Menerapkan AWS Well-Architected Kerangka Kerja untuk Amazon Neptunus

# AWS Panduan Preskriptif

# AWS Panduan Preskriptif: Menerapkan AWS Well-Architected Kerangka Kerja untuk Amazon Neptunus

# Table of Contents

Pengantar ..... 1
Audiens yang dituju ..... 1
Tujuan ..... 1
Pilar keunggulan operasional ..... 3
Mengotomatiskan penerapan menggunakan pendekatan IAc ..... 3
Buat perubahan yang sering, kecil, dan reversibel ..... 4
Antisipasi kegagalan ..... 4
Belajar dari semua kegagalan operasional ..... 5
Gunakan kemampuan logging untuk memantau aktivitas yang tidak sah atau anomali ..... 6
Pilar keamanan ..... 7
Menerapkan keamanan data ..... 8
Amankan jaringan Anda ..... 9
Menerapkan otentikasi dan otorisasi ..... 9
Pilar keandalan ..... 11
Memahami kuota layanan Neptunus ..... 11
Memahami pola penyebaran Neptunus ..... 12
Kelola dan skala cluster Neptunus ..... 13
Kelola pencadangan dan peristiwa failover ..... 14
Pilar efisiensi performa ..... 15
Memahami pemodelan grafik ..... 15
Optimalkan kueri ..... 16
Cluster ukuran kanan ..... 18
Optimalkan menulis ..... 20
Pilar optimasi biaya ..... 21
Memahami pola penggunaan dan layanan yang dibutuhkan ..... 21
Pilih sumber daya dengan memperhatikan biaya ..... 22
Pilih konfigurasi instans Neptunus terbaik untuk beban kerja Anda ..... 24
Penyimpanan dan transfer data ukuran kanan ..... 25
Pilar keberlanjutan ..... 27
Wilayah AWS seleksi ..... 27
Konsumsi berdasarkan pola perilaku pengguna ..... 28
Optimalkan pengembangan perangkat lunak dan pola arsitektur ..... 28
Sumber daya ..... 30
Referensi ..... 30

Unggahan blog ..... 30
Kursus Pembuat AWS Keterampilan Gratis ..... 30
Kontributor ..... 31
Riwayat dokumen ..... 32
..... xxxiii

# Menerapkan Kerangka Kerja AWS Well-Architected untuk Amazon Neptunus

Amazon Web Services (kontributor)

Januari 2026 (sejarah dokumen)

Anda dapat membuat solusi berbasis grafik di Amazon Web Services (AWS) dengan menggunakan Amazon Neptunus. Panduan ini memberikan panduan preskriptif untuk menerapkan prinsip AWS Well-Architected Framework saat Anda merencanakan penerapan Neptunus.

AWS Well-Architected Framework membantu Anda membangun infrastruktur yang aman, berkinerja tinggi, tangguh, dan efisien untuk berbagai aplikasi dan beban kerja. Ini juga memberikan pendekatan yang konsisten bagi Anda untuk mengevaluasi arsitektur dan menerapkan desain yang dapat diskalakan.

Kerangka AWS Well-Architected dibangun di sekitar enam pilar berikut:

- Keunggulan operasional
- Keamanan
- Keandalan
- Efisiensi kinerja
- Optimalisasi biaya
- Keberlanjutan

Panduan ini memberikan informasi dari pilar desain AWS Well-Architected Framework dan praktik terbaik, dan pertimbangan yang perlu diingat saat menerapkan Neptunus. AWS

## Audiens yang dituju

Panduan ini ditujukan untuk insinyur data, arsitek solusi, dan analis data yang merancang dan menerapkan solusi yang menggunakan grafik. AWS

## Tujuan

Panduan ini dapat membantu Anda dan organisasi Anda melakukan hal berikut:

- Pilih dari opsi penerapan dan bahasa kueri yang didukung, berdasarkan kasus penggunaan dan pola kueri Anda.
- Ikuti pola desain AWS Well-Architected yang akan membantu meningkatkan ketahanan dan keamanan.
- Rancang kueri Anda untuk kinerja optimal dan penghematan biaya.
- Pelajari cara menjadi efisien secara operasional saat mengelola klaster Neptunus Anda dalam produksi.

# Pilar keunggulan operasional

Pilar keunggulan operasional dari AWS Well-Architected Framework berfokus pada menjalankan dan memantau sistem, dan terus meningkatkan proses dan prosedur. Ini mencakup kemampuan untuk mendukung pengembangan dan menjalankan beban kerja secara efektif, mendapatkan wawasan tentang operasi mereka, dan terus meningkatkan proses dan prosedur pendukung untuk memberikan nilai bisnis. Anda dapat mengurangi kompleksitas operasional melalui beban kerja penyembuhan diri, yang mendeteksi dan memperbaiki sebagian besar masalah tanpa campur tangan manusia. Anda dapat bekerja menuju tujuan ini dengan mengikuti praktik terbaik yang dijelaskan di bagian ini. Gunakan metrik APIs, dan mekanisme Amazon Neptunus untuk merespons dengan benar saat beban kerja Anda menyimpang dari perilaku yang diharapkan.

Diskusi pilar keunggulan operasional ini berfokus pada bidang-bidang utama berikut:

- Infrastruktur sebagai kode (IAc)
- Manajemen perubahan
- Strategi ketahanan
- Manajemen insiden
- Pelaporan audit untuk kepatuhan
- Pencatatan log dan pemantauan

## Mengotomatiskan penerapan menggunakan pendekatan IAc

Praktik terbaik untuk mengotomatiskan penerapan di Neptunus menggunakan IAc meliputi:

- Terapkan infrastruktur sebagai kode (IAc) untuk menyebarkan kluster Neptunus bila memungkinkan. Untuk konfigurasi lingkungan yang konsisten, gunakan AWS CloudFormationtemplat, AWS Cloud Development Kit (AWS CDK), atau HashiCorp Terraform untuk membuat semua sumber daya yang diperlukan untuk klaster Anda.
- Mengotomatiskan prosedur operasional Neptunus, seperti mengubah ukuran instance, menambahkan atau menghapus replika baca, atau melakukan failover manual pada tabel global, bila memungkinkan.
- Simpan string koneksi secara eksternal dari klien Anda. Gunakan proses ekstrak, transformasi, dan muat (ETL) untuk memfasilitasi strategi blue/green penerapan, pemulihan bencana (DR), dan migrasi downtime mendekati nol ke cluster baru. String koneksi dapat disimpan di AWS Secrets

Manager, Amazon DynamoDB, atau lokasi mana pun di mana mereka dapat diubah secara dinamis.

- Gunakan tag untuk menambahkan metadata ke sumber daya Neptunus Anda, dan lacak penggunaan berdasarkan tag. Untuk informasi selengkapnya, lihat Menandai Sumber Daya Amazon Neptunus.

## Buat perubahan yang sering, kecil, dan reversibel

Rekomendasi berikut berfokus pada perubahan kecil dan reversibel untuk meminimalkan kompleksitas dan mengurangi kemungkinan gangguan beban kerja:

- Simpan templat dan skrip IAC dalam layanan kontrol sumber, seperti GitHub atau. GitLab

> **Important**
>
> Jangan menyimpan AWS kredensil dalam kontrol sumber.

- Memerlukan penyebaran IAC untuk menggunakan layanan integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD), seperti atau. AWS CodePipelineAWS CodeBuild Layanan ini mengkompilasi, menguji, dan menyebarkan kode di lingkungan non-produksi yang berisi klaster Neptunus sesaat sebelum memengaruhi kluster Amazon Neptunus produksi Anda.
- Uji infrastruktur dan kueri aplikasi di lingkungan yang lebih rendah sebelum Anda menerapkannya ke produksi. Ini akan meminimalkan kemungkinan gangguan dan membantu memastikan mereka bekerja dengan baik dengan beban kerja dan skala Anda.

## Antisipasi kegagalan

Infrastruktur penyembuhan diri mencontohkan keunggulan operasional dengan mengantisipasi kegagalan dan berusaha menyelesaikan masalah apa pun tanpa intervensi. Rekomendasi berikut membantu Anda mencapai kedewasaan itu dengan Neptunus:

- Buat rencana pemantauan yang menggunakan CloudWatch metrik Amazon untuk memantau penggunaan CPU dan memori instans DB Anda, dan pahami pola penggunaannya. Buat CloudWatch dasbor dan alarm untuk metrik utama dan respons klien Neptunus yang ditemukan di log aplikasi Anda. Untuk informasi selengkapnya tentang indikator pemanfaatan CPU tinggi atau

rendah, lihat Menggunakan CloudWatch untuk memantau kinerja instans DB di Neptunus dalam dokumentasi Neptunus.

Jika Anda sering mendapatkan out-of-memory pengecualian pada kueri Anda, pertimbangkan untuk mengurangi jumlah total node yang dilalui kueri Anda atau coba gunakan instance dari `X2` keluarga, yang memiliki rasio lebih tinggi. RAM-to-CPU

- Atur notifikasi untuk memantau kesehatan cluster Neptunus. Misalnya, `BufferCacheHitRatio` harus selalu tinggi (lebih besar dari 99,9 persen), sedangkan `MainRequestQueuePendingRequests` harus selalu rendah (idealnya 0 tetapi tergantung pada persyaratan dan toleransi latensi Anda).
- Pertimbangkan untuk menggunakan replika baca untuk mencapai ketersediaan tinggi di Neptunus. Anda harus memiliki setidaknya dua replika baca di Availability Zone yang berbeda dari instance penulis untuk memastikan instance selalu tersedia untuk menyajikan kueri baca selama peristiwa failover.
- Secara otomatis menskalakan replika baca berdasarkan metrik pemanfaatan. Untuk informasi selengkapnya, lihat Penskalaan otomatis jumlah replika di klaster DB Amazon Neptunus.
- Tes failover untuk instans DB Anda untuk memahami berapa lama proses untuk kasus penggunaan Anda.
- Jika aplikasi Anda perlu bertahan dari Wilayah AWS pemadaman total, pertimbangkan untuk menggunakan database global sebagai bagian dari rencana DR Anda.

## Belajar dari semua kegagalan operasional

Infrastruktur penyembuhan diri adalah upaya jangka panjang yang berkembang dalam iterasi karena masalah langka terjadi atau respons tidak seefektif yang diinginkan. Mengadopsi praktik-praktik berikut mendorong fokus ke arah tujuan itu:

- Mendorong peningkatan dengan belajar dari semua kegagalan.
- Bagikan apa yang dipelajari di seluruh tim dan organisasi. Jika beberapa tim dalam organisasi menggunakan Neptunus, buat ruang obrolan umum atau grup pengguna untuk berbagi pembelajaran dan praktik terbaik.

# Gunakan kemampuan logging untuk memantau aktivitas yang tidak sah atau anomali

Untuk mengamati pola kinerja dan aktivitas anomali, simpan log di Amazon CloudWatch Logs. Pertimbangkan praktik terbaik berikut:

- Aktifkan pencatatan kueri lambat. Tinjau log secara teratur dan diagnosa mengapa pertanyaan tertentu lambat. Gunakan titik akhir penjelasan dan profil Neptunus untuk Gremlin, SPARQL, atau OpenCypher untuk mendapatkan wawasan mengapa kueri ini lambat.
- Aktifkan log audit Neptunus, dan tinjau log secara teratur untuk akses atau anomali yang tidak sah.
- Jika Anda menggunakan pencatatan kueri lambat atau pencatatan audit, aktifkan penerbitan ke CloudWatch Log. Ini akan membantu Anda menghindari kehabisan ruang disk pada instance. Instans Neptunus memiliki kapasitas penyimpanan log terbatas dan akan menimpa file log lama ketika ruang log terlampaui. CloudWatch Log mendukung retensi log jangka panjang. Kemampuan pemantauan yang ditingkatkan di CloudWatch Log akan meningkatkan kemampuan Anda untuk menanyakan log dan mendiagnosis masalah.
- Untuk memfasilitasi alat analisis yang lebih baik untuk log audit Anda, Anda dapat mengonfigurasi klaster DB Neptunus untuk mempublikasikan data log audit ke grup log di Log. CloudWatch Dengan CloudWatch Log, Anda dapat melakukan analisis real-time dari data log, digunakan CloudWatch untuk membuat alarm dan melihat metrik, dan menggunakan CloudWatch Log untuk menyimpan catatan log Anda dalam penyimpanan yang sangat tahan lama. Untuk informasi selengkapnya, lihat Menerbitkan log Neptunus ke Log Amazon. CloudWatch
- Neptunus mendukung pencatatan tindakan bidang kontrol menggunakan. AWS CloudTrail Untuk informasi selengkapnya, lihat Mencatat Panggilan API Amazon Neptunus dengan. AWS CloudTrail

# Pilar keamanan

Keamanan cloud di AWS adalah prioritas tertinggi. Sebagai AWS pelanggan, Anda mendapat manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. Model tanggung jawab bersama menggambarkan ini sebagai keamanan cloud dan keamanan di cloud:

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang berjalan Layanan AWS di dalamnya AWS Cloud. AWS juga memberi Anda layanan yang dapat Anda gunakan dengan aman. Auditor pihak ketiga secara teratur menguji dan memverifikasi efektivitas AWS keamanan sebagai bagian dari program AWS kepatuhan. Untuk mempelajari tentang program kepatuhan yang berlaku di Amazon Neptune, lihat Layanan AWS dalam Cakupan melalui Program Kepatuhan.
- Keamanan di cloud — Tanggung jawab Anda ditentukan oleh Layanan AWS yang Anda gunakan. Anda juga bertanggung jawab atas faktor lain, termasuk sensitivitas data Anda, persyaratan perusahaan Anda, serta undang-undang dan peraturan yang berlaku. Untuk informasi selengkapnya tentang privasi data, silakan lihat Pertanyaan Umum Privasi Data. Untuk informasi tentang perlindungan data di Eropa, lihat Model Tanggung Jawab AWS Bersama dan posting blog GDPR.

Pilar keamanan membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan Neptunus. Topik berikut akan menunjukkan kepada Anda cara mengonfigurasi Neptune untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga belajar cara menggunakan Layanan AWS yang lain yang membantu Anda memantau dan mengamankan sumber daya Neptunus Anda.

Pilar keamanan mencakup area fokus utama berikut:

- Keamanan data
- Keamanan jaringan
- Autentikasi dan otorisasi

# Menerapkan keamanan data

Kebocoran data dan pelanggaran menempatkan pelanggan Anda pada risiko dan dapat menyebabkan dampak negatif yang substansif pada perusahaan Anda. Praktik terbaik berikut membantu melindungi data pelanggan Anda dari paparan yang tidak disengaja dan berbahaya:

- Nama klaster, tag, grup parameter, peran AWS Identity and Access Management (IAM), dan metadata lainnya tidak boleh berisi informasi rahasia atau sensitif karena data tersebut mungkin muncul di log penagihan atau diagnostik.
- URIs atau tautan ke server eksternal yang disimpan sebagai data di Neptunus tidak boleh berisi informasi kredensyal untuk memvalidasi permintaan.
- Instans yang dienkripsi oleh Neptune memberikan lapisan perlindungan data tambahan dengan membantu mengamankan data Anda dari akses yang tidak sah ke penyimpanan yang mendasari. Anda dapat menggunakan enkripsi Neptune untuk meningkatkan perlindungan data aplikasi Anda yang disebarkan di cloud. Anda juga dapat enkripsi Neptunus untuk memenuhi persyaratan kepatuhan untuk data saat istirahat.

  Untuk mengaktifkan enkripsi untuk instans DB Neptunus baru, pilih Ya di bagian Aktifkan enkripsi pada konsol Neptunus (dipilih secara default), atau dengan menyetel properti di. AWS::Neptune::DBCluster::StorageEncrypted CloudFormationJika enkripsi diaktifkan, Neptunus akan menggunakan kunci yang dikelola AWS Amazon Relational Database Service (Amazon RDS) AWS secara default, atau Anda dapat membuat kunci yang dikelola pelanggan. Untuk informasi tentang membuat instans DB Neptunus, lihat Membuat cluster DB Neptunus baru. Untuk detail selengkapnya, lihat Mengenkripsi Sumber Daya Neptunus saat Istirahat. Snapshot otomatis dan manual Anda menggunakan enkripsi yang sama dengan yang Anda pilih untuk cluster Neptunus Anda.
- Saat menggunakan bahasa SPARQL dan OpenCypher, praktikkan validasi input dan teknik parameterisasi yang tepat untuk mencegah injeksi SQL dan bentuk serangan lainnya. Hindari membuat kueri yang menggunakan penggabungan string dengan input yang disediakan pengguna. Gunakan kueri berparameter atau pernyataan yang disiapkan untuk meneruskan parameter input dengan aman ke database grafik. Untuk informasi lebih lanjut, lihat Contoh kueri parameter OpenCypher dan Pertahanan Injeksi SPARQL.
- Untuk bahasa Gremlin, gunakan Varian Bahasa Gremlin alih-alih langsung meneruskan Skrip Gremlin berbasis string untuk menghindari potensi masalah injeksi.

# Amankan jaringan Anda

Cluster DB Amazon Neptunus hanya dapat dibuat di cloud pribadi virtual (VPC) aktif. AWS Sampai Neptunus 1.4.6.0, titik akhir cluster DB Neptunus hanya dapat diakses dalam VPC itu. Pada Neptunus 1.4.6.0 dan yang lebih baru, instance Neptunus dapat dikonfigurasi agar dapat diakses publik melalui internet. Ini adalah praktik terbaik untuk menggunakan fitur ini hanya di lingkungan non-produksi untuk mengaktifkan akses yang disederhanakan ke Neptunus untuk pengembang Anda (meskipun otentikasi IAM selalu diperlukan untuk mengaktifkan aksesibilitas publik). Jika Anda mengaktifkan aksesibilitas publik, pertimbangkan untuk menetapkan aturan grup keamanan masuk untuk port database Anda ke hanya lalu lintas alamat IP yang diketahui. Di lingkungan produksi atau dengan cluster yang berisi data sensitif, amankan data Neptunus Anda dengan tidak mengizinkan aksesibilitas publik dan membatasi akses ke VPC tempat cluster DB Neptunus Anda berada. Untuk informasi selengkapnya, lihat Menghubungkan ke grafik Amazon Neptunus Anda.

Untuk melindungi data Anda saat transit, Neptunus menerapkan koneksi SSL melalui HTTPS ke instans atau titik akhir cluster apa pun menggunakan protokol dan cipher yang aman. Neptunus menyediakan sertifikat SSL untuk instans DB Neptunus Anda. Sertifikat SSL Neptunus hanya mendukung nama host titik akhir klaster, titik akhir pembaca, dan titik akhir instance.

Jika Anda menggunakan penyeimbang beban atau server proxy (seperti HAProxy), Anda harus menggunakan penghentian SSL dan memiliki sertifikat SSL Anda sendiri di server proxy. Passthrough SSL tidak bekerja karena sertifikat SSL yang disediakan tidak cocok dengan nama host server proxy. Untuk informasi selengkapnya tentang menghubungkan ke titik akhir Neptunus dengan SSL, lihat Menggunakan titik akhir HTTP REST untuk terhubung ke instans DB Neptunus.

# Menerapkan otentikasi dan otorisasi

Untuk mengontrol siapa yang dapat melakukan tindakan manajemen Neptunus pada cluster DB Neptunus dan instans DB, aktifkan otentikasi database IAM dan gunakan kredensyal IAM. Saat Anda terhubung AWS menggunakan kredensyal IAM, peran IAM Anda harus memiliki kebijakan IAM yang memberikan izin yang diperlukan untuk melakukan operasi manajemen Neptunus. Pastikan Anda mengikuti prinsip hak istimewa paling sedikit, hanya memberikan izin yang diperlukan untuk menyelesaikan tugas. Untuk informasi selengkapnya, lihat Menggunakan berbagai jenis kebijakan IAM untuk mengontrol akses ke Neptunus dan Otentikasi IAM Menggunakan Kredensyal Sementara.

Untuk mengontrol siapa yang dapat terhubung ke cluster Neptunus dan menanyakan data, Anda dapat menggunakan IAM untuk mengautentikasi instans DB Neptunus atau cluster DB Anda. Jika

Anda mengaktifkan otentikasi IAM di cluster DB Neptunus, siapa pun yang mengakses cluster DB harus diautentikasi terlebih dahulu. Untuk informasi selengkapnya, lihat Mengaktifkan autentikasi database IAM di Neptunus untuk langkah-langkah mengaktifkan otentikasi IAM.

Ketika autentikasi basis data IAM diaktifkan, setiap permintaan harus ditandatangani menggunakan AWS Signature Version 4. Untuk memahami cara mengirim permintaan yang ditandatangani ke semua titik akhir Neptunus dengan autentikasi IAM diaktifkan, lihat Menghubungkan dan Menandatangani dengan Versi Tanda Tangan 4. AWS Banyak perpustakaan dan alat, seperti awscurl, sudah mendukung AWS Signature Version 4.

Untuk berinteraksi dengan orang lain Layanan AWS, Amazon Neptunus menggunakan peran terkait layanan IAM. Peran tertaut layanan adalah tipe IAM role unik yang tertaut langsung ke Neptune. Peran terkait layanan telah ditentukan sebelumnya oleh Neptunus dan mencakup semua izin yang diperlukan layanan untuk memanggil orang lain atas nama Anda. Layanan AWS Untuk informasi selengkapnya, lihat Menggunakan Peran Tertaut Layanan untuk Neptunus.

# Pilar keandalan

Pilar keandalan mencakup kemampuan beban kerja untuk menjalankan fungsi yang dimaksudkan dengan benar dan konsisten ketika diharapkan. Ini termasuk kemampuan untuk mengoperasikan dan menguji beban kerja di seluruh siklus hidupnya.

Beban kerja yang andal dimulai dengan desain perangkat lunak dan infrastruktur yang diputuskan sejak awal. Pilihan arsitektur Anda akan memengaruhi perilaku beban kerja Anda di semua pilar AWS Well-Architected. Untuk keandalan, terdapat beberapa pola tertentu yang harus diikuti.

Pilar keandalan berfokus pada bidang-bidang utama berikut:

- Arsitektur beban kerja, termasuk kuota layanan dan pola penerapan
- Manajemen perubahan
- Manajemen kegagalan

## Memahami kuota layanan Neptunus

Volume cluster Neptunus dapat tumbuh hingga ukuran maksimum 128 tebibytes (TiB) di semua yang didukung Wilayah AWS kecuali Tiongkok GovCloud dan, di mana kuotanya 64 TiB.

Kuota 128 TiB cukup untuk menyimpan sekitar 200-400 miliar objek dalam grafik. Dalam grafik properti berlabel (LPG), objek adalah simpul, tepi, atau properti pada simpul atau tepi. Dalam grafik Resource Description Framework (RDF), sebuah objek adalah quad.

Untuk cluster Neptunus Tanpa Server, Anda menetapkan jumlah minimum dan maksimum Unit Kapasitas Neptunus (). NCUs Setiap NCU terdiri dari 2 gibibytes (GiB) memori dan vCPU dan jaringan terkait. Nilai NCU minimum dan maksimum berlaku untuk setiap instance tanpa server di cluster. Nilai NCU maksimum tertinggi yang dapat Anda atur adalah 128,0 NCUs, dan minimum terendah adalah 1,0. NCUs Optimalkan rentang NCU yang paling sesuai untuk aplikasi Anda dengan mengamati CloudWatch metrik Amazon `ServerlessDatabaseCapacity` dan `NCUUtilization` menangkap rentang yang biasa Anda jalankan dan mengkorelasikan perilaku atau biaya yang tidak diinginkan dalam rentang tersebut. Dalam banyak beban kerja, 1.0 NCU terlalu rendah dari titik awal dan menghasilkan perilaku yang tidak dapat diandalkan setelah periode tidak aktif. Jika Anda menemukan bahwa beban kerja Anda tidak berskala cukup cepat, tingkatkan minimum NCUs untuk menyediakan pemrosesan yang cukup untuk lonjakan awal saat skala.

Masing-masing Akun AWS memiliki kuota untuk setiap Wilayah pada jumlah sumber daya database yang dapat Anda buat. Sumber daya ini termasuk instans DB dan klaster DB. Setelah batas sumber daya tercapai, panggilan tambahan untuk membuat sumber daya itu gagal dengan pengecualian. Beberapa kuota adalah kuota lunak yang dapat ditingkatkan berdasarkan permintaan. Untuk daftar kuota yang dibagikan antara Amazon Neptunus dan Amazon RDS, Amazon Aurora, dan Amazon DocumentDB (dengan kompatibilitas MongoDB), bersama dengan tautan untuk meminta peningkatan kuota bila tersedia, lihat Kuota di Amazon RDS.

## Memahami pola penyebaran Neptunus

Dalam klaster DB Neptune, terdapat satu instans DB utama dan hingga 15 replika Neptune. Instans DB primer mendukung operasi baca dan tulis, dan melakukan semua modifikasi data pada volume cluster. Replika Neptunus terhubung ke volume penyimpanan yang sama dengan instans DB utama, dan mereka hanya mendukung operasi baca. Replika Neptune juga dapat memindahkan beban kerja baca dari instans DB utama.

Untuk mencapai ketersediaan tinggi, gunakan replika baca. Memiliki satu atau lebih instance replika baca yang tersedia di Availability Zone yang berbeda dapat meningkatkan ketersediaan karena replika baca berfungsi sebagai target failover untuk instance utama. Jika instance penulis gagal, Neptunus mempromosikan instance replika baca untuk menjadi contoh utama. Ketika ini terjadi, ada gangguan singkat (umumnya kurang dari 30 detik) saat instance yang dipromosikan di-boot ulang, di mana permintaan baca dan tulis yang dibuat ke instance utama gagal dengan pengecualian. Untuk keandalan tertinggi, pertimbangkan dua replika baca di Availability Zone yang berbeda. Jika instance utama di Availability Zone 1 offline, instance di Availability Zone 2 dipromosikan ke primer, tetapi tidak dapat menangani kueri saat itu terjadi. Jadi instance di Availability Zone 3 diperlukan untuk menangani kueri baca selama transisi.

Jika Anda menggunakan Neptunus Tanpa Server, instance pembaca dan penulis di semua Availability Zone akan naik dan turun, secara independen satu sama lain, tergantung pada pemuatan database mereka. Anda dapat mengatur tingkat promosi instance pembaca ke 0 atau 1 sehingga skala naik dan turun bersama dengan kapasitas instance penulis. Ini membuatnya siap untuk mengambil alih beban kerja saat ini kapan saja.

Jika aplikasi Anda memiliki jejak di seluruh dunia atau memerlukan failover Multi-region, pertimbangkan untuk menggunakan database global Neptunus. Basis data global Amazon Neptunus mencakup Wilayah AWS beberapa, memungkinkan pembacaan global latensi rendah dan memberikan pemulihan cepat dalam kasus yang jarang terjadi di mana pemadaman mempengaruhi

keseluruhan. Wilayah AWS Database global Neptunus terdiri dari cluster DB primer di satu Wilayah dan hingga lima cluster DB sekunder di Wilayah yang berbeda.

## Kelola dan skala cluster Neptunus

Anda dapat menggunakan auto-scaling Neptunus untuk secara otomatis menyesuaikan jumlah replika Neptunus dalam cluster DB untuk memenuhi persyaratan konektivitas dan beban kerja Anda berdasarkan ambang batas pemanfaatan CPU. Dengan auto-scaling, cluster DB Neptunus Anda dapat menangani peningkatan beban kerja yang tiba-tiba. Ketika beban kerja berkurang, auto-scaling menghapus replika yang tidak perlu sehingga Anda tidak membayar untuk kapasitas yang tidak digunakan. Ketahuilah bahwa startup instance baru dapat memakan waktu selama 15 menit, jadi auto-scaling saja bukanlah solusi yang cukup untuk perubahan permintaan yang cepat.

Anda dapat menggunakan auto-scaling hanya dengan cluster DB Neptunus yang sudah memiliki satu instance penulis utama dan setidaknya satu instance read-replica (lihat Amazon Neptunus DB Clusters and Instances). Juga, semua instance read-replica di cluster harus dalam keadaan tersedia. Jika ada replika baca dalam keadaan selain tersedia, auto-scaling Neptunus tidak melakukan apa pun sampai setiap replika baca di cluster tersedia.

Jika Anda mengalami perubahan permintaan yang cepat, pertimbangkan untuk menggunakan instance tanpa server. Instans tanpa server dapat menskalakan secara vertikal selama periode pendek sementara auto-scaling menskalakan secara horizontal selama periode yang lebih lama. Konfigurasi ini memberikan skalabilitas optimal karena instans tanpa server menskalakan secara vertikal sementara auto-scaling membuat instance replika baca baru untuk menangani beban kerja di luar kapasitas maksimum instans tanpa server tunggal. Untuk informasi selengkapnya tentang penskalaan kapasitas Amazon Neptunus Tanpa Server, lihat Penskalaan kapasitas dalam klaster DB Tanpa Server Neptunus.

Jika penskalaan Anda perlu berubah pada waktu yang dapat diprediksi, Anda dapat menjadwalkan perubahan pada instans minimum, instans maksimum, dan ambang batas untuk menangani kebutuhan pergeseran tersebut dengan lebih baik. Ingatlah untuk menjadwalkan acara penskalaan setidaknya 15 menit sebelumnya untuk memungkinkan contoh-contoh tersebut online saat diperlukan.

Anda mengelola konfigurasi database Anda di Amazon Neptunus dengan menggunakan parameter dalam grup parameter. Grup parameter bertindak sebagai wadah untuk nilai konfigurasi mesin yang diterapkan ke satu atau lebih instance DB. Saat memodifikasi parameter cluster dalam kelompok parameter, pahami perbedaan antara parameter statis dan dinamis, dan bagaimana dan kapan

parameter tersebut diterapkan. Gunakan titik akhir status untuk melihat konfigurasi yang diterapkan saat ini.

## Kelola pencadangan dan peristiwa failover

Neptunus mencadangkan volume cluster Anda secara otomatis, dan menyimpan data yang dicadangkan selama periode retensi cadangan. Cadangan Neptune bersifat terus-menerus dan bertahap, sehingga Anda dapat dengan cepat memulihkan ke titik mana pun dalam periode penyimpanan cadangan. Anda dapat menentukan periode retensi cadangan 1-35 hari saat membuat atau memodifikasi cluster DB.

Untuk menyimpan cadangan di luar periode retensi cadangan, Anda juga dapat mengambil snapshot data dalam volume cluster Anda. Menyimpan snapshot menimbulkan biaya penyimpanan standar untuk Neptune.

Saat Anda membuat snapshot Amazon Neptunus dari cluster DB, Neptunus membuat snapshot volume penyimpanan cluster, mencadangkan semua datanya, bukan hanya instance individual. Anda nantinya dapat membuat cluster DB baru dengan memulihkan dari snapshot cluster DB ini. Saat Anda memulihkan cluster DB, Anda memberikan nama snapshot cluster DB untuk dipulihkan, dan kemudian Anda memberikan nama untuk cluster DB baru yang dibuat oleh pemulihan.

Uji bagaimana sistem Anda merespons peristiwa failover. Gunakan Neptunus API untuk memaksa peristiwa failover. Reboot dengan failover bermanfaat ketika Anda ingin mensimulasikan kegagalan instans DB untuk pengujian atau untuk memulihkan operasi ke Availability Zone asli setelah failover terjadi. Untuk informasi selengkapnya, lihat Mengonfigurasi dan mengelola penerapan Multi-AZ. Saat Anda me-reboot instance DB writer, itu gagal ke replika siaga. Mereboot replika Neptune tidak memulai failover.

Rancang klien Anda untuk keandalan. Uji perilaku mereka selama peristiwa failover. Terapkan logika coba lagi di klien Anda dengan logika backoff eksponensial. Contoh kode yang menerapkan logika ini dapat ditemukan dalam dokumentasi di bawah contoh AWS Lambda fungsi untuk Amazon Neptunus.

Pertimbangkan untuk menggunakan AWS Backupjika Anda memiliki serangkaian persyaratan cadangan umum yang Anda terapkan di beberapa mesin database.

# Pilar efisiensi performa

Pilar efisiensi kinerja dari AWS Well-Architected Framework berfokus pada cara mengoptimalkan kinerja sambil menelan atau menanyakan data. Optimalisasi kinerja adalah proses inkremental dan berkelanjutan dari hal-hal berikut:

- Mengonfirmasi persyaratan bisnis
- Mengukur kinerja beban kerja
- Mengidentifikasi komponen yang berkinerja buruk
- Menyetel komponen untuk memenuhi kebutuhan bisnis Anda

Pilar efisiensi kinerja menyediakan pedoman khusus kasus penggunaan yang dapat membantu mengidentifikasi model data grafik dan bahasa kueri yang tepat untuk digunakan. Ini juga mencakup praktik terbaik untuk diikuti saat menelan data ke dalam dan mengkonsumsi data dari Amazon Neptunus.

Pilar efisiensi kinerja berfokus pada bidang-bidang utama berikut:

- Pemodelan grafik
- Pengoptimalan kueri
- Ukuran kanan cluster
- Tulis optimasi

## Memahami pemodelan grafik

Memahami perbedaan antara model Labeled Property Graph (LPG) dan Resource Description Framework (RDF). Dalam kebanyakan kasus, ini adalah masalah preferensi. Namun, ada beberapa kasus penggunaan, di mana satu model lebih cocok daripada yang lain. Jika Anda memerlukan pengetahuan tentang jalur yang menghubungkan dua node dalam grafik Anda, pilih LPG. Jika Anda ingin menggabungkan data di seluruh cluster Neptunus atau penyimpanan tiga grafik lainnya, pilih RDF.

Jika Anda membangun aplikasi perangkat lunak sebagai layanan (SaaS) atau aplikasi yang membutuhkan multi-tenancy, pertimbangkan untuk memasukkan pemisahan logis penyewa dalam

model data Anda alih-alih memiliki satu penyewa untuk setiap cluster. Untuk mencapai jenis desain tersebut, Anda dapat menggunakan grafik bernama SPARQL dan strategi pelabelan, seperti menambahkan pengidentifikasi pelanggan ke label atau menambahkan pasangan nilai kunci properti yang mewakili pengidentifikasi penyewa. Pastikan layer klien Anda menyuntikkan nilai-nilai ini untuk menjaga pemisahan logis itu. Untuk informasi selengkapnya tentang rekomendasi multi-tenancy, lihat Panduan multi-penyewaan untuk menjalankan database Amazon ISVs Neptunus.

Kinerja kueri Anda tergantung pada jumlah objek grafik (node, tepi, properti) yang perlu dievaluasi dalam pemrosesan kueri Anda. Dengan demikian, model grafik dapat berdampak signifikan pada kinerja aplikasi Anda. Gunakan label granular bila memungkinkan, dan simpan hanya properti yang Anda butuhkan untuk mencapai penentuan jalur atau penyaringan. Untuk mencapai kinerja yang lebih tinggi, pertimbangkan untuk menghitung bagian grafik Anda, seperti membuat simpul ringkasan atau lebih banyak tepi langsung yang menghubungkan jalur umum.

Cobalah untuk menghindari navigasi melintasi node yang memiliki jumlah tepi yang sangat tinggi dengan label yang sama. Node seperti itu sering memiliki ribuan tepi (di mana sebagian besar node memiliki jumlah tepi dalam puluhan). Hasilnya adalah komputasi dan kompleksitas data yang jauh lebih tinggi. Node ini mungkin tidak bermasalah dalam beberapa pola kueri, tetapi kami menyarankan pemodelan data Anda secara berbeda untuk menghindarinya, terutama jika Anda akan menavigasi di seluruh node sebagai langkah perantara. Anda dapat menggunakan log kueri lambat untuk membantu mengidentifikasi kueri yang menavigasi di seluruh node ini. Anda mungkin akan mengamati latensi dan metrik akses data yang jauh lebih tinggi daripada pola kueri rata-rata Anda, terutama jika Anda menggunakan mode debug.

Gunakan simpul deterministik IDs untuk node dan tepi jika kasus penggunaan Anda mendukungnya alih-alih menggunakan Neptunus untuk menetapkan nilai GUID acak. IDs Mengakses node dengan ID adalah metode yang paling efisien.

## Optimalkan kueri

Bahasa OpenCypher dan Gremlin dapat digunakan secara bergantian pada model LPG. Jika kinerja menjadi perhatian utama, pertimbangkan untuk menggunakan dua bahasa secara bergantian karena yang satu mungkin berkinerja lebih baik daripada yang lain untuk pola kueri tertentu.

Neptunus sedang dalam proses konversi ke mesin kueri alternatifnya (DFE). OpenCypher berjalan pada DFE saja, tetapi kueri Gremlin dan SPARQL dapat diatur secara opsional untuk berjalan pada DFE dengan menggunakan anotasi kueri. Pertimbangkan untuk menguji kueri Anda dengan DFE diaktifkan dan membandingkan kinerja pola kueri Anda saat tidak menggunakan DFE.

Neptunus dioptimalkan untuk kueri tipe transaksional yang dimulai pada satu node atau set node dan menyebar dari sana, bukan kueri analitis yang mengevaluasi seluruh grafik. Untuk beban kerja kueri analitis Anda, gunakan Neptune Analytics. Neptune Analytics adalah pilihan ideal untuk beban kerja investigasi, eksplorasi, atau ilmu data yang memerlukan iterasi cepat untuk pemrosesan data, analitis, dan algoritmik. Itu juga dapat melakukan pencarian vektor pada data grafik dan dapat memuat data langsung dari instance database Neptunus Anda. Jika Neptunus Analytics tidak memenuhi kebutuhan Anda, Anda juga dapat AWS mempertimbangkan SDK untuk Panda atau menggunakan neptune-export yang dikombinasikan dengan atau Amazon EMR. AWS Glue

Untuk mengidentifikasi inefisiensi dan kemacetan dalam model dan kueri Anda, gunakan dan `explain` APIs untuk setiap bahasa kueri untuk mendapatkan penjelasan rinci tentang rencana kueri `profile` dan metrik kueri. Untuk informasi lebih lanjut, lihat profil Gremlin, OpenCypher menjelaskan, dan SPARQL menjelaskan.

Pahami pola kueri Anda. Jika jumlah tepi yang berbeda dalam grafik menjadi besar, strategi akses Neptunus default dapat menjadi tidak efisien. Pertanyaan berikut mungkin menjadi sangat tidak efisien:

- Kueri yang menavigasi mundur melintasi tepi saat tidak ada label tepi yang diberikan.
- Klausa yang menggunakan pola yang sama ini secara internal, seperti `.both()` di Gremlin, atau klausa yang menjatuhkan node dalam bahasa apa pun (yang mengharuskan menjatuhkan tepi masuk tanpa sepengetahuan label).
- Kueri yang mengakses nilai properti tanpa menentukan label properti. Pertanyaan ini mungkin menjadi sangat tidak efisien. Jika ini cocok dengan pola penggunaan Anda, pertimbangkan untuk mengaktifkan indeks OSGP (objek, subjek, grafik, predikat).

Gunakan pencatatan kueri lambat untuk mengidentifikasi kueri lambat. Kueri yang lambat dapat disebabkan oleh rencana kueri yang tidak dioptimalkan atau sejumlah besar pencarian indeks yang tidak perlu, yang dapat meningkatkan biaya. I/O Neptunus menjelaskan dan membuat profil titik akhir untuk Gremlin, SPARQL, atau OpenCypher dapat membantu Anda memahami mengapa kueri ini lambat. Penyebabnya mungkin termasuk yang berikut:

- Node dengan jumlah tepi yang sangat tinggi dibandingkan dengan simpul rata-rata dalam grafik (misalnya, ribuan dibandingkan dengan puluhan) dapat menambah kompleksitas komputasi dan karenanya latensi yang lebih lama dan konsumsi sumber daya yang lebih besar. Tentukan apakah node ini dimodelkan dengan benar, atau apakah pola akses dapat ditingkatkan untuk mengurangi jumlah tepi yang harus dilalui.

- Kueri yang tidak dioptimalkan akan berisi peringatan bahwa langkah-langkah tertentu tidak dioptimalkan. Menulis ulang kueri ini untuk menggunakan langkah-langkah yang dioptimalkan dapat meningkatkan kinerja.
- Filter redundan dapat menyebabkan pencarian indeks yang tidak perlu. Demikian juga, pola redundan dapat menyebabkan pencarian indeks duplikat yang dapat dioptimalkan dengan meningkatkan kueri (lihat `Index Operations - Duplication ratio` di output profil).
- Beberapa bahasa seperti Gremlin tidak memiliki nilai numerik yang diketik dengan kuat, dan mereka menggunakan promosi tipe sebagai gantinya. Misalnya, jika nilainya 55, Neptunus mencari nilai yang merupakan bilangan bulat, panjang, float, dan tipe numerik lainnya yang setara dengan 55. Ini menghasilkan operasi tambahan. Jika Anda tahu jenis Anda cocok sebelumnya, Anda dapat menghindari ini dengan menggunakan petunjuk kueri.
- Model grafik Anda dapat sangat memengaruhi kinerja. Pertimbangkan untuk mengurangi jumlah objek yang perlu dievaluasi dengan menggunakan label yang lebih granular atau dengan menghitung pintasan ke jalur linier multiple-hop.

Jika optimasi kueri saja tidak memungkinkan Anda untuk mencapai persyaratan kinerja Anda, pertimbangkan untuk menggunakan berbagai teknik caching dengan Neptunus untuk mencapai persyaratan tersebut.

Kinerja Neptunus terus meningkat dengan setiap versi. Tinjau catatan rilis untuk melihat detail peningkatan pada setiap rilis. Pertimbangkan untuk merencanakan pembaruan rutin ke cluster DB Neptunus Anda untuk membantu mencapai kinerja yang optimal. Versi yang lebih baru juga mendukung instance yang lebih baru. Pertimbangkan untuk memutakhirkan ke 1.4.5.0 atau yang lebih baru untuk dapat memanfaatkan instance. `r8g` Untuk informasi selengkapnya tentang cara ini dapat meningkatkan kinerja beban kerja Anda, lihat 4,7 kali lebih baik menulis kinerja harga kueri dengan instans AWS Graviton4 R8g menggunakan Amazon Neptunus v1.4.5.

## Cluster ukuran kanan

Ukur ukuran cluster Anda untuk persyaratan konkurensi dan throughput Anda. Jumlah query bersamaan yang dapat ditangani oleh setiap instance di cluster sama dengan dua kali jumlah virtual CPUs (vCPUs) pada instance itu. Kueri tambahan yang muncul saat semua thread pekerja ditempati dimasukkan ke dalam antrean sisi server. Kueri tersebut ditangani berdasarkan first-in-first-out (FIFO) saat thread pekerja tersedia. CloudWatch Metrik `MainRequestQueuePendingRequests` Amazon menunjukkan kedalaman antrian saat ini untuk setiap instance. Jika nilai ini sering di atas

nol, pertimbangkan untuk memilih instance dengan lebih banyak vCPUs. Jika kedalaman antrian melebihi 8.192, Neptunus akan mengembalikan kesalahan. `ThrottlingException`

Sekitar 65 persen dari RAM untuk setiap instance dicadangkan untuk buffer cache. Cache buffer menyimpan kumpulan data yang berfungsi (bukan seluruh grafik; hanya data yang sedang ditanyakan). Untuk menentukan persentase data yang diambil dari cache buffer alih-alih penyimpanan, pantau metrik. CloudWatch `BufferCacheHitRatio` Jika metrik ini sering turun di bawah 99,9 persen, pertimbangkan untuk mencoba instance dengan lebih banyak memori untuk menentukan apakah itu mengurangi latensi dan biaya Anda. I/O

Replika baca tidak harus berukuran sama dengan instance penulis Anda. Namun, beban kerja tulis yang berat dapat menyebabkan replika yang lebih kecil tertinggal dan reboot karena mereka tidak dapat mengikuti replikasi. Oleh karena itu, kami merekomendasikan membuat replika sama dengan atau lebih besar dari contoh penulis.

Saat menggunakan auto-scaling untuk replika baca Anda, ingatlah bahwa mungkin perlu waktu hingga 15 menit untuk menghadirkan replika baca baru secara online. Ketika lalu lintas klien meningkat dengan cepat tetapi dapat diprediksi, pertimbangkan untuk menggunakan penskalaan terjadwal untuk menetapkan jumlah minimum replika baca yang lebih tinggi untuk memperhitungkan waktu inisialisasi tersebut.

Instans tanpa server mendukung beberapa kasus penggunaan dan beban kerja yang berbeda. Pertimbangkan tanpa server atas instance yang disediakan untuk skenario berikut:

- Beban kerja Anda sering berfluktuasi sepanjang hari.
- Anda membuat aplikasi baru dan Anda tidak yakin berapa ukuran beban kerja nantinya.
- Anda sedang melakukan pengembangan dan pengujian.

Penting untuk dicatat bahwa instans tanpa server lebih mahal daripada instance yang disediakan setara dengan satu dolar per GB basis RAM. Setiap instans tanpa server terdiri dari 2 GB RAM bersama dengan vCPU dan jaringan terkait. Lakukan analisis biaya di antara opsi Anda untuk menghindari tagihan kejutan. Secara umum, Anda akan mencapai penghematan biaya dengan tanpa server hanya ketika beban kerja Anda sangat berat hanya beberapa jam setiap hari dan hampir nol sepanjang hari atau jika beban kerja Anda berfluktuasi secara signifikan sepanjang hari.

Gunakan Kalkulator Harga Amazon Neptunus untuk membantu menilai konfigurasi yang benar untuk klaster Anda berdasarkan queries-per-second faktor-faktor seperti persyaratan (QPS).

# Optimalkan menulis

Untuk mengoptimalkan penulisan, pertimbangkan hal berikut:

- Pemuat Massal Neptunus adalah cara optimal untuk memuat database Anda atau menambahkan ke data yang ada pada awalnya. Pemuat Neptunus tidak transaksional dan tidak dapat menghapus data, jadi jangan gunakan jika ini adalah persyaratan Anda.
- Pembaruan transaksional dapat dilakukan dengan menggunakan bahasa kueri yang didukung. Untuk mengoptimalkan I/O operasi tulis, tulis data dalam batch 50-100 objek per komit. Objek adalah node, edge, atau properti pada node atau edge di LPG, atau triple store atau quad di RDF..
- Semua operasi penulisan Neptunus transaksional adalah ulir tunggal untuk setiap koneksi. Saat mengirim sejumlah besar data ke Neptunus, pertimbangkan untuk memiliki beberapa koneksi paralel yang masing-masing merupakan data penulisan. Saat Anda memilih instance yang disediakan Neptunus, ukuran instans dikaitkan dengan sejumlah v. CPUs Neptunus membuat dua utas database untuk setiap vCPU pada instance, jadi mulailah dengan dua kali jumlah CPUs v saat menguji paralelisasi optimal.  Instans tanpa server menskalakan jumlah v dengan CPUs kecepatan kira-kira satu untuk setiap 4. NCUs

  > **Note**
  >
  > Ini tidak berlaku untuk API pemuatan massal, hanya koneksi langsung.

- Rencanakan dan tangani secara efisien ConcurrentModificationExceptionsselama semua proses penulisan, bahkan jika hanya satu koneksi yang menulis data kapan saja. Rancang klien Anda untuk keandalan saat `ConcurrentModificationExceptions` terjadi.
- Jika Anda ingin menghapus semua data Anda, pertimbangkan untuk menggunakan API reset cepat alih-alih mengeluarkan kueri penghapusan bersamaan. Yang terakhir akan memakan waktu lebih lama dan menimbulkan I/O biaya besar dibandingkan dengan yang pertama.
- Jika Anda ingin menghapus sebagian besar data Anda, pertimbangkan untuk mengekspor data yang ingin Anda simpan dengan menggunakan neptune-export untuk memuat data ke dalam klaster baru. Kemudian hapus cluster asli.

# Pilar optimasi biaya

Pilar optimasi biaya dari AWS Well-Architected Framework berfokus pada menghindari biaya yang tidak perlu. Rekomendasi berikut dapat membantu Anda memenuhi prinsip desain pengoptimalan biaya dan praktik terbaik arsitektur untuk Amazon Neptunus.

Pilar optimasi biaya berfokus pada bidang-bidang utama berikut:

- Memahami pengeluaran dari waktu ke waktu dan mengendalikan alokasi dana
- Memilih sumber daya dari jenis dan kuantitas yang tepat
- Penskalaan untuk memenuhi kebutuhan bisnis tanpa pengeluaran berlebihan

## Memahami pola penggunaan dan layanan yang dibutuhkan

Neptunus cocok untuk beban kerja Anda jika model data Anda memiliki struktur grafik yang dapat dilihat, dan kueri Anda perlu mengeksplorasi hubungan dan melintasi beberapa lompatan. Database grafik tidak cocok untuk pola berikut:

- Terutama kueri single-hop (pertimbangkan apakah data Anda mungkin lebih baik direpresentasikan sebagai atribut objek)
- Data JSON atau BLOB disimpan sebagai properti
- Kueri yang digabungkan di seluruh kumpulan data, seperti menghitung jumlah properti numerik di sejumlah besar node

Pertimbangkan apakah menggunakan beberapa database yang dibuat khusus bersama-sama untuk pola akses tertentu dapat memenuhi semua kebutuhan Anda. Contoh:

- API yang membutuhkan navigasi grafik kompleks yang lebih jarang di samping pengambilan properti yang sangat bersamaan untuk satu node mungkin paling baik disajikan dengan menggunakan satu atau lebih Neptune, DynamoDB, atau Amazon DocumentDB.
- Database relasional dapat hidup berdampingan dengan Neptunus untuk mempertahankan fungsionalitas Anda yang ada, tetapi gunakan Neptunus hanya untuk traversal multiple-hop yang tidak berkinerja dan skala yang baik dalam database relasional.

Memahami biaya yang terkait dengan layanan yang berinteraksi dengan dan melengkapi Neptunus, termasuk yang berikut:

- Biaya penyimpanan Amazon Simple Storage Service (Amazon S3) untuk file data yang dimuat secara massal ke Neptunus
- Fungsi Lambda yang digunakan untuk menyisipkan atau meningkatkan kueri, membaca kueri, dan pemrosesan aliran Neptunus
- Lapisan API yang dibangun di Neptunus untuk berinteraksi dengan aplikasi klien (alih-alih memiliki koneksi langsung ke database) di Amazon API Gateway atau AWS AppSync
- AWS Glue pekerjaan yang digunakan untuk mentransfer data ke dan dari Neptunus
- Amazon Kinesis atau Amazon Managed Streaming for Apache Kafka (Amazon MSK) instans menerima data streaming untuk konsumsi hampir real-time ke Neptunus.
- AWS Database Migration Service untuk migrasi data relasional ke Neptunus
- Biaya Amazon SageMaker Runtime untuk notebook Jupyter dan model pembelajaran mesin perpustakaan grafik mendalam

## Pilih sumber daya dengan memperhatikan biaya

Harga Neptunus didasarkan pada biaya instans per jam (atau Unit Komputasi Neptunus yang digunakan untuk tanpa server), I/O data, dan penggunaan penyimpanan. Contoh merupakan, rata-rata, 85 persen dari keseluruhan biaya, sehingga ukuran yang tepat dapat memiliki implikasi biaya yang signifikan. Cara terbaik untuk mengukur instans yang tepat adalah dengan menguji kinerja aplikasi pada berbagai instance dan membandingkan faktor-faktor berikut:

- Apakah `MainRequestQueuePendingRequests` CloudWatch metrik tetap pada angka rendah secara konsisten mendekati nol?
- Apakah `BufferCacheHitRatio` CloudWatch metrik tetap pada atau di atas 99,9 persen sebagian besar waktu?
- Berapa kurva biaya dan kinerja untuk biaya misalnya dan untuk I/O biaya data terkait? Biaya pembacaan data mungkin meningkat secara signifikan dengan instance berukuran kecil yang memerlukan pertukaran cache buffer yang sering dengan penyimpanan. `BufferCacheHitRatio`akan sering jatuh dalam skenario ini.

Biaya instans diskalakan secara linier dengan ukuran dalam keluarga instance yang sama. Biaya per jam `db.r6i.2xlarge` instance adalah dua kali lipat dari `db.r6i.xlarge` instance dan juga memiliki alokasi sumber daya dua kali lipat. `db.r6i.24xlarge`Instans adalah 24 kali biaya per jam dari `db.r6i.xlarge` instans.

Perkirakan jumlah kueri bersamaan yang harus Anda dukung. Anda dapat memiliki antara nol dan lima belas replika baca untuk memproses kueri hanya-baca. Jika persyaratan Anda bervariasi menurut waktu hari, minggu, atau bulan, Anda dapat menggunakan beberapa instance yang lebih kecil untuk menskalakan jadwal. Setiap vCPU pada sebuah instance menyediakan dua thread untuk menangani kueri bersamaan. Tiga replika `db.r6i.xlarge` baca, dengan masing-masing 4 vCPU, dapat menangani 24 kueri bersamaan..

Jika volume lalu lintas Anda diukur dalam kueri per detik (QPS), Anda harus bereksperimen untuk menentukan latensi rata-rata kueri Anda. Jumlah kueri per detik yang dapat didukung oleh cluster Neptunus sama dengan. `vCPU × 2 × (1 second/average query latency)` Misalnya, jika Anda memiliki 4 vCPU dan latensi kueri 100 milidetik (0,1 detik),. `QPS = 4 × 2 × (1s/0.1s) = 80 queries per second`

Instans yang disediakan lebih murah daripada tanpa server untuk beban kerja yang berkelanjutan, stabil, dan dapat diprediksi. Tanpa server memberikan peluang untuk mengoptimalkan biaya ketika Anda memiliki beban kerja yang membutuhkan penggunaan sangat tinggi hanya beberapa jam per hari (misalnya,`db.r6i.4xlarge`) dan kemudian hampir tidak ada lalu lintas untuk sisa hari itu (misalnya, 1 Unit Komputasi Neptunus). Instans tanpa server yang ditingkatkan selama beberapa jam dan kemudian mundur akan lebih murah daripada menggunakan instance yang disediakan sepanjang `db.r6i.4xlarge` hari.

Pertimbangkan untuk meningkatkan ke Neptunus 1.4.5.0 atau yang lebih baru dan `r8g` menggunakan instance untuk mencapai throughput baca dan tulis yang lebih baik dengan biaya lebih rendah daripada instance generasi yang lebih tua, seperti atau. `r7g` `r6g` Untuk informasi selengkapnya, lihat 4,7 kali lebih baik menulis kinerja harga kueri dengan instans AWS Graviton4 R8g menggunakan Amazon Neptunus v1.4.5 (posting blog).AWS

Cluster Neptunus dibuat secara default dengan penyimpanan standar (jika Anda membuat menggunakan konsol, itu akan default untuk I/O-optimized storage). With I/O-optimized storage, you pay a slightly higher cost for storage and instances, but there are no I/O costs. This leads to more predictable recurring costs, but if your I/O usage is generally low, it may be more cost efficient to utilize standard storage. If you intend to load a lot of data initially, you can optimize cost by choosing I/O memilih penyimpanan yang dioptimalkan, melakukan pemuatan data awal Anda, dan kemudian

beralih ke penyimpanan standar. Jenis penyimpanan hanya memengaruhi model penagihan dan tidak memiliki perbedaan teknis dalam cluster DB Neptunus atau konfigurasi instans. Anda dapat mengubah jenis penyimpanan sekali per 30 hari. Setelah 30 hari, periksa detail biaya Neptunus Anda dan gunakan halaman harga Neptunus untuk menghitung apakah biaya Anda akan lebih tinggi menggunakan -optimasi. I/O-optimized storage. If they would have been, continue to use standard storage, otherwise switch back to I/O

## Pilih konfigurasi instans Neptunus terbaik untuk beban kerja Anda

Jika Anda membuat Akun AWS sebelum 15 Juli 2025, Anda dapat menggunakan Tingkat AWS Gratis untuk eksperimen entry-level dengan Neptunus. 750 jam gratis `db.t3.medium` dan penggunaan `db.t4g.medium` instance sudah cukup bagi Anda untuk mendapatkan pemahaman yang baik tentang Neptunus dalam skala rendah. Cluster Anda akan tetap ada setelah periode uji coba gratis berakhir, meskipun Anda akan dikenakan biaya untuk penggunaan selanjutnya sejak saat itu.

`db.t4g.medium`Instance `db.t3.medium` dan instance bagus untuk lingkungan pengembangan berbiaya rendah di mana Anda tidak menggunakan OpenCypher, Graph Explorer, atau berbagai integrasi AI generatif. Contoh ini memiliki RAM-to-vCPU rasio yang lebih kecil (2:1) daripada contoh `R` keluarga (8:1) atau contoh `X` keluarga (16:1). Ini mengurangi rasio mencegah penggunaan statistik mesin DFE yang memungkinkan kinerja OpenCypher, integrasi GenAI (untuk menginformasikan LLM skema grafik), dan Graph Explorer. Profil kinerja mungkin berbeda secara signifikan saat menggunakan instance `T` keluarga, terutama untuk beban kerja yang disebutkan sebelumnya. Contoh ini juga dapat meningkatkan kemunculan `OutOfMemoryExceptions` ketika kueri menavigasi di sebagian besar grafik. Untuk menentukan apakah kondisi terakhir mungkin terpengaruh, periksa `BufferCacheHitRatio` CloudWatch metrik.

Kami sangat menyarankan untuk tidak melakukan pengujian kinerja atau beban apa pun dengan instance `T` keluarga karena Anda mungkin mengalami hasil yang tidak konsisten yang tidak menunjukkan lingkungan produksi.

Instans yang disediakan memberi Anda kombinasi biaya dan kinerja terbaik ketika beban kerja Anda cukup stabil dan dapat diprediksi. Pilih ukuran instance berdasarkan konkurensi permintaan yang diperlukan dan kompleksitas kueri. Konkurensi yang lebih tinggi membutuhkan lebih banyak vCPUs. Kompleksitas kueri yang lebih tinggi membutuhkan lebih banyak RAM. Gunakan `MainRequestQueuePendingRequests` CloudWatch metrik untuk menentukan dampak dari yang pertama (lebih besar dari nol mewakili lebih banyak permintaan bersamaan daripada yang dapat ditangani). Gunakan `BufferCacheHitRatio` CloudWatch metrik untuk menentukan dampak yang

terakhir. Rasio yang sering turun lebih rendah dari 99,9 persen menunjukkan bahwa tidak ada cukup RAM untuk menampung bagian kerja grafik yang sedang dievaluasi, yang menghasilkan pertukaran cache yang lebih sering. Jika keluarga R contoh memberikan konkurensi yang cukup tetapi tidak cukup RAM, pertimbangkan untuk mencoba X keluarga instance.

Kasus penggunaan ideal untuk instance tanpa server dijelaskan dalam dokumentasi Neptunus. Jika Anda tidak yakin apakah provisioned atau serverless adalah yang terbaik untuk Anda, dan biaya adalah perhatian utama Anda, uji beban kerja Anda di tanpa server untuk menentukan jumlah yang NCUs digunakan dan bandingkan biaya provisioned () dengan serverless (). `N hours × hourly provisioned cost sum of NCUs × hourly cost per NCU` Jika Anda tidak yakin tentang instance penyediaan berukuran setara, satu NCU setara dengan sekitar 2 GB RAM dan vCPU dan jaringan terkait. Jika instans yang Anda sediakan berasal dari `r6i` keluarga, rasionya adalah 1 vCPU per 8 GB RAM, atau 4 NCUs, bersama dengan jaringan terkait. Kalkulator Harga Amazon Neptunus juga menyediakan perbandingan untuk membantu Anda memutuskan konfigurasi biaya optimal Anda.

Saat menggunakan tanpa server untuk instance primer dan replika, ingatlah bahwa replika baca di tingkat promosi 0 dan 1 akan menskalakan sesuai dengan instance penulis sehingga diskalakan dengan benar jika peristiwa failover terjadi. NCUs Tetapkan batas NCU Anda untuk instans ini berdasarkan instans Anda—penulis atau pembaca—yang menerima lalu lintas terbanyak.

Di lingkungan di mana cluster tidak diperlukan 24 jam per hari, 7 hari seminggu, pertimbangkan untuk menulis skrip yang akan mematikan instance Neptunus saat tidak digunakan dan memulainya lagi sebelum digunakan. Instans Neptunus akan dimulai ulang secara otomatis setiap 7 hari untuk memastikan pembaruan pemeliharaan yang diperlukan diterapkan. Jika Anda berniat untuk meninggalkan instance untuk jangka waktu yang lama, gunakan skrip mingguan untuk mematikannya lagi.

## Penyimpanan dan transfer data ukuran kanan

Kueri yang lebih efisien (misalnya, kueri yang perlu menyentuh lebih sedikit node, tepi, dan properti dalam grafik) memerlukan lebih sedikit I/O transfer dan berpotensi dapat menggunakan instance yang lebih kecil karena lebih sedikit cache buffer yang diperlukan. Gunakan profil atau jelaskan titik akhir untuk bahasa kueri Anda untuk mengoptimalkan kueri Anda, dan pertimbangkan untuk mengoptimalkan model grafik Anda untuk kinerja kueri Anda.

Neptunus menggunakan pengkodean kamus pada string besar, dan kamus itu dioptimalkan untuk kinerja, bukan efisiensi. Jika Anda memiliki string besar BLOBs, JSON, atau sering berubah untuk

properti, pertimbangkan untuk menyimpannya di luar Neptunus di Amazon S3, Amazon DynamoDB, atau Amazon DocumentDB, dan simpan hanya referensi dalam node Neptunus.

Dalam beberapa kasus, memilih ukuran instans yang lebih besar bisa lebih murah. Jika I/O biaya Anda sangat tinggi karena rendah`BufferCacheHitRatio`, ada kemungkinan bahwa cache buffer yang lebih besar akan secara signifikan mengurangi biaya itu. Itu karena semua data akan masuk ke dalam cache alih-alih sering ditukar dari penyimpanan dan menimbulkan kecepatan transfer. I/O

Neptunus menggunakan kloning copy-on-write. Saat mengkloning untuk membagi grafik menjadi beberapa pecahan, mungkin lebih efisien untuk tidak menghapus data yang tidak diinginkan pada kloning kloning karena itu akan melibatkan pembuatan halaman data baru, yang mengakibatkan peningkatan biaya penyimpanan. Data yang tidak berubah dari sebelum peristiwa kloning akan ada dalam satu halaman data yang dibagikan di dua cluster dan akan dikenakan biaya hanya untuk salinan tunggal itu.

Jangan aktifkan indeks OSGP atau gunakan instans R5d kecuali Anda telah menguji untuk mengonfirmasi bahwa indeks tersebut membuat perbedaan besar dalam beban kerja Anda. Keduanya dirancang untuk skenario yang jarang terjadi, dan mereka dapat meningkatkan biaya Anda untuk keuntungan minimal atau tanpa keuntungan.

# Pilar keberlanjutan

Pilar keberlanjutan berfokus pada meminimalkan dampak lingkungan dari menjalankan beban kerja cloud. Topik utama mencakup model tanggung jawab bersama untuk keberlanjutan, memahami dampak, dan memaksimalkan penggunaan untuk meminimalkan sumber daya yang dibutuhkan dan mengurangi dampak hilir.

Pilar keberlanjutan berisi area fokus utama berikut:

- Dampak Anda
- Tujuan keberlanjutan
- Penggunaan yang dimaksimalkan
- Mengantisipasi dan mengadopsi penawaran perangkat keras dan perangkat lunak baru yang lebih efisien
- Penggunaan layanan terkelola
- Pengurangan dampak hilir

Panduan ini berfokus pada dampak Anda. Untuk informasi lebih lanjut tentang prinsip desain keberlanjutan lainnya, lihat Kerangka AWS Well-Architected.

Pilihan dan persyaratan Anda berdampak pada lingkungan. Jika Anda dapat memilih Wilayah AWS yang memiliki intensitas karbon lebih rendah, dan jika kebutuhan Anda mencerminkan kebutuhan beban kerja yang sebenarnya, bukan hanya memaksimalkan waktu kerja dan daya tahan, keberlanjutan beban kerja meningkat. Bagian selanjutnya membahas praktik terbaik dan pertimbangan bijaksana yang akan memiliki dampak lingkungan positif jika diadopsi dalam desain beban kerja Anda dan operasi yang sedang berlangsung.

## Wilayah AWS seleksi

Beberapa Wilayah AWS berada di dekat proyek energi terbarukan Amazon atau terletak di mana jaringan memiliki intensitas karbon yang diterbitkan yang lebih rendah daripada yang lain. Pertimbangkan dampak keberlanjutan untuk Wilayah yang mungkin layak untuk beban kerja Anda, dan rujuk silang daftar Anda dengan Wilayah tempat Neptunus tersedia.

# Konsumsi berdasarkan pola perilaku pengguna

Mengukur konsumsi Anda dengan benar agar sesuai dengan lalu lintas dan perilaku pengguna Anda membantu AWS meminimalkan dampak layanan terhadap lingkungan. Pertimbangkan praktik terbaik berikut saat merancang solusi Anda:

- Pantau CloudWatch metrik Amazon sepertiCPUUtilization,MainRequestQueuePendingRequests, dan TotalRequestsPerSec untuk menentukan kapan permintaan Anda tertinggi dan terendah, dan pastikan bahwa sumber daya klaster Anda berukuran tepat selama waktu tersebut.
- Otomatiskan penghentian lingkungan non-produksi selama jam-jam ketika tidak digunakan. Untuk informasi selengkapnya, lihat posting blog Mengotomatiskan penghentian dan memulai sumber daya lingkungan Amazon Neptunus menggunakan tag sumber daya.
- Jika pola lalu lintas Anda sering bervariasi dan tidak terduga, pertimbangkan untuk menggunakan instance Neptunus Tanpa Server yang akan meningkat dan turun dengan permintaan alih-alih menggunakan instance yang disediakan untuk lalu lintas puncak.
- Pertimbangkan untuk menyelaraskan perjanjian tingkat layanan Anda dengan tujuan keberlanjutan selain tujuan kelangsungan bisnis. Persyaratan pelonggaran seperti pemulihan bencana multi-wilayah, ketersediaan tinggi, atau retensi cadangan jangka panjang, terutama untuk lingkungan non-produksi atau beban kerja kritis non-misi, dapat mengurangi jumlah sumber daya yang dibutuhkan untuk memenuhi tujuan tersebut.

# Optimalkan pengembangan perangkat lunak dan pola arsitektur

Untuk mencegah pemborosan, optimalkan model dan kueri, dan bagikan sumber daya komputasi sehingga Anda menggunakan semua sumber daya yang tersedia di instans dan cluster Neptunus. Praktik terbaik khusus meliputi:

- Mintalah pengembang berbagi instance Neptunus dan instance aplikasi Jupyter Notebook alih-alih masing-masing membuat sendiri. Berikan masing-masing pengembang partisi logisnya sendiri dalam satu cluster Neptunus melalui penggunaan strategi partisi multi-tenancy, dan buat folder notebook terpisah untuk setiap pengembang pada satu instance Jupyter.
- Menerapkan pola yang memaksimalkan penggunaan sumber daya dan meminimalkan waktu idle, seperti thread paralel untuk memuat data dan mengumpulkan catatan bersama-sama ke dalam transaksi yang lebih besar.

- Optimalkan kueri dan model grafik Anda untuk meminimalkan sumber daya yang diperlukan untuk menghitung hasil.
- Untuk hasil kueri Gremlin, gunakan fitur cache hasil untuk meminimalkan sumber daya yang dihabiskan untuk menghitung ulang kueri yang dipaginasi atau sering berulang.
- Perbarui lingkungan Neptunus Anda. Versi terbaru Neptunus mendukung instans Amazon EC2 terbaru, seperti Graviton, yang lebih efisien. Mereka juga memiliki peningkatan optimasi kueri dan perbaikan bug yang mengurangi jumlah sumber daya yang dibutuhkan untuk menghitung kueri Anda.

# Sumber daya

## Referensi

- AWS Well-Architected
- AWS Dokumentasi Kerangka Well-Architected
- Neptunus pembaruan terbaru
- Praktik terbaik: mendapatkan hasil maksimal dari Neptunus
- Kalkulator Harga Amazon Neptunus

## Unggahan blog

- Pengujian otomatis akses data Amazon Neptunus dengan Apache Gremlin TinkerPop
- Otomatiskan penghentian dan dimulainya sumber daya lingkungan Amazon Neptunus menggunakan tag sumber daya
- Kontrol Akses Berbutir Halus untuk tindakan bidang data Amazon Neptunus
- Kinerja harga kueri tulis 4,7 kali lebih baik dengan instans AWS Graviton4 R8g menggunakan Amazon Neptunus v1.4.5
- Bagaimana Orca Security mengoptimalkan kinerja database Amazon Neptunus mereka
- Buat aplikasi grafik lebih cepat dengan titik akhir publik Amazon Neptunus
- Versi mesin Amazon Neptune baru memberikan throughput hingga 9 kali lebih cepat dan 10 kali lebih tinggi untuk kinerja kueri OpenCypher

## Kursus Pembuat AWS Keterampilan Gratis

- Memulai dengan Amazon Neptunus
- Membangun Aplikasi di Amazon Neptunus
- Pemodelan Data untuk Amazon Neptunus

# Kontributor

Kontributor untuk panduan ini meliputi:

- Brian O'Keefe, Arsitek Solusi Utama Neptunus, AWS
- Abhishek Mishra, Arsitek Solusi Neptunus Senior, AWS
- Ganesh Sawhney, Ketua Tim - Arsitek Solusi Sukses Mitra Strategis, AWS
- Michael Havey, Arsitek Solusi Neptunus Senior, AWS
- Kevin Phillips, Arsitek Solusi Neptunus, AWS
- Melissa Kwok, Arsitek Solusi Neptunus, AWS
- Sakti Mishra, Arsitek Solusi Utama AWS
- Javed Ali, Arsitek Solusi Senior, AWS

# Riwayat dokumen

Tabel berikut menjelaskan perubahan signifikan pada panduan ini. Jika Anda ingin diberi tahu tentang pembaruan masa depan, Anda dapat berlangganan umpan RSS.

| Perubahan | Deskripsi | Tanggal |
| --- | --- | --- |
| Pembaruan rilis Neptunus | Kami memperbarui dokumentasi untuk menyertakan informasi tentang Amazon Neptunus 1.4.6.0 dan yang lebih baru. | Januari 2, 2026 |
| Publikasi awal | — | 27 September 2023 |

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.