Menyetel parameter Postgre SQL di Amazon dan Amazon RDS Aurora

AWS Bimbingan Preskriptif

AWS Bimbingan Preskriptif: Menyetel parameter Postgre SQL di Amazon dan Amazon RDS Aurora

Table of Contents

Pengantar 1
Menggunakan kelompok parameter cluster DB dan DB 2
Menyetel parameter memori 4
shared_buffers 5
temp_buffers 6
effective_cache_size 8
work_mem 9
maintenance_work_mem 10
random_page_cost 12
seq_page_cost 14
track_activity_query_size 15
idle_in_transaction_session_timeout 16
statement_timeout 17
search_path 19
max_connections 20
Menyetel parameter autovacuum 23
VACUUM dan ANALYSIS perintah 24
Memeriksa kembung 25
autovacuum 26
autovacuum_work_mem 27
autovacuum_naptime 28
autovacuum_max_workers 29
autovacuum_vacuum_scale_factor 30
autovacuum_vacuum_threshold 31
autovacuum_analyze_scale_factor 32
autovacuum_analyze_threshold 33
autovacuum_vacuum_cost_limit 35
Menyetel parameter logging 37
rds.force_autovacuum_logging 38
rds.force_admin_logging_level 39
log_duration 40
log_min_duration_statement 41
log_error_verbosity 42
log_statement 43

log_statement_stats 44
log_min_error_statement 46
log_min_messages 46
log_temp_files 47
log_connections 49
log_disconnections 50
Menggunakan parameter logging untuk menangkap variabel bind 51
Menyetel parameter replikasi 53
Contoh 54
Praktik terbaik 55
Langkah selanjutnya 56
Sumber daya 57
Riwayat dokumen 58
Glosarium 59
..... 59
A 60
B 63
C 65
D 68
E 72
F 74
G 75
H 76
I 77
L 80
M 81
O 85
P 87
Q 90
R 91
D 93
T 97
U 99
V 99
W 100
Z 101

..... cii

Menyetel parameter PostgreSQL di Amazon RDS dan Amazon Aurora

Sumana Yanamandra, Ramu Jagini, dan Rohit Kapoor, Amazon Web Services ()AWS

Februari 2024 (riwayat dokumen)

Amazon Aurora PostgreSQL Compatible Edition dan Amazon Relational Database Service (Amazon RDS) untuk PostgreSQL adalah layanan database relasional open-source canggih yang menawarkan berbagai fitur. Anda dapat menggunakan layanan ini untuk mengatur database PostgreSQL Anda di berbagai platform dan aplikasi.

Aurora dan Amazon RDS menawarkan cara yang disederhanakan untuk mengelola dan mengoperasikan database PostgreSQL Anda. Mereka dirancang untuk mengelola infrastruktur database dan menyediakan ketersediaan tinggi, daya tahan, dan skalabilitas saat Anda fokus pada pengembangan aplikasi. Namun, konfigurasi default layanan ini mungkin tidak optimal untuk semua beban kerja. Secara default, layanan ini dikonfigurasi untuk berjalan di mana saja dengan sumber daya sesedikit mungkin dan tanpa menimbulkan kerentanan. Menyetel parameter dapat membantu Anda mencapai kinerja yang lebih baik, mengurangi waktu henti, dan meningkatkan efisiensi database secara keseluruhan. Dengan mengoptimalkan parameter untuk beban kerja spesifik Anda, Anda dapat memanfaatkan sepenuhnya kemampuan yang disediakan Amazon RDS dan Aurora dan memaksimalkan manfaatnya.

Misalnya, Anda dapat meningkatkan kinerja dengan mengoptimalkan Aurora dan Amazon RDS untuk PostgreSQL dan mengonfigurasi parameternya. Anda juga harus mempertimbangkan kinerja saat membuat kueri basis data. Bahkan ketika Anda mengoptimalkan pengaturan basis data, sistem dapat berkinerja buruk jika kueri Anda melakukan pemindaian tabel lengkap, ketika mereka menggunakan indeks, atau jika mereka menjalankan operasi gabungan atau agregat yang mahal.

Panduan ini untuk pengembang database, insinyur database, dan administrator yang ingin menyetel memori, autovacuum, logging, dan parameter replikasi logis untuk database PostgreSQL mereka. Panduan ini juga mencakup parameter yang khusus untuk Amazon RDS for PostgreSQL dan Aurora PostgreSQL kompatibel. Menyetel parameter ini dapat membantu Anda mengoptimalkan kinerja database dan mengurangi penggunaan sumber daya untuk beban kerja spesifik Anda, menghasilkan kinerja dan penghematan biaya yang lebih baik.

Menggunakan kelompok parameter cluster DB dan DB

Amazon RDS dan Aurora dapat secara otomatis menentukan nilai parameter yang paling sesuai untuk pengaturan tertentu berdasarkan ukuran instans database Anda. Mereka juga mendukung kustomisasi parameter untuk penyetelan kinerja melalui grup parameter untuk instance database dan cluster.

Anda dapat menggunakan grup parameter cluster DB dan DB untuk memodifikasi parameter yang mengontrol berbagai aspek perilaku mesin database, seperti penggunaan memori, I/O disk, jaringan, dan penguncian. Dengan menyesuaikan parameter ini, Anda dapat mengoptimalkan mesin database untuk beban kerja spesifik Anda dan meningkatkan kinerja.

Anda dapat membuat dan mengonfigurasi grup parameter cluster DB dan DB dengan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau Amazon RDS API. Panduan ini mengasumsikan bahwa Anda menggunakan. AWS CLI Untuk petunjuk konsol dan API, lihat Bekerja dengan grup parameter DB dan Bekerja dengan grup parameter cluster DB dalam dokumentasi Amazon RDS.

> **Important**
>
> Untuk menggunakan AWS CLI perintah yang disediakan dalam panduan ini, Anda harus terlebih dahulu menginstal dan mengkonfigurasi AWS CLI.

Untuk membuat dan mengkonfigurasi grup parameter DB:

```
# Create a new DB parameter group
aws rds create-db-parameter-group \
  --db-parameter-group-name mydbparamgroup \
  --db-parameter-group-family postgres13 \
  --description "My DB Parameter Group"

# Modify a parameter on the DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <param group name> \
  --parameters "ParameterName=max_connections<parameter-
name>,ParameterValue=<value>,ApplyMethod=immediate"

# Verify DB parameters
```

```
aws rds describe-db-parameters \
  --db-parameter-group-name aurora-instance-1
```

Untuk membuat dan mengkonfigurasi grup parameter cluster DB:

```
# Create a new DB cluster parameter group
aws rds create-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --db-parameter-group-family postgres12 \
  --description "My new parameter group"

# Modify a parameter on the DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name aws-guide-cluster \
  --parameters "ParameterName=<parameter-name>,ParameterValue=,ApplyMethod=immediate"

# Allocate the new DB cluster parameter to your cluster
aws rds modify-db-cluster  \
  --db-cluster-identifier  \
  --db-cluster-parameter-group-name=-cluster

# Verify cluster parameters
aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name=-cluster
```

> **Note**
>
> Aurora dan Amazon RDS menyediakan grup parameter default dengan nilai yang telah dikonfigurasi sebelumnya yang tidak dapat diubah.

Kelompok parameter dapat diatur sebagai statis atau dinamis. Parameter dinamis diterapkan segera, terlepas dari apakah `ApplyMethod=immediate` opsi diaktifkan. Parameter statis memerlukan reboot manual untuk diterapkan.

Menyetel parameter memori

Menyetel parameter memori adalah tugas penting untuk mengoptimalkan kinerja database yang kompatibel dengan Amazon RDS dan Aurora PostgreSQL. Mengalokasikan memori dengan benar untuk berbagai operasi database seperti menjalankan kueri, penyortiran, pengindeksan, dan caching dapat secara signifikan meningkatkan kinerja database. Bagian ini mencakup beberapa parameter memori paling penting di Amazon RDS dan Aurora PostgreSQL yang kompatibel, termasuk nilai defaultnya, rumus untuk menghitung nilai yang sesuai, dan cara mengubahnya. Untuk daftar lengkap parameter, lihat Bekerja dengan parameter pada instans RDS untuk PostgreSQL DB di dokumentasi Amazon RDS dan parameter Amazon Aurora PostgreSQL dalam dokumentasi Aurora.

Mengoptimalkan parameter ini memerlukan pemahaman mendalam tentang beban kerja database Anda, serta sumber daya yang tersedia di instans Aurora atau Amazon RDS DB Anda. Kinerja sistem dipengaruhi oleh dua kategori parameter yang luas: parameter vital dan parameter kontingen.

Parameter vital adalah parameter yang sangat diperlukan yang memberikan pengaruh signifikan dan langsung pada kinerja sistem dan sangat penting untuk mencapai hasil yang optimal.

- shared_buffer
- temp_buffer
- effective_cache_size
- work_mem
- maintenance_work_mem

Parameter kontingen adalah skenario dan parameter khusus bisnis. Mereka bergantung pada faktor-faktor lain dan memainkan peran tidak langsung namun penting dalam mendukung parameter vital dan memaksimalkan kinerja sistem secara keseluruhan.

- random_page_cost
- seq_page_cost
- track_activity_query_size
- idle_in_transaction_session_timeout
- statement_timeout
- search_path

- max_koneksi

Parameter ini dibahas secara lebih rinci di bagian berikut.

shared_buffers

`shared_buffers`Parameter mengontrol jumlah memori yang PostgreSQL gunakan untuk menyimpan data dalam memori. Menyetel parameter ini ke nilai yang sesuai dapat membantu meningkatkan kinerja kueri.

Untuk Amazon RDS, nilai default untuk `shared_buffers` diatur ke `{DBInstanceClassMemory/32768}` byte, berdasarkan memori yang tersedia untuk instans DB. Untuk Aurora, nilai default diatur ke`{DBInstanceClassMemory/12038,-50003}`, berdasarkan memori yang tersedia untuk instans DB. Nilai optimal untuk parameter ini tergantung pada beberapa faktor, termasuk ukuran database, jumlah koneksi bersamaan, dan memori instance yang tersedia.

AWS CLI sintaks

Perintah berikut berubah `shared_buffers` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify shared_buffers on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=shared_buffers,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify shared_buffers on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters "ParameterName=shared_buffers,ParameterValue=<new-
value>,ApplyMethod=immediate"
```

Jenis: Statis (menerapkan perubahan membutuhkan reboot)

Nilai default: `{DBInstanceClassMemory/32768}` byte di Amazon RDS untuk PostgreSQL, di Aurora `{DBInstanceClassMemory/12038,-50003}` PostgreSQL kompatibel. Dalam kebanyakan kasus, persamaan ini ternyata sekitar 25 persen dari memori sistem Anda. Mengikuti pedoman ini, `shared_buffers` pengaturan dalam grup parameter diatur dengan menggunakan unit default PostgreSQL dari buffer 8K alih-alih byte atau kilobyte.

Pengaturan `shared_buffers` parameter dapat berdampak signifikan pada kinerja, jadi sebaiknya Anda menguji perubahan secara menyeluruh untuk memastikan bahwa nilainya sesuai dengan beban kerja Anda.

Contoh

Katakanlah Anda memiliki aplikasi layanan keuangan yang menjalankan database PostgreSQL di Amazon RDS atau Aurora. Database ini digunakan untuk menyimpan data transaksi pelanggan. Ini memiliki sejumlah besar tabel dan diakses oleh beberapa aplikasi pada sejumlah besar server. Aplikasi ini mengalami kinerja kueri yang lambat dan penggunaan CPU yang tinggi. Anda menentukan bahwa menyetel `shared_buffers` parameter dapat membantu meningkatkan kinerja.

Di Amazon RDS untuk PostgreSQL, nilai `shared_buffers` default diatur ke byte memori yang `{DBInstanceClassMemory/32768}` `db.r5.xlarge` tersedia di (misalnya, 3 GB). Untuk menentukan nilai yang sesuai`shared_buffers`, Anda menjalankan serangkaian pengujian dengan nilai yang bervariasi`shared_buffers`, dimulai dengan nilai default memori yang tersedia dan meningkatkan nilai secara bertahap. Untuk setiap pengujian, Anda mengukur kinerja kueri dan penggunaan CPU database.

Berdasarkan hasil pengujian, Anda menentukan bahwa menyetel nilai `shared_buffers` hingga 8 GB menghasilkan kinerja kueri keseluruhan terbaik dan penggunaan CPU untuk beban kerja Anda. Nilai ditentukan melalui kombinasi pengujian dan analisis karakteristik beban kerja, termasuk ukuran database, jumlah dan kompleksitas kueri, jumlah pengguna bersamaan, dan sumber daya sistem yang tersedia. Setelah Anda melakukan perubahan, sistem pemantauan Anda memeriksa kinerja database untuk memastikan bahwa nilai baru sesuai dengan beban kerja Anda. Anda kemudian dapat menyempurnakan parameter tambahan yang diperlukan untuk lebih meningkatkan kinerja.

temp_buffers

`temp_buffers`adalah parameter konfigurasi utama di Aurora PostgreSQL yang kompatibel dan Amazon RDS for PostgreSQL yang dapat secara signifikan memengaruhi kinerja untuk beban kerja yang melibatkan pengurutan, hash, dan operasi agregat pada tabel sementara. Parameter ini menentukan jumlah memori yang dialokasikan untuk buffer sementara, yang, pada gilirannya, mempengaruhi efisiensi dan kecepatan operasi tersebut. Jika tidak ada cukup memori yang dialokasikan`temp_buffers`, sistem mungkin harus menggunakan metode yang lebih lambat dan kurang efisien untuk pengurutan, hash, dan operasi agregasi pada tabel sementara, yang mengarah ke kinerja kurang optimal.

AWS CLI sintaks

Perintah berikut berubah `temp_buffers` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify temp_buffers on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=temp_buffers,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify temp_buffers on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=temp_buffers,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: 8 MB

Untuk informasi selengkapnya tentang parameter ini, lihat Konsumsi Sumber Daya dalam dokumentasi PostgreSQL.

Contoh

Jika beban kerja Anda melibatkan banyak pengurutan, hashing, dan operasi agregat pada tabel sementara, `temp_buffers` mungkin tidak mengalokasikan cukup memori. Dalam hal ini, sistem mungkin harus melakukan operasi pengurutan pada tabel sementara yang mengarah ke metode berbasis disk yang lebih lambat, bukan pengurutan dalam memori, hashing, dan operasi agregat. Hal ini dapat menyebabkan perlambatan kinerja kueri yang signifikan, terutama untuk kueri yang melibatkan kumpulan data besar. Meningkatkan nilai `temp_buffers` dapat memastikan bahwa ada cukup memori yang tersedia untuk melakukan operasi tersebut dalam memori, yang mengarah pada peningkatan kinerja yang signifikan.

Untuk menemukan nilai optimal`temp_buffers`, pantau kinerja sistem Anda dan identifikasi area kinerja suboptimal. Jika Anda melihat waktu respons kueri yang lambat atau pemanfaatan CPU yang tinggi, pertimbangkan untuk menyesuaikan`temp_buffers`. Misalnya, jika beban kerja Anda melibatkan banyak tabel sementara, meningkatkan nilai `temp_buffers` dapat membantu memastikan bahwa tabel ini disimpan dalam memori. Ini bisa jauh lebih cepat daripada menggunakan baca/tulis I/O dari penyimpanan.

Bereksperimenlah dengan nilai yang berbeda `temp_buffers` dalam peningkatan kecil dan pantau kinerja sistem dengan cermat setelah setiap perubahan. Menganalisis dampak nilai yang berbeda pada kinerja dan menyempurnakan pengaturan berdasarkan karakteristik spesifik dari beban kerja Anda.

effective_cache_size

`effective_cache_size`Parameter menentukan jumlah memori yang PostgreSQL harus mengasumsikan tersedia untuk data caching. Mengatur parameter ini dengan benar dapat meningkatkan kinerja dengan memungkinkan PostgreSQL memanfaatkan memori yang tersedia dengan lebih baik.

AWS CLI sintaks

Perintah berikut berubah `effective_cache_size` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify effective_cache_size on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=effective_cache_size,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify effective_cache_size on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=effective_cache_size,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: `SUM(DBInstanceClassMemory/12038,-50003)` KB

Contoh

Platform pembelajaran online memiliki database besar materi kursus, data siswa, dan konten lainnya yang sering diakses oleh pengguna. Aplikasi ini berjalan pada instans Amazon `db.r5.xlarge` RDS for PostgreSQL dengan memori 32 GB. Aplikasi mengalami kinerja yang lambat ketika pengguna mencoba membaca konten yang sering diakses. Setelah menganalisis penggunaan sumber daya

server database, Anda menentukan bahwa PostgreSQL tidak memanfaatkan memori yang tersedia secara optimal.

`effective_cache_size`Parameter di Amazon RDS untuk PostgreSQL mengontrol jumlah memori yang digunakan oleh server untuk disk caching. Nilai default diatur ke `SUM({DBInstanceClassMemory/12038},-50003)` KB untuk kelas contoh`db.r5.xlarge`, tetapi default ini mungkin tidak sesuai untuk semua beban kerja. Dalam contoh ini, server database mungkin menyimpan sejumlah besar materi kursus dan data siswa yang sering diakses. Meningkatkan nilai `effective_cache_size` parameter dapat menghasilkan lebih banyak data yang di-cache dalam memori, yang mengurangi jumlah pembacaan disk yang diperlukan dan meningkatkan kinerja kueri.

Saat Anda menjalankan kueri, Amazon RDS for PostgreSQL terlebih dahulu memeriksa apakah data yang diperlukan oleh kueri sudah ada dalam cache. Jika demikian, data dapat dibaca dari memori alih-alih dibaca dari disk. Jika data tidak ada dalam cache, itu harus dibaca dari disk, yang bisa menjadi operasi yang lambat.

Untuk platform pembelajaran online, Anda mungkin memutuskan untuk mengatur `effective_cache_size` ke 16 GB (setengah dari memori yang tersedia) setelah pengujian dan analisis. Nilai ini memungkinkan PostgreSQL memanfaatkan memori yang tersedia dengan lebih baik, yang mengurangi jumlah pembacaan disk yang diperlukan dan meningkatkan kinerja kueri.

work_mem

`work_mem`Parameter mengontrol jumlah memori yang digunakan oleh kueri untuk operasi penyortiran dan hashing. Nilai defaultnya adalah 4 MB. Jika kueri mencakup beberapa operasi, kueri dapat menggunakan hingga 4 MB untuk setiap operasi. Meningkatkan nilai `work_mem` dapat meningkatkan kinerja query yang memerlukan penyortiran atau hashing, karena operasi ini membutuhkan lebih banyak memori. Namun, pengaturan parameter ini terlalu tinggi dapat menyebabkan penggunaan memori yang berlebihan, yang menyebabkan penurunan kinerja.

Untuk menghitung nilai optimal`work_mem`, Anda dapat menggunakan rumus berikut:

```
work_mem = (available_memory / (max_connections * work_mem_fraction))
```

di mana `available_memory` jumlah total memori yang tersedia di server, max_connections adalah jumlah maksimum koneksi yang diizinkan, dan work_mem_fraction adalah pecahan yang menentukan berapa banyak memori yang tersedia harus dialokasikan untuk setiap koneksi.

AWS CLI sintaks

Perintah berikut berubah `work_mem` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify work_mem on a DB parameter group
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify work_mem on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
   --db-cluster-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: 4 MB

Contoh

Alat analisis media sosial memproses sejumlah besar data, dan kueri yang melibatkan operasi pengurutan dan gabungan yang kompleks menyebabkan I/O disk tinggi dan tumpah ke disk. Jika Anda meningkatkan nilai `work_mem` dari 4 MB menjadi 16 MB, PostgreSQL dapat menggunakan lebih banyak memori untuk operasi ini. Ini mengurangi jumlah I/O dan meningkatkan kinerja kueri.

maintenance_work_mem

`maintenance_work_mem`Parameter mengontrol jumlah memori yang digunakan oleh operasi pemeliharaan seperti`VACUUM`,`ANALYZE`, dan pembuatan indeks. Nilai default untuk parameter ini di Amazon RDS dan Aurora adalah 64 MB.

Untuk menghitung nilai yang sesuai untuk parameter ini, Anda dapat menggunakan rumus ini:

```
maintenance_work_mem = (total_memory - shared_buffers) / (max_connections * 5)
```

Aurora PostgreSQL Compatible Edition dan Amazon RDS for PostgreSQL menerapkan rumus berikut untuk menetapkan nilai optimal:

```
GREATEST({DBInstanceClassMemory/63963136*1024},65536)
```

AWS CLI sintaks

Perintah berikut berubah `maintenance_work_mem` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify maintenance_work_mem on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=maintenance_work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify maintenance_work_mem on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=maintenance_work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: 64 MB

Contoh

Aplikasi skala besar Anda menggunakan database PostgreSQL yang di-host di Aurora atau Amazon RDS. Anda melihat bahwa database lambat dan tidak responsif selama kegiatan pemeliharaan seperti menyedot debu dan pengindeksan. Anda dapat memantau metrik seperti penggunaan memori, waktu operasi pemeliharaan, dan penggunaan CPU untuk menentukan apakah nilai saat `maintenance_work_mem` ini menyebabkan masalah.

Untuk menentukan nilai optimal`maintenance_work_mem`, Anda dapat menyesuaikan parameter dan memantau dampaknya. Jika penggunaan memori secara konsisten tinggi atau waktu operasi lebih lama dari yang diharapkan selama operasi pemeliharaan, peningkatan `maintenance_work_mem` mungkin membantu. Sebaliknya, jika penggunaan CPU secara konsisten tinggi selama operasi pemeliharaan, penurunan mungkin membantu. `maintenance_work_mem` Dengan iterasi melalui penyesuaian dan pengujian, Anda dapat menemukan nilai optimal untuk `maintenance_work_mem` yang memberikan keseimbangan terbaik untuk penggunaan memori, waktu operasi pemeliharaan, dan penggunaan CPU.

Selama penyelidikan Anda, katakanlah Anda menentukan bahwa nilai default 64 MB untuk `maintenance_work_mem` terlalu rendah untuk ukuran database Anda. Akibatnya, operasi pemeliharaan membutuhkan waktu lebih lama untuk diselesaikan, menyebabkan downtime yang berlebihan, dan memperlambat kinerja aplikasi Anda. Untuk mengatasi masalah ini, Anda dapat memutuskan untuk menyetel `maintenance_work_mem` parameter dengan meningkatkannya dari 64 MB menjadi 512 MB (yang Anda identifikasi sebagai nilai optimal). Menerapkan perubahan dapat meningkatkan waktu operasi pemeliharaan Anda hingga dua pertiga. Misalnya, operasi vakum yang sebelumnya membutuhkan waktu 30 menit untuk diselesaikan sekarang mungkin hanya membutuhkan waktu 10 menit. Sebagai hasil dari pengoptimalan ini, database Anda sekarang dapat menangani aktivitas pemeliharaan dengan lebih efisien.

random_page_cost

`random_page_cost`Parameter membantu menentukan biaya melakukan akses halaman acak. Perencana kueri di Amazon RDS dan Aurora menggunakan parameter ini, bersama dengan statistik lain tentang tabel, untuk menentukan rencana paling efisien untuk menjalankan kueri.

`random_page_cost`Parameter `seq_page_cost` dan parameter terkait erat dan biasanya digunakan bersama oleh perencana untuk membandingkan biaya metode akses yang berbeda dan memutuskan mana yang paling efisien. Oleh karena itu, jika Anda mengubah salah satu parameter ini, Anda juga harus mempertimbangkan apakah parameter lain perlu disesuaikan.

Secara umum, perencana kueri mencoba meminimalkan biaya menjalankan kueri. Biaya ditentukan dengan menggunakan kombinasi jumlah pembacaan halaman disk dan nilai`random_page_cost`. Nilai yang lebih tinggi `random_page_cost` cenderung mendukung pemindaian berurutan sedangkan nilai yang lebih rendah cenderung mendukung pemindaian indeks. Nilai yang lebih rendah juga cenderung mendukung gabungan loop bersarang alih-alih bergabung dengan hash.

`random_page_cost`Parameter menggunakan nilai mesin PostgreSQL default (4) kecuali nilai ditetapkan dalam kelompok parameter atau dalam sesi lokal. Anda dapat menyetel nilai ini tergantung pada karakteristik spesifik server dan beban kerja Anda. Jika sebagian besar indeks yang digunakan dalam beban kerja Anda sesuai dengan memori atau di cache berjenjang Aurora, mengubah nilai menjadi nilai yang `random_page_cost` mendekati adalah tepat. `seq_page_cost`

AWS CLI sintaks

Perintah berikut berubah `random_page_cost` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify random_page_cost on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=random_page_cost,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify random_page_cost on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=random_page_cost,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengaturApplyMethod=immediate)

Nilai default: 4

Contoh

Katakanlah Anda memiliki database yang menyimpan sejumlah besar data dalam tabel yang sering ditanyakan dengan filter pada kolom yang tidak diindeks. Kueri membutuhkan waktu lama untuk diselesaikan, dan perencana kueri tidak memilih paket yang paling efisien untuk mengakses data.

Salah satu cara untuk meningkatkan kinerja adalah dengan mengurangi `random_page_cost` parameter. Jika Anda mengaturnya ke 1, biaya akses halaman acak akan empat kali lebih murah daripada nilai default. Jika Anda meninggalkan `random_page_cost` nilai default 4, akses halaman acak akan empat kali lebih mahal daripada akses halaman berurutan (seperti yang ditentukan oleh `seq_page_cost` parameter, yaitu 1.0 secara default). Namun, dalam kasus khusus ini, akses halaman acak mungkin sebenarnya jauh lebih mahal tergantung pada jenis penyimpanan.

Mengurangi nilai `random_page_cost` parameter dapat membuat perencana kueri lebih mungkin untuk memilih rencana berbasis indeks atau menggunakan metode akses berbeda yang lebih sesuai dengan karakteristik spesifik tabel.

Kami menyarankan Anda memantau kinerja kueri setelah Anda mengubah parameter dan membuat penyesuaian sesuai kebutuhan. Anda juga harus memeriksa perencana kueri dengan `EXPLAIN` pernyataan untuk memeriksa apakah itu memilih rencana yang efisien.

Ini hanya satu contoh. Pengaturan optimal tergantung pada karakteristik spesifik dari beban kerja Anda. Selain itu, ini hanyalah salah satu aspek penyetelan kinerja; Anda juga harus mempertimbangkan parameter dan opsi konfigurasi lain yang dapat memengaruhi kinerja kueri Anda.

seq_page_cost

`seq_page_cost`Parameter membantu menentukan biaya melakukan akses halaman berurutan. Perencana kueri di Amazon RDS dan Aurora menggunakan parameter ini, bersama dengan statistik lain tentang tabel, untuk menentukan rencana paling efisien untuk menjalankan kueri.

`random_page_cost` Parameter `seq_page_cost` dan parameter terkait erat dan biasanya digunakan bersama oleh perencana untuk membandingkan biaya metode akses yang berbeda dan memutuskan mana yang paling efisien. Oleh karena itu, jika Anda mengubah salah satu parameter ini, Anda juga harus mempertimbangkan apakah parameter lain perlu disesuaikan.

Ketika sebuah tabel diakses secara berurutan, PostgreSQL dapat menggunakan cache sistem file sistem operasi untuk menyediakan akses yang lebih cepat. Secara default, `seq_page_cost` diatur ke 1.0, yang mengasumsikan bahwa akses halaman berurutan semurah satu blok disk yang dibaca. Jika Anda memiliki tabel yang terutama diakses secara berurutan tetapi akses disk lambat karena dibatasi oleh IOPS yang lebih sedikit, Anda mungkin ingin meningkatkan nilai `seq_page_cost` untuk mencerminkan biaya tambahan untuk mengakses disk.

Mengubah nilai parameter ini memengaruhi semua kueri yang berjalan di sistem, jadi sebaiknya Anda menguji kueri dengan nilai yang berbeda untuk menentukan nilai optimal untuk kasus penggunaan spesifik Anda.

AWS CLI sintaks

Perintah berikut berubah `seq_page_cost` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify seq_page_cost on a DB parameter group
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=seq_page_cost,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify seq_page_cost on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
   --db-cluster-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=seq_page_cost,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: 1.0

Contoh

Katakanlah Anda memiliki database yang menyimpan sejumlah besar data dalam tabel yang terutama diakses secara berurutan. Tabel ini terutama digunakan untuk pelaporan, kueri bekerja sangat lambat, dan perencana kueri tidak dapat memilih paket yang paling efisien untuk mengakses data.

Salah satu cara untuk meningkatkan kinerja adalah dengan mengurangi `seq_page_cost` parameter. Nilai defaultnya adalah 1.0, yang mengasumsikan bahwa akses halaman berurutan semurah satu blok disk yang dibaca. Namun, dalam kasus khusus ini, akses halaman berurutan mungkin sebenarnya lebih mahal daripada itu karena jenis penyimpanan (tergantung pada I/O). Jika Anda mengatur `seq_page_cost` ke 0,5, biaya akses halaman berurutan setengah mahal dari nilai default. Perubahan ini dapat membuat perencana kueri lebih mungkin untuk memilih paket yang menggunakan metode akses sekuensial yang lebih sesuai dengan karakteristik spesifik tabel.

Kami menyarankan Anda memantau kinerja kueri setelah Anda mengubah parameter dan membuat penyesuaian sesuai kebutuhan. Anda juga harus memeriksa rencana kueri dengan `EXPLAIN` pernyataan untuk memeriksa apakah itu memilih rencana yang efisien.

Ini hanya satu contoh. Pengaturan optimal tergantung pada karakteristik spesifik dari beban kerja Anda. Selain itu, ini hanyalah salah satu aspek penyetelan kinerja; Anda juga harus mempertimbangkan parameter dan opsi konfigurasi lain yang memengaruhi kinerja kueri Anda.

track_activity_query_size

`track_activity_query_size`Parameter mengontrol ukuran string kueri yang dicatat untuk setiap sesi aktif dalam `pg_stat_activity` tampilan. Secara default, hanya 1.024 byte pertama dari string kueri yang dicatat di Amazon RDS for PostgreSQL, dan 4.096 byte dicatat di Aurora PostgreSQL kompatibel. Jika Anda ingin mencatat kueri yang lebih panjang, Anda dapat mengatur parameter ini ke nilai yang lebih tinggi.

AWS CLI sintaks

Perintah berikut berubah `track_activity_query_size` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify track_activity_query_size on a DB parameter group
aws rds modify-db-parameter-group \
```

```
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=track_activity_query_size,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify track_activity_query_size on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=track_activity_query_size,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Statis (menerapkan perubahan membutuhkan reboot)

Nilai default: 1.024 byte (Amazon RDS untuk PostgreSQL), 4.096 byte (Aurora PostgreSQL kompatibel)

Contoh

Database Amazon RDS for PostgreSQL Anda mengalami kinerja kueri yang lambat, dan Anda menduga bahwa masalah tersebut mungkin terkait dengan kueri yang berjalan lama. Anda dapat menyelidiki lebih lanjut dengan mencatat kueri yang lebih panjang ke `pg_stat_activity` tampilan.

Meningkatkan nilai `track_activity_query_size` parameter dapat mengakibatkan peningkatan logging, yang dapat berdampak pada kinerja pada database. Kami menyarankan Anda menyetel parameter kembali ke nilai default 1.024 setelah masalah diselesaikan.

idle_in_transaction_session_timeout

`idle_in_transaction_session_timeout`Parameter mengontrol jumlah waktu transaksi idle menunggu sebelum dihentikan.

Nilai default untuk parameter ini di Amazon RDS untuk PostgreSQL dan Aurora PostgreSQL kompatibel adalah 86.400.000 milidetik.

AWS CLI sintaks

Perintah berikut berubah `idle_in_transaction_session_timeout` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify idle_in_transaction_session_timeout on a DB parameter group
aws rds modify-db-parameter-group \
```

```
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=idle_in_transaction_session_timeout,ParameterValue=<new_value>,ApplyMethod=imme

# Modify idle_in_transaction_session_timeout on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
   --db-cluster-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=idle_in_transaction_session_timeout,ParameterValue=<new_value>,ApplyMethod=imme
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengaturApplyMethod=immediate)

Nilai default: 86.400.000 milidetik (Aurora PostgreSQL kompatibel)

Contoh

Anda memiliki aplikasi e-commerce yang memproses pesanan online. Aplikasi ini menggunakan database PostgreSQL yang di-host di Amazon RDS atau Aurora. Setiap kali pelanggan melakukan pemesanan, aplikasi memulai transaksi baru untuk memperbarui inventaris dan catatan pesanan.

Jika transaksi dibiarkan menganggur untuk waktu yang lama, itu dapat mencegah transaksi lain mengakses catatan yang sama, yang menyebabkan masalah kinerja dan bahkan berpotensi downtime aplikasi. Selain itu, transaksi idle yang tidak dihentikan dengan benar dapat mengkonsumsi sumber daya sistem yang berharga seperti memori dan CPU.

Untuk menghindari masalah seperti itu, Anda dapat mengatur `idle_in_transaction_session_timeout` parameter ke nilai yang masuk akal untuk aplikasi Anda. Misalnya, Anda dapat mengaturnya menjadi 5 menit (300 detik) sehingga setiap transaksi yang dibiarkan menganggur selama lebih dari 5 menit akan otomatis dihentikan. Ini dapat membantu memastikan bahwa sumber daya sistem digunakan secara efisien dan bahwa aplikasi dapat menangani sejumlah besar pesanan tanpa melambat. Dengan menetapkan nilai yang sesuai`idle_in_transaction_session_timeout`, Anda dapat membantu memastikan bahwa aplikasi Anda berkinerja optimal di Amazon RDS atau Aurora.

statement_timeout

`statement_timeout`Parameter menetapkan jumlah maksimum waktu kueri dapat dijalankan sebelum dihentikan.

AWS CLI sintaks

Perintah berikut berubah `statement_timeout` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify statement_timeout on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=statement_timeout,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify statement_timeout on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=statement_timeout,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: 0 milidetik (tidak ada batas waktu)

Contoh

Aplikasi web Anda memungkinkan pengguna untuk mencari melalui database produk yang besar. Kueri penelusuran terkadang membutuhkan waktu lama untuk diselesaikan, menyebabkan waktu respons yang lambat bagi pengguna. Untuk mengatasi masalah ini, Anda dapat mengatur `statement_timeout` parameter ke nilai rendah seperti 10 detik, yang akan memaksa kueri apa pun yang membutuhkan waktu lebih dari 10 detik untuk dihentikan.

Ini mungkin tampak seperti ukuran drastis, tetapi sebenarnya bisa sangat efektif dalam meningkatkan kinerja. Dalam banyak kasus, kueri yang berjalan lama disebabkan oleh kueri SQL yang dioptimalkan dengan buruk atau indeks yang tidak efisien. Dengan menetapkan `statement_timeout` nilai rendah, Anda dapat mengidentifikasi kueri bermasalah ini dan mengambil langkah-langkah untuk mengoptimalkannya.

Misalnya, Anda menemukan bahwa kueri penelusuran tertentu secara konsisten habis waktu. Anda dapat menggunakan alat seperti `EXPLAIN` dan `EXPLAIN ANALYZE` untuk menganalisis kueri dan mengidentifikasi hambatan kinerja apa pun. Setelah mengidentifikasi masalah, Anda dapat mengambil langkah-langkah untuk mengoptimalkan kueri dengan menambahkan indeks baru, menulis ulang kueri, atau menggunakan algoritme penelusuran yang berbeda. Dengan terus menganalisis dan mengoptimalkan kueri SQL Anda dengan cara ini, Anda dapat secara signifikan meningkatkan kinerja aplikasi Anda.

search_path

`search_path` Parameter menentukan urutan skema yang dicari untuk objek dalam pernyataan SQL. Nilai defaultnya adalah`$user, public`, yang berarti bahwa PostgreSQL mencari objek terlebih dahulu dalam skema yang cocok dengan nama pengguna, dan kemudian di skema publik.

Jika Anda memiliki sejumlah besar skema atau Anda perlu mengakses objek dalam skema tertentu, mengubah `search_path` parameter dapat membantu meningkatkan kinerja. Ketika Anda mengatur `search_path` ke skema tertentu, PostgreSQL dapat menemukan objek lebih cepat tanpa harus mencari melalui beberapa skema.

Untuk mengubah `search_path` parameter di Amazon RDS dan Aurora, Anda dapat menggunakan perintah `ROLE` berikut untuk level:

```
ALTER ROLE <username> SET search_path = <schema>;
```

AWS CLI sintaks

Perintah berikut berubah `search_path` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify search_path on a DB parameter group
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=search_path,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify search_path on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
   --db-cluster-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=search_path,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: `$user, public`

Contoh

Anda memiliki aplikasi multi-tenant dengan skema terpisah untuk setiap penyewa yang menggunakan Amazon RDS for PostgreSQL atau database yang kompatibel dengan Aurora

PostgreSQL, dan Anda perlu menjalankan kueri yang melibatkan penggabungan data dari beberapa skema.

Secara default, Amazon RDS dan Aurora menggunakan jalur pencarian untuk menentukan skema mana yang akan digunakan untuk tabel tertentu. Jalur pencarian adalah daftar nama skema yang ditelusuri PostgreSQL secara berurutan saat Anda merujuk ke tabel tanpa memenuhi syarat nama skema. Secara default, Amazon RDS dan Aurora pertama-tama mencari tabel dalam skema yang memiliki nama yang sama dengan pengguna saat ini, dan kemudian melihat di skema publik.

Katakanlah Anda ingin menjalankan kueri yang melibatkan menggabungkan tabel dari beberapa skema, bernama `tenant1tenant2`, dan`tenant3`. Untuk menggunakan skema penyewa, Anda dapat menyertakan nama skema dalam kueri:

```
SELECT *
FROM tenant1.table1
JOIN tenant2.table2 ON tenant1.table1.id = tenant2.table2.id
JOIN tenant3.table3 ON tenant2.table2.id = tenant3.table3.id;
```

Namun, metode yang lebih efisien adalah mengubah `search_path` parameter untuk memasukkan skema penyewa dengan menggunakan perintah di bagian AWS CLI sintaks. Anda juga dapat menggunakan `SET` perintah dalam sesi PostgreSQL:

```
SET search_path = tenant1, tenant2, tenant3, public;
```

Anda kemudian dapat menulis kueri tanpa memenuhi syarat nama skema:

```
SELECT *
FROM table1
JOIN table2 ON table1.id = table2.id
JOIN table3 ON table2.id = table3.id;
```

Ini dapat membuat kueri Anda lebih ringkas dan mudah dibaca, dan juga dapat menyederhanakan kode aplikasi Anda jika Anda memiliki banyak kueri yang melibatkan menggabungkan tabel dari beberapa skema.

max_connections

`max_connections`Parameter menetapkan jumlah maksimum koneksi bersamaan untuk database PostgreSQL Anda.

AWS CLI sintaks

Perintah berikut berubah `max_connections` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify max_connections on a DB parameter group
aws rds modify-db-parameter-group \
--db-parameter-group-name <parameter_group_name> \
--parameters
 "ParameterName=max_connections,ParameterValue=<new_value>,ApplyMethod=pending-reboot"

# Modify max_connections on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name <parameter_group_name> \
--parameters
 "ParameterName=max_connections,ParameterValue=<new_value>,ApplyMethod=pending-reboot"
```

Jenis: Statis (menerapkan perubahan membutuhkan reboot)

Nilai default: `LEAST(DBInstanceClassMemory/9531392,5000)` koneksi

Untuk mengoptimalkan penggunaan `max_connections` di Amazon RDS atau Aurora dan meminimalkan dampaknya terhadap kinerja, pertimbangkan praktik terbaik berikut:

- Tetapkan nilai parameter berdasarkan sumber daya sistem yang tersedia.
- Pantau penggunaan koneksi untuk mencegah mencapai batas dengan cepat.
- Gunakan penyatuan koneksi untuk mengurangi jumlah koneksi yang diperlukan.
- Gunakan Amazon RDS Proxy untuk penyatuan koneksi.

Saat Anda menyetel `max_connections` Amazon RDS untuk PostgreSQL atau yang kompatibel dengan Amazon Aurora PostgreSQL, pertimbangkan jenis instans yang tersedia dan sumber daya yang dialokasikan, serta fokus pada memori dan kapasitas CPU. Detail penyimpanan dan I/O dikelola oleh AWS, sehingga Anda dapat memantau karakteristik beban kerja umum dan metrik sistem seperti melalui `FreeableMemory` Amazon atau konsol Amazon CloudWatch RDS untuk mengonfirmasi bahwa ada cukup memori untuk koneksi. Pantau `CPUUtilization` nilai tinggi, yang mungkin menunjukkan bahwa penyesuaian diperlukan. Hindari pengaturan `max_connections` terlalu tinggi, karena dapat mempengaruhi penggunaan memori dan berpotensi mempengaruhi I/O secara tidak langsung. Perlu diingat bahwa setiap koneksi menghabiskan memori. Untuk menemukan keseimbangan yang tepat, perlahan-lahan tingkatkan `max_connections` dan lihat

bagaimana pengaruhnya terhadap sistem Anda. Perhatikan tanda-tanda kinerja yang lebih lambat atau penggunaan CPU yang lebih tinggi. Periksa apakah aplikasi Anda masih berfungsi dengan baik. Gunakan fitur seperti replika baca di Aurora untuk mendistribusikan lalu lintas baca dan mengurangi beban pada instance utama. Tinjau dan sesuaikan `max_connections` secara teratur berdasarkan pola penggunaan yang diamati untuk memastikan kinerja database yang optimal dalam batasan sumber daya yang diberikan.

Menyetel parameter autovacuum

Amazon RDS untuk database PostgreSQL dan Aurora PostgreSQL yang kompatibel dengan Aurora memerlukan perawatan berkala yang dikenal sebagai penyedot debu. Autovacuum adalah utilitas PostgreSQL bawaan yang menghapus data usang atau tidak perlu untuk membebaskan ruang dalam database. Proses autovacuum menjalankan `VACUUM` perintah di latar belakang secara berkala.

Menyetel pengaturan autovacuum adalah langkah penting dalam menjaga kinerja, stabilitas, dan ketersediaan sistem database yang kompatibel dengan Amazon RDS for PostgreSQL atau Aurora PostgreSQL. Dengan menyesuaikan parameter autovacuum agar sesuai dengan beban kerja dan ukuran database Anda, Anda dapat mengoptimalkan kinerja proses autovacuum dan mengurangi dampaknya pada sumber daya sistem, sehingga meningkatkan kesehatan database Anda secara keseluruhan.

Selain menyesuaikan pengaturan autovacuum, penting untuk memantau kinerja database Anda dan komponennya dengan menggunakan alat dan metrik yang tersedia di Amazon RDS dan Aurora. Dengan memantau metrik kinerja seperti bloat, ruang bebas, dan waktu proses kueri, Anda dapat mengidentifikasi potensi masalah sebelum menjadi masalah serius, dan mengambil tindakan yang tepat untuk menyelesaikannya.

Bagian ini membahas topik dan parameter autovacuum berikut:

- VACUUM dan ANALYSIS perintah
- Memeriksa kembung
- autovacuum
- autovacuum_work_mem
- autovacuum_naptime
- autovacuum_max_workers
- autovacuum_vacuum_scale_factor
- autovacuum_vacuum_threshold
- autovacuum_analyze_scale_factor
- autovacuum_analyze_threshold
- autovacuum_vacuum_cost_limit

Untuk informasi tambahan tentang autovacuum, lihat tautan berikut:

- Memahami autovacuum di Amazon RDS untuk lingkungan PostgreSQL (posting blog)
- Bekerja dengan autovacuum PostgreSQL di Amazon RDS untuk PostgreSQL (dokumentasi Amazon RDS)
- Penyedot debu paralel di Amazon RDS untuk PostgreSQL dan Amazon Aurora PostgreSQL (posting blog)

VACUUM dan ANALYSIS perintah

`VACUUM`mengumpulkan sampah dan secara opsional menganalisis database. Untuk sebagian besar aplikasi, cukup membiarkan daemon autovacuum melakukan penyedotan debu. Namun, beberapa administrator mungkin ingin memodifikasi parameter database untuk autovacuum, atau melengkapi atau mengganti aktivitas daemon dengan menggunakan `VACUUM` perintah yang dikelola secara manual yang dapat dijalankan sesuai dengan penjadwal.

`VACUUM`mengklaim kembali penyimpanan yang ditempati oleh tupel mati. Dalam operasi PostgreSQL standar, ketika tupel dihapus atau dibuat usang oleh pembaruan, tupel tersebut tidak dihapus secara fisik dari tabel sampai operasi dilakukan. `VACUUM` Oleh karena itu, kami menyarankan Anda menjalankan `VACUUM` secara berkala, terutama pada tabel yang sering diperbarui.

`VACUUM`Parameter penyetelan sangat penting di Amazon RDS for PostgreSQL dan Aurora PostgreSQL yang kompatibel, karena layanan database terkelola ini memiliki karakteristik yang berbeda dibandingkan dengan database PostgreSQL yang dikelola sendiri. Perbedaan-perbedaan ini dapat mempengaruhi kinerja operasi vakum. `VACUUM`Parameter penyetelan sangat penting untuk mengoptimalkan penggunaan sumber daya dan memastikan bahwa operasi vakum tidak berdampak negatif pada kinerja dan ketersediaan sistem basis data Anda.

Berikut adalah beberapa parameter yang dapat Anda gunakan dengan `VACUUM` perintah di Aurora PostgreSQL kompatibel dan Amazon RDS for PostgreSQL:

- `FULL`
- `FREEZE`
- `VERBOSE`
- `ANALYZE`
- `DISABLE_PAGE_SKIPPING`
- `table_name`

- `column_name`

`VACUUM ANALYZE`melakukan `VACUUM` operasi diikuti oleh `ANALYZE` operasi untuk setiap tabel yang dipilih. Ini memberikan cara yang efisien untuk melakukan perawatan rutin.

Menggunakan `VACUUM` perintah tanpa `FULL` opsi mengklaim kembali ruang untuk digunakan kembali. Itu tidak memerlukan kunci eksklusif di atas meja, sehingga Anda dapat menjalankan perintah ini selama operasi membaca dan menulis standar. Namun, dalam kebanyakan kasus, perintah tidak mengembalikan ruang ekstra ke sistem operasi tetapi membuatnya tersedia untuk digunakan kembali dalam tabel yang sama. `VACUUM FULL`menulis ulang seluruh isi tabel ke dalam file disk baru tanpa ruang ekstra, dan memungkinkan ruang yang tidak terpakai dikembalikan ke sistem operasi. Formulir ini jauh lebih lambat dan membutuhkan `ACCESS EXCLUSIVE` kunci di setiap meja.

Untuk informasi lengkap tentang parameter ini, lihat dokumentasi PostgreSQL.

Di Aurora dan Amazon RDS, autovacuum adalah proses daemon (utilitas latar belakang) yang menjalankan `VACUUM` dan `ANALYZE` memerintahkan secara teratur untuk membersihkan data yang berlebihan di database dan server. Bahkan jika Anda mengandalkan autovacuuming, kami sarankan Anda meninjau dan menyesuaikan pengaturan autovacuum yang dibahas di bagian berikut untuk memastikan kinerja yang optimal.

Memeriksa kembung

Kueri SQL berikut memeriksa setiap tabel dalam skema XMLdan mengidentifikasi baris mati (tupel) yang membuang ruang disk:

```
SELECT schemaname || '.' || relname as tuplename,
    n_dead_tup,
    (n_dead_tup::float / n_live_tup::float) * 100 as pfrag
FROM pg_stat_user_tables
WHERE schemaname = 'xml' and n_dead_tup > 0 and n_live_tup > 0 order by pfrag desc;
```

Jika kueri ini mengembalikan persentase tinggi (pfrag) tupel mati, Anda dapat menggunakan `VACUUM` perintah untuk merebut kembali ruang.

Untuk memantau ukuran data sebelum dan sesudah transaksi, jalankan kueri berikut di shell setelah tersambung ke database tertentu:

```
SELECT pg_size_pretty(pg_relation_size('table_name'));
```

autovacuum

Anda dapat mengatur autovacuum secara global dengan menggunakan parameter `autovacuum` konfigurasi, atau Anda dapat mengubahnya berdasarkan per tabel dengan mengatur `autovacuum_enabled` kolom tabel ke `true` atau `false` untuk `pg_class` tabel tertentu.

Ketika Anda mengaktifkan autovacuum di atas meja, server database secara berkala memindai tabel untuk baris mati dan tupel dan menghapusnya di latar belakang, tanpa intervensi dari administrator database. Ini membantu menjaga tabel tetap kecil, meningkatkan kinerja kueri, dan mengurangi ukuran cadangan.

AWS CLI sintaks

Perintah berikut memungkinkan `autovacuum` untuk kelompok parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify autovacuum on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters "ParameterName=autovacuum,ParameterValue=true,ApplyMethod=immediate"

# Modify autovacuum on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters "ParameterName=autovacuum,ParameterValue=true,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengaturApplyMethod=immediate)

Nilai default: Diaktifkan

Anda juga dapat menonaktifkan atau mengaktifkan autovacuum pada tabel tertentu dengan menggunakan psql:

```
ALTER TABLE <table_name> SET (autovacuum_enabled = true);
```

Terlalu banyak menyedot debu dapat memengaruhi kinerja, jadi penting untuk memantau kinerja proses autovacuum serta kinerja database Anda, dan menyesuaikan pengaturan sesuai kebutuhan.

Contoh

Database PostgreSQL Anda memiliki tabel yang menerima volume operasi tulis dan hapus yang tinggi. Tanpa autovacuum, tabel ini pada akhirnya akan diisi dengan baris mati (yaitu, baris yang telah ditandai untuk dihapus tetapi belum dihapus secara fisik dari tabel). Baris mati ini akan memakan ruang pada disk, memperlambat kueri, dan meningkatkan ukuran cadangan. Anda dapat mengaktifkan autovacuum di atas meja untuk secara otomatis memindai baris mati dan menghapusnya untuk mengurangi masalah ini.

autovacuum_work_mem

`autovacuum_work_mem`adalah parameter konfigurasi PostgreSQL yang mengontrol jumlah memori yang digunakan oleh proses autovacuum ketika melakukan tugas pemeliharaan tabel seperti menyedot debu atau analisis.

Di Aurora dan Amazon RDS, Anda dapat menyesuaikan nilai `autovacuum_work_mem` untuk mengoptimalkan kinerja.

AWS CLI sintaks

Perintah berikut memungkinkan `autovacuum_work_mem` untuk kelompok parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify autovacuum_work_mem on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_work_mem on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: `GREATEST({DBInstanceClassMemory/32768},131072)` KB di Aurora PostgreSQL kompatibel, 64 MB di Amazon RDS for PostgreSQL. Namun, nilai default mungkin bervariasi tergantung pada versi spesifik Amazon RDS atau Aurora yang Anda gunakan.

Contoh

Database Amazon RDS for PostgreSQL Anda memiliki tabel besar yang sering diperbarui. Seiring waktu, Anda melihat bahwa database menjadi lebih lambat, dan Anda menduga bahwa autovacuum terlalu lama untuk diselesaikan.

Sebagai bagian dari penyelidikan Anda, Anda memeriksa log sistem, menggunakan `pg_stat_activity` tampilan untuk melihat kueri dan proses mana yang sedang berjalan, memeriksa `pg_stat_user_tables` tampilan untuk melihat statistik untuk setiap tabel, menggunakan `pg_settings` tampilan untuk membandingkan nilai dengan memori yang tersedia pada sistem, dan memantau penggunaan memori untuk lonjakan. `autovacuum_work_mem` Setelah Anda mengumpulkan informasi ini, Anda dapat mengatur `autovacuum_work_mem` ke nilai optimal yang dibutuhkan beban kerja Anda. Untuk menemukan keseimbangan yang tepat antara penggunaan memori dan kinerja, Anda mungkin memutuskan untuk mengaturnya ke seperempat dari memori yang tersedia pada sistem. Setelah Anda mengubah nilai, Anda memantau kinerja database dan mungkin melihat bahwa autovacuum selesai jauh lebih cepat dari sebelumnya dan database Anda berkinerja lebih cepat secara keseluruhan.

autovacuum_naptime

`autovacuum_naptime`Parameter mengontrol interval waktu antara proses autovacuum yang berurutan. Nilai defaultnya adalah 15 detik untuk Amazon RDS for PostgreSQL dan 5 detik untuk Aurora PostgreSQL kompatibel.

Misalnya, katakanlah database Amazon RDS for PostgreSQL Anda memiliki tabel yang menerima volume operasi tulis dan hapus yang tinggi. Jika Anda mempertahankan pengaturan default, pemindaian autovacuum yang sering akan mengganggu tabel yang sangat transaktif ini. Jika Anda mengatur parameter ini ke nilai tinggi, akan ada interval yang lebih panjang antara pemindaian berturut-turut, dan baris mati akan dihapus lebih jarang.

Anda dapat menggunakan `autovacuum_naptime` untuk mengelola beban yang disebabkan oleh proses vakum, terutama jika Anda memiliki server sibuk yang sudah memiliki beban CPU atau I/O yang tinggi. Semakin lama Anda mengatur waktu tidur siang, semakin jarang autovacuum berjalan, yang mengurangi beban pada server. Namun, pengaturan `autovacuum_naptime` ke nilai yang sangat tinggi dapat menyebabkan tabel PostgreSQL Anda tumbuh dan baris mati terakumulasi, yang menyebabkan penurunan kinerja. Kami menyarankan Anda memantau kinerja proses autovacuum dan menyesuaikan `autovacuum_naptime` pengaturan sesuai kebutuhan.

AWS CLI sintaks

Perintah berikut berubah `autovacuum_naptime` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify autovacuum_naptime on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_naptime,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_naptime on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_naptime,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengaturApplyMethod=immediate)

Nilai default: 15 detik (Amazon RDS untuk PostgreSQL), 5 detik (Aurora PostgreSQL kompatibel)

autovacuum_max_workers

`autovacuum_max_workers`Parameter mengontrol jumlah maksimum proses pekerja yang dapat dibuat oleh proses autovacuum. Setiap proses pekerja bertanggung jawab untuk menyedot debu atau menganalisis satu tabel.

Misalnya, katakanlah Anda memiliki database besar dengan banyak tabel yang sering diperbarui dan dihapus. Jika Anda mengatur `autovacuum_max_workers` ke nilai rendah seperti 1, hanya satu tabel yang dapat disedot pada satu waktu, dan dibutuhkan waktu lebih lama untuk semua tabel dibersihkan. Jika Anda mengatur `autovacuum_max_workers` ke nilai tinggi seperti 8, hingga delapan tabel dapat disedot secara bersamaan. Ini dapat membuat proses pembersihan lebih cepat untuk database yang berisi banyak tabel.

AWS CLI sintaks

Perintah berikut berubah `autovacuum_max_workers` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify autovacuum_max_workers on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
```

```
  --parameters
 "ParameterName=autovacuum_max_workers,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_max_workers on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_max_workers,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Statis (menerapkan perubahan membutuhkan reboot)

Nilai default: `GREATEST(DBInstanceClassMemory/64371566592,3)` pekerja

Meningkatkan `autovacuum_max_workers` pengaturan dapat meningkatkan beban di server, yang dapat memengaruhi kinerja jika Anda tidak memiliki sumber daya yang cukup. Pengaturan optimal tergantung pada persyaratan spesifik database Anda, ukurannya, dan jumlah tabel yang dikandungnya. Kami menyarankan Anda bereksperimen dengan nilai yang berbeda dan memantau kinerja untuk menemukan pengaturan optimal untuk kasus penggunaan Anda.

autovacuum_vacuum_scale_factor

Parameter `autovacuum_vacuum_scale_factor` konfigurasi mengontrol seberapa agresif proses autovacuum seharusnya saat menyedot debu tabel.

Faktor skala vakum adalah sebagian kecil dari jumlah total tupel dalam tabel yang harus dimodifikasi sebelum autovacuum membersihkan tabel. Nilai default adalah 0,1 (yaitu, 10 persen tupel harus dimodifikasi). Misalnya, jika sebuah tabel memiliki 1.000.000 tupel, dan 100.000 tupel tersebut ditandai sebagai mati atau dihapus, autovacuum menyedot tabel, tergantung pada nilai sebagai faktor pengendali. `autovacuum_vacuum_threshold`

`autovacuum_vacuum_scale_factor`Parameter membantu Anda mengontrol seberapa sering proses vakum berjalan. Jika tabel menerima banyak operasi tulis, Anda mungkin ingin menurunkan faktor skala vakum sehingga autovacuum berjalan lebih sering, dan menjaga tabel lebih kecil. Sebaliknya, jika tabel menerima beberapa operasi tulis, Anda mungkin ingin meningkatkan faktor skala vakum sehingga autovacuum berjalan lebih jarang, dan menghemat sumber daya.

AWS CLI sintaks

Perintah berikut berubah `autovacuum_vacuum_scale_factor` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify autovacuum_vacuum_scale_factor on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_vacuum_scale_factor,ParameterValue=<new_value>,ApplyMethod=immediate

# Modify autovacuum_vacuum_scale_factor on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_vacuum_scale_factor,ParameterValue=<new_value>,ApplyMethod=immediate
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengaturApplyMethod=immediate)

Nilai default: 0,1

autovacuum_vacuum_scale_factorParameter bekerja bersama dengan autovacuum_vacuum_cost_limit, and autovacuum_naptime parameterautovacuum_vacuum_threshold,. Untuk informasi tambahan tentang parameter ini, lihat posting AWS blog Memahami autovacuum di lingkungan Amazon RDS for PostgreSQL.

autovacuum_vacuum_threshold

autovacuum_vacuum_thresholdParameter mengontrol jumlah minimum pembaruan tupel atau operasi penghapusan yang harus terjadi pada tabel sebelum autovacuum menyedotnya. Pengaturan ini dapat berguna untuk mencegah penyedotan debu yang tidak perlu pada tabel yang tidak memiliki tingkat operasi yang tinggi. Nilai defaultnya adalah 50, yang merupakan default mesin PostgreSQL, untuk Amazon RDS for PostgreSQL dan Aurora PostgreSQL yang kompatibel.

Misalnya, katakanlah Anda memiliki tabel dengan 100.000 baris dan autovacuum_vacuum_threshold diatur ke 50. Jika tabel hanya menerima 49 pembaruan atau penghapusan, autovacuum tidak akan menyedotnya. Jika tabel menerima 50 atau lebih pembaruan atau penghapusan, autovacuum akan menyedotnya, tergantung pada nilai autovacuum_vacuum_scale_factor dikalikan dengan jumlah baris tabel sebagai faktor pengendali.

Menyetel parameter ini terlalu tinggi dapat menyebabkan tabel tumbuh dan baris mati menumpuk, yang dapat memengaruhi kinerja.

AWS CLI sintaks

Perintah berikut berubah `autovacuum_vacuum_threshold` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify autovacuum_vacuum_threshold on a DB parameter group
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=autovacuum_vacuum_threshold,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_vacuum_threshold on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
   --db-cluster-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=autovacuum_vacuum_threshold,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: 50 operasi

`autovacuum_vacuum_threshold`Parameter bekerja bersama dengan `autovacuum_vacuum_cost_limit, and autovacuum_naptime` parameter`autovacuum_vacuum_scale_factor`,. Pengaturan optimal tergantung pada persyaratan spesifik database dan ukuran tabel Anda.

Untuk informasi tambahan tentang parameter ini, lihat posting AWS blog Memahami autovacuum di lingkungan Amazon RDS for PostgreSQL.

autovacuum_analyze_scale_factor

`autovacuum_analyze_scale_factor`Parameter mengontrol seberapa agresif proses autovacuum seharusnya ketika menganalisis (mengumpulkan statistik tentang distribusi data dalam tabel.

Proses autovacuum menggunakan parameter ini untuk menghitung ambang batas berdasarkan jumlah tupel dalam tabel. Jika jumlah sisipan tupel, pembaruan, atau penghapusan melebihi ambang batas ini, autovacuum menganalisis tabel. Nilai defaultnya adalah 0,05 (yaitu, 5 persen tupel harus dimodifikasi) untuk Amazon RDS for PostgreSQL dan Aurora PostgreSQL yang kompatibel.

Misalnya, katakanlah tabel Anda memiliki 1.000.000 tupel dan Anda mempertahankan nilai default `autovacuum_analyze_scale_factor` pada 0,05. Jika tabel menerima 50.000 atau lebih pembaruan atau penghapusan, autovacuum menyedotnya, tergantung pada

`autovacuum_analyze_threshold` nilainya dan menambahkan jumlah baris tabel sebagai faktor pengontrol.

AWS CLI sintaks

Perintah berikut berubah `autovacuum_analyze_scale_factor` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify autovacuum_analyze_scale_factor on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_analyze_scale_factor,ParameterValue=<new_value>,ApplyMethod=immediat

# Modify autovacuum_analyze_scale_factor on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_analyze_scale_factor,ParameterValue=<new_value>,ApplyMethod=immediat
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: 0,05 (5 persen)

Penting bagi perencana kueri untuk mengumpulkan statistik untuk membuat keputusan berdasarkan informasi, seperti cara mengakses data dan cara mengaturnya, jadi kami sarankan Anda memantau kinerja proses autovacuum dan menyesuaikan pengaturan yang diperlukan untuk memastikan bahwa statistik mutakhir.

`autovacuum_analyze_scale_factor`Parameter bekerja bersama dengan `autovacuum_analyze_cost_limit, and autovacuum_naptime` parameter`autovacuum_analyze_threshold`,. Pengaturan optimal tergantung pada persyaratan spesifik database dan ukuran tabel Anda serta frekuensi pembaruan. Untuk informasi tambahan tentang parameter ini, lihat posting AWS blog Memahami autovacuum di lingkungan Amazon RDS for PostgreSQL.

autovacuum_analyze_threshold

`autovacuum_analyze_threshold`Parameternya mirip dengan`autovacuum_vacuum_threshold`. Ini mengontrol jumlah minimum sisipan tupel,

pembaruan, atau penghapusan yang harus terjadi pada tabel sebelum autovacuum menganalisisnya. Pengaturan ini dapat berguna untuk mencegah penyedotan debu yang tidak perlu pada tabel yang tidak memiliki tingkat operasi yang tinggi. Nilai defaultnya adalah 50, yang merupakan default mesin PostgreSQL, untuk Amazon RDS for PostgreSQL dan Aurora PostgreSQL yang kompatibel.

Misalnya, katakanlah Anda memiliki tabel dengan 100.000 baris dan Anda mempertahankan `autovacuum_analyze_threshold` default di 50. Jika tabel hanya menerima 49 sisipan, pembaruan, atau penghapusan, autovacuum tidak akan menganalisisnya. Jika tabel menerima 50 atau lebih sisipan, pembaruan, atau penghapusan, autovacuum akan menganalisisnya, menjaga nilai `autovacuum_analyze_scale_factor` dikalikan dengan jumlah baris tabel sebagai faktor pengendali.

AWS CLI sintaks

Perintah berikut berubah `autovacuum_analyze_threshold` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify autovacuum_analyze_threshold on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_analyze_threshold,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_analyze_threshold on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_analyze_threshold,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: 50 operasi

Parameter ini bekerja bersama dengan `autovacuum_analyze_scale_factor` parameter, jadi pertimbangkan kedua pengaturan saat Anda mengkonfigurasi autovacuum.

Penting bagi perencana kueri untuk mengumpulkan statistik untuk membuat keputusan berdasarkan informasi, seperti cara mengakses data dan cara mengaturnya. Pengaturan yang `autovacuum_analyze_threshold` terlalu tinggi dapat menyebabkan statistik menjadi basi, yang menyebabkan kinerja yang buruk. Kami menyarankan Anda memantau kinerja proses autovacuum dan menyesuaikan pengaturan sesuai kebutuhan.

Untuk informasi tambahan tentang parameter ini, lihat posting AWS blog Memahami autovacuum di lingkungan Amazon RDS for PostgreSQL.

autovacuum_vacuum_cost_limit

`autovacuum_vacuum_cost_limit`Parameter mengontrol jumlah sumber daya CPU dan I/O yang dapat dikonsumsi oleh pekerja autovacuum.

Membatasi penggunaan sumber daya proses autovacuum dapat membantu mencegah mereka mengkonsumsi terlalu banyak CPU atau disk I/O, yang mungkin berdampak pada kinerja kueri lain yang berjalan pada sistem yang sama. Parameter menentukan batas biaya, yang merupakan unit pekerjaan yang diperbolehkan dilakukan pekerja sebelum harus berhenti dan memeriksa untuk melihat apakah masih di bawah batas. Misalnya, jika parameter diatur ke 2.000, seorang pekerja diizinkan untuk memproses 2.000 unit pekerjaan sebelum berhenti.

Anda dapat mengatur `autovacuum_vacuum_cost_limit` parameter dengan menggunakan `SET` perintah dalam sesi PostgreSQL, atau menggunakan perintah. AWS CLI

AWS CLI sintaks

Perintah berikut berubah `autovacuum_vacuum_cost_limit` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify autovacuum_vacuum_cost_limit on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_vacuum_cost_limit,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_vacuum_cost_limit on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_vacuum_cost_limit,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: `GREATEST({log(DBInstanceClassMemory/21474836480)*600},200)` unit kerja

Jika Anda menetapkan nilai `autovacuum_vacuum_cost_limit` terlalu tinggi, proses autovacuum mungkin menghabiskan terlalu banyak sumber daya dan memperlambat kueri lainnya. Jika Anda

mengaturnya terlalu rendah, proses autovacuum mungkin tidak merebut kembali ruang yang cukup, yang menyebabkan tabel menjadi lebih besar dari waktu ke waktu. Sangat penting untuk menemukan keseimbangan yang tepat yang bekerja untuk sistem Anda.

Parameter ini hanya mempengaruhi proses autovacuum, bukan perintah manual. `VACUUM` Juga, ini hanya berlaku untuk proses autovacuum untuk `VACUUM` tetapi tidak untuk. `ANALYZE`

Menyetel parameter logging

Menyetel parameter logging di PostgreSQL membantu memastikan bahwa Anda mengumpulkan informasi yang benar tanpa menghasilkan log besar yang membanjiri sistem Anda.

Mengoptimalkan parameter logging sangat penting untuk menyeimbangkan detail log dengan kinerja sistem dan penggunaan disk. Anda dapat menyesuaikan parameter logging berikut untuk menangkap tingkat detail yang sesuai dalam log, untuk mendiagnosis masalah, dan untuk menyelidiki insiden secara efektif sambil meminimalkan dampak pada kinerja sistem dan penggunaan disk.

- rds.force_autovacuum_logging
- rds.force_admin_logging_level
- log_durasi
- log_min_duration_statement
- log_error_verbosity
- log_statement
- log_statement_stats
- log_min_error_statement
- log_min_messages
- log_temp_file
- log_koneksi
- log_disonnections

Parameter ini dibahas secara lebih rinci di bagian berikut.

> **Warning**
>
> Pengaturan terbaik untuk parameter ini bergantung pada kebijakan dan persyaratan kepatuhan organisasi Anda. Namun, mengaktifkan parameter logging dapat menghasilkan sejumlah besar log dan pesan, yang dapat menggunakan penyimpanan dan memengaruhi kinerja, terutama untuk database yang sibuk. Kami menyarankan Anda menggunakan parameter ini dengan hati-hati. Misalnya, Anda mungkin memutuskan untuk mengaktifkannya sementara untuk mempersempit masalah dengan pernyataan SQL yang berkinerja lambat, dan mematikannya saat periode pemantauan selesai.

rds.force_autovacuum_logging

`rds.force_autovacuum_logging`Parameter (hanya tersedia di Amazon RDS untuk PostgreSQL) mengontrol apakah tindakan autovacuum dicatat di log server. Nilai-nilainya adalah `disabled` `debug5` `debug4``debug3`,`debug2`,,`debug1`,`info`,`notice`,`warning`,`error`,`log`,`fatal`,`panic`. Nilai default-nya adalah `warning`.

Saat Anda mengaktifkan`rds.force_autovacuum_logging`, semua tindakan proses autovacuum, seperti kapan proses dimulai, kapan berakhir, dan berapa banyak baris yang disedot, dicatat. Ini berguna untuk debugging atau pemecahan masalah kinerja autovacuum.

AWS CLI sintaks

Perintah berikut berubah `rds.force_autovacuum_logging` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify rds.force_autovacuum_logging on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=rds.force_autovacuum_logging,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify rds.force_autovacuum_logging on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=rds.force_autovacuum_logging,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: `warning`

Contoh

Anda dapat menggunakan `rds.force_autovacuum_logging` parameter untuk menganalisis kinerja autovacuum pada tabel yang memiliki tingkat tulis yang sangat tinggi. Misalnya, jika tabel Anda menerima sejumlah besar operasi tulis dan hapus per detik, dan Anda mengalami kinerja yang lambat, Anda dapat mengaktifkan parameter untuk mencatat waktu mulai dan akhir setiap proses autovacuum dan untuk menentukan berapa banyak baris yang disedot. Ini dapat memberikan informasi berharga tentang seberapa sering autovacuum berjalan, berapa

lama waktu yang dibutuhkan untuk berjalan, dan berapa banyak baris yang disedot. Anda kemudian dapat menggunakan informasi ini untuk menyempurnakan pengaturan autovacuum sepertiautovacuum_vacuum_scale_factor,autovacuum_vacuum_threshold, dan untuk mengoptimalkan kinerja. autovacuum_naptime

rds.force_admin_logging_level

rds.force_admin_logging_levelParameter (hanya tersedia di Amazon RDS for PostgreSQL) mengontrol tingkat detail dalam log yang dihasilkan oleh operasi administratif seperti penyedot debu, analisis, dan pengindeksan ulang. Ia menerima nilaidebug5,,,debug4,debug3,debug2, debug1log,info,notice,warning,error, logfatal, dan off (default). Pengaturan optimal tergantung pada kasus penggunaan Anda. Misalnya, jika Anda memecahkan masalah, Anda mungkin ingin menyetel parameter ke tingkat debug. Jika tidak, Anda dapat menggunakanlog,info, atau warning pengaturan.

Jika disetel rds.force_admin_logging_level kedebug1, Anda dapat mencatat informasi terperinci untuk operasi pengindeksan ulang, seperti waktu mulai dan akhir, jumlah baris yang diproses, dan kesalahan atau peringatan apa pun yang terjadi selama proses berlangsung. Ini dapat memberikan informasi berharga tentang kinerja proses pengindeksan ulang dan membantu Anda memecahkan masalah apa pun yang terjadi.

AWS CLI sintaks

Perintah berikut berubah rds.force_admin_logging_level untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify rds.force_admin_logging_level on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=rds.force_admin_logging_level,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify rds.force_admin_logging_level on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=rds.force_admin_logging_level,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengaturApplyMethod=immediate)

Nilai default: `off`

Contoh

Anda dapat menggunakan `rds.force_admin_logging_level` untuk memantau dan menganalisis kinerja operasi administratif di beberapa tabel dalam database besar. Misalnya, katakanlah Anda memiliki database besar dengan banyak tabel, dan Anda ingin mengoptimalkan kinerja tabel ini dengan secara teratur menjalankan operasi penyedot debu dan analisis pada mereka. Dengan mengatur `rds.force_admin_logging_level` parameter ke `info` ataulog, Anda dapat mencatat waktu mulai dan akhir dari setiap operasi dan tabel yang terpengaruh. Anda dapat menggunakan informasi ini untuk melacak kinerja operasi administratif di berbagai tabel dan mengidentifikasi tabel yang mungkin memerlukan pemeliharaan yang lebih sering atau lebih agresif.

Beberapa tingkat logging menghasilkan sejumlah besar file log dan pesan yang dapat mengisi ruang disk dengan cepat, terutama jika Anda memiliki database yang sibuk. Kami menyarankan Anda menggunakan parameter ini dengan hati-hati dan mematikannya ketika periode pemantauan selesai.

log_duration

`log_duration`Parameter mengontrol apakah durasi setiap kueri (yaitu, waktu yang diperlukan untuk menjalankan) dicatat dengan kueri. Saat Anda mengatur parameter inion, waktu yang diperlukan untuk menjalankan setiap kueri disertakan dalam output log bersama dengan teks kueri. Waktu diukur dalam milidetik.

Kasus penggunaan utama untuk `log_duration` parameter ini adalah untuk membantu penyetelan kinerja dan pemecahan masalah. Dengan mencatat durasi setiap kueri, Anda dapat mengidentifikasi kueri yang membutuhkan waktu paling lama untuk dijalankan, dan kemudian memfokuskan upaya Anda untuk mengoptimalkan kueri tersebut. Ini dapat membantu Anda mengidentifikasi dan memperbaiki kemacetan kinerja, dan membantu meningkatkan kinerja keseluruhan database Anda.

AWS CLI sintaks

Perintah berikut berubah `log_duration` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify log_duration on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_duration,ParameterValue=<new_value>,ApplyMethod=immediate"
```

```
# Modify log_duration on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_duration,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengaturApplyMethod=immediate)

Nilai default: off

Contoh

Anda dapat menggunakan parameter ini jika Anda mencurigai bahwa kueri atau kumpulan kueri tertentu menyebabkan masalah kinerja. Dengan mengaktifkan log_duration parameter dan memeriksa keluaran log, Anda dapat melihat kueri mana yang paling lama dijalankan dan kemudian mengambil tindakan yang sesuai, seperti mengoptimalkan indeks, menambahkan indeks baru, atau menulis ulang kueri.

Mengaktifkan log_duration dapat meningkatkan volume output log. Kami menyarankan Anda menggunakannya hanya saat diperlukan dan mematikannya selama operasi standar untuk menghindari pengisian penyimpanan atau membuat log sulit dibaca.

log_min_duration_statement

log_min_duration_statementParameter mengontrol jumlah waktu minimum, dalam milidetik, bahwa pernyataan SQL berjalan sebelum dicatat.

Parameter ini membantu Anda mengidentifikasi kueri yang berjalan lama yang dapat menyebabkan masalah kinerja. Anda dapat mengaturnya ke nilai ambang batas (runtime yang dianggap terlalu lama untuk beban kerja tertentu) untuk menangkap kueri yang melebihi ambang batas tersebut dan mengidentifikasi potensi hambatan kinerja. Untuk contoh kasus penggunaan, lihat Menggunakan parameter logging untuk menangkap variabel bind nanti dalam panduan ini.

AWS CLI sintaks

Perintah berikut berubah log_min_duration_statement untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify log_min_duration_statement on a DB parameter group
```

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_duration_statement,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_ min_duration_statement on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_duration_statement,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengaturApplyMethod=immediate)

Nilai default: 1 (dinonaktifkan, yang merupakan default mesin PostgreSQL)

Contoh

Perintah berikut mencatat pernyataan apa pun yang membutuhkan waktu lebih dari 100 milidetik untuk dijalankan:

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=log_min_duration_statement,ParameterValue=100,ApplyMethod=immediate"
```

log_error_verbosity

log_error_verbosityParameter mengontrol tingkat detail yang termasuk dalam output log untuk kesalahan dan pesan yang dicatat pada tingkat kesalahan atau lebih tinggi. Parameter ini dapat mengambil salah satu dari tiga nilai:terse,default, atauverbose.

- tersehanya mencakup teks pesan, tingkat kesalahan, dan file dan nomor baris tempat kesalahan terjadi.
- defaulttermasuk teks pesan, tingkat kesalahan, file dan nomor baris, dan konteks kesalahan.
- verbosetermasuk teks pesan, tingkat kesalahan, file dan nomor baris, konteks kesalahan, dan pesan kesalahan lengkap.

Atur parameter verbose untuk mendapatkan informasi paling rinci untuk pemecahan masalah dan debugging di lingkungan non-produksi. Dalam lingkungan produksi, Anda mungkin ingin

mengaturnya ke `default` atau lebih `terse` itu hanya menyediakan informasi penting dan tidak mengisi penyimpanan log dengan terlalu banyak detail.

AWS CLI sintaks

Perintah berikut berubah `log_error_verbosity` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify log_error_verbosity on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_error_verbosity,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_error_verbosity on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_error_verbosity,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: `default`

log_statement

`log_statement` Parameter mengontrol pernyataan SQL yang dicatat di log server. Parameter dapat mengambil salah satu nilai berikut:

- `none`(default) tidak mencatat pernyataan apa pun
- `ddl`hanya mencatat pernyataan bahasa definisi data (DDL) seperti dan `CREATE TABLE` `ALTER TABLE`
- `mod`hanya mencatat pernyataan pengubah data seperti`INSERT`,, dan `UPDATE` `DELETE`
- `all`mencatat semua pernyataan SQL

Anda dapat menggunakan `log_statement` parameter untuk mengontrol jumlah informasi yang ditulis ke log dengan mendokumentasikan hanya jenis pernyataan tertentu yang relevan dengan kasus penggunaan Anda.

AWS CLI sintaks

Perintah berikut berubah `log_statement` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify log_statement on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_statement,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_statement on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_statement,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: none

Contoh

Dalam lingkungan produksi, Anda mungkin ingin mengatur `log_statement ddl` untuk mencatat hanya pernyataan DDL dan melacak setiap perubahan yang dibuat pada skema database. Dalam lingkungan pengembangan, Anda mungkin ingin mengatur parameter untuk mencatat semua pernyataan `all` untuk membantu proses debug dan pemecahan masalah. Untuk contoh kasus penggunaan lainnya, lihat Menggunakan parameter logging untuk menangkap variabel bind nanti dalam panduan ini.

Mengaktifkan `log_statement` dapat meningkatkan volume output log, jadi gunakan hanya saat diperlukan dan matikan untuk menghindari pengisian penyimpanan atau membuat log sulit dibaca.

Kami menyarankan Anda memantau sistem Anda dan menyesuaikan nilai parameter ini untuk mencapai keseimbangan yang sesuai antara jumlah informasi yang dicatat dan penyimpanan dan kinerja sistem.

log_statement_stats

`log_statement_stats`Parameter mengontrol apakah statistik yang terkait dengan menjalankan pernyataan SQL dicatat dengan pernyataan. Saat Anda mengaktifkan parameter ini, statistik seperti

jumlah baris yang terpengaruh, jumlah blok disk yang dibaca dan ditulis, dan waktu yang diperlukan untuk menjalankan pernyataan disertakan dalam output log.

Anda dapat menggunakan `log_statement_stats` parameter untuk mengumpulkan informasi tambahan tentang kinerja pernyataan individu dan beban kerja keseluruhan. Dengan mencatat statistik pernyataan, Anda dapat mengidentifikasi pola dalam kinerja kueri dan penggunaan sumber daya, dan menggunakan informasi tersebut untuk mengoptimalkan database Anda dan meningkatkan kinerja secara keseluruhan.

AWS CLI sintaks

Perintah berikut berubah `log_statement_stats` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify log_statement_stats on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_statement_stats,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_statement_stas on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_statement_stats,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: `off` (PostgreSQL engine default); gunakan 0 atau 1 (Boolean) untuk mengatur dalam kelompok parameter

Contoh

Anda dapat menggunakan `log_statement_stats` untuk menganalisis perilaku kueri tertentu, melihat bagaimana ia menggunakan sumber daya seperti CPU, memori, dan disk I/O, dan mengidentifikasi apakah kueri dapat dioptimalkan. Anda juga dapat menggunakan parameter ini untuk melihat apakah tabel tertentu sering dibaca (yang mungkin menunjukkan perlunya membuat indeks pada kolom tertentu) atau jika tabel dipindai terlalu sering.

Mengaktifkan `log_statement_stats` dapat meningkatkan volume output log, jadi gunakan hanya saat diperlukan dan matikan untuk menghindari pengisian penyimpanan atau membuat log sulit dibaca.

log_min_error_statement

`log_min_error_statement`Parameter mengontrol pernyataan SQL mana yang menghasilkan kesalahan akan dicatat. Nilai-nilainya adalah `debug5` `debug4``debug3`,`debug2`,`debug1`,,`info`,`notice`,`warning`,`error`,`log`,,`fatal`, dan`panic`. Pengaturan ini mengontrol jumlah informasi yang ditulis ke log sehingga Anda dapat memfilter pesan dengan tingkat keparahan yang lebih rendah. Anda dapat mengatur parameter ini ke tingkat keparahan yang lebih tinggi untuk mengurangi jumlah output log dan menemukan pesan penting dengan lebih mudah.

AWS CLI sintaks

Perintah berikut berubah `log_min_error_statement` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify log_min_error_statement on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_error_statement,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_min_error_statement on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_error_statement,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: `error` (PostgreSQL engine default)

Contoh

Anda dapat mempertimbangkan untuk menggunakan `log_min_error_statement` saat memecahkan masalah tertentu dan ingin melihat pesan kesalahan dari pernyataan SQL yang menyebabkan kesalahan.

log_min_messages

`log_min_messages`Parameter mengontrol tingkat keparahan yang ditulis ke log. Anda dapat mengatur parameter ke`debug5`,,`debug4`,

`debug3`debug2,debug1,info,notice,warning,error,log,fatal, ataupanic. Pengaturan ini mengontrol jumlah informasi yang ditulis ke log sehingga Anda dapat memfilter pesan dengan tingkat keparahan yang lebih rendah. Anda dapat mengatur parameter ini ke tingkat keparahan yang lebih tinggi untuk mengurangi jumlah output log dan menemukan pesan penting dengan lebih mudah.

AWS CLI sintaks

Perintah berikut berubah `log_min_messages` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify log_min_messages on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_messages,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_min_messages on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_messages,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengaturApplyMethod=immediate)

Nilai default: `notice`

Contoh

Jika Anda memecahkan masalah tertentu dan ingin melihat semua pesan kesalahan, Anda dapat menyetel parameter ini `error` agar hanya mencatat kesalahan dan masalah tingkat keparahan yang lebih tinggi. Jika Anda tertarik untuk memantau kinerja sistem, Anda dapat mengatur parameter ini `info` untuk melihat informasi yang lebih rinci seperti durasi dan statistik setiap pernyataan.

Pengaturan `log_min_messages` ke tingkat keparahan yang lebih tinggi mengurangi volume log. Kami menyarankan Anda menyetel parameter ini tergantung pada kasus penggunaan spesifik Anda, ukuran log yang ingin Anda periksa, dan jumlah ruang disk yang Anda miliki.

log_temp_files

`log_temp_files`Parameter mengontrol pencatatan nama dan ukuran file sementara. Ini berlaku untuk file sementara yang dibuat untuk tujuan seperti jenis, hash, dan hasil kueri sementara. Ketika

parameter ini diaktifkan, entri log dihasilkan untuk setiap file sementara setelah penghapusan, termasuk ukuran filenya, dalam byte. Anda dapat mengatur parameter ini ke 0 (nol) untuk pencatatan komprehensif semua informasi file sementara, atau ke nilai positif untuk mencatat file yang melebihi ukuran itu (dalam kilobyte, jika unit tidak ditentukan). Ini dapat berguna untuk mengidentifikasi dan menyelesaikan kemacetan kinerja atau masalah lain yang terkait dengan penyimpanan sementara. Secara default, pencatatan file sementara dinonaktifkan.

AWS CLI sintaks

Perintah berikut berubah `log_temp_files` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify log_temp_files on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_temp_files,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_temp_files on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_temp_files,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: -1 (default mesin PostgreSQL)

Contoh

Anda dapat mengaktifkan parameter ini jika Anda mencurigai bahwa sistem menggunakan terlalu banyak penyimpanan sementara, atau bahwa file sementara tidak dihapus dengan benar. Saat Anda memeriksa output log, Anda dapat melihat kueri atau operasi yang menghasilkan file sementara dan bagaimana file-file ini digunakan.

Beberapa kueri atau operasi membuat sejumlah besar file sementara, sehingga mengaktifkan `log_temp_files` dapat memengaruhi kinerja keseluruhan sistem Anda.

log_connections

`log_connections`Parameter mengontrol apakah koneksi ke database dicatat. Ketika Anda mengatur parameter inion, log berisi informasi tentang setiap koneksi yang berhasil ke database, seperti alamat IP klien, nama pengguna, nama database, dan tanggal dan waktu koneksi.

Anda dapat menggunakan `log_connections` parameter untuk memantau dan memecahkan masalah koneksi ke database. Anda dapat melihat pengguna, aplikasi, terminal, dan bot yang terhubung ke database, dari mana mereka terhubung, dan seberapa sering. Informasi ini dapat berguna untuk mengidentifikasi dan menyelesaikan masalah terkait koneksi atau melacak pola penggunaan.

AWS CLI sintaks

Perintah berikut berubah `log_connections` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify log_connections on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_connections,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_connections on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_connections,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengaturApplyMethod=immediate)

Nilai default: `off` (PostgreSQL engine default)

Contoh

Anda dapat menggunakan parameter ini jika Anda mencurigai bahwa terlalu banyak koneksi ke database, atau pengguna atau alamat IP tertentu yang terlalu sering terhubung, memengaruhi kinerja. Dengan mengaktifkan `log_connections` parameter dan memeriksa output log, Anda dapat melihat nomor dan detail semua koneksi.

Sebelum Anda mengaktifkan parameter ini, periksa kebijakan organisasi Anda dan pertimbangkan implikasi keamanan dari pencatatan alamat IP dan nama pengguna.

log_disconnections

`log_disconnections`Parameter mengontrol pencatatan pemutusan dari database. Ketika Anda mengatur parameter inion, ia mencatat informasi tentang akhir setiap sesi, seperti alamat IP klien, nama pengguna, nama database, dan tanggal dan waktu pemutusan.

Anda dapat menggunakan `log_disconnections` parameter untuk memantau dan memecahkan masalah terminasi sesi database. Anda dapat melihat pengguna, aplikasi, terminal, dan bot yang terputus dari database, kapan, dan mengapa. Misalnya, Anda dapat meninjau penghentian yang tidak terduga seperti crash atau pemutusan koneksi yang dimulai oleh administrator. Informasi ini dapat berguna untuk mengidentifikasi dan menyelesaikan masalah yang terkait dengan pemutusan atau melacak pola penggunaan.

AWS CLI sintaks

Perintah berikut berubah `log_disconnections` untuk grup parameter DB tertentu. Perubahan ini berlaku untuk semua instance atau cluster yang menggunakan grup parameter.

```
# Modify log_disconnections on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_disconnections,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_disconnections on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_disconnections,ParameterValue=<new_value>,ApplyMethod=immediate"
```

Jenis: Dinamis (perubahan diterapkan segera jika Anda mengatur`ApplyMethod=immediate`)

Nilai default: `off` (PostgreSQL engine default)

Contoh

Anda mungkin menggunakan `log_disconnections` jika Anda mencurigai bahwa ada terlalu banyak pengguna yang memutuskan sambungan dari database, atau bahwa pengguna atau alamat IP tertentu terlalu sering terputus. Dengan mengaktifkan `log_disconnections` parameter dan memeriksa output log, Anda dapat melihat nomor dan detail semua pemutusan, termasuk siapa, kapan, dan apakah ada kesalahan yang muncul sebelum pemutusan.

Sebelum Anda mengaktifkan parameter ini, periksa kebijakan organisasi Anda dan pertimbangkan implikasi keamanan dari pencatatan alamat IP dan nama pengguna.

Menggunakan parameter logging untuk menangkap variabel bind

Kasus penggunaan umum untuk menangkap variabel bind di PostgreSQL adalah men-debug dan menyesuaikan kinerja kueri SQL. Variabel bind memungkinkan Anda meneruskan data ke kueri saat Anda menjalankannya. Dengan menangkap variabel bind, Anda dapat melihat data input yang diteruskan ke kueri, yang dapat membantu Anda mengidentifikasi masalah apa pun dengan data atau dengan kinerja kueri. Menangkap variabel bind juga dapat membantu Anda mengaudit data input dan mendeteksi potensi risiko keamanan atau aktivitas berbahaya.

Ada beberapa cara untuk menangkap variabel bind untuk PostgreSQL. Salah satu metode adalah mengaktifkan `debug_print_parse` dan `debug_print_rewritten` parameter. Hal ini menyebabkan PostgreSQL mengirim versi pernyataan SQL yang diurai dan ditulis ulang, bersama dengan variabel terikat, ke log server.

- `debug_print_parse`: Saat Anda mengaktifkan parameter ini, pohon parse dari kueri yang masuk dicetak ke log server. Ini dapat berguna untuk memahami struktur kueri dan nilai parameter terikat apa pun.
- `debug_print_rewritten`: Saat Anda mengaktifkan parameter ini, bentuk kueri masuk yang ditulis ulang dicetak ke log server. Ini dapat berguna untuk memahami bagaimana perencana kueri menafsirkan kueri dan nilai parameter terikat apa pun.

Anda dapat menggunakan dua parameter tambahan di Amazon RDS dan Aurora untuk menangkap variabel bind di database PostgreSQL Anda:

- `log_min_duration_statement`: Parameter ini menetapkan durasi minimum pernyataan sebelum dicatat, dalam milidetik. Ketika pernyataan membutuhkan waktu lebih lama dari durasi yang ditentukan, nilai bind-nya disertakan dalam output log.
- `log_statement`: Parameter ini mengontrol pernyataan SQL mana yang dicatat. Setel parameter ini ke `all` atau mengikat untuk menyertakan nilai terikat dalam log. Meningkatkan level logging memengaruhi kinerja, jadi sebaiknya Anda mengembalikan perubahan setelah pemecahan masalah.

Anda juga dapat menggunakan `pg_stat_statements` ekstensi, yang menyediakan statistik kinerja untuk semua pernyataan SQL yang dijalankan oleh server, termasuk teks kueri dan nilai terikat.

Ekstensi ini memungkinkan Anda menggunakan pgAdmin atau alat serupa untuk memantau dan menganalisis kinerja kueri.

Pilihan lain adalah menggunakan `pg_bind_parameter_status()` fungsi untuk mendapatkan nilai parameter terikat dari pernyataan yang disiapkan atau menggunakan `pg_get_parameter_status(paramname)` fungsi untuk mengambil status atau nilai parameter runtime tertentu.

Selain itu, Anda dapat menggunakan alat pihak ketiga seperti PGBadger untuk menganalisis log PostgreSQL dan mengekstrak variabel pengikat dan informasi lainnya untuk analisis lebih lanjut.

Menyetel parameter replikasi

Di PostgreSQL, Anda dapat mereplikasi perubahan data dari satu database PostgreSQL ke database PostgreSQL lainnya dengan menggunakan replikasi logis alih-alih replikasi berbasis file fisik. Replikasi logis menggunakan log write-ahead (WAL) untuk menangkap perubahan dan mendukung replikasi tabel yang dipilih atau seluruh database.

Amazon RDS for PostgreSQL dan Aurora PostgreSQL kompatibel keduanya mendukung replikasi logis, sehingga Anda dapat menyiapkan arsitektur database yang sangat tersedia dan skalabel yang dapat menangani lalu lintas baca dan tulis dari berbagai sumber. Layanan ini menggunakan pglogical, yang merupakan ekstensi PostgreSQL open-source, untuk mengimplementasikan replikasi logis.

Menyetel replikasi logis di Aurora dan Amazon RDS penting untuk mencapai kinerja, skalabilitas, dan ketersediaan yang optimal. Anda dapat menyetel parameter dalam ekstensi pglogical untuk mengelola kinerja replikasi logis. Sebagai contoh, Anda dapat:

- Meningkatkan kinerja replikasi dengan meningkatkan jumlah proses pekerja atau menyesuaikan alokasi memori mereka.
- Mengurangi risiko lag replikasi dengan menyesuaikan frekuensi sinkronisasi antara sumber dan database replika.
- Optimalkan penggunaan sumber daya dengan menyesuaikan memori dan alokasi CPU dari proses pekerja.
- Pastikan bahwa proses replikasi tidak menimbulkan dampak yang tidak semestinya pada kinerja database sumber.

Anda dapat menggunakan parameter berikut di Aurora dan Amazon RDS untuk mengontrol dan mengonfigurasi replikasi logis:

- `max_replication_slots`menetapkan jumlah maksimum slot replikasi yang dapat dibuat di server. Slot replikasi adalah reservasi persisten bernama untuk koneksi replikasi untuk mengirim data WAL ke replika.
- `max_wal_senders`menetapkan jumlah maksimum proses pengirim WAL yang terhubung secara bersamaan. Proses pengirim WAL digunakan untuk mengalirkan WAL dari server utama ke replika.
- `wal_sender_timeout`menetapkan waktu maksimum, dalam milidetik, bahwa pengirim WAL menunggu respons dari replika sebelum menyerah dan menyambung kembali.

- `wal_receiver_timeout`menetapkan waktu maksimum, dalam milidetik, bahwa replika menunggu data WAL dari database utama sebelum waktu habis.
- `log_replication_commands`, ketika diatur keon, menjalankan pernyataan SQL terkait replikasi.

Ketika Anda mengaktifkan `rds.logical_replication` parameter (dengan menyetelnya ke 1) `wal_level` parameter diatur k`elogical`, yang berarti bahwa semua perubahan yang dibuat untuk database ditulis ke WAL dalam format yang dapat dibaca dan diterapkan ke replika. Pengaturan ini diperlukan untuk mengaktifkan replikasi logis. Pengaturan ini juga memungkinkan replikasi `SELECT` pernyataan.

Pengaturan `wal_level` untuk `logical` dapat meningkatkan jumlah data yang ditulis ke WAL, dan karena itu ke disk, yang dapat mempengaruhi kinerja sistem. Kami menyarankan Anda mempertimbangkan ruang disk dan kinerja sistem yang tersedia saat mengaktifkan replikasi logis.

Contoh

Anda ingin mereplikasi data dari database utama Anda ke database sekunder untuk tujuan pencadangan dan pemulihan bencana. Namun, database sekunder memiliki volume operasi baca yang tinggi, jadi Anda ingin memastikan bahwa proses replikasi secepat dan seefisien mungkin tanpa mengorbankan integritas data.

Nilai default untuk replikasi logis di Amazon RDS dan Aurora memprioritaskan konsistensi daripada kinerja, jadi nilai tersebut mungkin tidak optimal untuk kasus penggunaan ini. Untuk mengoptimalkan pengaturan replikasi logis Anda untuk kecepatan dan efisiensi, Anda dapat menyesuaikan parameter sebagai berikut:

- Tingkatkan `max_replication_slots` dari 10 (default untuk Amazon RDS) atau 20 (default untuk Aurora) menjadi 30 untuk mengakomodasi kebutuhan pertumbuhan dan replikasi masa depan yang potensial.
- Tingkatkan `max_wal_senders` dari 10 (default) menjadi 20 untuk memastikan bahwa ada cukup proses pengirim WAL untuk memenuhi permintaan replikasi.
- Kurangi `wal_sender_timeout` dari 30 detik (default) menjadi 15 detik untuk memastikan bahwa proses pengirim WAL yang menganggur dihentikan lebih cepat, yang membebaskan sumber daya untuk replikasi aktif.
- Kurangi `wal_receiver_timeout` dari 30 detik (default) menjadi 15 detik untuk memastikan bahwa proses penerima WAL idle dihentikan lebih cepat, yang membebaskan sumber daya untuk replikasi aktif.

- Tingkatkan `max_logical_replication_workers` dari 4 (default) menjadi 8 untuk memastikan bahwa ada cukup proses pekerja replikasi logis untuk memenuhi permintaan replikasi.

Pengoptimalan ini memberikan replikasi data yang lebih cepat dan lebih efisien sambil menjaga integritas dan keamanan data.

Misalnya, jika bencana terjadi dan basis data utama menjadi tidak tersedia, database sekunder sudah memiliki data terbaru yang tersedia karena proses replikasi yang dioptimalkan. Ini akan memungkinkan operasi bisnis Anda untuk terus menyediakan layanan penting tanpa gangguan.

Praktik terbaik

Menyetel replikasi logis dengan beban kerja yang besar dapat menjadi tugas kompleks yang bergantung pada berbagai faktor, termasuk ukuran kumpulan data, jumlah tabel yang direplikasi, jumlah replika, dan sumber daya yang tersedia. Berikut adalah beberapa tips umum untuk menyetel replikasi logis dengan beban kerja yang besar:

- Memantau kelambatan replikasi. Replication lag adalah perbedaan waktu antara server utama dan server siaga. Memantau kelambatan replikasi dapat membantu Anda mengidentifikasi potensi kemacetan dan mengambil tindakan untuk meningkatkan kinerja replikasi. Anda dapat menggunakan `pg_current_wal_lsn()` fungsi ini untuk memeriksa lag replikasi saat ini.
- Setel pengaturan WAL. `pg_logical`Ekstensi menggunakan WAL untuk mengirimkan perubahan dari server utama ke server siaga. Jika pengaturan WAL tidak disetel dengan benar, replikasi bisa menjadi lambat dan tidak dapat diandalkan. Pastikan untuk mengatur `max_replication_slots` parameter `max_wal_senders` dan ke nilai yang memadai, tergantung pada beban kerja Anda.
- Memiliki strategi pengindeksan. Memiliki indeks yang tepat pada server utama dapat membantu meningkatkan kinerja replikasi logis, mengurangi I/O pada server utama, dan mengurangi beban pada sistem.
- Gunakan replikasi paralel. Menggunakan replikasi paralel dapat membantu meningkatkan kecepatan replikasi dengan memungkinkan beberapa proses pekerja paralel untuk mereplikasi data. Fitur ini tersedia di PostgreSQL 12 dan yang lebih baru.

Langkah selanjutnya

Setelah mengoptimalkan parameter memori, replikasi, autovacuum, dan logging untuk Amazon RDS for PostgreSQL atau database yang kompatibel dengan Aurora PostgreSQL, pertimbangkan langkah-langkah berikut untuk lebih meningkatkan kinerja database Anda:

- Pantau database Anda. Lacak kinerja database Anda dari waktu ke waktu dengan menggunakan alat pemantauan bawaan atau solusi pihak ketiga. Pantau metrik kinerja utama seperti pemanfaatan CPU, I/O disk, penggunaan memori, dan runtime kueri untuk mengidentifikasi potensi hambatan dan area untuk perbaikan.
- Menyetel parameter secara terus menerus. Saat beban kerja Anda berkembang, terus pantau dan sesuaikan parameter database Anda untuk memastikan kinerja yang optimal. Periksa log sistem, pesan kesalahan, dan metrik kinerja secara teratur untuk mengidentifikasi peluang penyetelan baru.
- Menerapkan caching. Gunakan caching untuk mengurangi jumlah kueri yang mengenai database. Anda dapat menerapkan caching di tingkat aplikasi dengan menggunakan alat seperti Memcached atau Redis, atau Anda dapat menggunakan Amazon ElastiCache untuk menyediakan cache dalam memori untuk database Anda.
- Optimalkan kueri Anda. Kueri yang dirancang dengan buruk dapat berdampak signifikan pada kinerja database. Gunakan `EXPLAIN` dan alat penyetelan kueri lainnya untuk mengidentifikasi kueri lambat, mengoptimalkannya, dan menghilangkan kueri yang tidak perlu.

Dengan mengikuti panduan ini, Anda dapat mengoptimalkan kinerja Aurora atau Amazon RDS for PostgreSQL database Anda dan memastikan bahwa itu memenuhi kebutuhan aplikasi Anda dengan peningkatan kinerja database, peningkatan keandalan, pengurangan waktu henti, keamanan yang lebih baik, dan penghematan biaya. Dengan mengoptimalkan parameter konfigurasi agar sesuai dengan beban kerja Anda, Anda dapat memastikan bahwa database Anda berjalan secara efisien dan menggunakan sumber daya secara efektif, yang mengarah ke kinerja yang lebih baik dan aplikasi yang lebih responsif. Selain itu, parameter yang dikonfigurasi dengan benar dapat mengurangi kemungkinan kesalahan dan kerentanan, menghasilkan peningkatan keandalan dan keamanan yang lebih baik. Ini dapat diterjemahkan ke penghematan biaya dalam hal pengurangan pemeliharaan dan downtime, serta pengalaman dan kepuasan pengguna secara keseluruhan yang lebih baik.

Sumber daya

- Parameter Amazon Aurora PostgreSQL, Bagian 1: Manajemen rencana memori dan kueri (posting blog)AWS
- Parameter Amazon Aurora PostgreSQL, Bagian 2: Replikasi, keamanan, dan logging (posting blog)AWS
- Parameter Amazon Aurora PostgreSQL, Bagian 3: Parameter pengoptimal (posting blog)AWS
- Parameter Amazon Aurora PostgreSQL, Bagian 4: Opsi kompatibilitas ANSI (posting blog)AWS
- Bekerja dengan Amazon Aurora PostgreSQL (dokumentasi)AWS
- Bekerja dengan Amazon RDS untuk PostgreSQL (dokumentasi)AWS
- Memantau beban DB dengan Performance Insights di Amazon RDS (dokumentasi)AWS
- Menggunakan CloudWatch metrik Amazon (AWS dokumentasi)
- pg_stats_statement dokumentasi PostgreSQL)

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
Informasi yang diperbarui tentang parameter memori dan autovacuum	Memperbarui deskripsi parameter random_page_cost; menambahkan unit yang hilang ke nilai default untuk memori dan parameter autovacuum; memperbar ui sintaks untuk parameter max_connections. AWS CLI	Februari 27, 2024
Informasi terbaru tentang `autovacuum`	Memperbaiki pengaturan default autovacuum (diaktifk an).	27 Desember 2023
Informasi terbaru tentang `max_connections`	Memperbarui bagian max_connections dengan panduan baru tentang menyetel parameter ini.	15 November 2023
Publikasi awal	—	31 Oktober 2023

AWS Glosarium Panduan Preskriptif

Berikut ini adalah istilah yang umum digunakan dalam strategi, panduan, dan pola yang disediakan oleh Panduan AWS Preskriptif. Untuk menyarankan entri, silakan gunakan tautan Berikan umpan balik di akhir glosarium.

Nomor

7 Rs

Tujuh strategi migrasi umum untuk memindahkan aplikasi ke cloud. Strategi ini dibangun di atas 5 Rs yang diidentifikasi Gartner pada tahun 2011 dan terdiri dari yang berikut:

- Refactor/Re-Architect — Memindahkan aplikasi dan memodifikasi arsitekturnya dengan memanfaatkan sepenuhnya fitur cloud-native untuk meningkatkan kelincahan, kinerja, dan skalabilitas. Ini biasanya melibatkan porting sistem operasi dan database. Contoh: Migrasikan database Oracle lokal Anda ke Amazon Aurora PostgreSQL Compatible Edition.
- Replatform (angkat dan bentuk ulang) — Pindahkan aplikasi ke cloud, dan perkenalkan beberapa tingkat pengoptimalan untuk memanfaatkan kemampuan cloud. Contoh: Migrasikan database Oracle lokal Anda ke Amazon Relational Database Service (Amazon RDS) untuk Oracle di. AWS Cloud
- Pembelian kembali (drop and shop) - Beralih ke produk yang berbeda, biasanya dengan beralih dari lisensi tradisional ke model SaaS. Contoh: Migrasikan sistem manajemen hubungan pelanggan (CRM) Anda ke Salesforce.com.
- Rehost (lift dan shift) — Pindahkan aplikasi ke cloud tanpa membuat perubahan apa pun untuk memanfaatkan kemampuan cloud. Contoh: Migrasikan database Oracle lokal Anda ke Oracle pada instans EC2 di. AWS Cloud
- Relokasi (hypervisor-level lift and shift) — Pindahkan infrastruktur ke cloud tanpa membeli perangkat keras baru, menulis ulang aplikasi, atau memodifikasi operasi yang ada. Anda memigrasikan server dari platform lokal ke layanan cloud untuk platform yang sama. Contoh: Migrasikan Microsoft Hyper-V aplikasi ke AWS.
- Pertahankan (kunjungi kembali) - Simpan aplikasi di lingkungan sumber Anda. Ini mungkin termasuk aplikasi yang memerlukan refactoring besar, dan Anda ingin menunda pekerjaan itu sampai nanti, dan aplikasi lama yang ingin Anda pertahankan, karena tidak ada pembenaran bisnis untuk memigrasikannya.

- Pensiun — Menonaktifkan atau menghapus aplikasi yang tidak lagi diperlukan di lingkungan sumber Anda.

A

ABAC

Lihat kontrol akses berbasis atribut.

layanan abstrak

Lihat layanan terkelola.

ASAM

Lihat atomisitas, konsistensi, isolasi, daya tahan.

migrasi aktif-aktif

Metode migrasi database di mana database sumber dan target tetap sinkron (dengan menggunakan alat replikasi dua arah atau operasi penulisan ganda), dan kedua database menangani transaksi dari menghubungkan aplikasi selama migrasi. Metode ini mendukung migrasi dalam batch kecil yang terkontrol alih-alih memerlukan pemotongan satu kali. Ini lebih fleksibel tetapi membutuhkan lebih banyak pekerjaan daripada migrasi aktif-pasif.

migrasi aktif-pasif

Metode migrasi database di mana database sumber dan target disimpan dalam sinkron, tetapi hanya database sumber yang menangani transaksi dari menghubungkan aplikasi sementara data direplikasi ke database target. Basis data target tidak menerima transaksi apa pun selama migrasi.

fungsi agregat

Fungsi SQL yang beroperasi pada sekelompok baris dan menghitung nilai pengembalian tunggal untuk grup. Contoh fungsi agregat meliputi `SUM` dan`MAX`.

AI

Lihat kecerdasan buatan.

AIOps

Lihat operasi kecerdasan buatan.

anonimisasi

Proses menghapus informasi pribadi secara permanen dalam kumpulan data. Anonimisasi dapat membantu melindungi privasi pribadi. Data anonim tidak lagi dianggap sebagai data pribadi.

anti-pola

Solusi yang sering digunakan untuk masalah berulang di mana solusinya kontra-produktif, tidak efektif, atau kurang efektif daripada alternatif.

kontrol aplikasi

Pendekatan keamanan yang memungkinkan penggunaan hanya aplikasi yang disetujui untuk membantu melindungi sistem dari malware.

portofolio aplikasi

Kumpulan informasi rinci tentang setiap aplikasi yang digunakan oleh organisasi, termasuk biaya untuk membangun dan memelihara aplikasi, dan nilai bisnisnya. Informasi ini adalah kunci untuk penemuan portofolio dan proses analisis dan membantu mengidentifikasi dan memprioritaskan aplikasi yang akan dimigrasi, dimodernisasi, dan dioptimalkan.

kecerdasan buatan (AI)

Bidang ilmu komputer yang didedikasikan untuk menggunakan teknologi komputasi untuk melakukan fungsi kognitif yang biasanya terkait dengan manusia, seperti belajar, memecahkan masalah, dan mengenali pola. Untuk informasi lebih lanjut, lihat Apa itu Kecerdasan Buatan?

operasi kecerdasan buatan (AIOps)

Proses menggunakan teknik pembelajaran mesin untuk memecahkan masalah operasional, mengurangi insiden operasional dan intervensi manusia, dan meningkatkan kualitas layanan. Untuk informasi selengkapnya tentang cara AIOps digunakan dalam strategi AWS migrasi, lihat panduan integrasi operasi.

enkripsi asimetris

Algoritma enkripsi yang menggunakan sepasang kunci, kunci publik untuk enkripsi dan kunci pribadi untuk dekripsi. Anda dapat berbagi kunci publik karena tidak digunakan untuk dekripsi, tetapi akses ke kunci pribadi harus sangat dibatasi.

atomisitas, konsistensi, isolasi, daya tahan (ACID)

Satu set properti perangkat lunak yang menjamin validitas data dan keandalan operasional database, bahkan dalam kasus kesalahan, kegagalan daya, atau masalah lainnya.

kontrol akses berbasis atribut (ABAC)

Praktik membuat izin berbutir halus berdasarkan atribut pengguna, seperti departemen, peran pekerjaan, dan nama tim. Untuk informasi selengkapnya, lihat ABAC untuk AWS dokumentasi AWS Identity and Access Management (IAM).

sumber data otoritatif

Lokasi di mana Anda menyimpan versi utama data, yang dianggap sebagai sumber informasi yang paling dapat diandalkan. Anda dapat menyalin data dari sumber data otoritatif ke lokasi lain untuk tujuan memproses atau memodifikasi data, seperti menganonimkan, menyunting, atau membuat nama samaran.

Zona Ketersediaan

Lokasi berbeda di dalam Wilayah AWS yang terisolasi dari kegagalan di Availability Zone lainnya dan menyediakan konektivitas jaringan latensi rendah yang murah ke Availability Zone lainnya di Wilayah yang sama.

AWS Kerangka Adopsi Cloud (AWS CAF)

Kerangka pedoman dan praktik terbaik AWS untuk membantu organisasi mengembangkan rencana yang efisien dan efektif untuk bergerak dengan sukses ke cloud. AWS CAF mengatur panduan ke dalam enam area fokus yang disebut perspektif: bisnis, orang, tata kelola, platform, keamanan, dan operasi. Perspektif bisnis, orang, dan tata kelola fokus pada keterampilan dan proses bisnis; perspektif platform, keamanan, dan operasi fokus pada keterampilan dan proses teknis. Misalnya, perspektif masyarakat menargetkan pemangku kepentingan yang menangani sumber daya manusia (SDM), fungsi kepegawaian, dan manajemen orang. Untuk perspektif ini, AWS CAF memberikan panduan untuk pengembangan, pelatihan, dan komunikasi orang untuk membantu mempersiapkan organisasi untuk adopsi cloud yang sukses. Untuk informasi lebih lanjut, lihat situs web AWS CAF dan whitepaper AWS CAF.

AWS Kerangka Kualifikasi Beban Kerja (AWS WQF)

Alat yang mengevaluasi beban kerja migrasi database, merekomendasikan strategi migrasi, dan memberikan perkiraan kerja. AWS WQF disertakan dengan AWS Schema Conversion Tool ()AWS SCT. Ini menganalisis skema database dan objek kode, kode aplikasi, dependensi, dan karakteristik kinerja, dan memberikan laporan penilaian.

B

bot buruk

Bot yang dimaksudkan untuk mengganggu atau menyebabkan kerugian bagi individu atau organisasi.

BCP

Lihat perencanaan kontinuitas bisnis.

grafik perilaku

Pandangan interaktif yang terpadu tentang perilaku dan interaksi sumber daya dari waktu ke waktu. Anda dapat menggunakan grafik perilaku dengan Amazon Detective untuk memeriksa upaya logon yang gagal, panggilan API yang mencurigakan, dan tindakan serupa. Untuk informasi selengkapnya, lihat Data dalam grafik perilaku di dokumentasi Detektif.

sistem big-endian

Sistem yang menyimpan byte paling signifikan terlebih dahulu. Lihat juga endianness.

klasifikasi biner

Sebuah proses yang memprediksi hasil biner (salah satu dari dua kelas yang mungkin). Misalnya, model ML Anda mungkin perlu memprediksi masalah seperti “Apakah email ini spam atau bukan spam?” atau “Apakah produk ini buku atau mobil?”

filter mekar

Struktur data probabilistik dan efisien memori yang digunakan untuk menguji apakah suatu elemen adalah anggota dari suatu himpunan.

deployment biru/hijau

Strategi penyebaran tempat Anda membuat dua lingkungan yang terpisah namun identik. Anda menjalankan versi aplikasi saat ini di satu lingkungan (biru) dan versi aplikasi baru di lingkungan lain (hijau). Strategi ini membantu Anda dengan cepat memutar kembali dengan dampak minimal.

bot

Aplikasi perangkat lunak yang menjalankan tugas otomatis melalui internet dan mensimulasikan aktivitas atau interaksi manusia. Beberapa bot berguna atau bermanfaat, seperti perayap web yang mengindeks informasi di internet. Beberapa bot lain, yang dikenal sebagai bot buruk, dimaksudkan untuk mengganggu atau membahayakan individu atau organisasi.

botnet

Jaringan bot yang terinfeksi oleh malware dan berada di bawah kendali satu pihak, yang dikenal sebagai bot herder atau operator bot. Botnet adalah mekanisme paling terkenal untuk skala bot dan dampaknya.

cabang

Area berisi repositori kode. Cabang pertama yang dibuat dalam repositori adalah cabang utama. Anda dapat membuat cabang baru dari cabang yang ada, dan Anda kemudian dapat mengembangkan fitur atau memperbaiki bug di cabang baru. Cabang yang Anda buat untuk membangun fitur biasanya disebut sebagai cabang fitur. Saat fitur siap dirilis, Anda menggabungkan cabang fitur kembali ke cabang utama. Untuk informasi selengkapnya, lihat Tentang cabang (GitHub dokumentasi).

akses break-glass

Dalam keadaan luar biasa dan melalui proses yang disetujui, cara cepat bagi pengguna untuk mendapatkan akses ke Akun AWS yang biasanya tidak memiliki izin untuk mengaksesnya. Untuk informasi lebih lanjut, lihat indikator Implementasikan prosedur break-glass dalam panduan Well-Architected AWS .

strategi brownfield

Infrastruktur yang ada di lingkungan Anda. Saat mengadopsi strategi brownfield untuk arsitektur sistem, Anda merancang arsitektur di sekitar kendala sistem dan infrastruktur saat ini. Jika Anda memperluas infrastruktur yang ada, Anda dapat memadukan strategi brownfield dan greenfield.

cache penyangga

Area memori tempat data yang paling sering diakses disimpan.

kemampuan bisnis

Apa yang dilakukan bisnis untuk menghasilkan nilai (misalnya, penjualan, layanan pelanggan, atau pemasaran). Arsitektur layanan mikro dan keputusan pengembangan dapat didorong oleh kemampuan bisnis. Untuk informasi selengkapnya, lihat bagian Terorganisir di sekitar kemampuan bisnis dari Menjalankan layanan mikro kontainer di whitepaper. AWS

perencanaan kelangsungan bisnis (BCP)

Rencana yang membahas dampak potensial dari peristiwa yang mengganggu, seperti migrasi skala besar, pada operasi dan memungkinkan bisnis untuk melanjutkan operasi dengan cepat.

C

KAFE

Lihat Kerangka Adopsi AWS Cloud.

penyebaran kenari

Rilis versi yang lambat dan bertahap untuk pengguna akhir. Ketika Anda yakin, Anda menyebarkan versi baru dan mengganti versi saat ini secara keseluruhan.

CCoE

Lihat Cloud Center of Excellence.

CDC

Lihat mengubah pengambilan data.

ubah pengambilan data (CDC)

Proses melacak perubahan ke sumber data, seperti tabel database, dan merekam metadata tentang perubahan tersebut. Anda dapat menggunakan CDC untuk berbagai tujuan, seperti mengaudit atau mereplikasi perubahan dalam sistem target untuk mempertahankan sinkronisasi.

rekayasa kekacauan

Sengaja memperkenalkan kegagalan atau peristiwa yang mengganggu untuk menguji ketahanan sistem. Anda dapat menggunakan AWS Fault Injection Service (AWS FIS) untuk melakukan eksperimen yang menekankan AWS beban kerja Anda dan mengevaluasi responsnya.

CI/CD

Lihat integrasi berkelanjutan dan pengiriman berkelanjutan.

klasifikasi

Proses kategorisasi yang membantu menghasilkan prediksi. Model ML untuk masalah klasifikasi memprediksi nilai diskrit. Nilai diskrit selalu berbeda satu sama lain. Misalnya, model mungkin perlu mengevaluasi apakah ada mobil dalam gambar atau tidak.

Enkripsi sisi klien

Enkripsi data secara lokal, sebelum target Layanan AWS menerimanya.

Cloud Center of Excellence (CCoE)

Tim multi-disiplin yang mendorong upaya adopsi cloud di seluruh organisasi, termasuk mengembangkan praktik terbaik cloud, memobilisasi sumber daya, menetapkan jadwal migrasi, dan memimpin organisasi melalui transformasi skala besar. Untuk informasi selengkapnya, lihat posting CCoE di Blog Strategi AWS Cloud Perusahaan.

komputasi cloud

Teknologi cloud yang biasanya digunakan untuk penyimpanan data jarak jauh dan manajemen perangkat IoT. Cloud computing umumnya terhubung ke teknologi edge computing.

model operasi cloud

Dalam organisasi TI, model operasi yang digunakan untuk membangun, mematangkan, dan mengoptimalkan satu atau lebih lingkungan cloud. Untuk informasi selengkapnya, lihat Membangun Model Operasi Cloud Anda.

tahap adopsi cloud

Empat fase yang biasanya dilalui organisasi ketika mereka bermigrasi ke AWS Cloud:

- Proyek — Menjalankan beberapa proyek terkait cloud untuk bukti konsep dan tujuan pembelajaran
- Foundation — Melakukan investasi dasar untuk meningkatkan adopsi cloud Anda (misalnya, membuat landing zone, mendefinisikan CCoE, membuat model operasi)
- Migrasi — Migrasi aplikasi individual
- Re-invention — Mengoptimalkan produk dan layanan, dan berinovasi di cloud

Tahapan ini didefinisikan oleh Stephen Orban dalam posting blog The Journey Toward Cloud-First & the Stages of Adoption di blog Strategi Perusahaan. AWS Cloud Untuk informasi tentang bagaimana kaitannya dengan strategi AWS migrasi, lihat panduan kesiapan migrasi.

CMDB

Lihat database manajemen konfigurasi.

repositori kode

Lokasi di mana kode sumber dan aset lainnya, seperti dokumentasi, sampel, dan skrip, disimpan dan diperbarui melalui proses kontrol versi. Repositori cloud umum termasuk GitHub atau. AWS CodeCommit Setiap versi kode disebut cabang. Dalam struktur layanan mikro, setiap repositori

dikhususkan untuk satu bagian fungsionalitas. Pipa CI/CD tunggal dapat menggunakan beberapa repositori.

cache dingin

Cache buffer yang kosong, tidak terisi dengan baik, atau berisi data basi atau tidak relevan. Ini mempengaruhi kinerja karena instance database harus membaca dari memori utama atau disk, yang lebih lambat daripada membaca dari cache buffer.

data dingin

Data yang jarang diakses dan biasanya historis. Saat menanyakan jenis data ini, kueri lambat biasanya dapat diterima. Memindahkan data ini ke tingkat penyimpanan atau kelas yang berkinerja lebih rendah dan lebih murah dapat mengurangi biaya.

visi komputer (CV)

Bidang AI yang menggunakan pembelajaran mesin untuk menganalisis dan mengekstrak informasi dari format visual seperti gambar dan video digital. Misalnya, AWS Panorama menawarkan perangkat yang menambahkan CV ke jaringan kamera lokal, dan Amazon SageMaker menyediakan algoritme pemrosesan gambar untuk CV.

konfigurasi drift

Untuk beban kerja, konfigurasi berubah dari status yang diharapkan. Ini dapat menyebabkan beban kerja menjadi tidak patuh, dan biasanya bertahap dan tidak disengaja.

database manajemen konfigurasi (CMDB)

Repositori yang menyimpan dan mengelola informasi tentang database dan lingkungan TI, termasuk komponen perangkat keras dan perangkat lunak dan konfigurasinya. Anda biasanya menggunakan data dari CMDB dalam penemuan portofolio dan tahap analisis migrasi.

paket kesesuaian

Kumpulan AWS Config aturan dan tindakan remediasi yang dapat Anda kumpulkan untuk menyesuaikan kepatuhan dan pemeriksaan keamanan Anda. Anda dapat menerapkan paket kesesuaian sebagai entitas tunggal di Akun AWS dan Wilayah, atau di seluruh organisasi, dengan menggunakan templat YAMM. Untuk informasi selengkapnya, lihat Paket kesesuaian dalam dokumentasi. AWS Config

integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD)

Proses mengotomatiskan sumber, membangun, menguji, pementasan, dan tahap produksi dari proses rilis perangkat lunak. CI/CD umumnya digambarkan sebagai pipa. CI/CD dapat membantu

Anda mengotomatiskan proses, meningkatkan produktivitas, meningkatkan kualitas kode, dan memberikan lebih cepat. Untuk informasi lebih lanjut, lihat Manfaat pengiriman berkelanjutan. CD juga dapat berarti penerapan berkelanjutan. Untuk informasi selengkapnya, lihat Continuous Delivery vs Continuous Deployment.

CV

Lihat visi komputer.

D

data saat istirahat

Data yang stasioner di jaringan Anda, seperti data yang ada di penyimpanan.

klasifikasi data

Proses untuk mengidentifikasi dan mengkategorikan data dalam jaringan Anda berdasarkan kekritisan dan sensitivitasnya. Ini adalah komponen penting dari setiap strategi manajemen risiko keamanan siber karena membantu Anda menentukan perlindungan dan kontrol retensi yang tepat untuk data. Klasifikasi data adalah komponen pilar keamanan dalam AWS Well-Architected Framework. Untuk informasi selengkapnya, lihat Klasifikasi data.

penyimpangan data

Variasi yang berarti antara data produksi dan data yang digunakan untuk melatih model ML, atau perubahan yang berarti dalam data input dari waktu ke waktu. Penyimpangan data dapat mengurangi kualitas, akurasi, dan keadilan keseluruhan dalam prediksi model ML.

data dalam transit

Data yang aktif bergerak melalui jaringan Anda, seperti antara sumber daya jaringan.

jala data

Kerangka arsitektur yang menyediakan kepemilikan data terdistribusi dan terdesentralisasi dengan manajemen dan tata kelola terpusat.

minimalisasi data

Prinsip pengumpulan dan pemrosesan hanya data yang sangat diperlukan. Mempraktikkan minimalisasi data AWS Cloud dapat mengurangi risiko privasi, biaya, dan jejak karbon analitik Anda.

perimeter data

Satu set pagar pembatas pencegahan di AWS lingkungan Anda yang membantu memastikan bahwa hanya identitas tepercaya yang mengakses sumber daya tepercaya dari jaringan yang diharapkan. Untuk informasi selengkapnya, lihat Membangun perimeter data pada AWS.

prapemrosesan data

Untuk mengubah data mentah menjadi format yang mudah diuraikan oleh model ML Anda. Preprocessing data dapat berarti menghapus kolom atau baris tertentu dan menangani nilai yang hilang, tidak konsisten, atau duplikat.

asal data

Proses melacak asal dan riwayat data sepanjang siklus hidupnya, seperti bagaimana data dihasilkan, ditransmisikan, dan disimpan.

subjek data

Individu yang datanya dikumpulkan dan diproses.

gudang data

Sistem manajemen data yang mendukung intelijen bisnis, seperti analitik. Gudang data biasanya berisi sejumlah besar data historis, dan biasanya digunakan untuk kueri dan analisis.

bahasa definisi database (DDL)

Pernyataan atau perintah untuk membuat atau memodifikasi struktur tabel dan objek dalam database.

bahasa manipulasi basis data (DHTML)

Pernyataan atau perintah untuk memodifikasi (memasukkan, memperbarui, dan menghapus) informasi dalam database.

DDL

Lihat bahasa definisi database.

ansambel yang dalam

Untuk menggabungkan beberapa model pembelajaran mendalam untuk prediksi. Anda dapat menggunakan ansambel dalam untuk mendapatkan prediksi yang lebih akurat atau untuk memperkirakan ketidakpastian dalam prediksi.

pembelajaran mendalam

Subbidang ML yang menggunakan beberapa lapisan jaringan saraf tiruan untuk mengidentifikasi pemetaan antara data input dan variabel target yang diinginkan.

defense-in-depth

Pendekatan keamanan informasi di mana serangkaian mekanisme dan kontrol keamanan dilapisi dengan cermat di seluruh jaringan komputer untuk melindungi kerahasiaan, integritas, dan ketersediaan jaringan dan data di dalamnya. Saat Anda mengadopsi strategi ini AWS, Anda menambahkan beberapa kontrol pada lapisan AWS Organizations struktur yang berbeda untuk membantu mengamankan sumber daya. Misalnya, defense-in-depth pendekatan mungkin menggabungkan otentikasi multi-faktor, segmentasi jaringan, dan enkripsi.

administrator yang didelegasikan

Di AWS Organizations, layanan yang kompatibel dapat mendaftarkan akun AWS anggota untuk mengelola akun organisasi dan mengelola izin untuk layanan tersebut. Akun ini disebut administrator yang didelegasikan untuk layanan itu. Untuk informasi selengkapnya dan daftar layanan yang kompatibel, lihat Layanan yang berfungsi dengan AWS Organizations AWS Organizations dokumentasi.

deployment

Proses pembuatan aplikasi, fitur baru, atau perbaikan kode tersedia di lingkungan target. Deployment melibatkan penerapan perubahan dalam basis kode dan kemudian membangun dan menjalankan basis kode itu di lingkungan aplikasi.

lingkungan pengembangan

Lihat lingkungan.

kontrol detektif

Kontrol keamanan yang dirancang untuk mendeteksi, mencatat, dan memperingatkan setelah suatu peristiwa terjadi. Kontrol ini adalah garis pertahanan kedua, memperingatkan Anda tentang peristiwa keamanan yang melewati kontrol pencegahan yang ada. Untuk informasi selengkapnya, lihat Kontrol Detektif dalam Menerapkan kontrol keamanan pada. AWS

pemetaan aliran nilai pengembangan (DVSM)

Sebuah proses yang digunakan untuk mengidentifikasi dan memprioritaskan kendala yang mempengaruhi kecepatan dan kualitas dalam siklus hidup pengembangan perangkat lunak. DVSM memperluas proses pemetaan aliran nilai yang awalnya dirancang untuk praktik

manufaktur ramping. Ini berfokus pada langkah-langkah dan tim yang diperlukan untuk menciptakan dan memindahkan nilai melalui proses pengembangan perangkat lunak.

kembar digital

Representasi virtual dari sistem dunia nyata, seperti bangunan, pabrik, peralatan industri, atau jalur produksi. Kembar digital mendukung pemeliharaan prediktif, pemantauan jarak jauh, dan optimalisasi produksi.

tabel dimensi

Dalam skema bintang, tabel yang lebih kecil yang berisi atribut data tentang data kuantitatif dalam tabel fakta. Atribut tabel dimensi biasanya bidang teks atau angka diskrit yang berperilaku seperti teks. Atribut ini biasanya digunakan untuk pembatasan kueri, pemfilteran, dan pelabelan set hasil.

musibah

Peristiwa yang mencegah beban kerja atau sistem memenuhi tujuan bisnisnya di lokasi utama yang digunakan. Peristiwa ini dapat berupa bencana alam, kegagalan teknis, atau akibat dari tindakan manusia, seperti kesalahan konfigurasi yang tidak disengaja atau serangan malware.

pemulihan bencana (DR)

Strategi dan proses yang Anda gunakan untuk meminimalkan downtime dan kehilangan data yang disebabkan oleh bencana. Untuk informasi selengkapnya, lihat Disaster Recovery of Workloads on AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML~

Lihat bahasa manipulasi database.

desain berbasis domain

Pendekatan untuk mengembangkan sistem perangkat lunak yang kompleks dengan menghubungkan komponennya ke domain yang berkembang, atau tujuan bisnis inti, yang dilayani oleh setiap komponen. Konsep ini diperkenalkan oleh Eric Evans dalam bukunya, Domain-Driven Design: Tackling Complexity in the Heart of Software (Boston: Addison-Wesley Professional, 2003). Untuk informasi tentang cara menggunakan desain berbasis domain dengan pola gambar pencekik, lihat Memodernisasi layanan web Microsoft ASP.NET (ASMX) lama secara bertahap menggunakan container dan Amazon API Gateway.

DR

Lihat pemulihan bencana.

deteksi drift

Melacak penyimpangan dari konfigurasi dasar. Misalnya, Anda dapat menggunakan AWS CloudFormation untuk mendeteksi penyimpangan dalam sumber daya sistem, atau Anda dapat menggunakannya AWS Control Tower untuk mendeteksi perubahan di landing zone yang mungkin memengaruhi kepatuhan terhadap persyaratan tata kelola.

DVSM

Lihat pemetaan aliran nilai pengembangan.

E

EDA

Lihat analisis data eksplorasi.

komputasi tepi

Teknologi yang meningkatkan daya komputasi untuk perangkat pintar di tepi jaringan IoT. Jika dibandingkan dengan komputasi awan, komputasi tepi dapat mengurangi latensi komunikasi dan meningkatkan waktu respons.

enkripsi

Proses komputasi yang mengubah data plaintext, yang dapat dibaca manusia, menjadi ciphertext.

kunci enkripsi

String kriptografi dari bit acak yang dihasilkan oleh algoritma enkripsi. Panjang kunci dapat bervariasi, dan setiap kunci dirancang agar tidak dapat diprediksi dan unik.

endianness

Urutan byte disimpan dalam memori komputer. Sistem big-endian menyimpan byte paling signifikan terlebih dahulu. Sistem little-endian menyimpan byte paling tidak signifikan terlebih dahulu.

titik akhir

Lihat titik akhir layanan.

layanan endpoint

Layanan yang dapat Anda host di cloud pribadi virtual (VPC) untuk dibagikan dengan pengguna lain. Anda dapat membuat layanan endpoint dengan AWS PrivateLink dan memberikan izin

kepada prinsipal lain Akun AWS atau ke AWS Identity and Access Management (IAM). Akun atau prinsipal ini dapat terhubung ke layanan endpoint Anda secara pribadi dengan membuat titik akhir VPC antarmuka. Untuk informasi selengkapnya, lihat Membuat layanan titik akhir di dokumentasi Amazon Virtual Private Cloud (Amazon VPC).

perencanaan sumber daya perusahaan (ERP)

Sistem yang mengotomatiskan dan mengelola proses bisnis utama (seperti akuntansi, MES, dan manajemen proyek) untuk suatu perusahaan.

enkripsi amplop

Proses mengenkripsi kunci enkripsi dengan kunci enkripsi lain. Untuk informasi selengkapnya, lihat Enkripsi amplop dalam dokumentasi AWS Key Management Service (AWS KMS).

lingkungan

Sebuah contoh dari aplikasi yang sedang berjalan. Berikut ini adalah jenis lingkungan yang umum dalam komputasi awan:

- Development Environment — Sebuah contoh dari aplikasi yang berjalan yang hanya tersedia untuk tim inti yang bertanggung jawab untuk memelihara aplikasi. Lingkungan pengembangan digunakan untuk menguji perubahan sebelum mempromosikannya ke lingkungan atas. Jenis lingkungan ini kadang-kadang disebut sebagai lingkungan pengujian.
- lingkungan yang lebih rendah — Semua lingkungan pengembangan untuk aplikasi, seperti yang digunakan untuk build awal dan pengujian.
- lingkungan produksi — Sebuah contoh dari aplikasi yang berjalan yang pengguna akhir dapat mengakses. Dalam pipa CI/CD, lingkungan produksi adalah lingkungan penyebaran terakhir.
- lingkungan atas — Semua lingkungan yang dapat diakses oleh pengguna selain tim pengembangan inti. Ini dapat mencakup lingkungan produksi, lingkungan praproduksi, dan lingkungan untuk pengujian penerimaan pengguna.

epik

Dalam metodologi tangkas, kategori fungsional yang membantu mengatur dan memprioritaskan pekerjaan Anda. Epik memberikan deskripsi tingkat tinggi tentang persyaratan dan tugas implementasi. Misalnya, epos keamanan AWS CAF mencakup manajemen identitas dan akses, kontrol detektif, keamanan infrastruktur, perlindungan data, dan respons insiden. Untuk informasi selengkapnya tentang epos dalam strategi AWS migrasi, lihat panduan implementasi program.

ERP

Lihat perencanaan sumber daya perusahaan.

analisis data eksplorasi (EDA)

Proses menganalisis dataset untuk memahami karakteristik utamanya. Anda mengumpulkan atau mengumpulkan data dan kemudian melakukan penyelidikan awal untuk menemukan pola, mendeteksi anomali, dan memeriksa asumsi. EDA dilakukan dengan menghitung statistik ringkasan dan membuat visualisasi data.

F

tabel fakta

Tabel tengah dalam skema bintang. Ini menyimpan data kuantitatif tentang operasi bisnis. Biasanya, tabel fakta berisi dua jenis kolom: kolom yang berisi ukuran dan yang berisi kunci asing ke tabel dimensi.

gagal cepat

Filosofi yang menggunakan pengujian yang sering dan bertahap untuk mengurangi siklus hidup pengembangan. Ini adalah bagian penting dari pendekatan tangkas.

batas isolasi kesalahan

Dalam AWS Cloud, batas seperti Availability Zone, Wilayah AWS, control plane, atau data plane yang membatasi efek kegagalan dan membantu meningkatkan ketahanan beban kerja. Untuk informasi selengkapnya, lihat Batas Isolasi AWS Kesalahan.

cabang fitur

Lihat cabang.

fitur

Data input yang Anda gunakan untuk membuat prediksi. Misalnya, dalam konteks manufaktur, fitur bisa berupa gambar yang diambil secara berkala dari lini manufaktur.

pentingnya fitur

Seberapa signifikan fitur untuk prediksi model. Ini biasanya dinyatakan sebagai skor numerik yang dapat dihitung melalui berbagai teknik, seperti Shapley Additive Explanations (SHAP) dan gradien terintegrasi. Untuk informasi lebih lanjut, lihat Interpretabilitas model pembelajaran mesin dengan:.AWS

transformasi fitur

Untuk mengoptimalkan data untuk proses ML, termasuk memperkaya data dengan sumber tambahan, menskalakan nilai, atau mengekstrak beberapa set informasi dari satu bidang data. Hal ini memungkinkan model ML untuk mendapatkan keuntungan dari data. Misalnya, jika Anda memecah tanggal “2021-05-27 00:15:37" menjadi “2021", “Mei”, “Kamis”, dan “15", Anda dapat membantu algoritme pembelajaran mempelajari pola bernuansa yang terkait dengan komponen data yang berbeda.

FGAC

Lihat kontrol akses berbutir halus.

kontrol akses berbutir halus (FGAC)

Penggunaan beberapa kondisi untuk mengizinkan atau menolak permintaan akses.

migrasi flash-cut

Metode migrasi database yang menggunakan replikasi data berkelanjutan melalui pengambilan data perubahan untuk memigrasikan data dalam waktu sesingkat mungkin, alih-alih menggunakan pendekatan bertahap. Tujuannya adalah untuk menjaga downtime seminimal mungkin.

G

pemblokiran geografis

Lihat pembatasan geografis.

pembatasan geografis (pemblokiran geografis)

Di Amazon CloudFront, opsi untuk mencegah pengguna di negara tertentu mengakses distribusi konten. Anda dapat menggunakan daftar izinkan atau daftar blokir untuk menentukan negara yang disetujui dan dilarang. Untuk informasi selengkapnya, lihat Membatasi distribusi geografis konten Anda dalam dokumentasi. CloudFront

Alur kerja Gitflow

Pendekatan di mana lingkungan bawah dan atas menggunakan cabang yang berbeda dalam repositori kode sumber. Alur kerja Gitflow dianggap warisan, dan alur kerja berbasis batang adalah pendekatan modern yang lebih disukai.

strategi greenfield

Tidak adanya infrastruktur yang ada di lingkungan baru. Saat mengadopsi strategi greenfield untuk arsitektur sistem, Anda dapat memilih semua teknologi baru tanpa batasan kompatibilitas dengan infrastruktur yang ada, juga dikenal sebagai brownfield. Jika Anda memperluas infrastruktur yang ada, Anda dapat memadukan strategi brownfield dan greenfield.

pagar pembatas

Aturan tingkat tinggi yang membantu mengatur sumber daya, kebijakan, dan kepatuhan di seluruh unit organisasi (OU). Pagar pembatas preventif menegakkan kebijakan untuk memastikan keselarasan dengan standar kepatuhan. Mereka diimplementasikan dengan menggunakan kebijakan kontrol layanan dan batas izin IAM. Detective guardrails mendeteksi pelanggaran kebijakan dan masalah kepatuhan, dan menghasilkan peringatan untuk remediasi. Mereka diimplementasikan dengan menggunakan AWS Config, AWS Security Hub, Amazon GuardDuty AWS Trusted Advisor, Amazon Inspector, dan pemeriksaan khusus AWS Lambda .

H

HA

Lihat ketersediaan tinggi.

migrasi database heterogen

Memigrasi database sumber Anda ke database target yang menggunakan mesin database yang berbeda (misalnya, Oracle ke Amazon Aurora). Migrasi heterogen biasanya merupakan bagian dari upaya arsitektur ulang, dan mengubah skema dapat menjadi tugas yang kompleks. AWS menyediakan AWS SCT yang membantu dengan konversi skema.

ketersediaan tinggi (HA)

Kemampuan beban kerja untuk beroperasi terus menerus, tanpa intervensi, jika terjadi tantangan atau bencana. Sistem HA dirancang untuk gagal secara otomatis, secara konsisten memberikan kinerja berkualitas tinggi, dan menangani beban dan kegagalan yang berbeda dengan dampak kinerja minimal.

modernisasi sejarawan

Pendekatan yang digunakan untuk memodernisasi dan meningkatkan sistem teknologi operasional (OT) untuk melayani kebutuhan industri manufaktur dengan lebih baik. Sejarawan

adalah jenis database yang digunakan untuk mengumpulkan dan menyimpan data dari berbagai sumber di pabrik.

migrasi database homogen

Memigrasi database sumber Anda ke database target yang berbagi mesin database yang sama (misalnya, Microsoft SQL Server ke Amazon RDS for SQL Server). Migrasi homogen biasanya merupakan bagian dari upaya rehosting atau replatforming. Anda dapat menggunakan utilitas database asli untuk memigrasi skema.

data panas

Data yang sering diakses, seperti data real-time atau data translasi terbaru. Data ini biasanya memerlukan tingkat atau kelas penyimpanan berkinerja tinggi untuk memberikan respons kueri yang cepat.

perbaikan terbaru

Perbaikan mendesak untuk masalah kritis dalam lingkungan produksi. Karena urgensinya, perbaikan terbaru biasanya dibuat di luar alur kerja DevOps rilis biasa.

periode hypercare

Segera setelah cutover, periode waktu ketika tim migrasi mengelola dan memantau aplikasi yang dimigrasi di cloud untuk mengatasi masalah apa pun. Biasanya, periode ini panjangnya 1-4 hari. Pada akhir periode hypercare, tim migrasi biasanya mentransfer tanggung jawab untuk aplikasi ke tim operasi cloud.

I

IAc

Lihat infrastruktur sebagai kode.

kebijakan berbasis identitas

Kebijakan yang dilampirkan pada satu atau beberapa prinsip IAM yang mendefinisikan izin mereka dalam lingkungan. AWS Cloud

aplikasi idle

Aplikasi yang memiliki penggunaan CPU dan memori rata-rata antara 5 dan 20 persen selama periode 90 hari. Dalam proyek migrasi, adalah umum untuk menghentikan aplikasi ini atau mempertahankannya di tempat.

IIoT

Lihat Internet of Things industri.

infrastruktur yang tidak dapat diubah

Model yang menyebarkan infrastruktur baru untuk beban kerja produksi alih-alih memperbarui, menambal, atau memodifikasi infrastruktur yang ada. Infrastruktur yang tidak dapat diubah secara inheren lebih konsisten, andal, dan dapat diprediksi daripada infrastruktur yang dapat berubah. Untuk informasi selengkapnya, lihat praktik terbaik Deploy using immutable infrastructure di AWS Well-Architected Framework.

masuk (masuknya) VPC

Dalam arsitektur AWS multi-akun, VPC yang menerima, memeriksa, dan merutekan koneksi jaringan dari luar aplikasi. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

migrasi inkremental

Strategi cutover di mana Anda memigrasikan aplikasi Anda dalam bagian-bagian kecil alih-alih melakukan satu cutover penuh. Misalnya, Anda mungkin hanya memindahkan beberapa layanan mikro atau pengguna ke sistem baru pada awalnya. Setelah Anda memverifikasi bahwa semuanya berfungsi dengan baik, Anda dapat secara bertahap memindahkan layanan mikro atau pengguna tambahan hingga Anda dapat menonaktifkan sistem lama Anda. Strategi ini mengurangi risiko yang terkait dengan migrasi besar.

Industri 4.0

Sebuah istilah yang diperkenalkan oleh Klaus Schwab pada tahun 2016 untuk merujuk pada modernisasi proses manufaktur melalui kemajuan dalam konektivitas, data real-time, otomatisasi, analitik, dan AI/ML.

infrastruktur

Semua sumber daya dan aset yang terkandung dalam lingkungan aplikasi.

infrastruktur sebagai kode (IAc)

Proses penyediaan dan pengelolaan infrastruktur aplikasi melalui satu set file konfigurasi. IAc dirancang untuk membantu Anda memusatkan manajemen infrastruktur, menstandarisasi sumber daya, dan menskalakan dengan cepat sehingga lingkungan baru dapat diulang, andal, dan konsisten.

Internet of Things industri (IIoT)

Penggunaan sensor dan perangkat yang terhubung ke internet di sektor industri, seperti manufaktur, energi, otomotif, perawatan kesehatan, ilmu kehidupan, dan pertanian. Untuk informasi selengkapnya, lihat Membangun strategi transformasi digital Internet of Things (IIoT) industri.

inspeksi VPC

Dalam arsitektur AWS multi-akun, VPC terpusat yang mengelola inspeksi lalu lintas jaringan antara VPC (dalam hal yang sama atau berbeda Wilayah AWS), internet, dan jaringan lokal. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

Internet of Things (IoT)

Jaringan objek fisik yang terhubung dengan sensor atau prosesor tertanam yang berkomunikasi dengan perangkat dan sistem lain melalui internet atau melalui jaringan komunikasi lokal. Untuk informasi selengkapnya, lihat Apa itu IoT?

interpretabilitas

Karakteristik model pembelajaran mesin yang menggambarkan sejauh mana manusia dapat memahami bagaimana prediksi model bergantung pada inputnya. Untuk informasi selengkapnya, lihat Interpretabilitas model pembelajaran mesin dengan AWS.

IoT

Lihat Internet of Things.

Perpustakaan informasi TI (ITIL)

Serangkaian praktik terbaik untuk memberikan layanan TI dan menyelaraskan layanan ini dengan persyaratan bisnis. ITIL menyediakan dasar untuk ITSM.

Manajemen layanan TI (ITSM)

Kegiatan yang terkait dengan merancang, menerapkan, mengelola, dan mendukung layanan TI untuk suatu organisasi. Untuk informasi tentang mengintegrasikan operasi cloud dengan alat ITSM, lihat panduan integrasi operasi.

ITIL

Lihat perpustakaan informasi TI.

ITSM

Lihat manajemen layanan TI.

L

kontrol akses berbasis label (LBAC)

Implementasi kontrol akses wajib (MAC) di mana pengguna dan data itu sendiri masing-masing secara eksplisit diberi nilai label keamanan. Persimpangan antara label keamanan pengguna dan label keamanan data menentukan baris dan kolom mana yang dapat dilihat oleh pengguna.

landing zone

Landing zone adalah AWS lingkungan multi-akun yang dirancang dengan baik yang dapat diskalakan dan aman. Ini adalah titik awal dari mana organisasi Anda dapat dengan cepat meluncurkan dan menyebarkan beban kerja dan aplikasi dengan percaya diri dalam lingkungan keamanan dan infrastruktur mereka. Untuk informasi selengkapnya tentang zona pendaratan, lihat Menyiapkan lingkungan multi-akun AWS yang aman dan dapat diskalakan.

migrasi besar

Migrasi 300 atau lebih server.

LBAC

Lihat kontrol akses berbasis label.

hak istimewa paling sedikit

Praktik keamanan terbaik untuk memberikan izin minimum yang diperlukan untuk melakukan tugas. Untuk informasi selengkapnya, lihat Menerapkan izin hak istimewa terkecil dalam dokumentasi IAM.

angkat dan geser

Lihat 7 Rs.

sistem endian kecil

Sebuah sistem yang menyimpan byte paling tidak signifikan terlebih dahulu. Lihat juga endianness.

lingkungan yang lebih rendah

Lihat lingkungan.

M

pembelajaran mesin (ML)

Jenis kecerdasan buatan yang menggunakan algoritma dan teknik untuk pengenalan pola dan pembelajaran. ML menganalisis dan belajar dari data yang direkam, seperti data Internet of Things (IoT), untuk menghasilkan model statistik berdasarkan pola. Untuk informasi selengkapnya, lihat Machine Learning.

cabang utama

Lihat cabang.

malware

Perangkat lunak yang dirancang untuk membahayakan keamanan atau privasi komputer. Malware dapat mengganggu sistem komputer, membocorkan informasi sensitif, atau mendapatkan akses yang tidak sah. Contoh malware termasuk virus, worm, ransomware, Trojan horse, spyware, dan keyloggers.

layanan terkelola

Layanan AWS yang AWS mengoperasikan lapisan infrastruktur, sistem operasi, dan platform, dan Anda mengakses titik akhir untuk menyimpan dan mengambil data. Amazon Simple Storage Service (Amazon S3) dan Amazon DynamoDB adalah contoh layanan terkelola. Ini juga dikenal sebagai layanan abstrak.

sistem eksekusi manufaktur (MES)

Sistem perangkat lunak untuk melacak, memantau, mendokumentasikan, dan mengendalikan proses produksi yang mengubah bahan baku menjadi produk jadi di lantai toko.

PETA

Lihat Program Percepatan Migrasi.

mekanisme

Proses lengkap di mana Anda membuat alat, mendorong adopsi alat, dan kemudian memeriksa hasilnya untuk melakukan penyesuaian. Mekanisme adalah siklus yang memperkuat dan meningkatkan dirinya sendiri saat beroperasi. Untuk informasi lebih lanjut, lihat Membangun mekanisme di AWS Well-Architected Framework.

akun anggota

Semua Akun AWS selain akun manajemen yang merupakan bagian dari organisasi di AWS Organizations. Akun dapat menjadi anggota dari hanya satu organisasi pada suatu waktu.

MES

Lihat sistem eksekusi manufaktur.

Transportasi Telemetri Antrian Pesan (MQTT)

Protokol komunikasi ringan machine-to-machine (M2M), berdasarkan pola terbitkan/berlangganan, untuk perangkat IoT yang dibatasi sumber daya.

layanan mikro

Layanan kecil dan independen yang berkomunikasi melalui API yang terdefinisi dengan baik dan biasanya dimiliki oleh tim kecil yang mandiri. Misalnya, sistem asuransi mungkin mencakup layanan mikro yang memetakan kemampuan bisnis, seperti penjualan atau pemasaran, atau subdomain, seperti pembelian, klaim, atau analitik. Manfaat layanan mikro termasuk kelincahan, penskalaan yang fleksibel, penyebaran yang mudah, kode yang dapat digunakan kembali, dan ketahanan. Untuk informasi selengkapnya, lihat Mengintegrasikan layanan mikro dengan menggunakan layanan tanpa AWS server.

arsitektur microservices

Pendekatan untuk membangun aplikasi dengan komponen independen yang menjalankan setiap proses aplikasi sebagai layanan mikro. Layanan mikro ini berkomunikasi melalui antarmuka yang terdefinisi dengan baik dengan menggunakan API ringan. Setiap layanan mikro dalam arsitektur ini dapat diperbarui, digunakan, dan diskalakan untuk memenuhi permintaan fungsi tertentu dari suatu aplikasi. Untuk informasi selengkapnya, lihat Menerapkan layanan mikro di AWS.

Program Percepatan Migrasi (MAP)

AWS Program yang menyediakan dukungan konsultasi, pelatihan, dan layanan untuk membantu organisasi membangun fondasi operasional yang kuat untuk pindah ke cloud, dan untuk membantu mengimbangi biaya awal migrasi. MAP mencakup metodologi migrasi untuk mengeksekusi migrasi lama dengan cara metodis dan seperangkat alat untuk mengotomatisasi dan mempercepat skenario migrasi umum.

migrasi dalam skala

Proses memindahkan sebagian besar portofolio aplikasi ke cloud dalam gelombang, dengan lebih banyak aplikasi bergerak pada tingkat yang lebih cepat di setiap gelombang. Fase

ini menggunakan praktik terbaik dan pelajaran yang dipetik dari fase sebelumnya untuk mengimplementasikan pabrik migrasi tim, alat, dan proses untuk merampingkan migrasi beban kerja melalui otomatisasi dan pengiriman tangkas. Ini adalah fase ketiga dari strategi AWS migrasi.

pabrik migrasi

Tim lintas fungsi yang merampingkan migrasi beban kerja melalui pendekatan otomatis dan gesit. Tim pabrik migrasi biasanya mencakup operasi, analis dan pemilik bisnis, insinyur migrasi, pengembang, dan DevOps profesional yang bekerja di sprint. Antara 20 dan 50 persen portofolio aplikasi perusahaan terdiri dari pola berulang yang dapat dioptimalkan dengan pendekatan pabrik. Untuk informasi selengkapnya, lihat diskusi tentang pabrik migrasi dan panduan Pabrik Migrasi Cloud di kumpulan konten ini.

metadata migrasi

Informasi tentang aplikasi dan server yang diperlukan untuk menyelesaikan migrasi. Setiap pola migrasi memerlukan satu set metadata migrasi yang berbeda. Contoh metadata migrasi termasuk subnet target, grup keamanan, dan akun. AWS

pola migrasi

Tugas migrasi berulang yang merinci strategi migrasi, tujuan migrasi, dan aplikasi atau layanan migrasi yang digunakan. Contoh: Rehost migrasi ke Amazon EC2 AWS dengan Layanan Migrasi Aplikasi.

Penilaian Portofolio Migrasi (MPA)

Alat online yang menyediakan informasi untuk memvalidasi kasus bisnis untuk bermigrasi ke. AWS Cloud MPA menyediakan penilaian portofolio terperinci (ukuran kanan server, harga, perbandingan TCO, analisis biaya migrasi) serta perencanaan migrasi (analisis data aplikasi dan pengumpulan data, pengelompokan aplikasi, prioritas migrasi, dan perencanaan gelombang). Alat MPA (memerlukan login) tersedia gratis untuk semua AWS konsultan dan konsultan APN Partner.

Penilaian Kesiapan Migrasi (MRA)

Proses mendapatkan wawasan tentang status kesiapan cloud organisasi, mengidentifikasi kekuatan dan kelemahan, dan membangun rencana aksi untuk menutup kesenjangan yang diidentifikasi, menggunakan CAF. AWS Untuk informasi selengkapnya, lihat panduan kesiapan migrasi. MRA adalah tahap pertama dari strategi AWS migrasi.

strategi migrasi

Pendekatan yang digunakan untuk memigrasikan beban kerja ke file. AWS Cloud Untuk informasi lebih lanjut, lihat entri 7 Rs di glosarium ini dan lihat Memobilisasi organisasi Anda untuk mempercepat migrasi skala besar.

ML

Lihat pembelajaran mesin.

modernisasi

Mengubah aplikasi usang (warisan atau monolitik) dan infrastrukturnya menjadi sistem yang gesit, elastis, dan sangat tersedia di cloud untuk mengurangi biaya, mendapatkan efisiensi, dan memanfaatkan inovasi. Untuk informasi selengkapnya, lihat Strategi untuk memodernisasi aplikasi di. AWS Cloud

penilaian kesiapan modernisasi

Evaluasi yang membantu menentukan kesiapan modernisasi aplikasi organisasi; mengidentifikasi manfaat, risiko, dan dependensi; dan menentukan seberapa baik organisasi dapat mendukung keadaan masa depan aplikasi tersebut. Hasil penilaian adalah cetak biru arsitektur target, peta jalan yang merinci fase pengembangan dan tonggak untuk proses modernisasi, dan rencana aksi untuk mengatasi kesenjangan yang diidentifikasi. Untuk informasi lebih lanjut, lihat Mengevaluasi kesiapan modernisasi untuk aplikasi di. AWS Cloud

aplikasi monolitik (monolit)

Aplikasi yang berjalan sebagai layanan tunggal dengan proses yang digabungkan secara ketat. Aplikasi monolitik memiliki beberapa kelemahan. Jika satu fitur aplikasi mengalami lonjakan permintaan, seluruh arsitektur harus diskalakan. Menambahkan atau meningkatkan fitur aplikasi monolitik juga menjadi lebih kompleks ketika basis kode tumbuh. Untuk mengatasi masalah ini, Anda dapat menggunakan arsitektur microservices. Untuk informasi lebih lanjut, lihat Mengurai monolit menjadi layanan mikro.

MPA

Lihat Penilaian Portofolio Migrasi.

MQTT

Lihat Transportasi Telemetri Antrian Pesan.

klasifikasi multiclass

Sebuah proses yang membantu menghasilkan prediksi untuk beberapa kelas (memprediksi satu dari lebih dari dua hasil). Misalnya, model ML mungkin bertanya "Apakah produk ini buku, mobil, atau telepon?" atau "Kategori produk mana yang paling menarik bagi pelanggan ini?"

infrastruktur yang bisa berubah

Model yang memperbarui dan memodifikasi infrastruktur yang ada untuk beban kerja produksi. Untuk meningkatkan konsistensi, keandalan, dan prediktabilitas, AWS Well-Architected Framework merekomendasikan penggunaan infrastruktur yang tidak dapat diubah sebagai praktik terbaik.

O

OAC

Lihat kontrol akses asal.

OAI

Lihat identitas akses asal.

OCM

Lihat manajemen perubahan organisasi.

migrasi offline

Metode migrasi di mana beban kerja sumber diturunkan selama proses migrasi. Metode ini melibatkan waktu henti yang diperpanjang dan biasanya digunakan untuk beban kerja kecil dan tidak kritis.

OI

Lihat integrasi operasi.

OLA

Lihat perjanjian tingkat operasional.

migrasi online

Metode migrasi di mana beban kerja sumber disalin ke sistem target tanpa diambil offline. Aplikasi yang terhubung ke beban kerja dapat terus berfungsi selama migrasi. Metode ini melibatkan waktu henti nol hingga minimal dan biasanya digunakan untuk beban kerja produksi yang kritis.

OPC-UA

Lihat Komunikasi Proses Terbuka - Arsitektur Terpadu.

Komunikasi Proses Terbuka - Arsitektur Terpadu (OPC-UA)

Protokol komunikasi machine-to-machine (M2M) untuk otomasi industri. OPC-UA menyediakan standar interoperabilitas dengan enkripsi data, otentikasi, dan skema otorisasi.

perjanjian tingkat operasional (OLA)

Perjanjian yang menjelaskan apa yang dijanjikan kelompok TI fungsional untuk diberikan satu sama lain, untuk mendukung perjanjian tingkat layanan (SLA).

Tinjauan Kesiapan Operasional (ORR)

Daftar pertanyaan dan praktik terbaik terkait yang membantu Anda memahami, mengevaluasi, mencegah, atau mengurangi ruang lingkup insiden dan kemungkinan kegagalan. Untuk informasi lebih lanjut, lihat Ulasan Kesiapan Operasional (ORR) dalam Kerangka Kerja Well-Architected AWS .

teknologi operasional (OT)

Sistem perangkat keras dan perangkat lunak yang bekerja dengan lingkungan fisik untuk mengendalikan operasi industri, peralatan, dan infrastruktur. Di bidang manufaktur, integrasi sistem OT dan teknologi informasi (TI) adalah fokus utama untuk transformasi Industri 4.0.

integrasi operasi (OI)

Proses modernisasi operasi di cloud, yang melibatkan perencanaan kesiapan, otomatisasi, dan integrasi. Untuk informasi selengkapnya, lihat panduan integrasi operasi.

jejak organisasi

Jejak yang dibuat oleh AWS CloudTrail itu mencatat semua peristiwa untuk semua Akun AWS dalam organisasi di AWS Organizations. Jejak ini dibuat di setiap Akun AWS bagian organisasi dan melacak aktivitas di setiap akun. Untuk informasi selengkapnya, lihat Membuat jejak untuk organisasi dalam CloudTrail dokumentasi.

manajemen perubahan organisasi (OCM)

Kerangka kerja untuk mengelola transformasi bisnis utama yang mengganggu dari perspektif orang, budaya, dan kepemimpinan. OCM membantu organisasi mempersiapkan, dan transisi ke, sistem dan strategi baru dengan mempercepat adopsi perubahan, mengatasi masalah transisi,

dan mendorong perubahan budaya dan organisasi. Dalam strategi AWS migrasi, kerangka kerja ini disebut percepatan orang, karena kecepatan perubahan yang diperlukan dalam proyek adopsi cloud. Untuk informasi lebih lanjut, lihat panduan OCM.

kontrol akses asal (OAC)

Di CloudFront, opsi yang disempurnakan untuk membatasi akses untuk mengamankan konten Amazon Simple Storage Service (Amazon S3) Anda. OAC mendukung semua bucket S3 di semua Wilayah AWS, enkripsi sisi server dengan AWS KMS (SSE-KMS), dan dinamis dan permintaan ke bucket S3. `PUT` `DELETE`

identitas akses asal (OAI)

Di CloudFront, opsi untuk membatasi akses untuk mengamankan konten Amazon S3 Anda. Saat Anda menggunakan OAI, CloudFront buat prinsipal yang dapat diautentikasi oleh Amazon S3. Prinsipal yang diautentikasi dapat mengakses konten dalam bucket S3 hanya melalui distribusi tertentu. CloudFront Lihat juga OAC, yang menyediakan kontrol akses yang lebih terperinci dan ditingkatkan.

ORR

Lihat tinjauan kesiapan operasional.

OT

Lihat teknologi operasional.

keluar (jalan keluar) VPC

Dalam arsitektur AWS multi-akun, VPC yang menangani koneksi jaringan yang dimulai dari dalam aplikasi. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

P

batas izin

Kebijakan manajemen IAM yang dilampirkan pada prinsipal IAM untuk menetapkan izin maksimum yang dapat dimiliki pengguna atau peran. Untuk informasi selengkapnya, lihat Batas izin dalam dokumentasi IAM.

Informasi Identifikasi Pribadi (PII)

Informasi yang, jika dilihat secara langsung atau dipasangkan dengan data terkait lainnya, dapat digunakan untuk menyimpulkan identitas individu secara wajar. Contoh PII termasuk nama, alamat, dan informasi kontak.

PII

Lihat informasi yang dapat diidentifikasi secara pribadi.

buku pedoman

Serangkaian langkah yang telah ditentukan sebelumnya yang menangkap pekerjaan yang terkait dengan migrasi, seperti mengirimkan fungsi operasi inti di cloud. Buku pedoman dapat berupa skrip, runbook otomatis, atau ringkasan proses atau langkah-langkah yang diperlukan untuk mengoperasikan lingkungan modern Anda.

PLC

Lihat pengontrol logika yang dapat diprogram.

PLM

Lihat manajemen siklus hidup produk.

kebijakan

Objek yang dapat menentukan izin (lihat kebijakan berbasis identitas), menentukan kondisi akses (lihat kebijakanberbasis sumber daya), atau menentukan izin maksimum untuk semua akun dalam organisasi di (lihat kebijakan kontrol layanan). AWS Organizations

persistensi poliglot

Secara independen memilih teknologi penyimpanan data microservice berdasarkan pola akses data dan persyaratan lainnya. Jika layanan mikro Anda memiliki teknologi penyimpanan data yang sama, mereka dapat menghadapi tantangan implementasi atau mengalami kinerja yang buruk. Layanan mikro lebih mudah diimplementasikan dan mencapai kinerja dan skalabilitas yang lebih baik jika mereka menggunakan penyimpanan data yang paling sesuai dengan kebutuhan mereka. Untuk informasi selengkapnya, lihat Mengaktifkan persistensi data di layanan mikro.

penilaian portofolio

Proses menemukan, menganalisis, dan memprioritaskan portofolio aplikasi untuk merencanakan migrasi. Untuk informasi selengkapnya, lihat Mengevaluasi kesiapan migrasi.

predikat

Kondisi kueri yang mengembalikan `true` atau`false`, biasanya terletak di `WHERE` klausa.

predikat pushdown

Teknik pengoptimalan kueri database yang menyaring data dalam kueri sebelum transfer. Ini mengurangi jumlah data yang harus diambil dan diproses dari database relasional, dan meningkatkan kinerja kueri.

kontrol preventif

Kontrol keamanan yang dirancang untuk mencegah suatu peristiwa terjadi. Kontrol ini adalah garis pertahanan pertama untuk membantu mencegah akses tidak sah atau perubahan yang tidak diinginkan ke jaringan Anda. Untuk informasi selengkapnya, lihat Kontrol pencegahan dalam Menerapkan kontrol keamanan pada. AWS

principal

Entitas AWS yang dapat melakukan tindakan dan mengakses sumber daya. Entitas ini biasanya merupakan pengguna root untuk Akun AWS, peran IAM, atau pengguna. Untuk informasi selengkapnya, lihat Prinsip dalam istilah dan konsep Peran dalam dokumentasi IAM.

Privasi oleh Desain

Pendekatan dalam rekayasa sistem yang memperhitungkan privasi di seluruh proses rekayasa.

zona host pribadi

Container yang menyimpan informasi tentang bagaimana Anda ingin Amazon Route 53 merespons kueri DNS untuk domain dan subdomainnya dalam satu atau beberapa VPC. Untuk informasi selengkapnya, lihat Bekerja dengan zona yang dihosting pribadi di dokumentasi Route 53.

kontrol proaktif

Kontrol keamanan yang dirancang untuk mencegah penyebaran sumber daya yang tidak sesuai. Kontrol ini memindai sumber daya sebelum disediakan. Jika sumber daya tidak sesuai dengan kontrol, maka itu tidak disediakan. Untuk informasi selengkapnya, lihat panduan referensi Kontrol dalam AWS Control Tower dokumentasi dan lihat Kontrol proaktif dalam Menerapkan kontrol keamanan pada AWS.

manajemen siklus hidup produk (PLM)

Manajemen data dan proses untuk suatu produk di seluruh siklus hidupnya, mulai dari desain, pengembangan, dan peluncuran, melalui pertumbuhan dan kematangan, hingga penurunan dan penghapusan.

lingkungan produksi

Lihat lingkungan.

pengontrol logika yang dapat diprogram (PLC)

Di bidang manufaktur, komputer yang sangat andal dan mudah beradaptasi yang memantau mesin dan mengotomatiskan proses manufaktur.

pseudonimisasi

Proses penggantian pengidentifikasi pribadi dalam kumpulan data dengan nilai placeholder. Pseudonimisasi dapat membantu melindungi privasi pribadi. Data pseudonim masih dianggap sebagai data pribadi.

terbitkan/berlangganan (pub/sub)

Pola yang memungkinkan komunikasi asinkron antara layanan mikro untuk meningkatkan skalabilitas dan daya tanggap. Misalnya, dalam MES berbasis layanan mikro, layanan mikro dapat mempublikasikan pesan peristiwa ke saluran yang dapat berlangganan layanan mikro lainnya. Sistem dapat menambahkan layanan mikro baru tanpa mengubah layanan penerbitan.

Q

rencana kueri

Serangkaian langkah, seperti instruksi, yang digunakan untuk mengakses data dalam sistem database relasional SQL.

regresi rencana kueri

Ketika pengoptimal layanan database memilih rencana yang kurang optimal daripada sebelum perubahan yang diberikan ke lingkungan database. Hal ini dapat disebabkan oleh perubahan statistik, kendala, pengaturan lingkungan, pengikatan parameter kueri, dan pembaruan ke mesin database.

R

Matriks RACI

Lihat bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI).

ransomware

Perangkat lunak berbahaya yang dirancang untuk memblokir akses ke sistem komputer atau data sampai pembayaran dilakukan.

Matriks RASCI

Lihat bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI).

RCAC

Lihat kontrol akses baris dan kolom.

replika baca

Salinan database yang digunakan untuk tujuan read-only. Anda dapat merutekan kueri ke replika baca untuk mengurangi beban pada database utama Anda.

arsitek ulang

Lihat 7 Rs.

tujuan titik pemulihan (RPO)

Jumlah waktu maksimum yang dapat diterima sejak titik pemulihan data terakhir. Ini menentukan apa yang dianggap sebagai hilangnya data yang dapat diterima antara titik pemulihan terakhir dan gangguan layanan.

tujuan waktu pemulihan (RTO)

Penundaan maksimum yang dapat diterima antara gangguan layanan dan pemulihan layanan.

refactor

Lihat 7 Rs.

Wilayah

Kumpulan AWS sumber daya di wilayah geografis. Masing-masing Wilayah AWS terisolasi dan independen dari yang lain untuk memberikan toleransi kesalahan, stabilitas, dan ketahanan. Untuk informasi selengkapnya, lihat Menentukan Wilayah AWS akun yang dapat digunakan.

regresi

Teknik ML yang memprediksi nilai numerik. Misalnya, untuk memecahkan masalah “Berapa harga rumah ini akan dijual?” Model ML dapat menggunakan model regresi linier untuk memprediksi harga jual rumah berdasarkan fakta yang diketahui tentang rumah (misalnya, luas persegi).

rehost

Lihat 7 Rs.

melepaskan

Dalam proses penyebaran, tindakan mempromosikan perubahan pada lingkungan produksi.

memindahkan

Lihat 7 Rs.

memplatform ulang

Lihat 7 Rs.

pembelian kembali

Lihat 7 Rs.

ketahanan

Kemampuan aplikasi untuk melawan atau pulih dari gangguan. Ketersediaan tinggi dan pemulihan bencana adalah pertimbangan umum ketika merencanakan ketahanan di. AWS Cloud Untuk informasi lebih lanjut, lihat AWS Cloud Ketahanan.

kebijakan berbasis sumber daya

Kebijakan yang dilampirkan ke sumber daya, seperti bucket Amazon S3, titik akhir, atau kunci enkripsi. Jenis kebijakan ini menentukan prinsip mana yang diizinkan mengakses, tindakan yang didukung, dan kondisi lain yang harus dipenuhi.

matriks yang bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI)

Matriks yang mendefinisikan peran dan tanggung jawab untuk semua pihak yang terlibat dalam kegiatan migrasi dan operasi cloud. Nama matriks berasal dari jenis tanggung jawab yang didefinisikan dalam matriks: bertanggung jawab (R), akuntabel (A), dikonsultasikan (C), dan diinformasikan (I). Jenis dukungan (S) adalah opsional. Jika Anda menyertakan dukungan, matriks disebut matriks RASCI, dan jika Anda mengecualikannya, itu disebut matriks RACI.

kontrol responsif

Kontrol keamanan yang dirancang untuk mendorong remediasi efek samping atau penyimpangan dari garis dasar keamanan Anda. Untuk informasi selengkapnya, lihat Kontrol responsif dalam Menerapkan kontrol keamanan pada AWS.

melestarikan

Lihat 7 Rs.

pensiun

Lihat 7 Rs.

rotasi

Proses memperbarui rahasia secara berkala untuk membuatnya lebih sulit bagi penyerang untuk mengakses kredensil.

kontrol akses baris dan kolom (RCAC)

Penggunaan ekspresi SQL dasar dan fleksibel yang telah menetapkan aturan akses. RCAC terdiri dari izin baris dan topeng kolom.

RPO

Lihat tujuan titik pemulihan.

RTO

Lihat tujuan waktu pemulihan.

buku runbook

Satu set prosedur manual atau otomatis yang diperlukan untuk melakukan tugas tertentu. Ini biasanya dibangun untuk merampingkan operasi berulang atau prosedur dengan tingkat kesalahan yang tinggi.

D

SAML 2.0

Standar terbuka yang digunakan oleh banyak penyedia identitas (IdPs). Fitur ini memungkinkan sistem masuk tunggal gabungan (SSO), sehingga pengguna dapat masuk ke AWS Management Console atau memanggil operasi AWS API tanpa Anda harus membuat pengguna di IAM untuk

semua orang di organisasi Anda. Untuk informasi lebih lanjut tentang federasi berbasis SAMP 2.0, lihat Tentang federasi berbasis SAMP 2.0 dalam dokumentasi IAM.

PENIPUAN

Lihat kontrol pengawasan dan akuisisi data.

SCP

Lihat kebijakan kontrol layanan.

Rahasia

Dalam AWS Secrets Manager, informasi rahasia atau terbatas, seperti kata sandi atau kredensi pengguna, yang Anda simpan dalam bentuk terenkripsi. Ini terdiri dari nilai rahasia dan metadata-nya. Nilai rahasia dapat berupa biner, string tunggal, atau beberapa string. Untuk informasi selengkapnya, lihat Apa yang ada di rahasia Secrets Manager? dalam dokumentasi Secrets Manager.

kontrol keamanan

Pagar pembatas teknis atau administratif yang mencegah, mendeteksi, atau mengurangi kemampuan pelaku ancaman untuk mengeksploitasi kerentanan keamanan. Ada empat jenis kontrol keamanan utama: preventif, detektif, responsif, dan proaktif.

pengerasan keamanan

Proses mengurangi permukaan serangan untuk membuatnya lebih tahan terhadap serangan. Ini dapat mencakup tindakan seperti menghapus sumber daya yang tidak lagi diperlukan, menerapkan praktik keamanan terbaik untuk memberikan hak istimewa paling sedikit, atau menonaktifkan fitur yang tidak perlu dalam file konfigurasi.

sistem informasi keamanan dan manajemen acara (SIEM)

Alat dan layanan yang menggabungkan sistem manajemen informasi keamanan (SIM) dan manajemen acara keamanan (SEM). Sistem SIEM mengumpulkan, memantau, dan menganalisis data dari server, jaringan, perangkat, dan sumber lain untuk mendeteksi ancaman dan pelanggaran keamanan, dan untuk menghasilkan peringatan.

otomatisasi respons keamanan

Tindakan yang telah ditentukan dan diprogram yang dirancang untuk secara otomatis merespons atau memulihkan peristiwa keamanan. Otomatisasi ini berfungsi sebagai kontrol keamanan detektif atau responsif yang membantu Anda menerapkan praktik terbaik AWS keamanan. Contoh

tindakan respons otomatis termasuk memodifikasi grup keamanan VPC, menambal instans Amazon EC2, atau memutar kredensil.

enkripsi sisi server

Enkripsi data di tujuannya, oleh Layanan AWS yang menerimanya.

kebijakan kontrol layanan (SCP)

Kebijakan yang menyediakan kontrol terpusat atas izin untuk semua akun di organisasi. AWS Organizations SCP menentukan pagar pembatas atau menetapkan batasan pada tindakan yang dapat didelegasikan oleh administrator kepada pengguna atau peran. Anda dapat menggunakan SCP sebagai daftar izin atau daftar penolakan, untuk menentukan layanan atau tindakan mana yang diizinkan atau dilarang. Untuk informasi selengkapnya, lihat Kebijakan kontrol layanan dalam AWS Organizations dokumentasi.

titik akhir layanan

URL titik masuk untuk file Layanan AWS. Anda dapat menggunakan endpoint untuk terhubung secara terprogram ke layanan target. Untuk informasi selengkapnya, lihat Layanan AWS titik akhir di Referensi Umum AWS.

perjanjian tingkat layanan (SLA)

Perjanjian yang menjelaskan apa yang dijanjikan tim TI untuk diberikan kepada pelanggan mereka, seperti waktu kerja dan kinerja layanan.

indikator tingkat layanan (SLI)

Pengukuran aspek kinerja layanan, seperti tingkat kesalahan, ketersediaan, atau throughputnya.

tujuan tingkat layanan (SLO)

Metrik target yang mewakili kesehatan layanan, yang diukur dengan indikator tingkat layanan.

model tanggung jawab bersama

Model yang menjelaskan tanggung jawab yang Anda bagikan AWS untuk keamanan dan kepatuhan cloud. AWS bertanggung jawab atas keamanan cloud, sedangkan Anda bertanggung jawab atas keamanan di cloud. Untuk informasi selengkapnya, lihat Model tanggung jawab bersama.

SIEM

Lihat informasi keamanan dan sistem manajemen acara.

titik kegagalan tunggal (SPOF)

Kegagalan dalam satu komponen penting dari aplikasi yang dapat mengganggu sistem.

SLA

Lihat perjanjian tingkat layanan.

SLI

Lihat indikator tingkat layanan.

SLO

Lihat tujuan tingkat layanan.

split-and-seed model

Pola untuk menskalakan dan mempercepat proyek modernisasi. Ketika fitur baru dan rilis produk didefinisikan, tim inti berpisah untuk membuat tim produk baru. Ini membantu meningkatkan kemampuan dan layanan organisasi Anda, meningkatkan produktivitas pengembang, dan mendukung inovasi yang cepat. Untuk informasi lebih lanjut, lihat Pendekatan bertahap untuk memodernisasi aplikasi di. AWS Cloud

SPOF

Lihat satu titik kegagalan.

skema bintang

Struktur organisasi database yang menggunakan satu tabel fakta besar untuk menyimpan data transaksional atau terukur dan menggunakan satu atau lebih tabel dimensi yang lebih kecil untuk menyimpan atribut data. Struktur ini dirancang untuk digunakan dalam gudang data atau untuk tujuan intelijen bisnis.

pola ara pencekik

Pendekatan untuk memodernisasi sistem monolitik dengan menulis ulang secara bertahap dan mengganti fungsionalitas sistem sampai sistem warisan dapat dinonaktifkan. Pola ini menggunakan analogi pohon ara yang tumbuh menjadi pohon yang sudah mapan dan akhirnya mengatasi dan menggantikan inangnya. Pola ini diperkenalkan oleh Martin Fowler sebagai cara untuk mengelola risiko saat menulis ulang sistem monolitik. Untuk contoh cara menerapkan pola ini, lihat Memodernisasi layanan web Microsoft ASP.NET (ASMX) lama secara bertahap menggunakan container dan Amazon API Gateway.

subnet

Rentang alamat IP dalam VPC Anda. Subnet harus berada di Availability Zone tunggal.

kontrol pengawasan dan akuisisi data (SCADA)

Di bidang manufaktur, sistem yang menggunakan perangkat keras dan perangkat lunak untuk memantau aset fisik dan operasi produksi.

enkripsi simetris

Algoritma enkripsi yang menggunakan kunci yang sama untuk mengenkripsi dan mendekripsi data.

pengujian sintetis

Menguji sistem dengan cara yang mensimulasikan interaksi pengguna untuk mendeteksi potensi masalah atau untuk memantau kinerja. Anda dapat menggunakan Amazon CloudWatch Synthetics untuk membuat tes ini.

T

tag

Pasangan nilai kunci yang bertindak sebagai metadata untuk mengatur sumber daya Anda. AWS Tanda dapat membantu Anda mengelola, mengidentifikasi, mengatur, dan memfilter sumber daya. Untuk informasi selengkapnya, lihat Menandai AWS sumber daya Anda.

variabel target

Nilai yang Anda coba prediksi dalam ML yang diawasi. Ini juga disebut sebagai variabel hasil. Misalnya, dalam pengaturan manufaktur, variabel target bisa menjadi cacat produk.

daftar tugas

Alat yang digunakan untuk melacak kemajuan melalui runbook. Daftar tugas berisi ikhtisar runbook dan daftar tugas umum yang harus diselesaikan. Untuk setiap tugas umum, itu termasuk perkiraan jumlah waktu yang dibutuhkan, pemilik, dan kemajuan.

lingkungan uji

Lihat lingkungan.

pelatihan

Untuk menyediakan data bagi model ML Anda untuk dipelajari. Data pelatihan harus berisi jawaban yang benar. Algoritma pembelajaran menemukan pola dalam data pelatihan yang memetakan atribut data input ke target (jawaban yang ingin Anda prediksi). Ini menghasilkan model ML yang menangkap pola-pola ini. Anda kemudian dapat menggunakan model ML untuk membuat prediksi pada data baru yang Anda tidak tahu targetnya.

gerbang transit

Hub transit jaringan yang dapat Anda gunakan untuk menghubungkan VPC dan jaringan lokal Anda. Untuk informasi selengkapnya, lihat Apa itu gateway transit dalam AWS Transit Gateway dokumentasi.

alur kerja berbasis batang

Pendekatan di mana pengembang membangun dan menguji fitur secara lokal di cabang fitur dan kemudian menggabungkan perubahan tersebut ke cabang utama. Cabang utama kemudian dibangun untuk pengembangan, praproduksi, dan lingkungan produksi, secara berurutan.

akses tepercaya

Memberikan izin ke layanan yang Anda tentukan untuk melakukan tugas di organisasi Anda di dalam AWS Organizations dan di akunnya atas nama Anda. Layanan tepercaya menciptakan peran terkait layanan di setiap akun, ketika peran itu diperlukan, untuk melakukan tugas manajemen untuk Anda. Untuk informasi selengkapnya, lihat Menggunakan AWS Organizations dengan AWS layanan lain dalam AWS Organizations dokumentasi.

penyetelan

Untuk mengubah aspek proses pelatihan Anda untuk meningkatkan akurasi model ML. Misalnya, Anda dapat melatih model ML dengan membuat set pelabelan, menambahkan label, dan kemudian mengulangi langkah-langkah ini beberapa kali di bawah pengaturan yang berbeda untuk mengoptimalkan model.

tim dua pizza

Sebuah DevOps tim kecil yang bisa Anda beri makan dengan dua pizza. Ukuran tim dua pizza memastikan peluang terbaik untuk berkolaborasi dalam pengembangan perangkat lunak.

U

waswas

Sebuah konsep yang mengacu pada informasi yang tidak tepat, tidak lengkap, atau tidak diketahui yang dapat merusak keandalan model ML prediktif. Ada dua jenis ketidakpastian: ketidakpastian epistemik disebabkan oleh data yang terbatas dan tidak lengkap, sedangkan ketidakpastian aleatorik disebabkan oleh kebisingan dan keacakan yang melekat dalam data. Untuk informasi lebih lanjut, lihat panduan Mengukur ketidakpastian dalam sistem pembelajaran mendalam.

tugas yang tidak terdiferensiasi

Juga dikenal sebagai angkat berat, pekerjaan yang diperlukan untuk membuat dan mengoperasikan aplikasi tetapi itu tidak memberikan nilai langsung kepada pengguna akhir atau memberikan keunggulan kompetitif. Contoh tugas yang tidak terdiferensiasi termasuk pengadaan, pemeliharaan, dan perencanaan kapasitas.

lingkungan atas

Lihat lingkungan.

V

menyedot debu

Operasi pemeliharaan database yang melibatkan pembersihan setelah pembaruan tambahan untuk merebut kembali penyimpanan dan meningkatkan kinerja.

kendali versi

Proses dan alat yang melacak perubahan, seperti perubahan kode sumber dalam repositori.

Peering VPC

Koneksi antara dua VPC yang memungkinkan Anda merutekan lalu lintas dengan menggunakan alamat IP pribadi. Untuk informasi selengkapnya, lihat Apa itu peering VPC di dokumentasi VPC Amazon.

kerentanan

Kelemahan perangkat lunak atau perangkat keras yang membahayakan keamanan sistem.

W

cache hangat

Cache buffer yang berisi data terkini dan relevan yang sering diakses. Instance database dapat membaca dari cache buffer, yang lebih cepat daripada membaca dari memori utama atau disk.

data hangat

Data yang jarang diakses. Saat menanyakan jenis data ini, kueri yang cukup lambat biasanya dapat diterima.

fungsi jendela

Fungsi SQL yang melakukan perhitungan pada sekelompok baris yang berhubungan dengan catatan saat ini. Fungsi jendela berguna untuk memproses tugas, seperti menghitung rata-rata bergerak atau mengakses nilai baris berdasarkan posisi relatif dari baris saat ini.

beban kerja

Kumpulan sumber daya dan kode yang memberikan nilai bisnis, seperti aplikasi yang dihadapi pelanggan atau proses backend.

aliran kerja

Grup fungsional dalam proyek migrasi yang bertanggung jawab atas serangkaian tugas tertentu. Setiap alur kerja independen tetapi mendukung alur kerja lain dalam proyek. Misalnya, alur kerja portofolio bertanggung jawab untuk memprioritaskan aplikasi, perencanaan gelombang, dan mengumpulkan metadata migrasi. Alur kerja portofolio mengirimkan aset ini ke alur kerja migrasi, yang kemudian memigrasikan server dan aplikasi.

CACING

Lihat menulis sekali, baca banyak.

WQF

Lihat Kerangka Kualifikasi Beban Kerja AWS.

tulis sekali, baca banyak (WORM)

Model penyimpanan yang menulis data satu kali dan mencegah data dihapus atau dimodifikasi. Pengguna yang berwenang dapat membaca data sebanyak yang diperlukan, tetapi mereka tidak dapat mengubahnya. Infrastruktur penyimpanan data ini dianggap tidak dapat diubah.

Z

eksploitasi zero-day

Serangan, biasanya malware, yang memanfaatkan kerentanan zero-day.

kerentanan zero-day

Cacat atau kerentanan yang tak tanggung-tanggung dalam sistem produksi. Aktor ancaman dapat menggunakan jenis kerentanan ini untuk menyerang sistem. Pengembang sering menyadari kerentanan sebagai akibat dari serangan tersebut.

aplikasi zombie

Aplikasi yang memiliki CPU rata-rata dan penggunaan memori di bawah 5 persen. Dalam proyek migrasi, adalah umum untuk menghentikan aplikasi ini.

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