Menerapkan PostgreSQL terkelola untuk aplikasi SaaS multi-tenant AWS

AWS Panduan Preskriptif

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Panduan Preskriptif: Menerapkan PostgreSQL terkelola untuk aplikasi SaaS multi-tenant AWS

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

Pengantar 1
Hasil bisnis yang ditargetkan 1
Memilih database untuk aplikasi SaaS 3
Memilih antara Amazon RDS dan Aurora 5
Multi-tenant Model partisi SaaS untuk PostgreSQL 7
Model silo PostgreSQL 8
Model kolam PostgreSQL 9
Model jembatan PostgreSQL 11
Matriks keputusan 13
Rekomendasi keamanan tingkat baris 29
Ketersediaan PostgreSQL untuk model kolam 31
Praktik terbaik 33
Bandingkan AWS opsi untuk PostgreSQL terkelola 33
Pilih model partisi SaaS multi-penyewa 33
Gunakan keamanan tingkat baris untuk model partisi SaaS kolam 33
Pertanyaan yang Sering Diajukan 34
Opsi PostgreSQL terkelola mana yang ditawarkan? AWS 34
Layanan mana yang optimal untuk aplikasi SaaS? 34
Persyaratan unik mana yang harus saya pertimbangkan jika saya memutuskan untuk menggunakan database PostgreSQL dengan aplikasi SaaS multi-penyewa? 34
Model mana yang dapat saya gunakan untuk mempertahankan isolasi data penyewa dengan PostgreSQL? 34
Bagaimana cara mempertahankan isolasi data penyewa dengan satu database PostgreSQL yang dibagikan di beberapa penyewa? 35
Langkah selanjutnya 36
Sumber daya 37
Referensi 37
Partner 37
Riwayat dokumen 38
..... xxxix

Menerapkan PostgreSQL terkelola untuk aplikasi SaaS multi-tenant AWS

Tabby Ward dan Thomas Davis, Amazon Web Services ()AWS

April 2024 (riwayat dokumen)

Ketika Anda memilih database untuk menyimpan data operasional, penting untuk mempertimbangkan bagaimana data harus disusun, pertanyaan mana yang akan dijawabnya, seberapa cepat akan memberikan jawaban, dan ketahanan platform data itu sendiri. Selain pertimbangan umum ini adalah implikasi perangkat lunak sebagai layanan (SaaS) untuk data operasional, seperti isolasi kinerja, keamanan penyewa, dan karakteristik unik serta pola desain yang khas dari data untuk aplikasi SaaS multi-penyewa. Panduan ini membahas bagaimana faktor-faktor ini berlaku untuk menggunakan database PostgreSQL di Amazon Web AWS Services () sebagai penyimpanan data operasional utama untuk aplikasi SaaS multi-penyewa. Secara khusus, panduan ini berfokus pada dua opsi PostgreSQL AWS terkelola: Amazon Aurora PostgreSQL Compatible Edition dan Amazon Relational Database Service (Amazon RDS) untuk PostgreSQL.

Hasil bisnis yang ditargetkan

Panduan ini memberikan analisis terperinci tentang praktik terbaik untuk aplikasi SaaS multi-penyewa menggunakan Aurora PostgreSQL kompatibel dan Amazon RDS for PostgreSQL. Kami menyarankan Anda menggunakan pola dan konsep desain yang disediakan dalam panduan ini untuk menginformasikan dan membakukan implementasi Aurora PostgreSQL yang kompatibel dengan Aurora atau Amazon RDS for PostgreSQL untuk aplikasi SaaS multi-penyewa Anda.

Panduan preskriptif ini membantu mencapai hasil bisnis berikut:

- Memilih opsi PostgreSQL AWS terkelola yang paling optimal untuk kasus penggunaan Anda — Panduan ini membandingkan opsi relasional dan non-relasional untuk penggunaan database dengan aplikasi SaaS. Ini juga membahas kasus penggunaan mana yang paling optimal untuk Aurora PostgreSQL kompatibel dan Amazon RDS for PostgreSQL. Informasi ini akan membantu dalam memilih opsi terbaik untuk aplikasi SaaS Anda.
- Penegakan praktik terbaik SaaS melalui adopsi model partisi SaaS - Panduan ini membahas dan membandingkan tiga model partisi SaaS luas yang berlaku untuk sistem manajemen basis data PostgreSQL (DBMS): kumpulan, jembatan, dan model silo, dan variasinya. Pendekatan ini

menangkap praktik terbaik SaaS dan memberikan fleksibilitas saat merancang aplikasi SaaS. Penegakan model partisi SaaS adalah bagian penting dari melestarikan praktik terbaik.

- Penggunaan RLS yang efektif dalam model partisi SaaS kolam - Keamanan tingkat baris (RLS) mendukung penegakan isolasi data penyewa dalam satu tabel PostgreSQL dengan membatasi baris yang dapat dilihat berdasarkan pengguna atau variabel konteks. Saat Anda menggunakan model partisi kolam, RLS diperlukan untuk mencegah akses penyewa silang.

Memilih database untuk aplikasi SaaS

Untuk banyak aplikasi SaaS multi-tenant, memilih database operasional dapat disuling menjadi pilihan antara database relasional dan non-relasional, atau kombinasi keduanya. Untuk membuat keputusan, pertimbangkan persyaratan dan karakteristik data aplikasi tingkat tinggi ini:

- Model data aplikasi
- Pola akses untuk data
- Persyaratan latensi basis data
- Integritas data dan persyaratan integritas transaksional (atomisitas, konsistensi, isolasi, dan daya tahan, atau ACID)
- Ketersediaan Lintas Wilayah dan persyaratan pemulihan

Tabel berikut mencantumkan persyaratan dan karakteristik data aplikasi, dan mendiskusikannya dalam konteks penawaran AWS database: Aurora PostgreSQL kompatibel dan Amazon RDS for PostgreSQL (relasional), dan Amazon DynamoDB (non-relasional). Anda dapat mereferensikan matriks ini ketika Anda mencoba memutuskan antara penawaran basis data operasional relasional dan non-relasional.

Basis Data	Persyaratan dan karakteristik data aplikasi SaaS				
	Model data	Pola akses	Persyaratan latensi	Integritas data dan transaksional	Ketersediaan dan pemulihan Lintas Wilayah
Relasional (Aurora PostgreSQL kompatibel dan Amazon RDS untuk PostgreSQL)	Relasional atau sangat dinormalisasi.	Tidak harus direncanakan secara menyeluruh sebelumnya.	Lebih disukai toleransi latensi yang lebih tinggi; dapat mencapai latensi yang lebih rendah	Data tinggi dan integritas transaksional dipertahankan secara default.	Di Amazon RDS, Anda dapat membuat replika baca untuk penskalaan dan failover

			secara default dengan Aurora dan dengan menerapkan replika baca, caching, dan fitur serupa.		lintas wilayah. Aurora sebagian besar mengotomatiskan proses ini. Untuk konfigurasi aktif-aktif di beberapa Wilayah AWS, Anda dapat menggunakan penerusan tulis bersama dengan database global Aurora.
Non-relasional (Amazon DynamoDB)	Biasanya didenormalisasi. Database ini memanfaatkan pola untuk pemodelan many-to-many hubungan, item besar, dan data deret waktu.	Semua pola akses (query) untuk data harus dipahami secara menyeluruh sebelum model data diproduksi.	Latensi sangat rendah dengan opsi seperti Amazon DynamoDB Accelerator (DAX) mampu meningkatkan kinerja lebih jauh.	Integritas transaksional opsional dengan mengorbankan kinerja. Masalah integritas data dialihkan ke aplikasi.	Pemulihan lintas wilayah yang mudah dan konfigurasi aktif-aktif dengan tabel global. (Kepatuhan ACID hanya dapat dicapai dalam satu AWS Wilayah.)

Beberapa aplikasi SaaS multi-tenant mungkin memiliki model data unik atau keadaan khusus yang lebih baik dilayani oleh database yang tidak termasuk dalam tabel sebelumnya. Misalnya, kumpulan data deret waktu, kumpulan data yang sangat terhubung, atau memelihara buku besar transaksi terpusat mungkin memerlukan menggunakan jenis database yang berbeda. Menganalisis semua kemungkinan berada di luar cakupan panduan ini. Untuk daftar lengkap penawaran AWS database dan bagaimana mereka dapat memenuhi kasus penggunaan yang berbeda pada tingkat tinggi, lihat bagian Database dari whitepaper Ikhtisar Amazon Web Services.

Sisa dari panduan ini berfokus pada layanan database AWS relasional yang mendukung PostgreSQL: Amazon RDS dan Aurora PostgreSQL kompatibel. DynamoDB memerlukan pendekatan yang berbeda untuk mengoptimalkan aplikasi SaaS, yang berada di luar cakupan panduan ini. Untuk informasi selengkapnya tentang DynamoDB, lihat posting blog Mempartisi Data SaaS AWS Multi-Penyewa yang Dikumpulkan dengan Amazon DynamoDB.

Memilih antara Amazon RDS dan Aurora

Dalam kebanyakan kasus, sebaiknya gunakan Aurora PostgreSQL yang kompatibel dengan Amazon RDS for PostgreSQL. Tabel berikut menunjukkan faktor-faktor yang harus Anda pertimbangkan ketika memutuskan antara dua opsi ini.

Komponen DBMS	Amazon RDS untuk PostgreSQL	Kompatibel dengan Aurora PostgreSQL
Skalabilitas	Jeda replikasi menit, maksimal 5 replika baca	Kelambatan replikasi kurang dari satu menit (biasanya kurang dari 1 detik dengan database global), maksimum 15 replika baca
Pemulihan kerusakan	Pos pemeriksaan terpisah 5 menit (secara default), dapat memperlambat kinerja database	Pemulihan asinkron dengan thread paralel untuk pemulihan cepat
Kegagalan	60-120 detik selain waktu pemulihan crash	Biasanya sekitar 30 detik (termasuk pemulihan crash)

Komponen DBMS	Amazon RDS untuk PostgreSQL	Kompatibel dengan Aurora PostgreSQL
Penyimpanan	IOPS maksimum 256.000	IOPS dibatasi hanya oleh ukuran dan kapasitas instans Aurora
Ketersediaan tinggi dan pemulihan bencana	Dua Availability Zone dengan instance siaga, failover lintas wilayah untuk membaca replika atau salinan cadangan	Tiga Availability Zone secara default, failover lintas wilayah dengan database global Aurora, tulis penerusan untuk konfigurasi aktif-aktif Wilayah AWS
Cadangan	Selama jendela cadangan, dapat memengaruhi kinerja	Pencadangan inkremental otomatis, tidak ada dampak kinerja
Kelas instance database	Lihat daftar kelas instans Amazon RDS	Lihat daftar kelas instance Aurora

Dalam semua kategori yang dijelaskan dalam tabel sebelumnya, Aurora PostgreSQL kompatibel biasanya pilihan yang lebih baik. Namun, Amazon RDS for PostgreSQL mungkin masih masuk akal untuk beban kerja kecil hingga menengah, karena memiliki lebih banyak pilihan kelas instans yang mungkin memberikan opsi yang lebih hemat biaya dengan mengorbankan set fitur Aurora yang lebih kuat.

Multi-tenant Model partisi SaaS untuk PostgreSQL

Metode terbaik untuk mencapai multi-tenancy tergantung pada persyaratan untuk aplikasi SaaS Anda. Bagian berikut menunjukkan model partisi untuk berhasil menerapkan multi-tenancy di PostgreSQL.

> **Note**
>
> Model yang dibahas di bagian ini berlaku untuk Amazon RDS for PostgreSQL dan Aurora. PostgreSQL-Compatible Referensi ke PostgreSQL di bagian ini berlaku untuk kedua layanan.

Ada tiga model tingkat tinggi yang dapat Anda gunakan di PostgreSQL untuk partisi SaaS: silo, jembatan, dan kolam. Gambar berikut merangkum trade-off antara model silo dan kolam renang. Model jembatan adalah hibrida dari model silo dan kolam renang.

Model partisi	Keuntungan	Kekurangan
Silo	• Penyelarasan kepatuhan • Tidak ada dampak lintas penyewa • Tenant-level penyetelan • Tenant-level ketersediaan	• Kelincahan yang dikompromikan • Tidak ada manajemen terpusat • Kompleksitas penyebaran • Biaya
Kolam	• Kelincahan • Optimalisasi biaya • Manajemen terpusat • Penyebaran yang disederhanakan	• Cross-tenant dampak • Tantangan kepatuhan • Semua atau tidak ada ketersediaan
Jembatan	• Beberapa penyelarasan kepatuhan • Kelincahan • Optimalisasi biaya	• Beberapa tantangan kepatuhan

Model partisi	Keuntungan	Kekurangan
	• Manajemen terpusat	• Ketersediaan semua atau tidak sama sekali (kebanyakan) • Cross-tenant dampak • Kompleksitas penyebaran

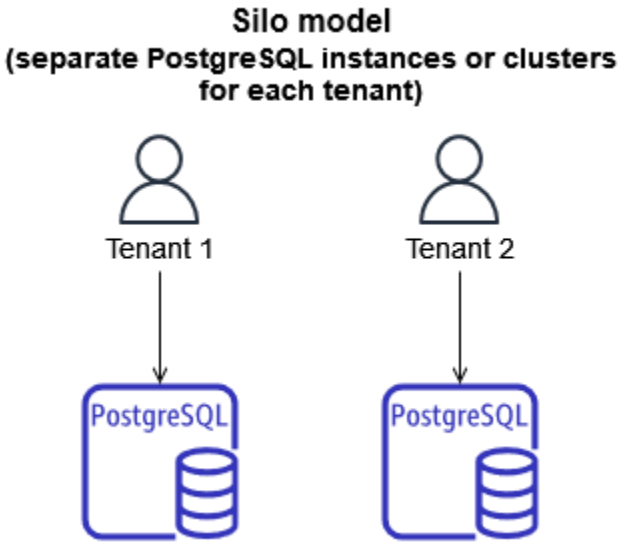
Bagian berikut membahas setiap model secara lebih rinci.

Model partisi:

- Model silo PostgreSQL
- Model kolam PostgreSQL
- Model jembatan PostgreSQL
- Matriks keputusan

Model silo PostgreSQL

Model silo diimplementasikan dengan menyediakan instance PostgreSQL untuk setiap penyewa dalam aplikasi. Model silo unggul dalam kinerja penyewa dan isolasi keamanan, dan sepenuhnya menghilangkan fenomena tetangga yang bising. Fenomena tetangga yang bising terjadi ketika satu penyewa menggunakan sistem mempengaruhi kinerja penyewa lain. Model silo memungkinkan Anda menyesuaikan kinerja secara khusus untuk setiap penyewa dan berpotensi membatasi pemadaman pada silo penyewa tertentu. Namun, apa yang umumnya mendorong adopsi model silo adalah batasan keamanan dan peraturan yang ketat. Kendala ini dapat dimotivasi oleh pelanggan SaaS. Misalnya, pelanggan SaaS mungkin menuntut agar data mereka diisolasi karena kendala internal, dan penyedia SaaS mungkin menawarkan layanan seperti itu dengan biaya tambahan.

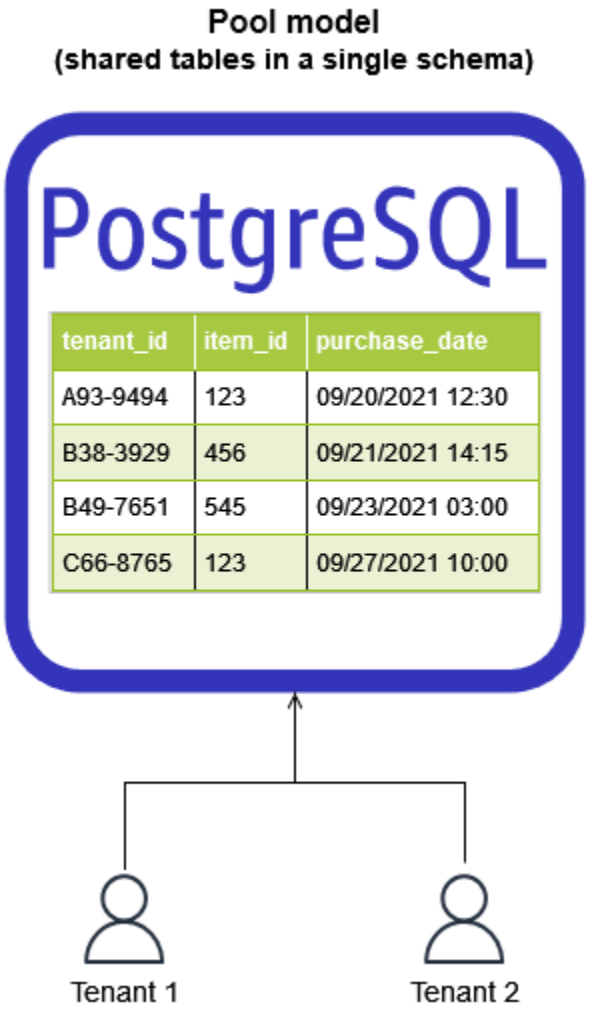


Meskipun model silo mungkin diperlukan dalam kasus-kasus tertentu, ia memiliki banyak kelemahan. Seringkali sulit untuk menggunakan model silo dengan cara yang hemat biaya, karena mengelola konsumsi sumber daya di beberapa instance PostgreSQL dapat menjadi rumit. Selain itu, sifat terdistribusi dari beban kerja database dalam model ini membuatnya lebih sulit untuk mempertahankan pandangan terpusat dari aktivitas penyewa. Mengelola begitu banyak beban kerja yang dioperasikan secara independen meningkatkan overhead operasional dan administratif. Model silo juga membuat orientasi penyewa lebih rumit dan memakan waktu, karena Anda harus menyediakan sumber daya khusus penyewa. Selain itu, seluruh sistem SaaS bisa lebih sulit untuk diskalakan, karena jumlah instans PostgreSQL khusus penyewa yang terus meningkat akan menuntut lebih banyak waktu operasional untuk dikelola. Satu pertimbangan terakhir adalah bahwa aplikasi atau lapisan akses data harus mempertahankan pemetaan penyewa ke instance PostgreSQL terkait mereka, yang menambah kompleksitas penerapan model ini.

Model kolam PostgreSQL

Model pool diimplementasikan dengan menyediakan satu instance PostgreSQL (Amazon RDS atau Aurora) dan menggunakan keamanan tingkat baris (RLS) untuk mempertahankan isolasi data penyewa. Kebijakan RLS membatasi baris mana dalam tabel yang dikembalikan oleh `SELECT` kueri atau baris mana yang terpengaruh oleh`INSERT`,`UPDATE`, dan perintah. `DELETE` Model pool memusatkan semua data penyewa dalam satu skema PostgreSQL, sehingga secara signifikan lebih hemat biaya dan membutuhkan lebih sedikit overhead operasional untuk mempertahankannya. Pemantauan solusi ini juga jauh lebih sederhana karena sentralisasinya. Namun, pemantauan dampak spesifik penyewa dalam model kumpulan biasanya memerlukan beberapa instrumentasi

tambahan dalam aplikasi. Ini karena PostgreSQL secara default tidak mengetahui penyewa mana yang mengkonsumsi sumber daya. Orientasi penyewa disederhanakan karena tidak diperlukan infrastruktur baru. Kelincahan ini membuatnya lebih mudah untuk mencapai alur kerja orientasi penyewa yang cepat dan otomatis.



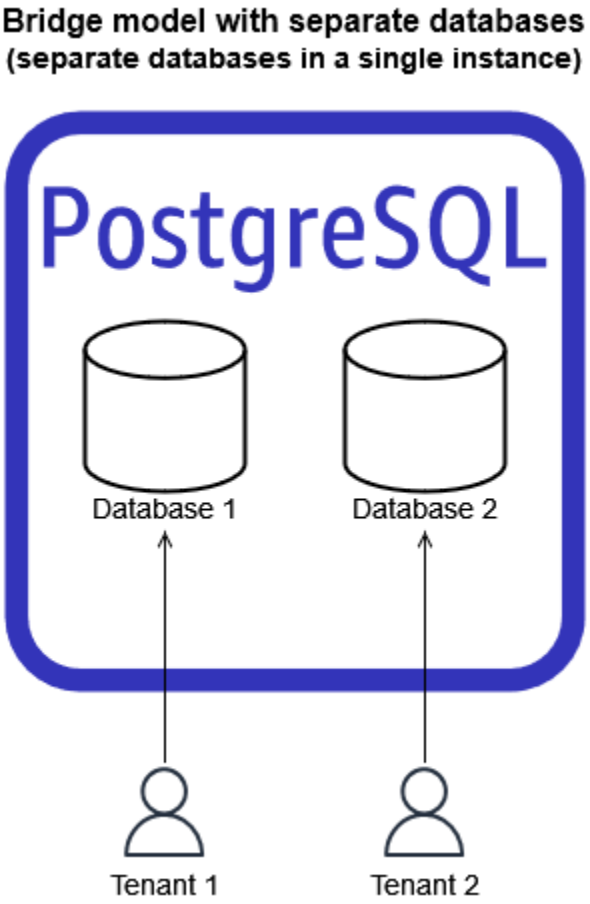
Meskipun model kolam umumnya lebih hemat biaya dan lebih mudah dikelola, model ini memiliki beberapa kelemahan. Fenomena tetangga yang bising tidak dapat sepenuhnya dihilangkan dalam model kolam renang. Namun, hal ini dapat dikurangi dengan memastikan bahwa sumber daya yang sesuai tersedia pada instance PostgreSQL dan dengan menggunakan strategi untuk mengurangi beban di PostgreSQL, seperti membongkar kueri untuk membaca replika atau ke Amazon. ElastiCache Pemantauan yang efektif juga berperan dalam menanggapi masalah isolasi kinerja penyewa, karena instrumentasi aplikasi dapat mencatat dan memantau aktivitas khusus

penyewa. Terakhir, beberapa pelanggan SaaS mungkin tidak menemukan pemisahan logis yang disediakan oleh RLS cukup dan mungkin meminta tindakan isolasi tambahan.

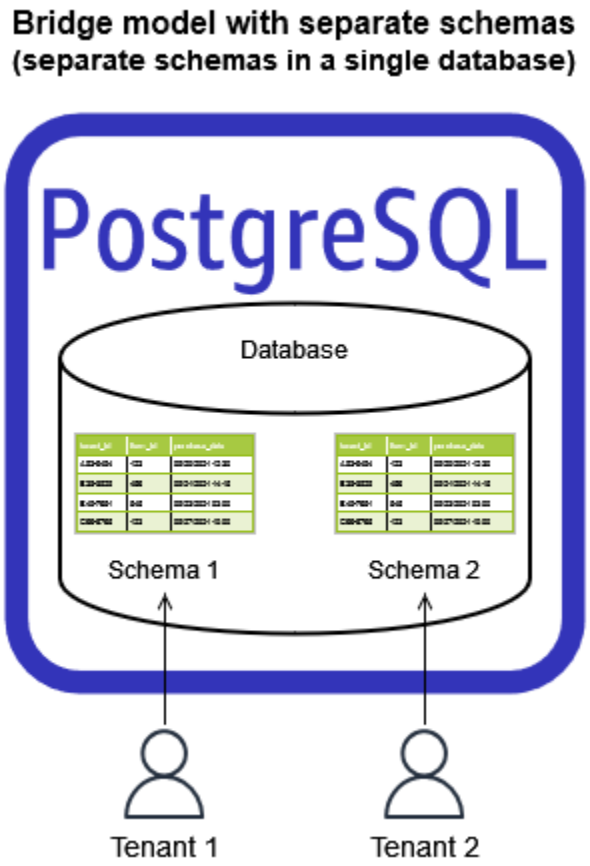
Model jembatan PostgreSQL

Model jembatan PostgreSQL adalah kombinasi dari pendekatan gabungan dan silo. Seperti model gabungan, Anda menyediakan satu instance PostgreSQL untuk setiap penyewa. Untuk mempertahankan isolasi data penyewa, Anda menggunakan konstruksi logis PostgreSQL. Dalam diagram berikut, database PostgreSQL digunakan untuk memisahkan data secara logis.

> **Note**
>
> Database PostgreSQL tidak merujuk ke Amazon RDS terpisah untuk instans DB Amazon RDS for PostgreSQL atau Aurora. PostgreSQL-Compatible Sebaliknya, ini mengacu pada konstruksi logis dari sistem manajemen database PostgreSQL untuk memisahkan data.



Anda juga dapat mengimplementasikan model jembatan dengan menggunakan database PostgreSQL tunggal, dengan skema khusus penyewa di setiap database, seperti yang diilustrasikan dalam diagram berikut.



Model jembatan menderita masalah isolasi kinerja tetangga yang bising dan penyewa yang sama dengan model kolam renang. Ini juga menimbulkan beberapa overhead operasional dan penyediaan tambahan dengan mengharuskan database atau skema terpisah untuk disediakan berdasarkan per-penyewa. Diperlukan pemantauan yang efektif untuk merespons dengan cepat masalah kinerja penyewa. Ini juga membutuhkan instrumentasi aplikasi untuk memantau penggunaan khusus penyewa. Secara keseluruhan, model bridge dapat dilihat sebagai alternatif dari RLS yang sedikit menambah upaya orientasi penyewa dengan membutuhkan database atau skema PostgreSQL baru. Seperti halnya model silo, aplikasi atau lapisan akses data harus mempertahankan pemetaan penyewa ke database atau skema PostgreSQL terkait.

Matriks keputusan

Untuk memutuskan model partisi SaaS multi-tenant mana yang harus Anda gunakan dengan PostgreSQL, lihat matriks keputusan berikut. Matriks menganalisis empat opsi partisi ini:

- Silo — Sebuah instance PostgreSQL terpisah atau cluster untuk setiap penyewa.

- Bridge dengan database terpisah - Database terpisah untuk setiap penyewa dalam satu instance PostgreSQL atau cluster.
- Jembatan dengan skema terpisah - Skema terpisah untuk setiap penyewa dalam database PostgreSQL tunggal, dalam satu instance PostgreSQL atau cluster.
- Pool — Tabel bersama untuk penyewa dalam satu contoh dan skema.

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
Kasus penggunaan	Isolasi data dengan kontrol penuh penggunaan sumber daya adalah persyaratan utama, atau Anda memiliki penyewa yang sangat besar dan sangat sensitif terhadap kinerja.	Isolasi data adalah persyaratan utama, dan referensi silang data penyewa terbatas atau tidak diperlukan.	Jumlah penyewa moderat dengan jumlah data moderat. Ini adalah model yang disukai jika Anda harus referensi silang data penyewa.	Sejumlah besar penyewa dengan lebih sedikit data per penyewa.
Kelincahan orientasi penyewa baru	Sangat lambat. (Sebuah instance atau cluster baru diperlukan untuk setiap penyewa.)	Cukup lambat. (Membutuhkan pembuatan database baru untuk setiap penyewa untuk menyimpan objek skema.)	Cukup lambat. (Membutuhkan pembuatan skema baru untuk setiap penyewa untuk menyimpan objek.)	Opsi tercepat. (Diperlukan pengaturan minimal.)

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
Upaya dan efisiensi konfigurasi kumpulan koneksi basis data	Diperlukan upaya yang signifikan. (Satu kolam koneksi per penyewa.) Kurang efisien. (Tidak ada berbagi koneksi database antara penyewa.)	Diperlukan upaya yang signifikan. (Satu konfigurasi kumpulan koneksi per penyewa kecuali Anda menggunakan Amazon RDS Proxy.) Kurang efisien. (Tidak ada berbagi koneksi database antara penyewa dan jumlah total koneksi. Penggunaan di semua penyewa terbatas berdasarkan kelas instans DB.)	Lebih sedikit usaha yang dibutuhkan. (Satu konfigurasi kumpulan koneksi untuk semua penyewa.) Cukup efisien. (Koneksi digunakan kembali melalui `SET SCHEMA` perintah `SET ROLE` or dalam mode kumpulan sesi saja. `SET`perintah juga menyebabkan penyematan sesi saat menggunakan Amazon RDS Proxy, tetapi kumpulan koneksi klien dapat dihilangkan dan koneksi langsung dapat dibuat untuk	Usaha paling sedikit yang dibutuhkan. Paling efisien. (Satu kumpulan koneksi untuk semua penyewa dan penggunaan kembali koneksi yang efisien di semua penyewa. Batas koneksi database didasarkan pada kelas instans DB.)

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
			setiap permintaan efisiensi.)	

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
Pemeliharaan basis data (manajemen vakum) dan penggunaan sumber daya	Manajemen yang lebih sederhana.	Kompleksitas sedang. (Mungkin menyebabkan konsumsi sumber daya yang tinggi, karena pekerja vakum harus dimulai untuk setiap database setelahnya `avacuum_naptime` , yang mengarah ke penggunaan CPU peluncur autovacuum yang tinggi. Mungkin juga ada overhead tambahan yang terkait dengan menyedot debu tabel katalog sistem PostgreSQL untuk setiap database.)	Tabel katalog sistem PostgreSQL besar. (`pg_catalog` Ukuran total dalam puluhan GB, tergantung pada jumlah penyewa dan hubungan. Kemungkinan memerlukan modifikasi pada parameter terkait penyedotan debu untuk mengontrol kembung tabel.)	Tabel mungkin besar, tergantung pada jumlah penyewa dan data per penyewa. (Kemungkinan memerlukan modifikasi pada parameter terkait penyedotan debu untuk mengontrol kembung tabel.)

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
Upaya manajemen ekstensi	Upaya yang signifikan (untuk setiap database dalam contoh terpisah).	Upaya yang signifikan (di setiap tingkat database).	Upaya minimal (satu kali dalam database umum).	Upaya minimal (satu kali dalam database umum).
Ubah upaya penerapan	Upaya yang signifikan. (Connect ke setiap instance terpisah dan luncurkan perubahan.)	Upaya yang signifikan. (Connect ke setiap database dan skema, dan luncurkan perubahan.)	Upaya moderat. (Connect ke database umum dan luncurkan perubahan untuk setiap skema.)	Usaha minimal. (Connect ke database umum dan luncurkan perubahan.)
Ubah penyebaran — ruang lingkup dampak	Minimal. (Penyewa tunggal terpengaruh.)	Minimal. (Penyewa tunggal terpengaruh.)	Minimal. (Penyewa tunggal terpengaruh.)	Sangat besar. (Semua penyewa terpengaruh.)

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
Manajemen dan upaya kinerja kueri	Kinerja kueri yang dapat dikelola.	Kinerja kueri yang dapat dikelola.	Kinerja kueri yang dapat dikelola.	Upaya signifikan mungkin diperlukan untuk mempertahankan kinerja kueri. (Seiring waktu, kueri mungkin berjalan lebih lambat karena peningkatan ukuran tabel. Anda dapat menggunakan partisi tabel dan sharding database untuk mempertahankan kinerja.)
Cross-tenant dampak sumber daya	Tidak ada dampak. (Tidak ada pembagian sumber daya di antara penyewa.)	Dampak sedang. (Penyewa berbagi sumber daya umum seperti CPU dan memori instance.)	Dampak sedang. (Penyewa berbagi sumber daya umum seperti CPU dan memori instance.)	Dampak berat. (Penyewa saling mempengaruhi dalam hal sumber daya, konflik kunci, dan sebagainya.)

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
Tenant-level penyetelan (misalnya, pembuatan indeks tambahan per penyewa atau penyesuaian parameter DB untuk penyewa tertentu)	Mungkin.	Agak mungkin. (Schema-level perubahan dapat dilakukan untuk setiap penyewa, tetapi parameter database bersifat global di semua penyewa.)	Agak mungkin. (Schema-level perubahan dapat dilakukan untuk setiap penyewa, tetapi parameter database bersifat global di semua penyewa.)	Tidak mungkin. (Tabel dibagikan oleh semua penyewa.)

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
Upaya menyeimbangkan kembali untuk penyewa yang peka terhadap kinerja	Minimal. (Tidak perlu menyeimbangkan kembali. Skala server dan I/O sumber daya untuk menangani skenario ini.)	Sedang. (Gunakan replikasi logis atau pg_dump untuk mengekspor database, tetapi waktu henti mungkin panjang tergantung pada ukuran data. Anda dapat menggunakan fitur database yang dapat diangkut di Amazon RDS for PostgreSQL untuk menyalin database antar instance dengan lebih cepat.)	Sedang tetapi kemungkinan melibatkan waktu henti yang lama. (Gunakan replikasi logis atau pg_dump untuk mengekspor skema, tetapi waktu henti mungkin panjang tergantung pada ukuran data.)	Signifikan, karena semua penyewa berbagi tabel yang sama. (Sharding database membutuhkan penyalinan semuanya ke instance lain dan langkah tambahan untuk membersihkan data penyewa.) Kemungkinan besar membutuhkan perubahan dalam logika aplikasi.

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
Waktu henti basis data untuk peningkatan versi utama	Waktu henti standar. (Tergantung pada ukuran katalog sistem PostgreSQL.)	Kemungkinan downtime lebih lama. (Tergantung pada ukuran katalog sistem, waktunya akan bervariasi. Tabel katalog sistem PostgreSQL juga diduplikasi di seluruh database)	Kemungkinan downtime lebih lama. (Tergantung pada ukuran katalog sistem PostgreSQL, waktunya akan bervariasi.)	Waktu henti standar. (Tergantung pada ukuran katalog sistem PostgreSQL.)
Administrasi overhead (misalnya, untuk analisis log database atau pemantauan pekerjaan cadangan)	Upaya yang signifikan	Usaha minimal.	Usaha minimal.	Usaha minimal.

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
Tenant-level ketersediaan	Tertinggi. (Setiap penyewa gagal dan pulih secara mandiri.)	Lingkup dampak yang lebih tinggi. (Semua penyewa gagal dan pulih bersama jika terjadi masalah perangkat keras atau sumber daya.)	Lingkup dampak yang lebih tinggi. (Semua penyewa gagal dan pulih bersama jika terjadi masalah perangkat keras atau sumber daya.)	Lingkup dampak yang lebih tinggi. (Semua penyewa gagal dan pulih bersama jika terjadi masalah perangkat keras atau sumber daya.)
Tenant-level upaya pencadangan dan pemulihan	Usaha paling sedikit. (Setiap penyewa dapat didukung dan dipulihkan secara independen.)	Upaya moderat. (Gunakan ekspor dan impor logis untuk setiap penyewa. Beberapa pengkodean dan otomatisasi diperlukan.)	Upaya moderat. (Gunakan ekspor dan impor logis untuk setiap penyewa. Beberapa pengkodean dan otomatisasi diperlukan.)	Upaya yang signifikan. (Semua penyewa berbagi tabel yang sama.)
Tenant-level upaya pemulihan point-in-time	Usaha minimal. (Gunakan pemulihan waktu point-in dengan menggunakan snapshot, atau gunakan backtracking di Amazon Aurora.)	Upaya moderat. (Gunakan pemulihan snapshot, diikuti oleh export/import. Namun, ini akan menjadi operasi yang lambat.)	Upaya moderat. (Gunakan pemulihan snapshot, diikuti oleh export/import. Namun, ini akan menjadi operasi yang lambat.)	Upaya dan kompleksitas yang signifikan.

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
Nama skema seragam	Nama skema yang sama untuk setiap penyewa.	Nama skema yang sama untuk setiap penyewa.	Skema yang berbeda untuk setiap penyewa.	Skema umum.
Per-tenant kustomisasi (misalnya, kolom tabel tambahan untuk penyewa tertentu)	Mungkin.	Mungkin.	Mungkin.	Rumit (karena semua penyewa berbagi tabel yang sama).
Efisiensi manajemen katalog pada lapisan pemetaan relasional objek (ORM) (misalnya, Ruby)	Efisien (karena koneksi klien khusus untuk penyewa).	Efisien (karena koneksi klien khusus untuk database).	Cukup efisien. (Tergantung pada ORM yang digunakan, model user/role keamanan, dan `search_path` konfigurasi, klien terkadang menyimpan metadata untuk semua penyewa, yang mengarah ke penggunaan memori koneksi DB yang tinggi.)	Efisien (karena semua penyewa berbagi tabel yang sama).

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
Upaya pelaporan penyewa konsolidasi	Upaya yang signifikan. (Anda harus menggunakan pembungkus data asing [FDW] untuk mengkonsolidasikan data di semua penyewa atau mengekstrak, mengubah, dan memuat [ETL] ke database pelaporan lain.)	Upaya yang signifikan. (Anda harus menggunakan FDW untuk mengkonsolidasikan data di semua penyewa atau ETL ke database pelaporan lain.)	Upaya moderat. (Anda dapat mengumpulkan data di semua skema dengan menggunakan serikat pekerja.)	Usaha minimal. (Semua data penyewa ada dalam tabel yang sama, jadi pelaporannya sederhana.)
Tenant-specific instance read-only untuk pelaporan (misalnya, berdasarkan langganan)	Usaha paling sedikit. (Buat replika baca.)	Upaya moderat. (Anda dapat menggunakan replikasi logis atau AWS Database Migration Service [AWS DMS] untuk mengonfigurasi.)	Upaya moderat. (Anda dapat menggunakan replikasi logis atau AWS DMS untuk mengkonfigurasi.)	Rumit (karena semua penyewa berbagi tabel yang sama).

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
Isolasi data	Terbaik.	Lebih baik. (Anda dapat mengelola izin tingkat database dengan menggunakan peran PostgreSQL.)	Lebih baik. (Anda dapat mengelola izin tingkat skema dengan menggunakan peran PostgreSQL.)	Lebih buruk. (Karena semua penyewa berbagi tabel yang sama, Anda harus menerapkan fitur seperti keamanan tingkat baris [RLS] untuk isolasi penyewa.)
Tenant-specific kunci enkripsi penyimpanan	Mungkin. (Setiap cluster PostgreSQL dapat memiliki kunci AWS KMS[] AWS Key Management Service sendiri untuk enkripsi penyimpanan.)	Tidak mungkin. (Semua penyewa berbagi kunci KMS yang sama untuk enkripsi penyimpanan.)	Tidak mungkin. (Semua penyewa berbagi kunci KMS yang sama untuk enkripsi penyimpanan.)	Tidak mungkin. (Semua penyewa berbagi kunci KMS yang sama untuk enkripsi penyimpanan.)
Menggunakan AWS Identity and Access Management (IAM) untuk otentikasi database untuk setiap penyewa	Mungkin.	Mungkin.	Kemungkinan (dengan memiliki pengguna PostgreSQL terpisah untuk setiap skema).	Tidak mungkin (karena tabel dibagikan oleh semua penyewa).

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
Biaya infrastru ktur	Tertinggi (karena tidak ada yang dibagikan).	Sedang.	Sedang.	Terendah.
Duplikasi data dan penggunaan penyimpanan	Agregat tertinggi di semua penyewa. (Tabel katalog sistem PostgreSQL dan data statis dan umum aplikasi digandaka n di semua penyewa.)	Agregat tertinggi di semua penyewa. (Tabel katalog sistem PostgreSQL dan data statis dan umum aplikasi digandaka n di semua penyewa.)	Sedang. (Data statis dan umum aplikasi dapat berada dalam skema umum dan diakses oleh penyewa lain.)	Minimal. (Tidak ada duplikasi data. Data statis dan umum aplikasi dapat berada dalam skema yang sama.)
Tenant-centric pemantauan (dengan cepat mencari tahu penyewa mana yang menyebabk an masalah)	Usaha paling sedikit. (Karena setiap penyewa dipantau secara terpisah, mudah untuk memeriksa aktivitas penyewa tertentu.)	Upaya moderat. (Karena semua penyewa berbagi sumber daya fisik yang sama, Anda harus menerapka n penyaring an tambahan untuk memeriksa aktivitas penyewa tertentu.)	Upaya moderat. (Karena semua penyewa berbagi sumber daya fisik yang sama, Anda harus menerapka n penyaring an tambahan untuk memeriksa aktivitas penyewa tertentu.)	Upaya yang signifikan. (Karena semua penyewa berbagi semua sumber daya, termasuk tabel, Anda harus menggunak an tangkapan variabel bind untuk memeriksa penyewa mana kueri SQL tertentu milik.)

	Silo	Jembatan dengan database terpisah	Jembatan dengan skema terpisah	Kolam
Manajemen dan health/activity pemantauan terpusat	Upaya yang signifikan (untuk mengatur pemantauan pusat dan pusat komando pusat).	Upaya moderat (karena semua penyewa berbagi contoh yang sama).	Upaya moderat (karena semua penyewa berbagi contoh yang sama).	Upaya minimal (karena semua penyewa berbagi sumber daya yang sama, termasuk skema).
Peluang pengenal objek (OID) dan ID transaksi (XID) sampul	Minimal.	Tinggi. (Karena OID, XID adalah penghitung cluster PostgreSQL tunggal dan mungkin ada masalah penyedot debu secara efektif di seluruh database fisik).	Sedang. (Karena OID, XID adalah penghitung cluster PostgreSQL tunggal).	Tinggi. (Misalnya, satu tabel dapat mencapai batas TOAST OID sebesar 4 miliar, tergantung pada jumlah kolom out-of-line.)

Rekomendasi keamanan tingkat baris

Keamanan tingkat baris (RLS) diperlukan untuk mempertahankan isolasi data penyewa dalam model gabungan dengan PostgreSQL. RLS memusatkan penegakan kebijakan isolasi di tingkat database dan menghilangkan beban mempertahankan isolasi ini dari pengembang perangkat lunak. Cara paling umum untuk mengimplementasikan RLS adalah dengan mengaktifkan fitur ini di PostgreSQL DBMS. RLS melibatkan penyaringan akses ke baris data berdasarkan nilai dalam kolom tertentu. Anda dapat menggunakan dua metode untuk memfilter akses ke data:

- Kolom data tertentu dalam tabel dibandingkan dengan nilai pengguna PostgreSQL saat ini. Nilai dalam kolom yang setara dengan pengguna PostgreSQL yang masuk dapat diakses oleh pengguna tersebut.
- Kolom data tertentu dalam tabel dibandingkan dengan nilai variabel runtime yang ditetapkan oleh aplikasi. Nilai dalam kolom yang setara dengan variabel runtime dapat diakses selama sesi itu.

Opsi kedua lebih disukai, karena opsi pertama memerlukan pembuatan pengguna PostgreSQL baru untuk setiap penyewa. Sebagai gantinya, aplikasi SaaS yang menggunakan PostgreSQL harus bertanggung jawab untuk mengatur konteks khusus penyewa saat runtime saat menanyakan PostgreSQL. Ini akan memiliki efek menegakkan RLS. Anda juga dapat mengaktifkan RLS secara table-by-table dasar. Sebagai praktik terbaik, Anda harus mengaktifkan RLS pada semua tabel yang berisi data penyewa.

Contoh berikut membuat dua tabel dan memungkinkan RLS. Contoh ini membandingkan kolom data dengan nilai variabel runtime. `app.current_tenant`

```
-- Create a table for our tenants with indexes on the primary key and the tenant’s name
CREATE TABLE tenant (
    tenant_id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
    name VARCHAR(255) UNIQUE,
    status VARCHAR(64) CHECK (status IN ('active', 'suspended', 'disabled')),
    tier VARCHAR(64) CHECK (tier IN ('gold', 'silver', 'bronze'))
);

-- Create a table for users of a tenant
CREATE TABLE tenant_user (
    user_id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
    tenant_id UUID NOT NULL REFERENCES tenant (tenant_id) ON DELETE RESTRICT,
    email VARCHAR(255) NOT NULL UNIQUE,
```

```
    given_name VARCHAR(255) NOT NULL CHECK (given_name <> ''),
    family_name VARCHAR(255) NOT NULL CHECK (family_name <> '')
);

-- Turn on RLS
ALTER TABLE tenant ENABLE ROW LEVEL SECURITY;

-- Restrict read and write actions so tenants can only see their rows
-- Cast the UUID value in tenant_id to match the type current_setting
-- This policy implies a WITH CHECK that matches the USING clause
CREATE POLICY tenant_isolation_policy ON tenant
USING (tenant_id = current_setting('app.current_tenant')::UUID);

-- And do the same for the tenant users
ALTER TABLE tenant_user ENABLE ROW LEVEL SECURITY;

CREATE POLICY tenant_user_isolation_policy ON tenant_user
USING (tenant_id = current_setting('app.current_tenant')::UUID);
```

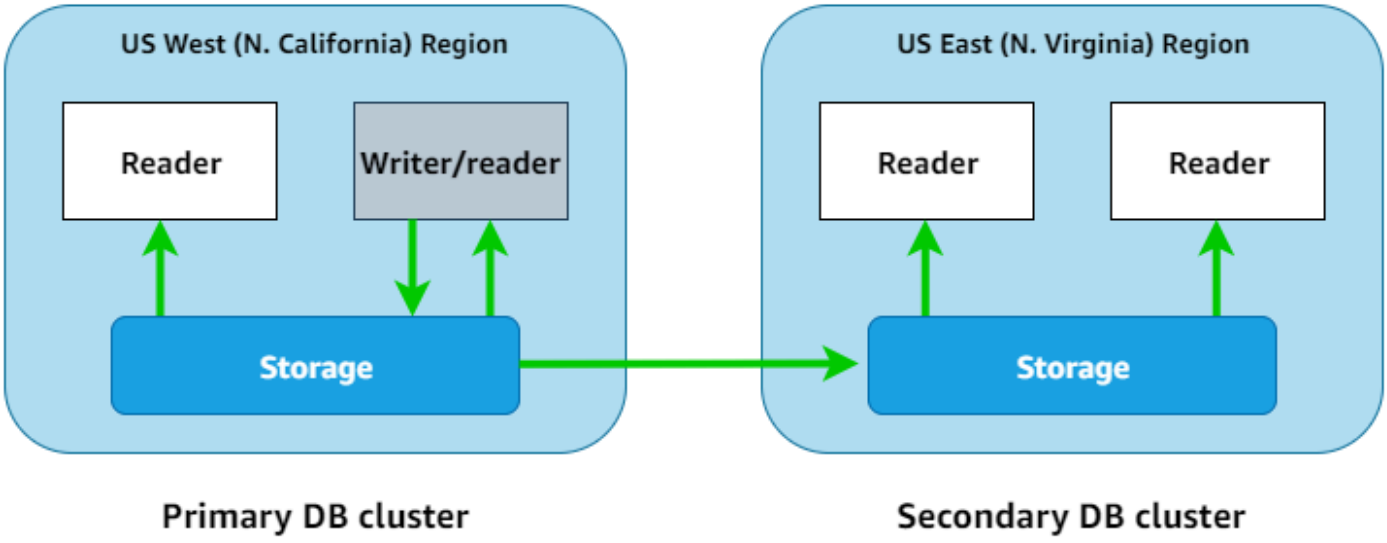
Untuk informasi lebih lanjut, lihat posting blog isolasi data multi-tenant dengan PostgreSQL Row Level Security. Tim Pabrik AWS SaaS juga memiliki beberapa contoh GitHub untuk membantu dalam mengimplementasikan RLS.

Ketersediaan PostgreSQL untuk model kolam

Model kolam menurut sifatnya hanya memiliki satu instance PostgreSQL. Oleh karena itu, merancang aplikasi Anda untuk ketersediaan tinggi sangat penting. Kegagalan atau pemadaman database yang dikumpulkan mengakibatkan aplikasi Anda terdegradasi atau menjadi tidak dapat diakses oleh semua penyewa Anda.

Instans Amazon RDS for PostgreSQL DB dapat dibuat redundan di dua Availability Zone dengan mengaktifkan fitur ketersediaan tinggi. Untuk informasi selengkapnya, lihat Ketersediaan tinggi (Multi-AZ) untuk Amazon RDS dalam dokumentasi Amazon RDS. Untuk failover lintas wilayah, Anda dapat membuat replika baca di Wilayah yang berbeda. AWS (Replika baca ini harus dipromosikan sebagai bagian dari proses failover.) Selain itu, Anda dapat mereplikasi cadangan yang direplikasi di seluruh AWS Wilayah untuk pemulihan. Untuk informasi selengkapnya, lihat Mereplikasi pencadangan otomatis ke AWS Wilayah lain dalam dokumentasi Amazon RDS.

Aurora PostgreSQL-Compatible secara otomatis mencadangkan data dengan cara yang dapat mempertahankan kegagalan beberapa Availability Zone. (Lihat Ketersediaan tinggi untuk Amazon Aurora di dokumentasi Aurora.) Untuk membuat Aurora lebih tangguh dan pulih lebih cepat, Anda dapat membuat replika baca Aurora di Availability Zone lainnya. Anda dapat menggunakan database global Aurora untuk mereplikasi data menjadi lima AWS Wilayah tambahan untuk pemulihan Lintas wilayah dan failover otomatis. (Lihat Menggunakan database global Amazon Aurora dalam dokumentasi Aurora.) Selain itu, Anda dapat mengaktifkan penerusan tulis dengan database global Aurora untuk mencapai ketersediaan tinggi di beberapa. Wilayah AWS



Terlepas dari apakah Anda menggunakan Amazon RDS untuk PostgreSQL PostgreSQL-Compatible atau Aurora, sebaiknya Anda menerapkan fitur ketersediaan tinggi untuk mengurangi dampak pemadaman untuk semua aplikasi SaaS multi-tenant yang menggunakan model pool.

Praktik terbaik

Bagian ini mencantumkan beberapa takeaways tingkat tinggi dari panduan ini. Untuk diskusi terperinci tentang setiap poin, ikuti tautan ke bagian yang sesuai.

Bandingkan AWS opsi untuk PostgreSQL terkelola

AWS menawarkan dua cara utama untuk menjalankan PostgreSQL di lingkungan yang dikelola. (Dalam konteks ini, dikelola berarti bahwa infrastruktur PostgreSQL dan DBMS sebagian atau seluruhnya didukung oleh layanan.) AWS Opsi AWS PostgreSQL yang dikelola memiliki manfaat mengotomatiskan pencadangan, failover, pengoptimalan, dan beberapa administrasi PostgreSQL. Sebagai opsi terkelola, AWS menawarkan Amazon Aurora PostgreSQL yang kompatibel dengan Edition dan Amazon Relational Database Service (Amazon RDS) untuk PostgreSQL. Anda dapat memilih pilihan terbaik dari dua model ini dengan menganalisis kasus penggunaan PostgreSQL Anda. Untuk informasi selengkapnya, lihat bagian Memilih antara Amazon RDS dan Aurora dalam panduan ini.

Pilih model partisi SaaS multi-penyewa

Anda dapat memilih dari tiga model partisi SaaS yang berlaku untuk PostgreSQL: silo, bridge, dan pool. Setiap model memiliki kelebihan dan kekurangan, dan Anda harus memilih model yang paling optimal tergantung pada kasus penggunaan Anda. Amazon RDS untuk PostgreSQL dan Aurora PostgreSQL yang kompatibel mendukung ketiga model. Memilih model sangat penting untuk menjaga isolasi data penyewa di aplikasi SaaS Anda. Untuk diskusi mendetail tentang model-model ini, lihat bagian Model partisi SaaS multi-penyewa untuk PostgreSQL dalam panduan ini.

Gunakan keamanan tingkat baris untuk model partisi SaaS kolam

Keamanan tingkat baris (RLS) diperlukan untuk mempertahankan isolasi data penyewa dalam model kumpulan dengan PostgreSQL. Hal ini karena tidak ada pemisahan logis antara infrastruktur, database PostgreSQL, atau skema berdasarkan per-tenant dalam model pool. RLS memusatkan penegakan kebijakan isolasi di tingkat database dan menghilangkan beban mempertahankan isolasi ini dari pengembang perangkat lunak. Anda dapat menggunakan RLS untuk membatasi operasi database untuk penyewa tertentu. Untuk informasi selengkapnya dan contoh, lihat bagian Rekomendasi keamanan tingkat baris dalam panduan ini.

Pertanyaan yang Sering Diajukan

Bagian ini memberikan jawaban atas pertanyaan yang sering diajukan tentang penerapan PostgreSQL terkelola dalam aplikasi SaaS multi-penyewa.

Opsi PostgreSQL terkelola mana yang ditawarkan? AWS

AWS menawarkan Amazon Aurora PostgreSQL yang kompatibel dan Amazon Relational Database Service (Amazon RDS) untuk PostgreSQL. AWS juga memiliki katalog luas penawaran database terkelola.

Layanan mana yang optimal untuk aplikasi SaaS?

Anda dapat menggunakan Aurora PostgreSQL yang kompatibel dan Amazon RDS for PostgreSQL untuk aplikasi SaaS dan semua model partisi SaaS yang dibahas dalam panduan ini. Kedua layanan ini memiliki perbedaan dalam skalabilitas, pemulihan kerusakan, failover, opsi penyimpanan, ketersediaan tinggi, pemulihan bencana, cadangan, dan kelas instance yang tersedia untuk setiap opsi. Pilihan optimal akan tergantung pada kasus penggunaan spesifik Anda. Gunakan matriks keputusan dalam panduan ini untuk memilih opsi terbaik untuk kasus penggunaan Anda.

Persyaratan unik mana yang harus saya pertimbangkan jika saya memutuskan untuk menggunakan database PostgreSQL dengan aplikasi SaaS multi-penyewa?

Seperti halnya penyimpanan data yang digunakan dengan aplikasi SaaS, pertimbangan terpenting adalah metode untuk mempertahankan isolasi data penyewa. Seperti yang dibahas dalam panduan ini, ada beberapa cara Anda dapat mencapai isolasi data penyewa dengan penawaran PostgreSQL AWS terkelola. Selain itu, Anda harus mempertimbangkan isolasi kinerja berdasarkan per-penyewa untuk implementasi PostgreSQL apa pun.

Model mana yang dapat saya gunakan untuk mempertahankan isolasi data penyewa dengan PostgreSQL?

Anda dapat menggunakan model silo, jembatan, dan kumpulan sebagai strategi partisi SaaS untuk mempertahankan isolasi data penyewa. Untuk diskusi tentang model-model ini dan bagaimana

mereka dapat diterapkan ke PostgreSQL, lihat bagian Multi-tenant SaaS model partisi untuk PostgreSQL dalam panduan ini.

Bagaimana cara mempertahankan isolasi data penyewa dengan satu database PostgreSQL yang dibagikan di beberapa penyewa?

PostgreSQL mendukung fitur keamanan tingkat baris (RLS) yang dapat Anda gunakan untuk menerapkan isolasi data penyewa dalam satu database atau instance PostgreSQL. Selain itu, Anda dapat menyediakan database PostgreSQL terpisah per penyewa dalam satu contoh, atau membuat skema berdasarkan per-penyewa untuk mencapai tujuan ini. Untuk keuntungan dan kerugian dari pendekatan ini, lihat bagian Rekomendasi keamanan tingkat baris dalam panduan ini.

Langkah selanjutnya

AWS menawarkan dua opsi untuk mengoperasikan PostgreSQL terkelola: Aurora PostgreSQL kompatibel dan Amazon RDS for PostgreSQL. Kami menyarankan Anda mengevaluasi dua layanan dan memilih opsi yang paling mendukung kasus penggunaan spesifik Anda untuk aplikasi SaaS multi-penyewa Anda. Sesuai dengan model partisi SaaS dapat memastikan bahwa aplikasi SaaS yang menggunakan PostgreSQL mematuhi secara ketat praktik terbaik untuk mempertahankan penyewaan. Model partisi silo, jembatan, dan kolam SaaS mendukung banyak kasus penggunaan SaaS. Model-model ini memberikan berbagai keuntungan di antara faktor-faktor seperti isolasi kinerja, overhead operasional, dan keamanan penyewa.

Langkah selanjutnya:

- Evaluasi Aurora PostgreSQL yang kompatibel dan Amazon RDS untuk PostgreSQL, dan pilih opsi terbaik untuk aplikasi SaaS Anda.
- Pilih model partisi SaaS yang memenuhi persyaratan untuk aplikasi Anda: silo, jembatan, atau kolam renang.
- Terapkan PostgreSQL sesuai dengan model partisi SaaS yang Anda pilih.

Sumber daya

Referensi

- Strategi Penyimpanan SaaS: Membangun Model Penyimpanan Multitenant di (whitepaper) AWSAWS
- Pemulihan bencana Lintas Wilayah menggunakan Amazon Aurora Global Database untuk Amazon Aurora PostgreSQL (posting blog)AWS
- Isolasi data multitenant dengan PostgreSQL Row Level Security (posting blog)AWS
- Bekerja dengan Amazon Aurora PostgreSQL (dokumentasi Aurora)
- PostgreSQL di Amazon RDS (dokumentasi Amazon RDS)

Partner

- Amazon Aurora untuk Mitra PostgreSQL
- Amazon RDS untuk Mitra 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
Perbarui	Pembaruan untuk mencerminkan ketersediaan penerusan tulis di Aurora.	April 29, 2024
Perbarui	Memperbarui tabel perbandingan Amazon RDS dan Aurora.	21 Oktober 2022
—	Publikasi awal	30 September 2021

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