa permintaan Spot Anda sangat mungkin tetapi tidak dijamin berhasil. Skor 1 menunjukkan bahwa permintaan Spot Anda tidak mungkin berhasil. Skor yang sama mungkin ditampilkan untuk Wilayah atau Zona Ketersediaan yang berbeda. Untuk mempelajari selengkapnya, lihat Panduan Membangun Dasbor Pelacak Skor Penempatan Spot di AWS. Karena kapasitas Spot berfluktuasi sepanjang waktu, SPS akan membantu Anda mengidentifikasi kombinasi jenis instans AZs, dan wilayah mana yang paling sesuai untuk kendala beban kerja Anda (yaitu fleksibilitas, kinerja, ukuran, dll.). Jika AI/ML beban kerja Anda memerlukan akselerator tertentu atau jumlah terbatas tetapi fleksibel antar wilayah, Anda dapat menggunakan skor penempatan Spot untuk mengidentifikasi wilayah optimal secara dinamis untuk menerapkan beban kerja Anda sebelum diluncurkan. Untuk membantu Anda mengetahui secara otomatis kemungkinan memperoleh kapasitas Spot, kami memberikan panduan untuk membangun dasbor pelacak SPS. Solusi ini memantau skor SPS dari waktu ke waktu menggunakan konfigurasi YAMAL untuk penyiapan yang beragam (misalnya, persyaratan instans termasuk GPUs), menyimpan metrik CloudWatch, dan menyediakan dasbor untuk membandingkan konfigurasi. Tentukan dasbor per beban kerja untuk mengevaluasi kebutuhan vCPU, memori, dan GPU, memastikan pengaturan optimal untuk kluster EKS termasuk pertimbangan penggunaan Wilayah AWS lainnya. Untuk mempelajari lebih lanjut, lihat Cara kerja skor penempatan Spot.
- Dengan anggun menangani interupsi dan pengujian Spot. Untuk pod dengan periode terminasi lebih dari dua menit, node lama akan terganggu sebelum pod tersebut dijadwalkan ulang, yang dapat memengaruhi ketersediaan beban kerja. Pertimbangkan pemberitahuan interupsi Spot dua menit saat merancang aplikasi Anda, terapkan checkpointing dalam aplikasi yang berjalan lama

(misalnya, menyimpan kemajuan ke penyimpanan persisten seperti Amazon S3) untuk melanjutkan setelah interupsi, memperpanjang terminationGracePeriod Detik (default adalah 30 detik) dalam spesifikasi Pod untuk memungkinkan lebih banyak waktu untuk shutdown yang anggun, dan menangani interupsi menggunakan siklus hidup PreStop menghubungkan sinyal and/or SIGTERM dalam aplikasi Anda untuk aktivitas shutdown yang anggun seperti pembersihan, penghematan negara, dan penutupan koneksi. Untuk beban kerja real-time, di mana waktu penskalaan penting dan beban kerja membutuhkan waktu lebih dari dua menit agar aplikasi siap melayani lalu lintas, pertimbangkan untuk mengoptimalkan waktu start-up kontainer dan pemuatan model ML dengan meninjau dan praktik terbaik. the section called “Penyimpanan” the section called “Performa” Untuk menguji node pengganti, gunakan AWS Fault Injection Service (FIS) untuk mensimulasikan interupsi Spot.

Selain praktik terbaik Spot inti ini, pertimbangkan faktor-faktor ini saat mengelola beban kerja GPU di Amazon EKS. Tidak seperti beban kerja berbasis CPU, beban kerja GPU sangat sensitif terhadap detail perangkat keras seperti kemampuan GPU dan memori GPU yang tersedia. Beban kerja GPU mungkin dibatasi oleh jenis instance yang dapat mereka gunakan, dengan lebih sedikit opsi yang tersedia dibandingkan dengan. CPUs Sebagai langkah pertama, menilai apakah beban kerja Anda fleksibel. Jika Anda tidak tahu berapa banyak jenis instans yang dapat digunakan oleh beban kerja Anda, ujilah satu per satu untuk memastikan kompatibilitas dan fungsionalitas. Identifikasi seberapa fleksibel Anda dapat melakukan diversifikasi sebanyak mungkin, sambil mengonfirmasi bahwa diversifikasi membuat beban kerja tetap bekerja dan memahami dampak kinerja apa pun (misalnya, pada throughput atau waktu penyelesaian). Sebagai bagian dari diversifikasi beban kerja Anda, pertimbangkan hal berikut:

- Tinjau CUDA dan kompatibilitas kerangka kerja. Beban kerja GPU Anda mungkin dioptimalkan untuk perangkat keras tertentu, jenis GPU (misalnya, V100 di p3 vs. A100 di p4), atau ditulis untuk versi CUDA tertentu untuk pustaka seperti TensorFlow, jadi pastikan untuk meninjau kompatibilitas untuk beban kerja Anda. Kompatibilitas ini sangat penting untuk mencegah kesalahan runtime, crash, kegagalan dalam akselerasi GPU (misalnya, versi CUDA yang tidak cocok dengan kerangka kerja seperti PyTorch atau TensorFlow dapat mencegah eksekusi), atau kemampuan untuk memanfaatkan fitur perangkat keras seperti/presisi. FP16 INT8
- Memori GPU. Pastikan untuk mengevaluasi persyaratan memori model Anda dan profil penggunaan memori model Anda selama runtime menggunakan alat seperti DCGM Exporter dan atur memori GPU minimum yang diperlukan untuk jenis instans di label terkenal seperti karpenter.k8s.aws/. instance-gpu-memory GPU VRAM bervariasi antar jenis instans (misalnya, NVIDIA T4 memiliki 16GB, A10G memiliki 24GB, V100 memiliki 16-32GB), dan model ML

(misalnya, model bahasa besar) dapat melebihi memori yang tersedia, menyebabkan (OOM) kesalahan atau crash. out-of-memory Untuk Instans Spot di EKS, ini dapat membatasi diversifikasi. Misalnya, Anda tidak dapat menyertakan tipe VRAM yang lebih rendah jika model Anda tidak cocok, yang dapat membatasi akses ke kumpulan kapasitas dan meningkatkan risiko gangguan. Perhatikan bahwa untuk GPU tunggal, inferensi simpul tunggal (misalnya, beberapa pod yang dijadwalkan pada node yang sama untuk memanfaatkan sumber daya GPU-nya), ini mungkin membatasi diversifikasi, karena Anda hanya dapat menyertakan tipe instance dengan VRAM yang cukup dalam konfigurasi Spot Anda.

- Presisi dan kinerja floating-point. Tidak semua arsitektur GPU Nvidia memiliki presisi floating point yang sama (misalnya, FP16/INT8). Evaluasi kinerja tipe inti (CUDA/Tensor/RT) dan presisi floating point yang diperlukan untuk beban kerja Anda. Berjalan pada GPU dengan harga lebih rendah dan kurang berkinerja tidak berarti itu lebih baik, jadi pertimbangkan untuk mengevaluasi kinerja dalam hal pekerjaan yang diselesaikan dalam kerangka waktu tertentu untuk memahami dampak diversifikasi.

Skenario: Diversifikasi untuk beban kerja inferensi waktu nyata

Untuk beban kerja inferensi online real-time di Instans Spot, Anda dapat mengonfigurasi Karpenter NodePool untuk melakukan diversifikasi di seluruh keluarga dan generasi instans GPU yang kompatibel. Pendekatan ini memastikan ketersediaan tinggi dengan menggambar dari beberapa kumpulan Spot, sambil mempertahankan kinerja melalui kendala pada kemampuan GPU, memori, dan arsitektur. Ini mendukung penggunaan alternatif ketika kapasitas instance dibatasi, meminimalkan interupsi dan mengoptimalkan latensi inferensi. Contoh ini NodePool menyatakan, gunakan instance seri g dan p lebih besar dari 3, yang memiliki memori GPU lebih dari 20GB.

Contoh

```
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
```

Menerapkan Checkpointing untuk Pekerjaan Pelatihan Jangka Panjang

Checkpointing adalah teknik toleransi kesalahan yang melibatkan penyelamatan keadaan proses secara berkala, memungkinkannya untuk melanjutkan dari titik terakhir yang disimpan jika terjadi gangguan. Dalam pembelajaran mesin, ini umumnya dikaitkan dengan pelatihan, di mana pekerjaan yang berjalan lama dapat menghemat bobot model dan status pengoptimal untuk melanjutkan pelatihan setelah kegagalan, seperti masalah perangkat keras atau gangguan Instans Spot.

Anda menggunakan pos pemeriksaan untuk menyimpan status model pembelajaran mesin (ML) selama pelatihan. Checkpoint adalah snapshot dari model dan dapat dikonfigurasi oleh fungsi callback kerangka kerja HTML. Anda dapat menggunakan pos pemeriksaan yang disimpan untuk memulai kembali pekerjaan pelatihan dari pos pemeriksaan terakhir yang disimpan. Dengan menggunakan pos pemeriksaan, Anda menyimpan snapshot model Anda di bawah pelatihan karena

gangguan yang tidak terduga pada pekerjaan atau instance pelatihan. Ini memungkinkan Anda untuk melanjutkan pelatihan model di masa depan dari pos pemeriksaan. Selain menerapkan sistem ketahanan node, sebaiknya implementasikan checkpointing untuk mengurangi dampak interupsi, termasuk yang disebabkan oleh kegagalan perangkat keras atau interupsi Instans Spot Amazon EC2.

Tanpa checkpointing, interupsi dapat mengakibatkan waktu komputasi yang terbuang dan kehilangan kemajuan, yang mahal untuk pekerjaan pelatihan yang berjalan lama. Checkpointing memungkinkan pekerjaan untuk menyimpan statusnya secara berkala (misalnya, bobot model dan status pengoptimal) dan melanjutkan dari pos pemeriksaan terakhir (batch yang diproses terakhir) setelah gangguan. Untuk menerapkan pos pemeriksaan, rancang aplikasi Anda untuk memproses data dalam batch besar dan menyimpan hasil perantara ke penyimpanan persisten, seperti bucket Amazon S3 melalui Mountpoint for Amazon S3 CSI Driver saat tugas pelatihan berlangsung.

Kasus penggunaan

Checkpointing sangat bermanfaat dalam skenario tertentu untuk menyeimbangkan toleransi kesalahan dengan overhead kinerja. Pertimbangkan untuk menggunakan pos pemeriksaan dalam kasus berikut:

- Durasi pekerjaan melebihi beberapa jam: Untuk pekerjaan pelatihan jangka panjang (mis., > 1-2 jam untuk model kecil, atau days/weeks untuk model pondasi besar dengan miliaran parameter), di mana kerugian kemajuan dari interupsi mahal. Pekerjaan yang lebih pendek mungkin tidak membenarkan I/O overhead.
- Untuk instans Spot atau kegagalan perangkat keras: Di lingkungan yang rentan terhadap gangguan, seperti EC2 Spot (pemberitahuan 2 menit) atau kegagalan perangkat keras (misalnya, kesalahan memori GPU), pos pemeriksaan memungkinkan dimulainya kembali dengan cepat, membuat Spot layak untuk penghematan biaya dalam beban kerja yang toleran terhadap kesalahan.
- Pelatihan terdistribusi pada skala: Untuk pengaturan dengan hundreds/thousands akselerator (misalnya, >100 GPUs), di mana waktu rata-rata antara kegagalan berkurang secara linier dengan skala. Gunakan untuk model/data paralelisme untuk menangani akses pos pemeriksaan bersamaan dan menghindari restart lengkap.
- Model skala besar dengan permintaan sumber daya yang tinggi: Dalam pelatihan LLM skala petabyte, di mana kegagalan tidak dapat dihindari karena ukuran cluster; pendekatan berjenjang (lokal cepat setiap 5-30 menit untuk transien, tahan lama setiap jam untuk kegagalan besar) mengoptimalkan waktu pemulihan vs. efisiensi.

Gunakan Blok Kapasitas ML untuk jaminan kapasitas instans P dan Trainium

Blok Kapasitas untuk ML memungkinkan Anda untuk memesan instans GPU yang sangat dicari, khususnya instans P (misalnya, p6-b200, p5, p5e, p5en, p4d, p4de) dan instans Trainium (misalnya, trn1, trn2), untuk memulai baik segera atau di masa mendatang untuk mendukung beban kerja machine learning (ML) durasi pendek Anda. Reservasi ini ideal untuk memastikan kapasitas untuk tugas-tugas komputasi intensif seperti pelatihan model dan fine-tuning. Harga Blok Kapasitas EC2 terdiri dari biaya reservasi dan biaya sistem operasi. Untuk mempelajari lebih lanjut tentang harga, lihat Blok Kapasitas EC2 untuk harga ML.

GPUs Untuk memesan AI/ML beban kerja di Amazon EKS untuk jaminan kapasitas yang dapat diprediksi, kami merekomendasikan untuk memanfaatkan Blok Kapasitas ML untuk Reservasi Kapasitas jangka pendek atau Sesuai Permintaan () ODCRs untuk jaminan kapasitas tujuan umum.

- ODCR memungkinkan Anda untuk memesan kapasitas instans EC2 (misalnya, instans GPU seperti g5 atau p5) di Availability Zone tertentu selama durasi tertentu, memastikan ketersediaan, bahkan selama permintaan tinggi. ODCRs tidak memiliki komitmen jangka panjang, tetapi Anda membayar tarif On-Demand untuk kapasitas cadangan, baik digunakan atau tidak digunakan. Di EKS, ODCRs didukung oleh tipe node seperti Karpenter dan grup node terkelola. Untuk memprioritaskan ODCRs di Karpenter, konfigurasikan NodeClass untuk menggunakan bidang. `capacityReservationSelectorTerms` Lihat Dokumentasi Karpenter NodePools .
- Blok Kapasitas adalah mekanisme reservasi khusus untuk instans GPU (misalnya, p5, p4d) atau Trainium (trn1, trn2), yang dirancang untuk beban kerja MS jangka pendek seperti pelatihan model, fine-tuning, atau eksperimen. Anda memesan kapasitas untuk jangka waktu tertentu (biasanya 24 jam hingga 182 hari) dimulai pada tanggal future, hanya membayar untuk waktu yang dipesan. Mereka pra-bayar, memerlukan pra-perencanaan untuk kebutuhan kapasitas dan tidak mendukung penskalaan otomatis, tetapi mereka ditempatkan di EC2 untuk jaringan latensi rendah. UltraClusters Mereka hanya mengenakan biaya untuk periode yang dipesan. Untuk mempelajari selengkapnya, lihat Temukan dan beli Blok Kapasitas, atau mulai dengan menyiapkan grup node terkelola dengan Blok Kapasitas menggunakan petunjuk dalam Membuat grup node terkelola dengan Blok Kapasitas untuk ML.

Cadangan kapasitas melalui AWS Management Console dan konfigurasikan node Anda untuk menggunakan blok kapasitas ML. Rencanakan reservasi berdasarkan jadwal beban kerja dan uji dalam klaster pementasan. Lihat Dokumentasi Blok Kapasitas untuk informasi lebih lanjut.

Pertimbangkan Pemesanan Kapasitas Sesuai Permintaan, Amazon EC2 Spot, atau Sesuai Permintaan (ODCR) untuk instans G Amazon EC2

Untuk Instans G Amazon EC2, pertimbangkan opsi pembelian yang berbeda dari On-Demand, Instans Spot Amazon EC2, dan Reservasi Kapasitas Sesuai Permintaan. ODCRsmemungkinkan Anda untuk memesan kapasitas instans EC2 di Availability Zone tertentu untuk durasi tertentu, memastikan ketersediaan bahkan selama permintaan tinggi. Tidak seperti Blok Kapasitas ML, yang hanya tersedia untuk instans P dan Trainium, ODCR dapat digunakan untuk berbagai jenis instans yang lebih luas, termasuk instans G, sehingga cocok untuk beban kerja yang memerlukan kemampuan GPU berbeda, seperti inferensi atau grafik. Saat menggunakan Instans Spot Amazon EC2, dapat beragam di berbagai jenis instans, ukuran, dan zona ketersediaan adalah kunci untuk dapat tetap berada di Spot lebih lama.

ODCRs tidak memiliki komitmen jangka panjang, tetapi Anda membayar tarif On-Demand untuk kapasitas cadangan, baik digunakan atau tidak digunakan. ODCRs dapat dibuat untuk segera digunakan atau dijadwalkan untuk masa depan, memberikan fleksibilitas dalam perencanaan kapasitas. Di Amazon EKS, ODCRs didukung oleh tipe node seperti Karpenter dan grup node terkelola. Untuk memprioritaskan ODCRs di Karpenter, konfigurasikan NodeClass untuk menggunakan bidang. `capacityReservationSelectorTerms` Lihat Dokumentasi Karpenter NodePools . Untuk informasi selengkapnya tentang pembuatan ODCRs, termasuk perintah CLI, lihat Memulai Reservasi Kapasitas Sesuai Permintaan.

Pertimbangkan jenis dan ukuran instans akselerasi lainnya

Memilih instans dan ukuran akselerasi yang sesuai sangat penting untuk mengoptimalkan kinerja dan biaya dalam beban kerja ML Anda di Amazon EKS. Misalnya, keluarga instans GPU yang berbeda memiliki kinerja dan kemampuan yang berbeda seperti memori GPU. Untuk membantu Anda memilih opsi berkinerja harga paling tinggi, tinjau instans GPU yang tersedia di halaman Jenis Instans EC2 di bawah Komputasi Akselerasi. Evaluasi beberapa jenis dan ukuran instans untuk menemukan yang paling sesuai dengan kebutuhan beban kerja spesifik Anda. Pertimbangkan faktor-faktor seperti jumlah GPUs, memori, dan kinerja jaringan. Dengan hati-hati memilih jenis dan ukuran instans GPU yang tepat, Anda dapat mencapai pemanfaatan sumber daya dan efisiensi biaya yang lebih baik di kluster EKS Anda.

Jika Anda menggunakan instance GPU di node EKS maka `nvidia-device-plugin-daemonset` pod akan ada di `kube-system` namespace secara default. Untuk mengetahui dengan cepat apakah Anda sepenuhnya menggunakan GPU dalam instance Anda, Anda dapat menggunakan nvidia-smi seperti yang ditunjukkan di sini:

```
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
```

- Jika `utilization.memory` mendekati 100%, maka kode Anda kemungkinan terikat memori. Ini berarti bahwa GPU (memori) sepenuhnya digunakan tetapi dapat menyarankan bahwa optimasi kinerja lebih lanjut harus diselidiki.
- Jika mendekati 100%, ini tidak berarti GPU sepenuhnya digunakan. `utilization.gpu` Metrik yang lebih baik untuk dilihat adalah rasio `power.draw` to`power.limit`. Jika rasio ini 100% atau lebih, maka kode Anda sepenuhnya memanfaatkan kapasitas komputasi GPU.
- `-l 5`Bendera mengatakan untuk menampilkan metrik setiap 5 detik. Dalam kasus jenis instans GPU tunggal, flag kueri indeks tidak diperlukan.

Untuk mempelajari selengkapnya, lihat instans GPU dalam dokumentasi AWS.

Optimalkan Alokasi Sumber Daya GPU dengan Time-Slicing, MIG, dan Alokasi GPU Fraksional

Batas sumber daya statis di Kubernetes (misalnya, CPU, memori, jumlah GPU) dapat menyebabkan penyediaan berlebih atau kurang dimanfaatkan, terutama untuk beban kerja dinamis seperti inferensi. AI/ML Memilih GPU yang tepat adalah penting. Untuk beban kerja bervolume rendah atau runcing, time-slicing memungkinkan beberapa beban kerja untuk berbagi satu GPU dengan berbagi sumber daya komputasi, berpotensi meningkatkan efisiensi dan mengurangi pemborosan. Berbagi GPU dapat dicapai melalui berbagai opsi:

- Memanfaatkan Node Selectors/Node affinity untuk mempengaruhi penjadwalan: Pastikan node yang disediakan dan pod dijadwalkan sesuai GPUs untuk beban kerja (mis.,) `karpenter.k8s.aws/instance-gpu-name: "a100"`
- Time-Slicing: Menjadwalkan beban kerja untuk berbagi sumber daya komputasi GPU dari waktu ke waktu, memungkinkan eksekusi bersamaan tanpa partisi fisik. Ini sangat ideal untuk beban kerja dengan tuntutan komputasi variabel, tetapi mungkin tidak memiliki isolasi memori.
- Multi-Instance GPU (MIG): MIG memungkinkan satu GPU NVIDIA untuk dipartisi menjadi beberapa instance yang terisolasi dan didukung dengan NVIDIA Ampere (misalnya, GPU A100), NVIDIA Hopper (misalnya, GPU H100), dan NVIDIA Blackwell (misalnya, Blackwell). GPUs GPUs Setiap

instans MIG menerima sumber daya komputasi dan memori khusus, memungkinkan berbagi sumber daya di lingkungan multi-penyewa atau beban kerja yang memerlukan jaminan sumber daya, yang memungkinkan Anda mengoptimalkan pemanfaatan sumber daya GPU, termasuk skenario seperti menyajikan beberapa model dengan ukuran batch berbeda melalui pengiris waktu.

- Alokasi GPU pecahan: Menggunakan penjadwalan berbasis perangkat lunak untuk mengalokasikan bagian dari komputasi atau memori GPU ke beban kerja, menawarkan fleksibilitas untuk beban kerja dinamis. NVIDIA KAI Scheduler, bagian dari platform Run:AI, memungkinkan pod untuk meminta sumber daya GPU fraksional.

Untuk mengaktifkan fitur-fitur ini di EKS, Anda dapat menerapkan Plugin Perangkat NVIDIA, yang mengekspos GPUs sebagai sumber daya yang dapat dijadwalkan dan mendukung pengiris waktu dan MIG. Untuk mempelajari lebih lanjut, lihat Time-Slicing GPUs di Kubernetes dan berbagi GPU di Amazon EKS dengan NVIDIA time-slicing dan instans EC2 yang dipercepat.

Contoh

Misalnya, untuk mengaktifkan pengiris waktu dengan Plugin Perangkat NVIDIA:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nvidia-device-plugin-config
  namespace: kube-system
data:
  config.yaml: |
    version: v1
    sharing:
      timeSlicing:
        resources:
        - name: nvidia.com/gpu
          replicas: 4  # Allow 4 pods to share each GPU
```

Contoh

Misalnya, untuk menggunakan KAI Scheduler untuk alokasi GPU fraksional, terapkan bersama Operator GPU NVIDIA dan tentukan sumber daya GPU fraksional dalam spesifikasi 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
```

Ketahanan Node dan Manajemen Job Pelatihan

Menerapkan Pemeriksaan Kesehatan Node dengan Pemulihan Otomatis

Untuk pekerjaan pelatihan terdistribusi di Amazon EKS yang memerlukan komunikasi antar simpul yang sering, seperti pelatihan model multi-GPU di beberapa node, masalah perangkat keras seperti kegagalan GPU atau EFA dapat menyebabkan gangguan pada pekerjaan pelatihan. Gangguan ini dapat menyebabkan hilangnya kemajuan pelatihan dan peningkatan biaya, terutama untuk AI/ML beban kerja jangka panjang yang bergantung pada perangkat keras yang stabil.

Untuk membantu menambah ketahanan terhadap kegagalan perangkat keras, seperti kegagalan GPU di kluster EKS yang menjalankan beban kerja GPU, kami sarankan untuk memanfaatkan Agen Pemantauan Node EKS dengan Perbaikan Otomatis atau Amazon. SageMaker HyperPod Sementara Agen Pemantauan Node EKS dengan Perbaikan Otomatis menyediakan fitur-fitur seperti pemantauan kesehatan node dan perbaikan otomatis menggunakan mekanisme Kubernetes standar, SageMaker HyperPod menawarkan ketahanan yang ditargetkan dan fitur tambahan yang dirancang khusus untuk pelatihan ML skala besar, seperti pemeriksaan kesehatan mendalam dan dimulainya kembali pekerjaan otomatis.

- Agen Pemantauan Node EKS dengan Perbaikan Otomatis Node terus memantau kesehatan node dengan membaca log dan menerapkan NodeConditions, termasuk kondisi standar seperti `Ready` dan kondisi khusus untuk perangkat keras yang dipercepat untuk mengidentifikasi masalah seperti GPU atau kegagalan jaringan. Ketika sebuah node dianggap tidak sehat, Node Auto Repair

mengikatnya dan menggantinya dengan node baru. Penjadwalan ulang pod dan restart pekerjaan bergantung pada mekanisme Kubernetes standar dan kebijakan restart pekerjaan.

- Pemeriksaan kesehatan SageMaker HyperPodmendalam dan agen pemantauan kesehatan terus memantau status kesehatan GPU dan instance berbasis Trainium. Ini disesuaikan untuk AI/ML beban kerja, menggunakan label (misalnya, node-health-status) untuk mengelola kesehatan node. Ketika sebuah node dianggap tidak sehat, HyperPod memicu penggantian otomatis perangkat keras yang rusak, seperti. GPUs Ini mendeteksi kegagalan terkait jaringan untuk EFA melalui pemeriksaan kesehatan dasarnya secara default dan mendukung resume otomatis untuk pekerjaan pelatihan yang terputus, memungkinkan pekerjaan untuk melanjutkan dari pos pemeriksaan terakhir, meminimalkan gangguan untuk tugas-tugas ML skala besar.

Untuk Agen Pemantau Node EKS dengan Perbaikan Otomatis dan SageMaker HyperPod cluster yang menggunakan EFA, untuk memantau metrik khusus EFA seperti kesalahan Remote Direct Memory Access (RDMA) dan penurunan paket, pastikan driver AWS EFA diinstal. Selain itu, kami merekomendasikan untuk menggunakan Pengaya CloudWatch Observabilitas atau menggunakan alat seperti Eksportir DCGM dengan Prometheus dan Grafana untuk memantau EFA, GPU, dan, untuk, metrik spesifik yang terkait dengan fitur-fiturnya. SageMaker HyperPod

Nonaktifkan Konsolidasi Karpenter untuk beban kerja sensitif interupsi

Untuk beban kerja yang sensitif terhadap gangguan, seperti pemrosesan, tugas AI/ML prediksi skala besar, atau pelatihan, kami merekomendasikan untuk menyetel kebijakan konsolidasi Karpenter untuk mencegah gangguan selama pelaksanaan pekerjaan. Fitur konsolidasi Karpenter secara otomatis mengoptimalkan biaya cluster dengan menghentikan node yang kurang dimanfaatkan atau menggantinya dengan alternatif yang lebih murah. Namun, bahkan ketika beban kerja sepenuhnya menggunakan GPU, Karpenter dapat mengkonsolidasikan node jika mengidentifikasi jenis instance berukuran tepat dengan harga lebih rendah yang memenuhi persyaratan pod, yang mengarah ke gangguan pekerjaan.

Kebijakan `WhenEmptyOrUnderutilized` konsolidasi dapat menghentikan node sebelum waktunya, yang mengarah ke waktu eksekusi yang lebih lama. Misalnya, interupsi dapat menunda dimulainya kembali pekerjaan karena penjadwalan ulang pod, pemuatan ulang data, yang bisa mahal untuk pekerjaan inferensi batch yang berjalan lama. Untuk mengurangi ini, Anda dapat mengatur `consolidationPolicy` ke `WhenEmpty` dan mengonfigurasi `consolidateAfter` durasi, seperti 1 jam, untuk mempertahankan node selama lonjakan beban kerja. Contoh:

```
disruption:
  consolidationPolicy: WhenEmpty
```

```
  consolidateAfter: 60m
```

Pendekatan ini meningkatkan latensi startup pod untuk beban kerja inferensi batch runcing dan pekerjaan sensitif interupsi lainnya, seperti pemrosesan data inferensi online real-time atau pelatihan model, di mana biaya interupsi melebihi penghematan biaya komputasi. Anggaran NodePool Gangguan Karpenter adalah fitur lain untuk mengelola gangguan Karpenter. Dengan anggaran, Anda dapat memastikan bahwa tidak lebih dari sejumlah node node akan terganggu pada yang dipilih NodePool pada suatu titik waktu. Anda juga dapat menggunakan anggaran gangguan untuk mencegah semua node terganggu pada waktu tertentu (misalnya jam sibuk). Untuk mempelajari lebih lanjut, lihat dokumentasi Konsolidasi Karpenter.

Gunakan Pekerjaan ttlSecondsAfter Kubernetes Selesai untuk Membersihkan Otomatis

Kami merekomendasikan pengaturan `ttlSecondsAfterFinished` untuk pekerjaan Kubernetes di Amazon EKS untuk secara otomatis menghapus objek pekerjaan yang sudah selesai. Objek pekerjaan yang tersisa menggunakan sumber daya klaster, seperti memori server API, dan mempersulit pemantauan dengan mengacaukan dasbor (misalnya, Grafana, Amazon). CloudWatch Misalnya, menyetel TTL 1 jam memastikan pekerjaan dihapus segera setelah selesai, menjaga cluster Anda tetap rapi. Untuk detail selengkapnya, lihat Pembersihan Otomatis untuk Pekerjaan Selesai.

Konfigurasikan Preemption Job Prioritas Rendah untuk Pekerjaan/Beban Kerja Prioritas Tinggi

Untuk beban AI/ML kerja prioritas campuran di Amazon EKS, Anda dapat mengonfigurasi preemption pekerjaan dengan prioritas rendah untuk memastikan tugas dengan prioritas lebih tinggi (misalnya, inferensi waktu nyata) menerima sumber daya dengan segera. Tanpa preemption, beban kerja dengan prioritas rendah seperti proses batch (misalnya, inferensi batch, pemrosesan data), layanan non-batch (misalnya, tugas latar belakang, pekerjaan cron), atau pekerjaan intensif CPU/memori (misalnya, layanan web) dapat menunda pod kritis dengan menempati node. Preemption memungkinkan Kubernetes untuk mengusir Pod dengan prioritas rendah ketika Pod dengan prioritas tinggi membutuhkan sumber daya, memastikan alokasi sumber daya yang efisien pada node dengan, atau memori. GPUs CPUs Sebaiknya gunakan Kubernetes `PriorityClass` untuk menetapkan prioritas dan `PodDisruptionBudget` mengendalikan perilaku penggusuran.

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
```

```
metadata:
  name: low-priority
value: 100
---
spec:
  priorityClassName: low-priority
```

Lihat Dokumentasi Prioritas dan Preemption Kubernetes untuk informasi selengkapnya.

Penskalaan dan Kinerja Aplikasi

Sesuaikan Kapasitas Komputasi untuk beban kerja ML dengan Karpenter atau Static Nodes

Untuk memastikan kapasitas komputasi yang hemat biaya dan responsif untuk alur kerja machine learning (ML) di Amazon EKS, sebaiknya sesuaikan strategi penyediaan node Anda dengan karakteristik beban kerja dan komitmen biaya Anda. Di bawah ini adalah dua pendekatan yang perlu dipertimbangkan: just-in-time penskalaan dengan Karpenter dan grup node statis untuk kapasitas cadangan.

- Just-in-time scaler bidang data seperti Karpenter: Untuk alur kerja HTML dinamis dengan tuntutan komputasi variabel (misalnya, inferensi berbasis GPU diikuti oleh plotting berbasis CPU), kami merekomendasikan penggunaan scaler bidang data seperti Karpenter. just-in-time
- Gunakan grup simpul statis untuk beban kerja yang dapat diprediksi: Untuk beban kerja HTML kondisi mapan yang dapat diprediksi atau saat menggunakan instans Cadangan, grup node terkelola EKS dapat membantu memastikan kapasitas cadangan sepenuhnya disediakan dan digunakan, memaksimalkan penghematan. Pendekatan ini sangat ideal untuk jenis contoh tertentu yang dilakukan melalui RIs atau ODCRs.

Contoh

Ini adalah contoh dari Karpenter beragam NodePoolyang memungkinkan peluncuran instans Amazon g EC2 di mana pembuatan instans lebih dari tiga.

```
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
```

Contoh

Contoh menggunakan grup node statis untuk beban kerja pelatihan:

```
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
```

Gunakan taint dan toleransi untuk mencegah beban kerja yang tidak dipercepat dijadwalkan pada instans yang dipercepat

Menjadwalkan beban kerja yang tidak dipercepat pada sumber daya GPU tidak efisien komputasi, kami merekomendasikan penggunaan taint dan toleransi untuk memastikan pod beban kerja yang tidak dipercepat tidak dijadwalkan pada node yang tidak sesuai. Lihat dokumentasi Kubernetes untuk informasi lebih lanjut.

Skala Berdasarkan Kinerja Model

Untuk beban kerja inferensi, sebaiknya gunakan Kubernetes Event-Driven Autoscaling (KEDA) untuk menskalakan berdasarkan metrik kinerja model seperti permintaan inferensi atau throughput token, dengan periode cooldown yang sesuai. Kebijakan penskalaan statis dapat menyediakan sumber daya yang berlebihan atau kurang, berdampak pada biaya dan latensi. Pelajari lebih lanjut di Dokumentasi KEDA.

Alokasi sumber daya dinamis untuk manajemen GPU tingkat lanjut

Alokasi sumber daya dinamis (DRA) merupakan kemajuan mendasar dalam manajemen sumber daya GPU Kubernetes. DRA bergerak melampaui batasan plugin perangkat tradisional untuk memungkinkan berbagi GPU yang canggih, kesadaran topologi, dan koordinasi sumber daya lintas simpul. Tersedia di Amazon EKS versi 1.33, DRA mengatasi tantangan kritis dalam AI/ML beban kerja dengan menyediakan hal-hal berikut:

- Alokasi GPU berbutir halus
- Mekanisme berbagi tingkat lanjut, seperti Multi-Process service (MPS) dan Multi-Instance GPU (MIG)
- Dukungan untuk arsitektur perangkat keras generasi berikutnya, termasuk NVIDIA GB200 UltraServers

Alokasi GPU tradisional diperlakukan GPUs sebagai sumber daya bilangan bulat buram, menciptakan pemanfaatan yang kurang signifikan (seringkali 30-40% dalam cluster produksi). Ini terjadi karena beban kerja menerima akses eksklusif ke seluruh GPUs bahkan ketika hanya

membutuhkan sumber daya fraksional. DRA mengubah model ini dengan memperkenalkan alokasi deklaratif terstruktur yang menyediakan penjadwal Kubernetes dengan visibilitas lengkap ke dalam karakteristik perangkat keras dan persyaratan beban kerja. Hal ini memungkinkan keputusan penempatan cerdas dan berbagi sumber daya yang efisien.

Keuntungan menggunakan DRA bukan plugin perangkat NVIDIA

Plugin perangkat NVIDIA (mulai dari versi`0.12.0`) mendukung mekanisme berbagi GPU termasuk time-slicing, MPS, dan MIG. Namun, ada batasan arsitektur yang ditangani DRA.

Keterbatasan plugin perangkat NVIDIA

- Konfigurasi statis: Konfigurasi berbagi GPU (replika pengiris waktu dan pengaturan MPS) memerlukan pra-konfigurasi di seluruh klaster. `ConfigMaps` Ini membuat penyediaan strategi berbagi yang berbeda untuk beban kerja yang berbeda menjadi sulit.
- Pemilihan granular terbatas: Meskipun plugin perangkat mengekspos karakteristik GPU melalui label node, beban kerja tidak dapat secara dinamis meminta konfigurasi GPU tertentu (ukuran memori dan kemampuan komputasi) sebagai bagian dari keputusan penjadwalan.
- Tidak ada koordinasi sumber daya lintas simpul: Tidak dapat mengelola sumber daya GPU terdistribusi di beberapa node atau mengekspresikan persyaratan topologi kompleks seperti NVLink domain untuk sistem seperti NVIDIA. GB200
- Batasan penjadwal: Penjadwal Kubernetes memperlakukan sumber daya GPU sebagai bilangan bulat buram, membatasi kemampuannya untuk membuat keputusan sadar topologi atau menangani dependensi sumber daya yang kompleks.
- Kompleksitas konfigurasi: Menyiapkan strategi berbagi yang berbeda membutuhkan pelabelan node yang banyak `ConfigMaps` dan hati-hati, menciptakan kompleksitas operasional.

Solusi dengan DRA

- Pemilihan sumber daya dinamis: DRA memungkinkan beban kerja untuk menentukan persyaratan terperinci (memori GPU, versi driver, dan atribut tertentu) pada saat permintaan selesai. `resourceclaims` Ini memungkinkan pencocokan sumber daya yang lebih fleksibel.
- Kesadaran topologi: Melalui parameter terstruktur dan pemilih perangkat, DRA menangani persyaratan kompleks seperti komunikasi GPU lintas simpul dan interkoneksi yang koheren memori.
- Manajemen sumber daya lintas-node: `computeDomains` memungkinkan koordinasi sumber daya GPU terdistribusi di beberapa node, penting untuk sistem seperti GB200 dengan saluran IMEX.

- Konfigurasi khusus beban kerja: Masing-masing **`ResourceClaim`** menentukan strategi dan konfigurasi berbagi yang berbeda, memungkinkan kontrol berbutir halus per beban kerja daripada pengaturan di seluruh cluster.
- Integrasi penjadwal yang disempurnakan: DRA menyediakan penjadwal dengan informasi perangkat terperinci dan memungkinkan keputusan penempatan yang lebih cerdas berdasarkan topologi perangkat keras dan karakteristik sumber daya.

Penting: DRA tidak menggantikan plugin perangkat NVIDIA sepenuhnya. Driver NVIDIA DRA bekerja bersama plugin perangkat untuk memberikan kemampuan yang ditingkatkan. Plugin perangkat terus menangani penemuan dan manajemen GPU dasar, sementara DRA menambahkan fitur alokasi dan penjadwalan lanjutan.

Instans yang didukung oleh DRA dan fitur-fiturnya

Dukungan DRA bervariasi menurut keluarga instans Amazon EC2 dan arsitektur GPU, seperti yang ditunjukkan pada tabel berikut.

Keluarga instans	Jenis GPU	Mengiris waktu	Dukungan MIG	Dukungan MPS	Dukungan IMEX	Kasus penggunaan
G5	NVIDIA A10G	Ya	Tidak	Ya	Tidak	Beban kerja inferensi dan grafis
G6	NVIDIA L4	Ya	Tidak	Ya	Tidak	Inferensi AI dan pemrosesan video
G6e	NVIDIA L40-AN	Ya	Tidak	Ya	Tidak	Pelatihan, inferensi, dan grafik
P4D/ P4DE	NVIDIA A100	Ya	Ya	Ya	Tidak	Pelatihan skala besar dan HPC
P5	NVIDIA H100	Ya	Ya	Ya	Tidak	Pelatihan model pondasi
P6	NVIDIA B200	Ya	Ya	Ya	Tidak	Model miliaran atau triliunan parameter,

Keluarga instans	Jenis GPU	Mengiris waktu	Dukungan MIG	Dukungan MPS	Dukungan IMEX	Kasus penggunaan
						pelatihan terdistribusi, dan inferensi
P6e	NVIDIA GB200	Ya	Ya	Ya	Ya	Model miliaran atau triliunan parameter, pelatihan terdistribusi, dan inferensi

Berikut ini adalah deskripsi dari setiap fitur dalam tabel:

- Time-slicing: Memungkinkan beberapa beban kerja untuk berbagi sumber daya komputasi GPU dari waktu ke waktu.
- Multi-Instance GPU (MIG): Partisi tingkat perangkat keras yang membuat instance GPU terisolasi.
- Multi-Process service (MPS): Memungkinkan eksekusi bersamaan dari beberapa proses CUDA pada satu GPU.
- Internode Memory Exchange (IMEX): Komunikasi koheren memori lintas node untuk. GB200 UltraServers

Sumber daya tambahan

Untuk informasi selengkapnya tentang driver Kubernetes DRA dan NVIDIA DRA, lihat sumber daya berikut di: GitHub

- Kubernetes dynamic-resource-allocation
- Proposal peningkatan Kubernetes untuk DRA
- Driver NVIDIA DRA untuk GPUs
- Contoh NVIDIA DRA dan mulai cepat

Siapkan alokasi sumber daya dinamis untuk manajemen GPU tingkat lanjut

Topik berikut menunjukkan cara mengatur alokasi sumber daya dinamis (DRA) untuk manajemen GPU tingkat lanjut.

Prasyarat

Sebelum menerapkan DRA di Amazon EKS, pastikan lingkungan Anda memenuhi persyaratan berikut.

Konfigurasi klaster

- Amazon EKS cluster menjalankan versi `1.33` atau yang lebih baru
- Grup node terkelola Amazon EKS (DRA saat ini hanya didukung oleh grup node terkelola dengan AL2023 dan Bottlerocket NVIDIA dioptimalkan AMIs, bukan dengan Karpenter)
- Node pekerja berkemampuan GPU NVIDIA dengan tipe instans yang sesuai

Komponen yang dibutuhkan

- Versi plugin perangkat NVIDIA `0.17.1` atau yang lebih baru
- Versi driver NVIDIA DRA `25.3.0` atau yang lebih baru

Langkah 1: Buat cluster dengan grup node yang mendukung DRA menggunakan eksctl

1. Buat file konfigurasi cluster bernama`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. Buat cluster:

```
eksctl create cluster -f dra-eks-cluster.yaml
```

Langkah 2: Menyebarkan plugin perangkat NVIDIA

Terapkan plugin perangkat NVIDIA untuk mengaktifkan penemuan GPU dasar:

1. Tambahkan repositori Helm plugin perangkat NVIDIA:

```
helm repo add nvidia https://nvidia.github.io/k8s-device-plugin
helm repo update
```

2. Buat nilai kustom untuk plugin perangkat:

```
cat <<EOF > nvidia-device-plugin-values.yaml
gfd:
  enabled: true
nfd:
  enabled: true
tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
EOF
```

3. Instal plug-in perangkat 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
```

Langkah 3: Menyebarkan bagan Helm driver NVIDIA DRA

1. Buat file `dra-driver-values.yaml` nilai untuk 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. Tambahkan repositori NVIDIA NGC Helm:

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
```

3. Instal 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
```

Langkah 4: Verifikasi instalasi DRA

1. Verifikasi bahwa sumber daya DRA API tersedia:

   ```
   kubectl api-resources | grep resource.k8s.io/v1beta1
   ```

   Berikut ini adalah output yang diharapkan:

   ```
   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. Periksa kelas perangkat yang tersedia:

   ```
   kubectl get deviceclasses
   ```

   Berikut ini adalah contoh output yang diharapkan:

   ```
   NAME                                        AGE
   compute-domain-daemon.nvidia.com            4h39m
   compute-domain-default-channel.nvidia.com   4h39m
   gpu.nvidia.com                              4h39m
   mig.nvidia.com                              4h39m
   ```

   Saat instans GPU G6 yang baru dibuat bergabung dengan cluster Amazon EKS Anda dengan DRA diaktifkan, tindakan berikut akan terjadi:

   - Driver NVIDIA DRA secara otomatis menemukan GPU A10G dan membuat dua `resourceslices` pada node itu.
   - `gpu.nvidia.com`Slice mendaftarkan perangkat GPU A10G fisik dengan spesifikasinya (memori, kemampuan komputasi, dan banyak lagi).
   - Karena A10G tidak mendukung partisi MIG, `compute-domain.nvidia.com` irisan membuat domain komputasi tunggal yang mewakili seluruh konteks komputasi GPU.

- Ini kemudian `resourceslices` dipublikasikan ke server API Kubernetes, membuat sumber daya GPU tersedia untuk penjadwalan. `resourceclaims`

  Penjadwal DRA sekarang dapat dengan cerdas mengalokasikan GPU ini ke Pod yang meminta sumber daya GPU melalui`resourceclaimtemplates`, menyediakan manajemen sumber daya yang lebih fleksibel dibandingkan dengan pendekatan plugin perangkat tradisional. Ini terjadi secara otomatis tanpa intervensi manual. Node hanya tersedia untuk beban kerja GPU setelah driver DRA menyelesaikan penemuan sumber daya dan proses pendaftaran.

  Ketika Anda menjalankan perintah berikut:

  ```
  kubectl get resourceslices
  ```

  Berikut ini adalah contoh output yang diharapkan:

  ```
  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
  ```

Lanjutkan ke the section called “Jadwalkan beban kerja GPU sederhana menggunakan alokasi sumber daya dinamis”.

Jadwalkan beban kerja GPU sederhana menggunakan alokasi sumber daya dinamis

Untuk menjadwalkan beban kerja GPU sederhana menggunakan alokasi sumber daya dinamis (DRA), lakukan langkah-langkah berikut. Sebelum melanjutkan, pastikan Anda telah mengikutithe section called “Siapkan alokasi sumber daya dinamis untuk manajemen GPU tingkat lanjut”.

1. Buat dasar `ResourceClaimTemplate` untuk alokasi GPU dengan file bernama: `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. Terapkan template:

```
kubectl apply -f basic-gpu-claim-template.yaml
```

3. Verifikasi status:

```
kubectl get resourceclaimtemplates -n gpu-test1
```

   Berikut ini adalah output contoh:

```
NAME         AGE
single-gpu   9m16s
```

4. Buat Pod yang menggunakan file `ResourceClaimTemplate` dengan nama`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. Terapkan dan pantau Pod:

```
kubectl apply -f basic-gpu-pod.yaml
```

6. Periksa status Pod:

```
kubectl get pod -n gpu-test1
```

   Berikut ini adalah contoh output yang diharapkan:

```
NAME      READY   STATUS    RESTARTS   AGE
gpu-pod   1/1     Running   0          13m
```

7. Periksa `ResourceClaim` statusnya:

```
kubectl get resourceclaims -n gpu-test1
```

   Berikut ini adalah contoh output yang diharapkan:

```
NAME                 STATE                AGE
gpu-pod-gpu0-l76cg   allocated,reserved   9m6s
```

8. Lihat log Pod untuk melihat informasi GPU:

```
kubectl logs gpu-pod -n gpu-test1
```

Berikut ini adalah contoh output yang diharapkan:

```
GPU 0: NVIDIA L4 (UUID: GPU-da7c24d7-c7e3-ed3b-418c-bcecc32af7c5)
```

Lanjutkan ke the section called “Teknik optimasi GPU dengan alokasi sumber daya dinamis” teknik optimasi GPU yang lebih canggih menggunakan DRA.

Teknik optimasi GPU dengan alokasi sumber daya dinamis

Beban kerja GPU modern membutuhkan manajemen sumber daya yang canggih untuk mencapai pemanfaatan dan efisiensi biaya yang optimal. DRA memungkinkan beberapa teknik optimasi canggih yang menangani berbagai kasus penggunaan dan kemampuan perangkat keras:

- Time-slicing memungkinkan beberapa beban kerja untuk berbagi sumber daya komputasi GPU dari waktu ke waktu, menjadikannya ideal untuk beban kerja inferensi dengan penggunaan GPU sporadis. Sebagai contoh, lihat the section called “Optimalkan beban kerja GPU dengan pengiris waktu”.
- Multi-Process service (MPS) memungkinkan eksekusi bersamaan dari beberapa proses CUDA pada satu GPU dengan isolasi yang lebih baik daripada time-slicing. Sebagai contoh, lihat the section called “Optimalkan beban kerja GPU dengan MPS”.
- Multi-Instance GPU (MIG) menyediakan partisi tingkat perangkat keras, membuat instance GPU terisolasi dengan sumber daya komputasi dan memori khusus. Sebagai contoh, lihat the section called “Optimalkan beban kerja GPU dengan GPU Multi-Instance”.
- Internode Memory Exchange (IMEX) memungkinkan komunikasi yang koheren memori di seluruh node untuk pelatihan terdistribusi pada sistem NVIDIA. GB200 Sebagai contoh, lihat the section called “Optimalkan beban kerja GPU dengan IMEX menggunakan instans P6e GB200”.

Teknik-teknik ini secara signifikan dapat meningkatkan pemanfaatan sumber daya. Organizations melaporkan pemanfaatan GPU meningkat dari 30-40% dengan alokasi tradisional menjadi 80-90% dengan strategi berbagi yang dioptimalkan. Pilihan teknik tergantung pada karakteristik beban kerja, persyaratan isolasi, dan kemampuan perangkat keras.

Optimalkan beban kerja GPU dengan pengiris waktu

Time-slicing memungkinkan beberapa beban kerja untuk berbagi sumber daya komputasi GPU dengan menjadwalkannya untuk berjalan secara berurutan pada GPU fisik yang sama. Ini sangat ideal untuk beban kerja inferensi dengan penggunaan GPU sporadis.

Lakukan langkah-langkah berikut.

1. Tentukan `ResourceClaimTemplate` for time-slicing dengan file bernama: `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. Tentukan Pod menggunakan time-slicing dengan file bernama: `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. Terapkan template dan Pod:

```
kubectl apply -f timeslicing-claim-template.yaml
kubectl apply -f timeslicing-pod.yaml
```

4. Pantau klaim sumber daya:

```
kubectl get resourceclaims -n timeslicing-gpu -w
```

Berikut ini adalah output contoh:

```
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
```

Pod Pertama (`inference-pod-1`)

- Negara: `allocated,reserved`
- Artinya: DRA menemukan GPU yang tersedia dan memesannya untuk Pod ini

- Status pod: Mulai berjalan segera

Pod Kedua (`training-pod-2`)

- Negara: `pending`
- Artinya: Menunggu DRA mengonfigurasi pengiris waktu pada GPU yang sama
- Status pod: Menunggu untuk dijadwalkan
- Negara akan beralih dari `pending` `allocated,reserved` ke `running`

Optimalkan beban kerja GPU dengan MPS

Multi-Process Service (MPS) memungkinkan eksekusi bersamaan dari beberapa konteks CUDA pada satu GPU dengan isolasi yang lebih baik daripada time-slicing.

Lakukan langkah-langkah berikut.

1. Tentukan `ResourceClaimTemplate` untuk MPS dengan file bernama`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. Mendefinisikan Pod menggunakan MPS dengan file bernama`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. Terapkan template dan buat beberapa MPS Pod:

```
kubectl apply -f mps-claim-template.yaml
kubectl apply -f mps-pod.yaml
```

4. Pantau klaim sumber daya:

```
kubectl get resourceclaims -n mps-gpu -w
```

   Berikut ini adalah output contoh:

```
NAME                                                STATE                AGE
mps-multi-container-pod-shared-gpu-claim-2p9kx   allocated,reserved   86s
```

Konfigurasi ini menunjukkan berbagi GPU sejati menggunakan NVIDIA Multi-Process Service (MPS) melalui dynamic resource allocation (DRA). Tidak seperti time-slicing di mana beban kerja bergiliran menggunakan GPU secara berurutan, MPS memungkinkan kedua kontainer berjalan secara bersamaan pada GPU fisik yang sama. Insight kuncinya adalah bahwa berbagi MPS DRA membutuhkan beberapa kontainer dalam satu Pod, bukan beberapa Pod terpisah. Ketika di-deploy, driver DRA mengalokasikan satu `ResourceClaim` ke Pod dan secara otomatis mengkonfigurasi MPS untuk memungkinkan kedua kontainer inferensi dan pelatihan untuk mengeksekusi secara bersamaan.

Setiap kontainer mendapatkan ruang memori GPU yang terisolasi dan sumber daya komputasi, dengan daemon MPS mengoordinasikan akses ke perangkat keras yang mendasarinya. Anda dapat memverifikasi bahwa ini berfungsi dengan melakukan hal berikut:

- Memeriksa`nvidia-smi`, yang akan menampilkan kedua kontainer sebagai proses M+C (`MPS + Compute`) berbagi perangkat GPU yang sama.
- Memantau log dari kedua kontainer, yang akan menampilkan stempel waktu yang disisipkan yang membuktikan eksekusi simultan.

Pendekatan ini memaksimalkan pemanfaatan GPU dengan memungkinkan beban kerja pelengkap untuk berbagi perangkat keras GPU yang mahal secara efisien, daripada membiarkannya kurang dimanfaatkan oleh satu proses.

Kontainer1: **`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 |
+-----------------------------------------------------------------------------------------
+
```

Kontainer2: **`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 |
+-----------------------------------------------------------------------------------------
+
```

Optimalkan beban kerja GPU dengan GPU Multi-Instance

Multi-instance GPU (MIG) menyediakan partisi tingkat perangkat keras, membuat instance GPU terisolasi dengan sumber daya komputasi dan memori khusus.

Menggunakan partisi MIG dinamis dengan berbagai profil memerlukan Operator GPU NVIDIA. Operator GPU NVIDIA menggunakan MIG Manager untuk membuat profil MIG dan me-reboot instance GPU seperti P4D, P4De, P5, P6, dan lainnya untuk menerapkan perubahan konfigurasi. Operator GPU mencakup kemampuan manajemen MIG yang komprehensif melalui komponen MIG Manager, yang mengawasi perubahan label node dan secara otomatis menerapkan konfigurasi MIG yang sesuai. Ketika perubahan profil MIG diminta, operator dengan anggun mematikan semua klien GPU, menerapkan geometri partisi baru, dan memulai ulang layanan yang terpengaruh. Proses ini memerlukan reboot node untuk instance GPU untuk memastikan transisi status GPU yang bersih. Inilah sebabnya mengapa mengaktifkan `WITH0REB00T=true` konfigurasi MIG Manager sangat penting untuk penerapan MIG yang sukses.

Anda memerlukan Driver NVIDIA DRA dan Operator GPU NVIDIA untuk bekerja dengan MIG di Amazon EKS. Anda tidak memerlukan Plugin Perangkat NVIDIA dan Eksportir DCGM selain ini karena ini adalah bagian dari Operator GPU NVIDIA. Karena EKS NVIDIA AMIs datang dengan Driver NVIDIA yang sudah diinstal sebelumnya, kami menonaktifkan penyebaran driver oleh Operator GPU untuk menghindari konflik dan memanfaatkan driver yang dioptimalkan yang sudah ada pada instance. Driver NVIDIA DRA menangani alokasi sumber daya dinamis untuk instans MIG, sementara Operator GPU mengelola seluruh siklus hidup GPU. Ini termasuk konfigurasi MIG, fungsionalitas plugin perangkat, pemantauan melalui DCGM, dan penemuan fitur node. Pendekatan terintegrasi ini memberikan solusi lengkap untuk manajemen GPU perusahaan, dengan isolasi tingkat perangkat keras dan kemampuan alokasi sumber daya dinamis.

Langkah 1: Menyebarkan Operator GPU NVIDIA

1. Tambahkan repositori Operator GPU NVIDIA:

```
helm repo add nvidia https://nvidia.github.io/gpu-operator
helm repo update
```

2. Buat `gpu-operator-values.yaml` file:

```
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. Instal GPU Operator menggunakan `gpu-operator-values.yaml` file:

```
helm install gpu-operator nvidia/gpu-operator \
  --namespace gpu-operator \
  --create-namespace \
  --version v25.3.1 \
  --values gpu-operator-values.yaml
```

   Bagan Helm ini menyebarkan komponen berikut dan beberapa profil MIG:

   - Plugin Perangkat (penjadwalan sumber daya GPU)
   - Eksportir DCGM (metrik dan pemantauan GPU)
   - Penemuan Fitur Node (NFD - pelabelan perangkat keras)
   - Penemuan Fitur GPU (GFD - pelabelan khusus GPU)
   - Manajer MIG (Partisi GPU multi-instance)
   - Toolkit Kontainer (runtime kontainer GPU)
   - Operator Controller (manajemen siklus hidup)

4. Verifikasi Pod penerapan:

```
kubectl get pods -n gpu-operator
```

   Berikut ini adalah output contoh:

```
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. Buat klaster Amazon EKS dengan grup node terkelola P4DE untuk menguji contoh MIG:

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: dra-eks-cluster
  region: us-east-1
  version: '1.33'

managedNodeGroups:
# P4DE MIG Node Group with Capacity Block Reservation
- name: p4de-mig-nodes
  amiFamily: AmazonLinux2023
  instanceType: p4de.24xlarge

  # Capacity settings
  desiredCapacity: 0
  minSize: 0
  maxSize: 1

  # Use specific subnet in us-east-1b for capacity reservation
  subnets:
    - us-east-1b

  # AL2023 NodeConfig for RAID0 local storage only
  nodeadmConfig:
    apiVersion: node.eks.aws/v1alpha1
    kind: NodeConfig
    spec:
      instance:
```

```
        localStorage:
          strategy: RAID0

  # Node labels for MIG configuration
  labels:
    nvidia.com/gpu.present: "true"
    nvidia.com/gpu.product: "A100-SXM4-80GB"
    nvidia.com/mig.config: "p4de-half-balanced"
    node-type: "p4de"
    vpc.amazonaws.com/efa.present: "true"
    accelerator: "nvidia"

  # Node taints
  taints:
    - key: nvidia.com/gpu
      value: "true"
      effect: NoSchedule

  # EFA support
  efaEnabled: true

  # Placement group for high-performance networking
  placementGroup:
    groupName: p4de-placement-group
    strategy: cluster

  # Capacity Block Reservation (CBR)
  # Ensure CBR ID matches the subnet AZ with the Nodegroup subnet
  spot: false
  capacityReservation:
    capacityReservationTarget:
      capacityReservationId: "cr-abcdefghij"  # Replace with your capacity
 reservation ID
```

NVIDIA GPU Operator menggunakan label ditambahkan ke node `nvidia.com/mig.config: "p4de-half-balanced"` dan partisi GPU dengan profil yang diberikan.

6. Login ke `p4de` instance.
7. Jalankan perintah berikut:

```
nvidia-smi -L
```

Anda akan melihat contoh output berikut:

```
[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)
```

Operator GPU NVIDIA telah berhasil menerapkan profil `p4de-half-balanced` MIG ke instance P4DE Anda, membuat partisi GPU tingkat perangkat keras seperti yang dikonfigurasi. Begini cara kerja partisi:

Operator GPU menerapkan konfigurasi ini dari profil MIG yang disematkan:

```
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
```

Dari `nvidia-smi -L` output Anda, inilah yang dibuat oleh Operator GPU:

- MiG-enabled GPUs (0-3): perangkat keras dipartisi
  - GPU 0: NVIDIA A100- -80GB SXM4
    - MIG 3g.40gb Perangkat 0 - Beban kerja besar (memori 40GB, 42) SMs
    - MIG 2g.20gb Perangkat 1 - Beban kerja sedang (memori 20GB, 28) SMs
    - MIG 1g.10gb Perangkat 2 - Beban kerja kecil (memori 10GB, 14) SMs
    - MIG 1g.10gb Perangkat 3 - Beban kerja kecil (memori 10GB, 14) SMs
  - GPU 1: NVIDIA A100- -80GB SXM4
    - MIG 3g.40gb Perangkat 0 - Tata letak partisi identik
    - Perangkat MIG 2g.20gb 1
    - Perangkat MIG 1g.10gb 2
    - Perangkat MIG 1g.10gb 3
  - GPU 2 dan GPU 3 — Pola yang sama dengan GPU 0 dan GPU 1
- Penuh GPUs (4-7): Tidak ada partisi MIG
  - GPU 4: NVIDIA A100- SXM4 -80GB - GPU 80GB Penuh
  - GPU 5: NVIDIA A100- SXM4 -80GB - GPU 80GB Penuh
  - GPU 6: NVIDIA A100- SXM4 -80GB - GPU 80GB Penuh
  - GPU 7: NVIDIA A100- SXM4 -80GB - GPU 80GB Penuh

Setelah Operator GPU NVIDIA membuat partisi MIG, Driver NVIDIA DRA secara otomatis mendeteksi instans terisolasi perangkat keras ini dan membuatnya tersedia untuk alokasi sumber daya dinamis di Kubernetes. Driver DRA menemukan setiap instance MIG dengan profil spesifiknya (1g.10gb, 2g.20gb, 3g.40gb) dan mengeksposnya sebagai sumber daya yang dapat dijadwalkan melalui kelas perangkat. `mig.nvidia.com`

Driver DRA terus memantau topologi MIG dan memelihara inventaris instance yang tersedia di semua. GPUs Ketika Pod meminta profil MIG tertentu melalui a`ResourceClaimTemplate`, driver DRA secara cerdas memilih instance MIG yang sesuai dari GPU yang tersedia, memungkinkan multi-tenancy tingkat perangkat keras yang sebenarnya. Alokasi dinamis ini memungkinkan

beberapa beban kerja yang terisolasi berjalan secara bersamaan pada GPU fisik yang sama sambil mempertahankan batasan sumber daya yang ketat dan jaminan kinerja.

Langkah 2: Uji alokasi sumber daya MIG

Sekarang mari kita jalankan beberapa contoh untuk menunjukkan bagaimana DRA secara dinamis mengalokasikan instance MIG ke beban kerja yang berbeda. Terapkan `resourceclaimtemplates` dan uji pod untuk melihat bagaimana driver DRA menempatkan beban kerja di seluruh partisi MIG yang tersedia, memungkinkan beberapa kontainer untuk berbagi sumber daya GPU dengan isolasi tingkat perangkat keras.

1. Buat `mig-claim-template.yaml` untuk memuat 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. Terapkan tiga templat:

```
kubectl apply -f mig-claim-template.yaml
```

3. Jalankan perintah berikut:

```
kubectl get resourceclaimtemplates -n mig-gpu
```

   Berikut ini adalah output contoh:

```
NAME                  AGE
mig-large-template    71m
mig-medium-template   71m
```

```
mig-small-template    71m
```

4. Buat `mig-pod.yaml` untuk menjadwalkan beberapa pekerjaan untuk memanfaatkan ini`resourceclaimtemplates`:

```
---
# ConfigMap containing Python scripts for MIG pods
apiVersion: v1
kind: ConfigMap
metadata:
  name: mig-scripts-configmap
  namespace: mig-gpu
data:
  large-training-script.py: |
    import torch
    import torch.nn as nn
    import torch.optim as optim
    import time
    import os

    print(f"=== LARGE TRAINING POD (3g.40gb) ===")
    print(f"Process ID: {os.getpid()}")
    print(f"GPU available: {torch.cuda.is_available()}")
    print(f"GPU count: {torch.cuda.device_count()}")

    if torch.cuda.is_available():
        device = torch.cuda.current_device()
        print(f"Using GPU: {torch.cuda.get_device_name(device)}")
        print(f"GPU Memory: {torch.cuda.get_device_properties(device).total_memory /
 1e9:.1f} GB")

        # Large model for 3g.40gb instance
        model = nn.Sequential(
            nn.Linear(2048, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 10)
        ).cuda()

        optimizer = optim.Adam(model.parameters())
```

```
        criterion = nn.CrossEntropyLoss()

        print(f"Model parameters: {sum(p.numel() for p in model.parameters())}")

        # Training loop
        for epoch in range(100):
            # Large batch for 3g.40gb
            x = torch.randn(256, 2048).cuda()
            y = torch.randint(0, 10, (256,)).cuda()

            optimizer.zero_grad()
            output = model(x)
            loss = criterion(output, y)
            loss.backward()
            optimizer.step()

            if epoch % 10 == 0:
                print(f"Large Training - Epoch {epoch}, Loss: {loss.item():.4f}, GPU
Memory: {torch.cuda.memory_allocated()/1e9:.2f}GB")
            time.sleep(3)

        print("Large training completed on 3g.40gb MIG instance")

 medium-training-script.py: |
   import torch
   import torch.nn as nn
   import torch.optim as optim
   import time
   import os

   print(f"=== MEDIUM TRAINING POD (2g.20gb) ===")
   print(f"Process ID: {os.getpid()}")
   print(f"GPU available: {torch.cuda.is_available()}")
   print(f"GPU count: {torch.cuda.device_count()}")

   if torch.cuda.is_available():
       device = torch.cuda.current_device()
       print(f"Using GPU: {torch.cuda.get_device_name(device)}")
       print(f"GPU Memory: {torch.cuda.get_device_properties(device).total_memory /
1e9:.1f} GB")

       # Medium model for 2g.20gb instance
       model = nn.Sequential(
           nn.Linear(1024, 512),
```

```
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 10)
        ).cuda()

        optimizer = optim.Adam(model.parameters())
        criterion = nn.CrossEntropyLoss()

        print(f"Model parameters: {sum(p.numel() for p in model.parameters())}")

        # Training loop
        for epoch in range(100):
            # Medium batch for 2g.20gb
            x = torch.randn(128, 1024).cuda()
            y = torch.randint(0, 10, (128,)).cuda()

            optimizer.zero_grad()
            output = model(x)
            loss = criterion(output, y)
            loss.backward()
            optimizer.step()

            if epoch % 10 == 0:
                print(f"Medium Training - Epoch {epoch}, Loss: {loss.item():.4f}, GPU
Memory: {torch.cuda.memory_allocated()/1e9:.2f}GB")
            time.sleep(4)

        print("Medium training completed on 2g.20gb MIG instance")

 small-inference-script.py: |
   import torch
   import torch.nn as nn
   import time
   import os

   print(f"=== SMALL INFERENCE POD (1g.10gb) ===")
   print(f"Process ID: {os.getpid()}")
   print(f"GPU available: {torch.cuda.is_available()}")
   print(f"GPU count: {torch.cuda.device_count()}")

   if torch.cuda.is_available():
       device = torch.cuda.current_device()
       print(f"Using GPU: {torch.cuda.get_device_name(device)}")
```

```
        print(f"GPU Memory: {torch.cuda.get_device_properties(device).total_memory /
 1e9:.1f} GB")

        # Small model for 1g.10gb instance
        model = nn.Sequential(
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 10)
        ).cuda()

        print(f"Model parameters: {sum(p.numel() for p in model.parameters())}")

        # Inference loop
        for i in range(200):
            with torch.no_grad():
                # Small batch for 1g.10gb
                x = torch.randn(32, 512).cuda()
                output = model(x)
                prediction = torch.argmax(output, dim=1)

                if i % 20 == 0:
                    print(f"Small Inference - Batch {i}, Predictions:
 {prediction[:5].tolist()}, GPU Memory: {torch.cuda.memory_allocated()/1e9:.2f}GB")
            time.sleep(2)

        print("Small inference completed on 1g.10gb MIG instance")

---
# Pod 1: Large training workload (3g.40gb)
apiVersion: v1
kind: Pod
metadata:
  name: mig-large-training-pod
  namespace: mig-gpu
  labels:
    app: mig-large-training
    workload-type: training
spec:
  restartPolicy: Never
  containers:
  - name: large-training-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "/scripts/large-training-script.py"]
    volumeMounts:
```

```
    - name: script-volume
      mountPath: /scripts
      readOnly: true
    resources:
      claims:
      - name: mig-large-claim
  resourceClaims:
  - name: mig-large-claim
    resourceClaimTemplateName: mig-large-template
  nodeSelector:
    node.kubernetes.io/instance-type: p4de.24xlarge
    nvidia.com/gpu.present: "true"
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
  volumes:
  - name: script-volume
    configMap:
      name: mig-scripts-configmap
      defaultMode: 0755

---
# Pod 2: Medium training workload (2g.20gb) - can run on SAME GPU as Pod 1
apiVersion: v1
kind: Pod
metadata:
  name: mig-medium-training-pod
  namespace: mig-gpu
  labels:
    app: mig-medium-training
    workload-type: training
spec:
  restartPolicy: Never
  containers:
  - name: medium-training-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "/scripts/medium-training-script.py"]
    volumeMounts:
    - name: script-volume
      mountPath: /scripts
      readOnly: true
    resources:
      claims:
```

```
      - name: mig-medium-claim
  resourceClaims:
  - name: mig-medium-claim
    resourceClaimTemplateName: mig-medium-template
  nodeSelector:
    node.kubernetes.io/instance-type: p4de.24xlarge
    nvidia.com/gpu.present: "true"
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
  volumes:
  - name: script-volume
    configMap:
      name: mig-scripts-configmap
      defaultMode: 0755

---
# Pod 3: Small inference workload (1g.10gb) - can run on SAME GPU as Pod 1 & 2
apiVersion: v1
kind: Pod
metadata:
  name: mig-small-inference-pod
  namespace: mig-gpu
  labels:
    app: mig-small-inference
    workload-type: inference
spec:
  restartPolicy: Never
  containers:
  - name: small-inference-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "/scripts/small-inference-script.py"]
    volumeMounts:
    - name: script-volume
      mountPath: /scripts
      readOnly: true
    resources:
      claims:
      - name: mig-small-claim
  resourceClaims:
  - name: mig-small-claim
    resourceClaimTemplateName: mig-small-template
  nodeSelector:
```

```
    node.kubernetes.io/instance-type: p4de.24xlarge
    nvidia.com/gpu.present: "true"
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
  volumes:
  - name: script-volume
    configMap:
      name: mig-scripts-configmap
      defaultMode: 0755
```

5. Terapkan spesifikasi ini, yang seharusnya menyebarkan tiga Pod:

```
kubctl apply -f mig-pod.yaml
```

   Pod ini harus dijadwalkan oleh driver DRA.

6. Periksa log DRA driver Pod dan Anda akan melihat output yang mirip dengan ini:

```
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. Verifikasi `resourceclaims` untuk melihat status Pod:

```
kubectl get resourceclaims -n mig-gpu -w
```

Berikut ini adalah output contoh:

```
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
```

Seperti yang Anda lihat, semua Pod dipindahkan dari pending ke `allocated,reserved` driver DRA.

8. Jalankan `nvidia-smi` dari node. Anda akan melihat tiga prosesor Python sedang berjalan:

```
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 |**
+-----------------------------------------------------------------------------------------
+
```

Optimalkan beban kerja GPU dengan IMEX menggunakan instans P6e GB200

IMEX (Internode Memory Exchange) memungkinkan komunikasi yang koheren memori di seluruh node untuk pelatihan terdistribusi di NVIDIA. GB200 UltraServers

Lakukan langkah-langkah berikut.

1. Tentukan pelatihan `ComputeDomain` untuk multi-node dengan file bernama`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. Mendefinisikan Pod menggunakan saluran IMEX dengan file bernama`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**
>
> Ini membutuhkan instance P6e GB200 .

3. Menyebarkan IMEX dengan menerapkan `ComputeDomain` dan template:

```
kubectl apply -f imex-claim-template.yaml
```

```
kubectl apply -f imex-compute-domain.yaml
kubectl apply -f imex-pod.yaml
```

4. Periksa `ComputeDomain` statusnya.

```
kubectl get computedomain distributed-training-domain
```

5. Pantau penyebaran daemon IMEX.

```
kubectl get pods -n nvidia-dra-driver -l resource.nvidia.com/computeDomain
```

6. Periksa saluran IMEX di Pod:

```
kubectl exec imex-distributed-training -- ls -la /dev/nvidia-caps-imex-channels/
```

7. Lihat log Pod:

```
kubectl logs imex-distributed-training
```

   Berikut ini adalah contoh output yang diharapkan:

```
=== 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
```

Untuk informasi selengkapnya, lihat contoh NVIDIA di GitHub.

Jaringan

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Pertimbangkan Bandwidth Jaringan yang Lebih Tinggi atau Adaptor Kain Elastis Untuk Aplikasi dengan Komunikasi Antar Node Tinggi

Untuk beban kerja pelatihan terdistribusi di Amazon EKS dengan tuntutan komunikasi antar simpul yang tinggi, pertimbangkan untuk memilih instans dengan bandwidth jaringan yang lebih tinggi atau Elastic Fabric Adapter (EFA). Kinerja jaringan yang tidak memadai dapat menghambat transfer data, memperlambat tugas pembelajaran mesin seperti pelatihan multi-GPU terdistribusi. Perhatikan bahwa beban kerja inferensi biasanya tidak memiliki komunikasi antar simpul yang tinggi.

Contoh

Misalnya, menggunakan 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
```

Pastikan alat seperti MPI dan NCCL dipasang di gambar kontainer Anda untuk memanfaatkan EFA untuk pekerjaan pelatihan.

Tingkatkan jumlah alamat IP yang tersedia untuk mengaktifkan waktu peluncuran pod yang lebih cepat

Di EKS, setiap pod membutuhkan alamat IP dari blok CIDR VPC. Saat klaster Anda menskalakan dengan lebih banyak node dan pod, Anda berisiko kehabisan alamat IP atau kinerja yang lebih lambat, tetapi mengaktifkan delegasi awalan dapat mengurangi masalah ini dengan mengalokasikan rentang IP terlebih dahulu dan mengurangi panggilan API EC2, menghasilkan waktu peluncuran pod yang lebih cepat dan peningkatan skalabilitas.

Mengaktifkan delegasi awalan setelah membuat cluster Anda memungkinkan VPC Container Network Interface (CNI) untuk menetapkan awalan IP (/28, masing-masing memberikan 16 alamat IP) ke antarmuka jaringan pada instans EC2. Ini berarti setiap node dapat mendukung lebih banyak pod, mengurangi risiko kekurangan IP. Misalnya, pada sebuah `c5.4xlarge` instance, Anda dapat mendukung hingga 110 pod dengan delegasi awalan.

Sementara delegasi awalan sangat penting untuk mengoptimalkan penggunaan IP di lingkungan dengan banyak pod kecil, AI/ML beban kerja sering menggunakan pod yang lebih sedikit dan lebih besar (misalnya, satu pod per GPU). Mengaktifkan delegasi awalan memungkinkan VPC CNI untuk mengalokasikan awalan terlebih dahulu untuk startup pod yang lebih cepat dengan mempertahankan kolam yang hangat. Ini berarti alamat IP sudah tersedia, mengurangi waktu yang dibutuhkan untuk inisialisasi pod dibandingkan dengan alokasi sesuai permintaan dalam mode non-awalan. Dalam kasus seperti itu, penghematan IP dari mengaktifkan delegasi awalan menawarkan manfaat kinerja untuk beban kerja. AI/ML Dengan mengurangi jumlah panggilan API EC2 yang diperlukan untuk konfigurasi alamat IP dan pra-alokasi rentang IP, menggunakan delegasi awalan memungkinkan waktu peluncuran pod yang lebih cepat, yang sangat bermanfaat untuk penskalaan beban kerja dengan cepat. AI/ML

Untuk mengaktifkan delegasi awalan:

```
kubectl set env daemonset/aws-node -n kube-system ENABLE_PREFIX_DELEGATION=true
```

Pastikan perencanaan yang tepat untuk subnet VPC untuk menghindari kelelahan alamat IP, terutama dalam penerapan besar, dan kelola blok CIDR untuk menghindari tumpang tindih. VPCs Untuk mempelajari lebih lanjut, lihat Mengoptimalkan Pemanfaatan Alamat IP dan Menetapkan lebih banyak alamat IP ke node Amazon EKS dengan awalan.

Keamanan

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Keamanan dan Kepatuhan

Pertimbangkan S3 dengan KMS untuk penyimpanan yang sesuai dengan enkripsi

Kecuali Anda menentukan sebaliknya, semua bucket S3 menggunakan SSE-S3 secara default untuk mengenkripsi objek saat istirahat. Namun, Anda dapat memilih untuk mengonfigurasi bucket untuk menggunakan enkripsi sisi server dengan kunci AWS Key Management Service (AWS KMS) (SSE-KMS). Kontrol keamanan di AWS KMS dapat membantu Anda memenuhi persyaratan kepatuhan terkait enkripsi. Anda dapat menggunakan kunci KMS ini untuk melindungi data Anda di bucket Amazon S3. Saat Anda menggunakan enkripsi SSE-KMS dengan bucket S3, kunci AWS KMS harus berada di Wilayah yang sama dengan bucket.

Konfigurasikan bucket tujuan umum Anda untuk menggunakan Kunci Bucket S3 untuk SSE-KMS, untuk mengurangi biaya permintaan AWS KMS hingga 99 persen dengan mengurangi lalu lintas permintaan dari Amazon S3 ke AWS KMS. Kunci Bucket S3 selalu diaktifkan untuk `GET` dan `PUT` beroperasi di bucket direktori dan tidak dapat dinonaktifkan.

Perhatikan bahwa Amazon S3 Express One Zone menggunakan jenis bucket tertentu yang disebut bucket direktori S3. Bucket direktori khusus untuk kelas penyimpanan S3 Express One Zone dan memungkinkan akses berkinerja tinggi dan latensi rendah. Untuk mengonfigurasi enkripsi bucket default pada bucket direktori S3, gunakan AWS CLI, dan tentukan ID kunci KMS atau ARN, bukan alias, seperti pada contoh berikut:

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"}}]}'
```

Pastikan peran IAM pod EKS Anda memiliki izin KMS (misalnya,`kms:Decrypt`) untuk mengakses objek terenkripsi. Uji ini di lingkungan pementasan dengan mengunggah model sampel ke bucket, memasangnya di pod (misalnya, melalui driver Mountpoint S3 CSI), dan memverifikasi pod dapat membaca data terenkripsi tanpa kesalahan. Audit log melalui AWS CloudTrail untuk mengonfirmasi kepatuhan terhadap persyaratan enkripsi. Lihat Dokumentasi KMS untuk detail penyiapan dan manajemen kunci.

Penyimpanan

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Manajemen dan Penyimpanan Data

Menerapkan Model AI ke Pod Menggunakan Driver CSI

Beban kerja AI/ML seringkali memerlukan akses ke artefak model besar (misalnya, bobot terlatih, konfigurasi), dan pod memerlukan cara yang andal dan dapat diskalakan untuk mengaksesnya tanpa menyematkannya dalam gambar kontainer, yang dapat meningkatkan ukuran gambar dan waktu tarik registri Container. Untuk mengurangi overhead operasional pengelolaan volume mount, kami merekomendasikan penerapan model AI ke pod dengan memasang layanan penyimpanan Amazon (misalnya, S3, untuk Lustre, FSx untuk FSx OpenZFS, EFS) sebagai Persistent Volume () menggunakan driver CSI masing-masing. PVs Untuk detail implementasi, lihat topik selanjutnya di bagian ini.

Optimalkan Penyimpanan untuk Cache Model ML di EKS

Memanfaatkan solusi penyimpanan yang optimal sangat penting untuk meminimalkan latensi start-up pod dan aplikasi, mengurangi penggunaan memori, mendapatkan tingkat kinerja yang diinginkan untuk mempercepat beban kerja, dan memastikan skalabilitas beban kerja ML. Beban kerja ML sering bergantung pada file model (bobot), yang bisa berukuran besar dan memerlukan akses bersama ke data di seluruh pod atau node. Memilih solusi penyimpanan yang optimal bergantung pada karakteristik beban kerja Anda, seperti efisiensi simpul tunggal, akses multi-node, persyaratan latensi, kendala biaya, dan juga persyaratan integrasi data (seperti dengan repositori data Amazon S3). Kami merekomendasikan untuk membandingkan solusi penyimpanan yang berbeda dengan beban kerja Anda untuk memahami mana yang memenuhi persyaratan Anda, dan kami telah menyediakan opsi berikut untuk membantu Anda mengevaluasi berdasarkan persyaratan beban kerja Anda.

Driver EKS CSI mendukung layanan AWS Storage berikut, masing-masing memiliki driver CSI sendiri dan dilengkapi dengan kekuatan mereka sendiri untuk alur kerja AI dan ML:

- Mountpoint untuk Amazon S3

- Amazon FSx untuk Lustre
- Amazon FSx untuk OpenZFS
- Amazon EFS
- Amazon EBS

Pilihan layanan AWS Storage bergantung pada arsitektur penerapan, skala, persyaratan kinerja, dan strategi biaya Anda. Driver CSI penyimpanan harus diinstal pada kluster EKS Anda, yang memungkinkan driver CSI untuk membuat dan mengelola Persistent Volume (PV) di luar siklus hidup Pod. Dengan menggunakan driver CSI, Anda dapat membuat definisi PV dari layanan AWS Storage yang didukung sebagai sumber daya kluster EKS. Pod kemudian dapat mengakses volume penyimpanan ini untuk volume datanya melalui pembuatan Persistent Volume Claim (PVC) untuk PV. Bergantung pada layanan penyimpanan AWS dan skenario penerapan Anda, satu PVC (dan PV terkait) dapat dilampirkan ke beberapa Pod untuk beban kerja. Misalnya, untuk pelatihan ML, data pelatihan bersama disimpan di PV dan diakses oleh beberapa Pod; untuk inferensi online real-time, model LLM di-cache pada PV dan diakses oleh beberapa Pod. Contoh file PV dan PVC YAMB untuk layanan AWS Storage disediakan di bawah ini untuk membantu Anda memulai.

Kinerja pemantauan Kinerja disk yang buruk dapat menunda pembacaan gambar kontainer, meningkatkan latensi startup pod, dan menurunkan inferensi atau throughput pelatihan. Gunakan Amazon CloudWatch untuk memantau metrik kinerja untuk layanan penyimpanan AWS Anda. Saat Anda mengidentifikasi kemacetan kinerja, ubah parameter konfigurasi penyimpanan Anda untuk mengoptimalkan kinerja.

Skenario: Beberapa instans GPU beban kerja

Amazon FSx for Lustre: Dalam skenario di mana Anda memiliki beberapa lingkungan instans komputasi GPU EC2 dengan beban kerja dinamis throughput bandwidth tinggi yang sensitif terhadap latensi, seperti pelatihan terdistribusi dan penayangan model, dan Anda memerlukan integrasi repositori data Amazon S3 asli, kami merekomendasikan Amazon untuk Lustre. FSx FSx untuk Lustre menyediakan sistem file paralel kinerja tinggi yang dikelola sepenuhnya yang dirancang untuk beban kerja intensif komputasi seperti komputasi kinerja tinggi (HPC), Machine Learning.

Anda dapat Menginstal driver FSx for Lustre CSI untuk memasang sistem FSx file di EKS sebagai Persistent Volume (PV), lalu menerapkan sistem file Lustre sebagai cache kinerja tinggi mandiri atau sebagai sistem file terkait S3 FSx untuk bertindak sebagai cache kinerja tinggi untuk data S3, memberikan throughput cepat dan tinggi untuk akses data di seluruh instance komputasi GPU Anda. I/O FSx untuk Lustre dapat digunakan dengan opsi penyimpanan Scratch-SSD atau persistent-SSD:

- Penyimpanan Scratch-SSD: Direkomendasikan untuk beban kerja yang bersifat sementara atau berumur pendek (jam), dengan kapasitas throughput tetap per TIB yang disediakan.
- Penyimpanan SSD persisten: Direkomendasikan untuk beban kerja yang kritis dan berjalan lama yang memerlukan tingkat ketersediaan tertinggi, misalnya simulasi HPC, analisis data besar, atau pelatihan Machine Learning. Dengan penyimpanan SSD persisten, Anda dapat mengonfigurasi kapasitas penyimpanan dan kapasitas throughput (per-TIB) yang diperlukan.

Pertimbangan kinerja:

- Pod administratif FSx untuk mengelola sistem file Lustre: Konfigurasikan Pod “administratif” yang memiliki klien lustre diinstal dan memiliki sistem file FSx yang terpasang. Ini akan memungkinkan titik akses untuk mengaktifkan fine-tuning sistem file FSx, dan juga dalam situasi di mana Anda perlu melakukan pra-pemanasan sistem file FSx dengan data pelatihan ML atau model LLM Anda sebelum memulai instans komputasi GPU Anda. Hal ini sangat penting jika arsitektur Anda menggunakan GPU/compute instans Amazon EC2 berbasis Spot, di mana Anda dapat menggunakan Pod administratif untuk “menghangatkan” atau “pra-muat” data yang diinginkan ke dalam sistem file FSx, sehingga data siap diproses saat menjalankan instans Amazon EC2 berbasis Spot.
- Elastic Fabric Adapter (EFA): Jenis penyebaran penyimpanan SSD persisten mendukung Elastic Fabric Adapter (EFA), di mana penggunaan EFA sangat ideal untuk kinerja tinggi dan beban kerja berbasis GPU berbasis throughput. Perhatikan bahwa FSx untuk Lustre mendukung NVIDIA GPUDirect Storage (GDS), di mana GDS adalah teknologi yang menciptakan jalur data langsung antara penyimpanan lokal atau jarak jauh dan memori GPU, untuk memungkinkan akses data yang lebih cepat.
- Kompresi: Aktifkan kompresi data pada sistem file jika Anda memiliki jenis file yang dapat dikompresi. Ini dapat membantu meningkatkan kinerja karena kompresi data mengurangi jumlah data yang ditransfer antara FSx untuk server file Lustre dan penyimpanan.
- Konfigurasi striping sistem file lustre:
  - Data striping: Memungkinkan Luster FSx untuk mendistribusikan data file di beberapa Object Storage Target (OSTs) dalam sistem file Lustre memaksimalkan akses paralel dan throughput, terutama untuk pekerjaan pelatihan ML. skala besar.
  - Pengupasan sistem file mandiri: Secara default, konfigurasi striping Lustre 4 komponen dibuat untuk Anda melalui kemampuan tata letak file Progresif (PFL) untuk Lustre. FSx Dalam sebagian besar skenario, Anda tidak perlu memperbarui hitungan/ukuran garis PFL Lustre default. Jika

Anda perlu menyesuaikan striping data Lustre, maka Anda dapat secara manual menyesuaikan lustre striping dengan mengacu pada parameter striping dari sistem file for Lustre. FSx

- Sistem File Tertaut S3: File yang diimpor ke sistem FSx file menggunakan integrasi Amazon S3 asli (Data Repository Association atau DRA) tidak menggunakan tata letak PFL default, melainkan menggunakan tata letak dalam parameter sistem file. `ImportedFileChunkSize` File yang diimpor S3 yang lebih besar dari file `ImportedFileChunkSize` akan disimpan di beberapa OSTs dengan jumlah garis berdasarkan nilai yang `ImportedFileChunkSize` ditentukan (default 1GiB). Jika Anda memiliki file besar, kami sarankan menyetel parameter ini ke nilai yang lebih tinggi.
- Penempatan: Menerapkan sistem file FSx for Lustre di Availability Zone yang sama dengan node komputasi atau GPU Anda untuk mengaktifkan akses latensi terendah ke data, hindari pola akses akses lintas Availability Zone. Jika Anda memiliki beberapa node GPU yang terletak di Availability Zone yang berbeda, sebaiknya gunakan sistem FSx file di setiap Availability Zone untuk akses data latensi rendah.

Contoh

Definisi Persistent Volume (PV) untuk sistem file FSx for Lustre, menggunakan Static Provisioning (di mana FSx instance telah disediakan).

```
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]
```

Contoh

Definisi Klaim Volume Persisten untuk PV disebut`fsx-pv`:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: fsx-claim
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 1200Gi
  volumeName: fsx-pv
```

Contoh

Konfigurasikan pod untuk menggunakan Klaim Volume Persisten dari`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
```

Untuk contoh selengkapnya, lihat FSx untuk Contoh Pengemudi Lustre di. GitHub Pantau Amazon FSx untuk metrik kinerja Lustre menggunakan Amazon. CloudWatch Saat kemacetan kinerja diidentifikasi, sesuaikan parameter konfigurasi Anda sesuai kebutuhan.

Skenario: Beban kerja instance GPU tunggal

Mountpoint untuk Amazon S3 dengan Driver CSI: Anda dapat memasang bucket S3 sebagai volume di pod Anda menggunakan driver Mountpoint untuk Amazon S3 CSI. Metode ini memungkinkan kontrol akses berbutir halus di mana Pod dapat mengakses bucket S3 tertentu. Setiap pod memiliki instans mountpoint dan cache lokal (5-10GB) sendiri, mengisolasi kinerja pemuatan dan pembacaan model antar pod. Pengaturan ini mendukung otentikasi tingkat pod dengan IAM Roles for Service Accounts (IRSA) dan versi model independen untuk model atau pelanggan yang berbeda. Trade-off adalah peningkatan penggunaan memori dan lalu lintas API, karena setiap pod mengeluarkan panggilan API S3 dan mempertahankan cache-nya sendiri.

Contoh sebagian contoh penerapan Pod YAMB dengan 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
```

Pertimbangan kinerja:

- Caching data: Mountpoint untuk S3 dapat menyimpan konten cache untuk mengurangi biaya dan meningkatkan kinerja untuk pembacaan berulang ke file yang sama. Lihat konfigurasi Caching untuk opsi dan parameter caching.

- Ukuran bagian objek: Saat menyimpan dan mengakses file berukuran lebih dari 72GB, lihat Mengonfigurasi kinerja Mountpoint untuk memahami cara mengonfigurasi parameter `--write-part-size` baris perintah `--read-part-size` dan profil data untuk memenuhi persyaratan profil data dan beban kerja Anda.
- Shared-cache dirancang untuk objek berukuran hingga 1MB. Itu tidak mendukung benda besar. Gunakan opsi cache lokal untuk caching objek dalam NVMe atau volume EBS pada node EKS.
- Biaya permintaan API: Saat melakukan sejumlah besar operasi file dengan Mountpoint untuk S3, biaya permintaan API dapat menjadi bagian dari biaya penyimpanan. Untuk mengurangi ini, jika konsistensi yang kuat tidak diperlukan, selalu aktifkan caching metadata dan atur `metadata-ttl` periode untuk mengurangi jumlah operasi API ke S3.

Untuk detail selengkapnya, lihat Mountpoint untuk Driver CSI Amazon S3 di dokumentasi resmi Amazon EKS. Kami menyarankan untuk memantau metrik kinerja Amazon S3 dengan metrik CloudWatch Amazon jika terjadi kemacetan dan menyesuaikan konfigurasi Anda jika diperlukan.

Amazon FSx untuk penyimpanan bersama persisten OpenZFS

Untuk skenario yang melibatkan beberapa instans komputasi GPU EC2 dengan beban kerja yang sensitif terhadap latensi yang membutuhkan ketersediaan tinggi, kinerja tinggi, sensitivitas biaya, dan beberapa penerapan pod untuk aplikasi yang berbeda, kami merekomendasikan Amazon untuk OpenZFS. FSx Beberapa contoh beban kerja termasuk inferensi real-time, pembelajaran penguatan, dan pelatihan jaringan permusuhan generatif. FSx untuk OpenZFS sangat bermanfaat untuk beban kerja yang membutuhkan akses kinerja tinggi ke struktur direktori terfokus dengan file kecil menggunakan pola akses data IO kecil. Selain itu, FSx untuk OpenZFS memberikan fleksibilitas untuk menskalakan kinerja secara independen dari kapasitas penyimpanan, membantu Anda mencapai efisiensi biaya yang optimal dengan mencocokkan ukuran penyimpanan dengan kebutuhan aktual sambil mempertahankan tingkat kinerja yang diperlukan

Asli FSx untuk driver OpenZFS CSI memungkinkan pembuatan beberapa PVCs ke satu sistem file dengan membuat beberapa volume. Ini mengurangi overhead manajemen dan memaksimalkan pemanfaatan throughput sistem file dan IOPS melalui penerapan pod aplikasi terkonsolidasi pada satu sistem file. Selain itu, ini mencakup fitur perusahaan seperti snapshot zero-copy, klon zero-copy, dan kuota pengguna dan grup yang dapat disediakan secara dinamis melalui driver CSI.

FSx untuk OpenZFS mendukung tiga jenis penerapan yang berbeda saat pembuatan:

- Single-AZ: Opsi biaya terendah dengan latensi sub-milidetik, tetapi tidak memberikan ketersediaan tinggi di sistem file atau tingkat Availability Zone. Direkomendasikan untuk pengembangan dan pengujian beban kerja atau yang memiliki ketersediaan tinggi di lapisan aplikasi.
- Single-AZ (HA): Menyediakan ketersediaan tinggi pada tingkat sistem file dengan latensi sub-milidetik. Direkomendasikan untuk beban kerja kinerja tertinggi yang membutuhkan ketersediaan tinggi.
- Multi-AZ: Menyediakan ketersediaan tinggi di tingkat sistem file serta di seluruh Availability Zones. Direkomendasikan untuk beban kerja berkinerja tinggi yang memerlukan ketersediaan tambahan di seluruh Availability Zone.

Pertimbangan kinerja:

- Jenis penerapan: Jika ketersediaan tambahan di seluruh Availability Zones bukan persyaratan, pertimbangkan untuk menggunakan tipe penerapan Single-AZ (HA). Jenis penerapan ini menyediakan hingga 100% throughput untuk penulisan, mempertahankan latensi sub-milidetik, dan sistem file Gen2 memiliki NVMe cache tambahan untuk menyimpan hingga terrabyte data yang sering diakses. Sistem file multi-AZ menyediakan hingga 75% dari throughput untuk penulisan pada latensi yang meningkat untuk mengakomodasi lalu lintas lintas AZ.
- Throughput dan IOPS: Baik throughput dan IOPS yang dikonfigurasi untuk sistem file dapat ditingkatkan atau diturunkan pasca penerapan. Anda dapat menyediakan hingga 10 akses data GB/s of disk throughput providing up to 21GB/s yang di-cache. IOPS dapat ditingkatkan hingga 400.000 dari disk dan cache dapat menyediakan lebih dari 1 juta IOPS. Perhatikan bahwa penskalaan throughput dari sistem file Single-AZ memang menyebabkan pemadaman singkat sistem file karena tidak ada ketersediaan tinggi. Penskalaan throughput dari sistem file Single-AZ (HA) atau Multi-AZ dapat dilakukan tanpa gangguan. SSD IOPS dapat diskalakan setiap enam jam sekali.
- Storage Class: FSx untuk OpenZFS mendukung kelas penyimpanan SSD serta kelas penyimpanan Intelligent-Tiering. Untuk AI/ML beban kerja, disarankan untuk menggunakan kelas penyimpanan SSD yang memberikan kinerja yang konsisten terhadap beban kerja menjaga CPU/GPU sesibuk mungkin.
- Kompresi: Aktifkan algoritma LZ4 kompresi jika Anda memiliki beban kerja yang dapat dikompresi. Ini mengurangi jumlah data yang dikonsumsi setiap file dalam cache yang memungkinkan lebih banyak data dilayani langsung dari cache sebagai throughput jaringan dan IOPS mengurangi beban pada disk SSD.

- Ukuran rekaman: Sebagian besar AI/ML beban kerja akan mendapat manfaat dari meninggalkan ukuran catatan 128KiB default. Nilai ini hanya boleh dikurangi jika dataset terdiri dari file besar (di atas 10GiB) dengan akses blok kecil yang konsisten di bawah 128KiB dari aplikasi.

Setelah sistem file dibuat, volume root terkait secara otomatis dibuat oleh layanan. Ini adalah praktik terbaik untuk menyimpan data dalam volume anak dari volume root pada sistem file. Menggunakan driver CSI FSx for OpenZFS, Anda membuat Klaim Volume Persisten terkait untuk membuat volume anak secara dinamis.

Contoh:

Definisi Storage Class (SC) untuk volume OpenZFS, digunakan untuk membuat volume anak dari volume root ($ROOT_VOL_ID) pada sistem file yang ada dan mengekspor volume ke VPC CIDR ($VPC_CIDR) menggunakan protokol NFS v4.2. FSx

```
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
```

Klaim Volume Persisten (PVC) yang dibuat secara dinamis terhadap yang fsxz-vol-sc dibuat di atas. Catatan, kapasitas penyimpanan yang dialokasikan adalah 1Gi, ini diperlukan FSx untuk volume OpenZFS seperti yang tercantum dalam FAQ driver CSI. Volume akan diberikan kapasitas penuh yang disediakan untuk sistem file dengan konfigurasi ini. Jika kapasitas volume perlu dibatasi, Anda dapat melakukannya menggunakan kuota pengguna atau grup.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: dynamic-vol-pvc
  namespace: example
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: fsxz-vol-sc
  resources:
    requests:
      storage: 1Gi
```

Konfigurasikan pod untuk memasang volume menggunakan Persistent Volume Claim (PVC) dari 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 untuk cache model bersama

Dalam skenario di mana Anda memiliki beberapa lingkungan instans komputasi GPU EC2 dan memiliki beban kerja dinamis yang memerlukan akses model bersama di beberapa node dan Availability Zone (misalnya, inferensi online real-time dengan Karpenter) dengan kinerja moderat dan kebutuhan skalabilitas, sebaiknya gunakan sistem file Amazon Elastic File System (EFS) sebagai Volume Persisten melalui Driver EFS CSI. Amazon EFS adalah sistem file NFS berbasis cloud yang dikelola sepenuhnya, sangat tersedia, dan dapat diskalakan yang memungkinkan instans dan wadah EC2 dengan penyimpanan file bersama, dengan kinerja yang konsisten, dan di mana tidak diperlukan penyediaan penyimpanan di muka. Gunakan EFS sebagai volume model, dan pasang volume sebagai sistem file bersama dengan mendefinisikan Volume Persisten pada kluster EKS. Setiap Klaim Volume Persisten (PVC) yang didukung oleh sistem file EFS dibuat sebagai titik akses EFS ke sistem file EFS. EFS memungkinkan beberapa node dan pod untuk mengakses file model yang sama, menghilangkan kebutuhan untuk menyinkronkan data ke setiap sistem file node. Instal driver EFS CSI untuk mengintegrasikan EFS dengan EKS.

Anda dapat menerapkan sistem file Amazon EFS dengan mode throughput berikut:

- Meledak Throughput: Timbangan throughput dengan ukuran sistem file, cocok untuk berbagai beban kerja dengan semburan sesekali.
- Throughput yang Disediakan: Throughput khusus, ideal untuk pekerjaan pelatihan ML yang konsisten dengan kebutuhan kinerja yang dapat diprediksi dalam batasnya.
- Throughput Elastis (direkomendasikan untuk ML): Secara otomatis menskalakan berdasarkan beban kerja, efektivitas biaya untuk berbagai beban kerja ML.

Untuk melihat spesifikasi performa, lihat Spesifikasi performa Amazon EFS.

Pertimbangan kinerja:

- Gunakan Elastic Throughput untuk berbagai beban kerja.
- Gunakan kelas penyimpanan Standar untuk beban kerja MS aktif.

Untuk contoh lengkap penggunaan sistem file Amazon EFS sebagai Volume persisten dalam cluster EKS dan Pod Anda, lihat Contoh Driver EFS CSI di GitHub. Pantau metrik kinerja Amazon EFS menggunakan Amazon CloudWatch. Saat kemacetan kinerja diidentifikasi, sesuaikan parameter konfigurasi Anda sesuai kebutuhan.

Gunakan S3 Express One Zone untuk Alur Kerja Berorientasi Objek yang Sensitif Latensi

Untuk AI/ML beban kerja yang sensitif terhadap latensi di Amazon EKS, seperti pelatihan model skala besar, inferensi, atau analitik performa tinggi, sebaiknya gunakan S3 Express One Zone untuk penyimpanan dan pengambilan model berkinerja tinggi. S3 Express One Zone menawarkan namespace hierarkis, seperti sistem file, tempat Anda cukup mengunggah ke bucket direktori, cocok untuk “membuang semuanya”, sambil mempertahankan kecepatan tinggi. Ini sangat berguna jika Anda terbiasa dengan alur kerja berorientasi objek. Atau, jika Anda lebih terbiasa dengan sistem file (misalnya, Posix-compliant), Anda dapat memilih Amazon FSx untuk Lustre atau OpenZFS. Amazon S3 Express One Zone menyimpan data dalam Availability Zone (AZ) tunggal menggunakan bucket direktori dan menawarkan latensi yang lebih rendah daripada bucket S3 standar, yang mendistribusikan data ke beberapa. AZs Untuk hasil terbaik, pastikan untuk menempatkan komputasi EKS Anda di AZ yang sama dengan bucket Express One Zone Anda. Untuk mempelajari selengkapnya tentang perbedaan S3 Express One Zone, lihat Perbedaan untuk bucket direktori.

Untuk mengakses S3 Express One Zone dengan semantik sistem file, sebaiknya gunakan Driver Mountpoint S3 CSI, yang memasang bucket S3 (termasuk Express One Zone) sebagai sistem file lokal. Ini menerjemahkan operasi file (misalnya, buka, baca, tulis) ke dalam panggilan API S3, menyediakan akses throughput tinggi yang dioptimalkan untuk beban kerja baca-berat dari beberapa klien dan penulisan berurutan ke objek baru. Untuk detail tentang operasi dan batasan yang didukung (misalnya, tidak ada kepatuhan POSIX penuh, tetapi menambahkan dan mengganti nama yang didukung di Express One Zone), lihat dokumentasi semantik Mountpoint.

Manfaat kinerja

- Menyediakan akses data hingga 10x lebih cepat daripada Standar S3, dengan latensi milidetik satu digit yang konsisten dan biaya permintaan hingga 80% lebih rendah.
- Timbangan untuk menangani ratusan ribu hingga jutaan permintaan per detik per bucket direktori, menghindari throttling atau brownout yang terlihat di S3 standar selama beban ekstrim (misalnya, dari cluster dengan puluhan hingga ratusan ribu jaringan jenuh). GPUs/CPUs
- Menggunakan mekanisme otentikasi berbasis sesi. Otentikasi sekali untuk mendapatkan token sesi, lalu lakukan operasi berulang dengan kecepatan tinggi tanpa overhead auth per permintaan. Ini dioptimalkan untuk beban kerja seperti checkpointing yang sering atau pemuatan data.

Kasus penggunaan yang direkomendasikan

- Caching: Salah satu kasus penggunaan teratas menggunakan Driver Mountpoint S3 CSI dengan S3 Express One Zone adalah caching. Contoh pertama membaca data dari Standar S3 (tujuan umum), menyimpannya di Zona Satu Ekspres latensi rendah. Pembacaan selanjutnya oleh klien lain mengakses data yang di-cache lebih cepat, yang ideal untuk skenario multi-node di mana beberapa node EKS membaca data yang sama (misalnya, kumpulan data pelatihan bersama). Ini dapat meningkatkan kinerja hingga 7x untuk akses berulang dan mengurangi biaya komputasi. Untuk beban kerja yang memerlukan kepatuhan POSIX penuh (misalnya, penguncian file dan modifikasi di tempat), pertimbangkan Amazon FSx untuk Lustre atau OpenZFS sebagai alternatif.
- AI/ML Pelatihan dan inferensi skala besar: Ideal untuk beban kerja dengan ratusan atau ribuan node komputasi (misalnya, GPUs dalam kluster EKS) di mana pelambatan S3 tujuan umum dapat menyebabkan penundaan, membuang-buang sumber daya komputasi yang mahal. Misalnya, peneliti LLM atau organisasi yang menjalankan model harian tests/checkpoints mendapat manfaat dari akses yang cepat dan andal tanpa merusak S3 regional. Untuk beban kerja skala yang lebih kecil (misalnya, 10 detik node), Standar S3 atau kelas penyimpanan lainnya mungkin cukup.
- Pipa data: Load/prepare model, data pelatihan arsip, atau pos pemeriksaan aliran. Jika tim Anda lebih suka penyimpanan objek daripada sistem file tradisional (misalnya, karena keakraban dengan S3), gunakan ini alih-alih rekayasa perubahan untuk opsi yang sesuai dengan POSIX seperti untuk Lustre. FSx

Pertimbangan-pertimbangan

- Ketahanan: Desain single-AZ memberikan daya tahan 99,999999999% (sama dengan standar S3, melalui redundansi dalam AZ) tetapi ketersediaan lebih rendah (dirancang 99,95%, SLA 99,9%) dibandingkan dengan kelas Multi-AZ (ketersediaan 99,99%). Ini kurang tahan terhadap kegagalan AZ. Gunakan untuk data recreatable atau cache. Pertimbangkan replikasi atau pencadangan multi-AZ untuk beban kerja penting.
- API dan Dukungan Fitur: Mendukung subset S3 APIs (misalnya, tidak ada kebijakan siklus hidup atau replikasi); mungkin memerlukan perubahan aplikasi kecil untuk otentikasi sesi atau penanganan objek.
- Integrasi EKS: Temukan bersama EKS Anda pods/nodes di AZ yang sama dengan bucket direktori untuk meminimalkan latensi jaringan. Gunakan driver Mountpoint untuk Amazon S3 atau CSI untuk akses asli Kubernetes.
- Pengujian: Uji latensi pengambilan di lingkungan non-produksi untuk memvalidasi peningkatan kinerja. Pantau pelambatan dalam skenario S3 standar (misalnya, saturasi GPU tinggi) dan bandingkan.

Kelas penyimpanan S3 Express One Zone tersedia di beberapa wilayah dan terintegrasi dengan EKS untuk beban kerja yang membutuhkan akses objek tanpa menunggu penyimpanan. Untuk mempelajari lebih lanjut, lihat Memulai dengan S3 Express One Zone.

Observabilitas

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Pemantauan dan Observabilitas

Metrik GPU Dijelaskan

Metrik Pemanfaatan GPU menunjukkan apakah GPU menjalankan pekerjaan apa pun selama jendela sampel. Metrik ini menangkap persentase waktu GPU mengeksekusi setidaknya satu instruksi, tetapi tidak mengungkapkan seberapa efisien GPU menggunakan perangkat kerasnya. GPU berisi beberapa Streaming Multiprocessors (SMs), yang merupakan unit pemrosesan paralel yang menjalankan instruksi. Pembacaan pemanfaatan 100% dapat berarti GPU menjalankan beban kerja paralel yang berat di semua miliknya SMs, atau itu bisa berarti satu instruksi kecil mengaktifkan GPU selama periode sampel. Untuk memahami pemanfaatan aktual, Anda perlu memeriksa metrik GPU di berbagai tingkatan arsitektur perangkat keras. Setiap Multiprosesor Streaming dibangun dari tipe inti yang berbeda, dan setiap lapisan memperlihatkan karakteristik kinerja yang berbeda. Metrik tingkat atas (Pemanfaatan GPU, Pemanfaatan Memori, Daya GPU, dan Suhu GPU, terlihat melalui nvidia-smi) menunjukkan apakah perangkat aktif. Metrik yang lebih dalam (pemanfaatan SM, Aktivitas SM, dan penggunaan inti tensor) mengungkapkan seberapa efisien GPU menggunakan sumber dayanya.

Targetkan penggunaan daya GPU yang tinggi

Kapasitas komputasi GPUs limbah yang kurang dimanfaatkan dan meningkatkan biaya karena beban kerja gagal melibatkan semua komponen GPU secara bersamaan. Untuk AI/ML beban kerja di Amazon EKS, lacak penggunaan daya GPU sebagai proxy untuk mengidentifikasi aktivitas GPU yang sebenarnya. Pemanfaatan GPU melaporkan persentase waktu GPU mengeksekusi kernel apa pun, tetapi tidak mengungkapkan apakah Multiprosesor Streaming, pengontrol memori, dan inti tensor semuanya aktif pada saat yang bersamaan. Penggunaan daya memperlihatkan celah ini karena

perangkat keras yang terlibat penuh menarik daya secara signifikan lebih banyak daripada perangkat keras yang menjalankan kernel ringan atau duduk diam di antara tugas. Bandingkan daya tarik dengan daya desain termal (TDP) GPU untuk melihat kurangnya pemanfaatan, kemudian selidiki apakah beban kerja Anda terhambat oleh preprocessing CPU, I/O jaringan, atau ukuran batch yang tidak efisien.

Siapkan CloudWatch Container Insights di Amazon EKS untuk mengidentifikasi pod, node, atau beban kerja dengan konsumsi daya GPU rendah. Alat ini terintegrasi langsung dengan Amazon EKS dan memungkinkan Anda memantau konsumsi daya GPU dan menyesuaikan penjadwalan pod atau jenis instance saat penggunaan daya turun di bawah level target Anda. Jika Anda membutuhkan visualisasi lanjutan atau dasbor khusus, gunakan Eksportir DCGM NVIDIA dengan Prometheus dan Grafana untuk pemantauan asli Kubernetes. Keduanya mendekati metrik NVIDIA kunci permukaan seperti `nvidia_smi_power_draw` (konsumsi daya GPU) dan `nvidia_smi_temperature_gpu` (suhu GPU). Untuk daftar metrik, jelajahi https://docs.aws.amazon.com/AmazonCloudWatch/ latest/ monitoring/CloudWatch -Agent-NVIDIA-GPU.htm. Carilah pola seperti penggunaan daya rendah secara konsisten selama jam-jam tertentu atau untuk pekerjaan tertentu. Tren ini membantu Anda mengidentifikasi tempat untuk mengkonsolidasikan beban kerja atau menyesuaikan alokasi sumber daya.

Batas sumber daya statis di Kubernetes (seperti jumlah CPU, memori, dan GPU) sering menyebabkan penyediaan berlebih atau kurang dimanfaatkan, terutama untuk beban kerja dinamis seperti inferensi di mana permintaan berfluktuasi. AI/ML Analisis tren pemanfaatan Anda dan konsolidasikan beban kerja menjadi lebih sedikit. GPUs Pastikan setiap GPU mencapai pemanfaatan penuh sebelum Anda mengalokasikan yang tambahan. Pendekatan ini mengurangi limbah dan menurunkan biaya. Untuk panduan terperinci tentang mengoptimalkan strategi penjadwalan dan berbagi, lihat praktik terbaik EKS Compute and Autoscaling

Observabilitas dan Metrik

Menggunakan Alat Pemantauan dan Pengamatan untuk Beban Kerja Anda AI/ML

AI/ML Layanan modern membutuhkan koordinasi lintas infrastruktur, pemodelan, dan logika aplikasi. Insinyur platform mengelola infrastruktur dan tumpukan observabilitas. Mereka mengumpulkan, menyimpan, dan memvisualisasikan metrik. AI/ML insinyur menentukan metrik spesifik model dan memantau kinerja di bawah berbagai beban dan distribusi data. Pengembang aplikasi mengkonsumsi APIs, merutekan permintaan, dan melacak metrik tingkat layanan dan interaksi pengguna. Tanpa praktik observabilitas terpadu, tim-tim ini bekerja dalam silo dan kehilangan sinyal kritis tentang kesehatan dan kinerja sistem. Membangun visibilitas bersama di seluruh lingkungan memastikan

semua pemangku kepentingan dapat mendeteksi masalah sejak dini dan mempertahankan layanan yang andal.

Mengoptimalkan kluster Amazon EKS untuk AI/ML beban kerja menghadirkan tantangan pemantauan yang unik, terutama seputar manajemen memori GPU. Tanpa pemantauan yang tepat, organisasi menghadapi kesalahan out-of-memory (OOM), inefisiensi sumber daya, dan biaya yang tidak perlu. Pemantauan yang efektif memastikan kinerja yang lebih baik, ketahanan, dan biaya yang lebih rendah untuk pelanggan EKS. Gunakan pendekatan holistik yang menggabungkan tiga lapisan pemantauan. Pertama, pantau metrik GPU granular menggunakan NVIDIA DCGM Exporter untuk melacak penggunaan daya GPU, suhu GPU, aktivitas SM, hunian SM, dan kesalahan XID. Kedua, pantau kerangka kerja penyajian inferensi seperti Ray dan VllM untuk mendapatkan wawasan beban kerja terdistribusi melalui metrik asli mereka. Ketiga, kumpulkan wawasan tingkat aplikasi untuk melacak metrik kustom khusus untuk beban kerja Anda. Pendekatan berlapis ini memberi Anda visibilitas dari pemanfaatan perangkat keras melalui kinerja aplikasi.

Alat dan kerangka kerja

Beberapa alat dan kerangka kerja menyediakan out-of-the-box metrik asli untuk memantau AI/ML beban kerja. Metrik bawaan ini menghilangkan kebutuhan akan instrumentasi khusus dan mengurangi waktu penyiapan. Metrik berfokus pada aspek kinerja seperti latensi, throughput, dan pembuatan token, yang sangat penting untuk penyajian inferensi dan pembandingan. Menggunakan metrik asli memungkinkan Anda untuk segera memulai pemantauan tanpa membuat pipeline koleksi khusus.

- VllM: Mesin servis throughput tinggi untuk model bahasa besar (LLMs) yang menyediakan metrik asli seperti latensi permintaan dan penggunaan memori.
- Ray: Kerangka kerja komputasi terdistribusi yang memancarkan metrik untuk beban kerja AI yang dapat diskalakan, termasuk waktu eksekusi tugas dan pemanfaatan sumber daya.
- Hugging Face Text Generation Inference (TGI): Toolkit untuk menerapkan dan LLMs melayani, dengan metrik bawaan untuk kinerja inferensi.
- NVIDIA genai-perf: Alat baris perintah untuk membandingkan model AI generatif, mengukur throughput, latensi, dan metrik khusus LLM, seperti permintaan yang diselesaikan dalam interval waktu tertentu.

Metode observabilitas

Kami merekomendasikan untuk menerapkan mekanisme observabilitas tambahan dengan salah satu cara berikut.

CloudWatch Wawasan Kontainer Jika organisasi Anda lebih menyukai alat asli AWS dengan pengaturan minimal, kami merekomendasikan CloudWatch Wawasan Kontainer. Ini terintegrasi dengan Eksportir NVIDIA DCGM untuk mengumpulkan metrik GPU dan menawarkan dasbor pra-bangun untuk wawasan cepat. Diaktifkan dengan menginstal add-on CloudWatch Observability di cluster Anda, Container Insights menyebarkan dan mengelola siklus hidup Eksportir NVIDIA DCGM yang mengumpulkan metrik GPU dari driver Nvidia dan memaparkannya. CloudWatch

Setelah Anda menginstal Container Insights, CloudWatch secara otomatis mendeteksi NVIDIA GPUs di lingkungan Anda dan mengumpulkan metrik kesehatan dan kinerja yang penting. Metrik ini muncul di dasbor yang out-of-the-box dikuratori. Anda juga dapat mengintegrasikan Ray dan Vllm dengan CloudWatch menggunakan CloudWatch Agen Terpadu untuk mengirim metrik asli mereka. Pendekatan terpadu ini menyederhanakan observabilitas di lingkungan EKS dan memungkinkan tim fokus pada penyetelan kinerja dan pengoptimalan biaya alih-alih membangun infrastruktur pemantauan.

Untuk daftar lengkap metrik yang tersedia, lihat metrik Amazon EKS dan Kubernetes Container Insights. Untuk step-by-step panduan penerapan pemantauan GPU, lihat Dapatkan wawasan operasional untuk beban kerja GPU NVIDIA menggunakan Amazon CloudWatch Container Insights. Untuk contoh praktis mengoptimalkan latensi inferensi, lihat Mengoptimalkan respons AI: Panduan praktis untuk inferensi yang dioptimalkan dengan latensi Amazon Bedrock.

Prometheus dan Grafana Terkelola Jika organisasi Anda membutuhkan dasbor yang disesuaikan dan kemampuan visualisasi tingkat lanjut, terapkan Prometheus dengan Eksportir DCGM NVIDIA dan Grafana untuk pemantauan asli Kubernetes. Prometheus menggores dan menyimpan metrik GPU dari eksportir DCGM, sementara Grafana memberikan visualisasi yang fleksibel dan kemampuan peringatan. Pendekatan ini memberi Anda kontrol lebih besar atas desain dasbor dan retensi metrik dibandingkan dengan CloudWatch Wawasan Kontainer.

Anda dapat memperluas tumpukan pemantauan ini dengan mengintegrasikan kerangka kerja sumber terbuka seperti Ray dan Vllm Ray dan VllM untuk mengekspor metrik asli mereka ke Prometheus. Anda juga dapat menghubungkan Grafana ke sumber data AWS X-Ray untuk memvisualisasikan jejak terdistribusi dan mengidentifikasi kemacetan kinerja di seluruh pipeline inferensi Anda. Kombinasi ini memberikan end-to-end visibilitas dari metrik tingkat GPU melalui alur permintaan tingkat aplikasi.

Untuk step-by-step panduan tentang penerapan tumpukan pemantauan ini, lihat Memantau beban kerja GPU di Amazon EKS menggunakan layanan sumber terbuka yang dikelola AWS.

Pertimbangkan untuk Memantau Pelatihan Inti & Metrik Penyetelan Sempurna

Pantau metrik pelatihan inti untuk melacak kesehatan dan kinerja klaster Amazon EKS Anda dan beban kerja pembelajaran mesin yang berjalan di dalamnya. Beban kerja pelatihan memiliki persyaratan pemantauan yang berbeda dari beban kerja inferensi karena mereka berjalan untuk waktu yang lama, mengkonsumsi sumber daya secara berbeda, dan memerlukan visibilitas ke dalam konvergensi model dan efisiensi pipa data. Metrik di bawah ini membantu Anda mengidentifikasi kemacetan, mengoptimalkan alokasi sumber daya, dan memastikan pekerjaan pelatihan berhasil diselesaikan. Untuk step-by-step panduan dalam menerapkan pendekatan pemantauan ini, lihat Pengantar untuk mengamati beban kerja pembelajaran mesin di Amazon EKS.

Metrik Penggunaan Sumber Daya

Pantau metrik penggunaan sumber daya untuk memvalidasi bahwa sumber daya Anda dikonsumsi dengan benar. Metrik ini membantu Anda mengidentifikasi kemacetan dan akar penyebab masalah kinerja.

- CPU, Memori, Jaringan, Daya GPU, dan Suhu GPU - Pantau metrik ini untuk memastikan sumber daya yang dialokasikan memenuhi tuntutan beban kerja dan mengidentifikasi peluang pengoptimalan. Lacak metrik seperti gpu_memory_usage_bytes untuk mengidentifikasi pola konsumsi memori dan mendeteksi penggunaan puncak. Hitung persentil seperti persentil ke-95 (P95) untuk memahami tuntutan memori tertinggi selama pelatihan. Analisis ini membantu Anda mengoptimalkan model dan infrastruktur untuk menghindari kesalahan OOM dan mengurangi biaya.
- SM Occupancy, SM FPxx Activity, Activity - Pantau metrik ini untuk memahami bagaimana sumber daya dasar pada GPU digunakan. Target 0.8 untuk Aktivitas SM sebagai aturan praktis.
- Pemanfaatan Sumber Daya Node dan Pod - Melacak penggunaan sumber daya di tingkat node dan pod untuk mengidentifikasi pertentangan sumber daya dan potensi kemacetan. Pantau apakah node mendekati batas kapasitas, yang dapat menunda penjadwalan pod dan memperlambat pekerjaan pelatihan.
- Pemanfaatan Sumber Daya Dibandingkan dengan Permintaan dan Batas — Bandingkan penggunaan sumber daya aktual dengan permintaan dan batasan yang dikonfigurasi untuk menentukan apakah klaster Anda dapat menangani beban kerja saat ini dan mengakomodasi yang akan datang. Perbandingan ini mengungkapkan apakah Anda perlu menyesuaikan alokasi sumber daya untuk menghindari kesalahan OOM atau pemborosan sumber daya.
- Metrik Internal dari Kerangka Kerja ML - Menangkap pelatihan internal dan metrik konvergensi dari kerangka kerja ML seperti dan. TensorFlow PyTorch Metrik ini mencakup kurva kerugian,

tingkat pembelajaran, waktu pemrosesan batch, dan durasi langkah pelatihan. Visualisasikan metrik ini menggunakan TensorBoard atau alat serupa untuk melacak konvergensi model dan mengidentifikasi inefisiensi pelatihan.

Metrik Kinerja Model

Pantau metrik kinerja model untuk memvalidasi bahwa proses pelatihan Anda menghasilkan model yang memenuhi akurasi dan persyaratan bisnis. Metrik ini membantu Anda menentukan kapan harus menghentikan pelatihan, membandingkan versi model, dan mengidentifikasi penurunan kinerja.

- Akurasi, Presisi, Ingat, dan skor F1 — Lacak metrik ini untuk memahami seberapa baik kinerja model Anda pada data validasi. Hitung skor F1 pada set validasi setelah setiap zaman pelatihan untuk menilai apakah model membaik dan kapan mencapai tingkat kinerja yang dapat diterima.
- Metrik Khusus Bisnis dan KPIs — Tentukan dan lacak metrik yang secara langsung mengukur nilai bisnis dari inisiatif Anda. AI/ML Untuk sistem rekomendasi, lacak metrik seperti rasio klik-tayang atau rasio konversi untuk memastikan model mendorong hasil bisnis yang diinginkan.
- Kinerja dari waktu ke waktu — Bandingkan metrik kinerja di seluruh versi model dan proses pelatihan untuk mengidentifikasi tren dan mendeteksi degradasi. Lacak apakah versi model yang lebih baru mempertahankan atau meningkatkan kinerja dibandingkan dengan model dasar. Perbandingan historis ini membantu Anda memutuskan apakah akan menerapkan model baru atau menyelidiki masalah pelatihan.

Kualitas Data dan Metrik Drift

Pantau kualitas data dan metrik drift untuk memastikan data pelatihan Anda tetap konsisten dan representatif. Penyimpangan data dapat menyebabkan kinerja model menurun seiring waktu, sementara masalah kualitas data dapat mencegah model konvergen atau menghasilkan hasil yang tidak dapat diandalkan.

- Sifat Statistik Data Input — Melacak sifat statistik seperti mean, standar deviasi, dan distribusi fitur input dari waktu ke waktu untuk mendeteksi penyimpangan data atau anomali. Pantau apakah distribusi fitur bergeser secara signifikan dari data pelatihan dasar Anda. Misalnya, jika rata-rata fitur kritis berubah lebih dari dua standar deviasi, selidiki apakah pipeline data Anda telah berubah atau apakah sumber data yang mendasarinya telah bergeser.
- Deteksi dan Peringatan Drift Data — Menerapkan mekanisme otomatis untuk mendeteksi dan memperingatkan masalah kualitas data sebelum memengaruhi pelatihan. Gunakan uji statistik seperti uji Kolmogorov-Smirnov atau uji chi-kuadrat untuk membandingkan distribusi data saat ini

dengan data pelatihan asli Anda. Siapkan peringatan saat pengujian mendeteksi penyimpangan signifikan sehingga Anda dapat melatih ulang model dengan data yang diperbarui atau menyelidiki masalah saluran data.

Metrik Latensi dan Throughput

Pantau metrik latensi dan throughput untuk mengidentifikasi hambatan dalam jalur pelatihan Anda dan mengoptimalkan pemanfaatan sumber daya. Metrik ini membantu Anda memahami di mana waktu dihabiskan selama pelatihan dan di mana harus memfokuskan upaya pengoptimalan.

- End-to-End Latensi Pipa Pelatihan ML — Ukur total waktu data mengalir melalui seluruh pipeline pelatihan Anda, mulai dari konsumsi data hingga pembaruan model. Lacak metrik ini di seluruh proses pelatihan untuk mengidentifikasi apakah perubahan pipeline meningkatkan atau menurunkan kinerja. Latensi tinggi sering menunjukkan kemacetan dalam pemuatan data, preprocessing, atau komunikasi jaringan antar node.
- Throughput Pelatihan dan Tingkat Pemrosesan — Lacak volume data proses pipa pelatihan Anda per unit waktu untuk memastikan pemanfaatan sumber daya yang efisien. Pantau metrik seperti sampel yang diproses per detik atau batch yang diselesaikan per menit. Throughput yang rendah relatif terhadap kapasitas perangkat keras Anda menunjukkan inefisiensi dalam pemuatan data, pra-pemrosesan, atau perhitungan model yang membuang siklus GPU.
- Checkpoint Save and Restore Latency — Pantau waktu yang diperlukan untuk menyimpan pos pemeriksaan model ke penyimpanan (S3, EFS FSx,) dan mengembalikannya ke memori GPU atau CPU saat melanjutkan pekerjaan atau pulih dari kegagalan. Operasi pos pemeriksaan yang lambat memperpanjang waktu pemulihan pekerjaan dan meningkatkan biaya. Lacak ukuran pos pemeriksaan, durasi simpan, durasi pemulihan, dan jumlah kegagalan untuk mengidentifikasi peluang pengoptimalan seperti kompresi atau tingkatan penyimpanan yang lebih cepat.
- Waktu Pemuatan dan Pemrosesan Data - Ukur waktu yang dihabiskan untuk memuat data dari penyimpanan dan menerapkan transformasi pra-pemrosesan. Bandingkan waktu ini dengan waktu komputasi model untuk menentukan apakah pelatihan Anda terikat data atau terikat komputasi. Jika pemuatan data menghabiskan lebih dari 20% dari total waktu pelatihan, pertimbangkan untuk mengoptimalkan pipeline data Anda dengan caching, prefetching, atau penyimpanan yang lebih cepat.

Tingkat Kesalahan dan Kegagalan

Pantau tingkat kesalahan dan kegagalan di seluruh jalur pelatihan Anda untuk menjaga keandalan dan mencegah sumber daya komputasi yang terbuang. Kesalahan yang tidak terdeteksi dapat menyebabkan pekerjaan pelatihan gagal secara diam-diam, menghasilkan model yang tidak valid, atau membuang waktu berjam-jam GPU sebelum Anda melihat masalah.

- Pemantauan Kesalahan Pipeline — Lacak kesalahan di semua tahapan pipeline ML Anda, termasuk pra-pemrosesan data, pelatihan model, dan operasi pos pemeriksaan. Log jenis kesalahan, frekuensi, dan komponen yang terpengaruh untuk mengidentifikasi masalah dengan cepat. Kesalahan umum termasuk ketidakcocokan format data, out-of-memory kegagalan selama pra-pemrosesan, dan kegagalan penyimpanan pos pemeriksaan karena batas penyimpanan. Siapkan peringatan ketika tingkat kesalahan melebihi ambang batas dasar sehingga Anda dapat menyelidiki sebelum kesalahan mengalir.
- Analisis Kesalahan Berulang — Mengidentifikasi dan menyelidiki pola dalam kesalahan berulang untuk mencegah kegagalan di masa depan dan meningkatkan keandalan saluran pipa. Analisis log untuk menemukan apakah sampel data tertentu, ukuran batch, atau konfigurasi pelatihan secara konsisten menyebabkan kegagalan. Misalnya, jika tipe data input tertentu memicu kesalahan pra-pemrosesan, tambahkan pemeriksaan validasi sebelumnya di pipeline atau perbarui logika pembersihan data Anda. Lacak waktu rata-rata antara kegagalan (MTBF) untuk mengukur apakah keandalan pipa Anda meningkat seiring waktu. ”

Metrik Khusus Kubernetes dan EKS

Pantau metrik Kubernetes dan EKS untuk memastikan infrastruktur klaster Anda tetap sehat dan dapat mendukung beban kerja pelatihan Anda. Metrik ini membantu Anda mendeteksi masalah infrastruktur sebelum menyebabkan kegagalan pekerjaan pelatihan atau penurunan kinerja.

- Metrik Status Cluster Kubernetes — Memantau kesehatan dan status objek Kubernetes termasuk pod, node, penerapan, dan layanan. Lacak status pod untuk mengidentifikasi pod yang terjebak dalam status pending, failed, atau crash loop. Memantau kondisi node untuk mendeteksi masalah seperti tekanan disk, tekanan memori, atau tidak tersedianya jaringan. Gunakan kubectl atau alat pemantauan untuk memeriksa metrik ini secara terus menerus dan menyiapkan peringatan ketika pod gagal dimulai atau node menjadi tidak dapat dijadwalkan.
- Metrik Eksekusi Pipeline Training — Lacak proses pipeline yang berhasil dan gagal, durasi pekerjaan, waktu penyelesaian langkah, dan kesalahan orkestrasi. Pantau apakah pekerjaan pelatihan selesai dalam jangka waktu yang diharapkan dan apakah tingkat kegagalan meningkat seiring waktu. Lacak metrik seperti tingkat keberhasilan pekerjaan, durasi pekerjaan rata-rata,

dan waktu kegagalan. Metrik ini membantu Anda mengidentifikasi apakah masalah infrastruktur, masalah konfigurasi, atau masalah kualitas data menyebabkan kegagalan pelatihan.

- AWS Service Metrics — Lacak metrik untuk layanan AWS yang mendukung infrastruktur EKS dan beban kerja pelatihan Anda. Pantau metrik S3 seperti latensi permintaan, tingkat kesalahan, dan throughput untuk memastikan kinerja pemuatan data tetap konsisten. Lacak metrik volume EBS termasuk IOPS, throughput, dan panjang antrian untuk mendeteksi kemacetan penyimpanan. Pantau log aliran VPC dan metrik jaringan untuk mengidentifikasi masalah konektivitas antar node atau ke layanan eksternal.
- Kubernetes Control Plane Metrics — Memantau API server, scheduler, controller manager, dan database etcd untuk mendeteksi masalah kinerja atau kegagalan yang mempengaruhi operasi klaster. Lacak latensi permintaan server API, tingkat permintaan, dan tingkat kesalahan untuk memastikan bidang kontrol merespons permintaan penjadwalan dengan cepat. Pantau ukuran database etcd, durasi komit, dan perubahan pemimpin untuk mendeteksi masalah stabilitas. Latensi server API yang tinggi atau perubahan pemimpin etcd yang sering dapat menunda penjadwalan pod dan memperpanjang waktu startup pekerjaan pelatihan.

Aplikasi dan Log Instance

Kumpulkan dan analisis log aplikasi dan instance untuk mendiagnosis masalah yang tidak dapat dijelaskan oleh metrik saja. Log memberikan konteks terperinci tentang kesalahan, perubahan status, dan peristiwa sistem yang membantu Anda memahami mengapa pekerjaan pelatihan gagal atau berkinerja buruk. Mengkorelasikan log dengan metrik memungkinkan Anda menentukan akar penyebab lebih cepat.

- Log Aplikasi - Kumpulkan log aplikasi dari pekerjaan pelatihan, jaringan data, dan kerangka kerja ML Anda untuk mengidentifikasi kemacetan dan mendiagnosis kegagalan. Log ini menangkap informasi terperinci tentang eksekusi pekerjaan, termasuk kesalahan pemuatan data, kegagalan inisialisasi model, kesalahan penyimpanan pos pemeriksaan, dan peringatan khusus kerangka kerja. Korelasikan stempel waktu log dengan lonjakan metrik untuk memahami apa yang menyebabkan penurunan atau kegagalan kinerja. Misalnya, jika pemanfaatan GPU turun tiba-tiba, periksa log aplikasi untuk kesalahan yang menunjukkan penghentian pipa data atau kegagalan pra-pemrosesan. Gunakan alat logging terpusat seperti CloudWatch Logs atau Fluent Bit untuk mengumpulkan log dari semua pod dan membuatnya dapat dicari.
- Log Instance - Kumpulkan log tingkat instance seperti log jurnal sistem dan output dmesg untuk mendeteksi masalah perangkat keras dan masalah tingkat kernel. Log ini mengungkapkan masalah seperti kesalahan driver GPU, kegagalan alokasi memori, I/O kesalahan disk, dan

masalah antarmuka jaringan yang mungkin tidak muncul di log aplikasi. Korelasikan log instans dengan log aplikasi dan metrik untuk menentukan apakah kegagalan pelatihan berasal dari masalah perangkat keras atau masalah aplikasi. Misalnya, jika pekerjaan pelatihan gagal dengan out-of-memory kesalahan, periksa log dmesg untuk pesan pembunuh kernel OOM yang menunjukkan apakah sistem kehabisan memori atau apakah aplikasi melebihi batas kontainernya. Siapkan peringatan untuk kesalahan perangkat keras kritis seperti kesalahan GPU XID atau kegagalan disk sehingga Anda dapat mengganti instance yang gagal sebelum menyebabkan gangguan pelatihan yang meluas.

Bagian berikut menunjukkan cara mengumpulkan metrik yang dijelaskan di atas menggunakan dua pendekatan yang direkomendasikan AWS: Wawasan CloudWatch Kontainer dan Prometheus Amazon Managed Prometheus Amazon Managed Prometheus dengan Grafana yang Dikelola Amazon. Pilih CloudWatch Container Insights jika Anda lebih suka alat asli AWS dengan pengaturan minimal dan dasbor pra-bangun. Pilih Prometheus yang Dikelola Amazon dengan Grafana Terkelola Amazon jika Anda memerlukan dasbor yang disesuaikan, kemampuan visualisasi lanjutan, atau ingin berintegrasi dengan infrastruktur pemantauan berbasis Prometheus yang ada. Untuk daftar lengkap metrik Container Insights yang tersedia, lihat metrik Amazon EKS dan Kubernetes Container Insights.

Pertimbangkan untuk Memantau Metrik Inferensi Online Waktu Nyata

Dalam sistem real-time, latensi rendah sangat penting untuk memberikan respons tepat waktu kepada pengguna atau sistem dependen lainnya. Latensi tinggi dapat menurunkan pengalaman pengguna atau melanggar persyaratan kinerja. Komponen yang mempengaruhi latensi inferensi termasuk waktu pemuatan model, waktu pra-pemrosesan, waktu prediksi aktual, waktu pasca-pemrosesan, waktu transmisi jaringan. Kami merekomendasikan pemantauan latensi inferensi untuk memastikan respons latensi rendah yang memenuhi perjanjian tingkat layanan (SLAs) dan mengembangkan metrik khusus untuk hal-hal berikut. Uji di bawah beban yang diharapkan, termasuk latensi jaringan, memperhitungkan permintaan bersamaan, dan uji dengan berbagai ukuran batch.

- Time to First Token (TTFT) — Jumlah waktu dari saat pengguna mengirimkan permintaan hingga mereka menerima awal respons (kata pertama, token, atau potongan). Misalnya, di chatbots, Anda akan memeriksa berapa lama waktu yang dibutuhkan untuk menghasilkan bagian pertama dari output (token) setelah pengguna mengajukan pertanyaan.
- End-to-End Latensi — Ini adalah total waktu dari saat permintaan diterima hingga saat respons dikirim kembali. Misalnya, ukur waktu dari permintaan ke respons.

- Output Tokens Per Second (TPS) — Menunjukkan seberapa cepat model Anda menghasilkan token baru setelah mulai merespons. Misalnya, di chatbots, Anda akan melacak kecepatan pembuatan model bahasa untuk teks dasar.
- Tingkat Kesalahan - Melacak permintaan yang gagal, yang dapat menunjukkan masalah kinerja. Misalnya, pantau permintaan yang gagal untuk dokumen besar atau karakter tertentu.
- Throughput — Mengukur jumlah permintaan atau operasi yang dapat ditangani sistem per unit waktu. Misalnya, lacak permintaan per detik untuk menangani beban puncak.

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/Vatau penggunaan cache sesi):

- hit/miss Rasio cache — Untuk pengaturan inferensi yang memanfaatkan caching (K/V atau menyematkan cache), ukur seberapa sering cache membantu. Tingkat hit yang rendah dapat mengindikasikan konfigurasi cache suboptimal atau perubahan beban kerja, yang keduanya dapat meningkatkan latensi.

Dalam topik berikutnya, kami mendemonstrasikan pengumpulan data untuk beberapa metrik yang disebutkan di atas. Kami akan memberikan contoh dengan dua pendekatan yang direkomendasikan AWS: Wawasan CloudWatch Kontainer Asli AWS dan Prometheus Terkelola Amazon sumber terbuka dengan Grafana Terkelola Amazon. Anda akan memilih salah satu dari solusi ini berdasarkan kebutuhan observabilitas Anda secara keseluruhan. Lihat metrik Amazon EKS dan Kubernetes Container Insights untuk mengetahui daftar lengkap metrik Container Insights.

Melacak Penggunaan Memori GPU

Seperti yang dibahas dalam the section called “Pertimbangkan untuk Memantau Pelatihan Inti & Metrik Penyetelan Sempurna” topik, penggunaan memori GPU sangat penting untuk mencegah kesalahan out-of-memory (OOM) dan memastikan pemanfaatan sumber daya yang efisien. Contoh berikut menunjukkan cara instrumen aplikasi pelatihan Anda untuk mengekspos metrik histogram khusus,`gpu_memory_usage_bytes`, dan menghitung penggunaan memori P95 untuk mengidentifikasi konsumsi puncak. Pastikan untuk menguji dengan pekerjaan pelatihan sampel (misalnya, menyempurnakan model transformator) di lingkungan pementasan.

Contoh Wawasan CloudWatch Kontainer Asli AWS

Sampel ini menunjukkan cara instrumen aplikasi pelatihan Anda untuk mengekspos `gpu_memory_usage_bytes` sebagai histogram menggunakan pendekatan AWS-native. Perhatikan bahwa AI/ML container Anda harus dikonfigurasi untuk memancarkan log terstruktur dalam format CloudWatch Embedded Metrics Format (EMF). CloudWatch log mem-parsing EMF dan menerbitkan metrik. Gunakan aws_embedded_metrics dalam aplikasi pelatihan Anda untuk mengirim log terstruktur dalam format EMF ke Log, yang mengekstrak metrik 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()
```

Contoh Prometheus dan Grafana

Contoh ini menunjukkan cara instrumen aplikasi pelatihan Anda untuk mengekspos `gpu_memory_usage_bytes` sebagai histogram menggunakan pustaka klien Prometheus dengan 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()
```

Durasi Permintaan Inferensi Lacak untuk Inferensi Online Waktu Nyata

Seperti yang dibahas dalam the section called “Pertimbangkan untuk Memantau Pelatihan Inti & Metrik Penyetelan Sempurna” topik, latensi rendah sangat penting untuk memberikan respons tepat waktu kepada pengguna atau sistem dependen lainnya. Contoh berikut menunjukkan cara melacak metrik histogram kustom`inference_request_duration_seconds`, yang diekspos oleh aplikasi inferensi Anda. Hitung latensi persentil ke-95 (P95) untuk fokus pada skenario terburuk, uji dengan permintaan inferensi sintetis (misalnya, melalui Locust) di lingkungan pementasan, dan tetapkan ambang peringatan (misalnya, > 500 ms) untuk mendeteksi pelanggaran SLA.

Contoh Wawasan CloudWatch Kontainer Asli AWS

Contoh ini menunjukkan cara membuat metrik histogram kustom dalam aplikasi inferensi Anda untuk inference_request_duration_seconds menggunakan 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()
```

Contoh Prometheus dan Grafana

Contoh ini menunjukkan cara membuat metrik histogram kustom dalam aplikasi inferensi Anda untuk inference_request_duration_seconds menggunakan pustaka klien Prometheus dengan 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)
```

Di Grafana, gunakan kueri `histogram_quantile(0.95, sum(rate(inference_request_duration_seconds_bucket[5m])) by (le, pod))` untuk memvisualisasikan tren latensi P95. Untuk mempelajari lebih lanjut, lihat Dokumentasi Histogram Prometheus dan Dokumentasi Klien Prometheus.

Lacak Throughput Token untuk Inferensi Online Waktu Nyata

Seperti yang dibahas dalam the section called “Pertimbangkan untuk Memantau Pelatihan Inti & Metrik Penyetelan Sempurna” topik, kami merekomendasikan pemantauan waktu pemrosesan token untuk mengukur kinerja model dan mengoptimalkan keputusan penskalaan. Contoh berikut menunjukkan cara melacak metrik histogram kustom`token_processing_duration_seconds`, yang diekspos oleh aplikasi inferensi Anda. Hitung durasi persentil ke-95 (P95) untuk menganalisis efisiensi pemrosesan, uji dengan beban permintaan simulasi (misalnya, 100 hingga 1000 permintaan/detik) dalam cluster non-produksi, dan sesuaikan pemicu KEDA untuk mengoptimalkan penskalaan.

Contoh Wawasan CloudWatch Kontainer Asli AWS

Contoh ini menunjukkan cara membuat metrik histogram kustom dalam aplikasi inferensi Anda untuk token_processing_duration_seconds menggunakan AWS Embedded Metric Format. CloudWatch Ini menggunakan dimensi `() with a custom `get_duration_bucket`fungsi `set_dimension untuk mengkategorikan durasi ke dalam ember (misalnya, “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
```

Contoh Prometheus dan Grafana

Contoh ini menunjukkan cara membuat metrik histogram kustom dalam aplikasi inferensi Anda untuk token_processing_duration_seconds menggunakan pustaka klien Prometheus dengan 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)
```

Di Grafana, gunakan kueri `histogram_quantile(0.95, sum(rate(token_processing_duration_seconds_bucket[5m])) by (le, pod))` untuk memvisualisasikan tren waktu pemrosesan P95. Untuk mempelajari lebih lanjut, lihat Dokumentasi Histogram Prometheus dan Dokumentasi Klien Prometheus.

Ukur Latensi Pemulihan Pos Pemeriksaan

Seperti yang dibahas dalam the section called “Pertimbangkan untuk Memantau Pelatihan Inti & Metrik Penyetelan Sempurna” topik, latensi pos pemeriksaan adalah metrik penting selama beberapa fase siklus hidup model. Contoh berikut menunjukkan cara melacak metrik histogram kustom`checkpoint_restore_duration_seconds`, yang diekspos oleh aplikasi Anda. Hitung durasi persentil ke-95 (P95) untuk memantau kinerja pemulihan, uji dengan interupsi Spot di cluster non-produksi, dan atur ambang peringatan (misalnya, <30 detik) untuk mendeteksi penundaan.

Contoh Wawasan CloudWatch Kontainer Asli AWS

Contoh ini menunjukkan cara instrumen aplikasi batch Anda untuk mengekspos checkpoint_restore_duration_seconds sebagai histogram menggunakan 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")
```

Contoh Prometheus dan Grafana

Contoh ini menunjukkan cara instrumen aplikasi batch Anda untuk mengekspos `checkpoint_restore_duration_seconds` sebagai histogram menggunakan pustaka klien Prometheus dengan 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"))
```

Di Grafana, gunakan kueri `histogram_quantile(0.95, sum(rate(checkpoint_restore_duration_seconds_bucket[5m]) by (le))` untuk memvisualisasikan tren latensi pemulihan P95. Untuk mempelajari lebih lanjut, lihat Dokumentasi Histogram Prometheus dan Dokumentasi Klien Prometheus.

Penskalaan dan Kinerja Aplikasi

> **Tip**
>
> Jelajahi praktik terbaik melalui lokakarya Amazon EKS.

Mengelola Artefak ML, Melayani Kerangka Kerja, dan Optimasi Startup

Menerapkan model machine learning (ML) di Amazon EKS memerlukan pertimbangan yang matang tentang bagaimana model diintegrasikan ke dalam gambar kontainer dan lingkungan runtime. Ini memastikan skalabilitas, reproduktifitas, dan pemanfaatan sumber daya yang efisien. Topik ini menjelaskan berbagai pendekatan untuk menangani artefak model ML, memilih kerangka kerja

penyajian, dan mengoptimalkan waktu startup kontainer melalui teknik seperti pra-caching, semuanya disesuaikan untuk mengurangi waktu startup kontainer.

Menangani Artefak Model ML dalam Penerapan

Keputusan kuncinya adalah bagaimana menangani artefak model ML (seperti bobot dan konfigurasi) itu sendiri. Pilihannya memengaruhi ukuran gambar, kecepatan penerapan, frekuensi pembaruan model, dan overhead operasional. Perhatikan bahwa ketika mengacu pada penyimpanan “model”, kami mengacu pada artefak model (seperti parameter terlatih dan bobot model). Ada berbagai pendekatan untuk menangani artefak model ML di Amazon EKS. Masing-masing memiliki trade-off, dan yang terbaik tergantung pada ukuran model Anda, irama pembaruan, dan kebutuhan infrastruktur. Pertimbangkan pendekatan berikut dari yang paling sedikit hingga yang paling direkomendasikan:

- Memanggang model ke dalam gambar wadah: Salin file model (misalnya, .safetensors, .pth, .h5) ke dalam gambar wadah (misalnya, Dockerfile) selama pembuatan gambar. Model adalah bagian dari gambar yang tidak dapat diubah. Kami merekomendasikan menggunakan pendekatan ini untuk model yang lebih kecil dengan pembaruan yang jarang. Ini memastikan konsistensi dan reproduktifitas, tidak memberikan penundaan pemuatan, dan menyederhanakan manajemen ketergantungan, tetapi menghasilkan ukuran gambar yang lebih besar, memperlambat build dan dorongan, memerlukan pembangunan kembali dan penempatan kembali untuk pembaruan model, dan ini tidak ideal untuk model besar karena throughput tarik registri.
- Mengunduh model saat runtime: Saat startup container, aplikasi mengunduh model dari penyimpanan eksternal (misalnya, Amazon S3, didukung oleh S3 CRT untuk transfer throughput tinggi yang dioptimalkan menggunakan metode seperti Mountpoint untuk driver S3 CSI, AWS S3 CLI, atau CLI OSS s5cmd) melalui skrip dalam wadah init atau titik masuk. Kami merekomendasikan memulai dengan pendekatan ini untuk model besar dengan pembaruan yang sering. Ini membuat gambar kontainer tetap fokus pada kode/runtime, memungkinkan pembaruan model yang mudah tanpa membangun kembali, mendukung pembuatan versi melalui metadata penyimpanan, tetapi memperkenalkan potensi kegagalan jaringan (memerlukan logika coba lagi), memerlukan otentikasi dan caching.

Untuk mempelajari lebih lanjut, lihat Mempercepat proses tarik di AI di Lokakarya EKS.

Melayani Model ML

Menyebarkan dan menyajikan model machine learning (ML) di Amazon EKS memerlukan pemilihan pendekatan penyajian model yang sesuai untuk mengoptimalkan latensi, throughput, skalabilitas,

dan kesederhanaan operasional. Pilihannya tergantung pada jenis model Anda (misalnya, bahasa, model visi), tuntutan beban kerja (misalnya, inferensi waktu nyata), dan keahlian tim. Pendekatan umum termasuk pengaturan berbasis Python untuk pembuatan prototipe, server model khusus untuk fitur tingkat produksi, dan mesin inferensi khusus untuk kinerja dan efisiensi tinggi. Setiap metode melibatkan trade-off dalam kompleksitas pengaturan, kinerja, dan pemanfaatan sumber daya. Perhatikan bahwa kerangka kerja penyajian dapat meningkatkan ukuran gambar kontainer (beberapa GBs) karena dependensi, yang berpotensi memengaruhi waktu startup—pertimbangkan untuk memisahkan menggunakan teknik penanganan artefak untuk mengurangi hal ini. Opsi terdaftar dari yang paling sedikit hingga yang paling direkomendasikan:

Menggunakan kerangka kerja Python (misalnya, FastAP, HuggingFace Transformers dengan PyTorch) Kembangkan aplikasi khusus menggunakan kerangka kerja Python, menyematkan file model (bobot, konfigurasi, tokenizer) dalam pengaturan node kontainer.

- Kelebihan: Pembuatan prototipe yang mudah, hanya Python tanpa infrastruktur tambahan, kompatibel dengan semua HuggingFace model, penerapan Kubernetes sederhana.
- Kekurangan: Membatasi request/simple batching tunggal, pembuatan token lambat (tidak ada kernel yang dioptimalkan), memori tidak efisien, tidak memiliki penskalaan/pemantauan, dan melibatkan waktu startup yang lama.
- Rekomendasi: Gunakan untuk prototipe awal atau tugas simpul tunggal yang membutuhkan integrasi logika khusus.

Menggunakan kerangka kerja penyajian model khusus (misalnya, TensorRT-LLM, TGI) Mengadopsi server khusus seperti TensorRT-LLM atau TGI untuk inferensi MLM, mengelola pemuatan model, perutean, dan pengoptimalan. Format dukungan ini seperti safetensors, dengan kompilasi opsional atau plugin.

- Kelebihan: Menawarkan batching (static/in-flight or continuous), quantization (INT8, FP8, GPTQ), hardware optimizations (NVIDIA, AMD, Intel, Inferentia), and multi-GPU support (Tensor/ PipelineParalelisme). TensorRT-LLM mendukung beragam model (, Encoder-Decoder)LLMs, sementara TGI memanfaatkan integrasi. HuggingFace
- Kekurangan: TensorRT-LLM membutuhkan kompilasi dan hanya NVIDIA; TGI mungkin kurang efisien dalam pengelompokan; keduanya menambahkan overhead konfigurasi dan mungkin tidak cocok untuk semua jenis model (misalnya, non-transformator).
- Rekomendasi: Cocokkan untuk PyTorch/TensorFlow model yang membutuhkan kemampuan produksi seperti A/B pengujian atau throughput tinggi dengan perangkat keras yang kompatibel.

Menggunakan mesin inferensi throughput tinggi khusus (misalnya, VllM) Memanfaatkan mesin inferensi canggih seperti VllM, mengoptimalkan servis LLM dengan, in-flight batching, dan kuantisasi (, -KV, AWQ) PagedAttention, dapat diintegrasikan dengan penskalaan otomatis EKS. INT8 FP8

- Kelebihan: Throughput tinggi dan efisiensi memori (penghematan VRAM 40-60%), penanganan permintaan dinamis, streaming token, dukungan multi-GPU Tensor Paralel simpul tunggal, dan kompatibilitas perangkat keras yang luas.
- Kekurangan: Dioptimalkan untuk transformator khusus decoder (misalnya, LLa MA), kurang efektif untuk model non-transformator, memerlukan perangkat keras yang kompatibel (misalnya, NVIDIA) dan upaya penyiapan. GPUs
- Rekomendasi: Pilihan teratas untuk inferensi LLM volume tinggi dan latensi rendah pada EKS, memaksimalkan skalabilitas dan kinerja.

Mengoptimalkan waktu tarik gambar kontainer

Gambar kontainer yang besar dapat menyebabkan penundaan start dingin yang memengaruhi latensi start-up pod. Untuk beban kerja yang sensitif terhadap latensi, seperti beban kerja inferensi waktu nyata yang diskalakan secara horizontal, startup pod cepat sangat penting. Pertimbangkan pendekatan berikut untuk mengoptimalkan waktu tarik gambar kontainer:

Mengurangi Ukuran Gambar Kontainer

Mengurangi ukuran gambar kontainer selama startup adalah cara lain untuk membuat gambar lebih kecil. Anda dapat membuat pengurangan di setiap langkah proses pembuatan image container. Untuk memulai, pilih gambar dasar yang berisi jumlah dependensi paling sedikit yang diperlukan. Selama pembuatan gambar, sertakan hanya pustaka dan artefak penting yang diperlukan. Saat membuat gambar, coba gabungkan beberapa `RUN` atau `COPY` perintah untuk membuat lapisan yang lebih besar dalam jumlah yang lebih kecil. Untuk AI/ML kerangka kerja, gunakan build multi-tahap untuk memisahkan build dan runtime, hanya menyalin artefak yang diperlukan (misalnya, via `COPY –from=` untuk registri atau konteks lokal), dan pilih varian seperti gambar runtime saja (misalnya, pada 3,03 GB vs. devel pada 6,66 GB). `pytorch/pytorch:2.7.1-cuda11.8-cudnn9-runtime` Untuk mempelajari lebih lanjut, lihat Mengurangi ukuran gambar kontainer di AI di Workshop EKS.

Menggunakan snapshotter SOCI untuk Pra-tarik Gambar

Untuk gambar yang sangat besar yang tidak dapat Anda minimalkan dengan mudah, Anda dapat menggunakan snapshotter Seekable OCI (SOCI) open source yang dikonfigurasi dalam mode

pull and unpack paralel. Solusi ini memungkinkan Anda menggunakan gambar yang ada tanpa membangun kembali atau memodifikasi pipeline build Anda. Opsi ini sangat efektif saat menerapkan beban kerja dengan gambar yang sangat besar ke instans komputasi EC2 berkinerja tinggi. Ini bekerja dengan baik dengan jaringan throughput tinggi dan konfigurasi penyimpanan kinerja tinggi seperti biasa dengan beban kerja yang diskalakan. AI/ML

pull/unpack Mode paralel SOCI meningkatkan kinerja tarik end-to-end gambar melalui strategi paralelisasi yang dapat dikonfigurasi. Penarikan dan persiapan gambar yang lebih cepat berdampak langsung pada seberapa cepat Anda dapat menerapkan beban kerja baru dan menskalakan klaster Anda secara efisien. Penarikan gambar memiliki dua fase utama:

1. Mengambil lapisan dari registri ke node

   Untuk optimasi pengambilan lapisan, SOCI membuat beberapa koneksi HTTP bersamaan per lapisan, mengalikan throughput unduhan di luar batasan koneksi tunggal. Ini membagi lapisan besar menjadi beberapa bagian dan mengunduhnya secara bersamaan di beberapa koneksi. Pendekatan ini membantu menjenuhkan bandwidth jaringan Anda yang tersedia dan mengurangi waktu pengunduhan secara signifikan. Ini sangat berharga untuk AI/ML beban kerja di mana satu lapisan bisa beberapa gigabyte.

2. Membongkar dan menyiapkan lapisan-lapisan itu untuk membuat wadah

   Untuk optimasi pembongkaran lapisan, SOCI memproses beberapa lapisan secara bersamaan. Alih-alih menunggu setiap lapisan untuk membongkar sepenuhnya sebelum memulai yang berikutnya, ia menggunakan inti CPU Anda yang tersedia untuk mendekompresi dan mengekstrak beberapa lapisan secara bersamaan. Pemrosesan paralel ini mengubah fase pembongkaran terikat I/O secara tradisional menjadi operasi yang dioptimalkan CPU yang diskalakan dengan inti yang tersedia. Sistem dengan hati-hati mengatur paralelisasi ini untuk mempertahankan konsistensi sistem file sambil memaksimalkan throughput.

Mode tarik paralel SOCI menggunakan sistem kontrol ambang ganda dengan parameter yang dapat dikonfigurasi untuk mengunduh konkurensi dan membongkar paralelisme. Kontrol granular ini memungkinkan Anda menyempurnakan perilaku SOCI untuk memenuhi persyaratan kinerja spesifik dan kondisi lingkungan Anda. Memahami parameter ini membantu Anda mengoptimalkan runtime untuk kinerja tarik terbaik.

Referensi

- Untuk informasi selengkapnya tentang solusi dan tuning tradeoff, lihat dokumentasi fitur di repositori proyek SOCI pada. GitHub
- Untuk contoh langsung dengan Karpenter di Amazon EKS, lihat Karpenter Blueprint menggunakan mode paralel snapshotter SOCI. pull/unpack
- Untuk informasi tentang mengonfigurasi Bottlerocket untuk tarikan paralel, lihat Soci-snapshotter Parallel Pull Unpack Mode di Dokumentasi Bottlerocket.o

Menggunakan Snapshots EBS untuk Pra-tarik Gambar

Anda dapat mengambil snapshot Amazon Elastic Block Store (EBS) dari gambar kontainer yang di-cache dan menggunakan kembali snapshot ini untuk node pekerja EKS. Ini memastikan gambar diambil sebelumnya secara lokal saat node startup, mengurangi waktu inisialisasi pod. Lihat Mengurangi waktu startup container di Amazon EKS dengan volume data Bottlerocket untuk informasi selengkapnya menggunakan Karpenter dan EKS Terraform Blueprints untuk grup node terkelola.

Untuk mempelajari lebih lanjut, lihat Menggunakan snapshotter containerd dan Preload container image ke dalam volume data Bottlerocket dengan EBS Snapshots di AI di EKS Workshop.

Menggunakan Container Runtime Cache untuk Pra-tarik Gambar

Anda dapat melakukan pre-pull image container ke node menggunakan resource Kubernetes (misalnya, DaemonSet atau Deployment) untuk mengisi cache runtime container node. Cache runtime kontainer adalah penyimpanan lokal yang dikelola oleh runtime kontainer (misalnya, containerd tempat gambar disimpan setelah ditarik dari registri. Pra-penarikan memastikan gambar tersedia secara lokal, menghindari penundaan unduhan selama startup pod. Pendekatan ini sangat berguna ketika gambar sering berubah (misalnya, pembaruan yang sering), ketika snapshot EBS tidak dikonfigurasi sebelumnya, ketika membangun volume EBS akan lebih memakan waktu daripada menarik langsung dari registri kontainer, atau ketika node sudah ada di cluster dan perlu memutar pod sesuai permintaan menggunakan salah satu dari beberapa gambar yang mungkin.

Pra-penarikan semua varian memastikan waktu startup yang cepat terlepas dari gambar mana yang dibutuhkan. Misalnya, dalam beban kerja MS paralel besar-besaran yang membutuhkan 100.000 model kecil yang dibuat menggunakan 10 teknik berbeda, pra-penarikan 10 gambar melalui DaemonSet cluster besar (misalnya, ribuan node) meminimalkan waktu startup pod, memungkinkan penyelesaian dalam waktu kurang dari 10 detik dengan menghindari penarikan sesuai permintaan. Menggunakan pendekatan cache runtime kontainer menghilangkan kebutuhan untuk mengelola

snapshot EBS, memastikan Anda selalu mendapatkan versi gambar kontainer terbaru DaemonSets, tetapi untuk beban kerja inferensi waktu nyata di mana 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 pola skala node, ini dapat mengusir gambar pada node idle, yang memerlukan penarikan ulang selama peningkatan skala berikutnya dan mengurangi keandalan cache untuk beban kerja yang meledak.

Lihat AWS GitHub repository untuk contoh pra-penarikan gambar ke dalam cache runtime container.

Pertimbangkan NVMe untuk penyimpanan kubelet dan containerd

Pertimbangkan `containerd` untuk mengonfigurasi `kubelet` dan menggunakan disk penyimpanan NVMe instans singkat untuk kinerja disk yang lebih tinggi. Proses penarikan kontainer melibatkan pengunduhan gambar kontainer dari registri dan mendekompresi lapisannya menjadi format yang dapat digunakan. Untuk mengoptimalkan I/O operasi selama dekompresi, Anda harus mengevaluasi apa yang memberikan tingkat I/O kinerja dan throughput yang lebih tinggi untuk jenis NVMe instans host kontainer Anda: instance cadangan dengan penyimpanan lokal vs IOP/throughput Volume EBS. Untuk instans EC2 dengan penyimpanan NVMe lokal, pertimbangkan untuk mengonfigurasi sistem file dasar node untuk kubelet ()`/var/lib/kubelet`, containerd () dan Pod logs (`/var/log/pods`) untuk menggunakan disk penyimpanan instance sementara `/var/lib/containerd` untuk tingkat kinerja dan throughput yang lebih tinggi. NVMe I/O

Penyimpanan ephemeral node dapat dibagikan di antara Pod yang meminta penyimpanan sementara dan gambar kontainer yang diunduh ke node. Jika menggunakan Karpenter dengan Bottlerocket atau AL2023 EKS Dioptimalkan, AMIs ini dapat dikonfigurasi EC2NodeClassdengan pengaturan instanceStorePolicy ke RAID0atau, jika menggunakan Grup Node Terkelola, dengan mengatur localStoragePolicy in NodeConfigsebagai bagian dari data pengguna.

Berkontribusi pada panduan ini

Siapa pun dapat berkontribusi pada panduan praktik terbaik. Panduan Praktik Terbaik EKS ditulis dalam AsciiDoc format pada GitHub.

Ringkasan untuk kontributor yang ada

- Buka `bpg-docs.code-workspace`dengan VS Code untuk menginstal AsciiDoc ekstensi secara otomatis.
  - Pelajari lebih lanjut tentang AsciiDoc Ekstensi di Marketplace Visual Studio.
- File sumber untuk situs web AWS Docs disimpan di `latest/bpg`
- Sintaksnya sangat mirip dengan penurunan harga.
  - Tinjau Referensi Sintaks di AsciiDoctor dokumen.
- Platform dokumen hanya menerapkan`latest/bpg/images`. Setiap bagian panduan memiliki tautan simbolis kembali ke direktori ini. Misalnya, `latest/bpg/networking/images` menunjuk ke`latest/bpg/images`.

Siapkan lingkungan pengeditan lokal

Jika Anda berencana untuk sering mengedit panduan, siapkan lingkungan pengeditan lokal.

Garpu dan kloning repo

Anda harus terbiasa dengan`git`,`github`, dan editor teks. Untuk informasi tentang memulai `git` dan`github`, lihat Memulai GitHub akun Anda di GitHub dokumen.

1. Lihat Panduan Praktik Terbaik EKS di GitHub.
2. Buat fork dari repo proyek. Pelajari cara melakukan fork repositori di dokumen. GitHub
3. Kloning garpu repo proyek Anda. Pelajari cara mengkloning repositori bercabang Anda.

Buka Ruang Kerja VS Code

AWS merekomendasikan penggunaan Visual Studio Code dari Microsoft untuk mengedit panduan. Untuk informasi selengkapnya tentang VS Code, lihat Download Visual Studio Code dan Memulai Visual Studio Code di Visual Studio Code Documentation.

1. Buka Kode VS.
2. Buka `bpg-docs.code-workspace` file dari repo kloning.
3. Jika ini adalah pertama kalinya Anda membuka ruang kerja ini, terima prompt untuk menginstal AsciiDoc ekstensi. Ekstensi ini memeriksa sintaks AsciiDoc file dan menghasilkan pratinjau langsung.
4. Jelajahi `latest/bpg` direktori. Direktori ini menyimpan file sumber yang diterapkan ke situs dokumentasi AWS. File sumber diatur oleh bagian panduan, seperti “keamanan” atau “jaringan”.

Mengedit file

1. Buka file di editor.
   - Lihat AsciiDoc Sintaks untuk mempelajari cara membuat judul, tautan, dan daftar.
   - Anda dapat menggunakan sintaks Markdown untuk memformat teks, membuat daftar, dan judul. Anda tidak dapat menggunakan sintaks Markdown untuk membuat tautan.
2. Buka pratinjau langsung halaman.
   - Pertama, tekan `ctrl-k` atau `cmd-k` (tergantung keyboard). Kedua, tekan`v`. Ini membuka pratinjau dalam tampilan terpisah.

AWS menyarankan untuk menggunakan cabang fitur untuk mengatur perubahan Anda. Pelajari cara membuat cabang dengan git.

Kirim Permintaan Tarik

Anda dapat membuat permintaan tarik dari GitHub situs web atau GitHub cli.

Pelajari cara membuat permintaan tarik dari garpu dengan menggunakan GitHub Situs Web.

Pelajari cara membuat permintaan tarik dengan menggunakan GitHub cli.

Gunakan editor berbasis web github.dev

Editor `github.dev` berbasis web didasarkan pada VS Code. Ini adalah cara yang bagus untuk mengedit banyak file dan mempratinjau konten tanpa pengaturan apa pun.

Ini memiliki dukungan untuk AsciiDoc ekstensi. Anda dapat melakukan operasi git dengan menggunakan GUI. Editor berbasis web tidak memiliki shell atau terminal untuk menjalankan perintah.

Anda harus memiliki GitHub akun. Anda akan diminta untuk login jika diperlukan.

Luncurkan editor GitHub berbasis web.

Mengedit satu halaman

Anda dapat dengan cepat memperbarui halaman individual dengan menggunakan GitHub. Setiap halaman berisi tautan "# Edit halaman ini di GitHub" di bagian bawah.

1. Arahkan ke halaman dalam panduan ini yang ingin Anda edit
2. Klik tautan “Edit halaman ini di GitHub” di bagian bawah
3. Klik ikon edit pensil di kanan atas penampil GitHub file, atau tekan e
4. Edit file
5. Kirim perubahan Anda menggunakan tombol “Komit perubahan...”. Tombol ini membuat permintaan GitHub tarik. Pengelola panduan akan meninjau permintaan tarik ini. Peninjau akan menyetujui permintaan tarik, atau meminta perubahan.

Melihat dan mengatur ID untuk halaman

Halaman ini menjelaskan cara melihat dan mengatur ID halaman.

ID halaman adalah string unik yang mengidentifikasi setiap halaman di situs dokumentasi. Anda dapat melihat ID halaman di bilah alamat browser Anda saat Anda berada di halaman tertentu. ID halaman digunakan untuk URL, nama file, dan untuk membuat tautan referensi silang.

Misalnya, jika Anda melihat halaman ini, URL di bilah alamat browser Anda akan terlihat mirip dengan:

```
https://docs.aws.amazon.com/view-set-page-id.html
```

Bagian terakhir dari URL (`view-set-page-id`) adalah ID halaman.

Mengatur ID halaman

Saat membuat halaman baru, Anda perlu mengatur ID halaman di file sumber. ID halaman harus berupa string ringkas dan tanda hubung yang menjelaskan konten halaman.

1. Buka file sumber untuk halaman baru Anda di editor teks.
2. Di bagian atas file, tambahkan baris berikut. Itu harus di atas judul pertama.

```
[#my-new-page]
```

   Ganti `my-new-page` dengan ID halaman untuk halaman baru Anda.
3. Simpan file tersebut.

> **Note**
>
> Halaman IDs harus unik di seluruh situs dokumentasi. Jika Anda mencoba menggunakan ID halaman yang ada, Anda akan mendapatkan error build.

Buat halaman baru

Pelajari cara membuat halaman baru dan memperbarui daftar isi panduan.

Buat metadata halaman

1. Tentukan judul halaman, dan judul pendek halaman. Judul pendek halaman adalah opsional, tetapi disarankan jika judul halaman lebih dari beberapa kata.
2. Tentukan ID halaman. Ini harus unik dalam Panduan Praktik Terbaik EKS. Konvensi adalah menggunakan semua huruf kecil, dan memisahkan kata-kata dengan. -
3. Buat file asciidoc baru, dalam folder jika diperlukan, dan tambahkan teks berikut ke file:

   Example

   [.” topik "] [#<page-id>] = <page-title>:info_titleabbrev: < > page-short-title

   Misalnya,

   Example

   [.” topik "] [#scalability] = Praktik terbaik skalabilitas EKS:info_titleabbrev: Skalabilitas

Tambahkan ke daftar isi

1. Buka file untuk halaman induk dalam daftar isi. Untuk bagian panduan tingkat atas yang baru, file induknya adalah`book.adoc`.
2. Di bagian bawah file induk, perbarui dan masukkan arahan berikut:

   Example

   termasuk:: <new-filename>[leveloffset = +1]

   Sebagai contoh,

   Example

   termasuk: :dataplane.adoc [leveloffset = +1]

Sisipkan gambar

1. Temukan awalan gambar untuk halaman yang Anda edit. Tinjau `:imagesdir:` properti di judul file. Sebagai contoh, `` `:imagesdir: images/reliability/ ``
2. Tempatkan gambar Anda di jalur ini, seperti `latest/bpg/images/reliability`
3. Tentukan alt-text yang sesuai untuk gambar Anda. Tulis deskripsi singkat tingkat tinggi dari gambar. Misalnya, “diagram VPC dengan tiga zona ketersediaan” adalah teks alt yang sesuai.
4. Perbarui contoh berikut dengan alt-text dan nama file gambar. Masukkan di lokasi yang diinginkan.

   Example

   gambar:: <image-filename>[< image-alt-text >]

   Misalnya,

   Example

   gambar:: eks-data-plane-connectivity .jpeg [Diagram jaringan]

Periksa gaya dengan Vale

1. Instal CLI Vale.

2. Jalankan `vale sync`
3. Instal Vale Extension dari Visual Studio Marketplace.
4. Mulai ulang VS Code, dan buka AsciiDoc file
5. VS Code menggarisbawahi teks bermasalah. Pelajari cara bekerja dengan Kesalahan dan Peringatan di dokumen VS Code.

Membangun pratinjau lokal

1. Instal `asciidoctor` alat menggunakan `brew` di Linux atau macOS
   - Pelajari cara menginstal asciidoctor cli di dokumen. AsciiDoctor
   - Pelajari caranya instal manajer paket brew.
2. Buka terminal, dan arahkan ke `latest/bpg/`
3. Jalankan `asciidoctor book.adoc`
   - Tinjau peringatan dan kesalahan sintaks apa pun
4. Buka file `book.html` output.
   - Di macOS, Anda dapat menjalankan `open book.html` untuk membuka pratinjau di browser default Anda.

AsciiDoc Lembar Cheat

Pemformatan Dasar

```
*bold text*
_italic text_
`monospace text`
```

Header

```
= Document Title (Header 1)
== Header 2
=== Header 3
==== Header 4
===== Header 5
====== Header 6
```

Daftar

Daftar Tidak Berurutan:

```
- Item 1
- Item 2
-- Subitem 2.1
-- Subitem 2.2
- Item 3
```

Daftar yang Dipesan:

```
. First item
. Second item
.. Subitem 2.1
.. Subitem 2.2
. Third item
```

Tautan

```
External link:  https://example.com[Link text]
Internal link: <<page-id>>
Internal link: xref:page-id[Link text]
```

Citra

```
image::image-file.jpg[Alt text]
```

Blok Kode

```
 [source,python]
 ----
 def hello_world():
     print("Hello, World!")
 ----
```

Tabel

Pelajari cara membuat tabel dasar.

```
[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
|===
```

Peringatan

```
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.
```

Pratinjau:

> **Note**
>
> Ini adalah peringatan catatan.

Penyertaan

```
include::filename.adoc[]
```

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.