Kerangka Kerja AWS Well-Architected

Pilar Keandalan

Pilar Keandalan: Kerangka Kerja AWS Well-Architected

Table of Contents

Pilar Keandalan - Kerangka Kerja AWS Well-Architected

Tanggal publikasi: 27 Juni 2024 (Revisi Dokumen)

Laporan ini berfokus pada pilar keandalan Kerangka Kerja AWS Well-Architected. Laporan ini menyediakan panduan untuk membantu pelanggan menerapkan praktik terbaik dalam desain, pengiriman, dan pemeliharaan lingkungan Amazon Web Services (AWS).

Pengantar

Dengan Kerangka Kerja AWS Well-Architected , Anda dapat memahami pro dan kontra keputusan yang Anda ambil selama membangun beban kerja di AWS. Dengan menggunakan Kerangka Kerja ini, Anda akan mengetahui praktik terbaik arsitektur untuk mendesain dan mengoperasikan beban kerja yang andal, aman, efisien, hemat biaya, dan ramah lingkungan di cloud. Layanan ini menyediakan cara untuk secara terus menerus menilai arsitektur Anda berdasarkan praktik terbaik dan mengidentifikasi area yang perlu diperbaiki. Kami percaya bahwa memiliki beban kerja yang didesain dengan baik akan meningkatkan peluang keberhasilan bisnis.

Kerangka Kerja AWS Well-Architected didasarkan pada enam pilar kerangka kerja:

- Keunggulan Operasional
- Keamanan
- Keandalan
- Efisiensi Kinerja
- Optimasi Biaya
- Pelestarian Lingkungan

Artikel ini berfokus pada pilar keandalan dan cara menerapkannya ke solusi Anda. Mencapai keandalan dapat menjadi sebuah tantangan dalam lingkungan on-premise tradisional karena titik tunggal kegagalan, kurangnya otomatisasi, dan kurangnya elastisitas. Dengan mengadopsi praktik dalam artikel ini, Anda dapat membangun arsitektur yang memiliki fondasi kuat, arsitektur tangguh, manajemen perubahan yang konsisten, dan proses pemulihan kegagalan yang telah terbukti.

Artikel ini dimaksudkan untuk orang-orang yang memiliki peran di bidang teknologi, seperti kepala pejabat teknologi (chief technology officer/CTO), arsitek, developer, dan anggota tim operasi. Setelah membaca artikel ini, Anda akan memahami praktik terbaik AWS dan strategi yang digunakan ketika

merancang arsitektur cloud untuk keandalan. Artikel ini menyertakan detail implementasi tingkat tinggi dan pola arsitektur, juga referensi ke sumber daya tambahan.

Keandalan

Pilar keandalan berkenaan dengan kemampuan beban kerja untuk menjalankan fungsinya dengan benar dan konsisten sesuai harapan. Ini termasuk kemampuan untuk mengoperasikan dan menguji beban kerja di seluruh siklus hidupnya. Dokumen ini menyediakan panduan praktik terbaik yang mendalam untuk mengimplementasikan beban kerja yang andal di AWS.

Topik

Model Tanggung Jawab Bersama untuk Ketangguhan

Ketangguhan merupakan tanggung jawab bersama antara AWS dan Anda. Anda harus memahami cara pemulihan bencana (DR) dan ketersediaan beroperasi, sebagai bagian dari ketangguhan, menurut model bersama ini.

Tanggung jawab AWS - Ketangguhan cloud

AWS bertanggung jawab atas ketangguhan infrastruktur yang menjalankan semua layanan yang ditawarkan di AWS Cloud. Infrastruktur ini terdiri dari perangkat keras, perangkat lunak, jaringan, dan fasilitas yang menjalankan layanan AWS Cloud. AWS menggunakan upaya yang wajar secara komersial untuk membuat layanan-layanan AWS Cloud ini tersedia, yang memastikan ketersediaan layanan memenuhi atau melampaui Perjanjian Tingkat Layanan (SLA) AWS.

Infrastruktur Cloud Global AWS didesain untuk memungkinkan pelanggan membangun arsitektur beban kerja yang sangat tangguh. Setiap Wilayah AWS terisolasi penuh dan terdiri dari beberapa Zona Ketersediaan, yang merupakan partisi terisolasi secara fisik dari infrastruktur. Zona Ketersediaan mengisolasi kesalahan yang dapat memengaruhi ketangguhan beban kerja, yang mencegahnya memengaruhi zona-zona lain di Wilayah. Tetapi pada saat yang sama, semua zona di Wilayah AWS saling terhubung dengan bandwidth tinggi, jaringan berlatensi rendah, melalui serat metro khusus yang sepenuhnya redundan, yang menyediakan jaringan throughput tinggi dan latensi rendah antara zona. Semua lalu lintas antara zona dienkripsi. Performa jaringan ini cukup untuk mendapatkan replikasi sinkron antara zona. Ketika satu aplikasi dipartisi lintas AZ, perusahaan akan

lebih terisolasi dan terlindungi dari permasalahan seperti pemadaman listrik, sambaran petir, angin topan, angin puting beliung, dan lain-lain.

Tanggung jawab pelanggan - Ketangguhan di cloud

Tanggung jawab Anda ditentukan oleh layanan AWS Cloud yang Anda pilih. Hal ini menentukan jumlah konfigurasi kerja yang harus Anda kerjakan sebagai bagian dari tanggung jawab ketangguhan Anda. Contohnya, layanan seperti Amazon Elastic Compute Cloud (Amazon EC2) mengharuskan pelanggan melakukan semua tugas manajemen dan konfigurasi ketangguhan yang diperlukan. Pelanggan yang melakukan deployment instans Amazon EC2 bertanggung jawab untuk melakukan deployment instans Amazon EC2 ke beberapa lokasi (seperti Zona Ketersediaan AWS), mengimplementasikan pemulihan mandiri menggunakan layanan seperti Auto Scaling, dan menggunakan praktik terbaik arsitektur beban kerja tangguh untuk aplikasi yang diinstal di instans. Untuk layanan terkelola, seperti Amazon S3 dan Amazon DynamoDB, AWS mengoperasikan lapisan infrastruktur, sistem operasi, dan platform, sedangkan pelanggan mengakses titik akhir untuk menyimpan dan mengambil data. Anda bertanggung jawab untuk mengelola ketangguhan data Anda, termasuk strategi pencadangan, versioning, dan replikasi.

Melakukan deployment beban kerja Anda ke beberapa Zona Ketersediaan di Wilayah AWS merupakan bagian dari strategi ketersediaan tinggi yang didesain untuk melindungi beban kerja dengan mengisolasi masalah ke satu Zona Ketersediaan, yang menggunakan redundansi Zona Ketersediaan lain untuk terus melayani permintaan. Arsitektur Multi-AZ juga bagian dari strategi DR yang didesain untuk membuat beban kerja lebih terisolasi dan terlindungi dari masalah seperti pemadaman listrik, sambaran petir, angin topan, gempa bumi, dan lain-lain. Strategi DR juga dapat menggunakan beberapa Wilayah AWS. Contohnya, dalam konfigurasi aktif/pasif, layanan untuk beban kerja failover dari Wilayah aktifnya ke Wilayah DR-nya jika Wilayah aktif tidak dapat lagi melayani permintaan.

Tanggung jawab atas ketangguhan di cloud dan ketangguhan cloud untuk pelanggan dan AWS.

Anda dapat menggunakan layanan AWS untuk mencapai sasaran ketangguhan Anda. Sebagai pelanggan, Anda bertanggung jawab atas manajemen aspek-aspek berikut dari sistem Anda untuk mendapatkan ketangguhan di cloud. Untuk detail selengkapnya tentang setiap layanan secara khusus, lihat dokumentasi AWS.

Jaringan, kuota, dan kendala

- Praktik terbaik untuk area model tanggung jawab bersama ini dijelaskan secara mendetail di bawah Fondasi.
- Rencanakan arsitektur Anda dengan ruang yang cukup untuk menskalakan dan pahami kuota layanan dan kendala layanan yang Anda sertakan, berdasarkan ekspektasi peningkatan permintaan beban jika berlaku.
- Desain topologi jaringan Anda agar memiliki ketersediaan tinggi, redundan, dan dapat diskalakan.

Manajemen perubahan dan ketangguhan operasional

- Manajemen perubahan mencakup cara memperkenalkan dan mengelola perubahan dalam lingkungan Anda. Mengimplementasikan perubahan memerlukan strategi deployment serta pembangunan dan pemutakhiran runbook untuk aplikasi dan infrastruktur Anda.

- Strategi tangguh untuk memantau sumber daya beban kerja mempertimbangkan semua komponen, termasuk metrik teknis dan bisnis, notifikasi, otomatisasi, dan analisis.
- Beban kerja di cloud harus beradaptasi dengan perubahan dalam permintaan yang menskalakan ke dalam sebagai reaksi terhadap gangguan atau fluktuasi penggunaan.

Manajemen kegagalan dan observabilitas

- Mengamati kegagalan melalui pemantauan diperlukan untuk mengotomatiskan pemulihan sehingga beban kerja Anda dapat bertahan dari kegagalan komponen.
- Manajemen kegagalan memerlukan pencadangan data, penerapan praktik terbaik untuk memungkinkan beban kerja Anda bertahan dari kegagalan komponen, dan perencanaan pemulihan bencana.

Arsitektur beban kerja

- Arsitektur beban kerja Anda mencakup cara Anda mendesain layanan seputar domain bisnis, menerapkan SOA dan desain sistem terdistribusi untuk mencegah kegagalan, dan membangun kemampuan seperti throttling, pencobaan ulang, manajemen antrean, timeout, dan tuas darurat.
- Andalkan solusi AWS yang telah terbukti, Amazon Builders Library, dan pola nirserver untuk menyesuaikan dengan praktik terbaik dan memulai implementasi dengan cepat.
- Gunakan peningkatan berkelanjutan untuk menguraikan sistem Anda menjadi layanan terdistribusi guna menskalakan dan berinovasi lebih cepat. Gunakan opsi layanan terkelola dan panduan layanan mikro AWS untuk menyederhanakan dan mempercepat kemampuan Anda untuk memperkenalkan perubahan dan berinovasi.

Pengujian terus-menerus infrastruktur penting

- Keandalan pengujian berarti pengujian di tingkat fungsional, performa, dan kekacauan, serta adopsi praktik game day dan analisis insiden untuk membangun keahlian dalam menyelesaikan masalah yang tidak dipahami dengan baik.
- Untuk aplikasi cloud all-in maupun aplikasi hybrid, mengetahui perilaku aplikasi ketika timbul masalah atau ketika komponen tidak berfungsi akan memampukan Anda untuk pulih dari penghentian dengan cepat dan andal.
- Buat dan dokumentasikan eksperimen yang dapat diulang untuk memahami perilaku sistem ketika operasi tidak berjalan sesuai harapan. Pengujian ini akan membuktikan keefektifan ketangguhan

secara keseluruhan dan memberikan lingkaran umpan balik untuk prosedur operasional Anda sebelum menghadapi skenario kegagalan yang nyata.

Prinsip desain

Di cloud, ada sejumlah prinsip yang dapat membantu Anda meningkatkan keandalan. Pertimbangkan selalu prinsip-prinsip ini saat kita membahas praktik terbaik:

- Pemulihan otomatis dari kegagalan: Dengan memantau beban kerja untuk indikator performa utama (KPI), Anda dapat memicu otomatisasi ketika ambang batas dilanggar. KPI ini harus menjadi ukuran dari nilai bisnis, bukan menjadi ukuran aspek teknis operasi layanan. Hal ini memungkinkan notifikasi otomatis dan pelacakan kegagalan, serta proses pemulihan otomatis yang menangani atau memperbaiki kegagalan. Dengan otomatisasi yang lebih canggih, antisipasi dan perbaikan kegagalan dapat dilakukan sebelum kegagalan terjadi.
- Prosedur pemulihan pengujian: Dalam lingkungan on-premise, pengujian sering kali dijalankan untuk membuktikan bahwa beban kerja bekerja dalam skenario tertentu. Pengujian tidak digunakan untuk memvalidasi strategi pemulihan. Di cloud, Anda dapat menguji bagaimana beban kerja gagal, dan dapat memvalidasi prosedur pemulihan. Gunakan otomatisasi untuk memicu berbagai kegagalan atau untuk membuat ulang skenario yang mengarah pada kegagalan sebelumnya. Pendekatan ini memperlihatkan jalur kegagalan yang dapat diuji dan diperbaiki sebelum skenario kegagalan benar-benar terjadi, dan juga mengurangi risiko.
- Skalakan secara horizontal untuk meningkatkan ketersediaan beban kerja agregat: Ganti satu sumber daya besar dengan beberapa sumber daya kecil untuk mengurangi dampak kegagalan tunggal terhadap beban kerja keseluruhan. Distribusikan permintaan ke beberapa sumber daya yang lebih kecil untuk memastikan bahwa tidak terdapat kesamaan titik kegagalan.
- Hentikan penebakan kapasitas: Penyebab umum kegagalan dalam beban kerja on-premise adalah saturasi sumber daya, yaitu ketika permintaan yang ditempatkan di beban kerja melebihi kapasitas beban kerjanya (ini sering kali menjadi sasaran dari serangan denial of service). Di cloud, Anda dapat memantau pemanfaatan beban kerja dan permintaan, serta mengotomatiskan penambahan atau penghapusan sumber daya untuk mempertahankan tingkat keoptimalan guna memenuhi permintaan tanpa kekurangan atau kelebihan penyediaan. Batasan tentunya masih ada, tetapi sebagian kuota dapat dikontrol dan sebagian lainnya dapat dikelola (lihat Mengelola Kendala dan Kuota Layanan).
- Kelola perubahan melalui otomatisasi: Perubahan untuk infrastruktur Anda harus dibuat menggunakan otomatisasi. Perubahan yang harus dikelola mencakup perubahan untuk otomatisasi, yang kemudian dapat dilacak dan ditinjau.

Definisi

Laporan resmi ini membahas keandalan di cloud, menjelaskan praktik terbaik untuk keempat area ini:

- Fondasi
- Arsitektur Beban Kerja
- Manajemen Perubahan
- Manajemen Kegagalan

Untuk mencapai keandalan, Anda harus mengawalinya dengan fondasi—sebuah lingkungan tempat kuota layanan dan topologi jaringan mengakomodasi beban kerja. Arsitektur beban kerja sistem terdistribusi harus didesain untuk mencegah dan memitigasi kegagalan. Beban kerja harus menangani perubahan dalam permintaan dan persyaratan, serta harus didesain untuk mendeteksi kegagalan dan melakukan pemulihan mandiri secara otomatis.

Topik

- Ketangguhan, dan komponen keandalan
- Ketersediaan
- Tujuan Pemulihan Bencana (DR)

Ketangguhan, dan komponen keandalan

Keandalan beban kerja di cloud bergantung pada sejumlah faktor, dan faktor utamanya adalah Ketangguhan:

- Ketangguhan adalah kemampuan beban kerja untuk pulih dari gangguan infrastruktur atau layanan, secara dinamis memperoleh sumber daya komputasi untuk memenuhi permintaan, dan memitigasi gangguan, seperti kesalahan konfigurasi atau masalah jaringan sementara.

Faktor-faktor lain yang memengaruhi keandalan beban kerja adalah:

- Keunggulan Operasional, yang mencakup otomatisasi perubahan, penggunaan buku pedoman untuk merespons kegagalan, dan Peninjauan Kesiapan Operasional (ORR) untuk mengonfirmasi bahwa aplikasi siap untuk operasi produksi.

- Keamanan, yang mencakup pencegahan bahaya terhadap data atau infrastruktur dari pelaku kejahatan, yang dapat mengganggu ketersediaan. Misalnya, enskripsi cadangan untuk memastikan keamanan data.
- Efisiensi Kinerja, yang mencakup desain untuk tingkat permintaan maksimum dan meminimalkan latensi untuk beban kerja.
- Optimasi Biaya, yang mencakup kompromi seperti pilihan untuk mengeluarkan lebih banyak biaya untuk instans EC2 guna mencapai stabilitas statis, atau untuk mengandalkan penskalaan otomatis saat diperlukan kapasitas yang lebih besar.

Ketangguhan adalah fokus utama laporan resmi ini.

Empat aspek lainnya juga penting dan tercakup ke dalam pilarnya masing-masing di AWS Well-Architected Framework. Banyak praktik terbaik di sini juga menangani aspek-aspek keandalan tersebut, namun yang difokuskan di sini adalah ketangguhan.

Ketersediaan

Ketersediaan (juga disebut sebagai ketersediaan layanan) adalah metrik yang umum digunakan untuk mengukur ketangguhan secara kuantitatif, serta merupakan target tujuan ketangguhan.

- Ketersediaan adalah persentase waktu ketika beban kerja tersedia untuk digunakan.

Tersedia untuk digunakan artinya beban kerja berhasil memberikan kinerja sesuai fungsinya yang disepakati ketika diperlukan.

Persentase ini dihitung dengan jangka waktu, seperti bulan, tahun, atau tiga tahun ke belakang. Dengan memberlakukan interpretasi seketat mungkin, ketersediaan berkurang setiap kali aplikasi tidak beroperasi secara normal, termasuk gangguan terjadwal dan di luar jadwal. Kami mendefinisikan ketersediaan sebagai berikut:

$$Ketersediaan = \frac{Waktu\ Ketersediaan\ untuk\ Digunakan}{Waktu\ Total}$$

- Ketersediaan adalah persentase waktu aktif (seperti 99,9%) selama jangka waktu tertentu (umumnya satu bulan atau tahun)

- Penulisan singkat yang umum adalah “jumlah sembilan”; misalnya, “lima sembilan” artinya tingkat ketersediaan 99,999%
- Beberapa pelanggan memilih untuk mengecualikan waktu henti layanan terjadwal (misalnya pemeliharaan terencana) dari Waktu Total dalam rumus. Namun, ini tidak disarankan, karena pengguna Anda kemungkinan ingin menggunakan layanan Anda selama waktu-waktu tersebut.

Berikut ini adalah tabel tujuan desain ketersediaan aplikasi umum dan durasi maksimum gangguan dapat terjadi dalam satu tahun namun tetap memenuhi tujuan. Tabel ini berisi contoh-contoh tipe aplikasi yang umum kita lihat pada tiap-tiap tingkatan ketersediaan. Di sepanjang dokumen ini, kita mengacu pada nilai-nilai ini.

Ketersediaan	Ketidaktersediaan Maksimum (per tahun)	Kategori Aplikasi
99%	3 hari 15 jam	Pemrosesan batch, ekstraksi data, transfer, dan tugas-tugas pemuatan
99,9%	8 jam 45 menit	Alat internal seperti manajemen pengetahuan, pelacakan proyek
99,95%	4 jam 22 menit	Niaga online, titik penjualan
99,99%	52 menit	Pengiriman video, penyiaran beban kerja
99,999%	5 menit	Transaksi ATM, telekomun ikasi beban kerja

Mengukur ketersediaan berdasarkan permintaan. Untuk layanan Anda, mungkin lebih mudah untuk menghitung permintaan yang berhasil dan gagal daripada “waktu tersedia untuk digunakan”. Dalam kasus ini, dapat digunakan hitungan berikut:

$$Ketersediaan = \frac{Respons\ Berhasil}{Permintaan\ Valid}$$

Ini sering dihitung untuk periode satu menit atau lima menit. Lalu, persentase waktu aktif bulanan (pengukuran ketersediaan berbasis waktu) dapat dihitung dari rata-rata semua periode ini. Jika tidak ada permintaan yang diterima dalam jangka waktu tertentu, aplikasi terhitung 100% tersedia untuk waktu tersebut.

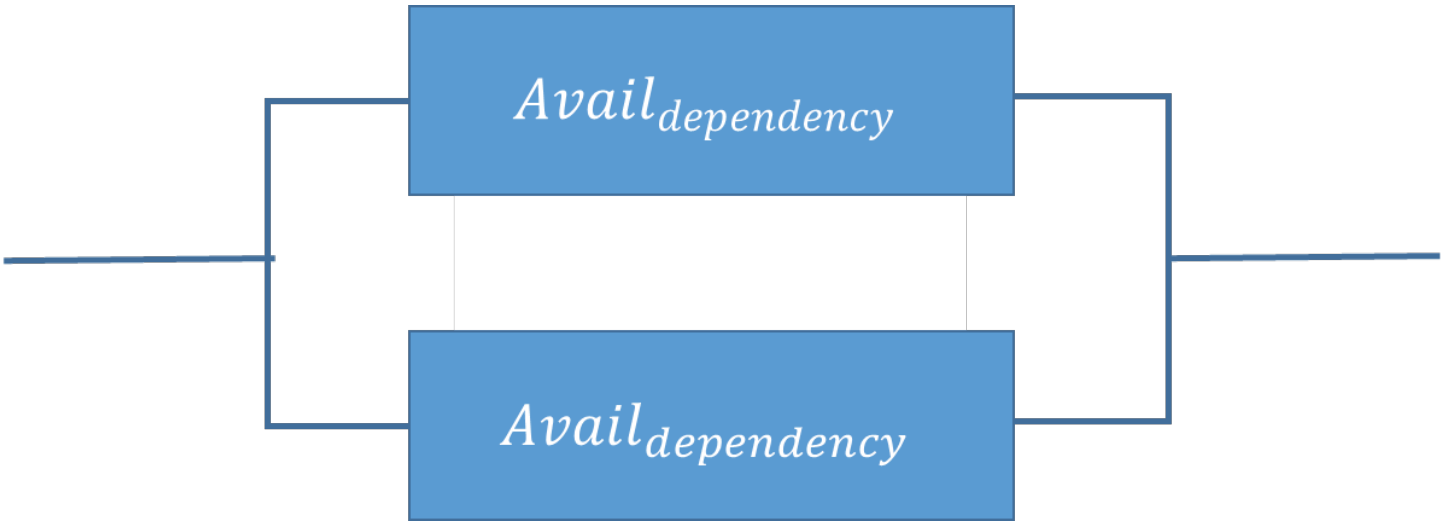
Menghitung ketersediaan dengan dependensi keras. Banyak sistem memiliki dependensi keras pada sistem lain, di mana gangguan pada sebuah sistem dependen langsung diartikan sebagai gangguan pada sistem yang memicu. Ini adalah kebalikan dari dependensi lembut, di mana kegagalan sistem dependen diimbangi di dalam aplikasi. Ketika dependensi keras terjadi, ketersediaan sistem pemicu adalah hasil dari ketersediaan sistem dependen. Misalnya, jika Anda memiliki sistem yang dirancang untuk ketersediaan 99,99% yang memiliki dependensi keras pada dua sistem independen lain yang masing-masing dirancang untuk ketersediaan 99,99%, secara teori beban kerja dapat meraih ketersediaan 99,97%:

$Avail_{invok} \times Avail_{dep1} \times Avail_{dep2} = Avail_{yang\ dapat\ diskalakan}$

99,99% × 99,99% × 99,99% = 99,97%

Oleh karena itu, penting memahami dependensi Anda serta tujuan desain ketersediaannya saat Anda menghitung ketersediaan Anda sendiri.

Menghitung ketersediaan dengan komponen redundan. Ketika suatu sistem melibatkan penggunaan komponen redundan dan independen (misalnya sumber daya redundan di beberapa Zona Ketersediaan yang berbeda), ketersediaan teoretisnya dihitung 100% dikurangi hasil tingkat kegagalan komponen. Misalny, jika suatu sistem memanfaatkan dua komponen independen, masing-masing memiliki ketersediaan 99,9%, maka ketersediaan efektif dependensi ini adalah 99,9999%.



$Avail_{hemat\ biaya} = Avail_{MAX} - ((100\% - Avail_{dependency}) \times (100\% - Avail_{dependency}))$

99,9999% = 100% − (0,1%×0,1%)

Kalkulasi pintasan: Jika ketersediaan semua komponen di dalam kalkulasi Anda hanya terdiri dari angka sembilan, maka Anda dapat menjumlah hitungan jumlah angka sembilan untuk mendapatkan jawaban. Pada contoh di atas, dua komponen independen redundan dengan ketersediaan tiga sembilan menghasilkan enam sembilan.

Menghitung ketersediaan dependensi. Beberapa dependensi menyediakan panduan tentang ketersediaannya, termasuk tujuan desain ketersediaan untuk banyak layanan AWS. Tetapi jika hal ini tidak tersedia (misalnya, komponen yang produsennya tidak mempublikasikan informasi ketersediaan), satu cara untuk memperkirakan adalah dengan menentukan Waktu Rata-Rata Antara Kegagalan (MTBF) dan Waktu Rata-Rata untuk Pulih (MTTR). Hitungan ketersediaan dapat dilakukan dengan:

$$\text{Ketersediaan yang diperkirakan} = \frac{MTBF}{MTBF + MTTR}$$

Misalnya, jika MTBF 150 hari dan MTTR 1 jam, hitungan ketersediaannya adalah 99,97%.

Untuk detail tambahan, lihat Ketersediaan dan Lain-Lain: Memahami dan meningkatkan ketangguhan sistem terdistribusi di AWS, yang dapat membantu Anda menghitung ketersediaan.

Biaya ketersediaan. Merancang aplikasi untuk tingkat ketersediaan yang lebih tinggi umumnya menyebabkan peningkatan biaya, sehingga sebaiknya Anda mengidentifikasi kebutuhan ketersediaan yang sebenarnya sebelum memulai desain aplikasi Anda. Tingkat ketersediaan yang tinggi menghadirkan persyaratan pengujian dan validasi yang lebih ketat di bawah skenario kegagalan yang lengkap. Tingkat ketersediaan ini memerlukan otomatisasi untuk pemulihan dari semua bentuk kegagalan, dan mengharuskan semua aspek operasi sistem untuk dibangun serupa dan diuji dengan standar yang sama. Misalnya, penambahan atau penghapusan kapasitas, deployment atau pembatalan pembaruan perangkat lunak atau perubahan konfigurasi, atau migrasi data sistem harus dilakukan sesuai tujuan ketersediaan yang diinginkan. Selain biaya untuk pengembangan perangkat lunak, pada tingkat ketersediaan yang sangat tinggi, inovasi juga terkena dampak dikarenakan kebutuhan untuk bergerak lebih lambat dalam men-deploy sistem. Oleh karena itu, panduan harus menyeluruh dalam menerapkan standar dan mempertimbangkan target ketersediaan yang tepat untuk seluruh siklus hidup pengoperasian sistem.

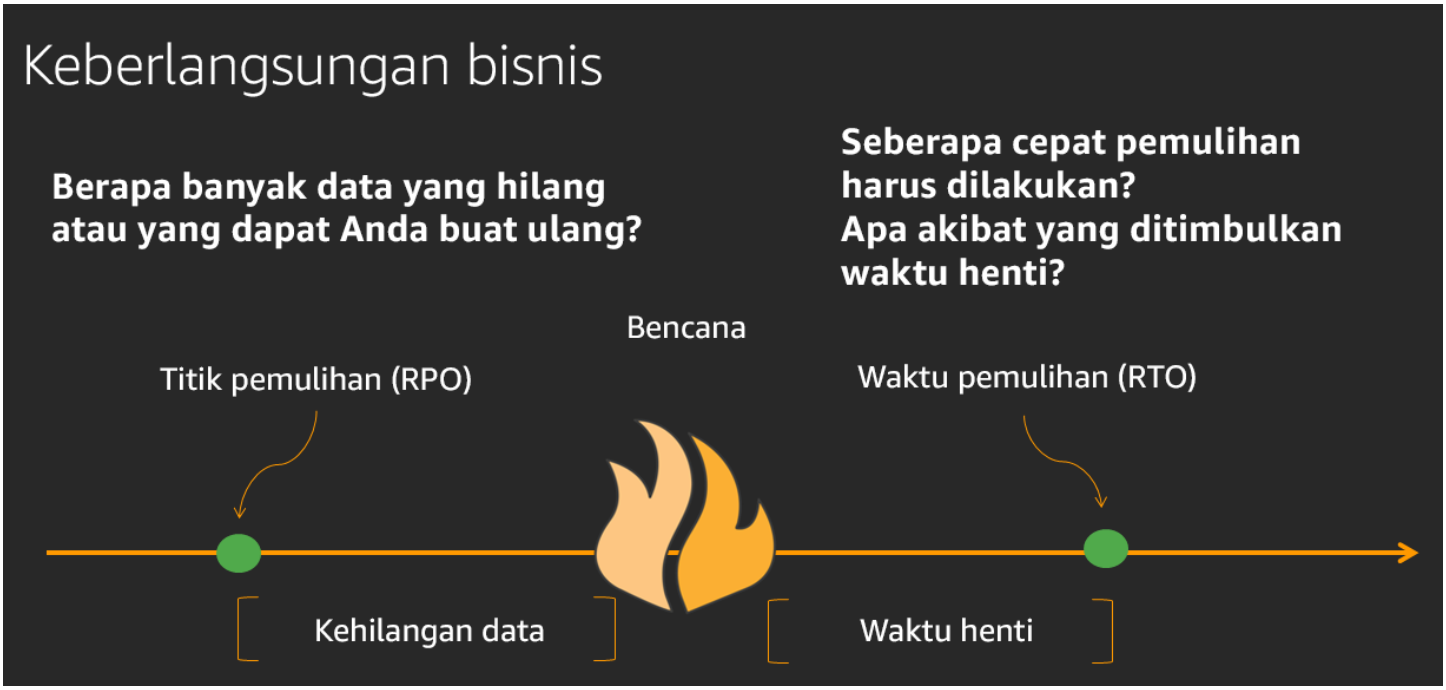
Faktor lain meningkatnya biaya dalam sistem yang beroperasi dengan tujuan desain ketersediaan yang lebih tinggi adalah pemilihan dependensi. Pada tujuan-tujuan yang lebih tinggi ini, set perangkat lunak atau layanan yang dapat dipilih sebagai dependensi berkurang berdasarkan layanan mana yang telah memiliki investasi mendalam yang telah kami jelaskan sebelumnya. Seiring meningkatnya tujuan desain ketersediaan, merupakan hal umum untuk menemukan lebih sedikit layanan multitujuan (seperti basis data relasional) dan lebih banyak layanan yang dibuat khusus. Alasannya adalah jenis layanan yang dibuat khusus lebih mudah untuk dievaluasi, diuji, dan diotomatisasi, serta memiliki lebih sedikit potensi interaksi kejutan dengan fungsionalitas yang disertakan namun tidak digunakan.

Tujuan Pemulihan Bencana (DR)

Selain tujuan ketersediaan, strategi ketangguhan Anda juga harus menyertakan tujuan Pemulihan Bencana (DR) berdasarkan strategi untuk memulihkan beban kerja Anda apabila terjadi peristiwa bencana. Pemulihan Bencana berfokus pada tujuan pemulihan satu kali untuk merespons bencana alam, kegagalan teknis skala besar, atau ancaman manusia seperti serangan atau kesalahan. Ini berbeda dengan ketersediaan yang mengukur rata-rata ketangguhan selama kurun waktu tertentu ketika merespons kegagalan komponen, lonjakan beban, atau bug perangkat lunak.

Sasaran Waktu Pemulihan (RTO) Ditetapkan oleh organisasi. RTO adalah jeda waktu maksimum yang dapat diterima antara gangguan layanan dan pemulihan layanan. Ini menentukan apa yang dianggap sebagai jendela waktu yang dapat diterima ketika layanan tidak tersedia.

Sasaran Titik Pemulihan (RPO) Ditetapkan oleh organisasi. RPO adalah jumlah waktu maksimum yang dapat diterima sejak titik pemulihan data terakhir. Ini menentukan apa yang dianggap sebagai kehilangan data yang dapat diterima antara titik pemulihan terakhir dan gangguan layanan.



Hubungan antara RPO (Sasaran Titik Pemulihan), RTO (Sasaran Waktu Pemulihan), dan peristiwa bencana.

RTO serupa dengan MTTR (Rata-Rata Waktu ke Pemulihan) yakni keduanya sama-sama mengukur waktu antara mulainya pemadaman dan pemulihan beban kerja. Namun, MTTR adalah nilai rata-rata yang diambil selama beberapa peristiwa yang berdampak pada ketersediaan dalam kurun waktu tertentu, sedangkan RTO adalah target, atau nilai maksimum yang diizinkan, untuk satu peristiwa yang memengaruhi ketersediaan.

Memahami kebutuhan ketersediaan

Merupakan hal normal ketika di awal, ketersediaan suatu aplikasi dianggap sebagai satu target untuk aplikasi secara keseluruhan. Namun, setelah diselidiki lebih teliti, kita sering menemukan bahwa aspek-aspek tertentu di suatu aplikasi atau layanan memiliki persyaratan ketersediaan yang berbeda. Misalnya, beberapa sistem mungkin mengutamakan kemampuan untuk menerima dan menyimpan data baru sebelum mengambil data yang sudah ada. Sistem lainnya mengutamakan operasi waktu nyata daripada operasi yang mengubah konfigurasi atau lingkungan sistem. Layanan mungkin memiliki persyaratan ketersediaan yang sangat tinggi selama jam tertentu, tetapi dapat mentoleransi gangguan yang jauh lebih lama di luar jam-jam tersebut. Ini adalah beberapa cara untuk mengurai satu aplikasi menjadi bagian-bagian komponen, dan mengevaluasi persyaratan ketersediaan untuk masing-masing. Manfaat melakukan hal ini adalah untuk memfokuskan upaya (serta pengeluaran)

Anda pada ketersediaan berdasarkan kebutuhan khusus, bukan merekayasa keseluruhan sistem sesuai persyaratan paling ketat.

Rekomendasi
Secara kritis, evaluasi aspek-aspek unik untuk aplikasi Anda dan, jika perlu, bedakan tujuan desain ketersediaan dan pemulihan bencana untuk mencerminkan kebutuhan bisnis Anda.

Di dalam AWS, kita umumnya membagi layanan ke dalam “bidang data” dan “bidang kendali.” Bidang data bertanggung jawab untuk menghadirkan layanan waktu nyata sedangkan bidang kendali digunakan untuk mengonfigurasi lingkungan. Misalnya, instans Amazon EC2, basis data Amazon RDS, dan operasi baca/tulis tabel Amazon DynamoDB semuanya adalah operasi bidang data. Sebaliknya, peluncuran instans EC2 atau basis data RDS baru, atau penambahan atau pengubahan metadata tabel di DynamoDB dianggap sebagai operasi bidang kendali. Meskipun tingkat ketersediaan yang tinggi penting untuk semua kemampuan ini, bidang data umumnya memiliki tujuan desain ketersediaan yang lebih tinggi daripada bidang kendali. Oleh karena itu, beban kerja dengan persyaratan ketersediaan yang tinggi harus menghindari dependensi run-time pada operasi bidang kendali.

Banyak pelanggan AWS menerapkan pendekatan serupa dalam mengevaluasi aplikasi mereka secara kritis dan mengidentifikasi subkomponen dengan kebutuhan ketersediaan yang berbeda-beda. Tujuan desain ketersediaan kemudian disesuaikan dengan berbagai aspek, dan upaya kerja yang tepat dijalankan untuk merekayasa sistem. AWS memiliki banyak pengalaman dalam merekayasa aplikasi dengan berbagai tujuan desain ketersediaan, termasuk layanan dengan ketersediaan 99,999% atau lebih. AWS Solution Architects (SA) dapat membantu Anda merancang tujuan ketersediaan Anda dengan baik. Pelibatan AWS sejak awal dalam proses desain Anda dapat meningkatkan kemampuan kami untuk membantu Anda memenuhi tujuan ketersediaan. Perencanaan ketersediaan tidak hanya dilakukan sebelum peluncuran beban kerja Anda. Perencanaan ini juga dilakukan secara berkelanjutan untuk menyempurnakan desain Anda seiring Anda mendapatkan pengalaman operasional, belajar dari peristiwa dunia nyata, dan bertahan di tengah berbagai jenis kegagalan. Anda kemudian dapat menerapkan upaya kerja yang tepat untuk melakukan perbaikan setelah implementasi Anda.

Kebutuhan ketersediaan yang diperlukan untuk suatu beban kerja harus diselaraskan dengan kebutuhan dan kekritisan bisnis. Dengan menetapkan terlebih dulu kerangka kerja kekritisan bisnis dengan RTO, RPO, dan ketersediaan yang ditetapkan, Anda dapat menilai tiap-tiap beban kerja. Pendekatan seperti ini mengharuskan orang-orang yang terlibat di dalam implementasi beban kerja

untuk memahami kerangka kerja tersebut, dan dampak yang diberikan oleh beban kerja mereka terhadap kebutuhan bisnis.

Fondasi

Persyaratan mendasar memiliki cakupan yang lebih luas dari satu beban kerja atau proyek. Sebelum merancang sistem apa pun, persyaratan mendasar yang memengaruhi keandalan harus diterapkan. Misalnya, Anda harus memiliki bandwidth jaringan yang memadai untuk pusat data Anda.

Di lingkungan on-premise, persyaratan ini dapat menyebabkan waktu jeda yang lama disebabkan dependensi sehingga harus disertakan selama perencanaan awal. Namun, dengan AWS, sebagian besar persyaratan mendasar ini sudah disertakan atau dapat ditangani sesuai kebutuhan. Cloud dirancang agar menjadi hampir tidak terbatas, sehingga AWS bertanggung jawab untuk memenuhi persyaratan jaringan dan kapasitas komputasi yang memadai, agar Anda dapat dengan leluasa mengubah alokasi dan ukuran sumber daya sesuai permintaan.

Bagian-bagian berikutnya menjelaskan praktik terbaik yang berfokus pada pertimbangan keandalan ini.

Topik

- Mengelola kuota layanan dan kendala
- Merencanakan topologi jaringan Anda

Mengelola kuota layanan dan kendala

Untuk arsitektur beban kerja berbasis cloud, terdapat kuota layanan (yang juga disebut sebagai batas layanan). Kuota ini ada agar penyediaan sumber daya tidak melebihi yang Anda butuhkan dan untuk membatasi tingkat permintaan di operasi API serta melindungi layanan dari penyalahgunaan. Selain itu, terdapat beberapa hal yang membatasi sumber daya, misalnya, rasio yang dapat menurunkan bit di kabel serat optik, atau jumlah penyimpanan di dalam disk fisik.

Jika Anda menggunakan aplikasi AWS Marketplace, Anda harus memahami batasan penggunaannya. Jika Anda menggunakan layanan web atau perangkat lunak sebagai layanan pihak ketiga, Anda juga harus mengetahui batasannya.

Praktik terbaik

- REL01-BP01 Kesadaran tentang kuota dan kendala layanan
- REL01-BP02 Mengelola kuota layanan di seluruh akun dan wilayah
- REL01-BP03 Mengakomodasi kuota layanan tetap dan kendala melalui arsitektur

- REL01-BP04 Memantau dan mengelola kuota
- REL01-BP05 Mengotomatiskan manajemen kuota
- REL01-BP06 Memastikan adanya selisih yang memadai antara kuota saat ini dan penggunaan maksimum untuk mengakomodasi failover

REL01-BP01 Kesadaran tentang kuota dan kendala layanan

Perhatikan kuota default Anda dan kelola permintaan penambahan kuota untuk arsitektur beban kerja Anda. Ketahui kendala sumber daya cloud mana, seperti disk atau jaringan, yang berpotensi memberi dampak.

Hasil yang diinginkan: Pelanggan dapat mencegah penurunan kualitas atau gangguan layanan dalam Akun AWS mereka dengan mengimplementasikan pedoman yang tepat untuk memantau metrik utama, peninjauan infrastruktur, dan langkah-langkah perbaikan otomatisasi untuk memverifikasi tidak tercapainya kuota dan kendala layanan yang dapat menyebabkan penurunan kualitas dan gangguan layanan.

Antipola umum:

- Melakukan deployment beban kerja tanpa memahami kuota keras dan lunak serta batasannya untuk layanan yang digunakan.
- Melakukan deployment beban kerja pengganti tanpa menganalisis dan mengonfigurasi ulang kuota yang diperlukan atau menghubungi tim Dukungan sebelumnya.
- Berasumsi bahwa layanan cloud tidak memiliki batasan dan layanan dapat digunakan tanpa mempertimbangkan angka permintaan, batas, hitungan, kuantitas.
- Berasumsi bahwa kuota akan ditingkatkan secara otomatis.
- Tidak mengetahui proses dan lini waktu permintaan kuota.
- Berasumsi bahwa kuota layanan cloud default selalu sama untuk setiap layanan di semua wilayah.
- Berasumsi bahwa kendala layanan dapat ditembus dan sistem akan diskalakan secara otomatis dan meningkatkan batas di luar kendala sumber daya.
- Tidak menguji aplikasi saat lalu lintas memuncak untuk membebani pemanfaatan sumber dayanya.
- Melakukan pengadaan sumber daya tanpa analisis ukuran sumber daya yang diperlukan.
- Berlebihan dalam pengadaan kapasitas dengan memilih jenis sumber daya yang jauh melampaui kebutuhan riil atau perkiraan lalu lintas puncak.

- Tidak menilai persyaratan kapasitas untuk tingkat lalu lintas baru sebelum peristiwa pelanggan baru atau deployment teknologi baru.

Manfaat menjalankan praktik terbaik ini: Pemantauan dan manajemen otomatis kuota layanan serta kendala sumber daya dapat mengurangi kegagalan secara proaktif. Perubahan pada pola lalu lintas untuk layanan pelanggan dapat menyebabkan gangguan atau penurunan kualitas jika praktik terbaik tidak diikuti. Dengan memantau dan mengelola nilai-nilai ini di semua wilayah dan semua akun, aplikasi dapat memiliki ketahanan yang lebih baik selama peristiwa yang tidak direncanakan atau tidak diinginkan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Service Quotas adalah sebuah layanan AWS yang membantu Anda mengelola kuota Anda untuk lebih dari 250 layanan AWS dari satu lokasi. Di samping mencari nilai kuota, Anda juga dapat meminta dan melacak peningkatan kuota dari konsol Service Quotas atau menggunakan SDK AWS. AWS Trusted Advisor menawarkan pemeriksaan kuota layanan yang menampilkan penggunaan dan kuota Anda untuk beberapa aspek dari beberapa layanan. Kuota layanan default per layanan juga ada di dalam dokumentasi AWS berdasarkan layanan masing-masing (misalnya, lihat Kuota Amazon VPC).

Beberapa batas layanan, seperti batas tingkat pada API yang diberikan throttling diatur di dalam Amazon API Gateway itu sendiri dengan mengonfigurasi rencana penggunaan. Batas yang diatur sebagai konfigurasi di layanannya masing-masing diantaranya adalah IOPS yang Disediakan, penyimpanan Amazon RDS yang dialokasikan, dan alokasi volume Amazon EBS. Amazon Elastic Compute Cloud memiliki dasbor batas layanannya sendiri yang dapat membantu Anda mengelola instans Anda, Amazon Elastic Block Store, dan batas alamat IP Elastis. Jika Anda memiliki kasus penggunaan di mana kuota layanan memengaruhi kinerja aplikasi Anda dan tidak dapat disesuaikan dengan kebutuhan Anda, hubungi AWS Support untuk mengetahui apakah terdapat langkah mitigasi.

Kuota layanan bisa menurut Wilayah atau bersifat global. Layanan AWS yang mencapai kuotanya tidak akan berfungsi sebagaimana mestinya dalam penggunaan normal dan dapat menyebabkan gangguan atau penurunan kualitas layanan. Misalnya, suatu kuota layanan membatasi jumlah DL Amazon EC2 yang dapat digunakan di sebuah Wilayah dan batasan tersebut dapat dicapai selama peristiwa penskalaan lalu lintas menggunakan grup Auto Scaling (ASG).

Kuota layanan untuk setiap akun harus dinilai secara rutin dalam hal penggunaan untuk menentukan batas layanan yang tepat untuk akun tersebut. Kuota layanan ini dibuat sebagai pagar pembatas

operasional, agar Anda tidak melakukan pengadaan sumber daya lebih dari yang dibutuhkan tanpa disadari. Kuota layanan juga berfungsi untuk membatasi angka permintaan pada operasi API guna melindungi layanan dari penyalahgunaan.

Kendala layanan berbeda dari kuota layanan. Kendala layanan mewakili batasan sumber daya tertentu yang ditentukan oleh jenis sumber daya tersebut. Kendala tersebut dapat berupa kapasitas penyimpanan (misalnya, gp2 memiliki batas ukuran 1 GB - 16 TB) atau disk throughput (10.0000 iops). Sangat penting untuk merancang dan terus menilai kendala jenis sumber daya untuk penggunaan yang mungkin mencapai batasnya. Jika suatu kendala tercapai di luar perkiraan, aplikasi dan layanan akun dapat terganggu atau mengalami penurunan kualitas.

Jika terdapat kasus penggunaan di mana kuota layanan memengaruhi kinerja aplikasi dan tidak dapat disesuaikan dengan kebutuhan, hubungi AWS Support untuk mengetahui apakah terdapat langkah mitigasi. Untuk detail selengkapnya tentang penyesuaian kuota tetap, lihat REL01-BP03 Mengakomodasi kuota layanan tetap dan kendala melalui arsitektur.

Terdapat sejumlah layanan dan alat AWS untuk membantu memantau dan mengelola Service Quotas. Layanan dan alat harus dimanfaatkan untuk menyediakan pemeriksaan level kuota secara otomatis atau manual.

- AWS Trusted Advisor menawarkan pemeriksaan kuota layanan yang menampilkan penggunaan dan kuota Anda untuk beberapa aspek dari beberapa layanan. Alat ini dapat membantu mengidentifikasi layanan yang mendekati kuota.
- AWS Management Console menyediakan metode untuk menampilkan nilai kuota layanan, mengelola, meminta kuota baru, memantau status permintaan kuota, dan menampilkan riwayat kuota.
- AWS CLI dan CDK menawarkan metode terprogram untuk mengelola dan memantau level serta penggunaan kuota layanan secara otomatis.

Langkah implementasi

Untuk Service Quotas:

- Pelajari AWS Service Quotas.
- Untuk mengetahui kuota layanan Anda saat ini, tentukan layanan (seperti IAM Access Analyzer) yang digunakan. Terdapat sekitar 250 layanan AWS yang dikontrol oleh kuota layanan. Lalu, tentukan nama kuota layanan tertentu yang dapat digunakan di dalam setiap akun dan wilayah. Terdapat sekitar 3000 nama kuota layanan per wilayah.

- Perkuat analisis kuota ini dengan AWS Config untuk menemukan semua sumber daya AWS yang digunakan di dalam Akun AWS Anda.
- Gunakan data AWS CloudFormation untuk menentukan sumber daya AWS Anda yang digunakan. Lihat sumber daya yang dibuat baik di AWS Management Console maupun dengan perintah `list-stack-resources` AWS CLI. Anda juga dapat melihat sumber daya yang dikonfigurasi untuk diterapkan di templat itu sendiri.
- Tentukan semua layanan yang diperlukan oleh beban kerja Anda dengan melihat kode deployment.
- Tentukan kuota layanan yang berlaku. Gunakan informasi yang dapat diakses secara terprogram dari Trusted Advisor dan Service Quotas.
- Bangun metode pemantauan otomatis (lihat REL01-BP02 Mengelola kuota layanan di seluruh akun dan wilayah dan REL01-BP04 Memantau dan mengelola kuota) untuk memberi peringatan dan pemberitahuan jika kuota layanan mendekati atau sudah mencapai batas.
- Bangun metode otomatis dan terprogram untuk memeriksa apakah kuota layanan telah diubah di satu wilayah tetapi tidak diubah di wilayah lain di dalam akun yang sama (lihat REL01-BP02 Mengelola kuota layanan di seluruh akun dan wilayah dan REL01-BP04 Memantau dan mengelola kuota).
- Otomatiskan pemindaian log dan metrik aplikasi untuk menentukan apakah terdapat kesalahan kuota atau kendala layanan. Jika terdapat kesalahan, kirimkan peringatan ke sistem pemantauan.
- Bangun prosedur rekayasa untuk menghitung perubahan yang diperlukan dalam kuota (lihat REL01-BP05 Mengotomatiskan manajemen kuota) setelah diidentifikasi bahwa diperlukan kuota yang lebih besar untuk layanan tertentu.
- Buat alur kerja pengadaan dan persetujuan untuk meminta perubahan dalam kuota layanan. Sertakan di dalamnya alur kerja pengecualian untuk mengantisipasi jika permintaan ditolak atau disetujui sebagian.
- Buat metode rekayasa untuk meninjau kuota layanan sebelum pengadaan dan menggunakan layanan AWS baru sebelum digulirkan ke produksi atau lingkungan yang berisi data (misalnya akun pengujian beban).

Untuk kendala layanan:

- Bangun metode pemantauan dan metrik untuk memberi peringatan jika pembacaan sumber daya mendekati kendala sumber dayanya. Manfaatkan CloudWatch apabila diperlukan untuk pemantauan metrik atau log.

- Bangun ambang batas peringatan untuk setiap sumber daya yang memiliki kendala yang dapat berpengaruh pada aplikasi atau sistem.
- Ciptakan prosedur manajemen alur kerja dan infrastruktur untuk mengubah jenis sumber daya jika pemanfaatan mendekati kendala. Alur kerja ini harus mencakup pengujian beban sebagai praktik terbaik untuk memverifikasi bahwa jenis sumber daya baru tersebut sudah tepat dengan kendala baru.
- Migrasikan sumber daya yang diidentifikasi ke jenis sumber daya baru yang disarankan, menggunakan prosedur dan proses yang ada.

Sumber daya

Praktik Terbaik Terkait:

- REL01-BP02 Mengelola kuota layanan di seluruh akun dan wilayah
- REL01-BP03 Mengakomodasi kuota layanan tetap dan kendala melalui arsitektur
- REL01-BP04 Memantau dan mengelola kuota
- REL01-BP05 Mengotomatiskan manajemen kuota
- REL01-BP06 Memastikan adanya selisih yang memadai antara kuota saat ini dan penggunaan maksimum untuk mengakomodasi failover
- REL03-BP01 Memilih cara untuk menyegmentasi beban kerja
- REL10-BP01 Melakukan deployment beban kerja ke beberapa lokasi
- REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan
- REL11-BP03 Mengotomatisasi pemulihan di semua lapisan
- REL12-BP05 Menguji ketahanan menggunakan chaos engineering

Dokumen terkait:

- Pilar Keandalan Kerangka Kerja AWS Well-Architected: Ketersediaan
- AWS Service Quotas (sebelumnya disebut batas layanan)
- AWS Trusted Advisor Pemeriksaan Praktik Terbaik (lihat bagian Batas Layanan)
- Pemantau batas AWS di AWS Answers
- Batas Layanan Amazon EC2
- Apa itu Service Quotas?

- Cara Meminta Peningkatan Kuota
- Endpoint dan kuota layanan
- Panduan Pengguna Service Quotas
- Pemantau Kuota untuk AWS
- Batas Isolasi Kesalahan AWS
- Ketersediaan dengan redundansi
- AWS untuk Data
- Apa itu Integrasi Berkelanjutan?
- Apa itu Pengiriman Berkelanjutan?
- Partner APN: partner yang dapat membantu manajemen konfigurasi
- Mengelola siklus hidup akun di dalam lingkungan SaaS akun per tenant di AWS
- Mengelola dan memanfatau throttling API di dalam beban kerja Anda
- Lihat rekomendasi AWS Trusted Advisor pada skala besar dengan AWS Organizations
- Mengotomatiskan Peningkatan Batas Layanan dan Dukungan Korporat dengan AWS Control Tower

Video terkait:

- AWS Live re:Inforce 2019 - Service Quotas
- Melihat dan Mengelola Kuota untuk Layanan AWS Menggunakan Service Quotas
- Demo Kuota AWS IAM

Alat terkait:

- Amazon CodeGuru Reviewer
- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor

- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP02 Mengelola kuota layanan di seluruh akun dan wilayah

Jika Anda menggunakan beberapa akun atau Wilayah, minta kuota yang sesuai di semua lingkungan tempat beban kerja produksi Anda dijalankan.

Hasil yang diinginkan: Layanan dan aplikasi tidak boleh dipengaruhi oleh habisnya kuota layanan untuk konfigurasi yang meliputi akun atau Wilayah atau yang memiliki desain ketahanan menggunakan failover zona, Wilayah, atau akun.

Antipola umum:

- Membiarkan penggunaan sumber daya di satu Wilayah terisolasi bertambah, tanpa mekanisme untuk mempertahankan kapasitas di wilayah lainnya.
- Mengatur semua kuota di Wilayah isolasi secara manual dan independen.
- Tidak mempertimbangkan efek arsitektur ketahanan (seperti aktif atau pasif) dalam kebutuhan kuota masa depan selama penurunan kualitas di dalam Wilayah non-primer.
- Tidak mengevaluasi kuota secara rutin dan membuat perubahan yang diperlukan di setiap Wilayah dan akun tempat beban kerja dijalankan.
- Tidak memanfaatkan templat pemintaan kuota untuk meminta penambahan di beberapa Wilayah dan akun.
- Tidak memperbarui kuota layanan disebabkan pemikiran yang tidak tepat bahwa peningkatan kuota memiliki dampak biaya seperti permintaan pemesanan komputasi.

Manfaat menjalankan praktik terbaik ini: Memverifikasi bahwa Anda dapat menangani beban saat ini di wilayah atau akun sekunder jika layanan wilayah tidak tersedia. Hal ini dapat membantu mengurangi jumlah kesalahan atau tingkat penurunan kualitas yang terjadi selama region loss.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Kuota layanan dilacak per akun. Setiap kuota disesuaikan dengan Wilayah AWS, kecuali ditetapkan sebaliknya. Selain lingkungan produksi, kelola juga kuota di semua lingkungan non-produksi yang

berlaku agar pengujian dan pengembangan tidak terhambat. Untuk mempertahankan tingkat ketahanan yang tinggi diperlukan penilaian kuota layanan secara kontinu (baik otomatis maupun manual).

Dengan makin banyaknya beban kerja yang meliputi beberapa Wilayah dikarenakan implementasi desain yang menggunakan pendekatan Aktif/Aktif, Aktif/Pasif – Panas, Aktif/Pasif-Dingin, dan Aktif/Pasif-Pilot Light, sangat penting memahami semua level kuota Wilayah dan akun. Pola lalu lintas terdahulu tidak selalu menjadi indikator yang tepat bahwa kuota layanan diatur dengan benar.

Sama pentingnya, batas nama kuota layanan tidak selalu sama untuk setiap Wilayah. Di satu Wilayah, nilainya bisa jadi lima, sedangkan di wilayah lain nilainya bisa jadi sepuluh. Manajemen kuota-kuota ini harus meliputi semua layanan, akun, dan Wilayah yang sama untuk menyediakan ketahanan yang konsisten saat beban meningkat.

Sesuaikan semua perbedaan kuota layanan di semua Wilayah yang berbeda (Wilayah Aktif atau Wilayah Pasif) dan buat proses untuk menyesuaikan semua perbedaan tersebut secara kontinu. Rencana pengujian failover Wilayah pasif sangat jarang diskalakan ke kapasitas aktif puncak, sehingga kegiatan game day atau table top bisa gagal menemukan perbedaan kuota layanan antar-Wilayah dan juga mempertahankan batas-batas yang tepat.

Penyimpangan kuota layanan, kondisi di mana batas kuota layanan untuk kuota bernama tertentu diubah di salah satu Wilayah tetapi tidak di semua Wilayah, sangat penting untuk dilacak dan dinilai. Mengubah kuota di Wilayah yang memiliki lalu lintas atau yang berpotensi mendatangkan lalu lintas harus dipertimbangkan.

- Pilih Wilayah dan akun yang relevan berdasarkan persyaratan layanan, latensi, peraturan, dan pemulihan bencana (DR) Anda.
- Identifikasikan kuota layanan di semua akun, Wilayah, dan Zona Ketersediaan yang relevan. Batasannya mencakup akun dan Wilayah. Nilai-nilai ini harus dibandingkan untuk mengetahui perbedaannya.

Langkah implementasi

- Tinjau nilai Service Quotas yang mungkin telah melampaui level risiko penggunaan a. AWS Trusted Advisor menyediakan peringatan untuk pelanggaran 80% dan 90% ambang batas.
- Tinjau nilai untuk kuota layanan di Wilayah Pasif mana pun (dalam desain Aktif/Pasif). Verifikasi bahwa muatan akan berhasil berjalan di dalam Wilayah sekunder apabila terjadi kegagalan di dalam Wilayah primer.

- Otomatiskan penilaian jika penyelewengan kuota layanan apa pun telah terjadi antar-Wilayah di dalam akun yang sama dan lakukan tindakan yang semestinya untuk mengubah batas.
- Jika Unit Organisasi (OU) pelanggan disusun dengan cara yang didukung, templat kuota layanan harus diperbarui agar mencerminkan perubahan pada kuota apa pun yang harus diterapkan ke beberapa Wilayah dan akun.
  - Buat templat dan kaitkan Wilayah dengan perubahan kuota.
  - Tinjau semua templat kuota layanan yang ada untuk mengetahui jika ada perubahan yang diperlukan (Wilayah, batas, dan akun).

Sumber daya

Praktik Terbaik Terkait:

- REL01-BP01 Kesadaran tentang kuota dan kendala layanan
- REL01-BP03 Mengakomodasi kuota layanan tetap dan kendala melalui arsitektur
- REL01-BP04 Memantau dan mengelola kuota
- REL01-BP05 Mengotomatiskan manajemen kuota
- REL01-BP06 Memastikan adanya selisih yang memadai antara kuota saat ini dan penggunaan maksimum untuk mengakomodasi failover
- REL03-BP01 Memilih cara untuk menyegmentasi beban kerja
- REL10-BP01 Melakukan deployment beban kerja ke beberapa lokasi
- REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan
- REL11-BP03 Mengotomatisasi pemulihan di semua lapisan
- REL12-BP05 Menguji ketahanan menggunakan chaos engineering

Dokumen terkait:

- Pilar Keandalan Kerangka Kerja AWS Well-Architected: Ketersediaan
- AWS Service Quotas (sebelumnya disebut batas layanan)
- AWS Trusted Advisor Pemeriksaan Praktik Terbaik (lihat bagian Batas Layanan)
- Pemantau batas AWS di AWS Answers
- Batas Layanan Amazon EC2

- Apa itu Service Quotas?
- Cara Meminta Peningkatan Kuota
- Endpoint dan kuota layanan
- Panduan Pengguna Service Quotas
- Pemantau Kuota untuk AWS
- Batas Isolasi Kesalahan AWS
- Ketersediaan dengan redundansi
- AWS untuk Data
- Apa itu Integrasi Berkelanjutan?
- Apa itu Pengiriman Berkelanjutan?
- Partner APN: partner yang dapat membantu manajemen konfigurasi
- Mengelola siklus hidup akun di dalam lingkungan SaaS akun per tenant di AWS
- Mengelola dan memanfatau throttling API di dalam beban kerja Anda
- Lihat rekomendasi AWS Trusted Advisor pada skala besar dengan AWS Organizations
- Mengotomatiskan Peningkatan Batas Layanan dan Dukungan Korporat dengan AWS Control Tower

Video terkait:

- AWS Live re:Inforce 2019 - Service Quotas
- Melihat dan Mengelola Kuota untuk Layanan AWS Menggunakan Service Quotas
- Demo Kuota AWS IAM

Layanan terkait:

- Amazon CodeGuru Reviewer
- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru

- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP03 Mengakomodasi kuota layanan tetap dan kendala melalui arsitektur

Waspadai kuota layanan, kendala layanan, dan batas sumber daya fisik yang tidak dapat diubah. Rancang arsitektur untuk aplikasi dan layanan untuk mencegah batasan ini memengaruhi keandalan.

Contohnya antara lain bandwidth jaringan, ukuran payload invokasi fungsi nirserver, tingkat burst throttle untuk gateway API, dan sambungan pengguna serentak ke basis data.

Hasil yang diinginkan: Performa aplikasi atau layanan sesuai yang diharapkan dalam kondisi normal dan lalu lintas tinggi. Aplikasi dan layanan telah dirancang untuk berfungsi dengan batasan untuk kuota layanan atau kendala tetap sumber daya tersebut.

Antipola umum:

- Memilih desain yang menggunakan sumber daya layanan, tidak menyadari bahwa terdapat kendala desain yang akan menyebabkan desain ini gagal begitu Anda menyesuaikan skala.
- Melakukan tolok ukur yang tidak realistis dan akan mencapai kuota tetap layanan selama pengujian. Contohnya, menjalankan pengujian pada batas burst tetapi dalam jangka waktu yang lama.
- Memilih desain yang tidak dapat diskalakan atau dimodifikasi jika kuota layanan tetap akan terlampaui. Contohnya, ukuran payload SQS sebesar 256 KB.
- Observabilitas belum dirancang dan diimplementasikan untuk memantau dan memberikan peringatan tentang ambang batas kuota layanan yang mungkin berisiko selama lalu lintas sedang tinggi.

Manfaat menjalankan praktik terbaik ini: Verifikasi bahwa aplikasi akan beroperasi di bawah semua tingkat beban layanan yang diproyeksikan tanpa gangguan atau penurunan kualitas.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Tidak seperti sumber daya atau kuota layanan lunak yang diganti dengan unit kapasitas lebih tinggi, kuota tetap layanan AWS tidak dapat diubah. Ini berarti semua jenis layanan AWS ini harus dievaluasi untuk mengetahui potensi batas kapasitas keras ketika digunakan dalam desain aplikasi.

Batas keras ditunjukkan di konsol Service Quotas. Jika kolom menunjukkan `DAPAT DISESUAIKAN = Tidak`, layanan memiliki batas keras. Batas keras juga ditunjukkan di beberapa halaman konfigurasi sumber daya. Contohnya, Lambda memiliki batas keras spesifik yang tidak dapat disesuaikan.

Sebagai contoh, ketika mendesain aplikasi python untuk beroperasi dalam fungsi Lambda, aplikasi harus dievaluasi untuk menentukan apakah ada peluang Lambda akan beroperasi lebih lama dari 15 menit. Jika kode mungkin akan dijalankan lebih dari batas kuota layanan ini, desain atau teknologi lain harus dipertimbangkan. Jika batas ini tercapai setelah deployment produksi, aplikasi akan mengalami penurunan kualitas dan gangguan sampai dapat diperbaiki. Tidak seperti kuota lunak, tidak ada metode untuk mengubah batas-batas ini meskipun dalam kondisi darurat Keparahan 1.

Setelah aplikasi telah di-deploy ke lingkungan pengujian, strategi harus digunakan untuk menemukan apakah batas keras dapat tercapai. Pengujian stres, pengujian beban, dan pengujian kekacauan harus menjadi bagian dari rencana pengujian pendahuluan.

Langkah implementasi

- Tinjau daftar lengkap layanan AWS yang dapat digunakan dalam fase desain aplikasi.
- Tinjau batas kuota lunak dan batas kuota keras untuk semua layanan ini. Tidak semua batas ditunjukkan di konsol Service Quotas. Beberapa layanan menjelaskan batas-batas ini di lokasi lain.
- Saat Anda mendesain aplikasi Anda, tinjau pendorong teknologi dan bisnis beban kerja Anda, seperti hasil bisnis, kasus penggunaan, sistem yang dependen, target ketersediaan, dan objek pemulihan bencana. Biarkan pendorong teknologi dan bisnis Anda memandu proses untuk mengidentifikasi sistem terdistribusi yang tepat untuk beban kerja Anda.
- Analisis beban layanan di berbagai Wilayah dan akun. Banyak batas keras yang berbasis wilayah untuk layanan. Tetapi, beberapa batas berbasis akun.
- Analisis arsitektur ketangguhan untuk penggunaan sumber daya selama kegagalan zona dan kegagalan Wilayah. Jika progres desain multi-Wilayah menggunakan pendekatan aktif/aktif, aktif/pasif - panas, aktif/pasif - dingin, dan aktif/pasif - pilot light, kasus-kasus kegagalan ini akan menyebabkan penggunaan yang lebih tinggi. Hal ini akan menimbulkan potensi kasus penggunaan untuk mencapai batas keras.

Sumber daya

Praktik Terbaik Terkait:

- REL01-BP01 Kesadaran tentang kuota dan kendala layanan
- REL01-BP02 Mengelola kuota layanan di seluruh akun dan wilayah
- REL01-BP04 Memantau dan mengelola kuota
- REL01-BP05 Mengotomatiskan manajemen kuota
- REL01-BP06 Memastikan adanya selisih yang memadai antara kuota saat ini dan penggunaan maksimum untuk mengakomodasi failover
- REL03-BP01 Memilih cara untuk menyegmentasi beban kerja
- REL10-BP01 Melakukan deployment beban kerja ke beberapa lokasi
- REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan
- REL11-BP03 Mengotomatisasi pemulihan di semua lapisan
- REL12-BP05 Menguji ketahanan menggunakan chaos engineering

Dokumen terkait:

- Pilar Keandalan Kerangka Kerja AWS Well-Architected: Ketersediaan
- AWS Service Quotas (sebelumnya disebut batas layanan)
- AWS Trusted Advisor Pemeriksaan Praktik Terbaik (lihat bagian Batas Layanan)
- Pemantau batas AWS di AWS Answers
- Batas Layanan Amazon EC2
- Apa itu Service Quotas?
- Cara Meminta Peningkatan Kuota
- Endpoint dan kuota layanan
- Panduan Pengguna Service Quotas
- Pemantau Kuota untuk AWS
- Batas Isolasi Kesalahan AWS
- Ketersediaan dengan redundansi
- AWS untuk Data
- Apa itu Integrasi Berkelanjutan?
- Apa itu Pengiriman Berkelanjutan?

- Partner APN: partner yang dapat membantu manajemen konfigurasi
- Mengelola siklus hidup akun di dalam lingkungan SaaS akun per tenant di AWS
- Mengelola dan memanfatau throttling API di dalam beban kerja Anda
- Lihat rekomendasi AWS Trusted Advisor pada skala besar dengan AWS Organizations
- Mengotomatiskan Peningkatan Batas Layanan dan Dukungan Korporat dengan AWS Control Tower
- Tindakan, sumber daya, dan kunci kondisi untuk Service Quotas

Video terkait:

- AWS Live re:Inforce 2019 - Service Quotas
- Melihat dan Mengelola Kuota untuk Layanan AWS Menggunakan Service Quotas
- Demo Kuota AWS IAM
- AWS re:Invent 2018: Menutup Lingkaran dan Membuka Pikiran: Cara Mengendalikan Sistem, Besar dan Kecil

Alat terkait:

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP04 Memantau dan mengelola kuota

Evaluasi potensi penggunaan Anda dan tingkatkan kuota dengan semestinya, sehingga terjadi pertumbuhan penggunaan sesuai rencana.

Hasil yang diinginkan: Sistem aktif dan otomatis yang mengelola dan memantau telah di-deploy. Solusi operasi ini memastikan ambang batas penggunaan kuota hampir dicapai. Hal ini akan secara proaktif diperbaiki oleh perubahan kuota yang diminta.

Antipola umum:

- Tidak mengonfigurasi pemantauan untuk memeriksa ambang batas kuota layanan
- Tidak mengonfigurasi pemantauan untuk batas keras, meskipun nilai-nilai tersebut tidak dapat diubah.
- Berasumsi bahwa jumlah waktu yang diperlukan untuk meminta dan mendapatkan perubahan kuota lunak adalah segera atau singkat.
- Mengonfigurasi alarm untuk ketika kuota layanan sudah dekat, namun tidak memiliki proses untuk merespons pemberitahuan.
- Hanya mengonfigurasi alarm untuk layanan yang didukung oleh AWS Service Quotas dan tidak memantau layanan AWS lain.
- Tidak mempertimbangkan manajemen kuota untuk beberapa desain ketangguhan Wilayah, seperti pendekatan aktif/aktif, aktif/pasif - panas, aktif/pasif - dingin, dan aktif/pasif - pilot light.
- Tidak menilai perbedaan kuota antara Wilayah.
- Tidak menilai kebutuhan di setiap Wilayah untuk permintaan peningkatan kuota spesifik.
- Tidak memanfaatkan templat untuk manajemen kuota multi-Wilayah.

Manfaat menjalankan praktik terbaik ini: Dengan melacak AWS Service Quotas secara otomatis dan memantau penggunaan Anda berdasarkan kuota tersebut, Anda dapat mengetahui ketika batas kuota hampir terpenuhi. Anda juga dapat menggunakan data pemantauan ini untuk membantu membatasi penurunan kualitas karena kehabisan kuota.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Untuk layanan yang didukung, Anda dapat memantau kuota Anda dengan mengonfigurasikan berbagai layanan yang berbeda yang dapat menilai kemudian mengirimkan pemberitahuan atau alarm. Hal ini dapat membantu memantau penggunaan dan dapat memberitahukan kuota yang akan habis kepada Anda. Alarm ini dapat dipicu dari AWS Config, fungsi Lambda, Amazon CloudWatch, atau dari AWS Trusted Advisor. Anda juga dapat menggunakan filter metrik di Log CloudWatch untuk

mencari dan mengekstrak pola dalam log untuk menentukan apakah penggunaan sudah mendekati ambang batas kuota.

Langkah implementasi

Untuk pemantauan:

- Catat pemakaian sumber daya saat ini (misalnya, bucket atau instans). Gunakan operasi API layanan, seperti Amazon EC2 `DescribeInstances` API, untuk mengumpulkan pemakaian sumber daya saat ini.
- Catat kuota saat ini yang penting dan berlaku untuk layanan menggunakan:
  - AWS Service Quotas
  - AWS Trusted Advisor
  - Dokumentasi AWS
  - Halaman spesifik layanan AWS
  - AWS Command Line Interface (AWS CLI)
  - AWS Cloud Development Kit (AWS CDK)
- Gunakan AWS Service Quotas, yakni layanan AWS yang membantu Anda mengelola kuota untuk lebih dari 250 layanan AWS dari satu lokasi.
- Gunakan batas layanan Trusted Advisor untuk memantau batas layanan Anda saat ini di berbagai ambang batas.
- Gunakan riwayat kuota layanan (konsol atau AWS CLI) untuk memeriksa peningkatan regional.
- Bandingkan perubahan kuota layanan di setiap Wilayah dan setiap akun untuk membuat kesetaraan, jika diperlukan.

Untuk manajemen:

- Otomatis: Siapkan aturan kustom AWS Config untuk memindai kuota layanan di berbagai Wilayah dan bandingkan untuk mengetahui perbedaannya.
- Otomatis: Siapkan fungsi Lambda terjadwal untuk memindai kuota layanan di berbagai Wilayah dan bandingkan untuk mengetahui perbedaannya.
- Manual: Pindai kuota layanan melalui AWS CLI, API, atau Konsol AWS untuk memindai kuota layanan di berbagai Wilayah dan bandingkan untuk mengetahui perbedaannya. Laporkan perbedaannya.
- Jika perbedaan kuota diidentifikasi antara Wilayah, minta perubahan kuota, jika perlu.

- Tinjau hasil dari semua permintaan.

Sumber daya

Praktik Terbaik Terkait:

- REL01-BP01 Kesadaran tentang kuota dan kendala layanan
- REL01-BP02 Mengelola kuota layanan di seluruh akun dan wilayah
- REL01-BP03 Mengakomodasi kuota layanan tetap dan kendala melalui arsitektur
- REL01-BP05 Mengotomatiskan manajemen kuota
- REL01-BP06 Memastikan adanya selisih yang memadai antara kuota saat ini dan penggunaan maksimum untuk mengakomodasi failover
- REL03-BP01 Memilih cara untuk menyegmentasi beban kerja
- REL10-BP01 Melakukan deployment beban kerja ke beberapa lokasi
- REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan
- REL11-BP03 Mengotomatisasi pemulihan di semua lapisan
- REL12-BP05 Menguji ketahanan menggunakan chaos engineering

Dokumen terkait:

- Pilar Keandalan Kerangka Kerja AWS Well-Architected: Ketersediaan
- AWS Service Quotas (sebelumnya disebut batas layanan)
- AWS Trusted Advisor Pemeriksaan Praktik Terbaik (lihat bagian Batas Layanan)
- Pemantau batas AWS di AWS Answers
- Batas Layanan Amazon EC2
- Apa itu Service Quotas?
- Cara Meminta Peningkatan Kuota
- Endpoint dan kuota layanan
- Panduan Pengguna Service Quotas
- Pemantau Kuota untuk AWS
- Batas Isolasi Kesalahan AWS
- Ketersediaan dengan redundansi

- AWS untuk Data
- Apa itu Integrasi Berkelanjutan?
- Apa itu Pengiriman Berkelanjutan?
- Partner APN: partner yang dapat membantu manajemen konfigurasi
- Mengelola siklus hidup akun di dalam lingkungan SaaS akun per tenant di AWS
- Mengelola dan memanfatau throttling API di dalam beban kerja Anda
- Lihat rekomendasi AWS Trusted Advisor pada skala besar dengan AWS Organizations
- Mengotomatiskan Peningkatan Batas Layanan dan Dukungan Korporat dengan AWS Control Tower
- Tindakan, sumber daya, dan kunci kondisi untuk Service Quotas

Video terkait:

- AWS Live re:Inforce 2019 - Service Quotas
- Melihat dan Mengelola Kuota untuk Layanan AWS Menggunakan Service Quotas
- Demo Kuota AWS IAM
- AWS re:Invent 2018: Menutup Lingkaran dan Membuka Pikiran: Cara Mengendalikan Sistem, Besar dan Kecil

Alat terkait:

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP05 Mengotomatiskan manajemen kuota

Implementasikan alat untuk memperingatkan Anda saat ambang batas terlampaui. Anda dapat mengotomatiskan permintaan peningkatan kuota dengan menggunakan API AWS Service Quotas, Anda dapat mengotomatiskan permintaan peningkatan kuota.

Jika Anda mengintegrasikan Basis Data Manajemen Konfigurasi (CMDB) atau sistem ticketing dengan Service Quotas, Anda dapat mengotomatiskan permintaan peningkatan kuota dan kuota saat ini. Selain SDK AWS, Service Quotas menawarkan otomatisasi menggunakan AWS Command Line Interface (AWS CLI).

Antipola umum:

- Melacak kuota dan penggunaan dalam spreadsheet.
- Menjalankan laporan pada penggunaan harian, mingguan, bulanan, lalu membandingkan penggunaan dengan kuota.

Manfaat menerapkan praktik terbaik ini: Dengan pelacakan kuota layanan AWS dan pemantauan terhadap penggunaan kuota, Anda dapat mengetahui ketika kuota hampir penuh. Anda dapat mengatur otomatisasi untuk membantu meminta peningkatan kuota saat diperlukan. Anda dapat mempertimbangkan pengurangan kuota saat penggunaan Anda cenderung tidak selaras untuk benar-benar mengoptimalkan manfaat dari pengurangan risiko (dalam kasus kredensial yang disusupi) dan penghematan biaya.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Sedang

Panduan implementasi

- Atur pemantauan otomatis: Implementasikan alat dengan menggunakan SDK untuk memperingatkan saat ambang batas terlampaui.
  - Gunakan Service Quotas dan tingkatkan layanan dengan solusi pemantauan kuota otomatis, seperti AWS Limit Monitor atau penawaran dari AWS Marketplace.
    - Apa itu Service Quotas?
    - Monitor Kuota di AWS - Solusi AWS
  - Atur respons terpicu berdasarkan ambang batas kuota menggunakan Amazon SNS dan API AWS Service Quotas.
  - Uji otomatisasi.

- Konfigurasikan ambang batas.
- Integrasikan dengan peristiwa perubahan dari AWS Config, pipeline deployment, Amazon EventBridge, atau pihak ketiga.
- Atur ambang batas kuota rendah secara artifisial untuk menguji respons.
- Atur pemicu untuk mengambil tindakan yang sesuai berdasarkan notifikasi dan hubungi AWS Support jika diperlukan.
- Picu peristiwa perubahan secara manual.
- Jalankan game day untuk menguji proses perubahan peningkatan kuota.

Sumber daya

Dokumen terkait:

- Partner APN: partner yang dapat membantu manajemen konfigurasi
- AWS Marketplace: Produk CMDB yang membantu melacak batasan
- AWS Service Quotas (yang sebelumnya dikenal sebagai batas layanan)
- Pemeriksaan Praktik Terbaik AWS Trusted Advisor (lihat bagian Batas Layanan)
- Monitor Kuota di AWS - Solusi AWS
- Batas Layanan Amazon EC2
- Apa itu Service Quotas?

Video terkait:

- AWS Live re:Inforce 2019 - Service Quotas

REL01-BP06 Memastikan adanya selisih yang memadai antara kuota saat ini dan penggunaan maksimum untuk mengakomodasi failover

Ketika sumber daya gagal atau tidak dapat diakses, sumber daya tersebut mungkin masih dihitung untuk kuota sampai berhasil dihentikan. Verifikasi apakah kuota meliputi tumpang tindih sumber daya yang gagal atau tidak dapat diakses dan penggantiannya. Anda harus mempertimbangkan kasus penggunaan seperti kegagalan jaringan, kegagalan Zona Ketersediaan, atau kegagalan Wilayah ketika menghitung selisih ini.

Hasil yang diinginkan: Kegagalan kecil atau besar dalam sumber daya atau kemampuan akses sumber daya dapat diatasi dalam ambang batas layanan saat ini. Kegagalan zona, kegagalan jaringan, atau bahkan kegagalan Wilayah telah dipertimbangkan dalam perencanaan sumber daya.

Antipola umum:

- Mengatur kuota layanan berdasarkan kebutuhan saat ini tanpa memperhitungkan skenario failover.
- Tidak mempertimbangkan prinsipal stabilitas statis ketika menghitung kuota puncak untuk layanan.
- Tidak mempertimbangkan potensi sumber daya yang tidak dapat diakses dalam menghitung kuota total yang diperlukan untuk setiap Wilayah.
- Tidak mempertimbangkan batas isolasi kesalahan layanan AWS untuk beberapa layanan dan potensi pola penggunaan abnormalnya.

Manfaat menjalankan praktik terbaik ini: Ketika peristiwa gangguan layanan memengaruhi ketersediaan aplikasi, cloud memungkinkan Anda untuk mengimplementasikan strategi guna memitigasi atau memulihkan dari peristiwa ini. Strategi tersebut sering kali mencakup pembuatan sumber daya tambahan untuk menggantikan sumber daya yang gagal atau tidak dapat diakses. Strategi kuota Anda akan mengakomodasi kondisi failover ini dan tidak menambahkan lapisan degradasi tambahan akibat tercapainya batas layanan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Ketika mengevaluasi batas kuota, pertimbangkan kasus failover yang dapat terjadi karena degradasi. Jenis kasus failover berikut ini harus dipertimbangkan:

- VPC yang terganggu atau tidak dapat diakses.
- Subnet yang tidak dapat diakses.
- Zona Ketersediaan telah cukup mengalami degradasi sehingga memengaruhi kemampuan akses banyak sumber daya.
- Berbagai titik keluar dan masuk atau rute jaringan terblokir atau berubah.
- Wilayah telah cukup mengalami degradasi sehingga memengaruhi kemampuan akses banyak sumber daya.
- Ada beberapa sumber daya tetapi tidak semua terpengaruh oleh kegagalan di Wilayah atau Zona Ketersediaan.

Kegagalan seperti yang tercantum di atas dapat menjadi pemicu awal terjadinya peristiwa failover. Keputusan untuk failover bersifat unik untuk setiap situasi dan pelanggan, karena dampak bisnisnya bisa sangat berbeda. Tetapi, ketika memutuskan untuk failover aplikasi atau layanan secara operasional, perencanaan kapasitas sumber daya di lokasi failover dan kuota terkait harus ditangani sebelum peristiwa terjadi.

Tinjau kuota layanan untuk setiap layanan dengan mempertimbangkan puncak yang lebih tinggi dari normal yang mungkin terjadi. Puncak ini mungkin terkait dengan sumber daya yang dapat dicapai karena jaringan atau izin tetapi masih aktif. Sumber daya aktif yang tidak dihentikan akan masih dihitung untuk memenuhi batas kuota layanan.

Langkah implementasi

- Verifikasi bahwa ada selisih yang memadai antara kuota layanan saat ini dan penggunaan maksimum untuk mengakomodasi failover atau hilangnya kemampuan akses.
- Tentukan kuota layanan, perhitungkan pola deployment, persyaratan ketersediaan, dan peningkatan pemakaian.
- Minta peningkatan kuota jika perlu. Rencanakan waktu yang diperlukan agar permintaan peningkatan kuota dapat terpenuhi.
- Tentukan persyaratan keandalan (juga disebut sebagai jumlah angka sembilan Anda).
- Tetapkan skenario kesalahan (misalnya kehilangan komponen, Zona Ketersediaan, atau Wilayah).
- Tetapkan metodologi deployment (misalnya canary, blue/green, red/black, atau rolling).
- Sertakan buffer yang sesuai (misalnya, 15%) ke batas saat ini.
- Sertakan perhitungan untuk stabilitas statis (Zona dan Wilayah) apabila sesuai.
- Rencanakan peningkatan pemakaian (misalnya, memantau tren pemakaian).
- Pertimbangkan dampak stabilitas statis untuk beban kerja Anda yang paling penting. Nilai sumber daya yang sesuai dengan sistem stabil secara statis di semua Wilayah dan Zona Ketersediaan.
- Pertimbangkan penggunaan Reservasi Kapasitas Sesuai Permintaan untuk menjadwalkan kapasitas sebelum failover. Hal ini dapat menjadi strategi yang bermanfaat untuk penjadwalan bisnis yang paling penting guna mengurangi potensi risiko mendapatkan kuantitas dan jenis sumber daya yang benar selama failover.

Sumber daya

Praktik Terbaik Terkait:

- REL01-BP01 Kesadaran tentang kuota dan kendala layanan
- REL01-BP02 Mengelola kuota layanan di seluruh akun dan wilayah
- REL01-BP03 Mengakomodasi kuota layanan tetap dan kendala melalui arsitektur
- REL01-BP04 Memantau dan mengelola kuota
- REL01-BP05 Mengotomatiskan manajemen kuota
- REL03-BP01 Memilih cara untuk menyegmentasi beban kerja
- REL10-BP01 Melakukan deployment beban kerja ke beberapa lokasi
- REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan
- REL11-BP03 Mengotomatisasi pemulihan di semua lapisan
- REL12-BP05 Menguji ketahanan menggunakan chaos engineering

Dokumen terkait:

- Pilar Keandalan Kerangka Kerja AWS Well-Architected: Ketersediaan
- AWS Service Quotas (sebelumnya disebut batas layanan)
- AWS Trusted Advisor Pemeriksaan Praktik Terbaik (lihat bagian Batas Layanan)
- Pemantau batas AWS di AWS Answers
- Batas Layanan Amazon EC2
- Apa itu Service Quotas?
- Cara Meminta Peningkatan Kuota
- Endpoint dan kuota layanan
- Panduan Pengguna Service Quotas
- Pemantau Kuota untuk AWS
- Batas Isolasi Kesalahan AWS
- Ketersediaan dengan redundansi
- AWS untuk Data
- Apa itu Integrasi Berkelanjutan?
- Apa itu Pengiriman Berkelanjutan?
- Partner APN: partner yang dapat membantu manajemen konfigurasi
- Mengelola siklus hidup akun di dalam lingkungan SaaS akun per tenant di AWS

- Mengelola dan memanfatau throttling API di dalam beban kerja Anda
- Lihat rekomendasi AWS Trusted Advisor pada skala besar dengan AWS Organizations
- Mengotomatiskan Peningkatan Batas Layanan dan Dukungan Korporat dengan AWS Control Tower
- Tindakan, sumber daya, dan kunci kondisi untuk Service Quotas

Video terkait:

- AWS Live re:Inforce 2019 - Service Quotas
- Melihat dan Mengelola Kuota untuk Layanan AWS Menggunakan Service Quotas
- Demo Kuota AWS IAM
- AWS re:Invent 2018: Menutup Lingkaran dan Membuka Pikiran: Cara Mengendalikan Sistem, Besar dan Kecil

Alat terkait:

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

Merencanakan topologi jaringan Anda

Beban kerja sering kali ada di beberapa lingkungan. Termasuk di dalamnya adalah beberapa lingkungan cloud (baik yang dapat diakses publik maupun privat) dan kemungkinan juga infrastruktur pusat data Anda yang sudah ada. Rencana harus mencakup pertimbangan jaringan, seperti

konektivitas dalam sistem dan antarsistem, pengelolaan alamat IP publik, pengelolaan alamat IP privat, dan resolusi nama domain.

Saat merancang sistem menggunakan jaringan berbasis alamat IP, Anda harus merencanakan topologi dan penanganan jaringan untuk mengantisipasi kemungkinan kegagalan, dan untuk mengakomodasi pertumbuhan dan integrasi mendatang dengan sistem-sistem lain serta jaringannya.

Amazon Virtual Private Cloud (Amazon VPC) memungkinakn Anda menyediakan bagian AWS Cloud privat yang terisolasi di mana Anda dapat meluncurkan sumber daya AWS dalam sebuah jaringan virtual.

Praktik terbaik

- REL02-BP01 Menggunakan konektivitas jaringan dengan ketersediaan tinggi untuk titik akhir publik beban kerja Anda
- REL02-BP02 Menyediakan konektivitas redundan antara jaringan privat di cloud dan lingkungan on-premise
- REL02-BP03 Pastikan alokasi subnet IP menjelaskan ekspansi dan ketersediaan
- REL02-BP04 Mengutamakan topologi hub-and-spoke daripada mesh many-to-many
- REL02-BP05 Terapkan rentang alamat IP privat yang tidak tumpang tindih di semua ruang alamat privat tempat semuanya terhubung

REL02-BP01 Menggunakan konektivitas jaringan dengan ketersediaan tinggi untuk titik akhir publik beban kerja Anda

Membuat konektivitas jaringan dengan ketersediaan tinggi untuk titik akhir publik beban kerja Anda dapat membantu Anda mengurangi waktu henti karena hilangnya konektivitas dan meningkatkan ketersediaan serta SLA beban kerja Anda. Untuk mencapai ini, gunakan DNS dengan ketersediaan tinggi, jaringan pengiriman konten (CDN), gateway API, penyeimbangan beban, atau proksi mundur.

Hasil yang diinginkan: Sangat penting untuk merencanakan, membuat, dan mengoperasikan konektivitas jaringan dengan ketersediaan tinggi untuk titik akhir publik Anda. Jika beban kerja Anda menjadi tidak terjangkau karena hilangnya konektivitas, meskipun beban kerja Anda beroperasi dan tersedia, pelanggan Anda akan menganggap sistem Anda tidak berfungsi. Dengan menggabungkan konektivitas jaringan yang tangguh dan memiliki ketersediaan tinggi untuk titik akhir publik beban kerja Anda, bersama dengan arsitektur tangguh untuk beban kerja itu sendiri, Anda dapat memberikan tingkat layanan dan ketersediaan yang sebaik mungkin untuk pelanggan Anda.

AWS Global Accelerator, Amazon CloudFront, Amazon API Gateway, URL Fungsi AWS Lambda, AWS AppSync API, dan Elastic Load Balancing (ELB) memberikan titik akhir publik dengan ketersediaan tinggi. Amazon Route 53 memberikan layanan DNS dengan ketersediaan tinggi untuk resolusi nama domain guna memverifikasi bahwa alamat titik akhir publik Anda dapat diatur.

Anda juga dapat mengevaluasi peralatan perangkat lunak AWS Marketplace untuk penyeimbangan beban dan proksi.

Antipola umum:

- Mendesain beban kerja dengan ketersediaan tinggi tanpa merencanakan konektivitas jaringan dan DNS untuk ketersediaan tinggi.
- Menggunakan alamat internet publik di instans atau kontainer secara individu dan mengelola konektivitasnya dengan DNS.
- Menggunakan alamat IP dan bukannya nama domain untuk mencari layanan.
- Tidak menguji skenario di mana konektivitas ke titik akhir publik Anda hilang.
- Tidak menganalisis pola distribusi dan kebutuhan throughput jaringan.
- Tidak menguji dan merencanakan skenario di mana konektivitas jaringan internet ke titik akhir publik beban kerja Anda mungkin terganggu.
- Memberikan konten (seperti halaman web, aset statis, atau file media) ke area geografis besar dan tidak menggunakan CDN.
- Tidak membuat rencana untuk serangan penolakan layanan terdistribusi (DDoS). Serangan DDoS berisiko menghalangi lalu lintas sah dan menurunkan ketersediaan untuk pengguna Anda.

Manfaat menjalankan praktik terbaik ini: Mendesain konektivitas jaringan yang tangguh dan memiliki ketersediaan tinggi memastikan beban kerja Anda dapat diakses dan tersedia bagi pengguna.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Inti dari pembuatan konektivitas jaringan dengan ketersediaan tinggi untuk titik akhir publik adalah pengarahan rute lalu lintas. Untuk memverifikasi lalu lintas Anda dapat menjangkau titik akhir, DNS harus dapat mengatur nama domain ke alamat IP-nya yang bersangkutan. Gunakan Sistem Nama Domain (DNS) yang dapat diskalakan dan memiliki ketersediaan tinggi seperti Amazon Route 53 untuk mengelola data DNS domain Anda. Anda juga dapat menggunakan pemeriksaan kondisi

yang disediakan oleh Amazon Route 53. Pemeriksaan kondisi memverifikasi bahwa aplikasi Anda dapat dijangkau, tersedia, dan berfungsi, dan pemeriksaan ini dapat diatur sedemikian sehingga menyerupai perilaku pengguna Anda, seperti meminta halaman web atau URL tertentu. Jika terjadi kegagalan, Amazon Route 53 merespons permintaan resolusi DNS dan mengarahkan lalu lintas ke titik akhir dengan kondisi bagus saja. Anda juga dapat mempertimbangkan penggunaan kemampuan Perutean Berbasis Latensi dan DNS Geo yang ditawarkan oleh Amazon Route 53.

Untuk memverifikasi bahwa beban kerja itu sendiri memiliki ketersediaan tinggi, gunakan Elastic Load Balancing (ELB). Amazon Route 53 dapat digunakan untuk menargetkan lalu lintas ke ELB, yang mendistribusikan lalu lintas ke instans komputasi target. Anda juga dapat menggunakan Amazon API Gateway bersama dengan AWS Lambda untuk solusi nirserver. Pelanggan juga dapat menjalankan beban kerja di beberapa Wilayah AWS. Dengan pola aktif/aktif multi-lokasi, beban kerja dapat menghadirkan lalu lintas dari beberapa Wilayah. Dengan pola aktif/pasif multi-lokasi, beban kerja menghadirkan lalu lintas dari wilayah aktif sementara data direplikasikan ke wilayah sekunder dan menjadi aktif untuk berjaga-jaga jika terjadi kegagalan di wilayah utama. Kemudian pemeriksaan kondisi Route 53 dapat digunakan untuk mengontrol failover DNS dari titik akhir mana pun di Wilayah utama ke titik akhir di Wilayah sekunder, dengan memverifikasi bahwa beban kerja Anda dapat dijangkau dan tersedia bagi pengguna Anda.

Amazon CloudFront memberikan API sederhana untuk mendistribusikan konten dengan laju transfer data tinggi dan latensi rendah dengan melayani permintaan menggunakan jaringan lokasi edge di seluruh dunia. Jaringan pengiriman konten (CDN) melayani pelanggan dengan menghadirkan konten yang berada di atau di-cache di lokasi yang dekat dengan pengguna. Hal ini juga meningkatkan ketersediaan aplikasi Anda karena beban untuk konten dialihkan dari server Anda ke CloudFront, yakni ke lokasi edge-nya. Lokasi edge dan cache edge regional menyimpan cache salinan konten Anda dekat dengan penonton Anda sehingga konten dapat diambil dengan cepat, konten lebih mudah dijangkau, dan beban kerja memiliki ketersediaan lebih tinggi.

Untuk beban kerja dengan pengguna yang tersebar secara geografis,AWS Global Accelerator membantu Anda meningkatkan ketersediaan dan performa aplikasi. AWS Global Accelerator memberikan alamat IP statis Anycast yang berfungsi sebagai titik masuk tetap ke aplikasi Anda yang di-host di satu atau lebih Wilayah AWS. Hal ini memungkinkan lalu lintas masuk ke jaringan global AWS sedekat mungkin ke pengguna Anda, yang meningkatkan keterjangkauan dan ketersediaan beban kerja Anda. AWS Global Accelerator juga memantau kondisi titik akhir aplikasi Anda menggunakan TCP, HTTP, dan pemeriksaan kondisi HTTPS. Setiap perubahan kondisi atau konfigurasi titik akhir Anda memicu pengarahan ulang lalu lintas pengguna ke titik akhir dengan kondisi bagus yang memberikan ketersediaan dan performa terbaik bagi pengguna Anda. Selain itu, AWS Global Accelerator memiliki desain yang mengisolasi kesalahan yang menggunakan dua alamat

IPv4 status yang dilayani oleh zona jaringan mandiri sehingga meningkatkan ketersediaan aplikasi Anda.

Untuk membantu melindungi pelanggan dari serangan DDoS, AWS memberikan AWS Shield Standard. Shield Standard tersedia sudah secara otomatis diaktifkan dan melindungi dari serangan infrastruktur umum (lapisan 3 dan 4) seperti SYN/UDP flood dan serangan refleksi untuk mendukung ketersediaan tinggi aplikasi Anda di AWS. Untuk perlindungan tambahan dari serangan yang lebih besar dan lebih canggih (seperti UDP flood), serangan penghentian layanan (seperti TCP SYN flood), dan untuk membantu melindungi aplikasi Anda dijalankan di Amazon Elastic Compute Cloud (Amazon EC2), Elastic Load Balancing (ELB), Amazon CloudFront, AWS Global Accelerator, dan Route 53, Anda dapat mempertimbangkan penggunaan AWS Shield Advanced. Untuk perlindungan dari serangan lapisan Aplikasi seperti HTTP POST atau GET flood, gunakan AWS WAF. AWS WAF dapat menggunakan alamat IP, header HTTP, bodi HTTP, string URI, injeksi SQL, dan kondisi skrip lintas situs untuk menentukan apakah permintaan harus diblokir atau diizinkan.

Langkah implementasi

1. Siapkan DNS dengan ketersediaan tinggi: Amazon Route 53 adalah layanan web sistem nama domain (DNS) yang dapat diskalakan dan memiliki ketersediaan tinggi. Route 53 menghubungkan permintaan pengguna dengan aplikasi internet yang dijalankan di AWS atau on-premise. Untuk informasi selengkapnya, lihat mengonfigurasi Amazon Route 53 sebagai layanan DNS Anda.
2. Siapkan pemeriksaan kondisi: Ketika menggunakan Route 53, verifikasi bahwa hanya target dengan kondisi bagus yang dapat diselesaikan. Mulai dengan membuat pemeriksaan kondisi Route 53 dan mengonfigurasi failover DNS. Aspek-aspek berikut penting untuk dipertimbangkan ketika mempersiapkan pemeriksaan kondisi:
   a. Bagaimana Amazon Route 53 menentukan apakah pemeriksaan kondisi bagus
   b. Membuat, memperbarui, dan menghapus pemeriksaan kondisi
   c. Memantau status pemeriksaan kondisi dan mendapatkan pemberitahuan
   d. Praktik terbaik untuk Amazon Route 53 DNS
3. Hubungkan layanan DNS Anda ke titik akhir Anda.
   a. Ketika menggunakan Elastic Load Balancing sebagai target untuk lalu lintas Anda, buat catatan alias menggunakan Amazon Route 53 yang menunjuk ke titik akhir wilayah penyeimbang beban Anda. Selama pembuatan catatan alias, atur opsi Evaluasi kondisi target ke Ya.
   b. Untuk beban kerja nirserver atau API privat ketika API Gateway digunakan, gunakan Route 53 untuk mengarahkan lalu lintas ke API Gateway.
4. Tentukan jaringan pengiriman konten.

a. Untuk pengiriman konten menggunakan lokasi edge yang lebih dekat dengan pengguna, mulai dengan memahami cara CloudFront memberikan konten.

b. Mulai dengan distribusi CloudFront sederhana. Kemudian CloudFront mengetahui dari mana Anda ingin konten dikirimkan, dan detail tentang cara melacak dan mengelola pengiriman konten. Aspek-aspek berikut penting untuk dipahami dan dipertimbangkan ketika mempersiapkan distribusi CloudFront:

   i. Cara kerja caching dengan lokasi edge CloudFront

   ii. Meningkatkan proporsi permintaan yang dihadirkan secara langsung dari cache CloudFront (rasio sukses cache)

   iii. Menggunakan Amazon CloudFront Origin Shield

   iv. Mengoptimalkan ketersediaan tinggi dengan failover asalah CloudFront

5. Siapkan perlindungan lapisan aplikasi: AWS WAF membantu Anda melindungi dari bot dan eksploitasi web umum yang dapat memengaruhi ketersediaan, mengancam keamanan, atau memakai sumber daya secara berlebihan. Untuk mendapatkan pemahaman lebih mendalam, tinjau cara kerja AWS WAF dan kapan Anda siap untuk mengimplementasikan perlindungan dari HTTP POST DAN GET flood lapisan aplikasi, tinjau Memulai AWS WAF. Anda juga dapat menggunakan AWS WAF dengan CloudFront, lihat dokumentasi tentang cara AWS WAF berfungsi dengan fitur Amazon CloudFront.
6. Siapkan perlindungan DDoS tambahan: Menurut default, semua pelanggan AWS menerima perlindungan dari serangan DDoS lapisan transpor dan jaringan paling sering terjadi yang menargetkan situs web atau aplikasi Anda, dengan menggunakan AWS Shield Standard tanpa biaya tambahan. Untuk perlindungan tambahan aplikasi yang dilihat internet, dijalankan di Amazon EC2, Elastic Load Balancing, Amazon CloudFront, AWS Global Accelerator, dan Amazon Route 53, Anda dapat mempertimbangkan AWS Shield Advanced dan meninjau contoh tentang arsitektur yang tangguh terhadap DDoS. Untuk melindungi beban kerja dan titik akhir publik Anda dari serangan DDoS, tinjau Memulai dengan AWS Shield Advanced.

Sumber daya

Praktik Terbaik Terkait:

- REL10-BP01 Melakukan deployment beban kerja ke beberapa lokasi
- REL10-BP02 Memilih lokasi yang sesuai untuk deployment multilokasi
- REL11-BP04 Mengandalkan bidang data dan bukan bidang kendali selama pemulihan
- REL11-BP06 Mengirimkan notifikasi ketika peristiwa memengaruhi ketersediaan

Dokumen terkait:

- Partner APN: partner yang dapat membantu merencanakan jaringan Anda
- AWS Marketplace untuk Infrastruktur Jaringan
- Apa Itu AWS Global Accelerator?
- Apa itu Amazon CloudFront?
- Apa itu Amazon Route 53?
- Apa itu Elastic Load Balancing?
- Kemampuan Konektivitas Jaringan - Menetapkan Fondasi Cloud Anda
- Apa itu Amazon API Gateway?
- Apa itu AWS WAF, AWS Shield, dan AWS Firewall Manager?
- Apa itu Pengontrol Pemulihan Aplikasi Amazon Route 53?
- Konfigurasi pemeriksaan kondisi kustom untuk failover DNS

Video terkait:

- AWS re:Invent 2022 - Meningkatkan performa dan ketersediaan dengan AWS Global Accelerator
- AWS re:Invent 2020: Manajemen lalu lintas global dengan Amazon Route 53
- AWS re:Invent 2022 - Mengoperasikan aplikasi Multi-AZ dengan ketersediaan tinggi
- AWS re:Invent 2022 - Memahami infrastruktur jaringan AWS
- AWS re:Invent 2022 - Membangun jaringan tangguh

Contoh terkait:

- Pemulihan Bencana dengan Pengontrol Pemulihan Aplikasi (ARC) Amazon Route 53
- Lokakarya Keandalan
- Lokakarya AWS Global Accelerator

REL02-BP02 Menyediakan konektivitas redundan antara jaringan privat di cloud dan lingkungan on-premise

Implementasikan redundansi dalam koneksi Anda antara jaringan pribadi di cloud dan lingkungan on-premise untuk mencapai ketahanan konektivitas. Ini dapat dicapai dengan melakukan deployment

dua atau lebih tautan dan jalur lalu lintas, sehingga menjaga konektivitas jika terjadi kegagalan jaringan.

Antipola umum:

- Anda bergantung hanya pada satu koneksi jaringan, yang menciptakan satu titik kegagalan.
- Anda hanya menggunakan satu terowongan VPN atau beberapa terowongan yang berakhir di Zona Ketersediaan yang sama.
- Anda mengandalkan satu ISP untuk konektivitas VPN, yang dapat menyebabkan kegagalan total selama pemadaman ISP.
- Tidak mengimplementasikan protokol perutean dinamis seperti BGP, yang sangat penting untuk merutekan ulang lalu lintas selama gangguan jaringan.
- Anda mengabaikan batasan bandwidth terowongan VPN dan melebih-lebihkan kemampuan cadangannya.

Manfaat menjalankan praktik terbaik ini: Dengan mengimplementasikan konektivitas redundan antara lingkungan cloud dan lingkungan perusahaan atau on-premise, layanan dependen antara dua lingkungan tersebut dapat berkomunikasi dengan andal.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Saat menggunakan AWS Direct Connect untuk menghubungkan jaringan on-premise ke AWS, Anda dapat mencapai ketahanan jaringan maksimum (SLA 99,99%) dengan menggunakan koneksi terpisah yang berakhir pada perangkat berbeda di lebih dari satu lokasi on-premise dan lebih dari satu lokasi AWS Direct Connect. Topologi ini menawarkan ketahanan terhadap kegagalan perangkat, masalah konektivitas, dan pemadaman lokasi total. Alternatifnya, Anda dapat mencapai ketahanan tinggi (SLA 99,9%) dengan menggunakan dua koneksi individual ke beberapa lokasi (setiap lokasi on-premise terhubung ke satu lokasi Direct Connect). Pendekatan ini melindungi dari gangguan konektivitas yang disebabkan oleh pemotongan serat atau kegagalan perangkat dan membantu mengurangi kegagalan lokasi total. AWS Direct Connect Resiliency Toolkit dapat membantu merancang topologi AWS Direct Connect Anda.

Anda juga dapat mempertimbangkan AWS Site-to-Site VPN yang berakhir pada AWS Transit Gateway sebagai cadangan yang hemat biaya ke koneksi AWS Direct Connect utama Anda. Pengaturan ini memungkinkan perutean multijalur biaya setara (ECMP) di beberapa terowongan

VPN, yang memungkinkan throughput hingga 50Gbps, meskipun setiap terowongan VPN dibatasi pada 1,25 Gbps. Namun, penting untuk diingat bahwa AWS Direct Connect tetap merupakan pilihan paling efektif untuk meminimalkan gangguan jaringan dan menyediakan konektivitas yang stabil.

Saat menggunakan VPN melalui internet untuk menghubungkan lingkungan cloud Anda ke pusat data on-premise Anda, konfigurasikan dua terowongan VPN sebagai bagian dari koneksi VPN situs-ke-situs tunggal. Setiap terowongan harus berakhir di Zona Ketersediaan yang berbeda untuk mendapatkan ketersediaan tinggi dan menggunakan perangkat keras redundan untuk mencegah kegagalan perangkat on-premise. Selain itu, pertimbangkan beberapa koneksi internet dari berbagai penyedia layanan internet (ISP) di lokasi on-premise Anda untuk menghindari gangguan konektivitas VPN total karena satu pemadaman ISP. Memilih ISP dengan perutean dan infrastruktur yang beragam, terutama yang memiliki jalur fisik terpisah ke titik akhir AWS, memberikan ketersediaan konektivitas yang tinggi.

Selain redundansi fisik dengan beberapa koneksi AWS Direct Connect dan beberapa terowongan VPN (atau kombinasi keduanya), implementasi perutean dinamis Protokol Gateway Batas (BGP) juga penting. BGP dinamis menyediakan pengalihan rute lalu lintas secara otomatis dari satu jalur ke jalur lain berdasarkan kondisi jaringan waktu nyata dan kebijakan yang dikonfigurasi. Perilaku dinamis ini sangat bermanfaat dalam menjaga ketersediaan jaringan dan kontinuitas layanan jika terjadi kegagalan tautan atau jaringan. Jalur alternatif dipilih dengan cepat, sehingga meningkatkan ketahanan dan keandalan jaringan.

Langkah implementasi

- Akuisisi konektivitas dengan ketersediaan tinggi antara AWS dan lingkungan on-premise Anda.
  - Gunakan beberapa koneksi AWS Direct Connect atau terowongan VPN antara jaringan privat yang di-deploy secara terpisah.
  - Gunakan lebih dari satu lokasi AWS Direct Connect untuk ketersediaan tinggi.
  - Ketika menggunakan beberapa Wilayah AWS, ciptakan redundansi setidaknya di dalam dua di antaranya.
- Gunakan AWS Transit Gateway, jika memungkinkan, untuk mengakhiri koneksi VPN Anda.
- Evaluasi peralatan AWS Marketplace untuk mengakhiri VPN atau memperluas SD-WAN Anda ke AWS. Ketika menggunakan peralatan AWS Marketplace, lakukan deployment instans redundan untuk ketersediaan tinggi di Zona Ketersediaan yang berbeda.
- Sediakan koneksi redundan ke lingkungan on-premise Anda.
  - Anda mungkin memerlukan koneksi redundan ke beberapa Wilayah AWS agar ketersediaan yang Anda butuhkan tercapai.

- Gunakan Alat Ketahanan AWS Direct Connect untuk memulai.

Sumber daya

Dokumen terkait:

- Rekomendasi Ketahanan AWS Direct Connect
- Menggunakan Koneksi Site-to-Site VPN Redundan untuk Menyediakan Failover
- Kebijakan pengalihan rute dan komunitas BGP
- Konfigurasi Aktif/Aktif dan Aktif/Pasif di AWS Direct Connect
- Partner APN: partner yang dapat membantu merencanakan jaringan Anda
- AWS Marketplace untuk Infrastruktur Jaringan
- Laporan Resmi Opsi Konektivitas Amazon Virtual Private Cloud
- Membangun Infrastruktur Jaringan AWS Multi-VPC yang Aman dan Dapat Diskalakan
- Menggunakan koneksi Site-to-Site VPN redundan untuk menyediakan failover
- Menggunakan Alat Ketahanan AWS Direct Connect untuk memulai
- Titik Akhir VPC dan Layanan Titik Akhir VPC (AWS PrivateLink)
- Apa Itu Amazon VPC?
- Apa itu gateway transit?
- Apa itu AWS Site-to-Site VPN?
- Bekerja dengan gateway Direct Connect

Video terkait:

- AWS re:Invent 2018: Desain VPC Tingkat Lanjut dan Kemampuan Baru untuk Amazon VPC
- AWS re:Invent 2019: Arsitektur referensi AWS Transit Gateway untuk berbagai VPC

REL02-BP03 Pastikan alokasi subnet IP menjelaskan ekspansi dan ketersediaan

Rentang alamat IP Amazon VPC harus cukup besar untuk mengakomodasi persyaratan beban kerja, termasuk pertimbangan ekspansi mendatang dan alokasi alamat IP ke subnet di seluruh Zona Ketersediaan. Ini mencakup penyeimbang beban, instans EC2, dan aplikasi berbasis kontainer.

Ketika Anda merencanakan topologi jaringan Anda, langkah pertama adalah menetapkan ruang alamat IP itu sendiri. Rentang alamat IP privat (mengikuti pedoman RFC 1918) harus dialokasikan untuk setiap VPC. Akomodasikan persyaratan berikut sebagai bagian dari proses ini:

- Berikan ruang alamat IP untuk lebih dari satu VPC per Wilayah.
- Di dalam VPC, berikan ruang untuk beberapa subnet sehingga Anda dapat mencakup beberapa Zona Ketersediaan.
- Pertimbangkan untuk membiarkan ruang blok CIDR yang tidak digunakan di dalam VPC untuk ekspansi mendatang.
- Pastikan ada ruang alamat IP untuk memenuhi kebutuhan armada sementara instans Amazon EC2 yang mungkin Anda gunakan, seperti Armada Spot untuk machine learning, klaster Amazon EMR, atau klaster Amazon Redshift. Pertimbangan serupa sebaiknya diberikan ke klaster Kubernetes, seperti Amazon Elastic Kubernetes Service (Amazon EKS), karena setiap pod Kubernetes diberi alamat yang dapat dirutekan dari blok CIDR VPC secara default.
- Perhatikan, empat alamat IP pertama dan alamat IP terakhir di setiap blok CIDR subnet disimpan dan tidak tersedia untuk Anda gunakan.
- Perhatikan, blok CIDR VPC awal yang dialokasikan ke VPC Anda tidak dapat diubah atau dihapus, tetapi Anda dapat menambahkan tambahan blok CIDR yang tidak tumpang tindih ke VPC. CIDR IPv4 subnet tidak dapat diubah, tetapi CIDR IPv6 dapat diubah.
- Blok CIDR VPC terbesar yang memungkinkan adalah /16, dan yang terkecil adalah /28.
- Pertimbangkan jaringan terkoneksi lain (VPC, on-premise, atau penyedia cloud lainnya) dan pastikan ruang alamat IP tidak tumpang tindih. Untuk informasi selengkapnya, lihat REL02-BP05 Terapkan rentang alamat IP privat yang tidak tumpang tindih di semua ruang alamat privat tempat semuanya terhubung.

Hasil yang diinginkan: Subnet IP yang dapat diskalakan dapat membantu Anda mengakomodasi pertumbuhan di masa depan dan menghindari pemborosan yang tidak perlu.

Antipola umum:

- Gagal mempertimbangkan pertumbuhan di masa depan, sehingga menghasilkan blok CIDR yang terlalu kecil dan membutuhkan konfigurasi ulang, serta berpotensi menyebabkan waktu henti.
- Salah memperkirakan jumlah alamat IP yang dapat digunakan oleh penyeimbang beban elastis.
- Melakukan deployment banyak penyeimbang beban lalu lintas tinggi ke subnet yang sama.
- Menggunakan mekanisme penskalaan otomatis tetapi gagal memantau pemakaian alamat IP.

- Mendefinisikan rentang CIDR yang terlalu besar melampaui ekspektasi pertumbuhan di masa depan, sehingga menyebabkan kesulitan dalam melakukan peering dengan jaringan lain dengan rentang alamat yang tumpang tindih.

Manfaat menjalankan praktik terbaik ini: Ini memastikan bahwa Anda dapat mengakomodasi pertumbuhan beban kerja Anda dan terus memberikan ketersediaan saat Anda menaikkan skala.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Sedang

Panduan implementasi

Rencanakan jaringan Anda untuk mengakomodasi pertumbuhan, kepatuhan terhadap peraturan, dan integrasi dengan yang lain. Pertumbuhan dapat lebih besar dari yang diperkirakan, kepatuhan terhadap peraturan dapat berubah, dan koneksi jaringan privat atau akuisisi dapat sulit diimplementasikan tanpa perencanaan yang baik.

- Pilih Wilayah dan Akun AWS yang relevan berdasarkan persyaratan layanan, latensi, peraturan, dan pemulihan bencana (DR) Anda.
- Identifikasi kebutuhan Anda untuk deployment VPC regional.
- Identifikasi ukuran VPC.
  - Tentukan apakah Anda akan melakukan deploy konektivitas multi-VPC.
    - Apa Itu Gateway Transit?
    - Konektivitas Multi-VPC Satu Wilayah
  - Tentukan apakah Anda membutuhkan jaringan terpisah untuk persyaratan peraturan.
  - Buat VPC dengan blok CIDR berukuran sesuai untuk mengakomodasi kebutuhan Anda saat ini dan di masa depan.
    - Jika Anda memiliki proyeksi pertumbuhan yang tidak diketahui, Anda mungkin ingin melakukan kesalahan di sisi blok CIDR yang lebih besar untuk mengurangi potensi konfigurasi ulang di masa depan
  - Pertimbangkan untuk menggunakan alamat IPv6 untuk subnet sebagai bagian dari VPC tumpukan ganda. IPv6 sangat cocok untuk digunakan di subnet privat yang berisi armada instans sementara atau kontainer yang seharusnya membutuhkan alamat IPv4 dalam jumlah besar.

Sumber daya

Praktik terbaik Well-Architected terkait:

- REL02-BP05 Terapkan rentang alamat IP privat yang tidak tumpang tindih di semua ruang alamat privat tempat semuanya terhubung

Dokumen terkait:

- Partner APN: partner yang dapat membantu merencanakan jaringan Anda
- AWS Marketplace untuk Infrastruktur Jaringan
- Laporan Resmi Opsi Konektivitas Amazon Virtual Private Cloud
- Konektivitas jaringan ketersediaan tinggi (HA) beberapa pusat data
- Konektivitas Multi-VPC Satu Wilayah
- Apa Itu Amazon VPC?
- IPv6 di AWS
- IPv6 di arsitektur referensi
- Amazon Elastic Kubernetes Service meluncurkan dukungan IPv6

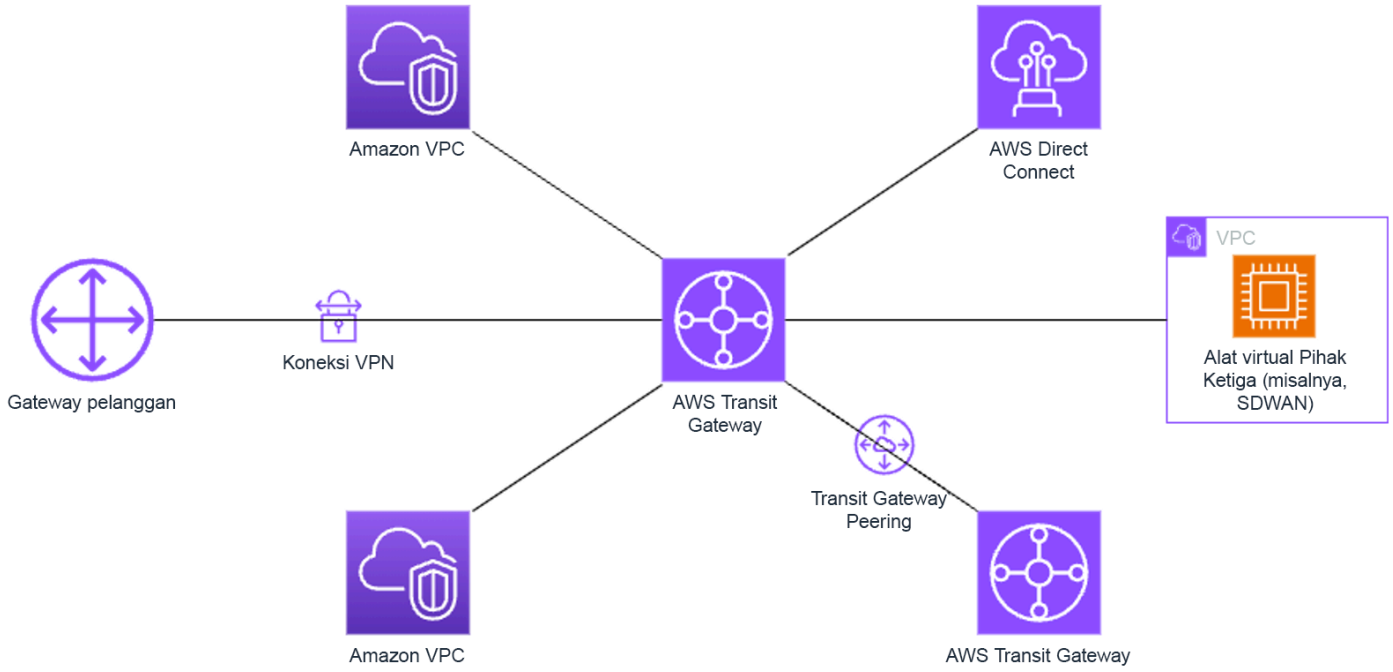
Video terkait:

- AWS re:Invent 2018: Desain VPC Tingkat Lanjut dan Kemampuan Baru untuk Amazon VPC (NET303)
- AWS re:Invent 2019: Arsitektur referensi AWS Transit Gateway untuk berbagai VPC (NET406-R1)
- AWS re:Invent 2023: AWS Siap dengan yang berikutnya? Merancang jaringan untuk pertumbuhan dan fleksibilitas (NET310)

REL02-BP04 Mengutamakan topologi hub-and-spoke daripada mesh many-to-many

Saat menghubungkan beberapa jaringan privat, seperti Cloud Privat Virtual (VPC) dan jaringan on-premise, pilihlah topologi hub and spoke daripada topologi mesh. Tidak seperti topologi mesh, di mana setiap jaringan terhubung langsung ke jaringan lain dan meningkatkan kompleksitas serta overhead manajemen, arsitektur hub and spoke memusatkan koneksi melalui satu hub. Sentralisasi ini menyederhanakan struktur jaringan dan meningkatkan operabilitas, skalabilitas, dan kontrolnya.

AWS Transit Gateway adalah layanan terkelola, dapat diskalakan, dan memiliki ketersediaan tinggi yang dirancang untuk pembangunan jaringan hub and spoke di AWS. Layanan ini berfungsi sebagai hub pusat jaringan Anda yang menyediakan segmentasi jaringan, perutean terpusat, dan koneksi yang disederhanakan ke lingkungan cloud dan on-premise. Gambar berikut menunjukkan cara menggunakan AWS Transit Gateway untuk membangun topologi hub and spoke Anda.



Antipola umum:

- Anda terlalu memperumit kebijakan perutean dalam arsitektur hub and spoke, sehingga mengurangi efisiensi jaringan dan mempersulit pemecahan masalah serta manajemen yang proaktif.
- Segmentasi berbasis perutean yang tidak memadai di dalam hub dapat menyebabkan kerentanan, yang berpotensi mengekspos jaringan ke akses yang tidak sah.
- Tanpa optimisasi yang cermat, lalu lintas yang dirutekan melalui hub dapat menyebabkan biaya transfer data yang lebih tinggi, terutama untuk lalu lintas yang melewati beberapa Zona Ketersediaan dan Wilayah. Strategi manajemen lalu lintas yang efektif sangat penting untuk mengontrol pengeluaran.

Manfaat menjalankan praktik terbaik ini: Seiring meningkatnya jumlah jaringan yang terhubung, manajemen dan perluasan konektivitas mesh menjadi makin menantang. AWS Transit Gateway

menawarkan hub terkelola yang dapat diskalakan dan andal untuk pembangunan dan pengoperasian topologi hub and spoke Anda. Saat Anda menggunakan AWS Transit Gateway, Anda dapat membangun koneksi dan memusatkan perutean lalu lintas di beberapa jaringan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

- Rencanakan jaringan Anda.
- Buat AWS Transit Gateway Anda.
- Lampirkan VPC Anda.
- Jika diperlukan, buat koneksi VPN atau gateway Direct Connect dan kaitkan dengan Transit Gateway.
- Tentukan bagaimana lalu lintas dirutekan antara VPC yang terhubung dan koneksi lain melalui konfigurasi tabel rute Transit Gateway Anda.
- Gunakan Amazon CloudWatch untuk memantau dan menyesuaikan konfigurasi yang diperlukan untuk optimisasi performa dan biaya.

Sumber daya

Dokumen terkait:

- What Is a Transit Gateway?
- Building a Scalable and Secure Multi-VPC AWS Network Infrastructure
- Building a global network using AWS Transit Gateway Inter-Region peering
- Amazon Virtual Private Cloud Connectivity Options
- Partner APN: partner yang dapat membantu merencanakan jaringan Anda
- AWS Marketplace untuk Infrastruktur Jaringan

Video terkait:

- AWS re:Invent 2023 - AWS networking foundations
- AWS re:Invent 2023 - Advanced VPC designs and new capabilities

REL02-BP05 Terapkan rentang alamat IP privat yang tidak tumpang tindih di semua ruang alamat privat tempat semuanya terhubung

Rentang alamat IP setiap VPC Anda tidak boleh tumpang tindih ketika peering, terhubung melalui Transit Gateway, atau terhubung melalui VPN. Hindari konflik alamat IP antara lingkungan on-premise dan VPC atau dengan penyedia cloud lain yang Anda gunakan. Selain itu, Anda harus memiliki cara untuk mengalokasikan rentang alamat IP privat ketika dibutuhkan. Sistem manajemen alamat IP (IPAM) dapat membantu mengotomatiskannya.

Hasil yang diinginkan:

- Tidak ada konflik rentang alamat IP antara VPC, lingkungan on-premise, atau penyedia cloud lain.
- Manajemen alamat IP yang tepat memungkinkan penskalaan infrastruktur jaringan yang lebih mudah untuk mengakomodasi pertumbuhan dan perubahan persyaratan jaringan.

Antipola umum:

- Menggunakan rentang IP yang sama di VPC Anda seperti yang Anda miliki on-premise, di jaringan perusahaan Anda, atau penyedia cloud lainnya.
- Tidak melacak rentang IP VPC yang digunakan untuk deployment beban kerja Anda.
- Mengandalkan proses manajemen alamat IP manual, seperti spreadsheet.
- Blok CIDR yang terlalu besar atau kecil, yang mengakibatkan pemborosan alamat IP atau ruang alamat tidak mencukupi untuk beban kerja Anda.

Manfaat menjalankan praktik terbaik ini: Perencanaan aktif jaringan Anda akan memastikan bahwa Anda tidak memiliki beberapa kejadian alamat IP yang sama di jaringan yang saling terhubung. Ini mencegah timbulnya masalah perutean di bagian beban kerja yang menggunakan aplikasi yang berbeda.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Manfaatkan IPAM, seperti Amazon VPC IP Address Manager, untuk memantau dan mengelola penggunaan CIDR Anda. Beberapa IPAM juga tersedia dari AWS Marketplace. Evaluasi potensi penggunaan Anda di AWS, tambahkan rentang CIDR ke VPC yang ada, dan buat VPC untuk memungkinkan pertumbuhan yang direncanakan dalam penggunaan.

Langkah implementasi

- Catat konsumsi CIDR saat ini (misalnya, VPC dan subnet).
  - Gunakan operasi API layanan untuk mengumpulkan konsumsi CIDR saat ini.
  - Gunakan Amazon VPC IP Address Manager untuk menemukan sumber daya.
- Catat penggunaan subnet Anda saat ini.
  - Gunakan operasi API layanan untuk mengumpulkan subnet per VPC di setiap Wilayah.
  - Gunakan Amazon VPC IP Address Manager untuk menemukan sumber daya.
- Catat penggunaan saat ini.
- Tentukan apakah Anda telah membuat rentang IP yang tumpang tindih.
- Hitung kapasitas cadangan.
- Identifikasi rentang IP yang tumpang tindih. Anda dapat bermigrasi ke rentang alamat baru atau mempertimbangkan penggunaan teknik seperti NAT Gateway privat atau AWS PrivateLink jika Anda perlu menghubungkan rentang yang tumpang tindih.

Sumber daya

Praktik Terbaik Terkait:

- Melindungi jaringan

Dokumen terkait:

- Partner APN: partner yang dapat membantu merencanakan jaringan Anda
- AWS Marketplace untuk Infrastruktur Jaringan
- Laporan Resmi Opsi Konektivitas Amazon Virtual Private Cloud
- Konektivitas jaringan ketersediaan tinggi (HA) beberapa pusat data
- Menghubungkan Jaringan dengan Rentang IP yang Tumpang Tindih
- Apa Itu Amazon VPC?
- Apa itu IPAM?

Video terkait:

- AWS re:Invent 2023 - Desain VPC tingkat lanjut dan kemampuan baru

- AWS re:Invent 2019: Arsitektur referensi AWS Transit Gateway untuk berbagai VPC
- AWS re:Invent 2023 - Siap dengan yang berikutnya? Merancang jaringan untuk pertumbuhan dan fleksibilitas
- AWS re:Invent 2021 - {Peluncuran Baru} Kelola alamat IP Anda dalam skala besar di AWS

Arsitektur beban kerja

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

Bagian-bagian berikutnya menjelaskan praktik terbaik untuk digunakan dengan pola-pola keandalan ini.

Topik

Mendesain arsitektur layanan beban kerja Anda

Bangun beban kerja yang mudah diskalakan dan andal menggunakan arsitektur berorientasi layanan (SOA) atau arsitektur layanan mikro. Arsitektur berorientasi layanan (SOA) merupakan praktik pembuatan komponen perangkat lunak yang dapat digunakan ulang lewat antarmuka layanan. Arsitektur layanan mikro melakukan hal yang lebih dengan membuat komponen menjadi lebih kecil dan lebih sederhana.

Antarmuka arsitektur berorientasi layanan (SOA) menggunakan standar komunikasi umum sehingga dapat dimasukkan dengan cepat ke dalam beban kerja baru. SOA menggantikan praktik pembangunan arsitektur monolit, yang terdiri dari unit-unit tak dapat dibagi dan saling bergantung satu sama lain.

Di AWS, kami selalu menggunakan SOA, tetapi kini kami sudah mulai membangun sistem-sistem kami menggunakan layanan mikro. Meskipun layanan mikro memiliki sejumlah kualitas yang menarik, manfaat paling penting untuk ketersediaan adalah ukurannya yang lebih kecil dan sifatnya yang lebih sederhana. Layanan mikro memungkinkan Anda untuk membedakan ketersediaan yang diperlukan dari layanan yang berbeda, dan oleh karena itu fokuskan investasi terutama ke layanan mikro yang memiliki kebutuhan ketersediaan paling besar. Sebagai contoh, untuk menghadirkan halaman informasi produk di Amazon.com (“halaman detail”), ratusan layanan mikro dipanggil untuk membangun bagian-bagian halaman yang terpisah. Meskipun terdapat beberapa layanan yang harus tersedia untuk menyediakan harga dan detail produk, sebagian besar konten di halaman tersebut

dapat dikecualikan jika layanan tidak tersedia. Bahkan hal-hal seperti foto dan ulasan tidak diperlukan untuk menyediakan pengalaman bagi pelanggan dalam membeli sebuah produk.

Praktik terbaik

- REL03-BP01 Memilih cara untuk menyegmentasi beban kerja
- REL03-BP02 Bangun layanan yang berfokus pada domain dan fungsionalitas bisnis khusus
- REL03-BP03 Memberikan kontrak layanan per API

REL03-BP01 Memilih cara untuk menyegmentasi beban kerja

Segmentasi beban kerja penting saat menentukan persyaratan ketahanan aplikasi Anda. Arsitektur monolitik harus dihindari jika memungkinkan. Sebagai gantinya, pertimbangkan dengan cermat komponen aplikasi mana yang dapat dipecah menjadi layanan mikro. Bergantung pada persyaratan aplikasi Anda, solusinya mungkin merupakan kombinasi arsitektur berorientasi layanan (SOA) dengan layanan mikro jika memungkinkan. Beban kerja yang mampu berada dalam kondisi stateless akan lebih mampu di-deploy sebagai layanan mikro.

Hasil yang diinginkan: Beban kerja harus dapat didukung, dapat diskalakan, dan di-coupling selonggar mungkin.

Saat membuat pilihan tentang cara menyegmentasikan beban kerja Anda, seimbangkan manfaat dengan kerumitannya. Hal yang tepat untuk produk baru yang mengejar jadwal peluncuran pertama akan berbeda dengan hal yang dibutuhkan oleh beban kerja yang dibangun untuk diskalakan dari awal. Saat memfaktor ulang monolit yang ada, Anda perlu mempertimbangkan seberapa baik aplikasi akan mendukung dekomposisi menuju kondisi stateless. Dengan memecah layanan menjadi bagian-bagian yang lebih kecil, tim kecil yang diberi tanggung jawab khusus akan dapat mengembangkan dan mengelolanya. Namun, layanan yang lebih kecil dapat menimbulkan kompleksitas yang mencakup kemungkinan peningkatan latensi, debugging yang lebih kompleks, dan peningkatan beban operasional.

Antipola umum:

- Dalam layanan mikro, Death Star adalah situasi saat komponen atomik menjadi sangat saling bergantung sehingga kegagalan salah satu komponen akan menghasilkan kegagalan yang jauh lebih besar, sehingga komponen ini pun menjadi kaku dan rapuh seperti monolit.

Manfaat menjalankan praktik ini:

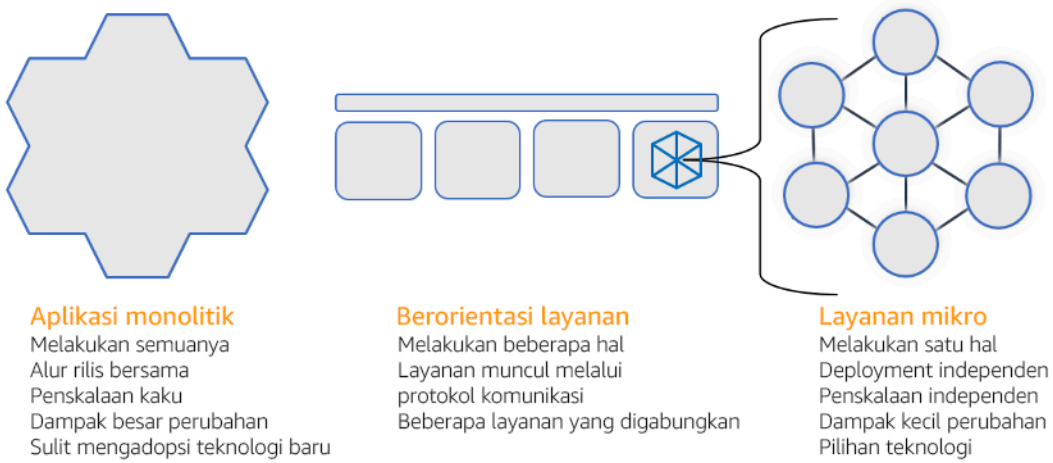
- Segmen yang lebih spesifik akan menghasilkan ketangkasan, fleksibilitas organisasi, dan skalabilitas yang lebih besar.
- Dampak gangguan layanan yang berkurang.
- Komponen-komponen aplikasi mungkin memiliki persyaratan ketersediaan yang berbeda-beda, yang dapat didukung oleh segmentasi yang lebih atomik.
- Tanggung jawab yang ditentukan khusus untuk tim yang mendukung beban kerja.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Pilih jenis arsitektur berdasarkan cara beban kerja disegmentasikan. Pilih arsitektur SOA atau layanan mikro (atau dalam beberapa kasus yang jarang terjadi, arsitektur monolitik). Bahkan jika Anda memilih untuk memulai arsitektur monolit, Anda harus memastikan bahwa arsitektur tersebut modular dan pada akhirnya dapat berkembang menjadi SOA atau layanan mikro seiring dengan skala produk Anda dengan adopsi pengguna. SOA dan layanan mikro masing-masing menawarkan segmentasi yang lebih kecil, yang lebih disarankan sebagai arsitektur modern yang dapat diskalakan dan andal, tetapi ada tarik ulur yang perlu dipertimbangkan, terutama saat melakukan deployment arsitektur layanan mikro.

Salah satu tarik ulur utama adalah Anda sekarang memiliki arsitektur komputasi terdistribusi yang dapat mempersulit dalam memenuhi persyaratan latensi pengguna dan ada kerumitan tambahan dalam proses debugging dan penelusuran interaksi pengguna. Anda dapat menggunakan AWS X-Ray untuk membantu Anda memecahkan masalah ini. Efek lain yang perlu dipertimbangkan adalah peningkatan kompleksitas operasional seiring Anda meningkatkan jumlah aplikasi yang Anda kelola, yang memerlukan deployment banyak komponen independen.



Arsitektur monolitik, berorientasi layanan, dan layanan mikro

Langkah implementasi

- Tentukan arsitektur yang sesuai untuk memfaktor ulang atau membangun aplikasi Anda. SOA dan layanan mikro masing-masing menawarkan segmentasi yang lebih kecil, serta diutamakan sebagai arsitektur modern yang dapat diskalakan dan diandalkan. SOA dapat menjadi kompensasi yang baik untuk mencapai segmentasi yang lebih kecil sembari menghindari beberapa kompleksitas dari layanan mikro. Untuk detail selengkapnya, lihat Tarik Ulur Layanan Mikro.
- Jika dapat diterima beban kerja dan didukung organisasi, Anda harus menggunakan arsitektur layanan mikro untuk mencapai ketangkasan dan keandalan terbaik. Untuk detail selengkapnya, lihat Menerapkan Layanan Mikro di AWS.
- Pertimbangkan untuk mengikuti karakteristik pohon Strangler Fig , yaitu pola untuk memfaktor ulang monolit menjadi komponen yang lebih kecil. Hal ini memerlukan penggantian komponen aplikasi tertentu secara bertahap dengan aplikasi dan layanan baru. AWS Migration Hub Refactor Spaces bertindak sebagai titik awal untuk pemfaktoran ulang secara bertahap. Untuk detail selengkapnya, lihat Memigrasikan beban kerja lama di on-premise dengan lancar menggunakan pola strangler.
- Implementasi layanan mikro mungkin memerlukan mekanisme penemuan layanan untuk memungkinkan layanan terdistribusi ini berkomunikasi satu sama lain. AWS App Mesh dapat digunakan dengan arsitektur berorientasi layanan untuk menyediakan penemuan dan akses layanan yang andal. AWS Cloud Map juga dapat digunakan untuk penemuan layanan berbasis DNS yang dinamis.

- Jika Anda bermigrasi dari monolit ke SOA, Amazon MQ dapat membantu menjembatani kesenjangan sebagai bus layanan saat mendesain ulang aplikasi lama di cloud.
- Untuk monolit yang ada dengan satu basis data bersama, pilih cara mengatur ulang data menjadi segmen yang lebih kecil. Segmentasi ini dapat didasarkan pada unit bisnis, pola akses, atau struktur data. Pada titik ini dalam proses pemfaktoran ulang, Anda harus memilih untuk melanjutkan dengan jenis basis data relasional atau nonrelasional (NoSQL). Untuk detail selengkapnya, lihat Dari SQL ke NoSQL.

Tingkat upaya untuk rencana implementasi: Tinggi

Sumber daya

Praktik terbaik terkait:

- REL03-BP02 Bangun layanan yang berfokus pada domain dan fungsionalitas bisnis khusus

Dokumen terkait:

- Amazon API Gateway: Mengonfigurasi API REST Menggunakan OpenAPI
- Apa itu Arsitektur Berorientasi Layanan?
- Konteks Terikat (pola sentral di Desain yang Didorong Domain)
- Mengimplementasikan Layanan Mikro di AWS
- Kompensasi Layanan Mikro
- Layanan mikro - definisi istilah arsitektur baru ini
- Layanan mikro di AWS
- Apa itu AWS App Mesh?

Contoh terkait:

- Lokakarya Modernisasi Aplikasi Iteratif

Video terkait:

- Memberikan Keunggulan dengan Layanan Mikro di AWS

REL03-BP02 Bangun layanan yang berfokus pada domain dan fungsionalitas bisnis khusus

Arsitektur berorientasi layanan (SOA) menetapkan layanan dengan fungsi yang digambarkan dengan baik berdasarkan kebutuhan bisnis. Layanan mikro menggunakan model domain dan konteks yang dibatasi untuk menarik batas-batas layanan di sepanjang batas konteks bisnis. Berfokus pada domain dan fungsionalitas bisnis dapat membantu tim untuk menentukan persyaratan keandalan sendiri untuk layanan mereka. Konteks yang dibatasi mengisolasi dan memisahkan logika bisnis, sehingga memungkinkan tim memiliki penalaran yang lebih baik tentang bagaimana menangani kegagalan.

Hasil yang diinginkan: Rekayasawan dan pemangku kepentingan bisnis bersama-sama menetapkan konteks yang dibatasi dan menggunakannya untuk merancang sistem sebagai layanan yang memenuhi fungsi bisnis tertentu. Tim-tim ini menggunakan praktik yang telah lazim seperti event storming untuk menentukan persyaratan. Aplikasi baru dirancang sebagai batas layanan yang ditetapkan dengan baik dan penggabungan longgar. Monolit yang ada diurai menjadi konteks-konteks yang dibatasi dan desain sistem beralih ke arsitektur SOA atau layanan mikro. Ketika monolit difaktorkan ulang, pendekatan lazim seperti konteks gelembung dan pola penguraian monolit diterapkan.

Layanan berorientasi domain dijalankan sebagai satu atau beberapa proses yang statusnya tidak sama. Layanan-layanan tersebut secara independen merespons fluktuasi permintaan dan menangani skenario kesalahan dengan berpatokan pada persyaratan khusus domain.

Antipola umum:

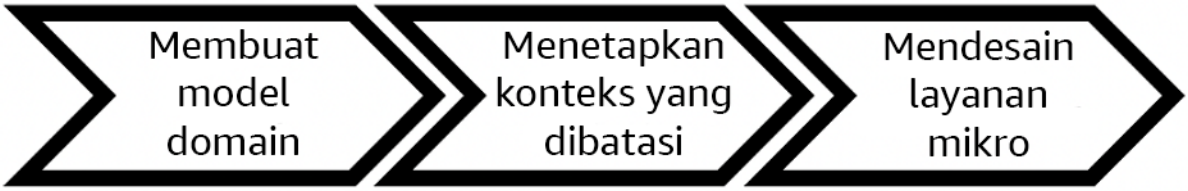
- Tim dibentuk berdasarkan domain-domain teknis tertentu seperti UI dan UX, perangkat lunak perantara (middleware), atau basis data, bukan berdasarkan domain bisnis tertentu.
- Aplikasi melibatkan tanggung jawab domain. Layanan yang mencakup konteks yang dibatasi bisa lebih sulit untuk dipelihara, memerlukan upaya pengujian yang lebih besar, dan memerlukan banyak tim domain untuk berpartisipasi dalam pembaruan perangkat lunak.
- Dependensi domain, seperti pustaka entitas domain, dibagikan di seluruh layanan sehingga perubahan untuk satu domain layanan memerlukan perubahan pada domain layanan lainnya
- Kontrak layanan dan logika bisnis tidak mengekspresikan entitas dalam bahasa domain yang umum dan konsisten, sehingga menghasilkan lapisan terjemahan yang merumitkan sistem dan meningkatkan upaya debugging.

Manfaat menjalankan praktik terbaik ini: Aplikasi dirancang sebagai layanan independen yang dibatasi oleh domain bisnis dan menggunakan bahasa bisnis umum. Layanan dapat diuji dan dapat di-deploy secara independen. Layanan memenuhi persyaratan ketahanan khusus domain untuk domain yang diterapkan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Keputusan berbasis domain (DDD) adalah pendekatan dasar perancangan dan pembangunan perangkat lunak berdasarkan domain bisnis. Bekerja dengan kerangka kerja yang ada memudahkan pembuatan layanan yang berfokus pada domain bisnis. Saat bekerja dengan aplikasi monolitik yang ada, Anda dapat memanfaatkan pola penguraian yang menyediakan teknik-teknik yang sudah lazim untuk memodernisasi aplikasi menjadi layanan.



Keputusan berbasis domain

Langkah implementasi

- Tim dapat menyelenggarakan lokakarya event storming untuk mengidentifikasi peristiwa, perintah, agregat, dan domain secara cepat dalam format catatan tempel ringan.
- Setelah entitas dan fungsi domain dibentuk dalam konteks domain, Anda dapat membagi domain Anda ke dalam layanan-layanan menggunakan konteks yang dibatasi, dengan mengelompokkan entitas dengan fitur dan atribut yang serupa. Dengan model yang dibagi ke dalam konteks, muncul templat untuk membatasi layanan mikro.
  - Misalnya, entitas situs web Amazon.com dapat meliputi paket, pengantaran, jadwal, harga, diskon, dan mata uang.
  - Paket, pengantaran, dan jadwal dikelompokkan ke dalam konteks pengiriman, sedangkan harga, diskon, dan mata uang dikelompokkan ke dalam konteks harga.
- Mengurai monolit menjadi layanan mikro menjelaskan pola-pola untuk pemfaktoran ulang layanan mikro. Menggunakan pola-pola penguraian berdasarkan kemampuan bisnis, subdomain, atau transaksi selaras dengan pendekatan berbasis domain.

- Teknik-teknik taktis seperti konteks gelembung memungkinkan Anda memasukkan DDD di dalam aplikasi yang ada atau aplikasi warisan tanpa penulisan ulang di awal dan komitmen penuh terhadap DDD. Dalam pendekatan konteks gelembung, konteks terbatas yang kecil dibuat menggunakan pemetaan layanan dan koordinasi, atau lapisan antikorupsi, yang melindungi model domain yang baru ditentukan dari pengaruh eksternal.

Setelah tim melakukan analisis domain dan menentukan entitas serta kontrak layanan, mereka dapat memanfaatkan layanan AWS untuk menerapkan desain berbasis domain mereka sebagai layanan berbasis cloud.

- Mulai pengembangan Anda dengan menentukan pengujian yang menggunakan aturan bisnis domain Anda. Pengembangan berbasis pengujian (TDD) dan pengembangan berbasis perilaku (BDD) membantu tim menjaga layanan tetap fokus pada pemecahan masalah bisnis.
- Pilih layanan AWS yang paling memenuhi persyaratan domain bisnis dan arsitektur layanan mikro Anda:
  - AWS Nirserver memungkinkan tim Anda untuk fokus pada logika domain tertentu, bukan pada pengelolaan server dan infrastruktur.
  - Kontainer di AWS menyederhanakan pengelolaan infrastruktur Anda, sehingga Anda dapat fokus pada persyaratan domain Anda.
  - Basis data yang dirancang khusus membantu Anda mencocokkan persyaratan domain Anda dengan jenis basis data yang paling sesuai.
- Membangun arsitektur heksagonal di AWS menguraikan kerangka kerja untuk membangun logika bisnis menjadi layanan yang bekerja mundur dari domain bisnis untuk memenuhi persyaratan fungsional dan kemudian melampirkan adaptor integrasi. Pola-pola yang memisahkan detail antarmuka dari logika bisnis dengan layanan AWS membantu tim untuk berfokus pada fungsionalitas domain dan meningkatkan kualitas perangkat lunak.

Sumber daya

Praktik terbaik terkait:

- REL03-BP01 Memilih cara untuk menyegmentasi beban kerja
- REL03-BP03 Memberikan kontrak layanan per API

Dokumen terkait:

- Layanan Mikro AWS
- Mengimplementasikan Layanan Mikro di AWS
- Cara memecah Monolit menjadi Layanan-Layanan Mikro
- Mulai Menggunakan DDD di Tengah-Tengah Sistem Warisan
- Desain Berbasis Domain: Mengatasi Kompleksitas di Dalam Inti Perangkat Lunak
- Membangun arsitektur heksagonal di AWS
- Mengurai monolit menjadi layanan mikro
- Event Storming
- Pesan Antara Konteks-Konteks yang Dibatasi
- Layanan mikro
- Pengembangan berbasis pengujian
- Pengembangan berbasis perilaku

Contoh terkait:

- Lokakarya Cloud-Native Korporat
- Merancang Layanan Mikro Cloud-Native di AWS (dari DDD/EventStormingWorkshop)

Alat terkait:

- Basis Data AWS Cloud
- Nirserver di AWS
- Kontainer di AWS

REL03-BP03 Memberikan kontrak layanan per API

Kontrak layanan adalah perjanjian terdokumentasi antara produsen dan konsumen API yang ditetapkan dalam definisi API yang dapat dibaca mesin. Strategi versioning kontrak memungkinkan konsumen untuk terus menggunakan API yang ada dan memigrasikan aplikasi mereka ke API yang lebih baru ketika mereka siap. Deployment produsen dapat terjadi kapan saja, selama kontrak dipatuhi. Tim layanan dapat menggunakan tumpukan teknologi pilihan mereka untuk memenuhi kontrak API.

Hasil yang diinginkan:

Antipola umum: Aplikasi yang dibangun dengan arsitektur berorientasi layanan atau layanan mikro dapat beroperasi secara independen sementara tetap memiliki dependensi runtime yang terintegrasi. Perubahan yang di-deploy ke konsumen atau produsen API tidak mengganggu stabilitas sistem secara keseluruhan ketika kedua belah pihak mematuhi kontrak API yang sama. Komponen yang berkomunikasi melalui API layanan dapat melakukan rilis fungsional independen, peningkatan ke dependensi runtime, atau melakukan failover ke situs pemulihan bencana (DR) dengan sedikit atau tanpa dampak terhadap satu sama lain. Selain itu, layanan diskret dapat menyesuaikan skala secara independen dengan menyerap permintaan sumber daya tanpa mengharuskan layanan lain untuk menyesuaikan skala secara serempak.

- Membuat API layanan tanpa skema strongly-typed. Hal ini menghasilkan API yang tidak dapat digunakan untuk menghasilkan pengikatan dan muatan API yang tidak dapat divalidasi secara terprogram.
- Tidak mengadopsi strategi versioning, yang memaksa konsumen API untuk memperbarui dan melepaskan atau gagal saat kontrak layanan berkembang.
- Pesan kesalahan yang membocorkan detail implementasi layanan yang mendasari, bukan menggambarkan kegagalan integrasi dalam bahasa dan konteks domain.
- Tidak menggunakan kontrak API untuk mengembangkan kasus pengujian dan implementasi API simulasi untuk memungkinkan pengujian komponen layanan secara independen.

Manfaat menjalankan praktik terbaik ini: Sistem terdistribusi yang terdiri dari komponen-komponen yang berkomunikasi melalui kontrak layanan API dapat meningkatkan keandalan. Developer dapat mengidentifikasi potensi masalah di awal proses pengembangan dengan pemeriksaan tipe selama kompilasi untuk memverifikasi bahwa bidang-bidang yang diperlukan ada dan permintaan serta respons mematuhi kontrak API. Kontrak API menyediakan antarmuka dokumentasi mandiri yang jelas untuk API dan menyediakan interoperabilitas yang lebih baik antara sistem dan bahasa pemrograman yang berbeda.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Setelah mengidentifikasi domain bisnis dan menentukan segmentasi beban kerja, Anda dapat mengembangkan API layanan. Pertama-tama, tentukan kontrak layanan yang dapat dibaca mesin untuk API, lalu implementasikan strategi versioning API. Setelah siap mengintegrasikan layanan

melalui protokol umum seperti REST, GraphQL, atau peristiwa asinkron, Anda dapat memasukkan layanan AWS ke dalam arsitektur Anda untuk mengintegrasikan komponen Anda dengan kontrak API strongly-typed.

Layanan AWS untuk kontrak API layanan

Sertakan layanan AWS seperti Amazon API Gateway, AWS AppSync, dan Amazon EventBridge ke dalam arsitektur Anda untuk menggunakan kontrak layanan API dalam aplikasi Anda. Amazon API Gateway membantu Anda terintegrasi dengan layanan AWS native langsung serta layanan web lainnya. API Gateway mendukung spesifikasi dan versioning OpenAPI. AWS AppSync adalah titik akhir GraphQL terkelola yang Anda konfigurasikan dengan menetapkan skema GraphQL untuk menetapkan antarmuka layanan untuk kueri, mutasi, dan langganan. Amazon EventBridge menggunakan skema peristiwa untuk menetapkan peristiwa dan menghasilkan kode binding untuk peristiwa Anda.

Langkah implementasi

- Pertama-tama, tetapkan kontrak untuk API Anda. Kontrak akan mengekspresikan kemampuan suatu API serta menetapkan objek dan bidang data strongly-typed untuk input dan output API.
- Saat mengonfigurasi API di API Gateway, Anda dapat mengimpor dan mengekspor OpenAPI Specification untuk titik akhir Anda.
  - Mengimpor definisi OpenAPI menyederhanakan pembuatan API Anda dan dapat diintegrasikan dengan infrastruktur AWS sebagai alat kode seperti AWS Serverless Application Model dan AWS Cloud Development Kit (AWS CDK).
  - Mengekspor definisi API menyederhanakan integrasi dengan alat pengujian API dan menyediakan spesifikasi integrasi untuk konsumen layanan.
- Anda dapat menetapkan dan mengelola API GraphQL dengan AWS AppSync dengan cara menetapkan file skema GraphQL untuk menghasilkan antarmuka kontrak Anda dan menyederhanakan interaksi dengan model REST kompleks, beberapa tabel basis data, atau layanan warisan.
- Proyek AWS Amplify yang terintegrasi dengan AWS AppSync menghasilkan file kueri JavaScript strongly-typed untuk digunakan dalam aplikasi Anda serta pustaka klien GraphQL AWS AppSync untuk tabel Amazon DynamoDB .
- Saat Anda menggunakan peristiwa layanan dari Amazon EventBridge, peristiwa mematuhi skema yang sudah ada di dalam registri skema atau yang Anda definisikan dengan OpenAPI Spec. Dengan skema yang didefinisikan dalam registri, Anda juga dapat menghasilkan binding klien dari kontrak skema untuk mengintegrasikan kode Anda dengan peristiwa.

- Memperluas atau melakukan versioning API Anda. Memperluas API adalah opsi yang lebih sederhana saat menambahkan bidang yang dapat dikonfigurasi dengan bidang opsional atau nilai default untuk bidang wajib.
  - Kontrak berbasis JSON untuk protokol seperti REST dan GraphQL bisa ideal untuk perluasan kontrak.
  - Kontrak berbasis XML untuk protokol seperti SOAP harus diuji dengan konsumen layanan untuk menentukan kelayakan perluasan kontrak.
- Saat melakukan versioning API, pertimbangkan implementasi versioning proksi yang menggunakan facade untuk mendukung versi sehingga logika dapat dipertahankan dalam satu basis kode.
  - Dengan API Gateway Anda dapat menggunakan pemetaan permintaan dan respons untuk menyederhanakan penyerapan perubahan kontrak dengan membuat facade untuk memberikan nilai default untuk bidang baru atau untuk membuang bidang yang dihapus dari permintaan atau respons. Dengan pendekatan ini, layanan yang mendasari dapat mempertahankan basis kode tunggal.

Sumber daya

Praktik terbaik terkait:

- REL03-BP01 Memilih cara untuk menyegmentasi beban kerja
- REL03-BP02 Bangun layanan yang berfokus pada domain dan fungsionalitas bisnis khusus
- REL04-BP02 Mengimplementasikan dependensi yang digabungkan secara longgar
- REL05-BP03 Mengontrol dan membatasi panggilan percobaan ulang
- REL05-BP05 Mengatur batas waktu klien

Dokumen terkait:

- Apa itu API (Antarmuka Pemrograman Aplikasi)?
- Mengimplementasikan Layanan Mikro di AWS
- Kompensasi Layanan Mikro
- Layanan mikro - definisi istilah arsitektur baru ini
- Layanan mikro di AWS
- Bekerja dengan ekstensi API Gateway ke OpenAPI

- OpenAPI-Specification
- GraphQL: Skema dan Jenis
- Binding kode Amazon EventBridge

Contoh terkait:

- Amazon API Gateway: Mengonfigurasi API REST Menggunakan OpenAPI
- Aplikasi CRUD Amazon API Gateway ke Amazon DynamoDB menggunakan OpenAPI
- Pola integrasi aplikasi modern di era nirserver: Integrasi Layanan API Gateway
- Mengimplementasikan versioning API Gateway berbasis header dengan Amazon CloudFront
- AWS AppSync: Membangun aplikasi klien

Video terkait:

- Menggunakan OpenAPI di AWS SAM untuk mengelola API Gateway

Alat terkait:

- Amazon API Gateway
- AWS AppSync
- Amazon EventBridge

Merancang interaksi di dalam sistem terdistribusi untuk mencegah kegagalan

Sistem terdistribusi mengandalkan jaringan komunikasi untuk membuat interkoneksi komponen, seperti server atau layanan. Beban kerja Anda harus beroperasi secara andal walaupun terdapat latensi atau kehilangan data di jaringannya. Komponen sistem terdistribusi harus beroperasi tanpa memberikan dampak secara negatif kepada komponen dan beban kerja yang lain. Berbagai praktik terbaik ini mencegah kegagalan dan meningkatkan waktu rata-rata antara kegagalan (MTBF).

Praktik terbaik

- REL04-BP01 Mengidentifikasi jenis sistem terdistribusi yang Anda perlukan
- REL04-BP02 Mengimplementasikan dependensi yang digabungkan secara longgar

- REL04-BP03 Melakukan tugas konstan
- REL04-BP04 Menjadikan semua respons idempoten

REL04-BP01 Mengidentifikasi jenis sistem terdistribusi yang Anda perlukan

Sistem terdistribusi bisa bersifat sinkron, asinkron, atau batch. Sistem sinkron harus memproses permintaan secepat mungkin dan berkomunikasi satu sama lain dengan membuat panggilan permintaan dan respons sinkron menggunakan protokol HTTP/S, REST, atau panggilan prosedur jarak jauh (RPC). Sistem asinkron berkomunikasi satu sama lain dengan bertukar data secara asinkron melalui layanan perantara tanpa memasangkan sistem-sistem terpisah. Sistem batch menerima data input dalam jumlah besar, menjalankan proses data otomatis tanpa campur tangan manusia, dan menghasilkan data output.

Hasil yang diinginkan: Rancang beban kerja yang secara efektif berinteraksi dengan dependensi sinkron, asinkron, dan batch.

Antipola umum:

- Beban kerja menunggu respons dari dependensinya tanpa batas waktu, yang dapat menyebabkan klien beban kerja mengalami waktu habis, tanpa mengetahui apakah permintaannya telah diterima.
- Beban kerja menggunakan serangkaian sistem dependen yang memanggil satu sama lain secara sinkron. Hal ini mengharuskan setiap sistem untuk tersedia dan berhasil memproses sebuah permintaan sebelum seluruh rangkaian dapat berhasil, sehingga menyebabkan perilaku dan ketersediaan secara keseluruhan menjadi rapuh.
- Beban kerja berkomunikasi dengan dependensinya secara asinkron dan mengandalkan konsep pengiriman pesan yang dijamin persis satu kali, padahal sering kali masih pesan duplikat masih memungkinkan untuk diterima.
- Beban kerja tidak menggunakan alat penjadwalan batch yang sesuai dan memungkinkan pelaksanaan pekerjaan batch yang sama secara bersamaan.

Manfaat menjalankan praktik terbaik ini: Hal biasa bagi beban kerja tertentu untuk mengimplementasikan satu atau lebih gaya komunikasi antara sinkron, asinkron, dan batch. Praktik terbaik ini membantu Anda mengidentifikasi kompromi berbeda yang terkait dengan setiap gaya komunikasi untuk membuat beban kerja Anda dapat menoleransi gangguan pada dependensinya.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Bagian berikutnya berisi panduan implementasi umum dan spesifik untuk setiap jenis dependensi.

Panduan umum

- Pastikan tujuan tingkat layanan (SLO) kinerja dan keandalan yang ditawarkan oleh dependensi Anda memenuhi persyaratan performa dan keandalan beban kerja Anda.
- Gunakan layanan observabilitas AWS untuk memantau waktu respons dan tingkat kesalahan untuk memastikan dependensi Anda menyediakan layanan pada tingkat yang dibutuhkan oleh beban kerja Anda.
- Identifikasi potensi tantangan yang mungkin dihadapi beban kerja Anda saat berkomunikasi dengan dependensinya. Sistem terdistribusi memiliki berbagai tantangan yang dapat menambah kompleksitas arsitektur, beban operasional, dan biaya. Tantangan umumnya mencakup latensi, gangguan jaringan, kehilangan data, penskalaan, dan jeda replikasi data.
- Implementasikan penanganan kesalahan dan pembuatan log yang kuat untuk membantu Anda memecahkan masalah saat dependensi Anda mengalami masalah.

Dependensi sinkron

Dalam komunikasi sinkron, beban kerja Anda mengirimkan permintaan ke dependensinya dan memblokir operasi yang menunggu respons. Ketika dependensinya menerima permintaan, dependensi tersebut mencoba menanganinya sesegera mungkin dan mengembalikan respons ke beban kerja Anda. Tantangan besar pada komunikasi sinkron adalah jenis komunikasi ini menyebabkan penggabungan temporal, yang mengharuskan beban kerja Anda dan dependensinya untuk tersedia pada saat yang bersamaan. Ketika beban kerja Anda perlu berkomunikasi secara sinkron dengan dependensinya, pertimbangkan panduan berikut:

- Beban kerja Anda seharusnya tidak bergantung pada beberapa dependensi sinkron untuk melakukan satu fungsi. Rangkaian dependensi ini meningkatkan kerapuhan secara keseluruhan karena semua dependensi di jalurnya harus tersedia agar permintaan berhasil diselesaikan.
- Ketika dependensi tidak sehat atau tidak tersedia, tentukan penanganan kesalahan dan strategi coba lagi. Hindari penggunaan perilaku bimodal. Perilaku bimodal adalah ketika beban kerja Anda menunjukkan perilaku yang berbeda dalam mode normal dan mode kegagalan. Untuk detail lebih lanjut tentang perilaku bimodal, lihat REL11-BP05 Menggunakan stabilitas statis untuk mencegah perilaku bimodal.

- Harap diingat bahwa gagal cepat (fail fast) lebih baik daripada membuat beban kerja Anda menunggu. Misalnya, Panduan Developer AWS Lambda menjelaskan cara menangani percobaan ulang dan kegagalan saat Anda menginvokasi fungsi Lambda.
- Tetapkan batas waktu saat beban kerja Anda memanggil dependensinya. Teknik ini menghindari waktu tunggu yang terlalu lama atau waktu tunggu respons tanpa batas. Untuk diskusi yang bermanfaat tentang masalah ini, lihat Menyetel pengaturan permintaan HTTP SDK Java AWS untuk aplikasi Amazon DynamoDB sadar latensi.
- Minimalkan jumlah panggilan yang dilakukan dari beban kerja Anda ke dependensinya untuk memenuhi satu permintaan. Memiliki banyak panggilan (chatty) di antara keduanya dapat meningkatkan penggabungan dan latensi.

Dependensi asinkron

Untuk memisahkan beban kerja Anda dari dependensinya secara sementara, keduanya harus berkomunikasi secara asinkron. Jika menggunakan pendekatan asinkron, beban kerja Anda dapat melanjutkan pemrosesan lain tanpa harus menunggu dependensinya, atau rangkaian dependensinya, untuk mengirimkan respons.

Ketika beban kerja Anda perlu berkomunikasi secara asinkron dengan dependensinya, pertimbangkan panduan berikut:

- Tentukan apakah perpesanan atau streaming peristiwa perlu digunakan berdasarkan kasus penggunaan dan kebutuhan Anda. Perpesanan memungkinkan beban kerja Anda untuk berkomunikasi dengan dependensinya dengan mengirimkan dan menerima pesan melalui broker pesan. Streaming peristiwa memungkinkan beban kerja Anda dan dependensinya untuk menggunakan layanan streaming untuk memublikasikan dan berlangganan peristiwa, yang disampaikan dalam bentuk aliran data berkelanjutan, yang perlu diproses sesegera mungkin.

- Perpesanan dan streaming peristiwa menangani pesan secara berbeda sehingga Anda perlu mengambil keputusan kompromi berdasarkan:
  - Prioritas pesan: broker pesan dapat memproses pesan prioritas tinggi sebelum pesan normal. Dalam streaming peristiwa, semua pesan memiliki prioritas yang sama.
  - Pemakaian pesan: broker pesan memastikan bahwa pemakai menerima pesan. Pemakai streaming peristiwa harus terus melacak pesan terakhir yang telah mereka baca.

  - Pengurutan pesan: dengan perpesanan, pesan tidak dijamin diterima dengan urutan sesuai pengirimannya kecuali Anda menggunakan pendekatan first-in-first-out (FIFO). Streaming peristiwa selalu mempertahankan urutan sesuai urutan data dibuat.
  - Penghapusan pesan: dengan perpesanan, konsumen harus menghapus pesan setelah memprosesnya. Layanan streaming peristiwa menambahkan pesan ke aliran dan tetap berada di sana sampai periode penyimpanan pesan berakhir. Kebijakan penghapusan ini menjadikan streaming peristiwa cocok untuk pemutaran ulang pesan.
- Tentukan bagaimana beban kerja Anda mengetahui kapan dependensinya menyelesaikan pekerjaan. Misalnya, ketika beban kerja Anda menginvokasi fungsi Lambda secara asinkron, Lambda menempatkan peristiwa di dalam antrean dan mengembalikan respons yang berhasil tanpa informasi tambahan. Setelah pemrosesan selesai, fungsi Lambda dapat mengirimkan hasil ke tujuan, yang dapat dikonfigurasi berdasarkan keberhasilan atau kegagalan.
- Bangun beban kerja Anda untuk menangani pesan duplikat dengan memanfaatkan idempotensi. Idempotensi adalah ketika hasil beban kerja Anda tidak berubah bahkan jika beban kerja Anda dihasilkan lebih dari sekali untuk pesan yang sama. Penting untuk ditekankan bahwa layanan perpesanan atau streaming akan mengirimkan kembali sebuah pesan jika terjadi kegagalan jaringan atau jika konfirmasi belum diterima.
- Jika beban kerja Anda tidak mendapatkan respons dari dependensinya, permintaan perlu dikirimkan kembali. Pertimbangkan untuk membatasi jumlah percobaan ulang untuk menghemat sumber daya CPU, memori, dan jaringan beban kerja Anda untuk menangani permintaan lainnya. Dokumentasi AWS Lambda menunjukkan cara menangani kesalahan untuk panggilan asinkron.
- Manfaatkan alat-alat observabilitas, debugging, dan pelacakan yang sesuai untuk mengelola dan mengoperasikan komunikasi asinkron beban kerja Anda dengan dependensinya. Anda dapat menggunakan Amazon CloudWatch untuk memantau layanan perpesanan dan streaming peristiwa. Anda juga dapat menginstrumentasi beban kerja Anda dengan AWS X-Ray untukmendapatkan wawasan secara cepat untuk memecahkan masalah.

Dependensi batch

Sistem batch mengambil data input, memulai serangkaian pekerjaan untuk memprosesnya, dan menghasilkan beberapa data output, tanpa intervensi manual. Tergantung ukuran data, pekerjaan dapat berjalan dari hitungan menit hingga, dalam beberapa kasus, beberapa hari. Ketika beban kerja Anda berkomunikasi dengan dependensinya, pertimbangkan panduan berikut:

- Tentukan rentang periode ketika beban kerja Anda harus menjalankan tugas batch. Beban kerja Anda dapat mengatur pola pengulangan untuk menginvokasi sistem batch, misalnya, setiap jam atau pada akhir setiap bulan.
- Tentukan lokasi input data dan output data yang diproses. Pilih layanan penyimpanan, seperti Amazon Simple Storage Service (Amazon S3), Amazon Elastic File System (Amazon EFS), dan Amazon FSx for Lustre, yang memungkinkan beban kerja Anda membaca dan menulis file dalam skala besar.
- Jika beban kerja Anda perlu menginvokasi beberapa pekerjaan batch, Anda dapat memanfaatkan AWS Step Functions untuk menyederhanakan orkestrasi pekerjaan batch yang berjalan di AWS atau on-premise. Proyek sampel ini menunjukkan orkestrasi pekerjaan batch menggunakan Step Functions, AWS Batch, dan Lambda.
- Pantau pekerjaan batch untuk mencari kelainan, seperti pekerjaan yang membutuhkan waktu lebih lama dari yang seharusnya. Anda dapat menggunakan alat seperti Wawasan Kontainer CloudWatch untuk memantau lingkungan dan pekerjaan AWS Batch. Dalam hal ini, beban kerja Anda akan menghentikan pekerjaan berikutnya dari awal dan memberitahukan pengecualian kepada staf yang relevan.

Sumber daya

Dokumen terkait:

- AWS Cloud Operations: Monitoring and Observability
- The Amazon's Builder Library: Challenges with distributed systems
- REL11-BP05 Menggunakan stabilitas statis untuk mencegah perilaku bimodal
- Panduan DeveloperAWS Lambda: Penanganan kesalahan dan percobaan ulang otomatis di AWS Lambda
- Menyetel pengaturan permintaan HTTP SDK Java AWS untuk aplikasi Amazon DynamoDB sadar latensi
- Perpesanan AWS
- Apa itu data streaming?
- Panduan Developer AWS Lambda: Panggilan asinkron
- Pertanyaan Umum Amazon Simple Queue Service: Antrean FIFO
- Panduan Developer Amazon Kinesis Data Streams: Menangani Catatan Duplikat

- Panduan Developer Amazon Simple Queue Service: Metrik CloudWatch yang tersedia untuk Amazon SQS
- Panduan Developer Amazon Kinesis Data Streams: Memantau Layanan Amazon Kinesis Data Streams dengan Amazon CloudWatch
- Panduan Developer AWS X-Ray: Konsep AWS X-Ray
- Sampel AWS di GitHub: Aplikasi Orkestrator Kompleks AWS Step Functions
- Panduan Pengguna AWS Batch: AWS Batch CloudWatch Container Insights

Video terkait:

- AWS Summit SF 2022 - Full-stack observability and application monitoring with AWS (COP310)

Alat terkait:

- Amazon CloudWatch
- Amazon CloudWatch Logs
- AWS X-Ray
- Amazon Simple Storage Service (Amazon S3)
- Amazon Elastic File System (Amazon EFS)
- Amazon FSx for Lustre
- AWS Step Functions
- AWS Batch

REL04-BP02 Mengimplementasikan dependensi yang digabungkan secara longgar

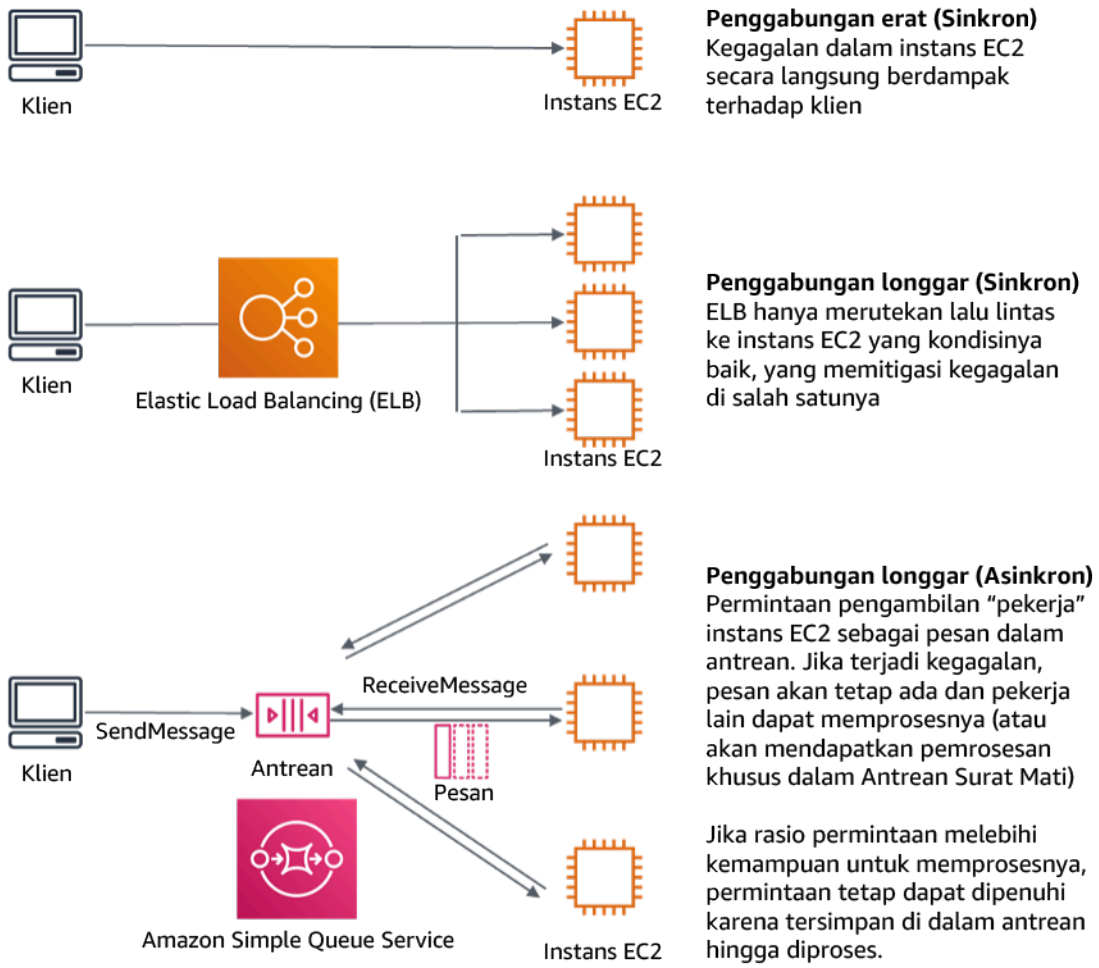
Dependensi seperti sistem pengantrean, sistem streaming, alur kerja, dan penyeimbang beban digabungkan secara longgar. Penggabungan longgar membantu memisahkan perilaku suatu komponen dari komponen lainnya yang bergantung pada komponen tersebut, sehingga meningkatkan ketahanan dan ketangkasan.

Dalam sistem penggabungan erat, perubahan pada satu komponen dapat menyebabkan perubahan pada komponen lain yang bergantung padanya, yang mengakibatkan penurunan performa di semua komponen. Penggabungan longgar menghilangkan dependensi ini sehingga komponen-komponen

yang bergantung hanya perlu mengetahui antarmuka versi terbaru dan yang dipublikasikan. Implementasi penggabungan longgar antar dependensi memisahkan kegagalan pada salah satu dependensi agar tidak memengaruhi dependensi lain.

Penggabungan longgar memungkinkan Anda untuk mengubah kode atau menambahkan fitur ke sebuah komponen sambil meminimalkan risiko pada komponen lain yang bergantung pada komponen tersebut. Hal ini juga memungkinkan ketahanan granular pada tingkat komponen sehingga Anda dapat menskalakan ke luar atau bahkan mengubah implementasi yang mendasari dependensi.

Agar makin meningkatkan ketahanan melalui penggabungan longgar, jadikan interaksi komponen asinkron apabila memungkinkan. Model ini cocok untuk interaksi apa pun yang tidak memerlukan respons cepat dan ketika terdaftarnya suatu permintaan cukup perlu diketahui. Ini melibatkan satu komponen yang menghasilkan peristiwa dan komponen lain yang menggunakannya. Kedua komponen tersebut tidak terintegrasi melalui interaksi titik ke titik langsung, tetapi biasanya melalui lapisan penyimpanan tahan lama perantara, seperti antrean Amazon SQS atau platform data streaming seperti Amazon Kinesis, atau AWS Step Functions.



Gambar 4: Dependensi seperti sistem pengantrean dan penyeimbang beban digabungkan secara longgar.

Antrean Amazon SQS dan Penyeimbang Beban Elastis hanyalah dua cara untuk menambahkan lapisan perantara untuk penggabungan longgar. Arsitektur yang didorong peristiwa juga dapat dibangun di AWS Cloud menggunakan Amazon EventBridge, yang dapat mengabstraksi klien (penghasil peristiwa) dari layanan yang mereka andalkan (pemakai peristiwa). Amazon Simple Notification Service (Amazon SNS) adalah solusi efektif ketika Anda memerlukan olah pesan dari banyak ke banyak dengan throughput tinggi dan berbasis push. Menggunakan topik Amazon SNS, sistem penerbit Anda dapat menyebarkan pesan ke titik akhir pelanggan dalam jumlah besar untuk pemrosesan paralel.

Meskipun antrean menawarkan sejumlah manfaat, di sebagian besar sistem waktu nyata yang keras, permintaan yang lebih lama dari waktu ambang batas (sering kali dalam hitungan detik) harus dianggap basi (klien telah menyerah dan sudah tidak menunggu respons), dan tidak diproses.

Dengan begitu, permintaan yang lebih baru (dan kemungkinan masih valid) dapat diproses sebagai gantinya.

Hasil yang diinginkan: Menerapkan dependensi penggabungan longgar memungkinkan Anda untuk meminimalkan area kegagalan ke tingkat komponen, yang membantu mendiagnosis dan menyelesaikan masalah. Cara ini juga dapat menyederhanakan siklus pengembangan, sehingga memungkinkan tim untuk menerapkan perubahan pada tingkat modular tanpa memengaruhi performa komponen lain yang bergantung padanya. Pendekatan ini memberikan kemampuan untuk menskalakan ke luar pada tingkat komponen berdasarkan kebutuhan sumber daya, serta pemanfaatan komponen yang berkontribusi terhadap efektivitas biaya.

Antipola umum:

- Melakukan deployment beban kerja monolitik.
- Memanggil API antar tingkatan beban kerja secara langsung tanpa kemampuan failover atau pemrosesan permintaan secara asinkron.
- Penggabungan erat menggunakan data bersama. Sistem yang digabungkan secara longgar sebaiknya tidak berbagi data melalui basis data bersama atau bentuk penyimpanan data yang digabungkan secara erat, yang dapat menimbulkan kembali penggabungan erat dan menghambat skalabilitas.
- Mengabaikan tekanan balik. Beban kerja Anda harus memiliki kemampuan untuk memperlambat atau menghentikan data yang masuk ketika komponen tidak dapat memprosesnya pada kecepatan yang sama.

Manfaat menetapkan praktik terbaik ini: Penggabungan longgar membantu mengisolasi perilaku komponen dari komponen lain yang bergantung padanya, sehingga meningkatkan ketahanan dan ketangkasan. Kegagalan di salah satu komponen dipisahkan dari komponen lain.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Implementasikan dependensi yang digabungkan secara longgar. Ada berbagai solusi yang memungkinkan Anda membangun aplikasi yang digabungkan secara longgar. Ini meliputi, beberapa di antaranya, layanan untuk mengimplementasikan antrean yang dikelola sepenuhnya, alur kerja otomatis, reaksi terhadap peristiwa, dan API yang dapat membantu mengisolasi perilaku komponen dari komponen lain, dan dengan demikian meningkatkan ketahanan dan ketangkasan.

- Membangun arsitektur yang didorong peristiwa: Amazon EventBridge membantu Anda membangun arsitektur berbasis peristiwa yang digabungkan secara longgar dan terdistribusi.
- Menerapkan antrean dalam sistem terdistribusi: Anda dapat menggunakan Amazon Simple Queue Service (Amazon SQS) untuk mengintegrasikan dan memisahkan sistem terdistribusi.
- Kontainerisasi komponen sebagai layanan mikro: Layanan mikro memungkinkan tim untuk membangun aplikasi yang terdiri dari komponen independen kecil yang berkomunikasi melalui API yang ditentukan dengan jelas. Amazon Elastic Container Service (Amazon ECS), dan Amazon Elastic Kubernetes Service (Amazon EKS) dapat membantu Anda memulai lebih cepat dengan kontainer.
- Kelola alur kerja dengan Step Functions: Step Functions membantu Anda mengoordinasikan beberapa layanan AWS menjadi alur kerja yang fleksibel.
- Manfaatkan arsitektur olah pesan publikasi-berlangganan (pub/sub): Amazon Simple Notification Service (Amazon SNS) menyediakan pengiriman pesan dari penerbit ke pelanggan (juga dikenal sebagai produsen dan konsumen).

Langkah implementasi

- Komponen dalam arsitektur yang didorong peristiwa dimulai oleh peristiwa. Peristiwa adalah tindakan yang terjadi dalam sistem, seperti pengguna menambahkan item ke keranjang. Ketika suatu tindakan berhasil, sebuah peristiwa dihasilkan, yang menggerakkan komponen berikutnya dari sistem.
  - Building Event-driven Applications with Amazon EventBridge
  - AWS re:Invent 2022 - Designing Event-Driven Integrations using Amazon EventBridge
- Sistem olah pesan terdistribusi memiliki tiga bagian utama yang perlu diimplementasikan untuk arsitektur berbasis antrean. Bagian-bagian tersebut meliputi komponen sistem terdistribusi, antrean yang digunakan untuk pemisahan (didistribusikan di server Amazon SQS), dan pesan dalam antrean. Sistem tipikal memiliki produsen yang memulai pesan ke dalam antrean, dan konsumen yang menerima pesan dari antrean tersebut. Antrean menyimpan pesan di beberapa server Amazon SQS untuk redundansi.
  - Basic Amazon SQS architecture
  - Send Messages Between Distributed Applications with Amazon Simple Queue Service
- Layanan mikro, jika dimanfaatkan dengan baik, akan meningkatkan pemeliharaan dan mendongkrak skalabilitas, karena komponen yang digabungkan secara longgar dikelola oleh tim independen. Hal ini juga memungkinkan isolasi perilaku ke satu komponen jika terjadi perubahan.

  - Mengimplementasikan Layanan Mikro di AWS
  - Let's Architect! Architecting microservices with containers
- Dengan AWS Step Functions Anda dapat membangun aplikasi terdistribusi, mengotomatiskan proses, mengorkestrasi layanan mikro, serta berbagai hal lainnya. Orkestrasi beberapa komponen ke dalam alur kerja otomatis memungkinkan Anda untuk memisahkan dependensi dalam aplikasi Anda.
  - Create a Serverless Workflow with AWS Step Functions and AWS Lambda
  - Mulai menggunakan AWS Step Functions

Sumber daya

Dokumen terkait:

- Amazon EC2: Ensuring Idempotency
- Amazon Builders' Library: Tantangan dengan sistem terdistribusi
- Amazon Builders' Library: Keandalan, tugas konstan, dan pilihan yang tepat
- Apa Itu Amazon EventBridge?
- Apa Itu Amazon Simple Queue Service?
- Break up with your monolith
- Orchestrate Queue-based Microservices with AWS Step Functions and Amazon SQS
- Basic Amazon SQS architecture
- Queue-Based Architecture

Video terkait:

- AWS New York Summit 2019: Intro to Event-driven Architectures and Amazon EventBridge (MAD205)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (includes loose coupling, constant work, static stability)
- AWS re:Invent 2019: Moving to event-driven architectures (SVS308)
- AWS re:Invent 2019: Scalable serverless event-driven applications using Amazon SQS and Lambda (API304)

- AWS re:Invent 2019: Scalable serverless event-driven applications using Amazon SQS and Lambda
- AWS re:Invent 2022 - Designing event-driven integrations using Amazon EventBridge
- AWS re:Invent 2017: Elastic Load Balancing Deep Dive and Best Practices

REL04-BP03 Melakukan tugas konstan

Sistem dapat gagal mengalami kegagalan saat ada perubahan besar dan cepat pada beban. Misalnya, jika beban kerja Anda sedang melakukan pemeriksaan kondisi yang memantau kondisi dari ribuan server, beban kerja Anda harus mengirimkan payload berukuran sama (snapshot penuh berisi status saat ini) setiap saat. Saat tidak ada server yang gagal, atau semuanya gagal, sistem pemeriksaan kondisi melakukan tugas konstan tanpa perubahan besar dan cepat.

Misalnya, jika sistem pemeriksaan kondisi sedang memantau 100.000 server, beban di dalamnya kecil, dengan tingkat kegagalan server normal yang ringan. Namun, jika sebuah peristiwa besar menjadikan separuh server tidak sehat, sistem pemeriksaan kondisi akan kewalahan untuk memperbarui sistem notifikasi dan menyampaikan status ke kliennya. Jadi sebagai gantinya, sistem pemeriksaan kondisi harus mengirimkan snapshot penuh berisi status saat ini setiap saat. 100.000 status sehat server, masing-masing diwakili satu bit, hanyalah satu payload berukuran 12,5 KB. Saat tidak ada server yang gagal, atau semuanya gagal, sistem pemeriksaan kondisi melakukan tugas konstan, dan perubahan yang besar dan cepat bukanlah ancaman untuk stabilitas sistem. Seperti inilah Amazon Route 53 menangani pemeriksaan kondisi untuk titik akhir (seperti alamat IP) untuk menentukan bagaimana pengguna akhir dirutekan ke sana.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Rendah

Panduan implementasi

- Lakukan tugas konstan sehingga sistem tidak gagal saat terdapat perubahan beban yang besar dan cepat.
- Implementasikan dependensi yang digabungkan secara longgar. Dependensi seperti sistem pengantrean, sistem streaming, alur kerja, dan penyeimbang beban digabungkan secara longgar. Penggabungan longgar membantu memisahkan perilaku suatu komponen dari komponen lainnya yang bergantung pada komponen tersebut, sehingga meningkatkan ketahanan dan ketangkasan.
  - Amazon Builders' Library: Keandalan, kerja konstan, dan secangkir kopi yang enak
  - AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (mencakup tugas konstan)

- Untuk contoh sistem pemeriksaan kondisi yang memantau 100.000 server, rekayasa beban kerja sehingga ukuran payload tetap sama berapa pun jumlah keberhasilan atau kegagalan.

Sumber daya

Dokumen terkait:

- Amazon EC2: Memastikan Idempotensi
- Amazon Builders' Library: Tantangan dengan sistem terdistribusi
- Amazon Builders' Library: Keandalan, kerja konstan, dan secangkir kopi yang enak

Video terkait:

- AWS New York Summit 2019: Intro to Event-driven Architectures and Amazon EventBridge (MAD205)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (mencakup tugas konstan)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (mencakup penggabungan longgar, kerja konstan, stabilitas statis)
- AWS re:Invent 2019: Moving to event-driven architectures (SVS308)

REL04-BP04 Menjadikan semua respons idempoten

Layanan idempoten menjanjikan setiap permintaan diselesaikan tepat satu kali, sehingga pembuatan beberapa permintaan yang sama memiliki efek yang sama seperti membuat satu permintaan. Layanan idempoten memudahkan klien untuk mengimplementasikan percobaan ulang tanpa takut permintaan akan salah diproses beberapa kali. Untuk melakukan ini, klien dapat mengeluarkan permintaan API dengan token idempotensi—token yang sama digunakan setiap permintaan diulang. API layanan idempoten menggunakan token untuk mengembalikan respons yang identik dengan respons yang dikembalikan saat pertama kali permintaan diselesaikan.

Dalam sistem terdistribusi, mudah untuk melakukan tindakan paling banyak satu kali (klien hanya membuat satu permintaan), atau setidaknya satu kali (tetap meminta sampai klien mendapat konfirmasi berhasil). Namun, sulit untuk menjamin suatu tindakan bersifat idempoten, yang berarti tindakan dilakukan tepat satu kali, sehingga pembuatan beberapa permintaan identik memiliki efek

yang sama seperti membuat satu permintaan. Menggunakan token idempotensi di API, layanan dapat menerima permintaan yang bermutasi satu kali atau lebih tanpa membuat rekaman ganda atau efek samping.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Sedang

Panduan implementasi

- Menjadikan semua respons idempoten. Layanan idempoten menjanjikan setiap permintaan diselesaikan tepat satu kali, sehingga pembuatan beberapa permintaan yang sama memiliki efek yang sama seperti membuat satu permintaan.
  - Klien dapat mengeluarkan permintaan API dengan token idempotensi—token yang sama digunakan setiap permintaan diulang. API layanan idempoten menggunakan token untuk mengembalikan respons yang identik dengan respons yang dikembalikan saat pertama kali permintaan diselesaikan.
    - Amazon EC2: Memastikan Idempotensi

Sumber daya

Dokumen terkait:

- Amazon EC2: Memastikan Idempotensi
- Amazon Builders' Library: Tantangan dengan sistem terdistribusi
- Amazon Builders' Library: Keandalan, kerja konstan, dan secangkir kopi yang enak

Video terkait:

- AWS New York Summit 2019: Intro to Event-driven Architectures and Amazon EventBridge (MAD205)
- AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small ARC337 (mencakup penggabungan longgar, kerja konstan, stabilitas statis)
- AWS re:Invent 2019: Moving to event-driven architectures (SVS308)

Mendesain interaksi dalam sistem terdistribusi untuk mitigasi atau bertahan dari kegagalan

Sistem terdistribusi mengandalkan jaringan komunikasi untuk membuat interkoneksi komponen (seperti server atau layanan). Beban kerja Anda harus beroperasi secara andal walaupun terdapat latensi atau kehilangan data di jaringannya. Komponen sistem terdistribusi harus beroperasi tanpa memberikan dampak secara negatif kepada komponen dan beban kerja yang lain. Berbagai praktik terbaik ini memungkinkan beban kerja bertahan dari tekanan atau kegagalan, pulih dengan lebih cepat, serta memitigasi dampak gangguan tersebut. Hasilnya adalah peningkatan waktu rata-rata untuk pemulihan (MTTR).

Berbagai praktik terbaik ini mencegah kegagalan dan meningkatkan waktu rata-rata antara kegagalan (MTBF).

Praktik terbaik

- REL05-BP01 Mengimplementasikan degradasi yang tepat (graceful degradation) untuk mengubah dependensi keras yang berlaku menjadi dependensi lunak
- REL05-BP02 Membatasi (throttling) permintaan
- REL05-BP03 Mengontrol dan membatasi panggilan percobaan ulang
- REL05-BP04 Melakukan gagal cepat (fail fast) dan membatasi antrean
- REL05-BP05 Mengatur batas waktu klien
- REL05-BP06 Membuat sistem menjadi stateless jika memungkinkan
- REL05-BP07 Mengimplementasikan tuas darurat

REL05-BP01 Mengimplementasikan degradasi yang tepat (graceful degradation) untuk mengubah dependensi keras yang berlaku menjadi dependensi lunak

Komponen aplikasi harus terus menjalankan fungsi intinya bahkan jika dependensi menjadi tidak tersedia. Komponen mungkin menyajikan data yang sedikit basi, data alternatif, atau bahkan tidak menyajikan data sama sekali. Hal ini memastikan fungsi sistem secara keseluruhan hanya terhambat secara minimum oleh kegagalan lokal sekaligus memberikan nilai bisnis utama.

Hasil yang diinginkan: Saat dependensi komponen tidak optimum, komponen tersebut masih dapat berfungsi, meskipun terbatas atau terdegradasi. Mode-mode kegagalan komponen harus dipandang

sebagai operasi normal. Alur kerja harus dirancang sedemikian rupa sehingga kegagalan tersebut tidak menyebabkan kegagalan total atau setidaknya hanya menyebabkan keadaan yang dapat diprediksi dan dapat dipulihkan.

Antipola umum:

- Tidak mengidentifikasi fungsi bisnis inti yang dibutuhkan. Tidak menguji bahwa komponen berfungsi bahkan selama kegagalan dependensi.
- Tidak menyajikan data jika terjadi kesalahan atau ketika hanya ada satu dari beberapa dependensi yang tidak tersedia dan hasil parsial masih dapat dikembalikan.
- Menciptakan keadaan yang tidak konsisten ketika transaksi gagal sebagian.
- Tidak memiliki cara alternatif untuk mengakses tempat penyimpanan parameter pusat.
- Membatalkan atau mengosongkan status lokal sebagai akibat dari penyegaran yang gagal tanpa mempertimbangkan konsekuensi tindakan tersebut.

Manfaat menjalankan praktik terbaik ini: Degradasi bertahap (graceful degradation) meningkatkan ketersediaan sistem secara keseluruhan dan mempertahankan fungsionalitas fungsi-fungsi yang paling penting, bahkan selama kegagalan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Menerapkan degradasi yang tepat membantu meminimalkan dampak kegagalan dependensi pada fungsi komponen. Idealnya, sebuah komponen mendeteksi kegagalan dependensi dan menanganinya dengan cara yang berdampak minim pada pelanggan atau komponen lain.

Merancang untuk degradasi yang tepat berarti mempertimbangkan potensi mode kegagalan selama desain dependensi. Untuk setiap mode kegagalan, miliki cara untuk menghadirkan sebagian besar atau setidaknya fungsionalitas paling penting dari komponen kepada pemanggil atau pelanggan. Pertimbangan ini dapat menjadi persyaratan tambahan yang dapat diuji dan diverifikasi. Idealnya, sebuah komponen mampu menjalankan fungsi intinya dengan cara yang dapat diterima bahkan ketika satu atau beberapa dependensi gagal.

Ini bukan hanya pembahasan teknis, melainkan juga pembahasan bisnis. Semua persyaratan bisnis penting dan harus dipenuhi jika memungkinkan. Namun, menanyakan apa yang seharusnya terjadi ketika tidak semua persyaratan tersebut dapat dipenuhi adalah hal yang wajar. Suatu sistem dapat

dirancang agar tersedia dan konsisten, tetapi dalam keadaan yang mengharuskan satu persyaratan untuk dikorbankan, mana yang lebih penting? Untuk pemrosesan pembayaran, jawabannya mungkin adalah konsistensi. Untuk aplikasi waktu nyata, jawabannya mungkin adalah ketersediaan. Untuk situs web yang digunakan langsung oleh pelanggan, jawabannya mungkin tergantung pada ekspektasi pelanggan.

Seberapa pentingnya, ini tergantung persyaratan komponen dan apa yang seharusnya dianggap sebagai fungsi intinya. Misalnya:

- Situs web ecommerce mungkin menampilkan data dari berbagai sistem seperti rekomendasi yang dipersonalisasi, produk dengan peringkat tertinggi, dan status pesanan pelanggan di halaman arahan. Ketika salah satu sistem hulu gagal, masih masuk akal untuk menampilkan semua daripada halaman kesalahan kepada pelanggan.
- Sebuah komponen yang menjalankan penulisan batch masih dapat melanjutkan pemrosesan batch jika salah satu operasi gagal. Implementasi mekanisme percobaan ulang harus sederhana. Hal ini dapat dilakukan dengan mengembalikan informasi tentang operasi yang berhasil, yang telah gagal, dan mengapa operasi tersebut gagal ke pemanggil, atau dengan menempatkan permintaan yang gagal ke dalam antrean surat mati untuk mengimplementasikan percobaan ulang asinkron. Informasi tentang operasi yang gagal juga harus dibuat log.
- Sistem yang memproses transaksi harus memverifikasi bahwa semua pembaruan individual dijalankan atau tidak sama sekali. Untuk transaksi terdistribusi, pola saga dapat digunakan untuk kembali ke operasi sebelumnya jika operasi selanjutnya dari transaksi yang sama gagal. Di sini, fungsi intinya adalah menjaga konsistensi.
- Sistem-sistem time-critical harus mampu menangani dependensi yang tidak merespons secara tepat waktu. Dalam kasus-kasus ini, pola pemutus sirkuit dapat digunakan. Ketika respons dari dependensi mulai mencapai batas waktu, sistem dapat beralih ke keadaan ditutup di mana tidak ada panggilan tambahan yang dibuat.
- Sebuah aplikasi dapat membaca parameter dari tempat penyimpanan parameter. Membuat image kontainer dengan serangkaian parameter default akan membantu agar apabila tempat penyimpanan parameter tidak tersedia image tersebut dapat digunakan.

Perhatikan bahwa jalur yang diambil jika terjadi kegagalan komponen perlu diuji dan harus jauh lebih sederhana daripada jalur utama. Umumnya, strategi fallback harus dihindari.

Langkah implementasi

Identifikasi dependensi eksternal dan internal. Pertimbangkan jenis-jenis kegagalan yang bisa terjadi di dalamnya. Pikirkan tentang cara-cara yang meminimalkan dampak negatif pada pelanggan serta sistem hulu dan hilir selama kegagalan-kegagalan tersebut.

Berikut ini adalah daftar dependensi dan cara melakukan degradasi yang tepat ketika dependensi gagal:

1. Kegagalan dependensi sebagian: Sebuah komponen dapat melakukan beberapa permintaan ke sistem-sistem hilir, baik beberapa permintaan ke satu sistem atau satu permintaan ke beberapa sistem. Tergantung konteks bisnis, mungkin ada berbagai cara penanganan yang sesuai (untuk detail lebih lanjut, lihat contoh-contoh sebelumnya dalam Panduan implementasi).
2. Sistem hilir tidak dapat memproses permintaan karena beban tinggi: Jika permintaan ke sistem hilir terus-menerus gagal, percobaan ulang tidak perlu dilanjutkan. Tindakan ini dapat menciptakan beban tambahan pada sistem yang sudah kelebihan beban dan mempersulit pemulihan. Pola pemutus sirkuit dapat digunakan di sini, yang memantau kegagalan panggilan ke sistem hilir. Jika ada banyak panggilan yang gagal, permintaan akan berhenti dikirimkan ke sistem hilir dan hanya sesekali panggilan dibiarkan masuk untuk menguji apakah sistem hilir sudah tersedia kembali.
3. Tempat penyimpanan parameter tidak tersedia: Untuk mengubah tempat penyimpanan parameter, caching dependensi lunak atau sane default yang disertakan di dalam image kontainer atau mesin dapat digunakan. Perhatikan bahwa default ini harus selalu diperbarui dan disertakan dalam rangkaian pengujian.
4. Layanan pemantauan atau dependensi non-fungsional lainnya tidak tersedia: Jika sebuah komponen sebentar-sebentar tidak dapat mengirim log, metrik, atau jejak ke layanan pemantauan pusat, langkah terbaiknya sering kali adalah tetap menjalankan fungsi-fungsi bisnis seperti biasa. Diam-diam tidak membuat log atau mendorong metrik dalam waktu yang lama sering kali tidak dapat diterima. Selain itu, beberapa kasus penggunaan mungkin memerlukan entri audit lengkap untuk memenuhi persyaratan kepatuhan.
5. Sebuah instans utama dari basis data relasional mungkin tidak tersedia: Amazon Relational Database Service, seperti hampir semua basis data relasional, hanya dapat memiliki satu instans penulis utama. Hal ini menciptakan satu titik kegagalan untuk beban kerja tulis dan menjadikan penskalaan lebih sulit. Hal ini dapat diatasi sebagiannya dengan menggunakan konfigurasi Multi-AZ untuk ketersediaan tinggi atau Nirserver Amazon Aurora untuk penskalaan yang lebih baik. Untuk persyaratan ketersediaan yang sangat tinggi, ada baiknya untuk tidak bergantung pada penulis utama sama sekali. Untuk kueri yang hanya membaca, replika baca dapat digunakan, yang memberikan redundansi dan kemampuan untuk melakukan scale-out, bukan hanya scale-

up. Tulis dapat di-buffer, misalnya dalam antrean Amazon Simple Queue Service, sehingga permintaan tulis dari pelanggan masih dapat diterima bahkan jika penulis utama tidak tersedia untuk sementara.

Sumber daya

Dokumen terkait:

- Amazon API Gateway: Membatasi Permintaan API untuk Peningkatan Throughput
- CircuitBreaker (rangkuman Pemutus Sirkuit dari buku “Release It!”)
- Kesalahan Percobaan Ulang dan Mundur Eksponensial di AWS
- Michael Nygard “Release It! Design and Deploy Production-Ready Software” (Rancang dan Lakukan Deployment Perangkat Lunak yang Siap Diproduksi)
- Pustaka Pengembang Amazon: Menghindari fallback dalam sistem terdistribusi
- Pustaka Pengembang Amazon: Menghindari backlog antrean yang tidak dapat diatasi
- Pustaka Pengembang Amazon: Tantangan dan strategi caching
- Pustaka Pengembang Amazon: Batas waktu, percobaan ulang, dan mundur (backoff) dengan jitter

Video terkait:

- Percobaan ulang, mundur, dan jitter: AWS re:Invent 2019: Memperkenalkan Pustaka Pengembang Amazon (DOP328)

Contoh terkait:

- Lab Well-Architected: Level 300: Mengimplementasikan Pemeriksaan Kondisi dan Mengelola Dependensi untuk Meningkatkan Keandalan

REL05-BP02 Membatasi (throttling) permintaan

Batasi permintaan untuk mengurangi keletihan sumber daya karena peningkatan permintaan yang tidak terduga. Permintaan di bawah tingkat throttling akan diproses, sedangkan permintaan di atas batas yang ditentukan akan ditolak dengan memunculkan pesan bahwa permintaan telah dibatasi.

Hasil yang diinginkan: Lonjakan volume besar baik dari peningkatan lalu lintas pelanggan yang tiba-tiba, serangan membanjir, atau banjir percobaan ulang akan diminimalkan dengan throttling

permintaan, sehingga beban kerja dapat melanjutkan pemrosesan volume permintaan normal yang didukung.

Antipola umum:

- Throttle titik akhir API tidak diimplementasikan atau dibiarkan pada nilai default tanpa mempertimbangkan volume yang diharapkan.
- Titik akhir API tidak diberi uji beban atau batas throttling tidak diuji.
- Membatasi angka permintaan tanpa mempertimbangkan ukuran atau kompleksitas permintaan.
- Menguji laju permintaan maksimum atau ukuran permintaan maksimum, tetapi tidak menguji keduanya bersama-sama.
- Sumber daya tidak disediakan untuk batas yang sama yang ditetapkan dalam pengujian.
- Rencana penggunaan belum dikonfigurasi atau dipertimbangkan untuk konsumen API aplikasi ke aplikasi (A2A).
- Tidak ada konfigurasi pengaturan konkurensi maksimum pada konsumen antrean yang diskalakan secara horizontal.
- Pembatasan tingkat per alamat IP belum diimplementasikan.

Manfaat menjalankan praktik terbaik ini: Beban kerja yang menetapkan batas throttle dapat beroperasi secara normal dan berhasil memproses beban permintaan yang diterima selama lonjakan volume yang tidak terduga. Lonjakan permintaan yang tiba-tiba atau terus menerus pada API dan antrean dibatasi dan tidak menghabiskan sumber daya pemrosesan permintaan. Batas angka permintaan membatasi setiap peminta sehingga volume lalu lintas yang tinggi dari satu alamat IP atau konsumen API tidak akan menghabiskan sumber daya atau berimbas pada konsumen lain.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Layanan harus dirancang untuk memproses kapasitas permintaan yang diketahui; kapasitas ini dapat ditetapkan melalui pengujian beban. Jika laju kedatangan permintaan melampaui batas, respons yang tepat menandakan bahwa permintaan telah dibatasi. Hal ini memungkinkan konsumen untuk menangani kesalahan dan mencoba ulang di lain waktu.

Saat layanan Anda memerlukan implementasi throttling, pertimbangkan mengimplementasikan algoritme bucket token, yang menghitung satu token sebagai satu permintaan. Token diisi ulang dengan laju throttle per detik dan dikosongkan secara asinkron oleh satu token per permintaan.

Algoritme bucket token.

Amazon API Gateway mengimplementasikan algoritme bucket token sesuai dengan batas yang dimiliki akun dan wilayah dan dapat dikonfigurasi per klien dengan rencana penggunaan. Selain itu, Amazon Simple Queue Service (Amazon SQS) dan Amazon Kinesis dapat melakukan buffer permintaan untuk memuluskan laju permintaan, dan memungkinkan tingkat throttling yang lebih tinggi untuk permintaan yang dapat diatasi. Terakhir, Anda dapat menerapkan pembatasan laju dengan AWS WAF untuk membatasi konsumen API tertentu yang menghasilkan beban yang terlalu tinggi.

Langkah implementasi

Anda dapat mengonfigurasi API Gateway dengan batas throttling untuk API dan mengembalikan pesan kesalahan `429 Terlalu Banyak Permintaan` ketika batas terlampaui. Anda dapat menggunakan AWS WAF dengan titik akhir AWS AppSync dan API Gateway Anda untuk mengaktifkan pembatasan laju per alamat IP. Selain itu, apabila sistem Anda dapat mentoleransi pemrosesan asinkron, Anda dapat memasukkan pesan ke dalam antrean atau aliran guna mempercepat respons terhadap klien layanan, yang memungkinkan Anda melakukan lonjakan ke tingkat throttle yang lebih tinggi.

Dengan pemrosesan asinkron, ketika Anda telah mengonfigurasi Amazon SQS sebagai sumber peristiwa untuk AWS Lambda, Anda dapat mengonfigurasi konkurensi maksimum untuk mencegah angka peristiwa yang tinggi memakai kuota eksekusi serentak akun yang tersedia yang diperlukan untuk layanan lain dalam beban kerja atau akun Anda.

Meskipun API Gateway menyediakan implementasi bucket token yang dikelola, apabila Anda tidak dapat menggunakan API Gateway, Anda dapat memanfaatkan implementasi sumber terbuka bahasa khusus (lihat contoh terkait di Sumber Daya) bucket token untuk layanan Anda.

- Pahami dan konfigurasi batas throttling API Gateway di tingkat akun per wilayah, API per tahap, dan kunci API per tingkat paket penggunaan.
- Terapkan aturan pembatas laju AWS WAF ke titik akhir API Gateway dan AWS AppSync untuk melindungi dari permintaan yang membanjir dan memblokir IP berbahaya. Aturan pembatas laju juga dapat dikonfigurasi pada kunci API AWS AppSync untuk konsumen A2A.
- Pertimbangkan apakah Anda memerlukan kontrol throttling yang lebih besar daripada pembatasan laju untuk API AWS AppSync, dan jika demikian, konfigurasikan API Gateway di depan titik akhir AWS AppSync Anda.
- Ketika antrean Amazon SQS diatur sebagai pemicu untuk konsumen antrean Lambda, tetapkan konkurensi maksimum ke nilai yang memproses cukup banyak untuk memenuhi tujuan tingkat layanan Anda tetapi tidak menggunakan batas konkurensi yang memengaruhi fungsi Lambda lain. Pertimbangkan untuk menetapkan konkurensi cadangan pada fungsi Lambda lain di akun dan wilayah yang sama saat Anda menggunakan antrean dengan Lambda.
- Gunakan API Gateway dengan integrasi layanan native ke Amazon SQS atau Kinesis untuk melakukan buffer permintaan.
- Jika Anda tidak dapat menggunakan API Gateway, lihat pustaka bahasa khusus untuk mengimplementasikan algoritme bucket token untuk beban kerja Anda. Periksa bagian contoh dan lakukan riset sendiri untuk menemukan pustaka yang cocok.
- Uji batas yang ingin Anda tetapkan, atau yang ingin Anda izinkan untuk ditingkatkan, dan dokumentasikan batas yang diuji.
- Jangan tingkatkan batas melebihi apa yang Anda tetapkan dalam pengujian. Saat meningkatkan batas, verifikasi bahwa sumber daya yang disediakan sudah setara atau lebih besar daripada yang ada dalam skenario pengujian sebelum menerapkan peningkatan.

Sumber daya

Praktik terbaik terkait:

- REL04-BP03 Melakukan tugas konstan
- REL05-BP03 Mengontrol dan membatasi panggilan percobaan ulang

Dokumen terkait:

- Amazon API Gateway: Membatasi Permintaan API untuk Peningkatan Throughput
- AWS WAF: Pernyataan aturan berbasis laju
- Memperkenalkan konkurensi maksimum AWS Lambda saat menggunakan Amazon SQS sebagai sumber peristiwa
- AWS Lambda: Konkurensi Maksimum

Contoh terkait:

- Tiga aturan berbasis laju AWS WAF yang paling penting
- Java Bucket4j
- Bucket token Python
- Bucket token Node
- Pembatasan Tingkat Threading Sistem .NET

Video terkait:

- Mengimplementasikan praktik terbaik keamanan API GraphQL dengan AWS AppSync

Alat terkait:

- Amazon API Gateway
- AWS AppSync
- Amazon SQS
- Amazon Kinesis
- AWS WAF

REL05-BP03 Mengontrol dan membatasi panggilan percobaan ulang

Gunakan mundur eksponensial untuk mencoba ulang permintaan dengan interval yang makin lama antara setiap percobaan ulang. Terapkan jitter antara percobaan ulang untuk mengacak interval percobaan ulang. Batasi jumlah percobaan ulang maksimum.

Hasil yang diinginkan: Komponen di sistem perangkat lunak terdistribusi biasanya mencakup server, penyeimbang beban, basis data, dan server DNS. Selama operasi normal, komponen-komponen ini dapat merespons permintaan dengan kesalahan yang bersifat sementara atau terbatas, dan juga kesalahan yang persisten terlepas dari percobaan ulang. Ketika klien membuat permintaan ke layanan, permintaan tersebut mengonsumsi sumber daya termasuk memori, thread, koneksi, port, atau sumber daya terbatas lainnya. Mengontrol dan membatasi percobaan ulang adalah strategi untuk melepaskan dan meminimalkan konsumsi sumber daya sehingga komponen sistem yang ada di bawah tekanan tidak kewalahan.

Ketika permintaan klien mengalami batas waktu atau menerima respons kesalahan, mereka harus menentukan apakah akan mencoba lagi atau tidak. Jika mereka mencoba lagi, mereka melakukannya dengan mundur eksponensial dengan jitter dan nilai coba ulang maksimum. Karena itu, layanan dan proses backend mendapat kelonggaran beban dan waktu untuk pulih secara mandiri, sehingga menghasilkan pemulihan yang lebih cepat dan pelayanan permintaan yang berhasil.

Antipola umum:

- Mengimplementasikan percobaan ulang tanpa menambahkan mundur eksponensial, jitter, dan nilai coba ulang maksimum. Mundur dan jitter membantu menghindari lonjakan lalu lintas semu yang disebabkan percobaan ulang yang dikoordinasikan secara tidak sengaja pada interval umum.
- Mengimplementasikan percobaan ulang tanpa menguji efeknya atau berasumsi bahwa percobaan ulang sudah terintegrasi ke dalam SDK tanpa menguji skenario percobaan ulang.
- Tidak memahami kode kesalahan yang dipublikasikan dari dependensi, yang menyebabkan percobaan ulang semua kesalahan, termasuk kesalahan dengan penyebab jelas yang menunjukkan tidak adanya izin, kesalahan konfigurasi, atau kondisi lain yang jelas tidak akan terselesaikan tanpa intervensi manual.
- Tidak menangani praktik observabilitas, termasuk pemantauan dan peringatan tentang kegagalan layanan berulang sehingga masalah yang mendasari dapat diketahui dan diatasi.
- Mengembangkan mekanisme percobaan ulang kustom saat kemampuan coba ulang bawaan atau pihak ketiga sudah mencukupi.
- Mencoba ulang pada beberapa lapisan tumpukan aplikasi dengan cara yang makin memperparah upaya-upaya percobaan ulang sehingga makin menyita sumber daya dalam badai percobaan ulang. Pastikan Anda memahami bagaimana kesalahan-kesalahan ini memengaruhi aplikasi Anda dan dependensi yang Anda andalkan, lalu terapkan percobaan ulang hanya pada satu tingkat.
- Mencoba ulang panggilan layanan yang tidak idempoten, sehingga menyebabkan efek samping yang tidak terduga seperti hasil-hasil ganda.

Manfaat menjalankan praktik terbaik ini: Percobaan ulang membantu klien memperoleh hasil yang diinginkan ketika permintaan gagal tetapi juga menyita lebih banyak waktu server untuk mendapatkan respons berhasil yang mereka inginkan. Ketika kegagalan jarang terjadi atau sementara, percobaan ulang dapat berfungsi dengan baik. Ketika kegagalan disebabkan oleh kelebihan beban sumber daya, percobaan ulang dapat memperburuk keadaan. Menambahkan mundur eksponensial dengan jitter ke percobaan ulang klien memungkinkan server pulih ketika kegagalan disebabkan oleh kelebihan beban sumber daya. Jitter menghindarkan penyelarasan permintaan menjadi lonjakan, dan mundur dapat mengurangi eskalasi beban yang disebabkan oleh penambahan percobaan ulang ke beban permintaan normal. Terakhir, penting untuk mengonfigurasi jumlah coba ulang maksimum atau waktu yang telah berlalu untuk menghindari terciptanya backlog yang menghasilkan kegagalan yang metastabil.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Mengontrol dan membatasi panggilan percobaan ulang. Gunakan mundur eksponensial untuk percobaan ulang setelah interval yang makin lama. Masukkan jitter untuk mengacak interval percobaan ulang dan batasi jumlah percobaan ulang maksimum.

Beberapa SDK AWS mengimplementasikan percobaan ulang dan mundur eksponensial secara default. Gunakan implementasi AWS bawaan ini jika diperlukan untuk beban kerja Anda. Implementasikan logika serupa dalam beban kerja Anda saat memanggil layanan yang idempoten dan apabila percobaan ulang meningkatkan ketersediaan klien Anda. Tentukan batas waktu dan kapan harus berhenti mencoba ulang berdasarkan kasus penggunaan Anda. Buat dan latih skenario pengujian untuk kasus penggunaan percobaan ulang tersebut.

Langkah implementasi

- Tentukan lapisan optimal dalam tumpukan aplikasi Anda untuk mengimplementasikan percobaan ulang untuk layanan yang diandalkan aplikasi Anda.
- Waspadai SDK yang ada yang menerapkan strategi percobaan ulang yang telah terbukti dengan mundur eksponensial dan jitter untuk bahasa pilihan Anda, dan pilih opsi ini daripada menulis implementasi percobaan ulang Anda sendiri.
- Verifikasikan bahwa layanan bersifat idempoten sebelum menerapkan percobaan ulang. Setelah percobaan ulang diterapkan, pastikan keduanya diuji dan latihlah secara rutin dalam produksi.

- Saat memanggil API layanan AWS, gunakan SDK AWS dan AWS CLI dan pahami opsi-opsi konfigurasi percobaan ulang. Tentukan apakah konfigurasi default cocok untuk kasus penggunaan Anda, uji, dan sesuaikan sesuai kebutuhan.

Sumber daya

Praktik terbaik terkait:

- REL04-BP04 Menjadikan semua respons idempoten
- REL05-BP02 Membatasi (throttling) permintaan
- REL05-BP04 Melakukan gagal cepat (fail fast) dan membatasi antrean
- REL05-BP05 Mengatur batas waktu klien
- REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan

Dokumen terkait:

- Kesalahan Percobaan Ulang dan Mundur Eksponensial di AWS
- Amazon Builders' Library: Batas waktu, percobaan ulang, dan mundur (backoff) dengan jitter
- Mundur Eksponensial dan Jitter
- Menjadikan percobaan ulang aman dengan API idempoten

Contoh terkait:

- Spring Retry
- Resilience4j Retry

Video terkait:

- Percobaan ulang, mundur, dan jitter: AWS re:Invent 2019: Memperkenalkan Pustaka Pengembang Amazon (DOP328)

Alat terkait:

- SDK dan Alat-Alat AWS: Perilaku percobaan ulang
- AWS Command Line Interface: Percobaan ulang AWS CLI

REL05-BP04 Melakukan gagal cepat (fail fast) dan membatasi antrean

Ketika layanan tidak berhasil merespons permintaan, lakukanlah gagal cepat (fail fast). Hal ini memungkinkan pelepasan sumber daya yang terkait dengan permintaan, dan mengizinkan layanan untuk melakukan pemulihan jika kehabisan sumber daya. Gagal cepat adalah pola desain perangkat lunak mapan yang dapat dimanfaatkan untuk membangun beban kerja yang sangat andal di cloud. Antrean juga merupakan pola integrasi korporat yang mapan yang dapat memperlancar beban dan memungkinkan klien untuk melepaskan sumber daya ketika pemrosesan asinkron dapat ditoleransi. Ketika layanan berhasil merespons dalam kondisi normal tetapi gagal ketika laju permintaan terlalu tinggi, gunakan antrean untuk melakukan buffer permintaan. Namun, jangan sampai ada penumpukan backlog antrean panjang yang dapat mengakibatkan diprosesnya permintaan yang telah kedaluwarsa dan telah ditinggalkan klien.

Hasil yang diinginkan: Ketika sistem berebut sumber daya, mengalami waktu habis, pengecualian, atau grey failure (kegagalan samar-samar) yang menyebabkan target tingkat layanan tidak dapat dicapai, strategi gagal cepat memungkinkan pemulihan sistem yang lebih cepat. Sistem yang harus menyerap lonjakan lalu lintas dan dapat mengakomodasi pemrosesan asinkron dapat meningkatkan keandalan dengan memungkinkan klien untuk secara cepat melepaskan permintaan dengan menggunakan antrean untuk melakukan buffer permintaan ke layanan backend. Ketika melakukan buffer permintaan ke antrean, strategi manajemen antrean diimplementasikan untuk menghindari backlog yang terlalu membebani.

Antipola umum:

- Mengimplementasikan antrean pesan tetapi tidak mengonfigurasi antrean surat mati (DLQ) atau alarm pada volume DLQ untuk mendeteksi kegagalan sistem.
- Tidak mengukur usia pesan dalam antrean, yaitu ukuran latensi untuk mengetahui kapan konsumen antrean tertinggal atau mengalami kesalahan yang menyebabkan percobaan ulang.
- Tidak menghapus pesan-pesan yang menumpuk dari antrean, padahal tidak ada gunanya memproses pesan-pesan tersebut jika kebutuhan bisnis sudah tidak ada.
- Mengonfigurasi antrean first in first out (FIFO), padahal antrean last in first out (LIFO) lebih memenuhi kebutuhan klien, misalnya ketika pengurutan yang ketat tidak diperlukan dan pemrosesan backlog menunda semua permintaan baru dan sensitif waktu sehingga semua klien merasa tingkat layanan gagal dipenuhi.
- Mengekspos antrean internal ke klien, bukan mengekspos API yang mengelola masuknya pekerjaan dan menempatkan permintaan ke dalam antrean internal.

- Menggabungkan terlalu banyak jenis permintaan kerja ke dalam satu antrean yang dapat memperburuk kondisi backlog dengan menyebarkan permintaan sumber daya di seluruh jenis permintaan.
- Memproses permintaan yang kompleks dan sederhana dalam antrean yang sama, sehingga mengabaikan perbedaan kebutuhan pemantauan, batas waktu, dan alokasi sumber daya.
- Tidak memvalidasi input atau menggunakan pernyataan untuk mengimplementasikan mekanisme gagal cepat dalam perangkat lunak yang menaikkan pengecualian ke komponen dengan level lebih tinggi yang dapat menangani kesalahan secara mulus.
- Tidak menghapus sumber daya yang rusak dari perutean permintaan, terutama ketika kegagalan samar-samar yang menunjukkan keberhasilan sekaligus kegagalan akibat crash dan mulai ulang, kegagalan dependensi intermiten, kapasitas yang menurun, atau hilangnya paket jaringan.

Manfaat menjalankan praktik terbaik ini: Sistem yang gagal cepat lebih mudah untuk di-debug dan diperbaiki, dan sering mengekspos masalah dalam pengodean dan konfigurasi sebelum rilis dipublikasikan ke tahap produksi. Sistem yang menggabungkan strategi antrean yang efektif memberikan ketahanan dan keandalan yang lebih baik terhadap lonjakan lalu lintas dan kondisi gangguan sistem intermiten.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Strategi gagal cepat dapat dikodekan ke dalam solusi perangkat lunak serta dikonfigurasi ke dalam infrastruktur. Selain gagal cepat, antrean adalah teknik arsitektur yang sederhana namun ampuh untuk memisahkan komponen-komponen sistem dan memperlancar beban. Amazon CloudWatch menyediakan kemampuan untuk memantau dan memberikan alarm kegagalan. Setelah sistem diketahui mengalami kegagalan, strategi mitigasi dapat dipanggil, termasuk gagal dan menjauh (fail away) dari sumber daya yang terdampak. Ketika sistem mengimplementasikan antrean dengan Amazon SQS dan teknologi antrean lainnya untuk melancarkan beban, sistem harus mempertimbangkan bagaimana mengelola backlog antrean, serta kegagalan konsumsi pesan.

Langkah implementasi

- Implementasikan pernyataan terprogram atau metrik tertentu dalam perangkat lunak Anda dan gunakan untuk memperingatkan secara eksplisit tentang masalah sistem. Amazon CloudWatch membantu Anda membuat metrik dan alarm berdasarkan pola log aplikasi dan instrumentasi SDK.

- Gunakan metrik dan alarm CloudWatch untuk gagal dan menjauh dari sumber daya terdampak yang menambahkan latensi ke pemrosesan atau berulang kali gagal memproses permintaan.
- Gunakan pemrosesan asinkron dengan merancang API untuk menerima permintaan dan menambahkan permintaan ke antrean internal menggunakan Amazon SQS kemudian menanggapi klien penghasil pesan dengan pesan keberhasilan sehingga klien dapat melepaskan sumber daya dan beralih dengan pekerjaan lain sementara konsumen antrean backend memproses permintaan.
- Ukur dan pantau latensi pemrosesan antrean dengan menghasilkan metrik CloudWatch setiap kali Anda melepaskan sebuah pesan dari antrean dengan membandingkan sekarang dengan stempel waktu pesan.
- Ketika kegagalan menghambat keberhasilan pemrosesan pesan atau volume lalu lintas melonjak sehingga tidak dapat diproses dalam batas perjanjian tingkat layanan, sisihkan lalu lintas yang lebih lama atau berlebih ke antrean spillover. Hal ini memungkinkan pemrosesan prioritas pada pekerjaan baru, dan pekerjaan yang lebih lama ketika kapasitas tersedia. Teknik ini mirip dengan pemrosesan LIFO dan memungkinkan pemrosesan sistem yang normal untuk semua pekerjaan baru.
- Gunakan antrean surat mati atau redrive untuk memindahkan pesan yang tidak dapat diproses dari backlog ke lokasi yang dapat dicari ulang dan diselesaikan lain waktu
- Coba lagi atau, apabila dapat ditoleransi, singkirkan pesan lama dengan membandingkan sekarang dengan stempel waktu pesan dan membuang pesan yang sudah tidak relevan dengan klien yang melakukan permintaan.

Sumber daya

Praktik terbaik terkait:

- REL04-BP02 Mengimplementasikan dependensi yang digabungkan secara longgar
- REL05-BP02 Membatasi (throttling) permintaan
- REL05-BP03 Mengontrol dan membatasi panggilan percobaan ulang
- REL06-BP02 Menetapkan dan menghitung metrik (Agregasi)
- REL06-BP07 Memantau pelacakan permintaan menyeluruh melalui sistem Anda

Dokumen terkait:

- Menghindari backlog antrian yang terlalu membebani
- Gagal Cepat (Fail Fast)

- Bagaimana cara mencegah peningkatan backlog pesan dalam antrean Amazon SQS saya?
- Elastic Load Balancing: Peralihan Zona
- Pengontrol Pemulihan Aplikasi Amazon Route 53: Kontrol perutean untuk failover lalu lintas

Contoh terkait:

- Pola Integrasi Korporat: Saluran Surat Mati

Video terkait:

- AWS re:Invent 2022 - Mengoperasikan aplikasi Multi-AZ dengan ketersediaan tinggi

Alat terkait:

- Amazon SQS
- Amazon MQ
- AWS IoT Core
- Amazon CloudWatch

REL05-BP05 Mengatur batas waktu klien

Atur batas waktu secara tepat pada koneksi dan permintaan, verifikasikan waktu tersebut secara sistematis, dan jangan selalu bergantung pada nilai default karena nilai tersebut mengabaikan hal-hal spesifik tentang beban kerja.

Hasil yang diinginkan: Batas waktu klien harus mempertimbangkan biaya untuk klien, server, dan beban kerja yang berkaitan dengan proses tunggu permintaan yang memerlukan waktu sangat lama untuk diselesaikan. Karena penyebab batas waktu tidak mungkin diketahui secara pasti, klien harus menggunakan pengetahuan tentang layanan untuk membangun ekspektasi tentang kemungkinan penyebab dan batas waktu yang tepat

Koneksi klien mengalami waktu habis berdasarkan nilai yang dikonfigurasi. Setelah mengalami batas waktu, klien mengambil keputusan untuk mundur dan mencobanya lagi atau membuka pemutus sirkuit. Pola-pola ini mencegah mengeluarkan permintaan yang dapat memperburuk kondisi kesalahan yang menyebabkannya.

Antipola umum:

- Tidak menyadari batas waktu sistem atau batas waktu default.
- Tidak menyadari waktu penyelesaian permintaan normal.
- Tidak menyadari kemungkinan penyebab permintaan membutuhkan waktu yang terlalu lama untuk diselesaikan, atau biaya untuk klien, layanan, atau kinerja beban kerja yang berkaitan dengan proses tunggu penyelesaian ini.
- Tidak menyadari kemungkinan jaringan rusak yang menyebabkan permintaan gagal hanya setelah batas waktu tercapai, dan biaya untuk klien dan kinerja beban kerja karena tidak mengadopsi batas waktu yang lebih singkat.
- Tidak menguji skenario batas waktu baik untuk koneksi maupun permintaan.
- Mengatur batas waktu terlalu tinggi, yang berimbas pada waktu tunggu yang lama dan meningkatkan pemanfaatan sumber daya.
- Mengatur batas waktu terlalu rendah, sehingga mengakibatkan kegagalan buatan.
- Mengabaikan pola-pola untuk menangani kesalahan batas waktu untuk panggilan jarak jauh seperti pemutus sirkuit dan percobaan ulang.
- Tidak mempertimbangkan pemantauan untuk angka kesalahan panggilan layanan, target latensi di tingkat layanan, dan outlier latensi. Metrik-metrik ini dapat memberikan wawasan tentang batas waktu yang agresif atau permisif

Manfaat menjalankan praktik terbaik ini: Waktu tunggu panggilan jarak jauh dikonfigurasi dan sistem dirancang untuk menangani batas waktu secara perlahan sehingga sumber daya dihemat ketika panggilan jarak jauh merespons terlalu lambat dan kesalahan batas waktu ditangani secara perlahan oleh klien layanan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Atur batas waktu koneksi dan batas waktu permintaan untuk panggilan dependensi layanan apa pun, serta secara umum untuk panggilan apa pun di seluruh proses. Banyak kerangka kerja yang menawarkan kemampuan batas waktu bawaan, tetapi Anda harus tetap memperhatikan bahwa nilai default bawaan bisa saja tidak terbatas atau lebih tinggi dari yang dapat diterima untuk sasaran layanan Anda. Nilai yang terlalu tinggi mengurangi kegunaan batas waktu karena sumber daya terus terpakai saat klien menunggu terjadinya batas waktu. Nilai yang terlalu rendah akan menyebabkan lalu lintas yang tinggi di backend serta meningkatkan latensi karena terlalu banyak permintaan yang dicoba ulang. Dalam beberapa kasus, hal ini dapat menyebabkan penghentian total karena semua permintaan dicoba ulang.

Pertimbangkan hal berikut saat menentukan strategi batas waktu:

- Permintaan mungkin membutuhkan waktu pemrosesan yang lebih lama dari biasanya dikarenakan kontennya, gangguan pada layanan target, atau kegagalan partisi jaringan.
- Permintaan dengan konten yang terlalu mahal dapat mengonsumsi sumber daya server dan klien yang tidak perlu. Dalam hal ini, membatasi waktu dan tidak mencoba ulang permintaan tersebut dapat menghemat sumber daya. Layanan juga harus melindungi diri dari konten yang terlalu mahal dengan throttle dan batas waktu sisi server.
- Permintaan yang memakan waktu terlalu lama karena gangguan layanan dapat diberikan batas waktu dan dicoba ulang. Pertimbangan harus diberikan pada biaya layanan untuk permintaan dan percobaan ulang, tetapi jika penyebabnya adalah gangguan yang terbatas di suatu tempat, percobaan ulang kemungkinan tidak mahal dan akan mengurangi konsumsi sumber daya klien. Batas waktu juga dapat melepaskan sumber daya server, tergantung sifat gangguan tersebut.
- Permintaan yang membutuhkan waktu penyelesaian yang lama karena permintaan atau respons gagal dikirimkan oleh jaringan dapat diberikan batas waktu dan dicoba ulang. Karena permintaan atau respons tidak dikirimkan, kegagalan akan terjadi, terlepas dari lamanya batas waktu. Memberikan batas waktu pada kasus ini tidak akan melepaskan sumber daya server, tetapi akan melepaskan sumber daya klien dan meningkatkan kinerja beban kerja.

Manfaatkan pola desain yang mapan seperti percobaan ulang dan pemutus sirkuit untuk menangani batas waktu dengan lancar dan mendukung pendekatan gagal cepat. SDK AWS dan AWS CLI memungkinkan konfigurasi batas waktu koneksi dan permintaan serta percobaan ulang dengan mundur eksponensial dan jitter. Fungsi AWS Lambda mendukung konfigurasi batas waktu, dan dengan AWS Step Functions, Anda dapat membangun pemutus sirkuit rendah kode yang memanfaatkan integrasi siap pakai dengan layanan dan SDK AWS. AWS App Mesh Envoy memberikan kemampuan batas waktu dan pemutus sirkuit.

Langkah implementasi

- Konfigurasikan batas waktu pada panggilan layanan jarak jauh dan manfaatkan fitur batas waktu bahasa bawaan atau pustaka batas waktu sumber terbuka.
- Saat beban kerja Anda melakukan panggilan dengan SDK AWS, tinjau dokumentasi untuk konfigurasi batas waktu untuk bahasa khusus.
  - Python
  - PHP
  - .NET

  - Ruby
  - Java
  - Go
  - Node.js
  - C++
- Saat menggunakan SDK AWS atau perintah AWS CLI dalam beban kerja Anda, konfigurasikan nilai batas waktu default dengan mengatur konfigurasi AWS default untuk `connectTimeoutInMillis` dan `tlsNegotiationTimeoutInMillis`.
- Terapkan opsi baris perintah `cli-connect-timeout` dan `cli-read-timeout` untuk mengontrol perintah AWS CLI satu kali ke layanan AWS.
- Pantau panggilan layanan jarak jauh untuk batas waktu, dan atur alarm pada kesalahan persisten sehingga Anda dapat menangani skenario kesalahan secara proaktif.
- Implementasikan Metrik CloudWatch dan deteksi anomali CloudWatch pada angka kesalahan panggilan, target latensi di tingkat layanan, dan outlier latensi untuk memberikan wawasan tentang pengelolaan batas waktu yang terlalu agresif atau permisif.
- Konfigurasikan batas waktu pada fungsi Lambda.
- Klien API Gateway harus mengimplementasikan percobaan ulang mereka sendiri saat menangani batas waktu. API Gateway mendukung batas waktu integrasi 50 milidetik hingga 29 detik untuk integrasi hilir dan tidak mencoba ulang saat integrasi meminta batas waktu.
- Implementasikan pola pemutus sirkuit untuk menghindari pembuatan panggilan jarak jauh ketika waktu habis. Buka sirkuit untuk menghindari kegagalan panggilan dan tutup sirkuit saat panggilan merespons secara normal.
- Untuk beban kerja berbasis kontainer, pelajari fitur App Mesh Envoy untuk memanfaatkan batas waktu dan pemutus sirkuit bawaan.
- Gunakan AWS Step Functions untuk membuat pemutus sirkuit rendah kode untuk panggilan layanan jarak jauh, terutama saat memanggil SDK native AWS dan integrasi Step Functions yang didukung untuk menyederhanakan beban kerja Anda.

Sumber daya

Praktik terbaik terkait:

- REL05-BP03 Mengontrol dan membatasi panggilan percobaan ulang

- REL05-BP04 Melakukan gagal cepat (fail fast) dan membatasi antrean
- REL06-BP07 Memantau pelacakan permintaan menyeluruh melalui sistem Anda

Dokumen terkait:

- SDK AWS: Percobaan Ulang dan Batas Waktu
- Amazon Builders' Library: Batas waktu, percobaan ulang, dan mundur (backoff) dengan jitter
- Kuota Amazon API Gateway dan catatan penting
- AWS Command Line Interface: Opsi baris perintah
- AWS SDK for Java 2.x: Mengonfigurasi Batas Waktu API
- AWS Botocore menggunakan objek konfigurasi dan Config Reference
- AWS SDK for .NET: Percobaan Ulang dan Batas Waktu
- AWS Lambda: Mengonfigurasi opsi fungsi Lambda

Contoh terkait:

- Menggunakan pola pemutus sirkuit dengan AWS Step Functions dan Amazon DynamoDB
- Martin Fowler: CircuitBreaker

Alat terkait:

- SDK AWS
- Fungsi AWS Lambda
- Amazon SQS
- AWS Step Functions
- AWS Command Line Interface

REL05-BP06 Membuat sistem menjadi stateless jika memungkinkan

Sistem seharusnya tidak memerlukan state, atau seharusnya mengalihkan state sedemikian rupa sehingga di antara permintaan klien yang berbeda, tidak ada dependensi di penyimpanan data lokal di disk dan memori. Hal ini membuat server dapat diganti sesuai keinginan tanpa menimbulkan dampak ketersediaan.

Ketika pengguna atau layanan berinteraksi dengan aplikasi, mereka sering melakukan serangkaian interaksi yang membentuk sesi. Sesi adalah data unik bagi pengguna yang lama berada di antara permintaan ketika mereka menggunakan aplikasi. Aplikasi stateless adalah aplikasi yang tidak memerlukan pengetahuan tentang interaksi sebelumnya dan tidak menyimpan informasi sesi.

Setelah dirancang menjadi stateless, Anda dapat menggunakan layanan komputasi nirserver, seperti AWS Lambda atau AWS Fargate.

Selain penggantian server, manfaat lain aplikasi stateless adalah kemampuannya untuk menyesuaikan skala secara horizontal karena sumber daya komputasi yang tersedia (seperti instans EC2 dan fungsi AWS Lambda) dapat melayani permintaan apa pun.

Manfaat menjalankan praktik terbaik ini: Sistem yang dirancang untuk menjadi stateless lebih mudah beradaptasi dengan penskalaan horizontal, sehingga memungkinkan penambahan atau penghapusan kapasitas berdasarkan lalu lintas dan permintaan yang berfluktuasi. Sistem ini juga memiliki sifat yang tahan terhadap kegagalan dan memberikan fleksibilitas serta ketangkasan dalam pengembangan aplikasi.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Menjadikan aplikasi Anda stateless. Aplikasi stateless memungkinkan penskalaan horizontal dan toleran terhadap kegagalan simpul individual. Analisis dan pahami komponen aplikasi Anda yang mempertahankan state di dalam arsitektur. Hal ini membantu Anda menilai dampak potensial dari transisi ke desain stateless. Arsitektur stateless memisahkan data pengguna dan memindahkan data sesi. Hal ini memberikan fleksibilitas untuk menskalakan setiap komponen secara independen untuk memenuhi permintaan beban kerja yang beragam dan mengoptimalkan pemanfaatan sumber daya.

Langkah implementasi

- Identifikasi dan pahami komponen stateful dalam aplikasi Anda.
- Pisahkan data dengan memisahkan dan mengelola data pengguna dari logika aplikasi inti.
  - Amazon Cognito dapat memisahkan data pengguna dari kode aplikasi dengan menggunakan fitur, seperti kolam identitas, kumpulan pengguna, dan Amazon Cognito Sync.
  - Anda dapat menggunakan AWS Secrets Manager untuk memisahkan data pengguna dengan menyimpan rahasia di lokasi terpusat yang aman. Ini berarti kode aplikasi tidak perlu menyimpan rahasia, sehingga membuatnya lebih aman.

    - Pertimbangkan menggunakan Amazon S3 untuk menyimpan data besar yang tidak terstruktur, seperti gambar dan dokumen. Aplikasi Anda dapat mengambil data tersebut apabila diperlukan, sehingga menghilangkan kebutuhan untuk menyimpannya di dalam memori.
    - Gunakan Amazon DynamoDB untuk menyimpan informasi seperti profil pengguna. Aplikasi Anda dapat mengueri data tersebut hampir dalam waktu nyata.
- Pindahkan data sesi ke basis data, cache, atau file eksternal.
    - Amazon ElastiCache, Amazon DynamoDB, Amazon Elastic File System (Amazon EFS), dan Amazon MemoryDB for Redis adalah contoh layanan AWS yang dapat Anda gunakan untuk memindahkan data sesi.
- Rancang arsitektur stateless setelah Anda mengidentifikasi state dan data pengguna mana yang perlu dipertahankan dengan solusi penyimpanan pilihan Anda.

Sumber daya

Praktik terbaik terkait:

- REL11-BP03 Mengotomatiskan pemulihan di semua lapisan

Dokumen terkait:

- Amazon Builders' Library: Menghindari fallback dalam sistem terdistribusi
- Amazon Builders' Library: Menghindari backlog antrean yang tidak dapat diatasi
- Amazon Builders' Library: Tantangan dan strategi caching
- Praktik Terbaik untuk Tingkat Web Stateless di AWS

REL05-BP07 Mengimplementasikan tuas darurat

Tuas darurat adalah proses cepat yang dapat memitigasi dampak ketersediaan pada beban kerja.

Tuas darurat bekerja dengan cara menonaktifkan, melakukan throttling, atau mengubah perilaku komponen atau dependensi menggunakan mekanisme yang diketahui dan diuji. Hal ini dapat mengurangi gangguan beban kerja yang disebabkan oleh kelelahan sumber daya karena permintaan yang meningkat secara tidak terduga dan mengurangi dampak kegagalan pada komponen non-kritis dalam beban kerja Anda.

Hasil yang diinginkan: Dengan mengimplementasikan tuas darurat, Anda dapat membuat proses yang telah diketahui dengan baik untuk menjaga ketersediaan komponen kritis dalam beban kerja Anda. Beban kerja akan mengalami degradasi perlahan (graceful degradation) dan terus menjalankan fungsi-fungsi kritis bisnisnya selama aktivasi tuas darurat. Untuk detail lebih lanjut tentang degradasi perlahan, lihat REL05-BP01 Mengimplementasikan degradasi perlahan untuk mengubah dependensi keras yang berlaku menjadi dependensi lunak.

Antipola umum:

- Kegagalan dependensi non-kritis berdampak pada ketersediaan beban kerja inti Anda.
- Tidak menguji atau memverifikasi perilaku komponen kritis selama gangguan komponen non-kritis.
- Tidak ada kriteria yang jelas dan deterministik yang ditentukan untuk pengaktifan atau penonaktifan tuas darurat.

Manfaat menetapkan praktik terbaik ini: Mengimplementasikan tuas darurat dapat meningkatkan ketersediaan komponen kritis dalam beban kerja Anda dengan menyediakan proses yang telah ditetapkan kepada penyedia resolusi untuk merespons lonjakan permintaan yang tidak terduga atau kegagalan dependensi non-kritis.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

- Identifikasi komponen kritis dalam beban kerja Anda.
- Buat agar rancangan dan arsitek komponen kritis dalam beban kerja Anda dapat menahan kegagalan komponen non-kritis.
- Lakukan pengujian untuk memvalidasi perilaku komponen kritis Anda selama kegagalan komponen non-kritis.
- Tentukan dan pantau metrik atau pemicu yang relevan untuk memulai prosedur tuas darurat.
- Tentukan prosedur (manual atau otomatis) yang mencakup tuas darurat.

Langkah implementasi

- Identifikasi komponen kritis bagi bisnis dalam beban kerja Anda.
  - Setiap komponen teknis dalam beban kerja Anda harus dipetakan ke fungsi bisnisnya yang relevan dan diberi peringkat sebagai kritis atau non-kritis. Contoh-contoh fungsionalitas kritis

dan non-kritis di Amazon dapat dilihat di Any Day Can Be Prime Day: How Amazon.com Search Uses Chaos Engineering to Handle Over 84K Requests Per Second.

  - Ini adalah keputusan teknis sekaligus bisnis, dan bervariasi berdasarkan organisasi dan beban kerja.
- Buat agar rancangan dan arsitek komponen kritis dalam beban kerja Anda dapat menahan kegagalan komponen non-kritis.
  - Selama analisis dependensi, pertimbangkan semua mode kegagalan yang dapat terjadi, dan verifikasikan bahwa mekanisme tuas darurat Anda memberikan fungsionalitas kritis pada komponen hilir.
- Lakukan pengujian untuk memvalidasi perilaku komponen kritis Anda saat tuas darurat Anda diaktifkan.
  - Hindari perilaku bimodal. Untuk detail lebih lanjut, lihat REL11-BP05 Menggunakan stabilitas statis untuk mencegah perilaku bimodal.
- Tentukan, pantau, dan munculkan peringatan pada metrik yang relevan untuk memulai prosedur tuas darurat.
  - Beban kerja Anda menentukan metrik yang tepat untuk dipantau. Beberapa contoh metrik adalah latensi atau jumlah permintaan yang gagal ke sebuah dependensi.
- Tentukan prosedur, manual atau otomatis, yang mencakup tuas darurat.
  - Prosedur bisa meliputi mekanisme seperti pelepasan beban, permintaan throttling, atau implementasi degradasi perlahan.

Sumber daya

Praktik Terbaik Terkait:

- REL05-BP01 Mengimplementasikan degradasi yang tepat (graceful degradation) untuk mengubah dependensi keras yang berlaku menjadi dependensi lunak
- REL05-BP02 Membatasi (throttling) permintaan
- REL11-BP05 Menggunakan stabilitas statis untuk mencegah perilaku bimodal

Dokumen terkait:

- Mengotomatiskan deployment aman tanpa campur tangan

- Any Day Can Be Prime Day: How Amazon.com Search Uses Chaos Engineering to Handle Over 84K Requests Per Second

Video terkait:

- AWS re:Invent 2020: Reliability, consistency, and confidence through immutability

Manajemen perubahan

Perubahan beban kerja atau lingkungannya harus diantisipasi dan diakomodasi guna mencapai operasi beban kerja yang andal. Perubahan mencakup semua yang terjadi ke beban kerja seperti lonjakan permintaan, serta perubahan dari dalam, seperti deployment fitur dan patch keamanan.

Praktik terbaik untuk manajemen perubahan dijelaskan dalam bagian berikut.

Topik

Memantau sumber daya beban kerja

Log dan metrik merupakan alat yang sangat andal untuk mendapatkan wawasan tentang kondisi beban kerja. Anda dapat mengonfigurasikan beban kerja untuk memantau log dan metrik serta mengirimkan notifikasi ketika ambang batas terlampaui atau terjadi peristiwa yang signifikan. Dengan pemantauan, beban kerja Anda dapat mengidentifikasi saat ambang batas kinerja rendah terlampaui atau kegagalan terjadi, sehingga dapat bereaksi dengan melakukan pemulihan otomatis.

Pemantauan sangat penting untuk memastikan Anda memenuhi persyaratan ketersediaan. Pemantauan harus mendeteksi kegagalan secara efektif. Mode kegagalan terburuk adalah kegagalan “senyap”, yaitu saat fungsionalitas tidak lagi bekerja, tetapi tidak dapat dideteksi secara langsung. Pelanggan mengetahuinya lebih dulu dari Anda. Salah satu alasan utama pemantauan adalah untuk memperingatkan saat ada masalah. Peringatan harus dipisahkan dari sistem sebanyak mungkin. Jika gangguan layanan menghapus kemampuan untuk memperingatkan, Anda akan mengalami gangguan lebih lama.

Di AWS, kami melengkapi aplikasi pada berbagai tingkat. Kami mencatat latensi, tingkat kesalahan, dan ketersediaan untuk semua permintaan, dependensi, serta operasi utama dalam proses. Kami juga mencatat metrik operasi yang berhasil. Dengan begitu, kami dapat melihat potensi masalah sebelum terjadi. Kami tidak hanya mengamati latensi rata-rata. Kami bahkan lebih fokus pada outlier latensi, seperti persentil 99,9 dan 99,99. Karena jika satu permintaan dari 1.000 atau 10.000 lambat, hal ini masih termasuk pengalaman yang buruk. Selain itu, meski rata-ratanya dapat diterima, jika satu dari 100 permintaan menyebabkan latensi ekstrem, hal ini nantinya dapat menjadi masalah seiring dengan meningkatnya lalu lintas Anda.

Pemantauan di AWS terdiri dari empat fase berbeda:

1. Pembuatan — Memantau semua komponen untuk beban kerja
2. Agregasi — Menentukan dan mengalkulasi metrik
3. Pemberian peringatan dan pemrosesan waktu nyata — Mengirimkan notifikasi dan mengotomatiskan respons
4. Penyimpanan dan Analitik

Praktik terbaik

- REL06-BP01 Memantau semua komponen untuk beban kerja (Pembuatan)
- REL06-BP02 Menetapkan dan menghitung metrik (Agregasi)
- REL06-BP03 Mengirimkan notifikasi (Pemrosesan dan pembuatan alarm waktu nyata)
- REL06-BP04 Mengotomatiskan respons (Peringatan dan pemrosesan waktu nyata)
- REL06-BP05 Analitik
- REL06-BP06 Lakukan peninjauan secara teratur
- REL06-BP07 Memantau pelacakan permintaan menyeluruh melalui sistem Anda

REL06-BP01 Memantau semua komponen untuk beban kerja (Pembuatan)

Pantau komponen beban kerja dengan Amazon CloudWatch atau alat pihak ketiga. Pantau layanan AWS dengan Dasbor AWS Health.

Semua komponen beban kerja Anda harus dipantau, mencakup front-end, logika bisnis, dan tingkat penyimpanan. Tetapkan metrik utama, jelaskan cara mengekstraknya dari log (jika diperlukan), dan tetapkan ambang batas untuk memicu peristiwa alarm yang sesuai. Pastikan metrik relevan dengan indikator kinerja utama (KPI) beban kerja Anda, dan gunakan metrik dan log untuk mengidentifikasi tanda-tanda peringatan dini penurunan layanan. Contohnya, metrik yang terkait dengan hasil bisnis seperti jumlah pesanan yang berhasil diproses per menit, dapat menunjukkan masalah beban kerja lebih cepat dari metrik teknis, seperti Pemanfaatan CPU. Gunakan Dasbor AWS Health untuk tampilan yang dipersonalisasi tentang kinerja dan ketersediaan layanan AWS yang mendasari sumber daya AWS Anda.

Pemantauan di cloud menawarkan peluang baru. Sebagian besar penyedia cloud telah mengembangkan hook yang dapat disesuaikan dan dapat menghadirkan wawasan untuk membantu

Anda memantau beberapa lapisan beban kerja Anda. Layanan AWS seperti Amazon CloudWatch menerapkan algoritme statis dan machine learning untuk terus menganalisis metrik sistem dan aplikasi, menentukan garis dasar normal dan anomali permukaan dengan sedikit campur tangan pengguna. Algoritme deteksi anomali memperhitungkan perubahan musiman dan tren metrik.

AWS menyediakan banyak informasi pemantauan dan log untuk digunakan untuk menentukan metrik khusus beban kerja, proses perubahan permintaan, dan mengadopsi teknik machine learning, terlepas dari keahlian ML.

Selain itu, pantau semua titik akhir eksternal Anda untuk memastikan independensinya dari implementasi dasar Anda. Pemantauan aktif ini dapat dilakukan dengan transaksi sintetis (kadang disebut sebagai canary pengguna, tetapi bedakan dengan deployment canary) yang secara berkala menjalankan sejumlah tugas umum yang sesuai dengan tindakan yang dilakukan oleh klien beban kerja. Buat tugas-tugas ini berdurasi singkat dan pastikan untuk tidak membebani beban kerja Anda saat pengujian. Amazon CloudWatch Synthetics memungkinkan Anda untuk membuat canary sintetis untuk memantau titik akhir dan API Anda. Anda juga dapat menggabungkan simpul klien canary sintetis dengan konsol AWS X-Ray untuk mengidentifikasi canary sintetis mana yang mengalami masalah berupa error, fault, atau tingkat throttling untuk jangka waktu yang dipilih.

Hasil yang Diharapkan:

Kumpulkan dan gunakan metrik kritis dari semua komponen beban kerja untuk memastikan keandalan beban kerja dan pengalaman pengguna yang optimal. Dengan mendeteksi bahwa beban kerja tidak mencapai hasil bisnis, Anda dapat dengan cepat mengumumkan bencana dan pulih dari insiden.

Antipola umum:

- Hanya memantau antarmuka eksternal beban kerja Anda.
- Tidak menghasilkan metrik khusus beban kerja dan hanya bergantung pada metrik yang diberikan kepada Anda oleh layanan AWS yang digunakan oleh beban kerja Anda.
- Hanya menggunakan metrik teknis di beban kerja Anda dan tidak memantau metrik apa pun yang terkait dengan KPI non-teknis yang menerima kontribusi dari beban kerja Anda.
- Mengandalkan lalu lintas produksi dan pemeriksaan kondisi sederhana untuk memantau dan mengevaluasi state beban kerja.

Manfaat menjalankan praktik terbaik ini: Dengan memantau semua tingkatan di beban kerja Anda, Anda dapat lebih cepat mengantisipasi dan menyelesaikan masalah di komponen dalam beban kerja.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

1. Mengaktifkan pencatatan log jika tersedia. Data pemantauan harus diperoleh dari semua komponen beban kerja. Aktifkan pencatatan log tambahan, seperti S3 Access Logs, dan aktifkan beban kerja Anda untuk mencatat log data spesifik beban kerja. Kumpulkan metrik rata-rata CPU, I/O jaringan, dan I/O disk dari layanan seperti Amazon ECS, Amazon EKS, Amazon EC2, Elastic Load Balancing, AWS Auto Scaling, dan Amazon EMR. Lihat Layanan AWS yang Memublikasikan Metrik CloudWatch untuk daftar layanan AWS yang memublikasikan metrik ke CloudWatch.
2. Tinjau semua metrik default dan telusuri celah pengumpulan data apa pun. Setiap layanan menghasilkan metrik default. Dengan mengumpulkan metrik default, Anda dapat lebih memahami dependensi antar komponen beban kerja dan bagaimana keandalan dan kinerja komponen memengaruhi beban kerja. Anda juga dapat membuat dan memublikasikan metrik Anda sendiri ke CloudWatch menggunakan AWS CLI atau API. Ini
3. Evaluasi semua metrik untuk menentukan mana yang harus dibuatkan peringatan untuk setiap layanan AWS di beban kerja Anda. Anda dapat memilih subset metrik yang memiliki dampak besar dalam keandalan beban kerja. Berfokus pada metrik dan ambang batas memungkinkan Anda untuk menyempurnakan jumlah pemberitahuan dan dapat membantu meminimalkan positif palsu.
4. Tetapkan peringatan dan proses pemulihan beban kerja Anda setelah peringatan dipicu. Dengan menetapkan peringatan, Anda dapat dengan cepat memberi tahu, mengeskalasi, dan mengikuti langkah-langkah yang diperlukan untuk pemulihan dari insiden dan memenuhi Sasaran Waktu Pemulihan (RTO) yang Anda tentukan. Anda dapat menggunakan Alarm Amazon CloudWatch untuk memanggil alur kerja otomatis dan memulai prosedur pemulihan berdasarkan ambang batas yang ditentukan.
5. Jelajahi penggunaan transaksi sintetis untuk mengumpulkan data yang relevan tentang state beban kerja. Pemantauan sintetis mengikuti rute yang sama dan menjalankan tindakan yang sama seperti pelanggan, sehingga memungkinkan Anda untuk terus memverifikasi pengalaman pelanggan Anda bahkan saat Anda tidak memiliki lalu lintas pelanggan apa pun pada beban kerja Anda. Dengan menggunakan transaksi sintetis, Anda dapat menemukan masalah sebelum pelanggan Anda menemukannya.

Sumber daya

Praktik Terbaik Terkait:

- REL11-BP03 Mengotomatisasi pemulihan di semua lapisan

Dokumen terkait:

- Memulai Dasbor AWS Health Anda – Kondisi akun Anda
- Layanan AWS yang Memublikasikan Metrik CloudWatch
- Mengakses Log untuk Penyeimbang Beban Jaringan Anda
- Akes log untuk penyeimbang beban aplikasi Anda
- Mengakses Amazon CloudWatch Logs untuk AWS Lambda
- Pencatatan Log Akses Server S3
- Mengaktifkan Log Akses untuk Penyeimbang Beban Klasik Anda
- Mengekspor data log ke Amazon S3
- Menginstal agen CloudWatch di instans Amazon EC2
- Memublikasikan Metrik Kustom
- Menggunakan Dasbor Amazon CloudWatch
- Menggunakan Metrik Amazon CloudWatch
- Menggunakan Canary (Amazon CloudWatch Synthetics)
- Apa itu Amazon CloudWatch Logs?

Panduan pengguna:

- Membuat trail
- Memantau metrik memori dan disk untuk instans Linux Amazon EC2
- Menggunakan CloudWatch Logs dengan instans kontainer
- VPC Flow Logs
- Apa itu Amazon DevOps Guru?
- Apa itu AWS X-Ray?

Blog terkait:

- Melakukan debug dengan Amazon CloudWatch Synthetics dan AWS X-Ray

Contoh dan lokakarya terkait:

- AWS Well-Architected Labs: Keunggulan Operasional - Pemantauan Dependensi
- Amazon Builders' Library: Instrumentasi sistem terdistribusi untuk visibilitas operasional

- Lokakarya observabilitas

REL06-BP02 Menetapkan dan menghitung metrik (Agregasi)

Simpan data log dan terapkan filter saat diperlukan untuk menghitung metrik, seperti jumlah log peristiwa tertentu, atau latensi yang dihitung dari stempel waktu log peristiwa.

Amazon CloudWatch dan Amazon S3 berfungsi sebagai lapisan agregasi dan penyimpanan utama. Untuk beberapa layanan, seperti AWS Auto Scaling dan Elastic Load Balancing, metrik default disediakan secara default untuk beban CPU atau latensi permintaan rata-rata di seluruh klaster atau instans. Untuk layanan streaming, seperti VPC Flow Logs dan AWS CloudTrail, data peristiwa diteruskan ke CloudWatch Logs dan Anda perlu menetapkan dan menerapkan filter metrik untuk mengekstrak metrik dari data peristiwa. Ini memberi Anda data rangkaian waktu, yang dapat berfungsi sebagai input ke alarm CloudWatch yang Anda tetapkan untuk memicu peringatan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Tinggi

Panduan implementasi

- Tetapkan dan hitung metrik (Agregasi). Simpan data log dan terapkan filter saat diperlukan untuk menghitung metrik, seperti jumlah log peristiwa tertentu, atau latensi yang dihitung dari stempel waktu log peristiwa.
  - Filter metrik menetapkan ketentuan dan pola yang harus dicari di data log saat dikirim ke CloudWatch Logs. CloudWatch Logs menggunakan filter metrik untuk mengubah data log menjadi metrik CloudWatch numerik yang dapat Anda buatkan grafik atau aktifkan alarm.
    - Mencari dan Menyaring Data Log
  - Gunakan pihak ketiga tepercaya untuk mengagregasi log.
    - Ikuti instruksi pihak ketiga. Sebagian besar produk pihak ketiga terintegrasi dengan CloudWatch dan Amazon S3.
  - Beberapa layanan AWS dapat memublikasikan log langsung ke Amazon S3. Jika kebutuhan utama Anda untuk log adalah penyimpanan di Amazon S3, Anda dapat dengan mudah meminta layanan yang memproduksi log untuk mengirimnya langsung ke Amazon S3 tanpa mengatur infrastruktur tambahan.
    - Mengirim Log Langsung ke Amazon S3

Sumber daya

Dokumen terkait:

- Contoh Kueri Amazon CloudWatch Logs Insight
- Melakukan debug dengan Amazon CloudWatch Synthetics dan AWS X-Ray
- One Observability Workshop
- Mencari dan Menyaring Data Log
- Mengirim Log Langsung ke Amazon S3
- Amazon Builders' Library: Menginstrumentasi sistem terdistribusi untuk visibilitas operasional

REL06-BP03 Mengirimkan notifikasi (Pemrosesan dan pembuatan alarm waktu nyata)

Ketika organisasi mendeteksi potensi masalah, mereka mengirimkan notifikasi dan peringatan waktu nyata kepada personel dan sistem yang sesuai untuk merespons masalah ini dengan cepat dan efektif.

Hasil yang diinginkan: Respons cepat terhadap event operasional dapat terjadi melalui konfigurasi alarm yang relevan berdasarkan metrik layanan dan aplikasi. Ketika ambang batas alarm dilanggar, personel dan sistem yang sesuai diberitahu sehingga mereka dapat mengatasi masalah mendasar.

Antipola umum:

- Mengonfigurasi alarm dengan ambang batas yang terlalu tinggi, sehingga mengakibatkan kegagalan untuk mengirim notifikasi penting.
- Mengonfigurasi alarm dengan ambang batas yang terlalu rendah, yang menyebabkan tidak adanya tindakan atas pemberitahuan penting karena kebisingan notifikasi yang berlebihan.
- Tidak memperbarui alarm dan ambang batasnya saat penggunaan berubah.
- Untuk alarm yang paling sesuai untuk ditangani melalui tindakan otomatis, mengirim notifikasi ke personel alih-alih menghasilkan tindakan otomatis, menyebabkan terkirimnya notifikasi yang berlebihan.

Manfaat menjalankan praktik terbaik ini: Mengirimkan notifikasi dan pemberitahuan waktu nyata kepada personel dan sistem yang sesuai memungkinkan deteksi dini masalah dan respons cepat terhadap insiden operasional.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Beban kerja harus dilengkapi dengan pemrosesan dan pembuatan alarm waktu nyata untuk meningkatkan pendeteksian masalah yang dapat memengaruhi ketersediaan aplikasi dan berfungsi sebagai pemicu respons otomatis. Organisasi dapat melakukan pemrosesan dan pembuatan alarm waktu nyata dengan menciptakan peringatan dengan metrik yang ditentukan untuk menerima notifikasi setiap kali peristiwa signifikan terjadi atau sebuah metrik melebihi ambang batas.

Amazon CloudWatch memungkinkan Anda untuk membuat metrik dan alarm komposit menggunakan alarm CloudWatch berdasarkan ambang batas statis, deteksi anomali, dan kriteria lainnya. Untuk detail selengkapnya tentang jenis alarm yang dapat Anda konfigurasikan menggunakan CloudWatch, lihat bagian alarm dalam dokumentasi CloudWatch.

Anda dapat membuat tampilan metrik dan pemberitahuan yang disesuaikan dari sumber daya AWS Anda untuk tim Anda menggunakan dasbor CloudWatch. Halaman beranda yang dapat disesuaikan di konsol CloudWatch memungkinkan Anda memantau sumber daya dalam satu tampilan di beberapa Region.

Alarm dapat melakukan satu atau beberapa tindakan, seperti mengirimkan notifikasi ke topik Amazon SNS, melakukan tindakan Amazon EC2 atau tindakan Amazon EC2 Auto Scaling , atau membuat OpsItem atau insiden di AWS Systems Manager.

Amazon CloudWatch menggunakan Amazon SNS untuk mengirimkan notifikasi ketika alarm berubah status, sehingga memberikan pengiriman pesan dari penerbit (produsen) ke pelanggan (konsumen). Untuk detail selengkapnya tentang pengaturan notifikasi Amazon SNS, lihat Mengonfigurasi Amazon SNS.

CloudWatch mengirim event EventBridge setiap kali alarm CloudWatch dibuat, diperbarui, dihapus, atau statusnya berubah. Anda dapat menggunakan EventBridge dengan event ini untuk membuat aturan yang melakukan tindakan, seperti memberi tahu Anda setiap kali status alarm berubah atau secara otomatis memicu event di akun Anda menggunakan Otomatisasi Systems Manager.

Kapan Anda harus menggunakan EventBridge atau Amazon SNS?

Baik EventBridge maupun Amazon SNS dapat digunakan untuk mengembangkan aplikasi berbasis event, dan pilihan Anda akan tergantung pada kebutuhan spesifik Anda.

Amazon EventBridge direkomendasikan ketika Anda ingin membangun aplikasi yang bereaksi terhadap event dari aplikasi Anda sendiri, aplikasi SaaS, dan layanan AWS. EventBridge adalah

satu-satunya layanan berbasis event yang terintegrasi langsung dengan partner SaaS pihak ketiga. EventBridge juga secara otomatis menyerap peristiwa dari lebih dari 200 layanan AWS tanpa mengharuskan developer untuk membuat sumber daya apa pun di akun mereka.

EventBridge menggunakan struktur berbasis JSON yang ditentukan untuk event, dan membantu Anda membuat aturan yang diterapkan di seluruh badan event untuk memilih event untuk diteruskan ke target. EventBridge saat ini mendukung lebih dari 20 layanan AWS sebagai target, termasuk AWS Lambda, Amazon SQS, Amazon SNS, Amazon Kinesis Data Streams, dan Amazon Data Firehose.

Amazon SNS direkomendasikan untuk aplikasi yang membutuhkan fan out tinggi (ribuan atau jutaan titik akhir). Pola umum yang kita lihat adalah bahwa pelanggan menggunakan Amazon SNS sebagai target aturan mereka untuk memfilter event yang mereka butuhkan dan fan out ke beberapa titik akhir.

Pesan memiliki sifat yang tidak terstruktur dan bisa dalam format apa pun. Amazon SNS mendukung penerusan pesan ke enam jenis target yang berbeda, termasuk Lambda, Amazon SQS, titik akhir HTTP/S, SMS, push seluler, dan email. Amazon SNS latensi tipikal di bawah 30 milidetik. Berbagai layanan AWS mengirimkan pesan Amazon SNS dengan mengonfigurasi layanan untuk melakukannya (lebih dari 30, termasuk Amazon EC2, Amazon S3, dan Amazon RDS).

Langkah implementasi

1. Buat alarm menggunakan alarm Amazon CloudWatch.
    a. Alarm metrik memonitor metrik CloudWatch tunggal atau ekspresi yang bergantung pada metrik CloudWatch. Alarm memulai satu atau beberapa tindakan berdasarkan nilai metrik atau ekspresi dibandingkan dengan ambang batas selama interval waktu tertentu. Tindakan tersebut dapat terdiri dari pengiriman notifikasi ke topik Amazon SNS, melakukan tindakan Amazon EC2 atau tindakan Amazon EC2 Auto Scaling , atau membuat OpsItem atau acara di AWS Systems Manager.
    b. Alarm komposit terdiri dari ekspresi aturan yang mempertimbangkan kondisi alarm dari alarm-alarm lain yang telah Anda buat. Alarm komposit hanya memasuki status alarm jika semua kondisi aturan terpenuhi. Alarm yang ditentukan dalam ekspresi aturan suatu alarm komposit dapat mencakup alarm metrik dan alarm komposit tambahan. Alarm komposit dapat mengirim notifikasi Amazon SNS ketika statusnya berubah dan dapat membuat Systems Manager OpsItems atau insiden ketika memasuki keadaan alarm, tetapi tidak dapat melakukan tindakan Amazon EC2 atau Auto Scaling.
2. Siapkan notifikasi Amazon SNS. Saat membuat alarm CloudWatch, Anda dapat menyertakan sebuah topik Amazon SNS untuk mengirimkan notifikasi saat status alarm berubah.

3. Buat aturan di EventBridge yang cocok dengan alarm CloudWatch yang ditentukan. Setiap aturan mendukung beberapa target, termasuk fungsi Lambda. Misalnya, Anda dapat menentukan alarm yang dimulai saat ruang disk yang tersedia hampir habis, yang memicu sebuah fungsi Lambda melalui aturan EventBridge, untuk mengosongkan ruang. Untuk detail selengkapnya tentang target EventBridge, lihat Target EventBridge.

Sumber daya

Praktik terbaik Well-Architected terkait:

- REL06-BP01 Memantau semua komponen untuk beban kerja (Pembuatan)
- REL06-BP02 Menetapkan dan menghitung metrik (Agregasi)
- REL12-BP01 Menggunakan buku pedoman untuk menyelidiki kegagalan

Dokumen terkait:

- Amazon CloudWatch
- Wawasan CloudWatch Logs
- Menggunakan alarm Amazon CloudWatch
- Menggunakan dasbor Amazon CloudWatch
- Menggunakan metrik Amazon CloudWatch
- Menyiapkan notifikasi Amazon SNS
- deteksi anomali CloudWatch
- Perlindungan data CloudWatch Logs
- Amazon EventBridge
- Amazon Simple Notification Service

Video terkait:

- video observabilitas reinvent 2022
- AWS re:Invent 2022 - Praktik terbaik observabilitas di Amazon

Contoh terkait:

- Lokakarya One Observability
- Amazon EventBridge ke AWS Lambda dengan kontrol umpan balik oleh Alarm Amazon CloudWatch

REL06-BP04 Mengotomatiskan respons (Peringatan dan pemrosesan waktu nyata)

Gunakan otomatisasi untuk melakukan tindakan ketika peristiwa terdeteksi, misalnya, mengganti komponen yang rusak.

Pemrosesan alarm waktu nyata secara otomatis diimplementasikan agar sistem dapat mengambil tindakan korektif yang cepat dan berupaya mencegah kegagalan atau penurunan layanan ketika alarm terpicu. Respons otomatis terhadap alarm dapat mencakup penggantian komponen yang gagal, penyesuaian kapasitas komputasi, pengalihan lalu lintas ke host yang sehat, zona ketersediaan, atau wilayah lain, dan pemberitahuan operator.

Hasil yang diinginkan: Alarm waktu nyata diidentifikasi, dan pemrosesan alarm secara otomatis diatur untuk menginvokasi tindakan yang tepat yang diambil untuk mempertahankan tujuan tingkat layanan dan perjanjian tingkat layanan (SLA). Otomatisasi dapat berupa berbagai hal, dari aktivitas pemulihan diri sebuah komponen hingga failover seluruh situs.

Antipola umum:

- Tidak memiliki inventaris atau katalog alarm waktu nyata utama yang jelas.
- Tidak ada respons otomatis terhadap alarm kritis (misalnya, penskalaan otomatis berjalan ketika komputasi hampir habis).
- Tindakan respons alarm yang kontradiktif.
- Tidak ada prosedur operasi standar (SOP) untuk diikuti operator ketika mereka menerima pemberitahuan peringatan.
- Tidak memantau perubahan konfigurasi, padahal perubahan konfigurasi yang tidak terdeteksi dapat menyebabkan waktu henti untuk beban kerja.
- Tidak memiliki strategi untuk membatalkan perubahan konfigurasi yang tidak diinginkan.

Manfaat menetapkan praktik terbaik ini: Mengotomatiskan pemrosesan alarm dapat meningkatkan ketahanan sistem. Sistem mengambil tindakan korektif secara otomatis, sehingga mengurangi

aktivitas manual yang memberi peluang adanya intervensi manusia yang rawan kesalahan. Operasi beban kerja memenuhi tujuan ketersediaan, dan mengurangi gangguan layanan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Untuk mengelola peringatan secara efektif dan mengotomatiskan responsnya, kategorikan peringatan berdasarkan tingkat kekritisan dan dampaknya, dokumentasikan prosedur respons, dan rencanakan respons sebelum menentukan peringkat tugas.

Identifikasi tugas yang membutuhkan tindakan tertentu (sering kali diperinci dalam runbook), dan periksa semua runbook dan playbook untuk menentukan tugas mana yang dapat diotomatisasi. Jika tindakan dapat digambarkan dengan jelas, tindakan tersebut sering kali dapat diotomatisasi. Jika tindakan tidak dapat diotomatisasi, dokumentasikan langkah-langkah manual dalam SOP dan latih operator untuk melakukannya. Terus cari peluang otomatisasi pada proses manual agar Anda dapat membuat dan menerapkan rencana untuk mengotomatiskan respons peringatan.

Langkah implementasi

1. Buat inventaris alarm: Untuk mendapatkan daftar semua alarm, Anda dapat memanfaatkan AWS CLI menggunakan perintah Amazon CloudWatch`describe-alarms`. Bergantung pada berapa banyak alarm yang telah Anda siapkan, Anda mungkin harus menggunakan paginasi untuk mengambil subset alarm untuk setiap panggilan, atau menggunakan SDK AWS untuk mendapatkan alarm menggunakan panggilan API.
2. Dokumentasikan semua tindakan alarm: Perbarui runbook dengan semua alarm dan tindakannya, baik itu manual maupun otomatis. AWS Systems Manager menyediakan runbook yang sudah ditetapkan sebelumnya. Untuk informasi selengkapnya tentang runbook, lihat Working with runbooks. Untuk detail tentang cara melihat konten runbook, lihat View runbook content.
3. Menyiapkan dan mengelola tindakan alarm: Untuk alarm apa pun yang memerlukan tindakan, tentukan tindakan otomatis menggunakan SDK CloudWatch. Misalnya, Anda dapat mengubah status instans Amazon EC2 secara otomatis berdasarkan alarm CloudWatch dengan membuat dan mengaktifkan tindakan pada alarm atau menonaktifkan tindakan pada alarm.

   Anda juga dapat menggunakan Amazon EventBridge untuk merespons peristiwa sistem secara otomatis, seperti masalah ketersediaan aplikasi atau perubahan sumber daya. Anda dapat membuat aturan untuk menunjukkan peristiwa mana yang perlu diperhatikan, dan tindakan yang harus diambil apabila peristiwa cocok dengan sebuah aturan. Tindakan yang dapat dimulai secara otomatis termasuk menginvokasi fungsi AWS Lambda, menginvokasi Amazon EC2 Run

`Command`, merelai peristiwa tersebut ke Amazon Kinesis Data Streams, dan melihat Otomatiskan Amazon EC2 menggunakan EventBridge.

4. Prosedur Operasi Standar (SOP): Berdasarkan komponen aplikasi Anda, AWS Resilience Hub merekomendasikan beberapa templat SOP. Anda dapat menggunakan SOP ini untuk mendokumentasikan semua proses yang harus diikuti operator jika peringatan muncul. Anda juga dapat menyusun SOP berdasarkan rekomendasi Resilience Hub, dan untuk ini Anda memerlukan aplikasi Resilience Hub dengan kebijakan ketahanan terkait, serta penilaian ketahanan historis terhadap aplikasi tersebut. Rekomendasi untuk SOP Anda berasal dari penilaian ketahanan.

   Resilience Hub bekerja dengan Systems Manager untuk mengotomatiskan langkah-langkah dalam SOP Anda dengan menyediakan sejumlah dokumen SSM yang dapat Anda gunakan sebagai dasar untuk SOP tersebut. Misalnya, Resilience Hub mungkin merekomendasikan SOP untuk menambahkan ruang disk berdasarkan dokumen otomatisasi SSM yang sudah ada.
5. Lakukan tindakan otomatis menggunakan Amazon DevOps Guru: Anda dapat menggunakan Amazon DevOps Guru untuk secara otomatis memantau perilaku anomali pada sumber daya aplikasi, serta memberikan rekomendasi terarah untuk mempercepat waktu perbaikan serta identifikasi masalah. DenganDevOps Guru, Anda dapat memantau aliran data operasional hampir secara waktu nyata dari berbagai sumber termasuk metrik Amazon CloudWatch, AWS Config, AWS CloudFormation, dan AWS X-Ray. Anda juga dapat menggunakan DevOps Guru untuk secara otomatis membuat OpsItems di OpsCenter dan mengirim peristiwa ke EventBridge untuk otomatisasi tambahan.

Sumber daya

Praktik Terbaik Terkait:

- REL06-BP01 Memantau semua komponen untuk beban kerja (Pembuatan)
- REL06-BP02 Menetapkan dan menghitung metrik (Agregasi)
- REL06-BP03 Mengirimkan notifikasi (Pemrosesan dan pembuatan alarm waktu nyata)
- REL08-BP01 Menggunakan runbook untuk aktivitas standar seperti deployment

Dokumen terkait:

- Otomatisasi AWS Systems Manager
- Membuat Aturan EventBridge yang Memicu Peristiwa dari Sumber Daya AWS
- Lokakarya One Observability

- Amazon Builders' Library: Instrumentasi sistem terdistribusi untuk visibilitas operasional
- Apa itu Amazon DevOps Guru?
- Bekerja dengan Dokumen Otomatisasi (Buku Pedoman)

Video terkait:

- AWS re:Invent 2022 - Praktik terbaik observabilitas di Amazon
- AWS re:Invent 2020: Automate anything with AWS Systems Manager
- Introduction to AWS Resilience Hub
- Create Custom Ticket Systems for Amazon DevOps Guru Notifications
- Enable Multi-Account Insight Aggregation with Amazon DevOps Guru

Contoh terkait:

- Lokakarya Keandalan
- Amazon CloudWatch and Systems Manager Workshop

REL06-BP05 Analitik

Kumpulkan riwayat metrik dan file log dan analisis ini untuk mendapatkan wawasan beban kerja dan tren lebih luas.

Wawasan Amazon CloudWatch Logs mendukung bahasa kueri sederhana tapi luar biasa yang dapat Anda gunakan untuk menganalisis data log. Amazon CloudWatch Logs juga mendukung langganan yang memungkinkan data mengalir dengan lancar ke Amazon S3 di mana Anda dapat menggunakannya atau Amazon Athena untuk kueri data. Amazon CloudWatch Logs juga mendukung kueri dengan berbagai macam format. Lihat Format Data dan SerDes yang didukung di Panduan Pengguna Amazon Athena untuk informasi selengkapnya. Untuk analisis set file log yang sangat besar, Anda dapat menjalankan klaster Amazon EMR untuk melakukan analisis skala petabita.

Ada sejumlah alat yang disediakan oleh Partner AWS dan pihak ketiga yang memungkinkan agregasi, pemrosesan, penyimpanan, dan analitik. Alat ini antara lain yakni New Relic, Splunk, Loggly, Logstash, CloudHealth, dan Nagios. Tetapi, generasi luar sistem dan log aplikasi bersifat unik untuk setiap penyedia cloud, dan sering kali unik untuk setiap layanan.

Bagian proses pemantauan yang sering kali tidak diperhatikan adalah manajemen data. Anda harus menentukan persyaratan retensi untuk memantau data, kemudian terapkan kebijakan siklus hidup yang sesuai. Amazon S3 mendukung manajemen siklus hidup di tingkat bucket S3. Manajemen siklus hidup ini dapat diterapkan secara berbeda ke jalur yang berbeda di bucket. Menjelang akhir siklus hidup, Anda dapat melakukan transisi data ke Amazon S3 Glacier untuk penyimpanan jangka panjang, kemudian kedaluwarsa setelah akhir jangka waktu retensi tercapai. Kelas penyimpanan Bertingkat Cerdas S3 didesain untuk mengoptimalkan biaya dengan secara otomatis memindahkan data ke tingkat akses yang paling hemat biaya, tanpa memengaruhi performa atau tambahan biaya operasional.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Sedang

Panduan implementasi

- Wawasan CloudWatch Logs memampukan Anda untuk secara interaktif mencari dan menganalisis data log Anda di Amazon CloudWatch Logs.
  - Menganalisis Data Log dengan Wawasan CloudWatch Logs
  - Kueri Sampel Wawasan Amazon CloudWatch Logs
- Gunakan Amazon CloudWatch Logs untuk mengirimkan log ke Amazon S3 di mana Anda dapat menggunakannya atau Amazon Athena untuk kueri data.
  - Bagaimana cara menganalisis log akses server Amazon S3 menggunakan Athena?
    - Buat kebijakan siklus hidup S3 untuk bucket log akses server Anda. Konfigurasikan kebijakan siklus hidup untuk secara berkala menghapus file log. Dengan melakukan tindakan ini, maka jumlah data yang dianalisis Athena untuk setiap kueri akan berkurang.
      - Bagaimana Cara Membuat Kebijakan Siklus Hidup untuk Bucket S3?

Sumber daya

Dokumen terkait:

- Kueri Sampel Wawasan Amazon CloudWatch Logs
- Menganalisis Data Log dengan Wawasan CloudWatch Logs
- Debugging dengan Amazon CloudWatch Synthetics and AWS X-Ray
- Bagaimana Cara Membuat Kebijakan Siklus Hidup untuk Bucket S3?
- Bagaimana cara menganalisis log akses server Amazon S3 menggunakan Athena?

- Satu Lokakarya Pengamatan
- Amazon Builders' Library: Menginstrumentasi sistem terdistribusi untuk visibilitas operasional

REL06-BP06 Lakukan peninjauan secara teratur

Sering kali tinjau bagaimana pemantauan beban kerja diimplementasikan dan perbarui berdasarkan perubahan dan peristiwa yang signifikan.

Pemantauan yang efektif didorong oleh metrik bisnis utama. Pastikan metrik-metrik ini diakomodasi di beban kerja Anda seiring dengan perubahan prioritas bisnis.

Mengaudit pemantauan Anda akan membantu memastikan Anda tahu kapan aplikasi memenuhi sasaran ketersediaannya. Analisis akar masalah memerlukan kemampuan untuk menemukan apa yang telah terjadi ketika ada kegagalan. AWS memberikan layanan yang memungkinkan Anda untuk melacak keadaan layanan Anda selama insiden:

- Amazon CloudWatch Logs: Anda dapat menyimpan log Anda di dalam layanan ini dan memeriksa kontennya.
- Wawasan Amazon CloudWatch Logs: Adalah layanan terkelola penuh yang memampukan Anda untuk menganalisis log yang sangat besar dalam hitungan detik. Layanan ini memberikan kepada Anda visualisasi dan kueri cepat dan interaktif.
- AWS Config: Anda dapat melihat infrastruktur AWS apa yang digunakan di berbagai titik waktu.
- AWS CloudTrail: Anda dapat melihat API AWS mana yang dipanggil pada waktu apa dan oleh prinsipal apa.

Di AWS, kami mengadakan rapat mingguan untuk meninjau performa operasional dan untuk berbagi pembelajaran antara tim. Karena ada begitu banyak tim di AWS, kami menciptakan Roda (The Wheel) untuk secara acak memilih beban kerja yang akan ditinjau. Menetapkan irama yang teratur untuk peninjauan performa operasional dan berbagi pengetahuan meningkatkan kemampuan Anda untuk mencapai performa lebih tinggi dari tim operasional Anda.

Antipola umum:

- Hanya mengumpulkan metrik default.
- Menetapkan strategi pemantauan dan tidak pernah meninjaunya.
- Tidak membahas pemantauan ketika ada deployment perubahan besar.

Manfaat menerapkan praktik terbaik ini: Secara teratur meninjau pemantauan Anda memampukan antisipasi potensi masalah, dan bukannya bereaksi terhadap notifikasi ketika masalah yang diantisipasi sesungguhnya terjadi.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Sedang

Panduan implementasi

- Buat beberapa dasbor untuk beban kerja. Anda harus memiliki dasbor tingkat teratas yang berisi metrik bisnis utama, serta metrik teknis yang telah Anda identifikasi sebagai paling relevan untuk kondisi beban kerja yang diproyeksikan sesuai penggunaan yang bervariasi. Anda juga harus memiliki dasbor yang dapat diinspeksi untuk berbagai tingkat aplikasi dan ketergantungan.
  - Menggunakan Dasbor Amazon CloudWatch
- Jadwalkan dan lakukan peninjauan dasbor beban kerja secara teratur. Lakukan inspeksi dasbor secara teratur. Anda mungkin memiliki irama yang berbeda untuk kedalaman inspeksi Anda.
  - Inspeksi apakah ada tren dalam metrik. Bandingkan nilai metrik dengan nilai historis untuk melihat apakah ada tren yang mungkin menandakan bahwa sesuatu perlu diselidiki. Contohnya antara lain: meningkatkan latensi, menurunkan fungsi bisnis utama, dan meningkatkan respons kegagalan.
  - Inspeksi apakah ada penyimpangan/anomali dalam metrik Anda. Rerata atau median dapat menutupi penyimpangan dan anomali. Lihat nilai tertinggi dan nilai terendah dalam kerangka waktu dan selidiki penyebab skor yang ekstrem. Saat Anda terus mengeliminasi penyebab-penyebab ini, menurunkan definisi ekstrem akan memungkinkan Anda untuk terus meningkatkan konsistensi performa beban kerja Anda.
  - Cari perubahan mendadak dalam perilaku. Perubahan cepat dalam jumlah atau arah metrik dapat menandakan telah ada perubahan dalam aplikasi, atau ada faktor eksternal yang mungkin perlu Anda tambahkan metrik tambahan untuk dilacak.

Sumber daya

Dokumen terkait:

- Kueri Sampel Wawasan Amazon CloudWatch Logs
- Debugging dengan Amazon CloudWatch Synthetics and AWS X-Ray
- Satu Lokakarya Pengamatan
- Amazon Builders' Library: Menginstrumentasi sistem terdistribusi untuk visibilitas operasional

- Menggunakan Dasbor Amazon CloudWatch

REL06-BP07 Memantau pelacakan permintaan menyeluruh melalui sistem Anda

Lacak permintaan yang sedang diproses melalui komponen layanan agar tim produk dapat lebih mudah menganalisis dan menemukan serta memperbaiki masalah dan meningkatkan kinerja.

Hasil yang diinginkan: Beban kerja dengan penelusuran yang komprehensif di semua komponen memudahkan pencarian dan perbaikan masalah, sehingga meningkatkan rata-rata waktu penyelesaian (MTTR) kesalahan dan latensi dengan menyederhanakan penemuan akar masalah. Penelusuran yang menyeluruh akan mempersingkat waktu yang diperlukan untuk menemukan komponen yang terdampak dan mencari tahu akar masalah kesalahan atau latensi secara mendetail.

Antipola umum:

- Penelusuran digunakan untuk beberapa komponen, tidak semuanya. Misalnya, tanpa penelusuran untuk AWS Lambda, tim mungkin tidak memahami dengan jelas latensi yang disebabkan oleh cold start dalam beban kerja fluktuatif.
- Canary sintetis atau pemantauan pengguna nyata (RUM) tidak dikonfigurasi dengan penelusuran. Tanpa canary atau RUM, telemetri interaksi klien dihilangkan dari analisis jejak yang berimbas pada profil kinerja yang tidak lengkap.
- Beban kerja hybrid mencakup alat penelusuran cloud native dan pihak ketiga, tetapi langkah-langkah belum dilakukan untuk memilih dan sepenuhnya mengintegrasikan solusi penelusuran tunggal. Berdasarkan solusi penelusuran yang dipilih, SDK penelusuran cloud-native harus digunakan untuk melengkapi instrumen yang bukan cloud-native, atau alat pihak ketiga harus dikonfigurasi untuk menyerap telemetri pelacakan cloud-native.

Manfaat menjalankan praktik terbaik ini: Saat tim pengembangan menerima peringatan masalah, mereka dapat melihat gambaran utuh tentang interaksi komponen sistem, termasuk korelasi tiap komponen dengan pembuatan log, kinerja, dan kegagalan. Karena penelusuran memudahkan identifikasi akar masalah secara visual, waktu penyelidikan akar masalah menjadi lebih singkat. Tim yang memahami interaksi komponen secara detail mengambil keputusan yang lebih baik dan lebih cepat saat menyelesaikan masalah. Keputusan seperti kapan harus memanggil failover pemulihan bencana (DR) atau lokasi terbaik untuk menerapkan strategi penyembuhan mandiri

dapat ditingkatkan dengan menganalisis jejak sistem, dan pada akhirnya meningkatkan kepuasan pelanggan terhadap layanan Anda.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Tim yang mengoperasikan aplikasi yang terdistribusi dapat menggunakan alat penelusuran untuk membuat pengidentifikasi korelasi, mengumpulkan jejak permintaan, dan membuat peta layanan komponen-komponen yang terhubung. Semua komponen aplikasi harus disertakan dalam jejak permintaan termasuk klien layanan, gateway perangkat lunak perantara (middleware) dan bus peristiwa, komponen komputasi, dan penyimpanan, termasuk penyimpanan nilai kunci dan basis data. Sertakan canary sintetis dan pemantauan pengguna nyata dalam konfigurasi penelusuran menyeluruh Anda untuk mengukur interaksi dan latensi klien jarak jauh sehingga Anda dapat secara akurat mengevaluasi kinerja sistem Anda berdasarkan perjanjian dan tujuan tingkat layanan Anda.

Anda dapat menggunakan layanan instrumentasi AWS X-Ray dan Pemantauan Aplikasi Amazon CloudWatch untuk memberikan tampilan utuh permintaan yang diproses melalui aplikasi Anda. X-Ray mengumpulkan telemetri aplikasi dan memungkinkan Anda untuk memvisualisasikan dan menyaringnya di seluruh muatan, fungsi, jejak, layanan, API, dan dapat diaktifkan untuk komponen sistem, dengan rendah kode atau tanpa kode. Pemantauan aplikasi CloudWatch mencakup ServiceLens untuk mengintegrasikan jejak Anda dengan metrik, log, dan alarm. Pemantauan aplikasi CloudWatch juga mencakup Syntethics untuk memantau titik akhir dan API Anda, serta pemantauan pengguna nyata untuk melengkapi klien aplikasi web Anda.

Langkah implementasi

- Gunakan AWS X-Ray pada semua layanan native yang didukung seperti Amazon S3, AWS Lambda, dan Amazon API Gateway. Semua layanan AWS ini mengaktifkan X-Ray dengan tombol konfigurasi menggunakan infrastruktur sebagai kode, SDK AWS, atau AWS Management Console.
- Aplikasi instrumen AWS Distro for Open Telemetry dan X-Ray atau agen pengumpulan pihak ketiga.
- Tinjau Panduan AWS X-Ray untuk Pengembang untuk implementasi bahasa pemrograman khusus. Bagian dokumentasi ini menjelaskan cara menginstrumentasi permintaan HTTP, kueri SQL, dan proses lain yang spesifik untuk bahasa pemrograman aplikasi Anda.
- Gunakan penelusuran X-Ray untuk Amazon CloudWatch Synthetic Canaries dan Amazon CloudWatch RUM untuk menganalisis jalur permintaan dari klien pengguna akhir Anda melalui infrastruktur AWS hilir Anda.

- Konfigurasikan metrik dan alarm CloudWatch berdasarkan telemetri canary dan kesehatan sumber daya sehingga tim menerima peringatan masalah dengan cepat, kemudian dapat mempelajari jejak dan peta layanan dengan ServiceLens.
- Aktifkan integrasi X-Ray untuk alat penelusuran pihak ketiga seperti Datadog, New Relic, atau Dynatrace jika Anda menggunakan alat pihak ketiga untuk solusi penelusuran utama Anda.

Sumber daya

Praktik terbaik terkait:

- REL06-BP01 Memantau semua komponen untuk beban kerja (Pembuatan)
- REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan

Dokumen terkait:

- Apa itu AWS X-Ray?
- Amazon CloudWatch: Pemantauan Aplikasi
- Melakukan debug dengan Amazon CloudWatch Synthetics dan AWS X-Ray
- Amazon Builders' Library: Instrumentasi sistem terdistribusi untuk visibilitas operasional
- Mengintegrasikan AWS X-Ray dengan layanan AWS lain
- AWS Distro for OpenTelemetry dan AWS X-Ray
- Amazon CloudWatch: Menggunakan pemantauan sintetis
- Amazon CloudWatch: Gunakan CloudWatch RUM
- Mengatur canary sintetis Amazon CloudWatch dan alarm Amazon CloudWatch
- Ketersediaan dan Lainnya: Memahami dan Meningkatkan Ketangguhan Sistem Terdistribusi di AWS

Contoh terkait:

- Lokakarya One Observability

Video terkait:

- AWS re:Invent 2022 - Cara memantau aplikasi di beberapa akun

- Cara Memantau Aplikasi AWS Anda

Alat terkait:

- AWS X-Ray
- Amazon CloudWatch
- Amazon Route 53

Rancang beban kerja Anda agar dapat beradaptasi dengan perubahan dalam permintaan

Beban kerja yang dapat diskalakan memberikan elastisitas untuk menambahkan atau mengeluarkan sumber daya secara otomatis sehingga sangat sesuai dengan permintaan yang sedang berjalan pada titik waktu tertentu.

Praktik terbaik

- REL07-BP01 Menggunakan otomatisasi ketika mendapatkan atau menskalakan sumber daya
- REL07-BP02 Mendapatkan sumber daya setelah deteksi gangguan pada beban kerja
- REL07-BP03 Menambah sumber daya berdasarkan deteksi bahwa beban kerja memerlukan lebih banyak sumber daya
- REL07-BP04 Menguji beban untuk beban kerja Anda

REL07-BP01 Menggunakan otomatisasi ketika mendapatkan atau menskalakan sumber daya

Ketika mengganti sumber daya yang terganggu atau menskalakan beban kerja Anda, otomatiskan proses menggunakan AWS Managed Services (AMS), seperti Amazon S3 dan AWS Auto Scaling. Anda juga dapat menggunakan alat pihak ketiga dan SDK AWS untuk mengotomatiskan penskalaan.

AWS Managed Services mencakup Amazon S3, Amazon CloudFront, AWS Auto Scaling, AWS Lambda, Amazon DynamoDB, AWS Fargate, dan Amazon Route 53.

Dengan AWS Auto Scaling, Anda dapat mendeteksi dan mengganti instans yang terganggu. Dengan ini, Anda juga dapat membuat rencana penskalaan untuk sumber daya, termasuk instans Amazon

EC2 dan Armada Spot, tugas Amazon ECS , tabel dan indeks Amazon DynamoDB , serta Replika Amazon Aurora .

Saat menskalakan instans EC2, pastikan bahwa Anda menggunakan Zona Ketersediaan (disarankan minimal tiga) serta menambahkan atau menghapus kapasitas untuk menjaga keseimbangan di seluruh Zona Ketersediaan ini. Tugas ECS atau pod Kubernetes (saat menggunakan Amazon Elastic Kubernetes Service) juga harus didistribusikan ke beberapa Zona Ketersediaan.

Ketika menggunakan AWS Lambda, instans menskalakan secara otomatis. Setiap kali notifikasi diterima untuk fungsi Anda, AWS Lambda langsung mencari kapasitas bebas di dalam armada komputasinya, serta menjalankan kode Anda sesuai konkurensi yang dialokasikan. Anda harus memastikan bahwa konkurensi yang diperlukan telah dikonfigurasikan di Lambda tertentu, dan di dalam Service Quotas.

Amazon S3 diskalakan secara otomatis untuk menangani tingkat permintaan tinggi. Misalnya, aplikasi Anda dapat memenuhi minimum 3.500 permintaan PUT/COPY/POST/DELETE atau 5.500 permintaan GET/HEAD per detik per prefiks dalam bucket. Tidak ada batasan jumlah prefiks dalam bucket. Anda dapat meningkatkan kinerja baca atau tulis Anda dengan memparalelkan pembacaan. Misalnya, jika Anda membuat 10 prefiks dalam sebuah bucket Amazon S3 untuk memparalelkan pembacaan, Anda dapat menskalakan kinerja baca Anda hingga 55.000 permintaan baca per detik.

Konfigurasikan dan gunakan Amazon CloudFront atau jaringan pengiriman konten (CDN) tepercaya. CDN dapat memberikan waktu respons pengguna akhir yang lebih cepat untuk konten dari cache, sehingga mengurangi kebutuhan untuk menskalakan beban kerja Anda.

Antipola umum:

- Mengimplementasikan grup Auto Scaling untuk pemulihan otomatis, tetapi tidak mengimplementasikan elastisitas.
- Menggunakan penskalaan otomatis untuk merespons peningkatan yang signifikan di lalu lintas.
- Melakukan deployment aplikasi yang sangat stateful, menghilangkan opsi elastisitas.

Manfaat menerapkan praktik terbaik ini: Otomatisasi menghilangkan potensi kesalahan manual dalam melakukan deployment dan penonaktifan sumber daya. Otomatisasi menghilangkan risiko pembengkakan biaya dan penolakan layanan akibat lambatnya respons saat dibutuhkan untuk melakukan deployment atau penonaktifan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Tinggi

Panduan implementasi

- Konfigurasikan dan gunakan AWS Auto Scaling. Ini memantau aplikasi Anda dan secara otomatis menyesuaikan kapasitas untuk mempertahankan kinerja yang stabil dan dapat diprediksi dengan biaya serendah mungkin. Menggunakan AWS Auto Scaling, Anda dapat mengonfigurasi penskalaan aplikasi untuk beberapa sumber daya di beberapa layanan.
  - Apa itu AWS Auto Scaling?
    - Konfigurasikan Penskalaan Otomatis dalam instans Amazon EC2 dan Armada Spot, tugas Amazon ECS, tabel dan indeks Amazon DynamoDB, Replika Amazon Aurora, serta perangkat AWS Marketplace Anda sebagai dapat diterapkan.
      - Mengelola kapasitas throughput secara otomatis dengan Penskalaan Otomatis DynamoDB.
        - Gunakan operasi API layanan untuk menentukan peringatan, kebijakan penskalaan, waktu warm up, dan waktu cool down.
- Gunakan Elastic Load Balancing. Penyeimbang beban dapat mendistribusikan beban berdasarkan jalur atau berdasarkan konektivitas jaringan.
  - Apa itu Elastic Load Balancing?
    - Application Load Balancers dapat mendistribusikan beban berdasarkan jalur.
      - Apa itu Application Load Balancer?
        - Konfigurasikan Application Load Balancer untuk mendistribusikan lalu lintas ke berbagai beban kerja berdasarkan nama domain.
        - Application Load Balancers dapat digunakan untuk mendistribusikan beban dengan cara yang terintegrasi dengan AWS Auto Scaling untuk mengelola permintaan.
          - Menggunakan penyeimbang beban dengan grup Auto Scaling.
    - Penyeimbang Beban Jaringan dapat mendistribusikan beban berdasarkan koneksi.
      - Apa itu Penyeimbang Beban Jaringan?
        - Konfigurasikan Penyeimbang Beban Jaringan untuk mendistribusikan lalu lintas ke berbagai beban kerja menggunakan TCP, atau untuk mendapatkan set konstan alamat IP untuk beban kerja Anda.
        - Penyeimbang Beban Jaringan dapat digunakan untuk mendistribusikan beban yang terintegrasi dengan AWS Auto Scaling untuk mengelola permintaan.
- Gunakan penyedia DNS dengan ketersediaan tinggi. Nama DNS memungkinkan pengguna Anda untuk memasukkan nama tanpa perlu memasukkan alamat IP guna mengakses beban kerja Anda

dan mendistribusikan informasi ini ke cakupan yang ditentukan, biasanya untuk pengguna beban kerja secara global.

- Gunakan Amazon Route 53 atau penyedia DNS tepercaya.
  - Apa itu Amazon Route 53?
- Gunakan Route 53 untuk mengelola penyeimbang beban dan distribusi CloudFront Anda.
  - Tentukan domain dan subdomain yang akan Anda kelola.
  - Buat set catatan yang sesuai menggunakan catatan ALIAS atau CNAME.
    - Bekerja dengan catatan

- Gunakan jaringan global AWS untuk optimasi jalur pengguna Anda ke aplikasi. AWS Global Accelerator memantau kondisi titik akhir aplikasi Anda secara terus-menerus dan mengalihkan lalu lintas ke titik akhir yang sehat dalam kurang dari 30 detik.
  - AWS Global Accelerator adalah layanan yang meningkatkan ketersediaan dan kinerja aplikasi Anda untuk pengguna global atau lokal. Ini menyediakan alamat IP statis yang berperan sebagai titik entri tetap ke titik akhir aplikasi Anda dalam satu atau beberapa Wilayah AWS, seperti Application Load Balancers, Penyeimbang Beban Jaringan, atau instans Amazon EC2 Anda.
    - Apa Itu AWS Global Accelerator?
- Konfigurasikan dan gunakan Amazon CloudFront atau jaringan pengiriman konten (CDN) tepercaya. Jaringan pengiriman konten dapat memberikan waktu respons pengguna akhir yang lebih cepat serta memenuhi permintaan konten yang dapat mengakibatkan penskalaan yang tidak perlu dari beban kerja Anda.
  - Apa itu Amazon CloudFront?
    - Konfigurasikan distribusi Amazon CloudFront untuk beban kerja Anda, atau gunakan CDN pihak ketiga.
      - Anda dapat membatasi akses ke beban kerja Anda agar hanya dapat diakses dari CloudFront menggunakan rentang IP untuk CloudFront dalam grup keamanan titik akhir atau kebijakan akses Anda.

Sumber daya

Dokumen terkait:

- Partner APN: partner yang dapat membantu Anda membuat solusi komputasi yang diotomatiskan.
- AWS Auto Scaling: Cara Kerja Rencana Penskalaan
- AWS Marketplace: produk yang dapat digunakan dengan penskalaan otomatis

- Mengelola Kapasitas Throughput Secara Otomatis dengan Penskalaan Otomatis DynamoDB.
- Menggunakan penyeimbang beban dengan grup Auto Scaling.
- Apa Itu AWS Global Accelerator?
- Apa Itu Amazon EC2 Auto Scaling?
- Apa itu AWS Auto Scaling?
- Apa itu Amazon CloudFront?
- Apa itu Amazon Route 53?
- Apa itu Elastic Load Balancing?
- Apa itu Penyeimbang Beban Jaringan?
- Apa itu Application Load Balancer?
- Bekerja dengan catatan

REL07-BP02 Mendapatkan sumber daya setelah deteksi gangguan pada beban kerja

Skalakan sumber daya secara reaktif saat diperlukan jika ketersediaan terganggu, guna memulihkan ketersediaan beban kerja.

Anda terlebih dahulu harus mengonfigurasi pemeriksaan kondisi dan kriteria pada pemeriksaan ini agar memberikan penanda saat ketersediaan terganggu oleh kurangnya sumber daya. Lalu, beri tahu personel yang bersangkutan untuk menskalakan sumber daya secara manual, atau mulai otomatisasi untuk menskalakannya secara otomatis.

Skala dapat disesuaikan secara manual untuk beban kerja Anda (misalnya, mengubah jumlah instans EC2 di grup Auto Scaling atau modifikasi throughput tabel DynamoDB melalui AWS Management Console atau AWS CLI). Namun, otomatisasi harus digunakan apabila memungkinkan (lihat Menggunakan otomatisasi ketika mendapatkan atau menskalakan sumber daya).

Hasil yang diinginkan: Aktivitas penskalaan (baik secara otomatis maupun manual) diinisiasi untuk memulihkan ketersediaan setelah terdeteksinya kegagalan atau menurunnya pengalaman pelanggan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Terapkan observabilitas dan pemantauan di semua komponen dalam beban kerja Anda untuk memantau pengalaman pelanggan dan mendeteksi kegagalan. Tentukan prosedur, manual atau otomatis, yang menskalakan sumber daya yang diperlukan. Untuk informasi lebih lanjut, lihat REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan.

Langkah implementasi

- Tentukan prosedur, manual atau otomatis, yang menskalakan sumber daya yang dibutuhkan.
  - Prosedur penskalaan tergantung pada bagaimana rancangan berbagai komponen dalam beban kerja Anda.
  - Prosedur penskalaan juga bervariasi, tergantung teknologi dasar yang digunakan.
    - Komponen yang menggunakan AWS Auto Scaling dapat menggunakan rencana penskalaan untuk mengonfigurasi serangkaian instruksi guna menskalakan sumber daya Anda. Jika Anda bekerja dengan AWS CloudFormation atau menambahkan tag ke sumber daya AWS, Anda dapat menyiapkan rencana penskalaan untuk berbagai set sumber daya per aplikasi. Auto Scaling menyediakan saran strategi penskalaan yang disesuaikan untuk tiap-tiap sumber daya. Setelah Anda membuat rencana penskalaan, Auto Scaling menggabungkan metode penskalaan dinamis dan penskalaan prediktif untuk mendukung strategi penskalaan Anda. Untuk detail selengkapnya, lihat Cara kerja rencana penskalaan.
    - Amazon EC2 Auto Scaling memverifikasi bahwa jumlah instans Amazon EC2 Anda yang tersedia sudah tepat untuk menangani beban aplikasi Anda. Anda membuat koleksi instans EC2, yang disebut grup Auto Scaling. Anda dapat menentukan jumlah instans minimum dan maksimum di setiap grup Auto Scaling, dan Amazon EC2 Auto Scaling memastikan bahwa grup Anda tidak pernah berada di bawah atau di atas batas ini. Untuk lebih jelasnya, lihat Apa itu Amazon EC2 Auto Scaling?
    - Penskalaan otomatis Amazon DynamoDB menggunakan layanan Application Auto Scaling untuk menyesuaikan secara dinamis kapasitas throughput yang disediakan atas nama Anda, sebagai respons terhadap pola lalu lintas aktual. Ini memungkinkan tabel atau indeks sekunder global untuk meningkatkan kapasitas baca dan tulis yang disediakan untuk menangani peningkatan lalu lintas yang mendadak, tanpa throttling. Untuk detail selengkapnya, lihat Mengelola kapasitas throughput secara otomatis dengan penskalaan otomatis DynamoDB.

Sumber daya

Praktik Terbaik Terkait:

- REL07-BP01 Menggunakan otomatisasi ketika mendapatkan atau menskalakan sumber daya
- REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan

Dokumen terkait:

- AWS Auto Scaling: Cara Kerja Rencana Penskalaan
- Mengelola Kapasitas Throughput Secara Otomatis dengan Penskalaan Otomatis DynamoDB
- Apa Itu Amazon EC2 Auto Scaling?

REL07-BP03 Menambah sumber daya berdasarkan deteksi bahwa beban kerja memerlukan lebih banyak sumber daya

Skalakan sumber daya secara proaktif untuk memenuhi permintaan dan menghindari dampak ketersediaan.

Banyak layanan AWS yang melakukan penskalaan secara otomatis untuk memenuhi permintaan. Dengan menggunakan instans Amazon EC2 atau klaster Amazon ECS, Anda dapat mengonfigurasikan penskalaan otomatis ini agar muncul berdasarkan penggunaan metrik yang sesuai dengan permintaan untuk beban kerja Anda. Untuk Amazon EC2, rata-rata pemanfaatan CPU, jumlah permintaan penyeimbang beban, atau bandwidth jaringan dapat digunakan untuk menskalakan ke luar (atau menskalakan ke dalam) instans EC2. Untuk Amazon ECS, rata-rata pemanfaatan CPU, jumlah permintaan penyeimbang beban, dan pemanfaatan memori dapat digunakan untuk menskalakan ke luar (atau menskalakan ke dalam) tugas ECS. Menggunakan Penskalaan Otomatis Target di AWS, penskala otomatis berperan seperti termostat, yang menambahkan atau menghapus sumber daya untuk mempertahankan nilai target (misalnya, 70% pemanfaatan CPU) yang Anda tentukan.

AWS Auto Scaling juga dapat melakukan Penskalaan Otomatis Prediktif, yang menggunakan machine learning untuk menganalisis setiap beban kerja historis sumber daya dan memperkirakan beban untuk dua hari mendatang secara rutin.

Little’s Law membantu menghitung banyaknya instans komputasi (instans EC2, fungsi Lambda bersamaan, dll.) yang Anda butuhkan.

$L = \lambda W$

L = jumlah instans (atau konkurensi nilai tengah dalam sistem)

λ = rasio rata-rata permintaan yang diterima (permintaan/detik)

W = waktu rata-rata yang diperlukan setiap permintaan di dalam sistem (detik)

Misalnya, dengan laju 100 rps (permintaan per detik), jika setiap permintaan memerlukan 0,5 detik untuk diproses, Anda akan memerlukan 50 instans untuk memenuhi permintaan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Sedang

Panduan implementasi

- Tambahkan sumber daya berdasarkan deteksi bahwa beban kerja memerlukan lebih banyak sumber daya. Skalakan sumber daya secara proaktif untuk memenuhi permintaan dan menghindari dampak ketersediaan.
  - Hitung banyaknya sumber daya komputasi yang akan Anda butuhkan (konkurensi komputasi) untuk menangani rasio permintaan tertentu.
    - Seputar Little's Law
  - Jika Anda memiliki pola historis untuk penggunaan, atur penskalaan terjadwal untuk penskalaan otomatis Amazon EC2.
    - Penskalaan Terjadwal untuk Amazon EC2 Auto Scaling
  - Gunakan penskalaan prediktif AWS.
    - Penskalaan Prediktif untuk EC2, Didukung oleh Machine Learning

Sumber daya

Dokumen terkait:

- AWS Auto Scaling: Cara Kerja Rencana Penskalaan
- AWS Marketplace: produk yang dapat digunakan dengan penskalaan otomatis
- Mengelola Kapasitas Throughput Secara Otomatis dengan Penskalaan Otomatis DynamoDB.
- Penskalaan Prediktif untuk EC2, Didukung oleh Machine Learning
- Penskalaan Terjadwal untuk Amazon EC2 Auto Scaling

- Seputar Little's Law
- Apa Itu Amazon EC2 Auto Scaling?

REL07-BP04 Menguji beban untuk beban kerja Anda

Adopsi metodologi pengujian beban untuk mengukur apakah aktivitas penskalaan memenuhi persyaratan beban kerja.

Pengujian beban yang berkelanjutan penting untuk dilakukan. Pengujian beban harus menemukan titik nadir dan menguji kinerja beban kerja Anda. AWS memudahkan penyiapan lingkungan pengujian sementara yang memodelkan skala beban kerja produksi Anda. Di cloud, Anda dapat membuat lingkungan pengujian berskala produksi sesuai permintaan, menyelesaikan pengujian, kemudian menonaktifkan sumber dayanya. Karena Anda hanya membayar lingkungan pengujian saat sedang berjalan, Anda dapat menyimulasikan lingkungan langsung Anda dengan biaya yang lebih murah daripada pengujian on-premise.

Pengujian beban di produksi juga harus dipertimbangkan sebagai bagian dari aktivitas game day di mana sistem produksi diberikan tekanan, selama jam-jam penggunaan pelanggan yang lebih rendah, dengan semua personel siap menerjemahkan hasilnya dan menangani masalah yang muncul.

Antipola umum:

- Melakukan pengujian beban di lingkungan deployment yang tidak memiliki konfigurasi yang sama dengan produksi Anda.
- Melakukan pengujian beban hanya pada beban kerja Anda secara terpisah-pisah, bukan pada keseluruhan beban kerja.
- Melakukan pengujian beban dengan subset permintaan, bukan set permintaan riil yang representatif.
- Melakukan pengujian beban ke faktor keselamatan kecil di atas beban yang diharapkan.

Manfaat menjalankan praktik terbaik ini: Anda mengetahui komponen apa saja di dalam arsitektur Anda yang gagal saat menerima beban, dan mampu mengidentifikasi metrik apa saja yang perlu diamati sebagai indikator bahwa Anda mendekati beban tersebut tepat waktu untuk mengatasi masalah dan mencegah dampak kegagalan tersebut.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Sedang

Panduan implementasi

- Lakukan pengujian beban untuk mengidentifikasi aspek dalam beban kerja Anda yang menunjukkan bahwa Anda harus menambah atau menghapus kapasitas. Pengujian beban harus memiliki lalu lintas representatif yang serupa dengan yang Anda terima di lingkungan produksi. Tingkatkan beban sambil mengamati metrik yang telah Anda instrumentasi untuk menentukan metrik mana yang menunjukkan kapan Anda harus menambah atau menghapus sumber daya.
  - Pengujian Beban Terdistribusi di AWS: simulasikan ribuan pengguna terhubung
    - Identifikasi gabungan permintaan. Anda mungkin memiliki gabungan permintaan yang beragam, sehingga Anda harus melihat berbagai kerangka waktu saat mengidentifikasi gabungan lalu lintas.
    - Implementasikan pendorong beban. Anda dapat menggunakan aplikasi kode kustom, sumber terbuka, atau komersial untuk mengimplementasikan pendorong beban.
    - Lakukan uji beban di awal menggunakan kapasitas kecil. Anda melihat beberapa dampak langsung dengan mendorong beban ke kapasitas yang lebih kecil, kemungkinan seukuran satu instans atau kontainer.
    - Uji beban dengan kapasitas yang lebih besar. Efek akan berbeda di beban yang terdistribusi, sehingga Anda harus menguji di lingkungan yang semirip mungkin dengan lingkungan produksi.

Sumber daya

Dokumen terkait:

- Pengujian Beban Terdistribusi di AWS: simulasikan ribuan pengguna terhubung
- Aplikasi pengujian beban

Video terkait:

- AWS Summit ANZ 2023: Lakukan akselerasi dengan percaya diri melalui Pengujian Beban Terdistribusi AWS

Implementasikan perubahan

Perubahan terkontrol diperlukan untuk melakukan deployment fungsionalitas baru, dan untuk memastikan bahwa beban kerja dan lingkungan operasi menjalankan perangkat lunak yang dikenal dan dapat di-patch. Jika perubahan-perubahan ini tidak terkontrol, akan sulit untuk memprediksi efek dari perubahan-perubahan tersebut, atau untuk mengatasi masalah yang ditimbulkannya.

Praktik terbaik

- REL08-BP01 Menggunakan runbook untuk aktivitas standar seperti deployment
- REL08-BP02 Integrasikan pengujian fungsional sebagai bagian dari deployment Anda
- REL08-BP03 Mengintegrasikan pengujian ketahanan sebagai bagian dari deployment Anda
- REL08-BP04 Melakukan deployment menggunakan infrastruktur tetap
- REL08-BP05 Melakukan deployment perubahan dengan otomatisasi

REL08-BP01 Menggunakan runbook untuk aktivitas standar seperti deployment

Runbook adalah prosedur terdokumentasi untuk mencapai hasil tertentu. Gunakan runbook untuk melakukan aktivitas standar, baik yang dilakukan secara manual maupun otomatis. Contohnya adalah men-deploy beban kerja, mem-patch beban kerja, atau membuat modifikasi DNS.

Misalnya, terapkan proses untuk memastikan keamanan pembatalan selama deployment. Memastikan bahwa Anda dapat membatalkan deployment tanpa gangguan terhadap pelanggan adalah sesuatu yang penting dalam menciptakan keandalan layanan.

Untuk prosedur runbook, mulailah dengan proses manual efektif yang valid, implementasikan dalam kode, dan picu agar berjalan secara otomatis saat diperlukan.

Bahkan untuk beban kerja canggih yang diotomatiskan dalam tingkat tinggi, runbook tetap bermanfaat untuk menjalankan aktivitas game day atau memenuhi persyaratan pelaporan dan audit yang ketat.

Ingat bahwa buku pedoman digunakan untuk merespons insiden tertentu, sedangkan runbook digunakan untuk mencapai hasil tertentu. Sering kali, runbook ditujukan untuk aktivitas rutin, sedangkan buku pedoman digunakan untuk merespons peristiwa nonrutin.

Antipola umum:

- Melakukan perubahan tidak terencana pada konfigurasi di lingkungan produksi.
- Melewatkan langkah-langkah dalam rencana Anda untuk men-deploy lebih cepat, sehingga mengakibatkan kegagalan deployment.
- Membuat perubahan tanpa menguji pembatalan perubahan.

Manfaat menjalankan praktik terbaik ini: Perencanaan perubahan yang efektif meningkatkan kemampuan Anda untuk berhasil mengeksekusi perubahan karena Anda mengetahui semua sistem yang terpengaruh. Validasi perubahan di lingkungan pengujian meningkatkan kepercayaan diri Anda.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

- Aktifkan respons yang cepat dan konsisten terhadap peristiwa yang dipahami dengan baik dengan cara mendokumentasikan prosedur di dalam runbook.
  - AWS Well-Architected Framework: Konsep: Runbook
- Gunakan prinsip infrastruktur sebagai kode untuk menetapkan infrastruktur Anda. Dengan menggunakan AWS CloudFormation (atau pihak ketiga tepercaya) untuk menetapkan infrastruktur Anda, Anda dapat menggunakan perangkat lunak kontrol versi untuk membuat versi baru dan melacak perubahan.
  - Gunakan AWS CloudFormation (atau penyedia pihak ketiga tepercaya) untuk menetapkan infrastruktur Anda.
    - Apa itu AWS CloudFormation?
  - Buat templat singular dan terpisah-pisah, menggunakan prinsip desain perangkat lunak yang baik.
    - Tentukan izin, templat, dan pihak-pihak yang bertanggung jawab untuk implementasi.
      - Mengontrol akses dengan AWS Identity and Access Management.
    - Gunakan kontrol sumber, seperti AWS CodeCommit atau alat pihak ketiga tepercaya, untuk kontrol versi.
      - Apa Itu AWS CodeCommit?

Sumber daya

Dokumen terkait:

- Partner APN: partner yang dapat membantu Anda membuat solusi deployment yang diotomatisasi
- AWS Marketplace: produk yang dapat digunakan untuk mengotomatisasi deployment Anda
- AWS Well-Architected Framework: Konsep: Runbook
- Apa itu AWS CloudFormation?
- Apa Itu AWS CodeCommit?

Contoh terkait:

- Mengotomatiskan operasi dengan Buku Pedoman dan Runbook

REL08-BP02 Integrasikan pengujian fungsional sebagai bagian dari deployment Anda

Uji fungsional dijalankan sebagai bagian dari deployment otomatis. Jika kriteria untuk sukses tidak terpenuhi, maka alur akan dihentikan atau dikembalikan. Pengujian ini dijalankan dalam lingkungan praproduksi, yang dilaksanakan sebelum perkembangan produksi. Idealnya, ini dilakukan sebagai bagian dari alur deployment.

Hasil yang diinginkan: Anda menggunakan otomatisasi untuk melakukan pengujian fungsional, dan data pengujian terkait mengurangi durasi dan biaya pengujian serta meningkatkan akurasi hasil pengujian. Anda mengintegrasikan pengujian fungsional sebagai bagian dari proses deployment Anda, yang membantu Anda mengotomatiskan pipeline rilis agar pembaruan aplikasi dan infrastruktur menjadi cepat dan andal.

Antipola umum:

- Anda melakukan pengujian secara manual di luar pipeline deployment.
- Anda melewatkan langkah-langkah pengujian dalam otomatisasi Anda melalui alur kerja darurat manual.
- Anda tidak mengikuti rencana dan proses pengujian yang telah ditetapkan demi mempercepat jadwal.

Manfaat menjalankan praktik terbaik ini: Pengujian fungsional memvalidasi bahwa sistem beroperasi sesuai dengan persyaratan yang ditentukan. Pengujian ini digunakan untuk memverifikasi urutan kerja komponen yang diinginkan seperti antarmuka pengguna, API, basis data, dan kode sumber secara konsisten. Ketika Anda memeriksa komponen-komponen sistem ini, pengujian fungsional memverifikasi bahwa setiap fitur berperilaku seperti yang diharapkan, yang melindungi harapan

pengguna sekaligus integritas perangkat lunak. Integrasikan pengujian fungsional sebagai bagian dari deployment rutin Anda, dan gunakan otomatisasi untuk melakukan deployment semua perubahan, yang mengurangi potensi kesalahan manusia.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Integrasikan pengujian fungsional sebagai bagian dari deployment Anda. Uji fungsional dijalankan sebagai bagian dari deployment otomatis. Jika kriteria keberhasilan tidak terpenuhi, pipeline dihentikan atau dibatalkan. AWS CodePipeline menyediakan pipeline pengiriman berkelanjutan untuk pengujian otomatis, yang memungkinkan penguji untuk mengotomatiskan seluruh proses pengujian dan deployment. Pipeline ini terintegrasi dengan layanan AWS seperti AWS CodeBuild dan AWS CodeDeploy untuk mengotomatiskan fase pembuatan, pengujian, dan deployment siklus pengembangan perangkat lunak.

Langkah implementasi

- Konfigurasikan pipeline Anda: Siapkan tahap pengadaan, pembuatan, pengujian, dan deployment menggunakan konsol AWS CodePipeline atau AWS Command Line Interface (CLI).
  - Tentukan sumber Anda: DenganAWS CodePipeline, Anda dapat secara otomatis mengambil kode sumber dari sistem kontrol versi seperti GitHub, AWS CodeCommit, atau Bitbucket, yang memverifikasi bahwa kode terbaru selalu digunakan untuk pengujian.
  - Otomatiskan pembangunan dan pengujian: AWS CodeBuild dapat membangun dan menguji kode Anda serta menghasilkan laporan pengujian. Ini mendukung kerangka pengujian populer seperti JUnit, NUnit, dan TestNG.
  - Lakukan deployment kode Anda: Setelah dibangun dan diuji, kode dapat di-deploy oleh AWS CodeDeploy ke lingkungan pengujian Anda, termasuk instans Amazon EC2, fungsi AWS Lambda, atau server on-premise.
  - Pantau pipeline: AWS CodePipeline dapat melacak kemajuan pipeline Anda dan status setiap tahap. Anda juga dapat menggunakan pemeriksaan kualitas untuk memblokir pipeline sesuai status pelaksanaan pengujian. Anda juga dapat menerima notifikasi untuk kegagalan tahap pipeline atau penyelesaian pipeline.

Sumber daya

Dokumen terkait:

- Use AWS CodePipeline with AWS CodeBuild to test code and run builds
- Logging and monitoring in AWS CodeBuild
- Indicators for functional testing

REL08-BP03 Mengintegrasikan pengujian ketahanan sebagai bagian dari deployment Anda

Integrasikan pengujian ketahanan dengan memasukkan kegagalan secara sadar ke dalam sistem Anda guna mengukur kemampuannya apabila terjadi skenario yang mengganggu. Pengujian ketahanan berbeda dari pengujian unit dan fungsi yang biasanya terintegrasi dalam siklus deployment, karena pengujian ini berfokus pada identifikasi kegagalan yang tidak terduga di dalam sistem Anda. Meskipun memulai dengan integrasi pengujian ketahanan dalam tahap praproduksi aman dilakukan, tetapkan tujuan untuk mengimplementasikan pengujian ini dalam produksi sebagai bagian dari gameday Anda.

Hasil yang diinginkan: Pengujian ketahanan membantu membangun kepercayaan pada kemampuan sistem untuk mengatasi penurunan kualitas dalam produksi. Eksperimen mengidentifikasi titik lemah yang dapat menyebabkan kegagalan, yang membantu Anda meningkatkan kualitas sistem Anda untuk mengurangi kegagalan dan penurunan kualitas secara otomatis dan efisien.

Antipola umum:

- Kurangnya observabilitas dan pemantauan dalam proses deployment
- Ketergantungan pada manusia untuk mengatasi kegagalan sistem
- Mekanisme analisis kualitas yang buruk
- Fokus pada masalah yang diketahui dalam suatu sistem dan kurangnya eksperimen untuk mengidentifikasi masalah yang tidak diketahui
- Identifikasi kegagalan, tetapi tidak ada penyelesaian
- Tidak ada dokumentasi temuan dan runbook

Manfaat menjalankan praktik terbaik: Pengujian ketahanan yang terintegrasi dalam deployment Anda membantu mengidentifikasi masalah yang tidak diketahui di dalam sistem yang biasanya tidak diketahui, yang dapat menyebabkan waktu henti dalam produksi. Identifikasi masalah yang tidak diketahui ini di dalam sistem membantu Anda mendokumentasikan temuan, mengintegrasikan

pengujian ke dalam proses CI/CD Anda, dan membangun runbook, yang menyederhanakan mitigasi melalui mekanisme yang efisien dan dapat diulang.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Bentuk pengujian ketahanan yang paling umum yang dapat diintegrasikan dalam deployment sistem Anda adalah pemulihan bencana dan chaos engineering.

- Sertakan pembaruan pada rencana pemulihan bencana dan prosedur operasi standar (SOP) Anda dengan deployment signifikan apa pun.
- Integrasikan pengujian keandalan ke dalam pipeline deployment otomatis Anda. Layanan seperti AWS Resilience Hub dapat diintegrasikan ke dalam pipeline CI/CD Anda untuk menetapkan penilaian ketahanan berkelanjutan yang dievaluasi secara otomatis sebagai bagian dari setiap deployment.
- Tentukan aplikasi Anda di AWS Resilience Hub. Penilaian ketahanan menghasilkan cuplikan kode yang membantu Anda menciptakan prosedur pemulihan dalam bentuk dokumen AWS Systems Manager untuk aplikasi Anda dan menyediakan daftar monitor dan alarm Amazon CloudWatch yang direkomendasikan.
- Setelah rencana DR dan SOP Anda diperbarui, selesaikan pengujian pemulihan bencana untuk memverifikasi efektivitasnya. Pengujian pemulihan bencana membantu Anda menentukan apakah Anda dapat memulihkan sistem setelah peristiwa tertentu dan kembali ke operasi normal. Anda dapat menyimulasikan berbagai strategi pemulihan bencana dan mengidentifikasi apakah perencanaan Anda memadai untuk memenuhi persyaratan waktu aktif Anda. Strategi pemulihan bencana umum antara lain pencadangan dan pemulihan, pilot light, cold standby, warm standby, hot standby, dan active-active, dan semuanya memiliki biaya dan kompleksitas yang berbeda-beda. Sebelum pengujian pemulihan bencana, sebaiknya tentukan sasaran waktu pemulihan (RTO) dan sasaran titik pemulihan (RPO) untuk menyederhanakan pilihan strategi yang akan disimulasikan. AWS menawarkan alat pemulihan bencana seperti AWS Elastic Disaster Recovery untuk membantu Anda memulai perencanaan dan pengujian Anda.
- Eksperimen chaos engineering memasukkan gangguan ke dalam sistem, seperti pemadaman jaringan dan kegagalan layanan. Dengan melakukan simulasi dengan kegagalan terkontrol, Anda dapat menemukan kerentanan sistem Anda sambil mengendalikan dampak dari kegagalan yang dimasukkan. Sama seperti strategi lainnya, jalankan simulasi kegagalan terkontrol di lingkungan non-produksi menggunakan layanan seperti AWS Fault Injection Service untuk mendapatkan kepercayaan diri sebelum deployment di lingkungan produksi.

Sumber daya

Dokumen terkait:

- Experiment with failure using resilience testing to build recovery preparedness
- Continually assessing application resilience with AWS Resilience Hub and AWS CodePipeline
- Disaster recovery (DR) architecture on AWS, part 1: Strategies for recovery in the cloud
- Verify the resilience of your workloads using Chaos Engineering
- Prinsip-Prinsip Chaos Engineering
- Lokakarya Chaos Engineering

Video terkait:

- AWS re:Invent 2020: Testing Resilience using Chaos Engineering
- Improve Application Resilience with AWS Fault Injection Service
- Prepare & Protect Your Applications From Disruption With AWS Resilience Hub

REL08-BP04 Melakukan deployment menggunakan infrastruktur tetap

Infrastruktur tetap adalah model yang menuntut bahwa tidak ada pembaruan, patch keamanan, atau perubahan konfigurasi yang terjadi di tempat pada beban kerja produksi. Saat perubahan diperlukan, arsitektur dibangun ke infrastruktur baru dan di-deploy ke dalam produksi.

Ikuti strategi penerapan infrastruktur tetap untuk meningkatkan keandalan, konsistensi, dan keterulangan dalam deployment beban kerja Anda.

Hasil yang diinginkan: Dengan infrastruktur tetap, tidak ada modifikasi di tempat yang diizinkan untuk menjalankan sumber daya infrastruktur dalam beban kerja. Sebaliknya, ketika perubahan diperlukan, kumpulan sumber daya infrastruktur baru yang diperbarui, yang berisi semua perubahan yang diperlukan, di-deploy secara paralel dengan sumber daya Anda yang ada. Deployment ini divalidasi secara otomatis, dan jika berhasil, lalu lintas dialihkan secara bertahap ke kumpulan sumber daya baru.

Strategi deployment ini berlaku di antaranya untuk pembaruan perangkat lunak, patch keamanan, perubahan infrastruktur, pembaruan konfigurasi, dan pembaruan aplikasi.

Antipola umum:

- Menerapkan perubahan di tempat untuk menjalankan sumber daya infrastruktur.

Manfaat menjalankan praktik terbaik ini:

- Meningkatnya konsistensi di seluruh lingkungan: Karena tidak ada perbedaan sumber daya infrastruktur di seluruh lingkungan, konsistensi meningkat dan pengujian menjadi lebih sederhana.
- Berkurangnya penyimpangan konfigurasi: Dengan mengganti sumber daya infrastruktur dengan konfigurasi yang diketahui dan dikontrol versinya, infrastruktur diatur ke status yang diketahui, diuji, dan tepercaya, sehingga menghindari penyimpangan konfigurasi.
- Deployment atomik yang dapat diandalkan: Deployment hanya berujung pada dua hal: berhasil diselesaikan atau tidak ada perubahan, sehingga konsistensi dan keandalan dalam proses deployment meningkat.
- Deployment yang disederhanakan: Deployment disederhanakan karena tidak memerlukan pembaruan dukungan. Pembaruan hanyalah deployment baru.
- Deployment yang lebih aman dengan proses rollback dan pemulihan yang cepat: Deployment lebih aman karena versi kerja sebelumnya tidak berubah. Anda dapat melakukan rollback jika kesalahan terdeteksi.
- Postur keamanan yang lebih baik: Karena perubahan pada infrastruktur tidak diizinkan, mekanisme akses jarak jauh (seperti SSH) dapat dinonaktifkan. Hal ini mengurangi vektor serangan, sehingga meningkatkan postur keamanan organisasi.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Automasi

Saat menentukan strategi penyebaran infrastruktur tetap, sebaiknya gunakan otomatisasi sebanyak mungkin untuk meningkatkan keterulangan dan meminimalkan potensi kesalahan manusia. Untuk detail selengkapnya, lihat REL08-BP05 Melakukan deployment perubahan dengan otomatisasi dan Mengotomatiskan deployment aman tanpa campur tangan.

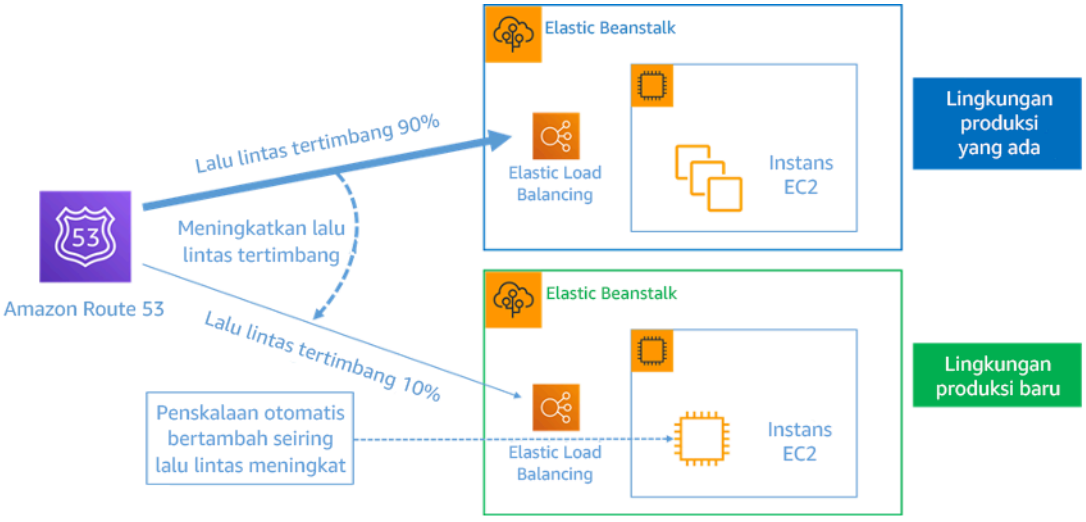
Dengan Infrastructure sebagai Kode (IaC), langkah-langkah penyediaan infrastruktur, orkestrasi, dan deployment ditentukan dalam cara yang terprogram, deskriptif, dan deklaratif dan disimpan dalam sistem kontrol sumber. Memanfaatkan infrastruktur sebagai kode makin memudahkan otomatisasi deployment infrastruktur dan membantu mewujudkan ketetapan infrastruktur.

Pola deployment

Ketika perubahan dalam beban kerja diperlukan, strategi deployment tetap mengharuskan deployment sumber daya infrastruktur yang baru, termasuk semua perubahan yang diperlukan. Penting agar kumpulan sumber daya baru ini mengikuti pola rollout yang meminimalkan dampak pengguna. Ada dua strategi utama untuk deployment ini:

Deployment canary: Praktik mengarahkan sejumlah kecil pelanggan ke versi baru, yang biasanya dijalankan di instans layanan tunggal (canary). Lalu, Anda meneliti secara mendalam setiap perubahan perilaku atau kesalahan yang dihasilkan. Anda dapat menghapus lalu lintas dari canary jika menemui masalah kritis dan mengembalikan pengguna ke versi sebelumnya. Jika deployment berhasil, Anda dapat melanjutkan melakukan deployment pada kecepatan yang diinginkan, sambil memantau perubahan kesalahan, hingga deployment sudah dilakukan sepenuhnya. AWS CodeDeploy dapat dikonfigurasi dengan konfigurasi deployment yang memungkinkan deployment canary.

Deployment blue/green: Serupa dengan deployment canary, tetapi di sini deployment armada penuh aplikasi dilakukan secara paralel. Anda mengubah deployment di dua tumpukan (blue dan green). Sekali lagi, Anda mengirimkan lalu lintas ke versi baru, dan kembali ke versi lama jika Anda melihat masalah dengan deployment. Biasanya semua lalu lintas dialihkan sekaligus, tetapi Anda juga dapat menggunakan sebagian lalu lintas ke setiap versi untuk meningkatkan adopsi versi baru menggunakan kemampuan perutean DNS tertimbang dari Amazon Route 53. AWS CodeDeploy dan AWS Elastic Beanstalk dapat dikonfigurasikan dengan konfigurasi deployment yang memungkinkan deployment blue/green.



Gambar 8: Deployment blue/green dengan AWS Elastic Beanstalk dan Amazon Route 53

Deteksi penyimpangan

Penyimpangan didefinisikan sebagai perubahan apa pun yang menyebabkan sumber daya infrastruktur memiliki status atau konfigurasi yang berbeda dengan apa yang diharapkan. Setiap jenis perubahan konfigurasi yang tidak dikelola bertentangan dengan gagasan infrastruktur tetap, dan harus dideteksi dan diperbaiki agar infrastruktur tetap berhasil diimplementasikan.

Langkah implementasi

- Larang modifikasi di tempat pada sumber daya infrastruktur yang sedang berjalan.
  - Anda dapat menggunakan AWS Identity and Access Management (IAM) untuk menentukan siapa atau apa yang dapat mengakses layanan dan sumber daya di AWS, mengelola izin dengan ketat secara terpusat, dan menganalisis akses untuk menyempurnakan izin di AWS.
- Otomatiskan deployment sumber daya infrastruktur untuk meningkatkan keterulangan dan meminimalkan potensi kesalahan manusia.
  - Seperti yang dijelaskan dalam laporan resmi Pengantar DevOps di AWS, otomatisasi merupakan landasan dalam layanan AWS dan didukung secara internal di semua layanan, fitur, dan penawaran.
  - Melakukan prapembuatan Amazon Machine Image (AMI) Anda dapat mempercepat waktu peluncurannya. EC2 Image Builder adalah layanan AWS yang dikelola sepenuhnya yang membantu Anda mengotomatiskan pembuatan, pemeliharaan, validasi, berbagi, dan deployment AMI kustom Linux atau Windows yang disesuaikan, aman, dan terbaru.
  - Beberapa layanan yang mendukung otomatisasi adalah:
    - AWS Elastic Beanstalk adalah layanan yang digunakan untuk dengan cepat melakukan deployment dan menskalakan aplikasi web yang dikembangkan dengan Java, .NET, PHP, Node.js, Python, Ruby, Go, dan Docker pada server yang sudah dikenal seperti Apache, NGINX, Passenger, dan IIS.
    - AWS Proton membantu tim platform menghubungkan dan mengoordinasikan semua alat berbeda yang dibutuhkan tim pengembangan Anda untuk penyediaan infrastruktur, deployment kode, pemantauan, dan pembaruan. AWS Proton memungkinkan penyediaan infrastruktur sebagai kode dan deployment aplikasi nirserver dan berbasis kontainer secara otomatis.
  - Memanfaatkan infrastruktur sebagai kode memudahkan otomatisasi deployment infrastruktur, dan membantu mewujudkan ketetapan infrastruktur. AWS menyediakan layanan yang memungkinkan pembuatan, deployment, dan pemeliharaan infrastruktur dengan cara yang terprogram, deskriptif, dan deklaratif.

    - AWS CloudFormation membantu developer membuat sumber daya AWS dengan cara yang teratur dan dapat diprediksi. Sumber daya ditulis dalam file teks menggunakan format JSON atau YAML. Templat memerlukan sintaks dan struktur tertentu yang bergantung pada jenis sumber daya yang dibuat dan dikelola. Anda menulis sumber daya Anda di JSON atau YAML dengan editor kode apa pun seperti AWS Cloud9, memeriksanya ke dalam sistem kontrol versi, dan kemudian CloudFormation membangun layanan yang ditentukan dengan cara yang aman dan dapat diulang.
    - AWS Serverless Application Model(AWS SAM) adalah kerangka kerja sumber terbuka yang dapat Anda gunakan untuk membangun aplikasi nirserver di AWS. AWS SAM terintegrasi dengan layanan AWS lainnya, dan merupakan pengembangan dari AWS CloudFormation.
    - AWS Cloud Development Kit (AWS CDK) adalah kerangka pengembangan perangkat lunak sumber terbuka untuk membuat model dan menyediakan sumber daya aplikasi cloud Anda menggunakan bahasa pemrograman yang sudah dipahami. Anda dapat menggunakan AWS CDK untuk membuat model infrastruktur aplikasi menggunakan TypeScript, Python, Java, dan .NET. AWS CDK menggunakan AWS CloudFormation di latar belakang untuk menyediakan sumber daya dengan cara yang aman dan dapat diulang.
    - AWS Cloud Control API memperkenalkan seperangkat API yang umum yaitu Membuat, Membaca, Memperbarui, Menghapus, dan Mencantumkan (CRUDL) untuk membantu developer mengelola infrastruktur cloud dengan mudah dan konsisten. API umum Cloud Control API memungkinkan developer untuk mengelola siklus hidup layanan AWS dan pihak ketiga secara seragam.
- Implementasikan pola deployment yang meminimalkan dampak pengguna.
    - Deployment canary:
        - Set up an API Gateway canary release deployment
        - Create a pipeline with canary deployments for Amazon ECS using AWS App Mesh
    - Deployment blue/green: laporan resmi Deployment Blue/Green di AWS menjelaskan contoh teknik dalam mengimplementasikan strategi deployment blue/green.
- Deteksi konfigurasi atau penyimpangan status. Untuk detail selengkapnya, lihat Detecting unmanaged configuration changes to stacks and resources.

Sumber daya

Praktik Terbaik Terkait:

- REL08-BP05 Melakukan deployment perubahan dengan otomatisasi

Dokumen terkait:

- Mengotomatiskan deployment aman tanpa campur tangan
- Leveraging AWS CloudFormation to create an immutable infrastructure at Nubank
- Infrastruktur sebagai kode
- Implementing an alarm to automatically detect drift in AWS CloudFormation stacks

Video terkait:

- AWS re:Invent 2020: Reliability, consistency, and confidence through immutability

REL08-BP05 Melakukan deployment perubahan dengan otomatisasi

Deployment dan patching diotomatisasi untuk menyingkirkan dampak negatif.

Membuat perubahan pada sistem produksi adalah salah satu area risiko terbesar untuk banyak organisasi. Kami menganggap deployment sebagai masalah kelas pertama untuk diatasi bersamaan dengan masalah-masalah bisnis yang ditangani oleh perangkat lunak. Saat ini, ini berarti penggunaan otomatisasi kapan saja memungkinkan dalam operasi, termasuk untuk menguji dan melakukan deployment perubahan, menambah atau menghapus kapasitas, dan memigrasikan data.

Hasil yang diinginkan: Anda membangun keamanan deployment otomatis ke dalam proses rilis dengan pengujian praproduksi yang ekstensif, rollback otomatis, dan deployment produksi bertahap. Otomatisasi ini meminimalkan potensi dampak pada produksi yang disebabkan oleh deployment yang gagal, dan developer tidak perlu lagi mengawasi deployment ke produksi secara aktif.

Antipola umum:

- Anda melakukan perubahan manual.
- Anda melewatkan langkah-langkah dalam otomatisasi Anda melalui alur kerja darurat manual.
- Anda tidak mengikuti rencana dan proses yang telah ditetapkan demi mempercepat jadwal.
- Anda melakukan deployment susulan cepat tanpa menyediakan waktu menanam.

Manfaat menjalankan praktik terbaik ini: Ketika Anda menggunakan otomatisasi untuk melakukan deployment semua perubahan, Anda menghilangkan potensi kesalahan manusia dan memberikan kemampuan untuk menguji sebelum Anda mengubah produksi. Melakukan proses ini sebelum deployment di produksi dapat memverifikasi bahwa rencana Anda sudah lengkap. Selain itu, rollback

otomatis ke dalam proses rilis Anda dapat mengidentifikasi masalah produksi dan mengembalikan beban kerja Anda ke keadaan operasional yang diketahui berfungsi sebelumnya.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Otomatiskan pipeline deployment Anda. Pipeline deployment memungkinkan Anda untuk memanggil pengujian dan deteksi anomali secara otomatis, serta memberi Anda pilihan untuk menghentikan pipeline pada langkah tertentu sebelum deployment produksi atau membatalkan perubahan secara otomatis. Bagian yang tidak terpisahkan dari hal ini adalah adopsi budaya integrasi berkelanjutan dan pengiriman/deployment berkelanjutan (CI/CD), di mana commit atau perubahan kode melewati berbagai gerbang tahap otomatis, mulai dari tahap build dan pengujian hingga deployment di lingkungan produksi.

Meskipun kebijaksanaan konvensional menyarankan Anda untuk melibatkan personel untuk prosedur operasional paling sulit, kami justru menyarankan Anda mengotomatiskan prosedur paling sulit untuk alasan tersebut.

Langkah implementasi

Anda dapat mengotomatiskan deployment untuk menghapus operasi manual dengan mengikuti langkah-langkah berikut:

- Siapkan repositori kode untuk menyimpan kode Anda dengan aman: Gunakan AWS CodeCommit, untuk membuat repositori berbasis Git yang aman.
- Buat konfigurasi layanan integrasi berkelanjutan untuk mengompilasi kode sumber Anda, menjalankan pengujian, dan membuat artefak deployment: Untuk menyiapkan proyek build untuk tujuan ini, lihat Mulai menggunakan AWS CodeBuild menggunakan konsol.
- Siapkan layanan deployment yang mengotomatiskan deployment aplikasi dan menangani kompleksitas pembaruan aplikasi tanpa bergantung pada deployment manual yang rawan kesalahan: AWS CodeDeploy mengotomatiskan deployment perangkat lunak ke berbagai layanan komputasi, seperti Amazon EC2, AWS Fargate, AWS Lambda, dan server on-premise Anda. Untuk mengonfigurasi langkah-langkah ini, lihat Mulai menggunakan CodeDeploy.
- Siapkan layanan pengiriman berkelanjutan yang mengotomatiskan pipeline rilis Anda untuk pembaruan aplikasi dan infrastruktur yang lebih cepat dan andal: Pertimbangkan AWS CodePipeline untuk membantu Anda mengotomatiskan pipeline rilis Anda. Untuk detail lebih lanjut, lihat Tutorial CodePipeline.

Sumber daya

Praktik terbaik terkait:

- OPS05-BP04 Menggunakan sistem manajemen build dan deployment
- OPS05-BP10 Mengotomatiskan integrasi dan deployment sepenuhnya
- OPS06-BP02 Menguji deployment
- OPS06-BP04 Mengotomatiskan pengujian dan pengembalian (rollback)

Dokumen terkait:

- Continuous Delivery of Nested AWS CloudFormation Stacks Using AWS CodePipeline
- Complete CI/CD with AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, and AWS CodePipeline
- Partner APN: partner yang dapat membantu Anda membuat solusi deployment yang diotomatiskan
- AWS Marketplace: produk yang dapat digunakan untuk mengotomatiskan deployment Anda
- Otomatiskan pesan obrolan dengan webhook.
- Amazon Builders' Library: Memastikan keamanan rollback selama deployment
- Amazon Builders' Library: Melaju lebih cepat dengan pengiriman berkelanjutan
- Apa Itu AWS CodePipeline?
- Apa Itu CodeDeploy?
- AWS Systems Manager Patch Manager
- Apa itu Amazon SES?
- Apa itu Amazon Simple Notification Service?

Video terkait:

- AWS Summit 2019: CI/CD on AWS

Manajemen kegagalan

> Kegagalan tidak bisa dihindari dan semua hal akan mengalami kegagalan seiring berjalannya waktu: mulai dari ruter hingga hard disk, dari sistem operasi hingga unit memori yang membuat TCP korup, dari kesalahan sementara hingga kegagalan permanen. Ini tidak bisa dihindari, meskipun Anda menggunakan perangkat keras berkualitas tinggi, apalagi komponen dengan biaya termurah - Werner Vogels, CTO - Amazon.com

Kegagalan komponen perangkat keras tingkat rendah menjadi hal yang harus dihadapi setiap hari di pusat data on-premise. Namun, di cloud, Anda harus terlindungi dari sebagian besar jenis kegagalan ini. Misalnya, volume Amazon EBS ditempatkan di Zona Ketersediaan tertentu dan direplikasi secara otomatis di dalamnya guna melindungi Anda dari kegagalan komponen tunggal. Semua volume EBS dirancang untuk ketersediaan 99,999%. Objek Amazon S3 disimpan di minimal tiga Zona Ketersediaan, menyediakan ketahanan objek sebesar 99,999999999% selama satu tahun. Terlepas dari penyedia cloud Anda, potensi kegagalan pasti ada dan berdampak pada beban kerja Anda. Oleh karena itu, Anda harus mengambil langkah untuk mengimplementasikan ketangguhan jika Anda membutuhkan beban kerja yang tangguh.

Prasyarat untuk menerapkan praktik terbaik yang didiskusikan di sini adalah Anda harus memastikan bahwa orang-orang yang merancang, mengimplementasikan, dan mengoperasikan beban kerja Anda paham tentang tujuan bisnis dan tujuan keandalan untuk mencapai hal ini. Mereka harus paham dan dilatih untuk persyaratan keandalan ini.

Bagian ini menjelaskan praktik terbaik untuk mengelola kegagalan guna mencegah dampak pada beban kerja Anda.

Topik

Cadangkan data

Cadangkan data, aplikasi, dan konfigurasi untuk memenuhi persyaratan untuk sasaran waktu pemulihan (RTO) dan sasaran titik pemulihan (RPO).

Praktik Terbaik

- REL09-BP01 Mengidentifikasi dan mencadangkan data yang perlu dicadangkan, atau memproduksi ulang data dari sumber
- REL09-BP02 Mengamankan dan mengenkripsikan cadangan
- REL09-BP03 Melakukan pencadangan data secara otomatis.
- REL09-BP04 Melakukan pemulihan data secara berkala untuk memverifikasi integritas dan proses pencadangan

REL09-BP01 Mengidentifikasi dan mencadangkan data yang perlu dicadangkan, atau memproduksi ulang data dari sumber

Pahami dan gunakan kemampuan pencadangan sumber daya dan layanan data yang digunakan oleh beban kerja. Sebagian besar layanan menyediakan kemampuan untuk mencadangkan data beban kerja.

Hasil yang diinginkan: Sumber data telah diidentifikasi dan diklasifikasikan berdasarkan tingkat kekritisan. Lalu, bangun strategi untuk pemulihan data berdasarkan RPO. Strategi ini melibatkan pencadangan sumber-sumber data, atau memiliki kemampuan untuk memproduksi ulang data dari sumber lain. Untuk kasus kehilangan data, strategi yang diimplementasikan memungkinkan pemulihan atau produksi ulang data dalam RPO dan RTO yang ditetapkan.

Fase kemapanan cloud: Fondasi

Antipola umum:

- Tidak mengetahui semua sumber data untuk beban kerja serta tingkat kekritisannya.
- Tidak melakukan pencadangan sumber data kritis.
- Melakukan pencadangan hanya beberapa sumber data tanpa menggunakan tingkat kekritisan sebagai kriteria.
- Tidak ada RPO yang ditetapkan, atau frekuensi pencadangan tidak memenuhi RPO.

- Tidak mengevaluasi apakah cadangan diperlukan atau apakah data dapat diproduksi ulang dari sumber lain.

Manfaat menjalankan praktik terbaik ini: Mengidentifikasi tempat-tempat yang memerlukan pencadangan dan mengimplementasikan mekanisme untuk membuat cadangan, atau mampu memproduksi ulang data dari sumber eksternal, semuanya dapat meningkatkan kemampuan untuk memulihkan dan mengembalikan data selama penghentian.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Semua penyimpanan data AWS menawarkan kemampuan pencadangan. Layanan seperti Amazon RDS dan Amazon DynamoDB memberikan dukungan tambahan pada pencadangan otomatis yang memungkinkan pemulihan titik waktu (PITR), yang memungkinkan Anda untuk memulihkan cadangan ke waktu kapan pun hingga lima menit atau kurang sebelum waktu saat ini. Banyak layanan AWS yang menawarkan kemampuan untuk menyalin cadangan ke Wilayah AWS lain. AWS Backup adalah alat yang memberikan kepada Anda kemampuan untuk memusatkan dan mengotomatiskan perlindungan data di layanan AWS. AWS Elastic Disaster Recovery memungkinkan Anda menyalin beban kerja server penuh dan mempertahankan perlindungan data berkelanjutan dari on-premise, lintas AZ, atau lintas Wilayah, dengan Sasaran Titik Pemulihan (RPO) yang diukur dalam detik.

Amazon S3 dapat digunakan sebagai tujuan pencadangan untuk sumber daya yang dikelola mandiri dan yang dikelola oleh AWS. Layanan AWS seperti Amazon EBS, Amazon RDS, dan Amazon DynamoDB memiliki kemampuan bawaan untuk membuat cadangan. Perangkat lunak pencadangan pihak ketiga juga dapat digunakan.

Data on-premise dapat dicadangkan ke AWS Cloud menggunakan AWS Storage Gateway atau AWS DataSync. Bucket Amazon S3 dapat digunakan untuk menyimpan data ini di AWS. Amazon S3 menawarkan beberapa tingkat penyimpanan seperti Amazon S3 Glacier atau S3 Glacier Deep Archive untuk mengurangi biaya penyimpanan data.

Anda mungkin dapat memenuhi kebutuhan pemulihan data dengan memproduksi ulang data dari sumber lain. Contohnya, simpul replika Amazon ElastiCache atau replika baca Amazon RDS dapat digunakan untuk memproduksi ulang data jika yang data utama hilang. Jika sumber seperti ini dapat digunakan untuk memenuhi Sasaran Titik Pemulihan (RPO) dan Sasaran Waktu Pemulihan (RTO), Anda mungkin tidak memerlukan cadangan. Contoh lainnya, jika bekerja dengan Amazon EMR, pencadangan penyimpanan data HDFS Anda mungkin tidak diperlukan, selama Anda dapat memproduksi ulang data ke Amazon EMR dari Amazon S3.

Ketika menyeleksi strategi pencadangan, pertimbangkan waktu yang diperlukan untuk memulihkan data. Waktu yang diperlukan untuk memulihkan data tergantung pada tipe cadangan (untuk kasus strategi pencadangan), atau kompleksitas mekanisme produksi ulang data. Waktu ini termasuk dalam RTO untuk beban kerja.

Langkah implementasi

1. Mengidentifikasi semua sumber data untuk beban kerja. Data dapat disimpan di sejumlah sumber daya seperti basis data, volume, sistem file, sistem pencatatan log, dan penyimpanan objek. Lihat bagian Sumber Daya untuk menemukan Dokumen terkait tentang berbagai layanan AWS tempat data disimpan, dan kemampuan pencadangan yang disediakan oleh layanan-layanan ini.
2. Klasifikasikan sumber data berdasarkan tingkat kekritisan. Set data yang berbeda akan memiliki tingkat kekritisan yang berbeda untuk suatu beban kerja, sehingga memiliki persyaratan untuk ketahanan yang berbeda-beda. Misalnya, beberapa data mungkin kritis dan memerlukan RPO hampir nol, sedangkan data lain mungkin tidak terlalu kritis dan dapat mentoleransi RPO yang lebih tinggi dan beberapa hilang data. Demikian juga, set data yang berbeda mungkin memiliki persyaratan RTO yang berbeda.
3. Gunakan AWS atau layanan pihak ketiga untuk membuat cadangan data. AWS Backup adalah layanan terkelola yang memungkinkan pembuatan cadangan berbagai sumber data di AWS. AWS Elastic Disaster Recovery menangani replikasi data otomatis sub-detik ke Wilayah AWS. Sebagian besar layanan AWS juga memiliki kemampuan native untuk membuat cadangan. AWS Marketplace juga memiliki banyak solusi untuk menyediakan kemampuan-kemampuan ini. Lihat Sumber Daya yang disebutkan di bawah ini untuk mendapatkan informasi tentang cara membuat cadangan data dari berbagai layanan AWS.
4. Untuk data yang tidak dicadangkan, buat mekanisme produksi ulang data. Anda mungkin memilih untuk tidak mencadangkan data yang dapat diproduksi ulang dari sumber lain karena berbagai alasan. Mungkin terdapat situasi di mana produksi ulang data dari sumber lain saat diperlukan lebih murah daripada membuat cadangan, karena mungkin ada biaya terkait penyimpanan cadangan. Contoh lainnya adalah ketika pemulihan dari cadangan memerlukan waktu lebih lama daripada produksi ulang data dari sumber lain, sehingga mengakibatkan pelanggaran RTO. Pada situasi-situasi demikian, pertimbangkan semua kompromi dan bangun proses yang ditetapkan dengan baik terkait bagaimana data dapat diproduksi ulang dari sumber-sumber ini saat pemulihan data diperlukan. Misalnya, jika Anda telah memuat data dari Amazon S3 ke gudang data (seperti Amazon Redshift), atau klaster MapReduce (seperti Amazon EMR) untuk melakukan analisis pada data tersebut, ini mungkin adalah contoh data yang dapat diproduksi ulang dari sumber lain. Selama hasil dari semua analisis ini disimpan di suatu tempat atau dapat diproduksi ulang, Anda tidak akan mengalami kehilangan data akibat kegagalan pada gudang data atau klaster

MapReduce. Contoh lain data yang dapat diproduksi ulang dari sumber lain adalah cache (seperti Amazon ElastiCache) atau replika baca RDS.

5. Buat jadwal rutin pencadangan data. Membuat cadangan sumber data adalah proses berkala dan frekuensinya tergantung pada RPO.

Tingkat upaya untuk rencana implementasi: Sedang

Sumber daya

Praktik Terbaik Terkait:

REL13-BP01 Tetapkan sasaran pemulihan untuk waktu henti dan kehilangan data

REL13-BP02 Menggunakan strategi pemulihan yang ditentukan untuk memenuhi sasaran pemulihan

Dokumen terkait:

- Apa Itu AWS Backup?
- Apa itu AWS DataSync?
- Apa itu Gateway Volume?
- Partner APN: partner yang dapat membantu terkait pencadangan
- AWS Marketplace: produk yang dapat digunakan untuk pencadangan
- Amazon EBS Snapshots
- Mencadangkan Amazon EFS
- Mencadangkan Amazon FSx untuk Windows File Server
- Pencadangan dan Pemulihan untuk ElastiCache for Redis
- Membuat Snapshot Klaster DB di Neptune
- Membuat Snapshot DB
- Membuat Aturan EventBridge yang Memicu Berdasarkan Jadwal
- Replika Lintas-Wilayah dengan Amazon S3
- EFS-ke-EFS AWS Backup
- Mengekspor Data Log ke Amazon S3
- Manajemen siklus hidup objek

- Pemulihan dan Pencadangan Sesuai Permintaan untuk DynamoDB
- Pemulihan titik waktu untuk DynamoDB
- Bekerja dengan Snapshot Indeks Amazon OpenSearch Service
- Apa itu AWS Elastic Disaster Recovery?

Video terkait:

- AWS re:Invent 2021 - Pencadangan, pemulihan bencana, dan perlindungan ransomware dengan AWS
- Demo AWS Backup: Pencadangan Lintas Akun dan Lintas Wilayah
- AWS re:Invent 2019: Memahami AWS Backup, dengan Rackspace (STG341)

Contoh terkait:

- Well-Architected Lab - Mengimplementasikan Replikasi Lintas Wilayah (CRR) Dua Arah untuk Amazon S3
- Well-Architected Lab - Pengujian Pencadangan dan Pemulihan Data
- Well-Architected Lab: Pencadangan dan Pemulihan dengan Failback untuk Beban Kerja Analitik
- Well-Architected Lab: Pemulihan Bencana - Pencadangan dan Pemulihan

REL09-BP02 Mengamankan dan mengenkripsikan cadangan

Kontrol dan deteksi akses ke cadangan menggunakan autentikasi dan otorisasi. Gunakan enkripsi untuk mencegah dan mendeteksi jika integritas data cadangan terancam.

Antipola umum:

- Memiliki akses yang sama ke cadangan dan otomatisasi pemulihan seperti yang dilakukan pada data.
- Tidak mengenkripsi cadangan.

Manfaat menjalankan praktik terbaik ini: Mengamankan cadangan Anda akan mencegah gangguan terhadap data, dan enkripsi data mencegah akses ke data tersebut jika tidak sengaja terekspos.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Kontrol dan deteksi akses ke cadangan menggunakan autentikasi dan otorisasi seperti AWS Identity and Access Management (IAM). Gunakan enkripsi untuk mencegah dan mendeteksi jika integritas data cadangan terancam.

Amazon S3 mendukung beberapa metode enkripsi data diam. Dengan menggunakan enkripsi di sisi server, Amazon S3 menerima objek sebagai data yang tidak terenkripsi dan mengenkripsinya saat disimpan. Dengan menggunakan enkripsi di sisi klien, aplikasi beban kerja bertanggung jawab untuk mengenkripsi data sebelum mengirimkannya ke Amazon S3. Kedua metode tersebut memungkinkan Anda untuk menggunakan AWS Key Management Service (AWS KMS) guna menciptakan dan menyimpan kunci data. Anda dapat menyediakan kunci Anda sendiri dan bertanggung jawab atas kunci tersebut. Dengan menggunakan AWS KMS, Anda dapat menetapkan kebijakan menggunakan IAM terkait siapa yang dapat dan tidak dapat mengakses kunci data dan data terdekripsi.

Untuk Amazon RDS, cadangan juga akan dienkripsi jika Anda memilih untuk mengenkripsikan basis data. Cadangan DynamoDB selalu terenkripsi. Ketika menggunakan AWS Elastic Disaster Recovery, semua data bergerak dan data diam dienkripsi. Dengan Elastic Disaster Recovery, data diam dapat dienkripsi menggunakan Kunci Enkripsi Volume enkripsi Amazon EBS atau kunci kustom yang dikelola pelanggan.

Langkah implementasi

1. Gunakan enkripsi untuk setiap penyimpanan data. Jika sumber data terenkripsi, maka cadangannya juga akan terenkripsi.
   - Gunakan enkripsi di Amazon RDS.. Anda dapat mengonfigurasi enkripsi diam menggunakan AWS Key Management Service saat membuat instans RDS.
   - Gunakan enkripsi di volume Amazon EBS.. Anda dapat mengonfigurasi enkripsi default atau menentukan kunci unik saat pembuatan volume.
   - Gunakan enkripsi Amazon DynamoDB yang diperlukan. DynamoDB mengenkripsi semua data diam. Anda dapat menggunakan kunci AWS KMS yang dimiliki AWS atau kunci KMS yang dikelola AWS, menentukan kunci yang disimpan di akun Anda.
   - Enkripsikan data yang disimpan di Amazon EFS. Konfigurasikan enkripsi saat Anda membuat sistem file.
   - Konfigurasikan enkripsi di Wilayah sumber dan tujuan. Anda dapat mengonfigurasi enkripsi diam di Amazon S3 menggunakan kunci yang disimpan di KMS, tetapi kuncinya bersifat spesifik Wilayah. Anda dapat menentukan kunci tujuan saat mengonfigurasi replikasi.

- Pilih apakah akan menggunakan enkripsi Amazon EBS default atau kustom untuk Elastic Disaster Recovery. Opsi ini akan mengenkripsi data diam yang direplikasi di disk Subnet Area Staging dan disk yang direplikasi.

2. Implementasikan izin hak akses paling rendah untuk mengakses cadangan. Ikuti praktik terbaik untuk membatasi akses ke cadangan, snapshot, dan replika sesuai dengan praktik terbaik keamanan.

Sumber daya

Dokumen terkait:

- AWS Marketplace: produk yang dapat digunakan untuk pencadangan
- Enkripsi Amazon EBS
- Amazon S3: Melindungi Data Menggunakan Enkripsi
- Konfigurasi Tambahan CRR: Mereplikasi Objek yang Dibuat dengan Enkripsi di Sisi Server (SSE) Menggunakan Kunci Enkripsi yang disimpan di AWS KMS
- Enkripsi Diam DynamoDB
- Mengenkripsi Sumber Daya Amazon RDS
- Mengenkripsi Data dan Metadata di Amazon EFS
- Enkripsi untuk Cadangan di AWS
- Mengelola Tabel Terenkripsi
- Pilar Keamanan - AWS Well-Architected Framework
- Apa itu AWS Elastic Disaster Recovery?

Contoh terkait:

- Well-Architected Lab - Mengimplementasikan Replikasi Lintas Wilayah (CRR) Dua Arah untuk Amazon S3

REL09-BP03 Melakukan pencadangan data secara otomatis.

Konfigurasikan pencadangan untuk dilakukan secara otomatis berdasarkan jadwal berkala mengacu pada Sasaran Titik Pemulihan (RPO), atau berdasarkan perubahan dalam set data. Set data kritis dengan persyaratan data hilang yang rendah perlu dicadangkan otomatis secara rutin, sedangkan

data yang tidak terlalu kritis di mana beberapa data hilang masih dapat diterima dapat dicadangkan tidak terlalu sering.

Hasil yang diinginkan: Proses otomatis yang membuat cadangan sumber data dengan jadwal yang ditetapkan.

Antipola umum:

- Melakukan pencadangan secara manual.
- Menggunakan sumber daya yang memiliki kemampuan pencadangan, tetapi tidak termasuk pencadangan dalam otomatisasi Anda.

Manfaat menjalankan praktik terbaik ini: Otomatisasi pencadangan memverifikasi pencadangan dilakukan secara teratur berdasarkan RPO Anda dan memberi tahu Anda jika pencadangan tidak dilakukan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

AWS Backup dapat digunakan untuk membuat cadangan data otomatis untuk berbagai sumber data AWS. Instans Amazon RDS dapat dicadangkan hampir secara berkelanjutan setiap lima menit dan objek Amazon S3 dapat dicadangkan hampir secara berkelanjutan setiap lima belas menit, dan memungkinkan pemulihan titik waktu (PITR) ke titik waktu tertentu di dalam riwayat pencadangan. Untuk sumber data AWS lainnya, seperti volume Amazon EBS, tabel Amazon DynamoDB, atau sistem file Amazon FSx, AWS Backup dapat menjalankan pencadangan otomatis setiap satu jam. Layanan ini juga menawarkan kemampuan pencadangan native. Layanan AWS yang menawarkan pencadangan otomatis dengan pemulihan titik waktu antara lain Amazon DynamoDB, Amazon RDS, dan Amazon Keyspaces (untuk Apache Cassandra) - ini dapat dipulihkan ke titik waktu tertentu dalam riwayat pencadangan. Sebagian besar layanan penyimpanan data AWS lainnya menawarkan kemampuan untuk menjadwalkan pencadangan berkala, dengan frekuensi setiap satu jam.

Amazon RDS dan Amazon DynamoDB menawarkan pencadangan berkelanjutan dengan pemulihan titik waktu. Versioning Amazon S3, setelah diaktifkan, bersifat otomatis. Amazon Data Lifecycle Manager dapat digunakan untuk mengotomatiskan pembuatan, penyalinan, dan penghapusan snapshot Amazon EBS. Layanan ini juga dapat mengotomatiskan pembuatan, penyalinan, penghentian, dan pembatalan registrasi Amazon Machine Images (AMI) yang dicadangkan Amazon EBS dan snapshot Amazon EBS yang melandasinya.

AWS Elastic Disaster Recovery memberikan replikasi tingkat blok yang berkelanjutan dari lingkungan sumber (on-premise atau AWS) ke wilayah pemulihan target. Snapshot Amazon EBS titik waktu dibuat dan dikelola secara otomatis oleh layanan.

Untuk tampilan otomatisasi dan riwayat pencadangan terpusat, AWS Backup menyediakan solusi pencadangan berbasis kebijakan yang terkelola penuh. Layanan ini memusatkan dan mengotomatiskan pencadangan data di beberapa layanan AWS di cloud serta on-premise menggunakan AWS Storage Gateway.

Selain versioning, Amazon S3 dilengkapi dengan replikasi. Seluruh bucket S3 dapat direplikasi secara otomatis ke bucket lain di Wilayah AWS yang sama atau berbeda.

Langkah implementasi

1. Identifikasi sumber data yang saat ini dicadangkan secara manual. Untuk detail selengkapnya, lihat REL09-BP01 Mengidentifikasi dan mencadangkan data yang perlu dicadangkan, atau memproduksi ulang data dari sumber.
2. Tentukan RPO untuk beban kerja. Untuk detail selengkapnya, lihat REL13-BP01 Tetapkan sasaran pemulihan untuk waktu henti dan kehilangan data.
3. Gunakan solusi cadangan otomatis atau layanan terkelola. AWS Backup adalah layanan terkelola penuh yang mempermudah pemusatan dan pengotomatisan perlindungan data di seluruh layanan AWS, di cloud, dan on-premise. Dengan menggunakan rencana cadangan di AWS Backup, buat aturan yang menetapkan sumber daya yang akan dicadangkan, dan frekuensi pembuatan cadangan ini. Frekuensi ini harus mengacu pada RPO yang ditetapkan pada Langkah 2. Untuk panduan praktik langsung tentang cara membuat cadangan otomatis menggunakan AWS Backup, lihat Pengujian Pencadangan dan Pemulihan Data. Kemampuan pencadangan native ditawarkan oleh sebagian besar layanan AWS yang menyimpan data. Misalnya, RDS dapat dimanfaatkan untuk pencadangan otomatis dengan pemulihan titik waktu (PITR).
4. Untuk sumber daya yang tidak didukung oleh solusi pencadangan otomatis atau layanan terkelola seperti sumber data on-premise atau antrean pesan, pertimbangkan penggunaan solusi pihak ketiga tepercaya untuk membuat cadangan otomatis. Pilihan lainnya, Anda dapat membuat otomatisasi untuk melakukannya menggunakan AWS CLI atau SDK. Anda dapat menggunakan Fungsi AWS Lambda atau AWS Step Functions untuk menetapkan logika yang terlibat dalam pembuatan cadangan data, dan gunakan Amazon EventBridge untuk melaksanakannya dengan frekuensi yang didasarkan pada RPO Anda.

Tingkat upaya untuk Rencana Implementasi: Rendah

Sumber daya

Dokumen terkait:

- Partner APN: partner yang dapat membantu terkait pencadangan
- AWS Marketplace: produk yang dapat digunakan untuk pencadangan
- Membuat Aturan EventBridge yang Memicu Berdasarkan Jadwal
- Apa Itu AWS Backup?
- Apa Itu AWS Step Functions?
- Apa itu AWS Elastic Disaster Recovery?

Video terkait:

- AWS re:Invent 2019: Memahami AWS Backup, dengan Rackspace (STG341)

Contoh terkait:

- Well-Architected Lab - Pengujian Pencadangan dan Pemulihan Data

REL09-BP04 Melakukan pemulihan data secara berkala untuk memverifikasi integritas dan proses pencadangan

Validasikan bahwa implementasi proses pencadangan Anda memenuhi Sasaran Waktu Pemulihan (RTO) dan Sasaran Titik Pemulihan (RPO) dengan melakukan uji pemulihan.

Hasil yang diinginkan: Data dari cadangan dipulihkan secara berkala menggunakan mekanisme yang ditentukan dengan baik untuk memverifikasi bahwa pemulihan tersebut dapat dilakukan dalam sasaran waktu pemulihan (RTO) yang ditetapkan untuk beban kerja. Verifikasikan bahwa pemulihan dari pencadangan menghasilkan sumber daya yang berisi data asli tanpa ada data yang rusak atau tidak dapat diakses, serta dengan kehilangan data dalam sasaran titik pemulihan (RPO).

Antipola umum:

- Memulihkan cadangan, tetapi tidak mengambil data atau membuat kueri data apa pun untuk memastikan pemulihan dapat digunakan.
- Dengan anggapan bahwa cadangan sudah ada.

- Dengan anggapan bahwa cadangan sistem dapat dioperasikan sepenuhnya dan data dapat dipulihkan dari sistem.
- Dengan anggapan bahwa waktu untuk memulihkan data dari cadangan termasuk dalam RTO untuk beban kerja.
- Dengan anggapan bahwa data dalam cadangan termasuk dalam RPO untuk beban kerja.
- Memulihkan apabila diperlukan, tanpa menggunakan runbook, atau di luar prosedur otomatis yang ditetapkan.

Manfaat menjalankan praktik terbaik ini: Pengujian pemulihan cadangan memastikan data dapat dipulihkan saat dibutuhkan tanpa perlu khawatir data akan hilang atau rusak, bahwa restorasi dan pemulihan dapat dilakukan dalam batas RTO untuk beban kerja, dan kehilangan data apa pun termasuk dalam RPO untuk beban kerja.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Pengujian kemampuan pencadangan dan pemulihan meningkatkan keyakinan pada kemampuan untuk menjalankan tindakan ini selama pemadaman. Pulihkan cadangan ke lokasi baru secara berkala dan lakukan pengujian untuk memverifikasi integritas data. Beberapa pengujian umum yang harus dilakukan yakni, memeriksa apakah semua data tersedia, tidak rusak, dapat diakses, dan setiap kehilangan data termasuk dalam RPO untuk beban kerja. Pengujian tersebut dapat juga membantu memastikan apakah mekanisme pemulihan cukup cepat untuk mengakomodasi RTO beban kerja.

Dengan menggunakan AWS, Anda dapat mempertahankan lingkungan pengujian dan memulihkan cadangan untuk menilai kemampuan RTO dan RPO, serta menjalankan pengujian pada konten dan integritas data.

Selain itu, Amazon RDS dan Amazon DynamoDB memungkinkan pemulihan titik waktu (PITR). Dengan menggunakan pencadangan berkelanjutan, Anda dapat memulihkan set data ke statusnya pada waktu dan tanggal yang ditentukan.

apakah semua data tersedia, tidak rusak, dapat diakses, dan kehilangan data apa pun termasuk dalam RPO untuk beban kerja. Pengujian tersebut dapat juga membantu memastikan apakah mekanisme pemulihan cukup cepat untuk mengakomodasi RTO beban kerja.

AWS Elastic Disaster Recovery menawarkan snapshot pemulihan titik waktu volume Amazon EBS secara berkelanjutan. Saat server sumber direplikasi, status titik waktu dicatat seiring waktu

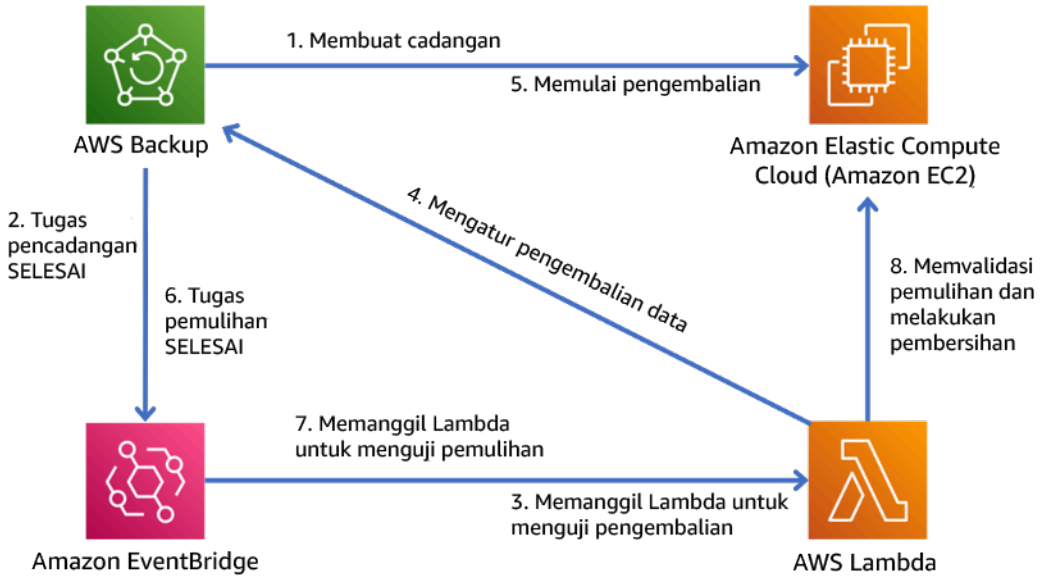
berdasarkan kebijakan yang dikonfigurasi. Elastic Disaster Recovery membantu Anda memverifikasi integritas snapshot ini dengan meluncurkan instans untuk tujuan pengujian dan latihan tanpa mengarahkan ulang lalu lintas.

Langkah implementasi

1. Identifikasi sumber data yang dicadangkan saat ini dan lokasi penyimpanan cadangan tersebut. Untuk panduan implementasi, lihat REL09-BP01 Mengidentifikasi dan mencadangkan data yang perlu dicadangkan, atau memproduksi ulang data dari sumber.
2. Tetapkan kriteria validasi data untuk setiap sumber data. Jenis data yang berbeda akan memiliki properti data yang berbeda, yang dapat memerlukan mekanisme validasi yang berbeda. Pertimbangkan bagaimana data ini dapat divalidasi sebelum Anda yakin untuk menggunakannya dalam produksi. Beberapa cara umum untuk memvalidasi adalah dengan menggunakan data dan properti pencadangan seperti jenis data, format, checksum, ukuran, atau kombinasi darinya dengan logika validasi kustom. Misalnya, hal ini dapat dilakukan dengan perbandingan nilai checksum antara sumber daya yang dipulihkan dan sumber data pada waktu cadangan dibuat.
3. Tetapkan RTO dan RPO untuk memulihkan data berdasarkan kekritisan data. Untuk panduan implementasi, lihat REL13-BP01 Tetapkan sasaran pemulihan untuk waktu henti dan kehilangan data.
4. Nilai kemampuan pemulihan Anda. Tinjau strategi pencadangan dan pemulihan untuk memahami apakah hal tersebut memenuhi RTO dan RPO, serta sesuaikan strategi yang dibutuhkan. Dengan menggunakan AWS Resilience Hub, Anda dapat menjalankan penilaian beban kerja. Penilaian tersebut mengevaluasi konfigurasi aplikasi terhadap kebijakan dan pelaporan ketahanan jika target RTO dan RPO dapat dipenuhi.
5. Lakukan pemulihan pengujian dengan menggunakan proses yang ditetapkan saat ini yang digunakan dalam produksi untuk pemulihan data. Proses ini bergantung pada cara sumber data asli dicadangkan, format dan lokasi penyimpanan cadangan tersebut, atau apakah data direproduksi dari sumber lainnya. Contohnya, jika Anda menggunakan layanan terkelola seperti AWS Backup, hal ini bisa sederhana seperti memulihkan cadangan ke sumber daya baru. Jika Anda menggunakan AWS Elastic Disaster Recovery, Anda dapat meluncurkan latihan pemulihan.
6. Validasikan pemulihan data dari sumber daya yang dipulihkan berdasarkan kriteria yang ditetapkan sebelumnya untuk validasi data. Apakah data yang direstorasi dan dipulihkan memiliki sebagian besar catatan atau item terbaru pada waktu pencadangan? Apakah data ini termasuk dalam RPO untuk beban kerja?
7. Ukur waktu yang diperlukan untuk restorasi dan pemulihan dan bandingkan dengan RTO yang telah Anda tetapkan. Apakah data ini termasuk dalam RTO untuk beban kerja? Misalnya,

bandingkan stempel waktu dari kapan proses pemulihan dimulai dan kapan validasi pemulihan selesai untuk menghitung waktu yang diperlukan proses ini. Semua panggilan API AWS diberi cap waktu dan informasi ini tersedia di AWS CloudTrail. Ketika informasi ini dapat menyediakan detail waktu kapan proses pemulihan dimulai, stempel waktu akhir untuk kapan validasi diselesaikan harus dicatat melalui logika validasi. Jika menggunakan proses otomatis, maka layanan seperti Amazon DynamoDB dapat digunakan untuk menyimpan informasi ini. Selain itu, banyak layanan AWS yang menyediakan riwayat peristiwa berisi informasi dengan stempel waktu tentang kapan tindakan diambil. Di dalam AWS Backup, tindakan pencadangan dan pemulihan disebut sebagai tugas, dan tugas tersebut berisi informasi cap waktu sebagai bagian dari metadata yang dapat digunakan untuk mengukur waktu yang diperlukan untuk restorasi dan pemulihan.

8. Beri notifikasi kepada para pemangku kepentingan jika validasi data gagal, atau jika waktu yang diperlukan untuk restorasi dan pemulihan melebihi RTO yang ditetapkan untuk beban kerja. Ketika mengimplementasikan otomatisasi untuk melakukan tindakan ini, seperti dalam lab ini, layanan seperti Amazon Simple Notification Service (Amazon SNS) dapat digunakan untuk mengirimkan notifikasi push seperti email atau SMS kepada para pemangku kepentingan. Pesan ini juga dapat dipublikasikan di aplikasi olahpesan seperti Amazon Chime, Slack, atau Microsoft Teams atau digunakan untuk membuat tugas sebagai OpsItems menggunakan AWS Systems Manager OpsCenter.
9. Otomatiskan proses ini untuk menjalankannya secara berkala. Misalnya, layanan seperti AWS Lambda atau State Machine di AWS Step Functions dapat digunakan untuk mengotomatiskan proses pemulihan, dan Amazon EventBridge dapat digunakan untuk memicu alur kerja otomatisasi ini secara berkala seperti yang ditampilkan dalam diagram arsitektur di bawah ini. Pelajar cara untuk Mengotomatiskan validasi pemulihan data dengan AWS Backup. Selain itu, Well-Architected lab ini memberikan pengalaman praktik langsung mengenai salah satu cara untuk melakukan otomatisasi untuk beberapa langkah di sini.



Gambar 9. Proses pencadangan dan pemulihan otomatis

Tingkat upaya untuk Rencana Implementasi: Sedang hingga tinggi, bergantung pada kompleksitas kriteria validasi.

Sumber daya

Dokumen terkait:

- Mengotomatiskan validasi pemulihan data dengan AWS Backup
- Partner APN: partner yang dapat membantu terkait pencadangan
- AWS Marketplace: produk yang dapat digunakan untuk pencadangan
- Membuat Aturan EventBridge yang Memicu Berdasarkan Jadwal
- Pemulihan dan pencadangan sesuai permintaan untuk DynamoDB
- Apa Itu AWS Backup?
- Apa Itu AWS Step Functions?
- Apa itu AWS Elastic Disaster Recovery
- AWS Elastic Disaster Recovery

Contoh terkait:

- Well-Architected Lab: Pengujian Pencadangan dan Pemulihan Data

Gunakan isolasi kesalahan untuk melindungi beban kerja Anda

Batas isolasi kesalahan membatasi efek kegagalan di dalam beban kerja untuk jumlah komponen yang terbatas. Komponen di luar batas ini tidak terpengaruh oleh kegagalan tersebut. Dengan beberapa batas isolasi kesalahan, Anda dapat membatasi dampak pada beban kerja Anda.

Praktik Terbaik

- REL10-BP01 Melakukan deployment beban kerja ke beberapa lokasi
- REL10-BP02 Memilih lokasi yang sesuai untuk deployment multilokasi
- REL10-BP03 Mengotomatiskan pemulihan untuk komponen yang dibatasi dalam satu lokasi
- REL10-BP04 Menggunakan arsitektur bulkhead untuk membatasi cakupan dampak

REL10-BP01 Melakukan deployment beban kerja ke beberapa lokasi

Distribusikan sumber daya dan data beban kerja ke beberapa Zona Ketersediaan atau, jika diperlukan, ke beberapa Wilayah AWS. Lokasi tersebut dapat beragam sesuai kebutuhan.

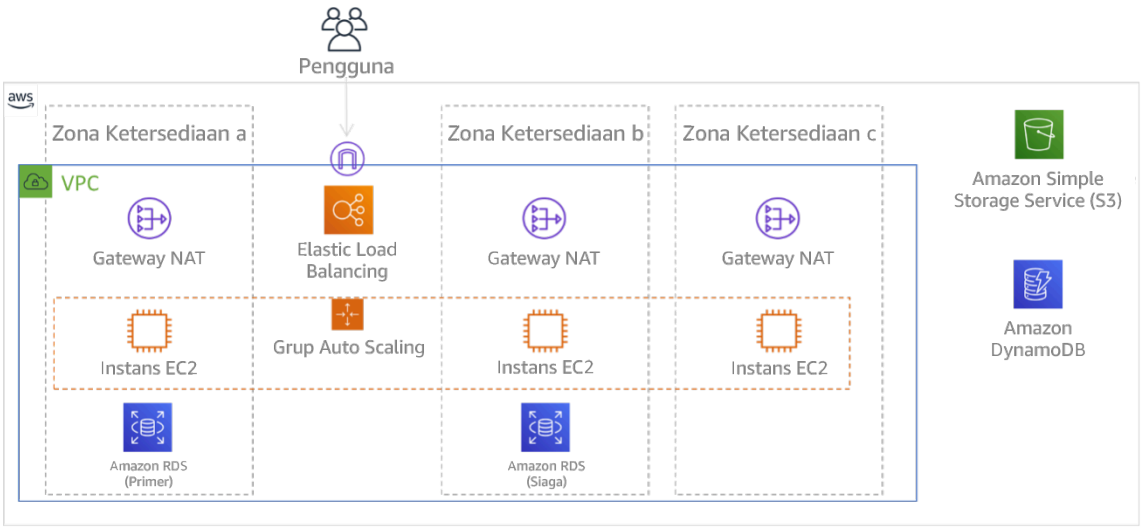
Salah satu prinsip dasar untuk desain layanan di AWS adalah menghindari titik kegagalan tunggal dalam infrastruktur fisik yang mendasarinya. Hal ini memotivasi kami untuk membangun sistem dan perangkat lunak yang menggunakan beberapa Zona Ketersediaan dan tahan terhadap kegagalan dari satu zona. Dengan cara yang serupa, sistem dibangun agar tahan terhadap kegagalan dari satu simpul komputasi, satu volume penyimpanan, atau satu instans basis data. Ketika membangun sistem yang mengandalkan komponen redundan, penting untuk memastikan bahwa komponen dapat beroperasi secara independen, dan dalam kasus Wilayah AWS, secara otomatis. Manfaat yang diperoleh dari kalkulasi ketersediaan teoretis dengan komponen redundan hanya valid jika dapat dibuktikan kebenarannya.

Zona Ketersediaan (AZ)

Wilayah AWS terdiri atas beberapa Zona Ketersediaan yang dirancang agar menjadi independen satu sama lain. Setiap Zona Ketersediaan dipisahkan oleh jarak fisik yang cukup dari zona lain untuk menghindari skenario kegagalan terkait karena bahaya lingkungan seperti kebakaran, banjir, dan tornado. Setiap Zona Ketersediaan juga memiliki infrastruktur fisik independen: koneksi khusus ke daya utilitas, sumber daya cadangan mandiri, layanan mekanis independen, dan konektivitas jaringan independen di dalam dan di luar Zona Ketersediaan. Desain ini membatasi kesalahan dalam satu sistem hingga hanya satu AZ yang terdampak. Meskipun terpisah secara geografis, Zona Ketersediaan berada di wilayah yang sama yang memungkinkan jaringan dengan latensi rendah dan

throughput tinggi. Seluruh Wilayah AWS (di semua Zona Ketersediaan, terdiri atas beberapa pusat data yang independen secara fisik) dapat dibuat menjadi target deployment logika tunggal untuk beban kerja, termasuk kemampuan untuk mereplikasi data secara sinkron (misalnya antarbasis data). Hal ini memungkinkan Anda untuk menggunakan Zona Ketersediaan dalam konfigurasi aktif/aktif atau aktif/siaga.

Zona Ketersediaan bersifat independen, dan oleh karena itu ketersediaan beban kerja meningkat saat beban kerja dirancang untuk menggunakan beberapa zona. Beberapa layanan AWS (termasuk bidang data instans Amazon EC2) di-deploy sebagai layanan zonal yang ketat dan memiliki sifat yang sama dengan Zona Ketersediaan tempatnya berada. Instans Amazon EC2 di AZ lainnya tidak akan terdampak dan tetap berfungsi. Dengan cara yang serupa, jika kesalahan di Zona Ketersediaan menyebabkan basis data Amazon Aurora gagal, instans Aurora replika baca di AZ yang tidak terdampak dapat dipindahkan ke AZ utama secara otomatis. Sebaliknya, layanan AWS regional seperti Amazon DynamoDB secara internal menggunakan beberapa Zona Ketersediaan dalam konfigurasi aktif/aktif guna mencapai tujuan desain ketersediaan untuk layanan tersebut, tanpa perlu mengonfigurasi penempatan AZ.



Gambar 9: Arsitektur multitingkat di-deploy di tiga Zona Ketersediaan. Perhatikan bahwa Amazon S3 dan Amazon DynamoDB selalu Multi-AZ secara otomatis. ELB juga di-deploy ke tiga zona.

Ketika umumnya bidang kendali AWS memberikan kemampuan untuk mengelola sumber daya di seluruh Wilayah (beberapa Zona Ketersediaan), bidang kendali tertentu (termasuk Amazon EC2 dan Amazon EBS) memiliki kemampuan untuk memfilter hasil hingga satu Zona Ketersediaan. Saat ini sudah dilakukan, permintaan hanya diproses di Zona Ketersediaan tertentu, mengurangi eksposur gangguan di Zona Ketersediaan lainnya. Contoh AWS CLI ini menggambarkan cara mendapatkan informasi instans Amazon EC2 hanya dari Zona Ketersediaan us-east-2c:

```
AWS ec2 describe-instances --filters Name=availability-zone,Values=us-east-2c
```

AWS Local Zones

AWS Local Zones bertindak serupa dengan Zona Ketersediaan dalam Wilayah AWS masing-masing sehingga dapat dipilih sebagai lokasi penempatan untuk sumber daya AWS zonal seperti subnet dan instans EC2. Hal yang membuatnya istimewa adalah mereka tidak berada di Wilayah AWS terkait, tetapi dekat dengan populasi yang besar, industri, dan pusat IT ketika tidak ada lagi Wilayah AWS. Namun zona-zona ini tetap mampu mempertahankan bandwidth tinggi, koneksi yang aman di antara beban kerja di zona lokal dan yang dijalankan di Wilayah AWS. Anda harus menggunakan AWS Local Zones untuk melakukan deployment beban kerja secara lebih dekat dengan pengguna untuk persyaratan latensi rendah.

Amazon Global Edge Network

Amazon Global Edge Network terdiri atas lokasi edge di kota seluruh dunia. Amazon CloudFront menggunakan jaringan ini untuk mengirimkan konten kepada pengguna akhir dengan latensi lebih rendah. AWS Global Accelerator memungkinkan Anda membuat titik akhir beban kerja di lokasi edge tersebut untuk memberikan onboarding ke jaringan global AWS yang dekat dengan pengguna. Amazon API Gateway memungkinkan titik akhir API yang dioptimasi edge menggunakan distribusi CloudFront agar klien mendapatkan akses melalui lokasi edge terdekat.

Wilayah AWS

Wilayah AWS dirancang agar menjadi otonom, akan tetapi, untuk menggunakan pendekatan multi-Wilayah, Anda perlu melakukan deployment salinan layanan yang dikhususkan untuk masing-masing Wilayah.

Pendekatan multi-Wilayah biasa digunakan untuk strategi pemulihan bencana yang memenuhi tujuan pemulihan saat satu peristiwa berskala besar terjadi. Lihat Rencanakan Pemulihan Bencana (DR) untuk informasi lebih lanjut tentang strategi ini. Namun, di sini kami lebih fokus pada ketersediaan, yang berupaya memberikan tujuan waktu aktif rata-rata dari waktu ke waktu. Untuk tujuan ketersediaan tinggi, arsitektur multi-wilayah umumnya akan dirancang menjadi aktif/aktif, dengan setiap salinan layanan (di wilayah masing-masing) yang aktif (permintaan layanan).

> **Rekomendasi**
>
> Tujuan ketersediaan untuk sebagian besar beban kerja dapat dipenuhi menggunakan strategi Multi-AZ dalam satu Wilayah AWS. Pertimbangkan arsitektur multi-Wilayah hanya saat beban

kerja memiliki persyaratan ketersediaan yang sangat tinggi, atau tujuan bisnis lain yang memerlukan arsitektur multi-Wilayah.

AWS memberikan kemampuan untuk mengoperasikan layanan lintas wilayah. Misalnya, AWS menyediakan replikasi data asinkron yang berkelanjutan menggunakan Replikasi Amazon Simple Storage Service (Amazon S3), Replika Baca Amazon RDS (termasuk Replika Baca Aurora), dan Tabel Global Amazon DynamoDB. Dengan replikasi berkelanjutan, versi data tersedia untuk penggunaan segera di setiap Wilayah aktif.

Dengan menggunakan AWS CloudFormation, Anda dapat menentukan infrastruktur dan melakukan deployment secara konsisten di seluruh Akun AWS dan seluruh Wilayah AWS. AWS CloudFormation StackSets meningkatkan fungsionalitas ini dengan memungkinkan Anda untuk membuat, memperbarui, atau menghapus tumpukan AWS CloudFormation di seluruh akun atau wilayah dalam satu kali operasi. Untuk deployment instans Amazon EC2, AMI (Amazon Machine Image) digunakan untuk memasok informasi seperti konfigurasi perangkat keras dan perangkat lunak yang diinstal. Anda dapat mengimplementasikan pipeline Amazon EC2 Image Builder yang membuat AMI yang diperlukan dan menyalinnya ke wilayah aktif. Hal ini memastikan AMI Emas memiliki segala yang dibutuhkan untuk melakukan deployment dan menskalakan beban kerja di setiap wilayah baru.

Untuk merutekan lalu lintas, Amazon Route 53 dan AWS Global Accelerator mengaktifkan definisi kebijakan yang menentukan titik akhir wilayah aktif yang dituju pengguna. Dengan Global Accelerator, Anda dapat mengatur panggilan lalu lintas untuk mengontrol persentase lalu lintas yang diarahkan ke setiap titik akhir aplikasi. Route 53 mendukung pendekatan persentase ini, dan juga beberapa kebijakan lain yang tersedia, termasuk kebijakan berdasarkan latensi dan geoproksimitas. Global Accelerator secara otomatis memanfaatkan jaringan server edge AWS yang luas, untuk mengarahkan lalu lintas ke pusat jaringan AWS secepatnya, sehingga menghasilkan latensi permintaan yang lebih rendah.

Semua kemampuan ini dioperasikan untuk menjaga setiap otonomi Wilayah. Ada beberapa pengecualian untuk pendekatan ini, termasuk layanan yang menyediakan pengiriman edge global, (seperti Amazon CloudFront dan Amazon Route 53), serta dengan bidang kendali untuk layanan AWS Identity and Access Management (IAM). Sebagian besar layanan dioperasikan sepenuhnya dalam satu Wilayah.

Pusat data on-premise

Rancang pengalaman hybrid jika memungkinkan, untuk beban kerja yang dijalankan di pusat data on-premise. AWS Direct Connect menyediakan koneksi jaringan khusus dari premise Anda ke AWS sehingga Anda dapat menjalankan beban kerja di kedua sistem.

Opsi lainnya adalah menjalankan layanan dan infrastruktur AWS on-premise dengan menggunakan AWS Outposts. AWS Outposts adalah layanan terkelola penuh yang memperluas infrastruktur AWS, layanan AWS, API, dan alat untuk pusat data. Infrastruktur perangkat keras yang sama yang digunakan di AWS Cloud juga diinstal di pusat data. Selanjutnya, AWS Outposts dihubungkan ke Wilayah AWS yang terdekat. Anda dapat menggunakan AWS Outposts untuk mendukung beban kerja yang memiliki latensi rendah atau persyaratan pemrosesan data lokal.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Tinggi

Panduan implementasi

- Gunakan beberapa Zona Ketersediaan dan Wilayah AWS. Distribusikan sumber daya dan data beban kerja ke beberapa Zona Ketersediaan atau, jika diperlukan, ke beberapa Wilayah AWS. Lokasi tersebut dapat beragam sesuai kebutuhan.
  - Layanan regional di-deploy secara permanen di seluruh Zona Ketersediaan.
    - Ini termasuk Amazon S3, Amazon DynamoDB, dan AWS Lambda (saat tidak terhubung ke VPC)
  - Lakukan deployment kontainer, instans, dan beban kerja berdasarkan fungsi ke dalam beberapa Zona Ketersediaan. Gunakan penyimpanan data multi-zona, termasuk cache. Gunakan fitur EC2 Auto Scaling, penempatan tugas ECS, dan konfigurasi fungsi AWS Lambda saat menjalankan VPC dan klaster ElastiCache.
    - Gunakan subnet yang berada di Zona Ketersediaan terpisah saat melakukan deployment grup Auto Scaling.
      - Misalnya: Mendistribusikan instans di seluruh Zona Ketersediaan
      - Strategi penempatan tugas Amazon ECS
      - Mengonfigurasi fungsi AWS Lambda untuk mengakses sumber daya di Amazon VPC
      - Memilih Zona Ketersediaan dan Wilayah
    - Gunakan subnet di Zona Ketersediaan terpisah saat melakukan deployment grup Auto Scaling.
      - Misalnya: Mendistribusikan instans di seluruh Zona Ketersediaan
    - Gunakan parameter penempatan tugas ECS, yang menentukan grup subnet DB.
      - Strategi penempatan tugas Amazon ECS

    - Gunakan subnet di beberapa Zona Ketersediaan saat mengonfigurasi fungsi untuk dijalankan di VPC.
        - Mengonfigurasi fungsi AWS Lambda untuk mengakses sumber daya di Amazon VPC
    - Gunakan beberapa Zona Ketersediaan dengan klaster ElastiCache.
        - Memilih Zona Ketersediaan dan Wilayah
- Jika beban kerja harus di-deploy di beberapa Wilayah, pilih strategi multi-Wilayah. Sebagian besar kebutuhan keandalan dapat dipenuhi dalam satu Wilayah AWS menggunakan strategi multi-Zona Ketersediaan. Gunakan strategi multi-Wilayah jika diperlukan untuk memenuhi kebutuhan bisnis.
    - AWS re:Invent 2018: Pola Arsitektur untuk Aplikasi Aktif-Aktif Multi-Wilayah (ARC209-R2)
        - Mencadangkan ke Wilayah AWS lain dapat menambah lapisan jaminan lain bahwa data akan tersedia saat dibutuhkan.
        - Beberapa beban kerja memiliki persyaratan regulasi yang memerlukan penggunaan strategi multi-Wilayah.
- Evaluasi AWS Outposts untuk beban kerja. Jika beban kerja memerlukan latensi rendah ke pusat data on-premise atau memiliki persyaratan pemrosesan data lokal. Jalankan layanan dan infrastruktur AWS on premise menggunakan AWS Outposts
    - Apa itu AWS Outposts?
- Tentukan apakah AWS Local Zones membantu menyediakan layanan untuk pengguna. Jika Anda memiliki persyaratan latensi rendah, periksa apakah AWS Local Zones berada dekat dengan pengguna. Jika iya, manfaatkan hal tersebut untuk melakukan deployment beban kerja dengan lebih dekat ke pengguna tersebut.
    - Pertanyaan Umum AWS Local Zones

Sumber daya

Dokumen terkait:

- Infrastruktur Global AWS
- Pertanyaan Umum AWS Local Zones
- Strategi penempatan tugas Amazon ECS
- Memilih Zona Ketersediaan dan Wilayah
- Misalnya: Mendistribusikan instans di seluruh Zona Ketersediaan
- Tabel Global: Replikasi Multi-Wilayah dengan DynamoDB

- Menggunakan basis data Amazon Aurora
- Membuat Aplikasi Multi-Wilayah dengan seri blog Layanan AWS
- Apa itu AWS Outposts?

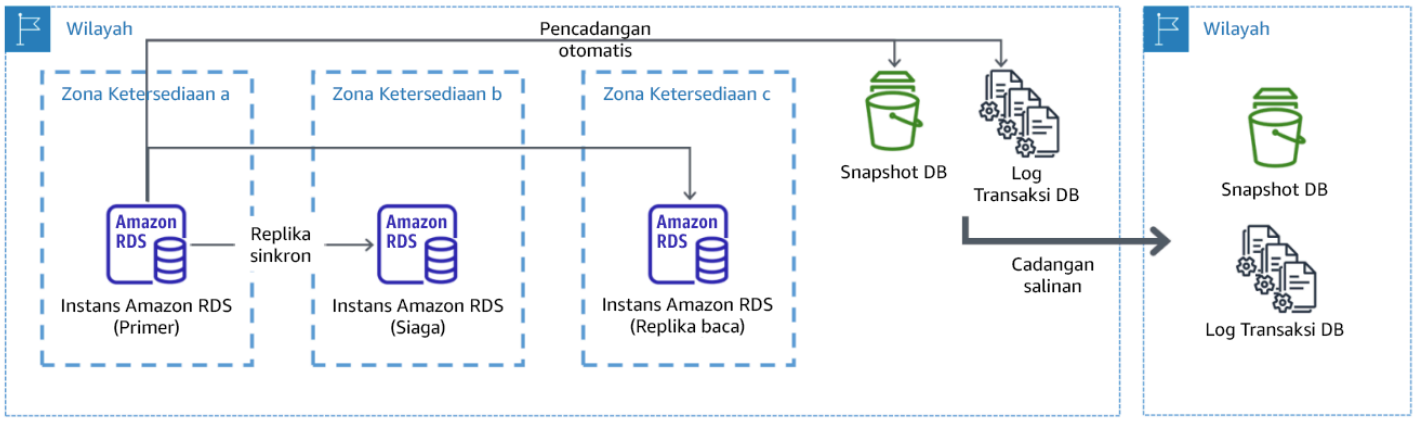
Video terkait:

- AWS re:Invent 2018: Pola Arsitektur untuk Aplikasi Aktif-Aktif Multi-Wilayah (ARC209-R2)
- AWS re:Invent 2019: Inovasi dan operasi infrastruktur jaringan global AWS (NET339)

REL10-BP02 Memilih lokasi yang sesuai untuk deployment multilokasi

Hasil yang diinginkan:

Untuk ketersediaan tinggi, selalu (jika memungkinkan) lakukan deployment komponen beban kerja ke beberapa Zona Ketersediaan (AZ), seperti yang ditampilkan dalam Gambar 10. Untuk beban kerja dengan persyaratan ketahanan yang sangat tinggi, evaluasikan dengan cermat opsi untuk arsitektur multi-Wilayah.



Gambar 10: Deployment basis data multi-AZ yang tangguh dengan pencadangan ke Wilayah AWS lainnya

Antipola umum:

- Memilih untuk merancang arsitektur multi-Wilayah saat arsitektur multi-AZ dapat memenuhi persyaratan.
- Tidak memperhitungkan dependensi antarkomponen aplikasi jika ketahanan dan persyaratan multilokasi antarkomponen tersebut berbeda.

Manfaat menerapkan praktik terbaik ini:

Untuk ketahanan, Anda harus menggunakan pendekatan yang membangun lapisan pertahanan. Satu lapisan melindungi terhadap gangguan yang lebih kecil dan lebih umum dengan membangun arsitektur yang memiliki ketersediaan tinggi menggunakan beberapa AZ. Lapisan pertahanan lainnya ditujukan untuk memberikan perlindungan terhadap peristiwa langka seperti bencana alam yang meluas dan gangguan tingkat Wilayah. Lapisan kedua ini melibatkan perancangan aplikasi agar menjangkau beberapa Wilayah AWS.

- Perbedaan antara ketersediaan 99,5% dan ketersediaan 99,99% adalah lebih dari 3,5 jam per bulan. Ketersediaan beban kerja yang diharapkan hanya dapat mencapai “empat angka sembilan” jika berada dalam beberapa AZ.
- Dengan menjalankan beban kerja di beberapa AZ, Anda dapat mengisolasi kesalahan dalam daya, pendinginan, dan jaringan, serta sebagian besar bencana alam seperti kebakaran dan banjir.
- Mengimplementasikan strategi multi-Wilayah untuk beban kerja membantu melindunginya dari bencana alam yang menjangkau dan memengaruhi wilayah geografis yang luas di suatu negara, atau kesalahan teknis yang mencakup seluruh Wilayah. Perhatikan bahwa mengimplementasikan arsitektur multi-Wilayah dapat menjadi sangat kompleks, dan biasanya tidak diperlukan untuk sebagian besar beban kerja.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Tinggi

Panduan implementasi

Untuk peristiwa bencana yang didasarkan pada gangguan atau hilangnya sebagian dari satu Zona Ketersediaan, mengimplementasikan beban kerja yang memiliki ketersediaan tinggi di beberapa Zona Ketersediaan dalam satu Wilayah AWS dapat membantu mitigasi bencana alam dan teknis. Setiap Wilayah AWS terdiri atas beberapa Zona Ketersediaan, masing-masing diisolasi dari kesalahan di zona lain dan dipisahkan oleh jarak yang cukup. Namun, untuk peristiwa bencana yang menyertakan risiko hilangnya beberapa komponen Zona Ketersediaan, yang jaraknya cukup jauh satu sama lain, Anda harus mengimplementasikan opsi pemulihan bencana untuk memitigasi kesalahan dalam cakupan Wilayah. Untuk beban kerja yang memerlukan ketahanan sangat tinggi (infrastruktur yang sangat penting, aplikasi terkait kesehatan, infrastruktur sistem keuangan, dll.), strategi multi-Wilayah mungkin diperlukan.

Langkah Implementasi

1. Evaluasikan beban kerja dan tentukan apakah ketahanan yang diperlukan dapat dipenuhi oleh pendekatan multi-AZ (satu Wilayah AWS), atau apakah pendekatan multi-Wilayah diperlukan. Mengimplementasikan arsitektur multi-Wilayah untuk memenuhi persyaratan tersebut akan menimbulkan kompleksitas tambahan, dengan demikian pertimbangkan secara cermat kasus penggunaan Anda dan persyaratannya. Persyaratan ketahanan dapat hampir selalu dipenuhi menggunakan satu Wilayah AWS. Pertimbangkan persyaratan yang memungkinkan berikut saat menentukan apakah Anda perlu menggunakan beberapa Wilayah:
   a. Pemulihan Bencana (DR): Untuk peristiwa bencana yang didasarkan pada gangguan atau kehilangan sebagian dari satu Zona Ketersediaan, mengimplementasikan beban kerja yang memiliki ketersediaan tinggi di beberapa Zona Ketersediaan dalam satu Wilayah AWS dapat membantu mitigasi bencana alam dan teknis. Untuk peristiwa bencana yang menyertakan risiko kehilangan beberapa komponen Zona Ketersediaan, yang jaraknya cukup jauh satu sama lain, Anda harus mengimplementasikan pemulihan bencana di seluruh Wilayah untuk memitigasi bencana alam atau kesalahan teknis dalam cakupan Wilayah.
   b. Ketersediaan tinggi (HA): Arsitektur multi-Wilayah (menggunakan beberapa AZ di setiap Wilayah) dapat digunakan untuk mencapai ketersediaan yang lebih tinggi dari empat angka 9 (> 99,99%).
   c. Pelokalan tumpukan: Saat melakukan deployment beban kerja ke audiens global, Anda dapat melakukan deployment tumpukan yang dilokalkan di Wilayah AWS yang berbeda untuk melayani audiens di Wilayah tersebut. Pelokalan dapat mencakup bahasa, mata uang, dan jenis data yang disimpan.
   d. Proksimitas kepada pengguna: Saat melakukan deployment beban kerja ke audiens global, Anda dapat mengurangi latensi dengan melakukan deployment tumpukan di Wilayah AWS yang dekat dengan tempat pengguna akhir.
   e. Residensi data: Beberapa beban kerja bergantung pada persyaratan residensi data, ketika data dari pengguna tertentu harus tetap berada dalam batasan negara tertentu. Berdasarkan regulasi dalam pertanyaan, Anda dapat memilih untuk melakukan deployment seluruh tumpukan, atau datanya saja, ke Wilayah AWS dalam batas tersebut.
2. Berikut beberapa contoh fungsionalitas multi-AZ yang disediakan oleh layanan AWS:
   a. Untuk melindungi beban kerja menggunakan EC2 atau ECS, lakukan deployment Elastic Load Balancer di depan sumber daya komputasi. Selanjutnya, Elastic Load Balancing menyediakan solusi untuk mendeteksi instans di zona yang kondisinya tidak baik dan merutekan lalu lintas ke zona yang kondisinya baik.

   i. Mulai menggunakan Application Load Balancers

   ii. Mulai menggunakan Penyeimbang Beban Jaringan

b. Dalam kasus instans EC2 yang menjalankan perangkat lunak komersial siap pakai yang tidak mendukung penyeimbangan beban, Anda dapat mencapai bentuk toleransi kesalahan dengan mengimplementasikan metodologi pemulihan bencana multi-AZ.

   i. the section called “REL13-BP02 Menggunakan strategi pemulihan yang ditentukan untuk memenuhi sasaran pemulihan”

c. Untuk tugas Amazon ECS, lakukan deployment secara merata di tiga AZ untuk mencapai keseimbangan ketersediaan dan biaya.

   i. Praktik terbaik ketersediaan Amazon ECS | Kontainer

d. Untuk non-Aurora Amazon RDS, Anda dapat memilih Multi-AZ sebagai opsi konfigurasi. Saat instans basis data utama mengalami kegagalan, Amazon RDS secara otomatis mendorong basis data standby untuk menerima lalu lintas di zona ketersediaan lainnya. Replika baca multi-Wilayah juga dapat dibuat untuk meningkatkan ketahanan.

   i. Deployment Multi AZ Amazon RDS

   ii. Membuat replika baca di Wilayah AWS yang berbeda

3. Berikut beberapa contoh fungsionalitas multi-Wilayah yang disediakan oleh layanan AWS:

a. Untuk beban kerja Amazon S3, ketika ketersediaan multi-AZ disediakan secara otomatis oleh layanan, pertimbangkan Poin Akses Multi-Wilayah jika deployment multi-Wilayah diperlukan.

   i. Poin Akses Multi-Wilayah di Amazon S3

b. Untuk tabel DynamoDB, ketika ketersediaan multi-AZ disediakan secara otomatis oleh layanan, Anda dapat mengonversi tabel yang ada ke tabel global dengan mudah untuk memperoleh manfaat dari beberapa wilayah.

   i. Konversi Tabel Amazon DynamoDB Wilayah Tunggal menjadi Tabel Global

c. Jika beban kerja didahului oleh Application Load Balancers atau Penyeimbang Beban Jaringan, gunakan AWS Global Accelerator untuk meningkatkan ketersediaan aplikasi dengan mengarahkan lalu lintas ke beberapa wilayah yang memiliki titik akhir dengan kondisi baik.

   i. Titik akhir untuk akselerator standar di AWS Global Accelerator - AWS Global Accelerator (amazon.com)

d. Untuk aplikasi yang memanfaatkan AWS EventBridge, pertimbangkan bus lintas Wilayah untuk meneruskan peristiwa ke Wilayah lain yang dipilih.

   i. Mengirim dan menerima peristiwa Amazon EventBridge di antara beberapa Wilayah AWS

e. Untuk basis data Amazon Aurora, pertimbangkan basis data global Aurora, yang menjangkau beberapa wilayah AWS. Klaster yang sudah ada juga dapat diubah untuk menambahkan Wilayah baru.
   i. Mulai menggunakan basis data global Amazon Aurora
f. Jika beban kerja mencakup kunci enkripsi AWS Key Management Service (AWS KMS), pertimbangkan apakah kunci multi-Wilayah sesuai untuk aplikasi.
   i. Kunci Multi-Wilayah di AWS KMS
g. Untuk fitur layanan AWS lainnya, lihat seri blog ini di Seri Membuat Aplikasi Multi-Wilayah dengan Layanan AWS

Tingkat upaya untuk Rencana Implementasi: Sedang hingga Tinggi

Sumber daya

Dokumen terkait:

- Seri Membuat Aplikasi Multi-Wilayah dengan Layanan AWS
- Arsitektur Pemulihan Bencana (DR) di AWS, Bagian IV: Multi-situs Aktif/Aktif)
- Infrastruktur Global AWS
- Pertanyaan Umum AWS Local Zones
- Arsitektur Pemulihan Bencana (DR) di AWS, Bagian I: Strategi untuk Pemulihan di Cloud
- Pemulihan bencana di cloud tidak sama dengan biasanya
- Tabel Global: Replikasi Multi-Wilayah dengan DynamoDB

Video terkait:

- AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications (ARC209-R2)
- Auth0: Arsitektur Ketersediaan Tinggi Multi-Wilayah yang Menskalakan hingga 1,5B+ Login Sebulan dengan failover otomatis

Contoh terkait:

- Arsitektur Pemulihan Bencana (DR) di AWS, Bagian I: Strategi untuk Pemulihan di Cloud
- DTCC mencapai ketangguhan yang lebih tinggi dari yang dapat dilakukan di on-premise

- Expedia Group menggunakan arsitektur multi-Wilayah dan multi-Zona Ketersediaan dengan layanan DNS eksklusif untuk menambah ketangguhan pada aplikasi
- Uber: Pemulihan Bencana untuk Kafka Multi-Wilayah
- Netflix: Aktif-Aktif untuk Ketahanan Multi-Wilayah
- Cara kami membangun Residensi Data untuk Atlassian Cloud
- Intuit TurboTax dijalankan di dua Wilayah

REL10-BP03 Mengotomatiskan pemulihan untuk komponen yang dibatasi dalam satu lokasi

Jika komponen beban kerja hanya dapat dijalankan di satu Zona Ketersediaan atau di pusat data on-premise, implementasikan kemampuan untuk membangun kembali beban kerja sepenuhnya dalam lingkup tujuan pemulihan yang telah ditetapkan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Jika praktik terbaik untuk melakukan deployment beban kerja ke beberapa lokasi tidak memungkinkan karena pembatasan teknologi, Anda harus mengimplementasikan jalur alternatif menuju ketahanan. Anda harus mengotomatiskan kemampuan untuk membuat ulang infrastruktur yang dibutuhkan, melakukan deployment ulang aplikasi, dan membuat ulang data yang diperlukan untuk kasus ini.

Misalnya, Amazon EMR meluncurkan semua simpul untuk klaster tertentu yang tersedia dalam Zona Ketersediaan yang sama karena menjalankan klaster di zona yang sama dapat meningkatkan kinerja aliran tugas berkat tingkat akses data yang lebih tinggi. Jika komponen ini tidak dibutuhkan untuk ketahanan beban kerja, Anda harus mencari cara lain untuk melakukan deployment klaster dan datanya. Selain itu, untuk Amazon EMR, Anda harus menyediakan redundansi selain dengan menggunakan Multi-AZ. Anda dapat menyediakan beberapa simpul. Dengan menggunakan Sistem File EMR (EMRFS), data di EMR dapat dipulihkan di Amazon S3, yang kemudian dapat direplikasi di beberapa Zona Ketersediaan atau Wilayah AWS.

Dengan cara yang serupa, Amazon Redshift secara default menyediakan klaster dalam Zona Ketersediaan yang dipilih secara acak dalam Wilayah AWS yang Anda pilih. Semua simpul klaster disediakan dalam zona yang sama.

Untuk beban kerja stateful berbasis server yang di-deploy ke pusat data on-premise, Anda dapat menggunakan AWS Elastic Disaster Recovery untuk melindungi beban kerja Anda di AWS. Jika Anda sudah di-host di AWS, Anda dapat menggunakan Elastic Disaster Recovery untuk melindungi beban kerja Anda di Wilayah atau Zona Ketersediaan alternatif. Elastic Disaster Recovery menggunakan replikasi tingkat blok secara berkelanjutan ke area staging ringan untuk memberikan pemulihan aplikasi on-premise dan aplikasi berbasis cloud secara cepat dan andal.

Langkah implementasi

1. Implementasikan pemulihan mandiri. Lakukan deployment instans atau kontainer dengan menggunakan penskalaan otomatis jika memungkinkan. Jika penskalaan otomatis tidak dapat digunakan, gunakan pemulihan otomatis untuk instans EC2 atau implementasikan otomatisasi pemulihan mandiri berdasarkan Amazon EC2 atau peristiwa siklus hidup kontainer ECS.
   - Gunakan grup Amazon EC2 Auto Scaling untuk instans atau beban kerja kontainer yang tidak memiliki persyaratan untuk alamat IP instans tunggal, alamat IP pribadi, alamat IP Elastis, dan metadata instans.
     - Data pengguna templat peluncuran dapat digunakan untuk mengimplementasikan otomatisasi yang dapat memulihkan sebagian besar beban kerja secara mandiri.
   - Gunakan pemulihan otomatis instans Amazon EC2untuk beban kerja yang memerlukan instans tunggal alamat ID, alamat IP pribadi, alamat IP elastis, dan instans metadata.
     - Pemulihan Otomatis akan mengirimkan peringatan status pemulihan kepada topik SNS saat kegagalan instans terdeteksi.
   - Gunakan peristiwa siklus hidup instans Amazon EC2 atau peristiwa Amazon ECS untuk mengotomatiskan pemulihan mandiri jika penskalaan otomatis atau pemulihan EC2 tidak dapat digunakan.
     - Gunakan peristiwa untuk memicu otomatisasi yang akan memulihkan komponen Anda berdasarkan proses logika yang diperlukan.
   - Lindungi beban kerja stateful yang dibatasi di satu lokasi menggunakan AWS Elastic Disaster Recovery.

Sumber daya

Dokumen terkait:

- Peristiwa Amazon ECS
- Pengait siklus hidup Amazon EC2 Auto Scaling

- Pulihkan instans Anda.
- Penskalaan otomatis layanan
- Apa Itu Amazon EC2 Auto Scaling?
- AWS Elastic Disaster Recovery

REL10-BP04 Menggunakan arsitektur bulkhead untuk membatasi cakupan dampak

Implementasikan arsitektur bulkhead (juga disebut sebagai arsitektur berbasis sel) untuk membatasi efek kegagalan dalam beban kerja hingga jumlah komponen yang terbatas.

Hasil yang diinginkan: Arsitektur berbasis sel menggunakan beberapa instans terisolasi beban kerja, di mana setiap instans disebut sebagai sel. Setiap sel bersifat mandiri, tidak berbagi status dengan sel lain, dan menangani subset permintaan beban kerja secara keseluruhan. Hal ini mengurangi potensi dampak kegagalan, seperti pembaruan perangkat lunak yang buruk, ke satu sel individu dan permintaan yang diprosesnya. Jika beban kerja menggunakan 10 sel untuk melayani 100 permintaan, ketika kegagalan terjadi, 90% dari seluruh permintaan akan tidak dipengaruhi oleh kegagalan.

Antipola umum:

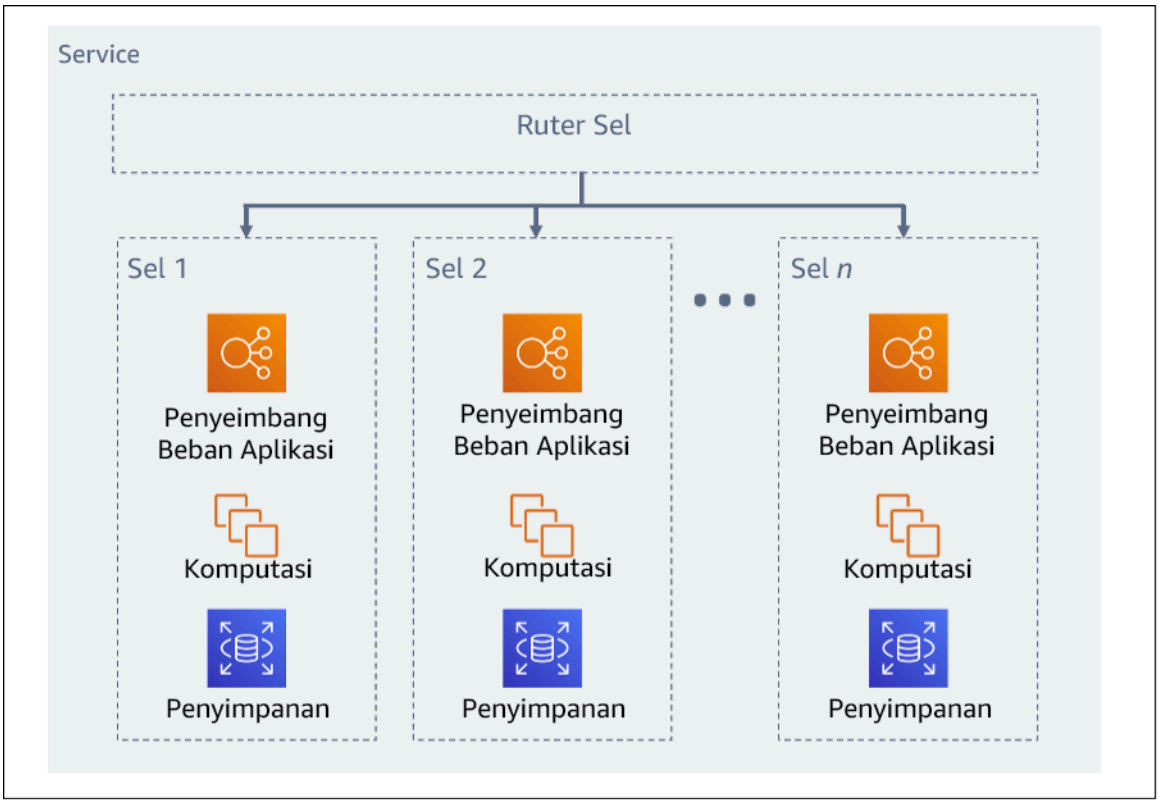
- Membiarkan sel bertumbuh tanpa batas.
- Menerapkan pembaruan kode atau deployment ke semua sel pada waktu yang sama.
- Berbagi status atau komponen antara sel (dengan pengecualian lapisan router).
- Menambahkan bisnis yang kompleks atau mengarahkan rute logika ke lapisan router.
- Tidak meminimalkan interaksi lintas sel.

Manfaat menjalankan praktik terbaik ini: Dengan arsitektur berbasis sel, banyak jenis kegagalan umum dibatasi dalam lingkup sel itu sendiri, yang memberikan isolasi kesalahan tambahan. Batas kesalahan ini dapat memberikan ketangguhan terhadap jenis kegagalan yang sulit dibatasi, seperti deployment kode yang gagal atau permintaan yang rusak atau memicu mode kegagalan tertentu (juga disebut sebagai permintaan poison pill).

Panduan implementasi

Di kapal, bulkhead memastikan kebocoran lambung kapal hanya dibatasi dalam satu bagian lambung kapal saja. Di sistem yang kompleks, pola ini sering kali direplikasi untuk memungkinkan isolasi kesalahan. Batas isolasi kesalahan membatasi efek kegagalan di dalam beban kerja hingga jumlah komponen yang terbatas. Komponen di luar batas ini tidak terpengaruh oleh kegagalan tersebut. Menggunakan beberapa batas isolasi kesalahan, Anda dapat membatasi dampak pada beban kerja Anda. Di AWS, pelanggan dapat menggunakan beberapa Zona Ketersediaan dan Wilayah untuk memberikan isolasi kesalahan, tetapi konsep isolasi kesalahan dapat diperluas ke arsitektur beban kerja juga.

Beban kerja secara keseluruhan adalah sel-sel yang dipartisi oleh kunci partisi. Kunci ini harus sesuai dengan grain layanan, atau cara alami beban kerja layanan dapat dibagi lebih lanjut dengan interaksi lintas sel yang minim. Contoh kunci partisi yakni ID pelanggan, ID sumber daya, atau parameter lainnya yang dapat diakses dengan mudah dalam sebagian besar panggilan API. Lapisan perutean sel mendistribusikan permintaan ke masing-masing sel berdasarkan kunci partisi dan menyampaikan satu titik akhir ke klien.



Gambar 11: Arsitektur berbasis sel

Langkah implementasi

Ketika mendesain arsitektur berbasis sel, ada beberapa pertimbangan desain untuk dipikirkan:

1. Kunci partisi: Pemilihan kunci partisi harus dipertimbangkan baik-baik.
    - Kunci ini harus sesuai dengan grain layanan, atau cara alami beban kerja layanan dapat dibagi lebih lanjut dengan interaksi lintas sel yang minim. Contohnya, `ID pelanggan` atau `ID sumber daya`.
    - Kunci partisi harus tersedia dalam semua permintaan, baik secara langsung atau dengan cara yang dapat disimpulkan dengan pasti dan mudah oleh parameter lain.
2. Pemetaan sel persisten: Layanan sebelumnya dalam proses hanya boleh berinteraksi dengan satu sel selama siklus hidup sumber dayanya.
    - Bergantung pada beban kerjanya, strategi migrasi sel mungkin diperlukan untuk memigrasikan data dari satu sel ke yang lain. Kemungkinan skenario ketika migrasi sel mungkin diperlukan yakni jika sumber daya atau pengguna tertentu di beban kerja Anda menjadi terlalu besar dan memerlukan sel khusus.
    - Sel tidak boleh berbagi status atau komponen antara sel.
    - Oleh karena itu, interaksi lintas sel harus dihindari atau dijaga agar tetap minim, karena interaksi tersebut menimbulkan dependensi antara sel, sehingga mengurangi peningkatan isolasi kesalahan.
3. Lapisan router: Lapisan router adalah komponen bersama antara sel, oleh karena itu tidak dapat mengikuti strategi kompartementalisasi yang sama seperti sel.
    - Sebaiknya lapisan router mendistribusikan permintaan ke sel secara individu menggunakan algoritme pemetaan partisi dengan cara yang efisien secara komputasi, seperti menggabungkan fungsi hash kriptografi dan aritmetika modul untuk memetakan kunci partisi ke sel.
    - Untuk menghindari dampak multi-sel, lapisan perutean harus tetap sesederhana mungkin dan dapat diskalakan sehorizontal mungkin, yang memerlukan penghindaran logika bisnis kompleks di dalam lapisan ini. Hal ini memiliki manfaat tambahan mempermudah pemahaman ekspektasi perilakunya di setiap waktu, yang memberikan kemampuan untuk diuji secara menyeluruh. Sebagaimana dijelaskan oleh Colm MacCárthaigh dalam Reliability, constant work, and a good cup of coffee, desain sederhana dan pola kerja konstan menghasilkan sistem yang andal dan mengurangi anti-kerentanan.
4. Ukuran sel: Sel harus memiliki ukuran maksimum dan tidak boleh diizinkan untuk bertumbuh melampauinya.
    - Ukuran maksimum harus diidentifikasi dengan melakukan pengujian yang menyeluruh, sampai titik rusak tercapai dan margin pengoperasian yang aman ditetapkan. Untuk detail selengkapnya

tentang cara mengimplementasikan praktik pengujian, lihat REL07-BP04 Menguji beban untuk beban kerja Anda

- Beban kerja secara keseluruhan harus bertumbuh dengan menambahkan sel tambahan, sehingga beban kerja dapat diskalakan seiring peningkatan permintaan.

5. Strategi Multi-AZ atau Multi-Wilayah: Beberapa lapisan ketangguhan harus dimanfaatkan untuk melindungi dari berbagai macam domain kegagalan.
   - Untuk ketahanan, Anda harus menggunakan pendekatan yang membangun lapisan pertahanan. Satu lapisan melindungi dari gangguan yang lebih kecil dan lebih umum dengan membangun arsitektur yang memiliki ketersediaan tinggi menggunakan beberapa AZ. Lapisan pertahanan lainnya ditujukan untuk memberikan perlindungan terhadap peristiwa langka seperti bencana alam yang meluas dan gangguan tingkat Wilayah. Lapisan kedua ini melibatkan perancangan aplikasi agar menjangkau beberapa Wilayah AWS. Mengimplementasikan strategi multi-Wilayah untuk beban kerja membantu melindunginya dari bencana alam yang menjangkau dan memengaruhi wilayah geografis yang luas di suatu negara, atau kesalahan teknis yang mencakup seluruh Wilayah. Perhatikan bahwa mengimplementasikan arsitektur multi-Wilayah dapat menjadi sangat kompleks, dan biasanya tidak diperlukan untuk sebagian besar beban kerja. untuk detail selengkapnya, lihat REL10-BP02 Memilih lokasi yang sesuai untuk deployment multilokasi.
6. Deployment kode: Strategi deployment kode bergiliran harus didahulukan dibandingkan deployment perubahan kode ke semua sel pada waktu yang sama.
   - Hal ini akan membantu meminimalkan potensi kegagalan pada beberapa sel karena deployment yang buruk atau kesalahan manusia. Untuk detail selengkapnya, lihat Mengotomatiskan deployment aman tanpa campur tangan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Sumber daya

Praktik Terbaik Terkait:

- REL07-BP04 Menguji beban untuk beban kerja Anda
- REL10-BP02 Memilih lokasi yang sesuai untuk deployment multilokasi

Dokumen terkait:

- Reliability, constant work, and a good cup of coffee

- AWS dan Kompartementalisasi
- Isolasi beban kerja menggunakan shuffle-sharding
- Mengotomatiskan deployment aman tanpa campur tangan

Video terkait:

- AWS re:Invent 2018: Menutup Lingkaran dan Membuka Pikiran: Cara Mengendalikan Sistem, Besar dan Kecil
- AWS re:Invent 2018: Cara AWS Meminimalkan Radius Dampak Kegagalan (ARC338)
- Shuffle-sharding: AWS re:Invent 2019: Memperkenalkan Pustaka Pengembang Amazon (DOP328)
- AWS Summit ANZ 2021 - Segala sesuatu gagal, setiap waktu: Mendesain agar memiliki ketangguhan

Contoh terkait:

- Well-Architected Lab - Isolasi kesalahan dengan shuffle sharding

Rancang beban kerja Anda agar bertahan dalam kegagalan komponen

Beban kerja dengan persyaratan ketersediaan tinggi dan waktu rata-rata untuk pemulihan (MTTR) rendah harus dirancang agar tangguh.

Praktik Terbaik

- REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan
- REL11-BP02 Melakukan failover ke sumber daya yang sehat
- REL11-BP03 Mengotomatisasi pemulihan di semua lapisan
- REL11-BP04 Mengandalkan bidang data dan bukan bidang kendali selama pemulihan
- REL11-BP05 Menggunakan stabilitas statis untuk mencegah perilaku bimodal
- REL11-BP06 Mengirimkan notifikasi ketika peristiwa memengaruhi ketersediaan
- REL11-BP07 Merancang produk Anda agar memenuhi target ketersediaan dan perjanjian tingkat layanan (SLA) waktu aktif

REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan

Terus pantau kondisi beban kerja agar Anda dan sistem otomatis Anda langsung mengetahui penurunan kualitas atau kegagalan ketika muncul. Pantau indikator kinerja utama (KPI) berdasarkan nilai bisnis.

Semua mekanisme pemulihan dan penyembuhan harus dimulai dengan kemampuan untuk mendeteksi masalah secara cepat. Kegagalan teknis harus dideteksi terlebih dahulu sehingga dapat diatasi. Namun, ketersediaan didasarkan pada kemampuan beban kerja Anda untuk menghadirkan nilai bisnis, sehingga indikator kinerja utama (KPI) yang mengukurnya perlu menjadi bagian dari strategi deteksi dan perbaikan Anda.

Hasil yang diinginkan: Komponen penting dari suatu beban kerja dipantau secara independen untuk mendeteksi dan memperingatkan adanya kegagalan pada saat dan di bagian mana kegagalan tersebut terjadi.

Antipola umum:

- Tidak ada alarm yang dikonfigurasi, sehingga pemadaman terjadi tanpa notifikasi.
- Alarm tersedia, tetapi pada ambang batas yang tidak menyediakan waktu yang cukup untuk bereaksi.
- Metrik tidak dikumpulkan cukup sering untuk memenuhi sasaran waktu pemulihan (RTO).
- Hanya antarmuka beban kerja yang terlihat oleh pelanggan yang aktif dipantau.
- Hanya mengumpulkan metrik teknis, dan mengabaikan metrik fungsi bisnis.
- Tidak ada metrik yang mengukur pengalaman pengguna beban kerja.
- Terlalu banyak pemantau yang dibuat.

Manfaat menjalankan praktik terbaik ini: Pemantauan yang sesuai di semua lapisan memungkinkan Anda menghemat waktu pemulihan karena berkurangnya waktu deteksi.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Identifikasikan semua beban kerja yang akan ditinjau untuk pemantauan. Setelah Anda mengidentifikasi semua komponen beban kerja yang perlu dipantau, selanjutnya Anda perlu

menentukan interval pemantauan. Interval pemantauan akan berdampak langsung pada seberapa cepat pemulihan dapat dimulai berdasarkan waktu yang diperlukan untuk mendeteksi kegagalan. Rata-rata waktu deteksi (MTTD) adalah lamanya waktu antara terjadinya kegagalan dan ketika operasi perbaikan dimulai. Daftar layanan harus luas dan lengkap.

Pemantauan harus mencakup semua lapisan tumpukan aplikasi termasuk aplikasi, platform, infrastruktur, dan jaringan.

Strategi pemantauan Anda harus mempertimbangkan dampak kegagalan abu-abu. Untuk detail lebih lanjut tentang kegagalan abu-abu, lihat Kegagalan abu-abu di laporan resmi Pola Ketangguhan Multi-AZ Lanjutan.

Langkah implementasi

- Interval pemantauan Anda bergantung pada seberapa cepat Anda harus pulih. Waktu pemulihan Anda didorong oleh waktu yang diperlukan untuk pulih, sehingga Anda harus menentukan frekuensi pengumpulan dengan cara menghitung waktu ini serta sasaran waktu pemulihan (RTO) Anda.
- Konfigurasikan pemantauan mendetail untuk komponen dan layanan terkelola.
  - Tentukan apakah pemantauan mendetail untuk instans EC2 dan Auto Scaling diperlukan. Pemantauan mendetail menyediakan metrik interval satu menit, sedangkan pemantauan default menyediakan metrik interval lima menit.
  - Tentukan apakah pemantauan yang ditingkatkan untuk RDS diperlukan. Pemantauan yang ditingkatkan menggunakan agen di instans RDS untuk memperoleh informasi bermanfaat tentang berbagai alur atau proses.
  - Tentukan persyaratan pemantauan komponen nirserver penting untuk Lambda, API Gateway, Amazon EKS, Amazon ECS, dan semua jenis penyeimbang beban.
  - Tentukan persyaratan pemantauan komponen penyimpanan untuk Amazon S3, Amazon FSx, Amazon EFS, dan Amazon EBS.
- Buat metrik kustom untuk mengukur indikator kinerja kunci (KPI) bisnis. Beban kerja mengimplementasikan fungsi-fungsi bisnis utama, yang harus digunakan sebagai KPI yang membantu mengidentifikasi kapan terjadinya masalah tidak langsung.
- Pantau pengalaman pengguna untuk mendeteksi kegagalan menggunakan canary pengguna. Pengujian transaksi sintetis (juga disebut pengujian canary, tetapi tidak sama dengan deployment canary) yang dapat menjalankan dan menyimulasikan perilaku pelanggan adalah salah satu proses pengujian yang paling penting. Jalankan pengujian ini secara konstan terhadap titik akhir beban kerja Anda dari beragam lokasi jarak jauh.

- Buat metrik kustom yang melacak pengalaman pengguna. Jika Anda dapat menginstrumentasi pengalaman pelanggan, Anda dapat menentukan saat pengalaman pelanggan mengalami degradasi.
- Atur alarm untuk mendeteksi saat ada bagian dari beban kerja Anda yang tidak berfungsi dengan baik, dan untuk menunjukkan kapan harus menskalakan sumber daya secara otomatis. Alarm dapat ditampilkan secara visual di dasbor, mengirimkan peringatan melalui Amazon SNS atau email, dan menggunakan Auto Scaling untuk menaikkan atau menurunkan skala sumber daya beban kerja.
- Buat dasbor untuk memvisualisasikan metrik Anda. Dasbor dapat digunakan untuk melihat tren, penyimpangan, dan indikator potensi masalah lainnya, atau menyediakan penanda untuk masalah yang ingin Anda selidiki.
- Buat pemantauan penelusuran terdistribusi untuk layanan Anda. Dengan pemantauan terdistribusi, Anda dapat memahami cara kerja aplikasi Anda dan layanan dasar dalam mengidentifikasi dan memecahkan akar masalah dan galat kinerja.
- Buat dasbor sistem pemantauan (menggunakan CloudWatch atau X-Ray) dan pengumpulan data di Wilayah dan akun terpisah.
- Buat integrasi untuk pemantauan Amazon Health Aware untuk memungkinkan visibilitas pemantauan ke sumber daya AWS yang mungkin mengalami degradasi. Untuk beban kerja yang penting untuk bisnis, solusi ini menyediakan akses ke peringatan proaktif dan waktu nyata untuk layanan AWS.

Sumber daya

Praktik terbaik terkait:

- Definisi Ketersediaan
- REL11-BP06 Mengirimkan Notifikasi ketika peristiwa memengaruhi ketersediaan

Dokumen terkait:

- Amazon CloudWatch Synthetics memungkinkan Anda untuk membuat canary pengguna
- Aktifkan atau Nonaktifkan Pemantauan Mendetail untuk Instans Anda
- Pemantauan yang Ditingkatkan
- Memantau Grup dan Instans Auto Scaling Anda Menggunakan Amazon CloudWatch
- Memublikasikan Metrik Kustom

- Menggunakan Alarm Amazon CloudWatch
- Menggunakan Dasbor CloudWatch
- Menggunakan Dasbor CloudWatch Lintas Akun dan Lintas Wilayah
- Menggunakan Penelusuran X-Ray Lintas Akun dan Lintas Wilayah
- Memahami ketersediaan
- Mengimplementasikan Amazon Health Aware (AHA)

Video terkait:

- Memitigasi kegagalan abu-abu

Contoh terkait:

- Lab Well-Architected: Level 300: Mengimplementasikan Pemeriksaan Kondisi dan Mengelola Dependensi untuk Meningkatkan Keandalan
- Lokakarya One Observability: Menjelajahi X-Ray

Alat terkait:

- CloudWatch
- CloudWatch X-Ray

REL11-BP02 Melakukan failover ke sumber daya yang sehat

Jika terjadi kegagalan sumber daya, sumber daya yang sehat harus terus melayani permintaan. Untuk kerusakan lokasi (seperti Zona Ketersediaan atau Wilayah AWS) pastikan Anda memiliki sistem untuk melakukan failover ke sumber daya yang sehat di lokasi yang tidak terkena gangguan.

Saat merancang layanan, distribusikan beban di seluruh sumber daya, Zona Ketersediaan, atau Wilayah. Oleh karena itu, kegagalan sumber daya individu atau gangguan dapat dimitigasi dengan mengalihkan lalu lintas ke sumber daya sehat yang masih ada. Pertimbangkan bagaimana layanan ditemukan dan dirutekan jika terjadi kegagalan.

Rancang layanan Anda dengan mempertimbangkan pemulihan kesalahan. Di AWS, kami merancang layanan untuk meminimalkan waktu untuk pulih dari kegagalan dan dampak terhadap data. Layanan

kami utamanya menggunakan penyimpanan data yang mengenali permintaan hanya setelah disimpan dalam waktu lama di beberapa replika di dalam suatu Wilayah. Layanan dan sumber daya ini dibangun untuk menggunakan isolasi berbasis sel dan menggunakan isolasi kesalahan yang disediakan oleh Zona Ketersediaan. Kami banyak menggunakan otomatisasi di dalam prosedur operasional kami. Kami juga mengoptimalkan fungsionalitas “ganti dan mulai ulang” kami untuk pulih secara cepat dari gangguan.

Pola dan desain yang memungkinkan failover bervariasi untuk setiap layanan platform AWS. Banyak layanan terkelola native AWS adalah layanan yang secara native multi-Zona Ketersediaan (seperti Lambda atau API Gateway). Layanan AWS lain (seperti EC2 dan EKS) memerlukan desain praktik terbaik khusus untuk mendukung failover sumber daya atau penyimpanan data di seluruh AZ.

Pemantauan harus disiapkan untuk memeriksa apakah sumber daya failover sehat, melacak kemajuan sumber daya yang melakukan failover, dan memantau pemulihan proses bisnis.

Hasil yang diinginkan: Sistem mampu secara otomatis atau manual menggunakan sumber daya baru untuk pulih dari degradasi.

Antipola umum:

- Perencanaan kegagalan bukan bagian dari fase perencanaan dan desain.
- RTO dan RPO tidak ditetapkan.
- Pemantauan yang tidak memadai untuk mendeteksi sumber daya yang gagal.
- Pemisahan domain kegagalan yang layak.
- Kegagalan Multi-Wilayah tidak dipertimbangkan.
- Deteksi kegagalan terlalu sensitif atau agresif saat memutuskan untuk melakukan failover.
- Tidak menguji atau memvalidasi desain failover.
- Melakukan otomatisasi pemulihan otomatis, tetapi tidak memberikan notifikasi bahwa pemulihan diperlukan.
- Kurangnya periode peredaman untuk menghindari gagal kembali yang terlalu cepat.

Manfaat menjalankan praktik terbaik ini: Anda dapat membangun sistem yang lebih tangguh yang mempertahankan keandalan saat mengalami kegagalan dengan melakukan degradasi secara mulus dan pulih dengan cepat.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Layanan AWS, seperti Elastic Load Balancing dan Amazon EC2 Auto Scaling, membantu mendistribusikan beban di seluruh sumber daya dan Zona Ketersediaan. Oleh karena itu, kegagalan sumber daya individu (seperti instans EC2) atau gangguan pada Zona Ketersediaan dapat dimitigasi dengan mengalihkan lalu lintas ke sumber daya sehat yang masih ada.

Untuk beban kerja multi-Wilayah, desainnya lebih rumit. Misalnya, replika baca lintas Wilayah memungkinkan Anda untuk melakukan deployment data ke beberapa Wilayah AWS. Namun, failover masih diperlukan untuk mempromosikan replika baca ke primer kemudian mengarahkan lalu lintas Anda ke titik akhir baru. Amazon Route 53, Route 53 Route 53 ARC, CloudFront, dan AWS Global Accelerator dapat membantu merutekan lalu lintas di seluruh Wilayah AWS.

Layanan AWS, seperti Amazon S3, Lambda, API Gateway, Amazon SQS, Amazon SNS, Amazon SES, Amazon Pinpoint, Amazon ECR, AWS Certificate Manager, EventBridge, atau Amazon DynamoDB, secara otomatis di-deploy ke beberapa Zona Ketersediaan oleh AWS. Jika terjadi kegagalan, layanan-layanan AWS ini secara otomatis merutekan lalu lintas ke lokasi yang sehat. Data disimpan secara redundan di beberapa Zona Ketersediaan dan tetap tersedia.

Untuk Amazon RDS, Amazon Aurora, Amazon Redshift, Amazon EKS, atau Amazon ECS, Multi-AZ adalah opsi konfigurasi. AWS dapat mengarahkan lalu lintas ke instans sehat jika failover dimulai. Tindakan failover ini dapat diambil oleh AWS atau sebagaimana diperlukan oleh pelanggan

Untuk instans Amazon EC2, Amazon Redshift, tugas Amazon ECS, atau pod Amazon EKS, Anda memilih Zona Ketersediaan mana untuk deployment. Untuk beberapa desain, Elastic Load Balancing memberikan solusi untuk mendeteksi instans di zona yang tidak sehat dan merutekan lalu lintas ke zona yang sehat. Elastic Load Balancing juga dapat merutekan lalu lintas ke komponen di pusat data on-premise Anda.

Untuk failover lalu lintas Multi-Wilayah, pengalihan rute dapat memanfaatkan Amazon Route 53, Route 53 ARC, AWS Global Accelerator, Route 53 Private DNS for VPCs, atau CloudFront untuk menyediakan cara untuk menentukan domain internet dan menetapkan kebijakan perutean, termasuk pemeriksaan kondisi, untuk merutekan lalu lintas ke Wilayah yang sehat. AWS Global Accelerator menyediakan alamat IP statis yang bertindak sebagai titik masuk tetap ke aplikasi Anda, lalu merutekan ke titik akhir di Wilayah AWS yang Anda pilih, menggunakan jaringan global AWS, bukan internet, demi performa dan keandalan yang lebih baik.

Langkah implementasi

- Buat desain failover untuk semua aplikasi dan layanan yang sesuai. Isolasi setiap komponen arsitektur dan buat desain failover yang memenuhi RTO dan RPO untuk setiap komponen.
- Konfigurasikan lingkungan yang lebih rendah (seperti pengembangan atau pengujian) dengan semua layanan yang diharuskan memiliki rencana failover. Deploy solusi menggunakan infrastruktur sebagai kode (IaC) untuk memastikan kemampuan pengulangan.
- Konfigurasikan lokasi pemulihan seperti Wilayah kedua untuk mengimplementasikan dan menguji desain failover. Jika perlu, sumber daya untuk pengujian dapat dikonfigurasi secara sementara untuk membatasi biaya tambahan.
- Tentukan rencana failover mana yang diotomatisasi olehAWS, yang dapat diotomatisasi oleh proses DevOps, dan mana yang mungkin dilakukan secara manual. Dokumentasikan dan ukur RTO dan RPO setiap layanan.
- Buat playbook failover dan sertakan semua langkah untuk melakukan failover setiap sumber daya, aplikasi, dan layanan.
- Buat playbook failback dan sertakan semua langkah untuk melakukan failback (dengan pengaturan waktu) setiap sumber daya, aplikasi, dan layanan
- Buat rencana untuk memulai dan melatih playbook. Gunakan simulasi dan pengujian kekacauan untuk menguji langkah-langkah dan otomatisasi playbook.
- Untuk gangguan lokasi (seperti Zona Ketersediaan atau Wilayah AWS), pastikan Anda memiliki sistem untuk melakukan failover ke sumber daya yang sehat di lokasi yang tidak terkena gangguan. Periksa kuota, tingkat penskalaan otomatis, dan sumber daya yang berjalan sebelum pengujian failover.

Sumber daya

Praktik terbaik Well-Architected terkait:

- REL13 - Merencanakan DR
- REL10 - Menggunakan isolasi kesalahan untuk melindungi beban kerja Anda

Dokumen terkait:

- Menetapkan Target RTO dan RPO
- Menyiapkan Route 53 ARC dengan penyeimbang beban aplikasi

- Failover menggunakan perutean Tertimbang Route 53
- DR dengan Route 53 ARC
- EC2 dengan penskalaan otomatis
- Deployment EC2 - Multi-AZ
- Deployment ECS - Multi-AZ
- Mengalihkan lalu lintas menggunakan Route 53 ARC
- Lambda dengan Application Load Balancer dan Failover
- Replikasi ACM dan Failover
- Replikasi Penyimpanan Parameter dan Failover
- Replikasi lintas wilayah ECR dan Failover
- Konfigurasi replikasi lintas wilayah manajer rahasia
- Mengaktifkan replikasi lintas wilayah untuk EFS dan Failover
- Replikasi Lintas Wilayah EFS dan Failover
- Failover Jaringan
- Failover titik akhir S3 menggunakan MRAP
- Membuat replikasi lintas wilayah untuk S3
- Failover API Gateway Wilayah dengan Route 53 ARC
- Failover menggunakan akselerator global multiwilayah
- Failover dengan DRS
- Membuat Mekanisme Pemulihan Bencana Menggunakan Amazon Route 53

Contoh terkait:

- Pemulihan Bencana di AWS
- Pemulihan Bencana Elastis di AWS

REL11-BP03 Mengotomatisasi pemulihan di semua lapisan

Setelah kegagalan dideteksi, gunakan kemampuan otomatis untuk melakukan tindakan perbaikan. Degradasi dapat dipulihkan secara otomatis melalui mekanisme servis internal atau memerlukan sumber daya untuk dimulai ulang atau dihapus melalui tindakan remediasi.

Untuk aplikasi yang dikelola secara mandiri dan perbaikan lintas-Wilayah, desain pemulihan dan proses perbaikan otomatis dapat ditarik dari praktik terbaik yang ada.

Kemampuan untuk memulai ulang atau menghapus sumber daya adalah alat yang penting untuk meremediasi kegagalan. Salah satu praktik terbaik adalah membuat layanan stateless jika memungkinkan. Praktik ini mencegah hilangnya data atau ketersediaan pada saat mulai ulang sumber daya. Di cloud, Anda dapat (dan umumnya harus) mengganti seluruh sumber daya (misalnya, instans komputasi atau fungsi nirserver) sebagai bagian dari mulai ulang. Mulai ulang itu sendiri adalah cara yang mudah dan andal untuk pulih dari kegagalan. Ada berbagai jenis kegagalan yang terjadi di dalam beban kerja. Kegagalan dapat terjadi di perangkat keras, perangkat lunak, komunikasi, dan operasi.

Memulai ulang atau mencoba ulang juga berlaku untuk permintaan jaringan. Terapkan pendekatan pemulihan yang sama ke waktu habis jaringan serta kegagalan dependensi yakni ketika dependensi menunjukkan kesalahan. Kedua peristiwa tersebut memiliki efek yang serupa terhadap sistem, sehingga alih-alih berupaya untuk menjadikan masing-masing sebagai kasus spesial, terapkan strategi serupa berupa coba ulang terbatas dengan mundur eksponensial dan jitter. Kemampuan untuk memulai ulang adalah mekanisme pemulihan yang disertakan dalam komputasi berorientasi pemulihan dan arsitektur klaster ketersediaan tinggi.

Hasil yang diinginkan: Tindakan otomatis dilakukan untuk meremediasi deteksi kegagalan.

Antipola umum:

- Menyediakan sumber daya tanpa penskalaan otomatis.
- Melakukan deployment aplikasi di instans atau kontainer secara terpisah.
- Melakukan deployment aplikasi yang tidak dapat dilakukan ke beberapa lokasi tanpa menggunakan pemulihan otomatis.
- Memulihkan secara manual aplikasi yang gagal dipulihkan oleh penskalaan otomatis dan pemulihan otomatis.
- Tidak ada otomatisasi untuk failover basis data.
- Tidak ada metode otomatis untuk mengalihkan rute lalu lintas ke titik akhir baru.
- Tidak ada replikasi penyimpanan.

Manfaat menjalankan praktik terbaik ini: Pemulihan otomatis dapat mengurangi waktu rata-rata pemulihan dan meningkatkan ketersediaan Anda.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Desain untuk Amazon EKS atau layanan Kubernetes lainnya harus mencakup replika minimum dan maksimum atau set stateful dan penyesuaian ukuran klaster dan grup simpul minimum. Mekanisme ini menyediakan jumlah minimum sumber daya pemrosesan yang tersedia secara terus-menerus sambil secara otomatis memulihkan kegagalan apa pun menggunakan bidang kendali Kubernetes.

Pola desain yang diakses melalui penyeimbang beban menggunakan klaster komputasi harus memanfaatkan grup Auto Scaling. Elastic Load Balancing (ELB) secara otomatis mendistribusikan lalu lintas aplikasi yang masuk di beberapa target dan perangkat virtual di satu atau beberapa Zona Ketersediaan (AZ).

Desain berbasis komputasi klaster yang tidak menggunakan penyeimbangan beban harus dirancang ukurannya untuk kehilangan setidaknya satu simpul. Dengan begitu, layanan dapat terus berjalan dalam kapasitas yang kemungkinan lebih rendah saat memulihkan simpul baru. Contoh layanannya adalah Mongo, DynamoDB Accelerator, Amazon Redshift, Amazon EMR, Cassandra, Kafka, MSK-EC2, Couchbase, ELK, dan Amazon OpenSearch Service. Banyak dari layanan ini dapat dirancang dengan fitur pemulihan otomatis tambahan. Beberapa teknologi klaster harus menghasilkan peringatan atas hilangnya simpul yang memicu alur kerja otomatis atau manual untuk membuat ulang simpul baru. Alur kerja ini dapat diotomatisasi menggunakan AWS Systems Manager untuk meremediasi masalah dengan cepat.

Amazon EventBridge dapat digunakan untuk memantau dan memfilter peristiwa seperti alarm CloudWatch atau perubahan status pada layanan AWS lain. Berdasarkan informasi peristiwa, layanan ini kemudian dapat memanggil AWS Lambda, Otomatisasi Systems Manager, atau target lain untuk menjalankan logika remediasi kustom pada beban kerja Anda. Amazon EC2 Auto Scaling dapat dikonfigurasi untuk memeriksa kondisi instans EC2. Jika instans sedang dalam status apa pun selain running (berjalan), atau jika status sistem terganggu, Amazon EC2 Auto Scaling menganggap instans tersebut tidak sehat dan meluncurkan instans pengganti. Untuk penggantian skala besar (seperti hilangnya seluruh Zona Ketersediaan), stabilitas statis lebih disarankan untuk ketersediaan tinggi.

Langkah implementasi

- Gunakan grup Auto Scaling untuk men-deploy tingkatan dalam beban kerja. Auto Scaling dapat melakukan pemulihan mandiri untuk aplikasi stateless serta menambahkan dan menghapus kapasitas.
- Untuk instans komputasi yang disebutkan sebelumnya, gunakan penyeimbangan beban dan pilih jenis penyeimbang beban yang sesuai.

- Pertimbangkan pemulihan untuk Amazon RDS. Dengan instans siaga, konfigurasikan untuk failover otomatis ke instans siaga. Untuk Amazon RDS Read Replica, alur kerja otomatis diperlukan untuk membuat replika baca primer.
- Implementasikan pemulihan otomatis pada instans EC2 yang telah melakukan deployment aplikasi yang tidak dapat di-deploy di beberapa lokasi, dan dapat menoleransi boot ulang setelah kegagalan. Pemulihan otomatis dapat digunakan untuk mengganti perangkat keras yang mengalami kegagalan dan memulai ulang instans ketika aplikasi tidak dapat diterapkan di beberapa lokasi. Metadata instans dan alamat IP terkait disimpan, serta volume EBS dan pasang poin ke Amazon Elastic File System atau File Systems for Lustre dan Windows. Jika menggunakan AWS OpsWorks, Anda dapat mengonfigurasi pemulihan otomatis instans EC2 pada tingkat lapisan.
- Implementasikan pemulihan otomatis menggunakan AWS Step Functions dan AWS Lambda ketika Anda tidak dapat menggunakan penskalaan otomatis atau pemulihan otomatis, atau ketika pemulihan otomatis gagal. Ketika Anda tidak dapat menggunakan penskalaan otomatis, dan tidak dapat menggunakan pemulihan otomatis atau pemulihan otomatis gagal, Anda dapat mengotomatiskan pemulihan menggunakan AWS Step Functions dan AWS Lambda.
- Amazon EventBridge dapat digunakan untuk memantau dan memfilter peristiwa seperti alarm CloudWatch atau perubahan status di layanan AWS lain. Berdasarkan informasi peristiwa, layanan ini kemudian dapat menginvokasi AWS Lambda (atau target lainnya) untuk menjalankan logika remediasi kustom pada beban kerja Anda.

Sumber daya

Praktik terbaik terkait:

- Definisi Ketersediaan
- REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan

Dokumen terkait:

- Cara Kerja AWS Auto Scaling
- Pemulihan Otomatis Amazon EC2
- Amazon Elastic Block Store (Amazon EBS)
- Amazon Elastic File System (Amazon EFS)
- Apa itu Amazon FSx for Lustre?

- Apa itu Amazon FSx for Windows File Server?
- AWS OpsWorks: Menggunakan Auto Healing untuk Mengganti Instans yang Gagal
- Apa itu AWS Step Functions?
- Apa itu AWS Lambda?
- Apa Itu Amazon EventBridge?
- Menggunakan Alarm Amazon CloudWatch
- Failover Amazon RDS
- SSM - Otomatisasi Systems Manager
- Praktik Terbaik Arsitektur Tangguh

Video terkait:

- Penyediaan dan Penskalaan OpenSearch Service Secara Otomatis
- Failover Amazon RDS Secara Otomatis

Contoh terkait:

- Lokakarya di Auto Scaling
- Lokakarya Failover Amazon RDS

Alat terkait:

- CloudWatch
- CloudWatch X-Ray

REL11-BP04 Mengandalkan bidang data dan bukan bidang kendali selama pemulihan

Bidang kendali menyediakan API administratif yang digunakan untuk membuat, membaca, dan mendeskripsikan, memperbarui, menghapus, dan mencantumkan (CRUDL) sumber daya, sedangkan bidang data menangani lalu lintas layanan sehari-hari. Saat mengimplementasikan respons pemulihan atau mitigasi terhadap peristiwa yang berpotensi berdampak pada ketahanan, fokuslah pada penggunaan operasi bidang kontrol dalam jumlah minim untuk memulihkan, mengubah skala,

mengembalikan, memperbaiki, atau melakukan failover layanan. Tindakan bidang data harus menggantikan aktivitas apa pun selama peristiwa degradasi ini.

Misalnya, berikut ini adalah semua tindakan bidang kendali: meluncurkan instans komputasi baru, membuat penyimpanan blok, dan mendeskripsikan layanan antrean. Saat Anda meluncurkan instans komputasi, bidang kendali harus melakukan beberapa tugas seperti menemukan host fisik dengan kapasitas, mengalokasikan antarmuka jaringan, menyiapkan volume penyimpanan blok lokal, menghasilkan kredensial, dan menambahkan aturan keamanan. Orkestrasi bidang kendali cenderung rumit.

Hasil yang diinginkan: Ketika sumber daya memasuki keadaan terganggu, sistem mampu pulih secara otomatis atau manual dengan mengalihkan lalu lintas dari sumber daya yang terganggu ke sumber daya yang sehat.

Antipola umum:

- Ketergantungan pada pengubahan catatan DNS untuk mengalihkan lalu lintas.
- Ketergantungan pada operasi penskalaan bidang kendali untuk menggantikan komponen yang terganggu karena sumber daya yang disediakan tidak memadai.
- Mengandalkan tindakan bidang kendali ekstensif multi-API dan multi layanan untuk meremediasi kategori gangguan apa pun.

Manfaat menjalankan praktik terbaik ini: Peningkatan tingkat keberhasilan untuk remediasi otomatis dapat mengurangi waktu rata-rata Anda untuk pemulihan dan meningkatkan ketersediaan beban kerja.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang: Untuk jenis degradasi layanan tertentu, bidang kendali terkena pengaruh. Ketergantungan pada penggunaan bidang kendali secara ekstensif untuk remediasi dapat meningkatkan waktu pemulihan (RTO) dan rata-rata waktu untuk pulih (MTTR).

Panduan implementasi

Untuk membatasi tindakan bidang data, nilai setiap layanan untuk menemukan tindakan yang diperlukan untuk memulihkan layanan.

Manfaatkan Amazon Route 53 Application Recovery Controller untuk mengalihkan lalu lintas DNS. Fitur-fitur ini terus-menerus memantau kemampuan aplikasi Anda untuk pulih dari kegagalan,

dan memampukan Anda untuk mengontrol pemulihan aplikasi di beberapa Wilayah AWS, Zona Ketersediaan, dan on-premise.

Kebijakan perutean Route 53 menggunakan bidang kendali, jadi jangan mengandalkannya untuk pemulihan. Bidang data Route 53 menjawab kueri DNS dan melakukan serta mengevaluasi pemeriksaan kondisi. Bidang data ini terdistribusi secara global dan didesain untuk perjanjian tingkat layanan (SLA) dengan ketersediaan 100%..

Konsol dan API manajemen Route 53 di mana Anda membuat, memperbarui, dan menghapus sumber daya Route 53 dijalankan di bidang kendali yang didesain untuk memprioritaskan durabilitas dan konsistensi tinggi yang Anda perlukan ketika mengelola DNS. Untuk mencapai hal ini, bidang kendali ditempatkan di satu Wilayah: US East (N. Virginia). Walaupun kedua sistem dibangun agar sangat andal, bidang kendali tidak disertakan dalam SLA. Mungkin ada peristiwa langka di mana desain tangguh bidang data memungkinkannya untuk mempertahankan ketersediaan sedangkan bidang kendali tidak. Untuk mekanisme failover dan pemulihan bencana, gunakan fungsi bidang data untuk memberikan keandalan yang sebaik mungkin.

Untuk Amazon EC2, gunakan desain stabilitas statis untuk membatasi tindakan bidang kendali. Tindakan bidang kendali mencakup peningkatan skala sumber daya secara individu atau menggunakan grup Auto Scaling (ASG). Untuk tingkat ketahanan tertinggi, berikan kapasitas yang cukup di klaster yang digunakan untuk failover. Jika ambang kapasitas ini harus dibatasi, tetapkan throttle pada keseluruhan sistem menyeluruh untuk membatasi lalu lintas total yang mencapai set sumber daya terbatas.

Untuk layanan seperti Amazon DynamoDB, Amazon API Gateway, penyeimbang beban, dan nirserver AWS Lambda, penggunaan layanan-layanan tersebut memanfaatkan bidang data. Namun, pembuatan fungsi baru, penyeimbang beban, gateway API, atau tabel DynamoDB adalah tindakan bidang kendali dan harus diselesaikan sebelum degradasi sebagai persiapan peristiwa dan latihan tindakan failover. Untuk Amazon RDS, tindakan bidang data memungkinkan akses ke data.

Untuk informasi selengkapnya tentang bidang data, bidang kendali, dan bagaimana AWS membangun layanan untuk memenuhi target ketersediaan tinggi, lihat Stabilitas statis menggunakan Zona Ketersediaan.

Pahami operasi mana yang ada di bidang data dan mana yang ada di bidang kendali.

Langkah implementasi

Untuk setiap beban kerja yang perlu dipulihkan setelah peristiwa degradasi, evaluasi runbook failover, desain ketersediaan tinggi, desain perbaikan otomatis, atau rencana pemulihan sumber daya HA. Identifikasikan setiap tindakan yang mungkin dianggap sebagai tindakan bidang kendali.

Pertimbangkan mengubah tindakan kendali ke tindakan bidang data:

- Auto Scaling (bidang kendali) dibandingkan dengan sumber daya Amazon EC2 yang telah diskalakan sebelumnya (bidang data)
- Migrasikan ke Lambda dan metode penskalaannya (bidang data) atau Amazon EC2 dan ASG (bidang kendali)
- Nilai desain apa pun menggunakan Kubernetes dan sifat tindakan bidang kendali. Menambahkan pod adalah tindakan bidang data di Kubernetes. Tindakan harus dibatasi ke penambahan pod dan bukan ke penambahan simpul. Jika menggunakan simpul yang disediakan secara berlebihan adalah metode yang lebih disukai untuk membatasi tindakan bidang kendali

Pertimbangkan pendekatan alternatif yang memungkinkan tindakan bidang data untuk memengaruhi remediasi yang sama.

- Route 53 Rekam perubahan (bidang kendali) atau Route 53 ARC (bidang data)
- Route 53 Pemeriksaan kondisi untuk pembaruan yang lebih otomatis

Pertimbangkan beberapa layanan di Wilayah sekunder, jika layanan sangat penting, untuk memungkinkan lebih banyak tindakan bidang kendali dan bidang data di Wilayah yang tidak terdampak.

- Amazon EC2 Auto Scaling atau Amazon EKS di Wilayah primer dibandingkan dengan Amazon EC2 Auto Scaling atau Amazon EKS di Wilayah sekunder dan merutekan lalu lintas ke Wilayah sekunder (tindakan bidang kendali)
- Membuat replika baca di primer sekunder atau mencoba tindakan yang sama di Wilayah primer (tindakan bidang kendali)

Sumber daya

Praktik terbaik terkait:

- Definisi Ketersediaan
- REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan

Dokumen terkait:

- Partner APN: partner yang dapat membantu dengan otomatisasi toleransi kesalahan Anda
- AWS Marketplace: produk yang dapat digunakan untuk toleransi kesalahan
- Amazon Builders' Library: Menghindari kelebihan beban dalam sistem terdistribusi dengan mengontrol layanan lebih kecil
- API Amazon DynamoDB (bidang kendali dan bidang data)
- Pelaksanaan AWS Lambda (dibagi menjadi bidang kendali dan bidang data)
- Bidang Data AWS Elemental MediaStore
- Membangun aplikasi yang sangat tangguh menggunakan Pengontrol Pemulihan Aplikasi Amazon Route 53, Bagian 1: Tumpukan Wilayah Tunggal
- Membangun aplikasi yang sangat tangguh menggunakan Pengontrol Pemulihan Aplikasi Amazon Route 53, Bagian 2: Tumpukan Multi-Wilayah
- Membuat Mekanisme Pemulihan Bencana Menggunakan Amazon Route 53
- Apa itu Pengontrol Pemulihan Aplikasi Route 53?
- Bidang Kendali dan bidang data Kubernetes

Video terkait:

- Kembali ke Dasar - Menggunakan Stabilitas Statis
- Membangun beban kerja multi-lokasi yang tangguh menggunakan layanan global AWS

Contoh terkait:

- Memperkenalkan Pengontrol Pemulihan Aplikasi Amazon Route 53
- Amazon Builders' Library: Menghindari kelebihan beban dalam sistem terdistribusi dengan mengontrol layanan lebih kecil
- Membangun aplikasi yang sangat tangguh menggunakan Pengontrol Pemulihan Aplikasi Amazon Route 53, Bagian 1: Tumpukan Wilayah Tunggal

- Membangun aplikasi yang sangat tangguh menggunakan Pengontrol Pemulihan Aplikasi Amazon Route 53, Bagian 2: Tumpukan Multi-Wilayah
- Stabilitas statis menggunakan Zona Ketersediaan

Alat terkait:

- Amazon CloudWatch
- AWS X-Ray

REL11-BP05 Menggunakan stabilitas statis untuk mencegah perilaku bimodal

Beban kerja harus stabil secara statis dan hanya beroperasi dalam mode normal tunggal. Perilaku bimodal adalah ketika beban kerja Anda menunjukkan perilaku yang berbeda dalam mode normal dan mode kegagalan.

Misalnya, Anda mungkin mencoba pulih dari kegagalan Zona Ketersediaan dengan meluncurkan instans baru di Zona Ketersediaan yang berbeda. Hal ini dapat menyebabkan respons bimodal selama mode kegagalan. Sebagai gantinya, Anda harus membangun beban kerja yang stabil secara statis dan beroperasi dalam satu mode saja. Dalam contoh ini, instans-instans tersebut seharusnya telah disediakan di Zona Ketersediaan kedua sebelum terjadinya kegagalan. Desain stabilitas statis ini memastikan beban kerja hanya beroperasi dalam satu mode.

Hasil yang diinginkan: Beban kerja tidak menunjukkan perilaku bimodal selama mode normal dan mode kegagalan.

Antipola umum:

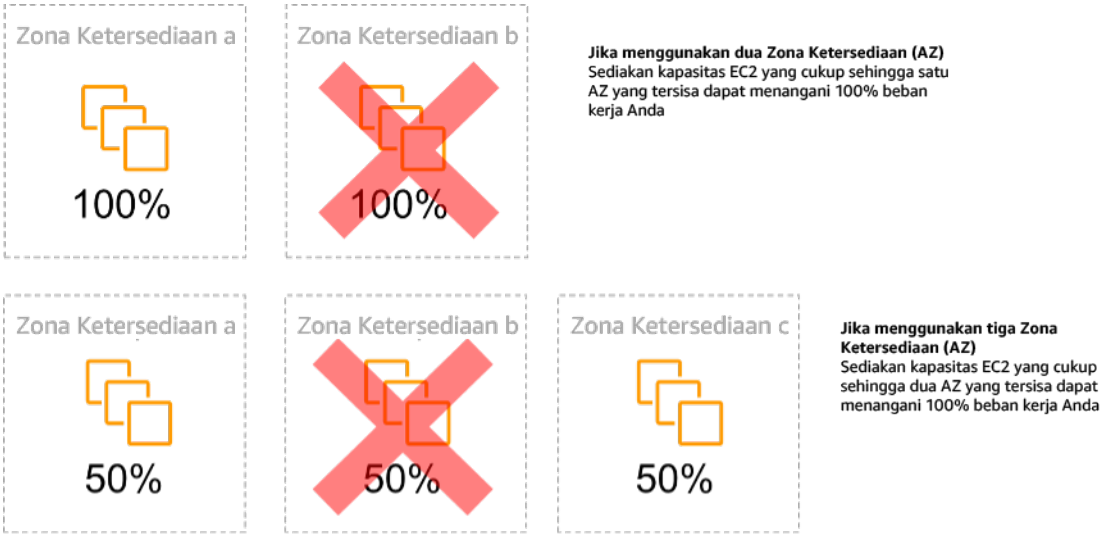
- Berasumsi bahwa sumber daya selalu dapat disediakan terlepas dari ruang lingkup kegagalannya.
- Mencoba memperoleh sumber daya secara dinamis selama kegagalan.
- Tidak menyediakan sumber daya yang memadai di seluruh zona atau Wilayah sampai terjadi kegagalan.
- Mempertimbangkan desain stabil statis hanya untuk sumber daya komputasi.

Manfaat menjalankan praktik terbaik ini: Beban kerja yang berjalan dengan desain yang stabil secara statis mampu memiliki hasil yang dapat diprediksi selama peristiwa normal dan kegagalan.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Perilaku bimodal terjadi ketika beban kerja Anda menunjukkan perilaku yang berbeda dalam mode normal dan mode gagal, (misalnya, mengandalkan peluncuran instans baru jika Zona Ketersediaan gagal). Contoh perilaku bimodal adalah ketika Amazon EC2 yang stabil menyediakan instans yang cukup di setiap Zona Ketersediaan untuk menangani beban dari beban kerja jika satu AZ disingkirkan. Elastic Load Balancing atau Amazon Route 53 akan melakukan pemeriksaan kondisi untuk memindahkan beban dari instans yang terganggu. Setelah lalu lintas dipindahkan, gunakan AWS Auto Scaling untuk mengganti instans secara asinkron dari zona yang gagal dan luncurkan di zona sehat. Stabilitas statis untuk deployment komputasi (seperti kontainer atau instans EC2) akan menghasilkan keandalan tertinggi.



Stabilitas statis instans EC2 di seluruh Zona Ketersediaan

Ini harus ditimbang berdasarkan biaya untuk model ini serta nilai bisnis untuk memelihara beban kerja di bawah semua kasus ketahanan. Menyediakan kapasitas komputasi yang lebih sedikit dan mengandalkan peluncuran instans baru apabila terjadi kegagalan memang lebih murah, tetapi untuk kegagalan berskala besar (seperti gangguan pada Zona Ketersediaan atau Regional), pendekatan ini kurang efektif karena bergantung pada bidang operasional serta sumber daya yang tersedia di zona atau Wilayah yang tidak terdampak.

Solusi Anda harus mengukur keandalan berdasarkan kebutuhan biaya untuk beban kerja Anda. Arsitektur stabilitas statis berlaku untuk berbagai arsitektur termasuk instans komputasi yang tersebar di Zona Ketersediaan, desain replika baca basis data, desain klaster Kubernetes (Amazon EKS), dan arsitektur failover multi-Wilayah.

Anda juga dapat menerapkan desain yang lebih stabil secara statis dengan menggunakan lebih banyak sumber daya di setiap zona. Dengan menggunakan lebih banyak zona, Anda mengurangi jumlah komputasi tambahan yang Anda perlukan untuk stabilitas statis.

Contoh perilaku bimodal adalah batas waktu jaringan yang dapat menyebabkan sistem mencoba melakukan refresh status konfigurasi seluruh sistem. Ini akan menambahkan beban yang tak terduga ke komponen lain, dan dapat menyebabkan komponen tersebut gagal, sehingga berimbas pada konsekuensi lain yang tak terduga. Putaran umpan balik negatif ini memengaruhi ketersediaan beban kerja Anda. Sebagai gantinya, Anda dapat membangun sistem yang stabil secara statis dan beroperasi dalam satu mode saja. Desain yang stabil secara statis akan melakukan tugas yang konstan, dan selalu menyegarkan status konfigurasi dengan irama yang teratur. Ketika panggilan gagal, beban kerja akan menggunakan nilai yang sebelumnya di-cache, dan memulai alarm.

Contoh perilaku bimodal lainnya adalah memperbolehkan klien untuk melewati cache beban kerja Anda ketika kegagalan terjadi. Ini mungkin terlihat seperti solusi yang mengakomodasi kebutuhan klien, tetapi hal ini secara signifikan dapat mengubah permintaan di beban kerja Anda dan kemungkinan akan mengakibatkan kegagalan.

Lakukan penilaian beban kerja kritis untuk menentukan beban kerja apa yang memerlukan jenis desain ketahanan ini. Untuk beban kerja yang dianggap kritis, setiap komponen aplikasi harus ditinjau. Contoh jenis layanan yang memerlukan evaluasi stabilitas statis adalah:

- Komputasi: Amazon EC2, EKS-EC2, ECS-EC2, EMR-EC2
- Basis Data: Amazon Redshift, Amazon RDS, Amazon Aurora
- Penyimpanan: Amazon S3 (Zona Tunggal), Amazon EFS (mount), Amazon FSx (mount)
- Penyeimbang beban: Di bawah desain tertentu

Langkah implementasi

- Bangun sistem yang stabil secara statis dan hanya beroperasi dalam satu mode. Dalam hal ini, sediakan cukup instans di setiap Zona Ketersediaan atau Wilayah untuk menangani kapasitas beban kerja jika satu Zona Ketersediaan atau Wilayah dihapus. Berbagai layanan dapat digunakan untuk perutean ke sumber daya yang sehat, seperti:
  - Perutean DNS Lintas Wilayah
  - Perutean Multiwilayah Amazon S3 MRAP
  - AWS Global Accelerator
  - Amazon Route 53 Application Recovery Controller

- Konfigurasikan replika baca basis data untuk memperhitungkan hilangnya satu instans utama atau replika baca. Jika lalu lintas dilayani oleh replika baca, kuantitas di setiap Zona Ketersediaan dan setiap Wilayah harus sama dengan kebutuhan keseluruhan jika terjadi kegagalan zona atau Wilayah.
- Konfigurasikan data kritis di dalam penyimpanan Amazon S3 yang dirancang agar stabil secara statis untuk data yang disimpan jika terjadi kegagalan Zona Ketersediaan. Jika kelas penyimpanan Amazon S3 One Zone-IA digunakan, ini tidak boleh dianggap stabil secara statis, karena hilangnya zona tersebut akan meminimalkan akses ke data yang disimpan.
- Penyeimbang beban terkadang dikonfigurasi secara salah atau secara bawaan untuk melayani satu Zona Ketersediaan tertentu. Dalam hal ini, desain yang stabil secara statis mungkin adalah menyebarkan beban kerja di beberapa AZ dalam desain yang lebih kompleks. Desain asli dapat digunakan untuk mengurangi lalu lintas antarzona untuk alasan keamanan, latensi, atau biaya.

Sumber daya

Praktik terbaik Well-Architected terkait:

- Definisi Ketersediaan
- REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan
- REL11-BP04 Mengandalkan bidang data dan bukan bidang kendali selama pemulihan

Dokumen terkait:

- Meminimalkan Ketergantungan dalam Rencana Pemulihan Bencana
- Amazon Builders' Library: Stabilitas statis menggunakan Zona Ketersediaan
- Batas Isolasi Kesalahan
- Stabilitas statis menggunakan Zona Ketersediaan
- RDS Multi-Zona
- Meminimalkan Ketergantungan dalam Rencana Pemulihan Bencana
- Perutean DNS Lintas Wilayah
- Perutean Multiwilayah Amazon S3 MRAP
- AWS Global Accelerator
- Route 53 ARC
- Amazon S3 Zona Tunggal

- Penyeimbangan Beban Lintas Zona

Video terkait:

- Stabilitas statis di AWS: AWS re:Invent 2019: Memperkenalkan Amazon Builders' Library (DOP328)

Contoh terkait:

- Amazon Builders' Library: Stabilitas statis menggunakan Zona Ketersediaan

REL11-BP06 Mengirimkan notifikasi ketika peristiwa memengaruhi ketersediaan

Notifikasi dikirimkan setelah pelanggaran ambang batas terdeteksi, bahkan apabila peristiwa yang menyebabkan masalah tersebut sudah diatasi secara otomatis.

Pemulihan otomatis menjadikan beban kerja Anda andal. Namun, kemampuan ini juga menyembunyikan masalah dasar yang perlu diatasi. Implementasikan pemantauan peristiwa yang baik agar Anda dapat mendeteksi pola masalah, termasuk masalah yang ditangani oleh pemulihan otomatis, sehingga Anda dapat mengatasi akar masalahnya.

Sistem yang tangguh dirancang sedemikian rupa sehingga peristiwa degradasi langsung dikomunikasikan kepada tim yang tepat. Notifikasi ini harus dikirim melalui satu atau banyak saluran komunikasi.

Hasil yang diinginkan: Pemberitahuan langsung dikirim ke tim operasi ketika ambang batas dilanggar, seperti tingkat kesalahan, latensi, atau metrik indikator kinerja utama (KPI) penting lainnya, sehingga masalah ini diselesaikan sesegera mungkin dan dampak terhadap pengguna dapat dicegah atau diminimalkan.

Antipola umum:

- Mengirimkan terlalu banyak alarm.
- Mengirimkan alarm yang tidak dapat ditindaklanjuti.
- Mengatur ambang alarm terlalu tinggi (terlalu sensitif) atau terlalu rendah (kurang sensitif).
- Tidak mengirimkan alarm untuk dependensi eksternal.

- Tidak mempertimbangkan kegagalan abu-abu saat merancang pemantauan dan alarm.
- Melakukan otomatisasi pemulihan, tetapi tidak memberikan notifikasi kepada tim yang tepat bahwa pemulihan diperlukan.

Manfaat menjalankan praktik terbaik ini: Notifikasi pemulihan membuat tim operasional dan bisnis menyadari adanya degradasi layanan sehingga mereka dapat segera bereaksi untuk meminimalkan waktu deteksi rata-rata (MTTD) dan waktu perbaikan rata-rata (MTTR). Notifikasi peristiwa pemulihan juga menjamin bahwa Anda tidak mengabaikan masalah yang jarang terjadi.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang. Kegagalan mengimplementasikan mekanisme pemantauan dan notifikasi peristiwa secara tepat dapat mengakibatkan kegagalan dalam mendeteksi pola masalah, termasuk masalah yang ditangani oleh pemulihan otomatis. Sebuah tim hanya akan menyadari adanya degradasi sistem ketika pengguna menghubungi layanan pelanggan atau secara kebetulan.

Panduan implementasi

Saat menetapkan strategi pemantauan, alarm yang dipicu adalah peristiwa umum. Peristiwa ini kemungkinan berisi pengidentifikasi untuk alarm, status alarm (seperti `IN ALARM` atau `OK`), dan detail mengenai pemicunya. Dalam banyak kasus, peristiwa alarm seharusnya dideteksi dan email notifikasi dikirimkan. Ini adalah contoh tindakan pada alarm. Notifikasi alarm sangat penting dalam hal observabilitas karena memberi tahu orang yang tepat bahwa terdapat sebuah masalah. Namun, ketika tindakan terhadap peristiwa sudah matang di dalam solusi observabilitas Anda, tindakan tersebut dapat secara otomatis memperbaiki masalah tanpa memerlukan campur tangan manusia.

Setelah alarm pemantauan KPI ditetapkan, peringatan seharusnya dikirimkan ke tim yang tepat ketika ambang batas terlampaui. Peringatan tersebut juga dapat digunakan untuk memicu proses otomatis yang akan mencoba memperbaiki degradasi.

Untuk pemantauan ambang batas yang lebih kompleks, alarm komposit harus dipertimbangkan. Alarm komposit menggunakan sejumlah alarm pemantauan KPI untuk membuat peringatan berdasarkan logika bisnis operasional. CloudWatch Alarm dapat dikonfigurasi untuk mengirimkan email, atau untuk mencatatkan insiden di dalam sistem pelacakan insiden pihak ketiga menggunakan integrasi Amazon SNS atau Amazon EventBridge.

Langkah implementasi

Buat berbagai jenis alarm berdasarkan bagaimana beban kerja dipantau, seperti:

- Alarm aplikasi digunakan untuk mendeteksi apabila ada bagian dari beban kerja Anda yang tidak berfungsi dengan baik.
- Alarm infrastruktur menunjukkan kapan sumber daya perlu diskalakan. Alarm dapat ditampilkan secara visual di dasbor, mengirimkan peringatan melalui Amazon SNS atau email, dan bekerja sama dengan Auto Scaling untuk mengecilkan atau memperluas skala sumber daya beban kerja.
- Alarm statis sederhana dapat dibuat untuk memantau apabila metrik melanggar ambang batas statis selama periode evaluasi tertentu.
- Alarm komposit dapat memperhitungkan alarm-alarm kompleks dari berbagai sumber.
- Setelah alarm dibuat, buat peristiwa notifikasi yang sesuai. Anda dapat langsung menginvokasi API Amazon SNS untuk mengirimkan notifikasi dan menautkan otomatisasi apa pun untuk perbaikan atau komunikasi.
- Integrasikan Amazon Health Aware untuk memungkinkan visibilitas pemantauan ke sumber daya AWS yang mungkin mengalami degradasi. Untuk beban kerja yang penting untuk bisnis, solusi ini menyediakan akses ke peringatan proaktif dan waktu nyata untuk layanan AWS.

Sumber daya

Praktik terbaik Well-Architected terkait:

- Definisi Ketersediaan

Dokumen terkait:

- Membuat Alarm CloudWatch Berdasarkan Ambang Batas Statis
- Apa Itu Amazon EventBridge?
- Apa Itu Amazon Simple Notification Service?
- Memublikasikan Metrik Kustom
- Menggunakan Alarm Amazon CloudWatch
- Amazon Health Aware (AHA)
- Menyiapkan Alarm komposit CloudWatch
- Apa yang baru di Observabilitas AWS pada re:Invent 2022

Alat terkait:

- CloudWatch
- CloudWatch X-Ray

REL11-BP07 Merancang produk Anda agar memenuhi target ketersediaan dan perjanjian tingkat layanan (SLA) waktu aktif

Rancang produk Anda agar memenuhi target ketersediaan dan perjanjian tingkat layanan (SLA) waktu aktif. Jika Anda memublikasi atau secara pribadi menyetujui target ketersediaan atau SLA yang berlaku, verifikasikan bahwa proses operasional dan arsitektur Anda didesain untuk mendukungnya.

Hasil yang diinginkan: Setiap aplikasi memiliki target yang ditetapkan untuk ketersediaan dan SLA untuk metrik performa, yang dapat dipantau dan dipertahankan untuk memenuhi hasil bisnis.

Antipola umum:

- Mendesain dan melakukan deployment beban kerja tanpa menetapkan SLA apa pun.
- Metrik SLA ditetapkan ke tingkat tinggi tanpa rasional atau persyaratan bisnis.
- Menetapkan SLA tanpa memperhitungkan dependensi dan SLA yang mendasarinya.
- Desain aplikasi dibuat tanpa mempertimbangkan Model Tanggung Jawab Bersama untuk Ketangguhan.

Manfaat menjalankan praktik terbaik ini: Mendesain aplikasi berdasarkan target ketangguhan utama membantu Anda memenuhi tujuan bisnis dan ekspektasi pelanggan. Tujuan ini membantu mendorong proses desain aplikasi yang mengevaluasi berbagai macam teknologi dan mempertimbangkan beragam kompromi.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Desain aplikasi harus memperhitungkan berbagai rangkaian persyaratan yang didapatkan dari tujuan bisnis, operasional, dan finansial. Dalam persyaratan operasional, beban kerja harus memiliki target metrik ketangguhan tertentu sehingga dapat dipantau dan dukung dengan sesuai. Metrik ketangguhan tidak boleh ditetapkan atau didapatkan setelah melakukan deployment beban kerja. Metrik ketangguhan harus ditetapkan selama fase desain dan membantu memandu berbagai keputusan dan kompromi.

- Setiap beban kerja harus memiliki rangkaian metrik ketangguhannya sendiri. Metrik-metrik tersebut mungkin berbeda dari aplikasi bisnis yang lain.
- Mengurangi dependensi dapat memiliki dampak positif pada ketersediaan. Setiap beban kerja harus mempertimbangkan dependensinya serta SLA-nya. Secara umum, pilih dependensi dengan target ketersediaan yang setara dengan atau lebih besar dari target beban kerja Anda.
- Pertimbangkan desain yang dipasangkan secara longgar sehingga beban kerja Anda dapat beroperasi dengan benar meskipun ada gangguan dependensi, apabila mungkin.
- Kurangi dependensi bidang kendali, terutama selama pemulihan atau degradasi. Evaluasi desain yang secara statis stabil untuk beban kerja yang penting bagi misi. Gunakan penghematan sumber daya untuk meningkatkan ketersediaan dependensi tersebut di beban kerja.
- Observabilitas dan instrumentasi sangat penting untuk mencapai SLA dengan mengurangi Waktu Rata-Rata ke Deteksi (MTTD) dan Waktu Rata-Rata ke Perbaikan (MTTR).
- Kegagalan lebih jarang (MTBF lebih lama), waktu deteksi kegagalan lebih pendek (MTTD lebih singkat), dan waktu perbaikan lebih singkat (MTTR lebih singkat) adalah tiga faktor yang digunakan untuk meningkatkan ketersediaan di sistem terdistribusi.
- Menetapkan dan memenuhi metrik ketangguhan untuk beban kerja merupakan fondasi dari desain yang efektif. Desain tersebut harus memperhitungkan kompromi terkait kompleksitas desain, dependensi layanan, performa, penskalaan, dan biaya.

Langkah implementasi

- Tinjau dan dokumentasikan desain beban kerja sambil mempertimbangkan pertanyaan berikut:
  - Di mana bidang kendali digunakan di beban kerja?
  - Bagaimana beban kerja mengimplementasikan toleransi kesalahan?
  - Apa saja pola desain untuk penskalaan, penskalaan otomatis, redundansi, dan komponen dengan ketersediaan tinggi?
  - Apa saja persyaratan untuk ketersediaan dan konsistensi data?
  - Apakah ada pertimbangan untuk penghematan sumber daya atau stabilitas statis sumber daya?
  - Apa saja dependensi layanan?
- Tetapkan metrik SLA berdasarkan arsitektur beban kerja sambil bekerja sama dengan para pemangku kepentingan. Pertimbangkan SLA semua dependensi yang digunakan oleh beban kerja.
- Setelah target SLA ditetapkan, optimalkan arsitektur untuk memenuhi SLA.

- Setelah desain yang akan memenuhi SLA dibuat, implementasikan perubahan operasional, otomatisasi proses, dan runbook yang juga akan memiliki fokus pada pengurangan MTTD dan MTTR.
- Setelah di-deploy, pantau dan laporkan SLA.

Sumber daya

Praktik Terbaik Terkait:

- REL03-BP01 Memilih cara untuk menyegmentasi beban kerja
- REL10-BP01 Melakukan deployment beban kerja ke beberapa lokasi
- REL11-BP01 Memantau semua komponen beban kerja untuk mendeteksi kegagalan
- REL11-BP03 Mengotomatisasi pemulihan di semua lapisan
- REL12-BP05 Menguji ketahanan menggunakan chaos engineering
- REL13-BP01 Tetapkan sasaran pemulihan untuk waktu henti dan kehilangan data
- Memahami kesehatan beban kerja

Dokumen terkait:

- Ketersediaan dengan redundansi
- Pilar keandalan - Ketersediaan
- Mengukur ketersediaan
- Batas Isolasi Kesalahan AWS
- Model Tanggung Jawab Bersama untuk Ketangguhan
- Stabilitas statis menggunakan Zona Ketersediaan
- Perjanjian Tingkat Layanan (SLA) AWS
- Panduan untuk Arsitektur Berbasis Sel di AWS
- Infrastruktur AWS
- Laporan resmi Pola Ketangguhan Multi-AZ Lanjutan

Layanan terkait:

- Amazon CloudWatch

- AWS Config
- AWS Trusted Advisor

Uji keandalan

Setelah Anda merancang beban kerja agar tangguh terhadap tekanan produksi, pengujian adalah satu-satunya cara untuk memastikan bahwa beban kerja akan beroperasi sesuai desain, dengan ketangguhan yang diharapkan.

Lakukan pengujian untuk memvalidasi bahwa beban kerja Anda memenuhi persyaratan fungsional dan nonfungsional, karena bug atau bottleneck kinerja dapat memengaruhi keandalan beban kerja Anda. Uji ketangguhan beban kerja Anda untuk membantu Anda menemukan bug laten yang hanya muncul dalam produksi. Lakukan pengujian ini secara rutin.

Praktik Terbaik

- REL12-BP01 Menggunakan buku pedoman untuk menyelidiki kegagalan
- REL12-BP02 Menjalankan analisis setelah insiden
- REL12-BP03 Menguji persyaratan fungsional
- REL12-BP04 Menguji persyaratan penskalaan dan kinerja
- REL12-BP05 Menguji ketahanan menggunakan chaos engineering
- REL12-BP06 Mengadakan game day secara rutin

REL12-BP01 Menggunakan buku pedoman untuk menyelidiki kegagalan

Dokumentasikan proses penyelidikan di buku pedoman agar dapat memberikan respons yang cepat dan konsisten terhadap skenario kegagalan yang tidak benar-benar dipahami. Buku pedoman adalah langkah-langkah yang telah ditetapkan di awal untuk mengidentifikasi faktor yang menyebabkan skenario kegagalan. Hasil dari langkah proses apa pun digunakan untuk menentukan langkah berikutnya yang akan dilakukan sampai masalah diidentifikasi atau dieskalasi.

Buku pedoman adalah perencanaan proaktif yang harus Anda lakukan, agar Anda dapat mengambil tindakan reaktif secara efektif. Ketika skenario kegagalan yang tidak tercakup dalam buku pedoman dialami di lingkungan produksi, tangani masalah terlebih dahulu (padamkan api). Lalu lihat kembali langkah-langkah yang telah Anda ambil untuk mengatasi masalah tersebut dan gunakan untuk menambahkan entri baru dalam buku pedoman.

Ingat bahwa buku pedoman digunakan untuk merespons insiden tertentu, sedangkan runbook digunakan untuk mencapai hasil tertentu. Sering kali, runbook digunakan untuk untuk aktivitas rutin, dan buku pedoman digunakan untuk merespons peristiwa nonrutin.

Antipola umum:

- Berencana untuk melakukan deployment beban kerja tanpa mengetahui proses untuk mendiagnosis masalah atau merespons insiden.
- Keputusan yang tidak direncanakan tentang sistem mana saja yang dikumpulkan log dan metriknya saat menyelidiki peristiwa.
- Tidak mempertahankan metrik dan peristiwa cukup lama agar dapat mengambil data.

Manfaat menjalankan praktik terbaik ini: Pencatatan runbook memastikan prosedur dapat diikuti secara konsisten. Kodifikasi runbook membatasi munculnya kesalahan dari aktivitas manual. Buku pedoman otomatis dapat menghemat waktu respons peristiwa dengan menghilangkan keharusan campur tangan anggota tim atau memberikan informasi tambahan ketika campur tangan mereka dimulai.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

- Gunakan buku pedoman untuk mengidentifikasi masalah. Buku pedoman adalah proses yang didokumentasikan untuk menyelidiki masalah. Dokumentasikan proses penyelidikan di buku pedoman agar dapat memberikan respons yang cepat dan konsisten terhadap skenario kegagalan. Buku pedoman harus memuat informasi dan panduan yang dapat digunakan oleh orang yang cukup terampil untuk mengumpulkan informasi, mengidentifikasi potensi sumber kegagalan, mengisolasi kesalahan, dan menentukan faktor penyebabnya (lakukan analisis pascainsiden).
  - Implementasikan buku pedoman sebagai kode. Jalankan operasi sebagai kode dengan membuat skrip buku pedoman Anda untuk memastikan konsistensi dan mengurangi kesalahan yang disebabkan proses manual. Buku pedoman dapat terdiri dari beberapa skrip sesuai dengan banyaknya langkah yang diperlukan untuk mengidentifikasi faktor penyebab masalah. Aktivitas runbook dapat dipicu atau dijalankan sebagai bagian dari aktivitas buku pedoman, atau mempercepat eksekusi buku pedoman untuk merespons peristiwa yang teridentifikasi.
    - Otomatiskan buku pedoman operasional Anda dengan AWS Systems Manager
    - AWS Systems Manager Run Command
    - AWS Systems Manager Automation

- Apa itu AWS Lambda?
- Apa Itu Amazon EventBridge?
- Menggunakan Alarm Amazon CloudWatch

Sumber daya

Dokumen terkait:

- AWS Systems Manager Automation
- AWS Systems Manager Run Command
- Otomatiskan buku pedoman operasional Anda dengan AWS Systems Manager
- Menggunakan Alarm Amazon CloudWatch
- Menggunakan Canary (Amazon CloudWatch Synthetics)
- Apa Itu Amazon EventBridge?
- Apa itu AWS Lambda?

Contoh terkait:

- Mengotomatiskan operasi dengan Buku Pedoman dan Runbook

REL12-BP02 Menjalankan analisis setelah insiden

Tinjau peristiwa yang memengaruhi pelanggan, dan identifikasi faktor yang berkontribusi serta tindakan pencegahannya. Gunakan informasi ini untuk mengembangkan mitigasi guna meminimalkan atau mencegah kemungkinan terjadi lagi. Kembangkan prosedur untuk respons efektif dan cepat. Komunikasikan faktor yang berkontribusi dan tindakan koreksi yang diperlukan, yang disesuaikan dengan audiens target. Miliki metode untuk mengomunikasikan penyebab ini ke lainnya seperti yang diperlukan.

Menilai alasan mengapa pengujian yang ada tidak dapat menemukan masalahnya. Menambahkan pengujian untuk kasus ini jika pengujian belum ada.

Hasil yang diinginkan: Tim Anda memiliki pendekatan yang konsisten dan disepakati untuk menangani analisis pascainsiden. Salah satu mekanismenya adalah proses koreksi kesalahan (COE). Proses COE membantu tim Anda mengidentifikasi, memahami, dan mengatasi akar

penyebab insiden, sekaligus membangun mekanisme dan pagar pembatas untuk membatasi kemungkinan insiden yang sama terjadi lagi.

Antipola umum:

- Menemukan faktor-faktor yang berkontribusi, tetapi tidak terus-menerus mencari lebih dalam untuk masalah potensial dan pendekatan lainnya untuk memitigasi.
- Hanya mengidentifikasi penyebab kesalahan manusia, dan tidak memberikan pelatihan atau otomatisasi apa pun yang dapat mencegah kesalahan manusia.
- Fokus menyalahkan, bukan memahami akar penyebabnya, sehingga tercipta budaya ketakutan dan menghambat komunikasi terbuka
- Tidak berbagi wawasan, yang membuat temuan analisis insiden hanya diketahui kelompok kecil saja, sehingga orang lain tidak dapat belajar dari pengalaman tersebut
- Tidak ada mekanisme untuk mencatat pengetahuan institusional, sehingga wawasan yang berharga hilang karena pelajaran yang didapat tidak diabadikan dalam bentuk praktik terbaik yang diperbarui dan mengakibatkan insiden berulang dengan akar penyebab yang sama atau serupa

Manfaat menyusun praktik terbaik ini: Dengan melakukan analisis setelah insiden dan membagikan hasilnya, beban kerja lain akan dapat memitigasi risiko jika beban kerja sudah mengimplementasikan faktor penyumbang yang sama, sehingga mitigasi atau pemulihan otomatis dapat diimplementasikan sebelum insiden terjadi.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Analisis setelah insiden yang baik memberikan peluang untuk mengusulkan solusi umum terhadap masalah dengan pola arsitektur yang digunakan di tempat lainnya dalam sistem.

Dokumentasi dan penanganan masalah merupakan landasan proses COE. Sebaiknya tentukan cara standar untuk mendokumentasikan akar penyebab kritis, dan memastikan penyebab tersebut ditinjau dan ditangani. Tetapkan kepemilikan yang jelas untuk proses analisis setelah insiden. Tunjuk individu atau tim penanggung jawab yang akan mengawasi penyelidikan dan tindak lanjut insiden.

Dorong budaya yang berfokus pada pembelajaran dan peningkatan, bukan menyalahkan. Tekankan bahwa tujuannya adalah untuk mencegah insiden di kemudian hari, bukan untuk menghukum individu.

Kembangkan prosedur yang jelas untuk melakukan analisis setelah insiden. Prosedur ini harus menguraikan langkah-langkah yang harus diambil, informasi yang akan dikumpulkan, dan pertanyaan-pertanyaan penting yang harus dicari jawabannya selama analisis. Selidiki insiden secara menyeluruh, tidak hanya pada penyebab langsung guna mengidentifikasi akar penyebab dan faktor penyumbangnya. Gunakan teknik seperti Analisis Lima Mengapa untuk menggali lebih dalam masalah yang mendasarinya.

Simpan repositori pelajaran yang didapat dari analisis insiden. Pengetahuan institusional ini dapat digunakan sebagai referensi untuk insiden dan upaya pencegahan ke depannya. Bagikan temuan dan wawasan dari analisis setelah insiden, dan pertimbangkan untuk mengadakan pertemuan tinjauan setelah insiden terbuka untuk membahas pelajaran yang didapatkan.

Langkah implementasi

- Saat melakukan analisis setelah insiden, pastikan tidak menyalahkan siapa pun dalam proses tersebut. Dengan begitu, orang-orang yang terlibat dalam insiden tersebut bersikap rasional terhadap tindakan korektif yang diusulkan dan mendorong penilaian mandiri yang jujur serta kolaborasi di seluruh tim.
- Tentukan cara standar untuk mendokumentasikan masalah kritis. Contoh struktur untuk dokumen tersebut:
  - Apa yang terjadi?
  - Apa dampaknya terhadap pelanggan dan bisnis Anda?
  - Apa akar penyebabnya?
  - Data apa yang Anda miliki untuk mendukung hal ini?
    - Misalnya, metrik dan grafik
  - Apa implikasi pilar kritis, terutama keamanan?
    - Saat merancang beban kerja, Anda memilah pilar-pilar sesuai dengan konteks bisnis Anda. Keputusan bisnis ini dapat menentukan prioritas rekayasa Anda. Anda dapat mengoptimalkan untuk mengurangi biaya dengan mengorbankan keandalan dalam lingkungan pengembangan, atau, untuk solusi yang sangat penting, Anda dapat mengoptimalkan keandalan dengan biaya yang lebih tinggi. Keamanan selalu menjadi hal yang didahulukan dan diutamakan, karena Anda harus melindungi pelanggan Anda.
  - Pelajaran apa hal yang Anda dapatkan?
  - Tindakan korektif apa yang Anda ambil?
    - Item tindakan

    - Item terkait
- Buat prosedur operasi standar yang jelas untuk melakukan analisis setelah insiden.
- Siapkan proses pelaporan insiden standar. Dokumentasikan semua insiden secara komprehensif, termasuk laporan insiden awal, log, komunikasi, dan tindakan yang diambil selama insiden.
- Ingatlah bahwa insiden tidak harus berupa terhentinya sistem. Insiden juga bisa berupa near-miss, atau performa sistem yang tidak sesuai harapan meski tetap memenuhi fungsi bisnisnya.
- Terus tingkatkan proses analisis setelah insiden Anda berdasarkan umpan balik dan pelajaran yang dipetik.
- Tangkap temuan utama dalam sistem manajemen pengetahuan, dan pertimbangkan pola apa pun yang perlu ditambahkan ke dalam panduan developer atau daftar periksa sebelum deployment.

Sumber daya

Dokumen terkait:

- Mengapa Anda harus mengembangkan koreksi kesalahan (COE)

Video terkait:

- Amazon’s approach to failing successfully
- AWS re:Invent 2021 - Amazon Builders’ Library: Operational Excellence at Amazon

REL12-BP03 Menguji persyaratan fungsional

Gunakan teknik seperti pengujian unit dan pengujian integrasi yang memvalidasi fungsionalitas.

Anda akan meraih hasil terbaik saat pengujian ini dijalankan secara otomatis sebagai bagian dari tindakan deployment dan build. Misalnya, dengan menggunakan AWS CodePipeline, developer melakukan perubahan ke repositori sumber tempat CodePipeline mendeteksi perubahan secara otomatis. Perubahan tersebut dibangun, dan pengujian dijalankan. Setelah pengujian selesai, kode yang dibangun di-deploy ke server penahapan untuk pengujian. Dari server penahapan, CodePipeline menjalankan lebih banyak pengujian, seperti integrasi atau pengujian beban. Setelah berhasil menyelesaikan pengujian tersebut, CodePipeline melakukan deployment kode yang telah diuji dan disetujui ke instans produksi.

Selain itu, pengalaman menunjukkan bahwa pengujian transaksi sintetis (juga disebut sebagai pengujian canary, tetapi bedakan dengan deployment canary) yang dapat menjalankan dan menyimulasikan perilaku pelanggan adalah salah satu proses pengujian yang paling penting. Jalankan pengujian ini secara konstan terhadap titik akhir beban kerja dari berbagai lokasi jarak jauh. Amazon CloudWatch Synthetics memungkinkan Anda untuk membuat canary untuk memantau titik akhir dan API.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Tinggi

Panduan implementasi

- Uji persyaratan fungsional. Hal ini termasuk pengujian unit dan pengujian integrasi yang memvalidasi fungsionalitas yang disyaratkan.
  - Gunakan CodePipeline dengan AWS CodeBuild untuk menguji kode dan menjalankan build
  - AWS CodePipeline Menambahkan Dukungan untuk Unit dan Pengujian Integrasi Kustom dengan AWS CodeBuild
  - Pengiriman Berkelanjutan dan Integrasi Berkelanjutan
  - Menggunakan Canary (Amazon CloudWatch Synthetics)
  - Otomatisasi uji perangkat lunak

Sumber daya

Dokumen terkait:

- Partner APN: partner yang dapat membantu implementasi pipeline integrasi berkelanjutan
- AWS CodePipeline Menambahkan Dukungan untuk Unit dan Pengujian Integrasi Kustom dengan AWS CodeBuild
- AWS Marketplace: produk yang dapat digunakan untuk integrasi berkelanjutan
- Pengiriman Berkelanjutan dan Integrasi Berkelanjutan
- Otomatisasi uji perangkat lunak
- Gunakan CodePipeline dengan AWS CodeBuild untuk menguji kode dan menjalankan build
- Menggunakan Canary (Amazon CloudWatch Synthetics)

REL12-BP04 Menguji persyaratan penskalaan dan kinerja

Gunakan teknik-teknik seperti pengujian beban untuk memvalidasi bahwa beban kerja memenuhi persyaratan kinerja dan penskalaan.

Di dalam cloud, Anda dapat membuat lingkungan pengujian dalam skala produksi sesuai permintaan untuk beban kerja Anda. Jika Anda menjalankan pengujian ini di infrastruktur yang skalanya diturunkan, Anda harus menskalakan hasil observasi Anda menurut apa yang Anda perkirakan terjadi di dalam produksi. Pengujian kinerja dan beban juga dapat dilakukan dalam produksi jika Anda ingin berhati-hati agar tidak berdampak pada pengguna aktual. Tandai data pengujian Anda agar tidak tercampur dengan data pengguna nyata dan mengubah laporan statistik atau produksi.

Dengan pengujian, Anda dapat memastikan bahwa sumber daya dasar, pengaturan penskalaan, kuota layanan, dan desain ketahanan Anda beroperasi sebagaimana mestinya saat menerima beban.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Tinggi

Panduan implementasi

- Uji persyaratan penskalaan dan kinerja. Jalankan pengujian beban untuk memvalidasi bahwa beban kerja memenuhi persyaratan kinerja dan penskalaan.
  - Pengujian Beban Terdistribusi di AWS: simulasikan ribuan pengguna terhubung
  - Apache JMeter
    - Lakukan deployment aplikasi ke lingkungan yang menyerupai lingkungan produksi Anda, lalu eksekusi pengujian beban.
      - Gunakan infrastruktur sebagai konsep kode untuk membuat lingkungan semirip mungkin dengan lingkungan produksi Anda.

Sumber daya

Dokumen terkait:

- Pengujian Beban Terdistribusi di AWS: simulasikan ribuan pengguna terhubung
- Apache JMeter

REL12-BP05 Menguji ketahanan menggunakan chaos engineering

Jalankan eksperimen chaos secara rutin di lingkungan yang berada dalam atau sedekat mungkin dengan produksi untuk memahami bagaimana sistem Anda merespons kondisi yang merugikan.

Hasil yang diinginkan:

Ketahanan beban kerja diverifikasi secara rutin dengan menerapkan chaos engineering dalam bentuk eksperimen injeksi kesalahan atau injeksi beban tak terduga. Selain itu, terdapat pengujian ketahanan yang memvalidasi perilaku sesuai ekspektasi yang diketahui dari beban kerja Anda selama berlangsungnya sebuah peristiwa. Gabungkan chaos engineering dan pengujian ketahanan agar Anda percaya bahwa beban kerja dapat bertahan dari kegagalan komponen dan dapat pulih dari gangguan tak terduga dengan dampak minimal atau tanpa dampak.

Antipola umum:

- Menentukan desain untuk mendapatkan ketahanan, tetapi tidak memverifikasi bagaimana beban kerja berfungsi secara keseluruhan saat terjadi kesalahan.
- Tidak pernah bereksperimen dalam kondisi dunia nyata dan dengan beban yang diharapkan.
- Tidak memperlakukan eksperimen Anda sebagai kode atau memeliharanya melalui siklus pengembangan.
- Tidak menjalankan eksperimen chaos baik sebagai bagian dari alur CI/CD Anda maupun di luar deployment.
- Tidak menggunakan analisis pascainsiden terdahulu saat menentukan kesalahan mana yang akan digunakan dalam eksperimen.

Manfaat menjalankan praktik terbaik ini: Injeksi kesalahan untuk memverifikasi ketahanan beban kerja Anda akan membuat Anda percaya bahwa prosedur pemulihan dari desain Anda yang tangguh akan efektif jika terjadi kesalahan nyata.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

Chaos engineering memberi tim Anda kemampuan untuk terus menginjeksi gangguan (simulasi) dunia nyata dengan cara yang terkontrol di tingkat penyedia layanan, infrastruktur, beban kerja, dan komponen, dengan dampak minimal atau tanpa dampak bagi pelanggan Anda. Hal ini

memungkinkan tim Anda belajar dari kesalahan serta mengamati, mengukur, dan meningkatkan ketahanan beban kerja Anda, serta memvalidasi bahwa peringatan akan diluncurkan dan tim mendapatkan notifikasi jika terjadi suatu peristiwa.

Jika dilakukan terus-menerus, chaos engineering dapat menunjukkan kekurangan dalam beban kerja Anda yang, jika dibiarkan tidak ditangani, dapat berdampak negatif pada ketersediaan dan pengoperasian.

> **Note**
>
> Chaos engineering adalah bidang ilmu yang bereksperimen pada sistem guna membangun kepercayaan pada kemampuan sistem untuk bertahan dari kondisi gangguan dalam produksi. – Prinsip-prinsip Chaos Engineering

Jika sistem mampu bertahan dari gangguan ini, eksperimen chaos harus dipertahankan sebagai pengujian regresi otomatis. Dengan demikian, eksperimen chaos harus dilakukan sebagai bagian dari siklus hidup pengembangan sistem (SDLC) Anda dan sebagai bagian dari alur CI/CD Anda.

Untuk memastikan bahwa beban kerja Anda dapat bertahan dari kegagalan komponen, lakukan injeksi peristiwa dunia nyata sebagai bagian dari eksperimen Anda. Misalnya, lakukan eksperimen dengan kehilangan instans Amazon EC2 atau failover instans basis data Amazon RDS utama, lalu verifikasi bahwa beban kerja Anda tidak terpengaruh (atau hanya sedikit terpengaruh). Gunakan kombinasi kesalahan komponen untuk menyimulasikan peristiwa yang mungkin disebabkan oleh gangguan di Zona Ketersediaan.

Untuk kesalahan tingkat aplikasi (seperti crash), Anda dapat memulai dengan stressor seperti kehabisan memori dan daya CPU.

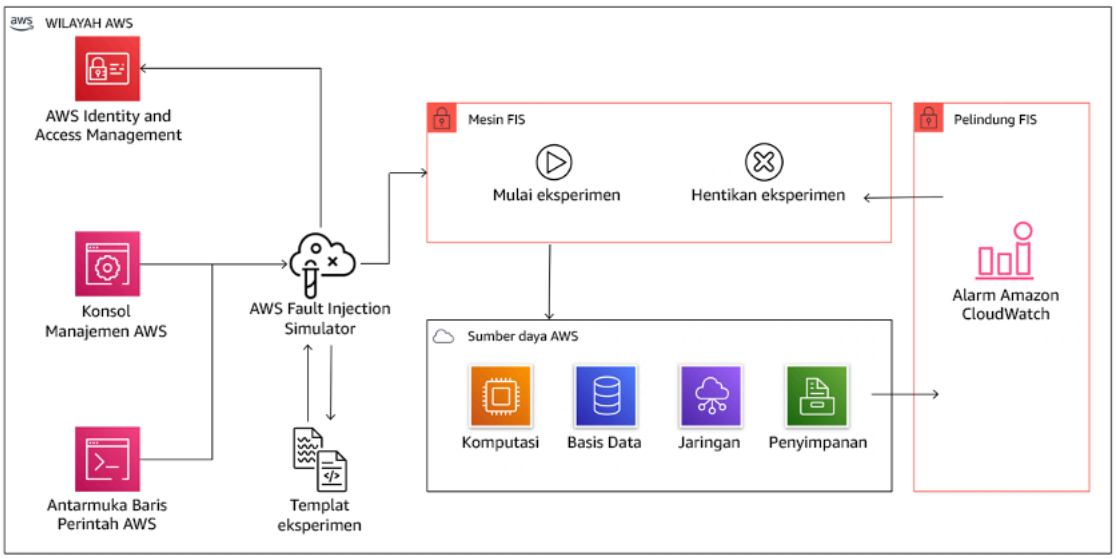
Untuk memvalidasi mekanisme fallback atau failover untuk dependensi eksternal karena gangguan jaringan yang terputus-putus, komponen Anda harus menyimulasikan peristiwa tersebut dengan memblokir akses ke penyedia pihak ketiga selama durasi tertentu yang dapat berlangsung dari hitungan detik hingga jam.

Mode degradasi lainnya dapat menyebabkan berkurangnya fungsionalitas dan respons yang lambat, sehingga sering kali mengakibatkan gangguan pada layanan Anda. Degradasi ini umumnya disebabkan oleh peningkatan latensi pada layanan yang sangat penting dan komunikasi jaringan yang tidak dapat diandalkan (paket yang tidak dikirim). Eksperimen dengan kesalahan ini, termasuk efek jaringan seperti latensi, pesan yang tidak terkirim, dan kegagalan DNS, dapat mencakup

ketidakmampuan untuk meresolusi nama, menjangkau layanan DNS, atau membuat koneksi ke layanan yang dependen.

Alat chaos engineering:

AWS Fault Injection Service (AWS FIS) adalah layanan terkelola penuh untuk menjalankan eksperimen injeksi kesalahan yang dapat digunakan sebagai bagian dari alur CD Anda, atau di luar alur. AWS FIS adalah pilihan yang baik untuk digunakan selama game day chaos engineering. Layanan ini mendukung penerapan kesalahan secara bersamaan di berbagai jenis sumber daya, termasuk Amazon EC2, Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), dan Amazon RDS. Kesalahan ini termasuk menghentikan sumber daya, memaksa failover, membebani CPU atau memori, throttling, latensi, dan kehilangan paket. Karena layanan ini terintegrasi dengan Amazon CloudWatch Alarms, Anda dapat mengatur kondisi berhenti sebagai pagar pembatas untuk melakukan rollback jika eksperimen menyebabkan dampak tak terduga.



AWS Fault Injection Service terintegrasi dengan sumber daya AWS untuk memungkinkan Anda menjalankan eksperimen injeksi kesalahan untuk beban kerja Anda.

Ada juga beberapa opsi pihak ketiga untuk eksperimen injeksi kesalahan. Opsi ini mencakup alat sumber terbuka seperti Chaos Toolkit, Chaos Mesh, dan Litmus Chaos, serta opsi komersial seperti Gremlin. Untuk memperluas cakupan kesalahan yang dapat diinjeksikan di AWS, AWS FIS terintegrasi dengan Chaos Mesh dan Litmus Chaos, sehingga Anda dapat mengoordinasikan alur kerja injeksi kesalahan di antara beberapa alat. Misalnya, Anda dapat menjalankan pengujian pada CPU sebuah pod menggunakan kesalahan Chaos Mesh atau Litmus sambil menghentikan sebagian simpul klaster yang dipilih secara acak menggunakan tindakan kesalahan AWS FIS.

Langkah implementasi

- Tentukan kesalahan mana yang akan digunakan untuk eksperimen.

  Lakukan penilaian desain beban kerja Anda untuk mengetahui ketahanannya. Desain tersebut (yang dibuat menggunakan praktik terbaik dari Well-Architected Framework) memperhitungkan risiko berdasarkan dependensi krusial, peristiwa terdahulu, masalah yang diketahui, dan persyaratan kepatuhan. Buat daftar yang berisi setiap elemen desain yang dimaksudkan untuk menjaga ketahanan dan kesalahan yang akan dimitigasi oleh elemen desain tersebut. Untuk informasi lebih lanjut tentang cara membuat daftar tersebut, lihat laporan resmi Operational Readiness Review yang memandu Anda tentang cara membuat proses untuk mencegah pengulangan insiden sebelumnya. Proses Analisis Mode dan Efek Kegagalan (FMEA) memberi Anda kerangka kerja untuk melakukan analisis tingkat komponen terhadap kegagalan dan bagaimana dampaknya terhadap beban kerja Anda. FMEA diuraikan secara lebih mendetail oleh Adrian Cockcroft dalam Failure Modes and Continuous Resilience.

- Tetapkan prioritas untuk setiap kesalahan.

  Mulailah dengan kategorisasi yang umum seperti tinggi, sedang, atau rendah. Untuk menilai prioritas, pertimbangkan frekuensi kesalahan dan dampak kegagalan terhadap beban kerja secara keseluruhan.

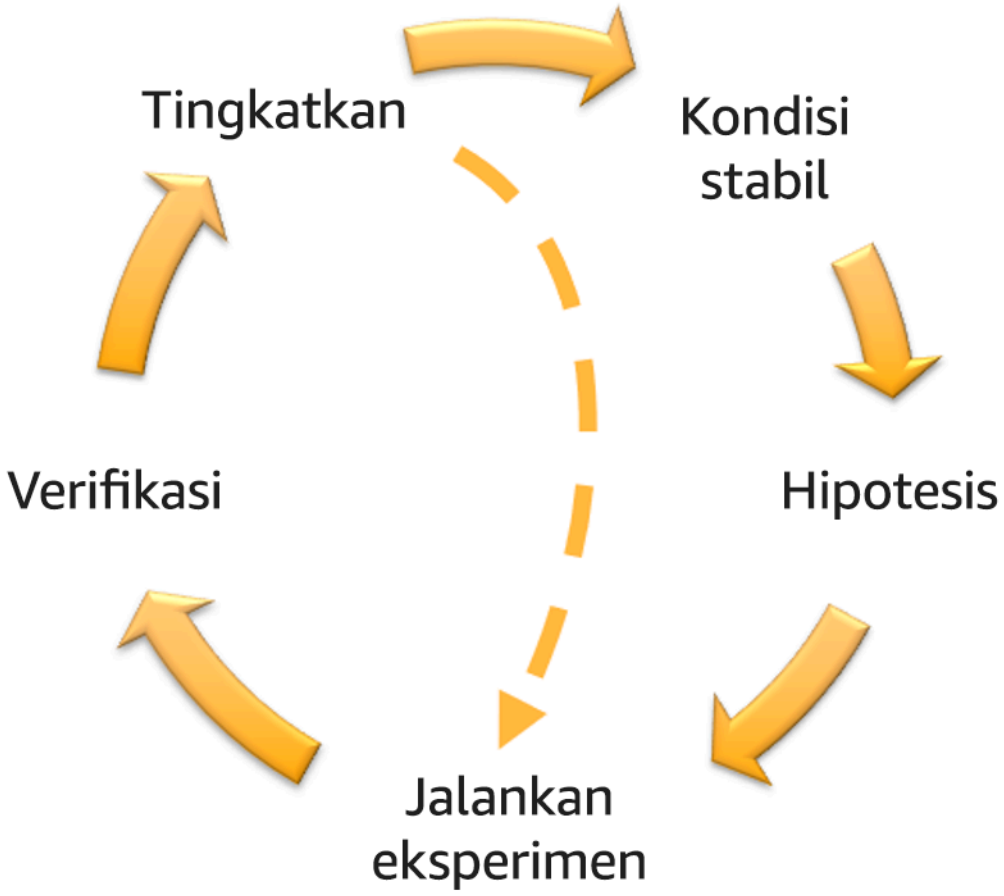
  Saat mempertimbangkan frekuensi kesalahan tertentu, lakukan analisis pada data terdahulu untuk beban kerja ini jika tersedia. Jika tidak tersedia, gunakan data dari beban kerja lain yang berjalan di lingkungan yang serupa.

  Ketika mempertimbangkan dampak dari kesalahan tertentu, makin besar cakupan kesalahan, biasanya makin besar dampaknya. Pertimbangkan juga desain dan tujuan beban kerja. Misalnya, kemampuan untuk mengakses penyimpanan data sumber sangat krusial untuk beban kerja yang melakukan transformasi dan analisis data. Dalam hal ini, Anda akan memprioritaskan eksperimen untuk kesalahan akses, serta akses yang di-throttling dan penyisipan latensi.

  Analisis pascainsiden adalah sumber data yang baik untuk memahami frekuensi dan dampak mode kegagalan.

  Gunakan prioritas yang ditetapkan untuk menentukan kesalahan mana yang akan digunakan terlebih dahulu dalam eksperimen beserta urutannya agar dapat mengembangkan eksperimen injeksi kesalahan baru.

- Untuk setiap eksperimen yang Anda lakukan, gunakan roda chaos engineering dan ketahanan berkelanjutan.



Roda chaos engineering dan ketahanan berkelanjutan yang menggunakan metode ilmiah dari Adrian Hornsby.

- Definisikan kondisi stabil sebagai output terukur dari beban kerja yang menunjukkan perilaku normal.

  Beban kerja Anda menunjukkan kondisi stabil jika beroperasi dengan andal dan seperti yang diharapkan. Oleh karena itu, validasikan bahwa beban kerja Anda berkondisi baik sebelum menentukan kondisi stabil. Dalam kondisi stabil, bukan berarti tidak akan ada dampak pada beban kerja saat terjadi kesalahan, karena sejumlah kesalahan tertentu mungkin berada dalam batas yang dapat diterima. Kondisi stabil adalah acuan dasar yang akan Anda amati selama eksperimen, yang akan menunjukkan anomali jika hipotesis yang Anda tentukan pada langkah berikutnya tidak berjalan seperti yang diharapkan.

Misalnya, kondisi stabil sistem pembayaran dapat didefinisikan sebagai pemrosesan 300 TPS dengan tingkat keberhasilan 99% dan waktu round-trip 500 md.

- Bentuk hipotesis tentang bagaimana beban kerja akan bereaksi terhadap kesalahan.

  Hipotesis yang baik didasarkan pada bagaimana beban kerja diharapkan akan memitigasi kesalahan untuk mempertahankan kondisi stabil. Hipotesis menyatakan bahwa dengan kesalahan jenis tertentu, sistem atau beban kerja akan terus berkondisi stabil karena beban kerja ini dirancang dengan mitigasi tertentu. Jenis spesifik kesalahan dan mitigasi harus ditentukan dalam hipotesis.

  Templat berikut dapat digunakan untuk hipotesis (tetapi pernyataan lain juga dapat diterima):

  > **Note**
  >
  > Jika *[kesalahan tertentu]* terjadi, beban kerja *[nama beban kerja]* akan *[deskripsikan kontrol mitigasi]* untuk mempertahankan *[dampak metrik bisnis atau teknis]*.

  Misalnya:

  - Jika 20% dari total simpul dalam grup simpul Amazon EKS dihapus, Transaction Create API akan terus melayani persentil ke-99 dari permintaan dalam waktu kurang dari 100 md (kondisi stabil). Simpul Amazon EKS akan pulih dalam waktu lima menit, dan pod akan dijadwalkan dan memproses lalu lintas dalam waktu delapan menit setelah dimulainya eksperimen. Peringatan akan diaktifkan dalam waktu tiga menit.
  - Jika terjadi kegagalan instans Amazon EC2 tunggal, pemeriksaan kondisi Elastic Load Balancing untuk sistem pemesanan akan membuat Elastic Load Balancing hanya mengirim permintaan ke instans berkondisi baik yang tersisa, sedangkan Amazon EC2 Auto Scaling mengganti instans yang gagal, sehingga mempertahankan peningkatan kesalahan sisi server (5xx) sebanyak kurang dari 0,01% (kondisi stabil).
  - Jika instans basis data Amazon RDS utama gagal, beban kerja pengumpulan data Rantai Pasokan akan melakukan failover dan terhubung ke instans basis data Amazon RDS yang siaga untuk mempertahankan kesalahan baca atau tulis basis data selama kurang dari 1 menit (kondisi stabil).

- Jalankan eksperimen dengan menginjeksikan kesalahan.

Eksperimen secara default harus memiliki kemampuan fail-safe dan ditoleransi oleh beban kerja. Jika Anda tahu bahwa beban kerja akan gagal, jangan jalankan eksperimen. Chaos engineering harus digunakan untuk menemukan “known-unknown” atau “unknown-unknown”. “Known-unknown” adalah hal-hal yang Anda ketahui, tetapi tidak sepenuhnya dipahami, dan “unknown-unknown” adalah hal-hal yang tidak Anda ketahui atau pahami sepenuhnya. Bereksperimen dengan beban kerja yang Anda tahu dalam kondisi rusak tidak akan memberi Anda wawasan baru. Eksperimen Anda harus direncanakan dengan cermat, memiliki cakupan dampak yang jelas, dan menyediakan mekanisme rollback yang dapat diterapkan jika terjadi gangguan tak terduga. Jika uji tuntas Anda menunjukkan bahwa beban kerja Anda dapat bertahan dalam eksperimen, lanjutkan eksperimen. Ada beberapa opsi untuk menginjeksikan kesalahan. Untuk beban kerja di AWS, AWS FIS menyediakan banyak simulasi kesalahan standar yang disebut tindakan. Anda juga dapat menentukan tindakan kustom yang berjalan di AWS FIS menggunakan dokumen AWS Systems Manager.

Kami tidak menyarankan penggunaan skrip kustom untuk eksperimen chaos, kecuali jika skrip tersebut memiliki kemampuan untuk memahami status terkini beban kerja, mampu menghasilkan log, dan menyediakan mekanisme untuk rollback dan kondisi berhenti jika memungkinkan.

Kerangka kerja atau kumpulan alat efektif yang mendukung chaos engineering harus melacak kondisi terkini eksperimen, menghasilkan log, dan menyediakan mekanisme rollback untuk mendukung pelaksanaan eksperimen yang terkontrol. Mulailah dengan layanan andal seperti AWS FIS yang memungkinkan Anda melakukan eksperimen dengan cakupan yang jelas dan mekanisme keamanan yang melakukan rollback jika eksperimen menimbulkan gangguan tak terduga. Untuk mempelajari tentang beragam variasi eksperimen menggunakan AWS FIS, lihat juga lab Aplikasi Tangguh dan Well-Architected dengan Chaos Engineering. Selain itu, AWS Resilience Hub akan menganalisis beban kerja Anda dan membuat eksperimen yang dapat Anda pilih untuk diterapkan dan dijalankan di AWS FIS.

> **Note**
>
> Untuk setiap eksperimen, pahami dengan jelas cakupan dan dampaknya. Kami merekomendasikan bahwa kesalahan harus disimulasikan terlebih dahulu di lingkungan nonproduksi sebelum dijalankan dalam produksi.

Eksperimen harus dijalankan dalam produksi dengan beban dunia nyata menggunakan deployment canary yang melakukan deployment sistem kontrol dan eksperimental, jika

memungkinkan. Menjalankan eksperimen selama waktu sepi adalah praktik yang baik untuk mengurangi potensi dampak saat pertama kali bereksperimen dalam produksi. Selain itu, jika menggunakan lalu lintas pelanggan yang sebenarnya akan menimbulkan terlalu banyak risiko, Anda dapat menjalankan eksperimen menggunakan lalu lintas sintetis di infrastruktur produksi terhadap deployment kontrol dan eksperimental. Jika tidak dapat menggunakan produksi, jalankan eksperimen di lingkungan praproduksi yang semirip mungkin dengan produksi.

Anda harus membuat dan memantau pagar pembatas untuk memastikan eksperimen tidak memengaruhi lalu lintas produksi atau sistem lain di luar batas yang dapat diterima. Tetapkan kondisi berhenti untuk menghentikan eksperimen jika mencapai ambang batas pada metrik pagar pembatas yang Anda tentukan. Hal ini harus mencakup metrik untuk kondisi stabil beban kerja, serta metrik berdasarkan komponen yang diinjeksi dengan kesalahan. Sebuah pemantauan sintetis (juga dikenal sebagai user canary) adalah salah satu metrik yang biasanya harus Anda sertakan sebagai proksi pengguna. Kondisi berhenti untuk AWS FIS didukung sebagai bagian dari templat eksperimen, sehingga memungkinkan maksimal lima kondisi berhenti per templat.

Salah satu prinsip chaos adalah meminimalkan cakupan eksperimen dan dampaknya:

Meskipun harus ada kelonggaran untuk beberapa dampak negatif dalam jangka pendek, Chaos Engineer bertanggung jawab dan berkewajiban untuk memastikan gangguan dari eksperimen diminimalkan dan dikendalikan.

Metode untuk memverifikasi cakupan dan dampak potensial adalah dengan melakukan eksperimen di lingkungan nonproduksi terlebih dahulu, memverifikasi bahwa ambang batas untuk kondisi berhenti diaktifkan seperti yang diharapkan selama eksperimen dan kemampuan pengamatan diterapkan untuk menemukan pengecualian, bukan langsung bereksperimen dalam produksi.

Saat menjalankan eksperimen injeksi kesalahan, verifikasikan bahwa semua pihak yang bertanggung jawab sudah mengetahui informasi yang jelas. Berkomunikasilah dengan tim yang sesuai seperti tim operasi, tim keandalan layanan, dan dukungan pelanggan untuk memberi tahu mereka kapan eksperimen akan dijalankan dan apa yang diharapkan. Berikan alat komunikasi kepada berbagai tim ini untuk memberi tahu tim tertentu yang menjalankan eksperimen jika muncul efek yang merugikan.

Anda harus memulihkan beban kerja dan sistem yang mendasarinya kembali ke kondisi awal yang diketahui berfungsi baik. Sering kali, desain beban kerja yang tangguh akan pulih sendiri.

Namun, beberapa desain yang salah atau eksperimen yang gagal dapat membuat beban kerja Anda berada dalam kondisi kegagalan yang tidak terduga. Pada akhir eksperimen, Anda harus menyadari hal ini dan memulihkan beban kerja dan sistem. Dengan AWS FIS, Anda dapat mengatur konfigurasi rollback (juga disebut post action) dalam parameter tindakan. Post action mengembalikan target ke keadaan sebelum tindakan dijalankan. Baik diotomatiskan (seperti menggunakan AWS FIS) maupun manual, post action ini harus menjadi bagian dari playbook yang menjelaskan cara mendeteksi dan menangani kegagalan.

- Verifikasikan hipotesisnya.

  Prinsip-prinsip Chaos Engineering memberikan panduan tentang cara memverifikasi kondisi stabil beban kerja Anda:

  Fokus pada output terukur dari suatu sistem, bukan atribut internal sistem. Pengukuran output tersebut selama periode waktu yang singkat merupakan proksi untuk kondisi stabil sistem. Throughput sistem secara keseluruhan, tingkat kesalahan, dan persentil latensi semuanya dapat menjadi metrik penting yang merepresentasikan perilaku kondisi stabil. Dengan berfokus pada pola perilaku sistemik selama eksperimen, chaos engineering memverifikasi bahwa sistem berfungsi, bukan mencoba memvalidasi cara kerjanya.

  Dalam dua contoh sebelumnya, kami menyertakan metrik kondisi stabil dengan peningkatan kesalahan sisi server (5xx) sebanyak kurang dari 0,01% serta kesalahan baca dan tulis basis data selama kurang dari satu menit.

  Kesalahan 5xx adalah metrik yang baik karena merupakan konsekuensi dari mode kegagalan yang akan dialami langsung oleh klien yang menggunakan beban kerja. Pengukuran kesalahan basis data cocok digunakan sebagai konsekuensi langsung dari kesalahan, tetapi juga harus dilengkapi dengan pengukuran dampak klien seperti permintaan pelanggan yang gagal atau kesalahan yang muncul bagi klien. Selain itu, sertakan pemantauan sintetis (juga dikenal sebagai user canary) pada API atau URI apa pun yang diakses langsung oleh klien yang menggunakan beban kerja Anda.

- Tingkatkan desain beban kerja agar memiliki ketahanan.

  Jika kondisi stabil tidak dipertahankan, selidiki cara desain beban kerja dapat ditingkatkan untuk mengurangi kesalahan, dengan menerapkan praktik terbaik dari pilar Keandalan AWS Well-Architected. Panduan dan sumber daya tambahan dapat ditemukan di AWS Builder's Library, yang berisi artikel tentang cara meningkatkan pemeriksaan kondisi Anda atau menerapkan percobaan ulang dengan backoff dalam kode aplikasi Anda, dll.

Setelah perubahan ini diterapkan, jalankan eksperimen lagi (ditunjukkan dengan garis putus-putus pada roda chaos engineering) untuk mengetahui keefektifannya. Jika langkah verifikasi menunjukkan bahwa hipotesisnya benar, beban kerja akan berada dalam kondisi stabil, dan siklusnya berlanjut.

- Jalankan eksperimen secara rutin.

  Eksperimen chaos adalah sebuah siklus, dan eksperimen harus dijalankan secara rutin sebagai bagian dari chaos engineering. Setelah beban kerja memenuhi hipotesis eksperimen, eksperimen harus diotomatiskan untuk terus berjalan sebagai bagian regresi dalam alur CI/CD Anda. Untuk mempelajari cara melakukannya, lihat blog tentang cara menjalankan eksperimen AWS FIS menggunakan AWS CodePipeline. Lab tentang eksperimen AWS FIS berulang dalam alur CI/CD memungkinkan Anda melakukan praktik langsung.

  Eksperimen injeksi kesalahan juga merupakan bagian dari game day (lihat REL12-BP06 Mengadakan game day secara rutin). Game day mensimulasikan kegagalan atau peristiwa untuk memverifikasi sistem, proses, dan respons tim. Tujuannya adalah untuk benar-benar menerapkan tindakan yang perlu dilakukan oleh tim seolah memang terjadi peristiwa yang tidak diharapkan.

- Catat dan simpan hasil eksperimen.

  Hasil eksperimen injeksi kesalahan harus dicatat dan dijadikan persisten. Sertakan semua data yang diperlukan (seperti waktu, beban kerja, dan kondisi) agar dapat menganalisis hasil dan tren eksperimen nantinya. Contoh hasilnya dapat mencakup tangkapan layar dasbor, dump CSV dari basis data metrik Anda, atau catatan ketik manual yang berisi peristiwa dan pengamatan dari eksperimen. Pencatatan log eksperimen dengan AWS FIS dapat menjadi bagian dari pencatatan data ini.

Sumber daya

Praktik terbaik terkait:

- REL08-BP03 Mengintegrasikan pengujian ketahanan sebagai bagian dari deployment Anda
- REL13-BP03 Menguji implementasi pemulihan bencana untuk memvalidasi implementasi

Dokumen terkait:

- Apa itu AWS Fault Injection Service?

- Apa itu AWS Resilience Hub?
- Prinsip-prinsip Chaos Engineering
- Chaos Engineering: Merencanakan eksperimen pertama Anda
- Rekayasa Ketahanan: Belajar untuk Mengatasi Kegagalan
- Kisah Chaos Engineering
- Menghindari fallback dalam sistem terdistribusi
- Deployment Canary untuk Eksperimen Chaos

Video terkait:

- AWS re:Invent 2020: Menguji ketahanan menggunakan chaos engineering (ARC316)
- AWS re:Invent 2019: Meningkatkan ketahanan dengan chaos engineering (DOP309-R1)
- AWS re:Invent 2019: Melakukan chaos engineering di dunia nirserver (CMY301)

Contoh terkait:

- Lab Well-Architected: Level 300: Pengujian Ketahanan Amazon EC2, Amazon RDS, dan Amazon S3
- Lab Chaos Engineering di AWS
- lab Aplikasi Tangguh dan Well-Architected dengan Chaos Engineering
- Lab Chaos Nirserver
- Lab Ukur dan Tingkatkan Ketahanan Aplikasi Anda dengan AWS Resilience Hub

Alat terkait:

- AWS Fault Injection Service
- AWS Marketplace: Platform Chaos Engineering Gremlin
- Chaos Toolkit
- Chaos Mesh
- Litmus

REL12-BP06 Mengadakan game day secara rutin

Manfaatkan game day untuk secara rutin melatih prosedur Anda dalam merespons peristiwa dan kegagalan. Buat game day semirip mungkin dengan produksi (termasuk lingkungan produksi) bersama orang-orang yang akan terlibat dalam skenario kegagalan aktual. Game day menerapkan tindakan yang diperlukan guna memastikan peristiwa produksi tidak berdampak pada pengguna.

Game day menyimulasikan kegagalan atau peristiwa untuk menguji respons tim, sistem, dan proses. Tujuannya adalah untuk benar-benar menerapkan tindakan yang perlu dilakukan oleh tim seolah memang terjadi peristiwa yang tidak diharapkan. Hal ini akan membantu Anda memahami sisi mana yang perlu ditingkatkan dan membantu mengembangkan pengalaman organisasi dalam menangani peristiwa. Aktivitas ini harus dilakukan secara rutin untuk memperkuat memori otot dalam merespons kejadian tersebut.

Setelah desain ketangguhan Anda diterapkan dan diuji dalam lingkungan nonproduksi, game day dapat menjadi cara untuk memastikan bahwa segala sesuatu akan berjalan sesuai rencana ketika produksi. Game day, terutama yang dilakukan untuk pertama kali, merupakan aktivitas “wajib untuk semua tim”. Rekayasawan dan operasi akan diberitahu kapan ini dilakukan, dan apa yang akan terjadi. Runbook telah diterapkan. Simulasi peristiwa, termasuk peristiwa kegagalan yang mungkin terjadi, dieksekusi di sistem produksi dengan cara yang sudah ditentukan, dan dampaknya dievaluasi. Jika sistem beroperasi sesuai rancangan, deteksi dan pemulihan mandiri akan berlangsung dengan sedikit atau tanpa dampak. Namun, jika timbul dampak negatif, pengujian akan diulang dan masalah beban kerja diperbaiki, secara manual jika perlu (menggunakan runbook). Karena game day biasanya berlangsung di dalam produksi, semua pencegahan harus dilakukan guna memastikan bahwa ketersediaan untuk pelanggan tidak terganggu.

Antipola umum:

- Mendokumentasikan prosedur Anda, tetapi tidak pernah melatihnya.
- Tidak melibatkan pembuat keputusan bisnis dalam pengujian pelatihan.

Manfaat menerapkan praktik terbaik ini: Mengadakan game day secara rutin memastikan bahwa staf mengikuti kebijakan dan prosedur ketika insiden aktual terjadi, dan memvalidasi bahwa kebijakan dan prosedur tersebut sudah sesuai.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Sedang

Panduan implementasi

- Jadwalkan game day untuk menggunakan runbook dan buku pedoman Anda secara rutin. Game day harus mengikutsertakan semua orang yang akan terlibat dalam kejadian produksi: pemilik bisnis, staf pengembangan, staf operasional, dan tim respons insiden.
  - Jalankan pengujian beban atau kinerja Anda, kemudian jalankan injeksi kegagalan.
  - Cari anomali dalam runbook Anda dan peluang untuk menggunakan buku pedoman Anda.
    - Jika Anda tidak mengikuti runbook, perbaiki runbook atau koreksi perilakunya. Jika Anda menggunakan buku pedoman, identifikasi buku pedoman yang seharusnya digunakan atau buat yang baru.

Sumber daya

Dokumen terkait:

- Apa itu AWS GameDay?

Video terkait:

- AWS re:Invent 2019: Meningkatkan ketahanan dengan chaos engineering (DOP309-R1)

  Contoh terkait:

- Lab AWS Well-Architected - Pengujian Ketangguhan

Rencanakan Pemulihan Bencana (DR)

Memiliki cadangan dan komponen beban kerja berlebih adalah awal strategi DR Anda. RTO dan RPO adalah sasaran untuk pemulihan beban kerja Anda. Atur hal ini berdasarkan kebutuhan bisnis. Implementasikan strategi sesuai sasaran ini, dengan mempertimbangkan lokasi dan fungsi data serta sumber daya beban kerja. Probabilitas gangguan dan biaya pemulihan juga merupakan faktor penting yang membantu memahami nilai bisnis dari penyediaan pemulihan bencana untuk beban kerja.

Ketersediaan dan Pemulihan Bencana sama-sama mengandalkan praktik terbaik seperti pemantauan kegagalan, deployment ke beberapa lokasi, dan failover otomatis. Namun, Ketersediaan

berfokus pada komponen beban kerja, sedangkan Pemulihan Bencana berfokus pada salinan diskret dari keseluruhan beban kerja. Pemulihan Bencana memiliki tujuan yang berbeda dari Ketersediaan, yang berfokus pada waktu pemulihan setelah bencana.

Praktik Terbaik

- REL13-BP01 Tetapkan sasaran pemulihan untuk waktu henti dan kehilangan data
- REL13-BP02 Menggunakan strategi pemulihan yang ditentukan untuk memenuhi sasaran pemulihan
- REL13-BP03 Menguji implementasi pemulihan bencana untuk memvalidasi implementasi
- REL13-BP04 Mengelola penyimpangan konfigurasi di lokasi atau Wilayah Pemulihan Bencana (DR)
- REL13-BP05 Mengotomatiskan pemulihan

REL13-BP01 Tetapkan sasaran pemulihan untuk waktu henti dan kehilangan data

Beban kerja memiliki sasaran waktu pemulihan (RTO) dan sasaran titik pemulihan (RPO).

Sasaran Waktu Pemulihan (RTO) adalah penundaan maksimum yang dapat diterima antara gangguan layanan dan pemulihan layanan. Ini menentukan apa yang dianggap sebagai jendela waktu yang dapat diterima ketika layanan tidak tersedia.

Sasaran Titik Pemulihan (RPO) adalah jumlah waktu maksimum yang dapat diterima sejak titik pemulihan data terakhir. Ini menentukan apa yang dianggap sebagai kehilangan data yang dapat diterima antara titik pemulihan terakhir dan gangguan layanan.

Nilai RTO dan RPO merupakan pertimbangan penting ketika memilih strategi Pemulihan Bencana (DR) yang sesuai untuk beban kerja Anda. Sasaran-sasaran ini ditentukan oleh bisnis, kemudian digunakan oleh tim teknis untuk memilih dan mengimplementasikan strategi DR.

Hasil yang Diinginkan:

Setiap beban kerja memiliki penetapan RTO dan RPO, yang ditetapkan berdasarkan dampak bisnis. Beban kerja ditetapkan ke tingkat yang telah ditetapkan sebelumnya, yang menetapkan ketersediaan layanan dan kehilangan data yang dapat diterima, dengan RTO dan RPO terkait. Jika penetapan tingkat tersebut tidak dapat dilakukan, maka ini dapat diberi tingkat khusus yang disesuaikan per beban kerja, dengan maksud untuk membuat tingkat di lain waktu. RTO dan RPO digunakan sebagai salah satu pertimbangan utama untuk pemilihan implementasi strategi pemulihan

bencana untuk beban kerja. Pertimbangan tambahan dalam memilih strategi DR yakni kendala biaya, ketergantungan beban kerja, dan persyaratan operasional.

Untuk RTO, pahami dampak berdasarkan durasi pemadaman. Apakah implikasinya linier, atau adakah implikasi non-linier? (contohnya, setelah empat jam, Anda mematikan jalur produksi sampai dimulainya giliran kerja berikutnya).

Matriks pemulihan bencana, seperti berikut ini, dapat membantu Anda memahami bagaimana kritikalitas beban kerja berkaitan dengan sasaran pemulihan. (Perhatikan, nilai aktual untuk sumbu X dan Y harus disesuaikan dengan kebutuhan organisasi Anda).

Matriks Pemulihan Bencana						
		Sasaran Titik Pemulihan				
		< 1 Menit	< 1 Jam	< 6 Jam	< 1 Hari	+ 1 Hari
Sasaran Waktu Pemulihan	< 10 Menit	Kritis	Kritis	Tinggi	Sedang	Sedang
	< 2 Jam	Kritis	Tinggi	Sedang	Sedang	Rendah
	< 8 Jam	Tinggi	Sedang	Sedang	Rendah	Rendah
	< 24 Jam	Sedang	Sedang	Rendah	Rendah	Rendah
	Lebih dari 24 Jam	Sedang	Rendah	Rendah	Rendah	Rendah

Gambar 16: Matriks pemulihan bencana

Antipola umum:

- Tidak ditetapkan sasaran pemulihan.
- Memilih sasaran pemulihan semaunya.
- Memilih sasaran pemulihan yang terlalu longgar dan tidak memenuhi tujuan bisnis.
- Tidak memahami dampak waktu henti dan kehilangan data.
- Memilih sasaran pemulihan yang tidak realistis, seperti tanpa adanya waktu untuk pemulihan dan tanpa adanya kehilangan data, yang mungkin tidak dapat dicapai untuk konfigurasi beban kerja Anda.
- Memilih sasaran pemulihan yang lebih ketat daripada tujuan bisnis yang sesungguhnya. Ini memaksakan implementasi DR yang lebih mahal dan lebih rumit dibandingkan yang dibutuhkan beban kerja.
- Memilih sasaran pemulihan yang tidak kompatibel dengan sasaran beban kerja yang bergantung.

- Sasaran pemulihan Anda tidak mempertimbangkan persyaratan kepatuhan terhadap peraturan.
- RTO dan RPO ditetapkan untuk beban kerja, tetapi tidak pernah diuji.

Manfaat menerapkan praktik terbaik ini: Sasaran pemulihan Anda untuk waktu dan kehilangan data diperlukan untuk memandu implementasi DR Anda.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Tinggi

Panduan implementasi

Untuk beban kerja tertentu, Anda harus memahami dampak waktu henti dan kehilangan data pada bisnis Anda. Umumnya, dampak akan semakin meningkat jika waktu henti atau kehilangan data semakin besar, tetapi bentuk peningkatan ini bisa berbeda, tergantung pada jenis beban kerjanya. Contohnya, Anda mungkin dapat menoleransi waktu henti hingga satu jam dengan dampak kecil, tetapi setelah itu dampaknya meningkat dengan cepat. Ada banyak bentuk dampak pada bisnis, termasuk kerugian moneter (seperti hilangnya pendapatan), hilangnya kepercayaan pelanggan (dan dampak pada reputasi), masalah operasional (seperti penurunan produktivitas atau gaji tidak terbayarkan), dan risiko yang terkait dengan peraturan. Gunakan langkah-langkah berikut untuk memahami dampak-dampak ini, dan tetapkan RTO dan RPO untuk beban kerja Anda.

Langkah Implementasi

1. Tentukan pemangku kepentingan bisnis Anda untuk beban kerja ini, dan libatkan mereka untuk mengimplementasikan langkah-langkah ini. Sasaran pemulihan untuk beban kerja merupakan keputusan bisnis. Kemudian tim teknis bekerja dengan pemangku kepentingan bisnis untuk menggunakan sasaran-sasaran ini untuk memilih strategi DR.

   > **Note**
   >
   > Untuk langkah 2 dan 3, Anda dapat menggunakan the section called “Lembar kerja implementasi”.

2. Kumpulkan informasi yang diperlukan untuk mengambil keputusan dengan menjawab pertanyaan-pertanyaan di bawah ini.
3. Apakah Anda memiliki kategori atau tingkat kritikalitas untuk dampak beban kerja di organisasi Anda?

   a. Jika ya, tetapkan beban kerja ini ke salah satu kategori

   b. Jika tidak, maka tetapkan kategori-kategori ini. Buat lima kategori atau lebih sedikit dan sempurnakan rentang sasaran waktu pemulihan Anda untuk setiap kategori. Contoh kategori antara lain: kritis, tinggi, sedang, rendah. Untuk memahami cara pemetaan beban kerja ke kategori, pertimbangkan apakah beban kerja itu kritis untuk misi perusahaan, penting bagi bisnis, atau tidak mendorong bisnis.

   c. Tetapkan RTO dan RPO beban kerja berdasarkan kategori. Selalu pilih kategori yang lebih ketat (RTO dan RPO lebih rendah) daripada nilai mentah yang dihitung saat memasuki langkah ini. Jika ini menghasilkan perubahan nilai yang besar dan tidak sesuai, maka pertimbangkan untuk membuat kategori baru.

4. Berdasarkan jawaban-jawaban ini, tetapkan nilai RTO dan RPO ke beban kerja. Ini dapat dilakukan secara langsung, atau dengan menetapkan beban kerja ke tingkat layanan yang ditetapkan sebelumnya.
5. Dokumentasikan rencana pemulihan bencana (DRP) untuk beban kerja ini, yang merupakan bagian dari rencana keberlangsungan bisnis (BCP) organisasi Anda, di lokasi yang dapat diakses oleh pemangku kepentingan dan tim beban kerja

   a. Catat RTO dan RPO, dan informasi yang digunakan untuk menentukan nilai-nilai ini. Sertakan strategi yang digunakan untuk mengevaluasi dampak beban kerja pada bisnis

   b. Catat metrik lain selain RTO dan RPO yang Anda lacak, atau rencanakan untuk melacak sasaran pemulihan bencana

   c. Anda akan menambahkan detail strategi DR Anda dan runbook pada rencana ini ketika Anda membuat ini.

6. Dengan mencari kritikalitas beban kerja di dalam matriks seperti yang ada dalam Gambar 15, Anda dapat mulai menetapkan tingkat layanan yang ditetapkan di muka untuk organisasi Anda.
7. Setelah Anda mengimplementasikan strategi DR (atau bukti konsep untuk strategi DR) sesuai the section called “REL13-BP02 Menggunakan strategi pemulihan yang ditentukan untuk memenuhi sasaran pemulihan”, uji strategi ini untuk menentukan RPC (Kemampuan Titik Pemulihan) dan RTC (Kemampuan Waktu Pemulihan) aktual beban kerja. Jika ini tidak memenuhi sasaran pemulihan target, maka bekerjalah dengan pemangku kepentingan bisnis Anda untuk menyesuaikan sasaran-sasaran tersebut, atau buat perubahan pada strategi DR yang memungkinkan untuk memenuhi sasaran target.

Pertanyaan utama

1. Berapakah waktu henti maksimum untuk beban kerja sebelum timbul dampak serius pada bisnis?

   a. Tentukan kerugian moneter (dampak finansial langsung) pada bisnis per menit jika beban kerja terganggu.
   b. Pertimbangkan bahwa dampak tidak selalu linier. Pada awalnya, dampak bisa terbatas, tetapi kemudian meningkat dengan cepat melampaui titik kritis dalam waktu.
2. Berapakah jumlah data maksimum yang bisa hilang sebelum timbul dampak serius pada bisnis?
   a. Pertimbangkan nilai ini untuk penyimpanan data Anda yang paling kritis. Identifikasi kritikalitas masing-masing untuk penyimpanan data lainnya.
   b. Dapatkah data beban kerja dibuat jika hilang? Jika hal ini secara operasional lebih mudah daripada mencadangkan dan memulihkan, maka pilih RPO berdasarkan kritikalitas data sumber yang digunakan untuk membuat ulang data beban kerja.
3. Apa saja sasaran pemulihan dan harapan ketersediaan beban kerja yang hal ini andalkan (hilir), atau beban kerja yang mengandalkan hal ini (hulu)?
   a. Pilih sasaran pemulihan yang memampukan beban kerja ini untuk memenuhi persyaratan ketergantungan hulu
   b. Pilih sasaran pemulihan yang dapat dicapai mengingat kemampuan pemulihan ketergantungan hilir. Ketergantungan hilir non-kritis (yang dapat Anda “tangani”) dapat dikecualikan. Atau, bekerjalah dengan ketergantungan hilir kritis atau tingkatkan kemampuan pemulihannya apabila perlu.

Pertanyaan tambahan

Pertimbangkan pertanyaan-pertanyaan ini, dan bagaimana pertanyaan tersebut mungkin berlaku pada beban kerja ini:

4. Apakah Anda memiliki RTO dan RPO yang berbeda, tergantung pada jenis pemadaman (Wilayah vs. AZ, dll.)?
5. Apakah ada waktu spesifik (musim, acara penjualan, peluncuran produk) ketika RTO/RPO Anda mungkin berubah? Jika ya, apakah batas waktu dan pengukurannya yang berbeda?
6. Berapa jumlah pelanggan yang akan terkena dampak jika beban kerja terganggu?
7. Apakah dampak pada reputasi jika beban kerja terganggu?
8. Dampak operasional lain apakah yang dapat timbul jika beban kerja terganggu? Contohnya, dampak pada produktivitas karyawan jika sistem email tidak tersedia, atau jika sistem Gaji tidak dapat mengirimkan transaksi.

9. Bagaimanakah RTO dan RPO beban kerja sesuai dengan Strategi DR Organisasi dan Bidang Bisnis?
10. Apakah ada kewajiban kontrak internal untuk memberikan layanan? Apakah ada penalti jika tidak memenuhinya?
11. Apa saja kendala kepatuhan atau peraturan terkait data?

Lembar kerja implementasi

Anda dapat menggunakan lembar kerja ini untuk langkah implementasi 2 dan 3. Anda dapat menyesuaikan lembar kerja ini agar cocok dengan kebutuhan spesifik Anda, seperti menambahkan pertanyaan tambahan.

Langkah 2: Pertanyaan utama	Berlaku untuk beban kerja?	RTO beban kerja	RPO beban kerja	Penyesuaian RTO.	Penyesuaian RPO.	Instruksi
[1] waktu maksimum beban kerja dapat mengalami waktu henti						diukur pada waktunya dari mulai pemadaman hingga pemulihan
[2] jumlah maksimum data yang bisa hilang						diukur pada waktunya sejak set data dapat dipulihkan berkualitas baik terakhir diketahui
[3a] dependensi hulu						masukkan tujuan pemulihan hilir yang paling ketat
[3b] dependensi hilir						masukkan tujuan pemulihan hilir yang paling longgar
[3a] dependensi hulu yang direkonsiliasi						Jika nilai hulu kurang dari nilai saat ini dan nilai hilir lebih besar, bekerjalah dengan
[3b] dependensi hilir yang direkonsiliasi						dependensi untuk merekonsiliasi dan masukkan nilai yang direkonsiliasi di sini
[3] dependensi						turunkan nilai untuk memenuhi dependensi hulu atau naikkan berdasarkan kemampuan dependensi hilir
Langkah 2: Pertanyaan tambahan						Tunjukkan apakah pertanyaan berlaku. Jika tidak, lewati
RTO/RPO dasar						Turunkan nilai RTO dan RPO dari atas ke sini
[4] tipe pemadaman	[ ]Y / [ ]N					Masukkan tujuan pemulihan untuk tipe peristiwa dengan persyaratan paling ketat
[5] tujuan berbasis waktu khusus	[ ]Y / [ ]N					Masukkan tujuan pemulihan untuk waktu-waktu dengan persyaratan paling ketat
[6] pelanggan terganggu	[ ]Y / [ ]N					Buat grafik pelanggan yang terkena dampak saat fungsi mengalami waktu henti atau data hilang. Gunakan grafik tersebut untuk memasukkan RTO dan RPO maksimum yang diizinkan berdasarkan dampak pelanggan
[7] dampak reputasi	[ ]Y / [ ]N					Bekerjalah dengan bisnis untuk menentukan RTO dan RPO berdasarkan dampak terhadap reputasi
[8] dampak operasi	[ ]Y / [ ]N					Masukkan RTO dan RPO maksimum berdasarkan dampak operasional
[9] penyelarasan organisasi	[ ]Y / [ ]N					Masukkan RTO dan RPO maksimum untuk beban kerja tipe ini sesuai LOB dan persyaratan organisasi
[10] kewajiban kontrak	[ ]Y / [ ]N					Masukkan RTO dan RPO maksimum berdasarkan kewajiban kontrak
[11] kepatuhan peraturan	[ ]Y / [ ]N					Masukkan RTO dan RPO maksimum berdasarkan kepatuhan peraturan yang berlaku
target berdasarkan pertanyaan tambahan						Ambil nilai minimum (nilai yang lebih ketat) dari Q 4-11 dan masukkan di sini
target yang disesuaikan						Jika tujuan di baris di atas tidak dapat diakomodasi, bekerjalah dengan pemangku kepentingan untuk mengurai hambatan, dan masukkan jumlah minimum baru di sini
RTO/RPO yang disesuaikan						Masukkan nilai RPO/RTO acuan, atau target yang disesuaikan, mana saja yang lebih rendah
Langkah 3						
Peta ke kategori atau tingkat yang ditetapkan sebelumnya						Sesuaikan kedua nilai ke bawah (lebih ketat) agar selaras dengan tingkat yang ditetapkan terdekat

Lembar kerja

Tingkat upaya untuk Rencana Implementasi: Rendah

Sumber daya

Praktik Terbaik Terkait:

- the section called “REL09-BP04 Melakukan pemulihan data secara berkala untuk memverifikasi integritas dan proses pencadangan”
- the section called “REL13-BP02 Menggunakan strategi pemulihan yang ditentukan untuk memenuhi sasaran pemulihan”
- the section called “REL13-BP03 Menguji implementasi pemulihan bencana untuk memvalidasi implementasi”

Dokumen terkait:

- Blog Arsitektur AWS: Seri Pemulihan Bencana
- Pemulihan Bencana Beban Kerja di AWS: Pemulihan di Cloud (Laporan Resmi AWS)
- Mengelola kebijakan ketangguhan dengan Pusat Ketangguhan AWS
- Partner APN: partner yang dapat membantu pemulihan bencana
- AWS Marketplace: produk yang dapat digunakan untuk pemulihan bencana

Video terkait:

- AWS re:Invent 2018: Pola Arsitektur untuk Aplikasi Aktif-Aktif Multi-Wilayah (ARC209-R2)
- Pemulihan Bencana Beban Kerja di AWS

REL13-BP02 Menggunakan strategi pemulihan yang ditentukan untuk memenuhi sasaran pemulihan

Tentukan strategi pemulihan bencana (DR) yang memenuhi sasaran pemulihan beban kerja. Pilih strategi seperti pencadangan dan pemulihan, standby (aktif/pasif), atau aktif/aktif.

Hasil yang diinginkan: Strategi DR ditentukan dan diimplementasikan untuk setiap beban kerja agar beban kerja dapat mencapai sasaran DR. Strategi DR antara beban kerja menggunakan pola yang dapat digunakan kembali (seperti strategi yang telah dijelaskan sebelumnya),

Antipola umum:

- Mengimplementasikan prosedur pemulihan yang tidak konsisten untuk beban kerja dengan sasaran DR yang serupa.
- Membiarkan strategi DR diimplementasikan secara ad-hoc saat bencana terjadi.

- Tidak memiliki rencana untuk pemulihan bencana.
- Dependensi pada operasi bidang kendali selama pemulihan.

Manfaat menjalankan praktik terbaik ini:

- Dengan strategi pemulihan yang ditentukan, Anda dapat menggunakan prosedur tes dan peralatan umum.
- Menggunakan strategi pemulihan yang ditentukan akan meningkatkan penyebaran pengetahuan antara tim dan implementasi DR pada beban kerja milik mereka.

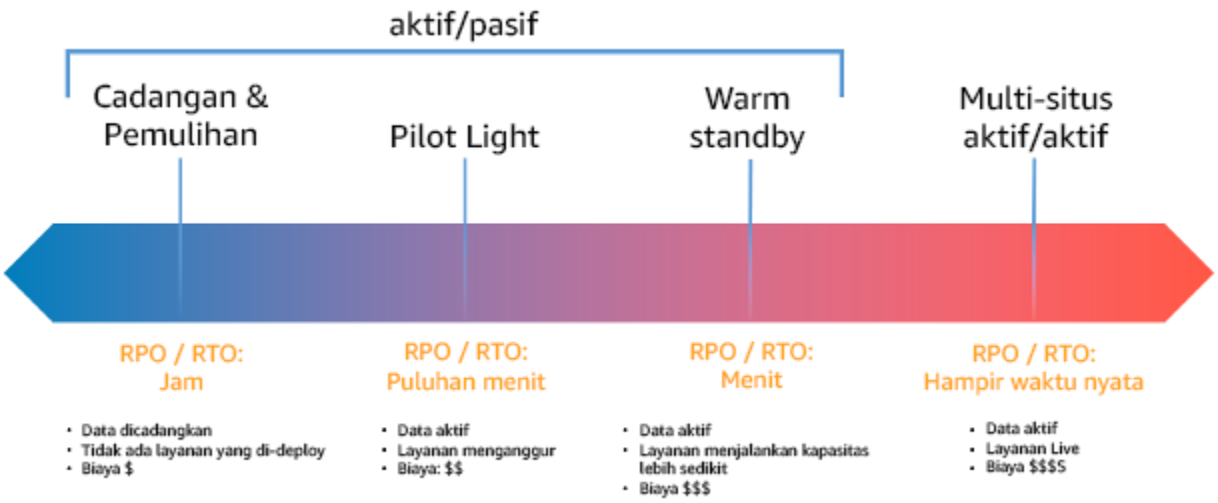
Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi. Tanpa strategi DR yang direncanakan, diimplementasikan, dan diuji, Anda akan kesulitan mencapai sasaran pemulihan ketika bencana terjadi.

Panduan implementasi

Strategi DR mengandalkan kemampuan untuk mempertahankan beban kerja di situs pemulihan jika lokasi utama tidak dapat menjalankan beban kerja. Sasaran pemulihan yang paling umum adalah RTO dan RPO, seperti yang didiskusikan dalam REL13-BP01 Tetapkan sasaran pemulihan untuk waktu henti dan kehilangan data.

Strategi DR di beberapa Zona Ketersediaan (AZ) dalam Wilayah AWS tunggal, dapat menyediakan mitigasi bencana seperti kebakaran, banjir, dan pemadaman listrik besar-besaran. Anda dapat menggunakan strategi DR yang menggunakan beberapa Wilayah jika memang perlu mengimplementasikan perlindungan terhadap peristiwa yang membuat beban kerja tidak dapat dijalankan di Wilayah AWS.

Anda harus memilih salah satu dari strategi berikut saat merancang strategi DR di beberapa Wilayah. Strategi didaftar dan diurutkan berdasarkan biaya dan kompleksitas dari kecil ke besar, serta diurutkan berdasarkan RTO dan RPO dari besar ke kecil. Wilayah Pemulihan berarti Wilayah AWS selain dari yang utama yang digunakan untuk beban kerja Anda.



Gambar 17: Strategi pemulihan bencana (DR)

- Pencadangan dan pemulihan (RPO dalam jam, RTO dalam 24 jam atau kurang): Cadangkan data dan aplikasi ke dalam Wilayah pemulihan. Menggunakan pencadangan otomatis atau berkelanjutan dapat mengaktifkan pemulihan titik waktu, yang dalam beberapa kasus dapat menurunkan RPO hingga 5 menit. Saat terjadi bencana, Anda akan melakukan deployment infrastruktur (menggunakan infrastruktur sebagai kode untuk mengurangi RTO), melakukan deploymennt kode, dan memulihkan data yang dicadangkan untuk memulihkan dari bencana di Wilayah pemulihan.
- Pilot light (RPO dalam menit, RTO dalam kelipatan sepuluh menit): Sediakan salinan infrastruktur beban kerja inti di Wilayah pemulihan. Replikasikan data ke Wilayah pemulihan dan buat cadangan di sana. Sumber daya yang diperlukan untuk mendukung replikasi dan pencadangan data, misalnya basis data dan penyimpanan objek, selalu aktif. Elemen lainnya seperti server aplikasi atau komputasi nirserver tidak di-deploy, tetapi dapat dibuat saat dibutuhkan dengan kode aplikasi dan konfigurasi yang diperlukan.
- Warm standby (RPO dalam detik, RTO dalam menit): Pertahankan beban kerja dalam versi yang diturunkan skalanya tetapi berfungsi sepenuhnya yang selalu dijalankan di Wilayah pemulihan. Sistem bisnis kritis sepenuhnya digandakan dan selalu diaktifkan, tetapi dengan armada yang diturunkan skalanya. Data direplikasi dan berada dalam Wilayah pemulihan. Saat memasuki waktu pemulihan, sistem dinaikkan skalanya dengan cepat untuk menangani beban produksi. Semakin warm standby dinaikkan skalanya, akan semakin rendah pengandalan RTO dan bidang kendali. Ketika diskalakan sepenuhnya, ini disebut sebagai hot standby.

- Multi-Wilayah (multi-situs) aktif-aktif (RPO mendekati nol, RTO berpotensi nol): Beban kerja di-deploy ke, dan aktif menangani lalu lintas dari, beberapa Wilayah AWS. Strategi ini perlu menyinkronkan data di seluruh Wilayah. Konflik potensial yang disebabkan oleh menulis catatan yang sama di dua replika wilayah yang berbeda harus dihindari atau ditangani, karena bisa menjadi kompleks. Replikasi data bermanfaat untuk sinkronisasi data dan akan melindungi Anda terhadap beberapa jenis bencana, tetapi tidak melindungi terhadap kerusakan atau kehilangan data kecuali solusi juga disertai opsi untuk pemulihan titik waktu.

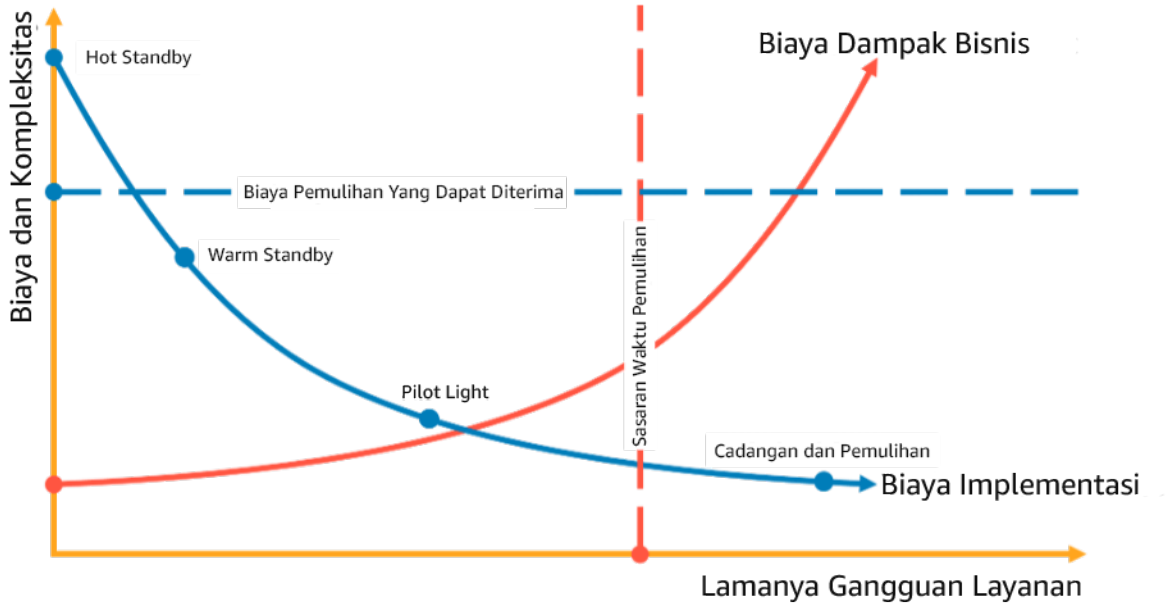
> **Note**
>
> Perbedaan antara pilot light dan warm standby terkadang sulit dimengerti. Keduanya menyertakan lingkungan di Wilayah pemulihan dengan salinan aset wilayah utama. Perbedaannya adalah pilot light tidak dapat memproses permintaan tanpa lebih dulu melakukan tindakan tambahan, sedangkan warm standby dapat menangani lalu lintas (pada kapasitas yang dikurangi) dengan cepat. Pilot light mengharuskan Anda mengaktifkan server, menaikkan skala, dan mungkin mengharuskan Anda melakukan deployment infrastruktur tambahan (bukan inti). Sementara itu, warm standby hanya meminta Anda untuk menaikkan skala (semuanya sudah di-deploy dan dijalankan). Pilih berdasarkan kebutuhan RTO dan RPO Anda.
>
> Apabila ada kekhawatiran tentang biaya, dan Anda ingin mencapai sasaran RPO dan RTO yang serupa dengan yang ditetapkan dalam strategi warm standby, Anda dapat mempertimbangkan solusi cloud native, seperti AWS Elastic Disaster Recovery, yang mengambil pendekatan pilot light dan menawarkan target RPO dan RTO lebih baik.

Langkah implementasi

1. Tentukan strategi DR yang akan memenuhi persyaratan pemulihan untuk beban kerja ini.

Saat memilih strategi DR, Anda harus memilih antara mengurangi waktu henti dan kehilangan data (RTO dan RPO) dan meningkatkan biaya dan kompleksitas untuk mengimplementasikan strategi, atau sebaliknya. Sebaiknya hindari strategi yang lebih sulit dari yang dibutuhkan, karena hal ini akan menambah biaya yang tidak perlu.

Misalnya, dalam diagram berikut, bisnis telah menentukan RTO maksimum yang diizinkan serta batas yang dapat digunakan pada strategi pemulihan layanan. Berdasarkan sasaran bisnis, strategi DR pilot light atau warm standby akan memenuhi kriteria biaya dan RTO.



Gambar 18: Pemilihan strategi DR berdasarkan RTO dan biaya

Untuk mempelajari selengkapnya, lihat Rencana Keberlangsungan Bisnis (BCP).

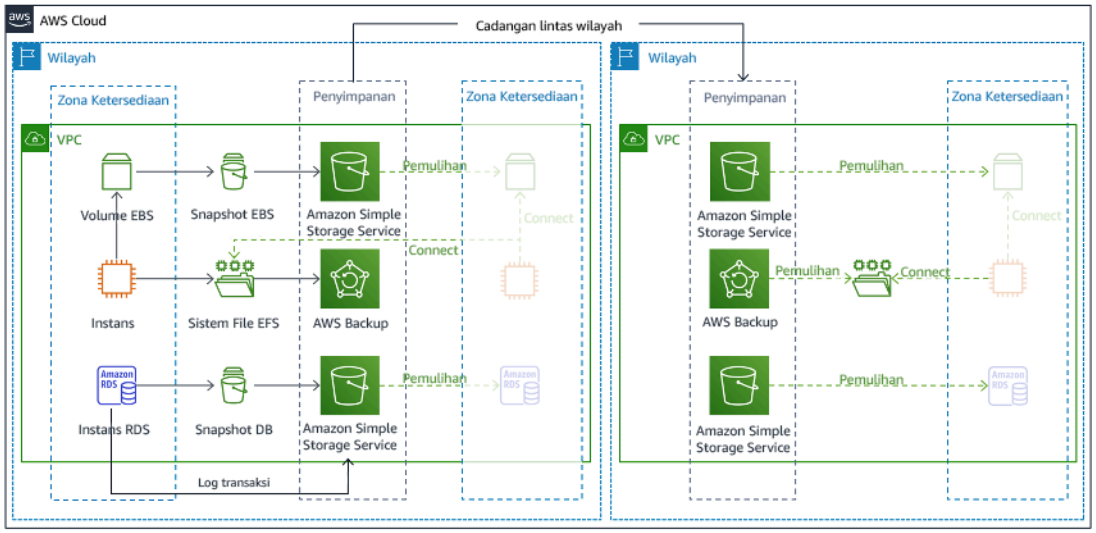
2. Tinjau pola tentang bagaimana strategi DR yang dipilih dapat diimplementasikan.

Langkah ini digunakan untuk memahami cara Anda mengimplementasikan strategi yang dipilih. Strategi dijelaskan menggunakan Wilayah AWS sebagai situs utama dan pemulihan. Namun, Anda juga dapat memilih untuk menggunakan Zona Ketersediaan dalam Wilayah tunggal sebagai strategi DR, yang menggunakan beberapa elemen dari berbagai strategi tersebut.

Dalam langkah berikut ini, Anda dapat menerapkan strategi pada beban kerja spesifik Anda.

Pencadangan dan pemulihan

Pencadangan dan pemulihan adalah strategi yang tidak terlalu kompleks untuk diimplementasikan, tetapi akan memerlukan waktu dan usaha lebih untuk mengembalikan beban kerja, sehingga RTO dan RPO menjadi lebih tinggi. Sebaiknya selalu buat cadangan data, dan salin cadangan tersebut ke situs lain (misalnya Wilayah AWS lain).

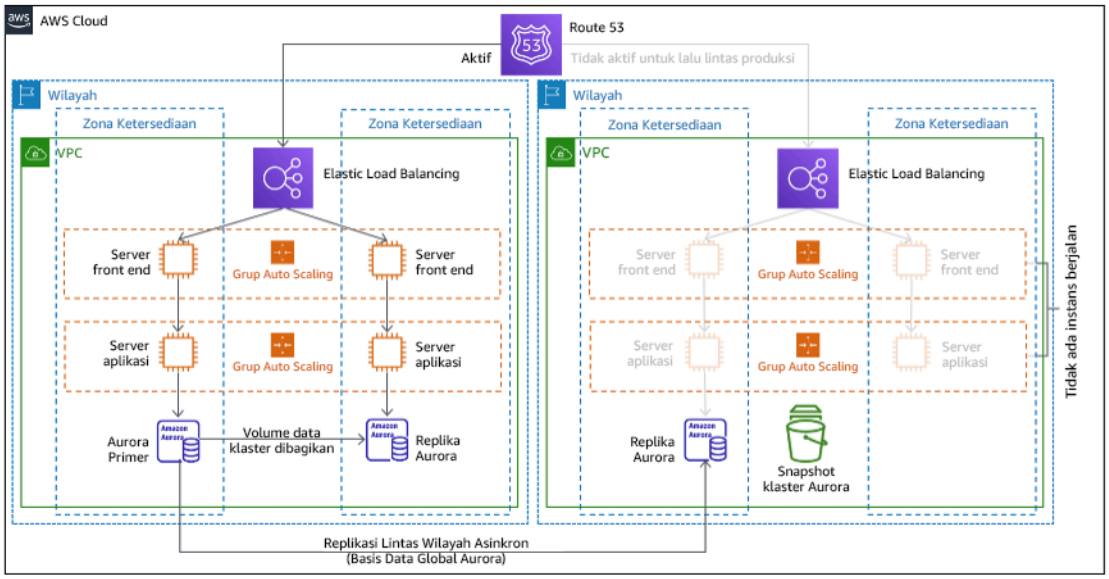


Gambar 19: Arsitektur pencadangan dan pemulihan

Untuk detail selengkapnya tentang strategi ini, lihat [Arsitektur Pemulihan Bencana (DR) di AWS, Bagian II: Pencadangan dan Pemulihan dengan Pemulihan Cepat].

Pilot light

Dengan pendekatan pilot light, Anda mereplikasi data dari Wilayah utama ke Wilayah pemulihan. Sumber daya inti yang digunakan untuk infrastruktur beban kerja di-deploy di Wilayah pemulihan. Namun, sumber daya tambahan dan dependensi lainnya masih diperlukan untuk membuat tumpukan fungsional ini. Misalnya, dalam gambar 20, tidak ada instans komputasi yang di-deploy.

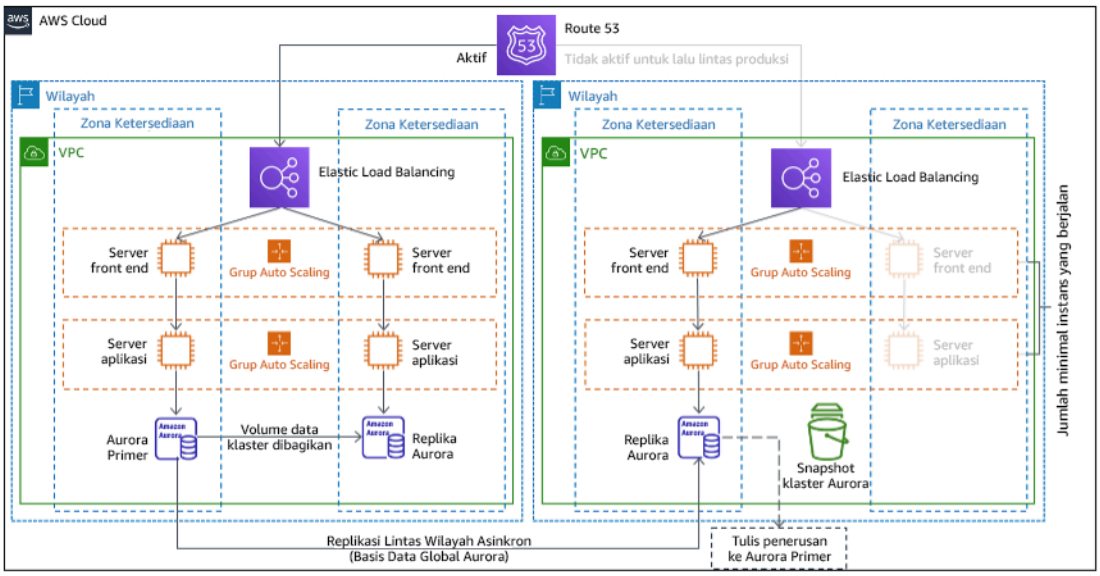


Gambar 20: Arsitektur pilot light

Untuk detail selengkapnya tentang strategi ini, lihat Arsitektur Pemulihan Bencana (DR) di AWS, Bagian III: Pilot Light dan Warm Standby.

Warm standby

Pendekatan warm standby memastikan ada salinan lingkungan produksi yang skalanya diturunkan tetapi berfungsi sepenuhnya di Wilayah lainnya. Pendekatan ini memperpanjang konsep pilot light dan mempercepat waktu pemulihan karena beban kerja selalu aktif di Wilayah lainnya. Jika Wilayah pemulihan di-deploy pada kapasitas penuh, hal ini disebut dengan hot standby.



Gambar 21: Arsitektur warm standby

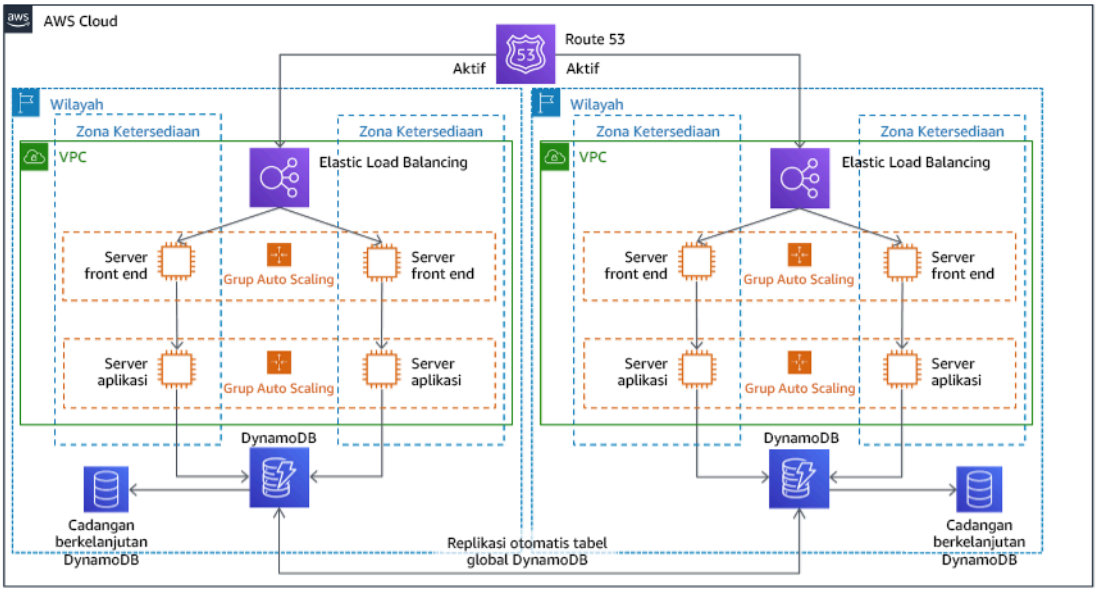
Saat menggunakan warm standby atau pilot light, Anda perlu menaikkan skala sumber daya di Wilayah pemulihan. Untuk memverifikasi kapasitas tersedia ketika diperlukan, pertimbangkan penggunaan reservasi kapasitas untuk instans EC2. Jika menggunakan AWS Lambda, maka konkurensi yang disediakan dapat menyediakan lingkungan pelaksanaan sehingga siap untuk merespons dengan segera ke panggilan fungsi Anda.

Untuk detail selengkapnya tentang strategi ini, lihat Arsitektur Pemulihan Bencana (DR) di AWS, Bagian III: Pilot Light dan Warm Standby.

Multi-situs aktif/aktif

Anda dapat menjalankan beban kerja secara bersamaan di beberapa Wilayah sebagai bagian dari strategi multi-situs aktif/aktif. Multi-situs aktif/aktif menjalankan lalu lintas dari semua wilayah ke

wilayah tempatnya di-deploy. Pelanggan dapat memilih strategi ini untuk alasan selain DR. Strategi ini dapat digunakan untuk meningkatkan ketersediaan, atau saat melakukan deployment beban kerja ke audiens global (untuk menempatkan titik akhir lebih dekat dengan pengguna dan/atau melakukan deployment tumpukan yang dilokalkan untuk audiens di wilayah tersebut). Sebagai strategi DR, jika beban kerja tidak dapat didukung di salah satu dari Wilayah AWS tempatnya di-deploy, Wilayah tersebut dievakuasi, dan Wilayah sisanya digunakan untuk mempertahankan ketersediaan. Multi-situs aktif/aktif adalah strategi DR yang paling sulit dioperasikan, dan sebaiknya hanya dipilih saat persyaratan bisnis mengharuskannya.



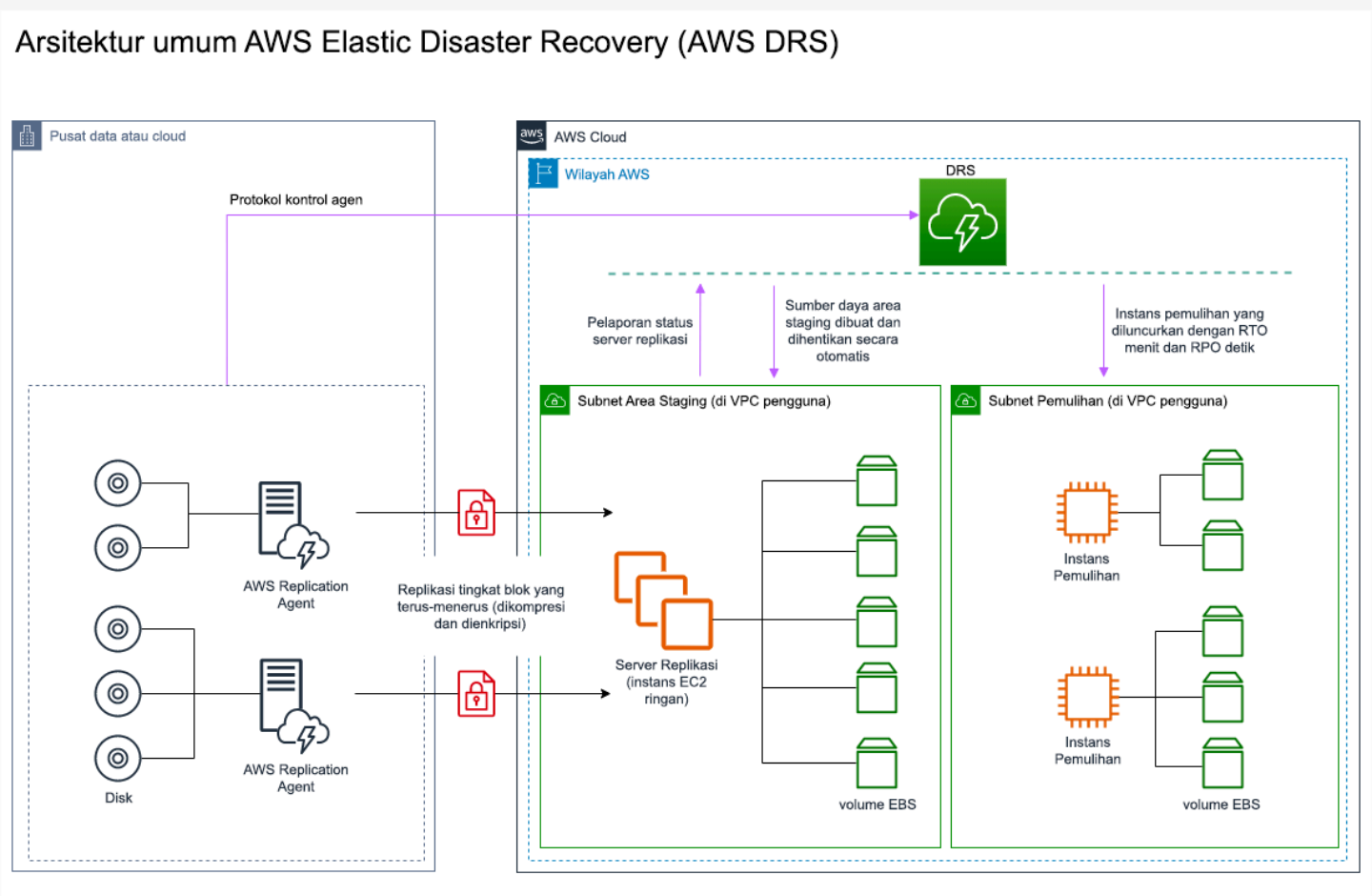
Gambar 22: Arsitektur multi-situs aktif/aktif

Untuk detail selengkapnya tentang strategi ini, lihat Arsitektur Pemulihan Bencana (DR) di AWS, Bagian IV: Multi-situs Aktif/Aktif.

AWS Elastic Disaster Recovery

Jika Anda mempertimbangkan strategi pilot light atau warm standby untuk pemulihan bencana, AWS Elastic Disaster Recovery dapat memberikan pendekatan alternatif dengan peningkatan manfaat. Elastic Disaster Recovery dapat menawarkan target RPO dan RTO yang serupa dengan warm standby, tetapi mempertahankan pendekatan pilot light dengan biaya rendah. Elastic Disaster Recovery mereplikasi data Anda dari wilayah utama ke Wilayah pemulihan, menggunakan perlindungan data berkelanjutan untuk mencapai RPO yang diukur dalam detik dan RTO yang dapat diukur dalam menit. Hanya sumber daya yang diperlukan untuk mereplikasi data yang di-deploy di wilayah pemulihan, yang menekan biaya tetap rendah, serupa dengan strategi pilot light. Ketika

menggunakan Elastic Disaster Recovery, layanan mengoordinasi dan mengatur pemulihan sumber daya komputasi ketika dimulai sebagai bagian dari failover atau latihan.



Gambar 23: arsitektur AWS Elastic Disaster Recovery

Praktik tambahan untuk melindungi data

Dengan semua strategi, Anda juga harus melakukan mitigasi terhadap bencana data. Replikasi data berkelanjutan melindungi Anda terhadap beberapa jenis bencana, tetapi tidak melindungi terhadap kerusakan atau kehilangan data kecuali strategi juga disertai versioning data yang disimpan atau opsi pemulihan titik waktu. Selain replika, Anda juga harus mencadangkan data yang direplikasi di situs pemulihan untuk membuat pencadangan titik waktu.

Menggunakan beberapa Zona Ketersediaan (AZ) dalam Wilayah AWS tunggal

Saat menggunakan beberapa AZ dalam Wilayah tunggal, implementasi DR Anda menggunakan beberapa elemen dari strategi di atas. Anda harus terlebih dahulu membuat arsitektur ketersediaan

tinggi (HA) menggunakan beberapa AZ yang ditampilkan dalam Gambar 23. Arsitektur ini memanfaatkan pendekatan multi-situs aktif/aktif, karena instans Amazon EC2 dan Penyeimbang Beban Elastis memiliki sumber daya yang di-deploy di beberapa AZ, yang secara aktif menangani permintaan. Arsitektur ini juga mendemonstrasikan hot standby, di mana jika instans Amazon RDS utama gagal (atau AZ itu sendiri gagal), maka instans standby dipromosikan ke utama.

Gambar 24: Arsitektur Multi-AZ

Selain arsitektur HA ini, Anda perlu menambahkan cadangan data yang dibutuhkan untuk menjalankan beban kerja. Hal ini sangat penting untuk data yang dibatasi ke zona tunggal seperti volume Amazon EBS atau klaster Amazon Redshift. Jika sebuah AZ gagal, Anda perlu memulihkan data ini ke AZ lainnya. Jika memungkinkan, Anda perlu menyalin cadangan data ke Wilayah AWS sebagai lapisan perlindungan tambahan.

Pendekatan alternatif yang kurang umum untuk DR multi-AZ Wilayah tunggal diilustrasikan di posting blog ini, Membangun aplikasi yang sangat tangguh menggunakan Pengontrol Pemulihan Aplikasi Amazon Route 53, Bagian 1: Tumpukan Wilayah Tunggal. Strategi yang digunakan di sini adalah mempertahankan isolasi sebanyak mungkin di antara AZ, seperti bagaimana Wilayah dioperasikan. Dengan menggunakan strategi alternatif ini, Anda dapat memilih pendekatan aktif/aktif atau aktif/pasif.

> **Note**
>
> Beberapa beban kerja memiliki persyaratan residensi data peraturan. Jika ini diterapkan untuk beban kerja di lokalitas yang saat ini hanya memiliki satu Wilayah AWS, maka

multi-Wilayah tidak akan sesuai untuk kebutuhan bisnis. Strategi multi-AZ memberikan perlindungan yang baik terhadap sebagian besar bencana.

3. Evaluasikan sumber daya beban kerja, dan seperti apa konfigurasinya di Wilayah pemulihan sebelum failover (selama operasi normal).

Untuk infrastruktur dan sumber daya AWS, gunakan infrastruktur sebagai kode seperti AWS CloudFormation atau alat pihak ketiga seperti Hashicorp Terraform. Untuk melakukan deployment di beberapa akun dan Wilayah dengan operasi tunggal, Anda dapat menggunakan AWS CloudFormation StackSets. Untuk strategi Multi-situs aktif/aktif dan Hot Standby, infrastruktur yang di-deploy di Wilayah pemulihan memiliki sumber daya yang sama seperti Wilayah utama. Untuk strategi Pilot Light dan Warm Standby, infrastruktur yang di-deploy memerlukan tindakan tambahan agar berubah menjadi siap produksi. Dengan menggunakan parameter dan logika bersyarat CloudFormation, Anda dapat mengontrol tumpukan yang di-deploy agar aktif atau standby dengan templat tunggal. Ketika menggunakan Elastic Disaster Recovery, layanan akan mereplikasi dan mengatur pemulihan konfigurasi aplikasi dan sumber daya komputasi.

Semua strategi DR memerlukan sumber data yang dicadangkan dalam Wilayah AWS, dan cadangan tersebut disalin ke Wilayah pemulihan. AWS Backup memberikan tampilan terpusat tempat Anda dapat mengonfigurasi, menjadwalkan, dan memantau cadangan untuk sumber daya ini. Untuk Pilot Light, Warm Standby, dan Multi-situs aktif/aktif, Anda juga harus mereplikasi data dari Wilayah utama ke sumber daya data di Wilayah pemulihan, seperti instans DB Amazon Relational Database Service (Amazon RDS) atau tabel Amazon DynamoDB. Dengan demikian, sumber data ini aktif dan siap menangani permintaan di Wilayah pemulihan.

Untuk mempelajari lebih lanjut tentang cara layanan AWS beroperasi di seluruh Wilayah, lihat seri blog ini di Membuat Aplikasi Multi-Wilayah dengan Layanan AWS.

4. Tentukan dan implementasikan cara Anda mempersiapkan Wilayah untuk failover saat dibutuhkan (selama peristiwa bencana).

Untuk multi-situs aktif/aktif, failover berarti mengevakuasi Wilayah dan mengandalkan Wilayah aktif yang tersisa. Secara umum, Wilayah tersebut siap menerima lalu lintas. Untuk strategi Pilot Light dan Warm Standby, tindakan pemulihan perlu mencakup deployment sumber daya yang hilang, seperti instans EC2 dalam Gambar 20, juga sumber daya yang hilang lainnya.

Untuk semua strategi di atas, Anda mungkin perlu mengubah instans hanya-baca basis data menjadi instans baca/tulis.

Untuk pencadangan dan pemulihan, pemulihan data dari cadangan menghasilkan sumber daya untuk data tersebut seperti volume EBS, instans RDS DB, dan tabel DynamoDB. Anda juga perlu memulihkan infrastruktur dan melakukan deployment kode. Anda dapat menggunakan AWS Backup untuk memulihkan data di Wilayah pemulihan. Lihat REL09-BP01 Mengidentifikasi dan mencadangkan data yang perlu dicadangkan, atau memproduksi ulang data dari sumber untuk detail lebih lanjut. Saat membangun kembali infrastruktur, Anda juga membuat sumber daya seperti instans EC2 sebagai tambahan untuk Amazon Virtual Private Cloud (Amazon VPC), subnet, dan grup keamanan yang diperlukan. Anda dapat mengotomatiskan banyak proses pemulihan. Untuk mempelajari caranya, lihat posting blog ini.

5. Tentukan dan implementasikan cara Anda merutekan kembali lalu lintas ke failover saat dibutuhkan (selama peristiwa bencana).

Operasi failover ini dapat dimulai secara otomatis dan manual. Failover yang dimulai secara otomatis berdasarkan pemeriksaan kondisi atau alarm harus digunakan dengan hati-hati karena failover yang tidak perlu (alarm palsu) dapat dikenakan biaya seperti ketidaktersediaan dan kehilangan data. Oleh karena itu, Failover yang dimulai secara manual sering digunakan. Dalam kasus ini, Anda masih harus mengotomatiskan langkah failover, sehingga inisiasi manual akan seperti menekan tombol.

Ada beberapa opsi manajemen lalu lintas yang perlu dipertimbangkan saat menggunakan layanan AWS. Salah satu opsinya adalah menggunakan Amazon Route 53. Dengan menggunakan Amazon Route 53, Anda dapat mengaitkan beberapa titik akhir IP di satu Wilayah AWS atau lebih dengan nama domain Route 53. Untuk mengimplementasikan failover yang dimulai secara manual, Anda dapat menggunakan Pengontrol Pemulihan Aplikasi Amazon Route 53, yang memberikan API bidang data dengan ketersediaan tinggi untuk merutekan kembali lalu lintas ke Wilayah pemulihan. Saat mengimplementasikan failover, gunakan operasi bidang data dan hindari bidang kendali yang dideskripsikan di REL11-BP04 Mengandalkan bidang data dan bukan bidang kendali selama pemulihan.

Untuk mempelajari selengkapnya tentang hal ini dan opsi lainnya, lihat bagian ini di Laporan Resmi Pemulihan Bencana.

6. Rancang rencana terkait bagaimana beban kerja akan failback.

Failback adalah saat Anda mengembalikan operasi beban kerja ke Wilayah utama, setelah bencana berakhir. Penyediaan infrastruktur dan kode untuk Wilayah utama umumnya mengikuti langkah yang sama yang digunakan saat memulai, dengan mengandalkan infrastruktur sebagai kode dan pipeline deployment kode. Tantangan failback adalah mengembalikan penyimpanan data, dan memastikan konsistensi dengan Wilayah pemulihan dalam operasi.

Dalam status failed over, basis data dalam Wilayah pemulihan bersifat waktu nyata dan memiliki data terbaru. Tujuannya adalah untuk menyinkronkan kembali dari Wilayah pemulihan ke Wilayah utama, memastikannya tetap terbaru.

Hal ini dilakukan secara otomatis untuk beberapa layanan AWS. Jika menggunakan tabel global Amazon DynamoDB, meskipun tabel di Wilayah utama menjadi tidak tersedia, saat kembali online, DynamoDB akan melanjutkan penulisan yang tertunda. Jika menggunakan Basis Data Global Amazon Aurora dan menggunakan failover terencana dan terkelola, maka topologi replikasi yang ada untuk basis data global Aurora dipertahankan. Dengan demikian, instans baca/tulis sebelumnya di Wilayah utama akan menjadi replika dan menerima pembaruan dari Wilayah pemulihan.

Dalam kasus saat ini tidak dibuat otomatis, Anda perlu menetapkan ulang basis data di Wilayah utama sebagai replika dari basis data di Wilayah pemulihan. Dalam banyak kasus, ini akan melibatkan penghapusan basis data utama yang lama dan membuat replika yang baru. Misalnya, untuk instruksi tentang cara melakukan ini dengan Basis Data Global Amazon Aurora yang mengasumsikan failover tak terencana, lihat lab ini: Fail Back Basis Data Global.

Setelah failover, jika Anda dapat tetap menjalankannya di Wilayah pemulihan, pertimbangkan untuk membuat ini menjadi Wilayah utama yang baru. Anda masih harus melakukan semua langkah di atas untuk membuat Wilayah utama sebelumnya menjadi Wilayah pemulihan. Beberapa organisasi melakukan rotasi terjadwal, menukar Wilayah utama dan pemulihan secara berkala (misalnya setiap tiga bulan).

Semua langkah yang diperlukan untuk failover dan failback harus diperiksa di buku pedoman yang tersedia untuk semua anggota tim dan ditinjau secara berkala.

Ketika menggunakan Elastic Disaster Recovery, layanan akan membantu mengatur dan mengotomatiskan proses failback. Untuk detail selengkapnya, lihat Melakukan failback.

Tingkat upaya untuk rencana implementasi: Tinggi

Sumber daya

Praktik Terbaik Terkait:

- the section called “REL09-BP01 Mengidentifikasi dan mencadangkan data yang perlu dicadangkan, atau memproduksi ulang data dari sumber”
- the section called “REL11-BP04 Mengandalkan bidang data dan bukan bidang kendali selama pemulihan”
- the section called “REL13-BP01 Tetapkan sasaran pemulihan untuk waktu henti dan kehilangan data”

Dokumen terkait:

- Blog Arsitektur AWS: Seri Pemulihan Bencana
- Pemulihan Bencana Beban Kerja di AWS: Pemulihan di Cloud (Laporan Resmi AWS)
- Opsi pemulihan bencana di cloud
- Bangun solusi backend aktif-aktif nirserver multi-wilayah dalam satu jam
- Backend nirserver multi-wilayah — dimuat ulang
- RDS: Mereplikasi Replika Baca di Seluruh Wilayah
- Route 53: Mengonfigurasi Failover DNS
- S3: Replika Lintas-Wilayah
- Apa Itu AWS Backup?
- Apa itu Pengontrol Pemulihan Aplikasi Route 53?
- AWS Elastic Disaster Recovery
- HashiCorp Terraform: Memulai - AWS
- Partner APN: partner yang dapat membantu pemulihan bencana
- AWS Marketplace: produk yang dapat digunakan untuk pemulihan bencana

Video terkait:

- Pemulihan Bencana Beban Kerja di AWS
- AWS re:Invent 2018: Pola Arsitektur untuk Aplikasi Aktif-Aktif Multi-Wilayah (ARC209-R2)
- Memulai AWS Elastic Disaster Recovery | Amazon Web Services

Contoh terkait:

- Well-Architected Lab - Pemulihan Bencana - Seri lokakarya yang mengilustrasikan strategi DR

REL13-BP03 Menguji implementasi pemulihan bencana untuk memvalidasi implementasi

Secara rutin uji failover ke situs pemulihan Anda untuk memastikan operasi yang baik dan RTO serta RPO terpenuhi.

Antipola umum:

- Tidak pernah melakukan failover di lingkungan produksi.

Manfaat menjalankan praktik terbaik ini: Pengujian rencana pemulihan bencana secara rutin memverifikasi bahwa rencana tersebut akan berfungsi saat diperlukan, dan tim Anda tahu cara menjalankan strategi.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Tinggi

Panduan implementasi

Pola untuk dihindari adalah mengembangkan jalur pemulihan yang sangat jarang dilakukan. Misalnya, Anda mungkin memiliki penyimpanan data sekunder yang digunakan untuk kueri hanya-baca. Saat Anda menulis ke penyimpanan data dan penyimpanan primer gagal, Anda mungkin ingin melakukan failover ke penyimpanan data sekunder. Jika Anda tidak sering menguji failover ini, Anda mungkin akan mendapati bahwa asumsi Anda tentang kemampuan penyimpanan data sekunder ternyata salah. Kapasitas sekunder, yang selama ini mungkin mencukupi saat terakhir Anda uji, mungkin sudah tidak mampu mentoleransi beban di bawah skenario ini. Pengalaman kami menunjukkan bahwa satu-satunya pemulihan kesalahan yang berfungsi adalah jalur yang Anda uji secara sering. Inilah alasan memiliki sedikit jalur pemulihan adalah yang terbaik. Anda dapat membuat pola pemulihan dan mengujinya secara rutin. Jika Anda memiliki jalur pemulihan yang kompleks atau kritis, Anda tetap perlu secara rutin melatih kegagalan tersebut dalam lingkungan produksi agar Anda yakin bahwa jalur pemulihan tersebut berfungsi. Pada contoh yang baru saja kita bahas, Anda harus melakukan failover ke penyimpanan siaga secara rutin, terlepas ada tidaknya kebutuhan.

Langkah implementasi

1. Rekayasa beban kerja Anda untuk pemulihan. Uji jalur pemulihan Anda secara rutin. Komputasi yang berorientasi pada pemulihan mengidentifikasi karakteristik dalam sistem yang meningkatkan pemulihan: isolasi dan redundansi, kemampuan di seluruh sistem untuk membatalkan perubahan,

kemampuan untuk memantau dan menentukan kondisi, kemampuan untuk menyediakan diagnostik, pemulihan otomatis, desain modular, dan kemampuan untuk memulai ulang. Latih jalur pemulihan untuk memverifikasi bahwa Anda dapat menyelesaikan pemulihan dalam waktu yang ditentukan ke status yang ditentukan. Gunakan runbook selama pemulihan ini untuk mendokumentasikan masalah dan menemukan solusinya sebelum pengujian berikutnya.

2. Untuk beban kerja berbasis Amazon EC2, gunakan AWS Elastic Disaster Recovery untuk mengimplementasikan dan meluncurkan instans latihan untuk strategi DR Anda. AWS Elastic Disaster Recovery menyediakan kemampuan untuk menjalankan latihan secara efisien, yang membantu Anda bersiap untuk peristiwa failover. Anda juga dapat sering-sering meluncurkan instans menggunakan Elastic Disaster Recovery untuk tujuan pengujian dan latihan tanpa mengarahkan ulang lalu lintas.

Sumber daya

Dokumen terkait:

- Partner APN: partner yang dapat membantu pemulihan bencana
- Blog Arsitektur AWS: Seri Pemulihan Bencana
- AWS Marketplace: produk yang dapat digunakan untuk pemulihan bencana
- AWS Elastic Disaster Recovery
- Pemulihan Bencana Beban Kerja di AWS: Pemulihan di Cloud (Laporan Resmi AWS)
- Bersiap untuk Failover AWS Elastic Disaster Recovery
- Proyek Berkeley/Stanford komputasi berorientasi pemulihan
- Apa itu Simulator Injeksi Kesalahan AWS?

Video terkait:

- AWS re:Invent 2018: Pola Arsitektur untuk Aplikasi Aktif-Aktif Multi-Wilayah
- AWS re:Invent 2019: Pencadangan dan pemulihan serta solusi pemulihan bencana dengan AWS

Contoh terkait:

- Well-Architected Lab - Pengujian Ketangguhan

REL13-BP04 Mengelola penyimpangan konfigurasi di lokasi atau Wilayah Pemulihan Bencana (DR)

Pastikan infrastruktur, data, dan konfigurasi diperlukan di lokasi atau Wilayah DR. Misalnya, periksa apakah AMI dan kuota layanan sudah mutakhir.

AWS Config terus memantau dan merekam konfigurasi sumber daya AWS Anda. Layanan ini dapat mendeteksi penyimpangan dan memicu AWS Systems Manager Automation untuk memperbaikinya dan memunculkan alarm. AWS CloudFormation juga dapat mendeteksi penyimpangan dalam tumpukan yang telah Anda deploy.

Antipola umum:

- Gagal melakukan pembaruan pada lokasi pemulihan Anda, saat Anda membuat perubahan konfigurasi atau infrastruktur pada lokasi primer.
- Tidak mempertimbangkan potensi pembatasan (seperti perbedaan layanan) di lokasi primer dan pemulihan Anda.

Manfaat menjalankan praktik terbaik ini: Lingkungan DR yang sesuai dengan lingkungan Anda saat ini menjamin pemulihan yang lengkap.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak dijalankan: Sedang

Panduan implementasi

- Pastikan pipeline pengiriman Anda menjangkau lokasi primer dan cadangan Anda. Pipeline pengiriman untuk men-deploy aplikasi ke lingkungan produksi harus menyebarkan ke semua lokasi strategi pemulihan bencana yang ditentukan, termasuk lingkungan pengembangan dan pengujian.
- Aktifkan AWS Config untuk melacak lokasi dengan potensi penyimpangan. Gunakan aturan AWS Config untuk membuat sistem yang menerapkan strategi pemulihan bencana Anda dan menghasilkan pemberitahuan saat mendeteksi penyimpangan.
  - Mengatasi Sumber Daya AWS yang Tidak Patuh dengan Aturan AWS Config
  - AWS Systems Manager Automation
- Gunakan AWS CloudFormation untuk men-deploy infrastruktur Anda. AWS CloudFormation dapat mendeteksi penyimpangan antara yang ditentukan oleh templat CloudFormation Anda dan apa yang sebenarnya di-deploy.
  - AWS CloudFormation: Mendeteksi Penyimpangan di Seluruh Tumpukan CloudFormation

Sumber daya

Dokumen terkait:

- Partner APN: partner yang dapat membantu pemulihan bencana
- Blog Arsitektur AWS: Seri Pemulihan Bencana
- AWS CloudFormation: Mendeteksi Penyimpangan di Seluruh Tumpukan CloudFormation
- AWS Marketplace: produk yang dapat digunakan untuk pemulihan bencana
- AWS Systems Manager Automation
- Pemulihan Bencana Beban Kerja di AWS: Pemulihan di Cloud (Laporan Resmi AWS)
- Bagaimana cara mengimplementasikan solusi Manajemen Konfigurasi Infrastruktur di AWS?
- Mengatasi Sumber Daya AWS yang Tidak Patuh dengan Aturan AWS Config

Video terkait:

- AWS re:Invent 2018: Pola Arsitektur untuk Aplikasi Multi-Wilayah Aktif-Aktif (ARC209-R2)

REL13-BP05 Mengotomatiskan pemulihan

Gunakan AWS atau alat pihak ketiga untuk mengotomatiskan pemulihan sistem dan merutekan lalu lintas ke situs DR atau Wilayah.

Berdasarkan pemeriksaan kondisi yang dikonfigurasi, layanan AWS, seperti Elastic Load Balancing dan AWS Auto Scaling, dapat mendistribusikan beban ke Zona Ketersediaan yang kondisinya baik, sedangkan layanan seperti Amazon Route 53 dan AWS Global Accelerator, dapat merutekan beban ke Wilayah AWS yang kondisinya baik. Pengontrol Pemulihan Aplikasi Amazon Route 53 membantu Anda mengelola dan mengoordinasikan failover menggunakan fitur pemeriksaan kesiapan dan kontrol perutean. Fitur tersebut terus memantau kemampuan aplikasi untuk pulih dari kegagalan, sehingga Anda dapat mengontrol pemulihan aplikasi di beberapa Wilayah AWS, Zona Ketersediaan, dan on-premise.

Untuk beban kerja yang ada di pusat data fisik atau virtual atau cloud pribadi, AWS Elastic Disaster Recovery, tersedia melalui AWS Marketplace, memungkinkan organisasi untuk mengatur strategi pemulihan bencana otomatis ke AWS. CloudEndure juga mendukung pemulihan bencana lintas Wilayah/lintas AZ di AWS.

Antipola umum:

- Mengimplementasikan failover dan failback otomatis yang serupa dapat menyebabkan flapping saat kesalahan terjadi.

Manfaat menerapkan praktik terbaik ini: Pemulihan otomatis mengurangi waktu pemulihan dengan menghilangkan peluang untuk kesalahan manual.

Tingkat risiko yang terjadi jika praktik terbaik ini tidak diterapkan: Sedang

Panduan implementasi

- Otomatiskan jalur pemulihan. Untuk pemulihan pendek, tindakan dan penilaian manusia tidak dapat digunakan untuk skenario ketersediaan tinggi. Sistem harus pulih secara otomatis dalam setiap situasi.
  - Gunakan CloudEndure Disaster Recovery untuk Failback dan Failover otomatis. CloudEndure Disaster Recovery terus mereplikasi mesin (termasuk sistem operasi, konfigurasi status sistem, basis data, aplikasi, dan file) ke dalam area penahapan rendah biaya di Akun AWS target dan Wilayah utama. Dalam kasus bencana, Anda dapat menginstruksikan CloudEndure Disaster Recovery untuk meluncurkan mesin dalam status yang tersedia sepenuhnya dalam hitungan menit secara otomatis.
    - Menjalankan Failover dan Failback Pemulihan Bencana
    - CloudEndure Disaster Recovery

Sumber daya

Dokumen terkait:

- Partner APN: partner yang dapat membantu pemulihan bencana
- Blog Arsitektur AWS: Seri Pemulihan Bencana
- AWS Marketplace: produk yang dapat digunakan untuk pemulihan bencana
- AWS Systems Manager Automation
- CloudEndure Disaster Recovery ke AWS
- Pemulihan Bencana Beban Kerja di AWS: Pemulihan di Cloud (Laporan Resmi AWS)

Video terkait:

- AWS re:Invent 2018: Pola Arsitektur untuk Aplikasi Aktif-Aktif Multi-Wilayah (ARC209-R2)

Contoh implementasi untuk tujuan ketersediaan

Di bagian ini, kita akan meninjau desain beban kerja menggunakan deployment aplikasi web umum yang terdiri dari proksi mundur, konten statis di Amazon S3, server aplikasi, dan basis data SQL untuk penyimpanan data yang berkelanjutan. Untuk setiap target ketersediaan, kami memberikan contoh implementasi. Sebagai gantinya, beban kerja ini dapat menggunakan kontainer atau AWS Lambda untuk komputasi dan NoSQL (seperti Amazon DynamoDB) untuk basis data, tetapi pendekatannya serupa. Dalam setiap skenario, kami menunjukkan cara memenuhi tujuan ketersediaan melalui desain beban kerja untuk topik-topik ini:

Topik	Untuk informasi selengkapnya, lihat bagian ini
Pantau sumber daya	Memantau sumber daya beban kerja
Beradaptasi dengan perubahan dalam permintaan	Rancang beban kerja Anda agar dapat beradaptasi dengan perubahan dalam permintaan
Implementasikan perubahan	Implementasikan perubahan
Cadangkan data	Cadangkan data
Rancang untuk ketangguhan	Gunakan isolasi kesalahan untuk melindungi beban kerja Anda Rancang beban kerja Anda agar bertahan dalam kegagalan komponen
Uji ketangguhan	Uji keandalan
Rencanakan pemulihan bencana (DR)	Rencanakan Pemulihan Bencana (DR)

Pemilihan ketergantungan

Kami telah memilih untuk menggunakan Amazon EC2 untuk aplikasi kami. Kami akan memperlihatkan bagaimana menggunakan Amazon RDS dan beberapa Zona Ketersediaan akan meningkatkan ketersediaan aplikasi kami. Kami akan menggunakan Amazon Route 53 untuk DNS.

Ketika kami menggunakan beberapa Zona Ketersediaan, kami akan menggunakan Elastic Load Balancing. Amazon S3 digunakan untuk cadangan dan konten statis. Saat kami mendesain untuk keandalan lebih tinggi, kami harus menggunakan layanan dengan ketersediaan lebih tinggi itu sendiri.

Skenario Wilayah Tunggal

Topik

Skenario 2 angka 9 (99%)

Beban kerja ini bermanfaat untuk bisnis, tetapi hanya akan merepotkan jika tidak tersedia. Jenis beban kerja ini bisa berupa alat internal, manajemen pengetahuan internal, atau pelacakan proyek. Atau bisa berupa beban kerja aktual yang dilihat pelanggan tetapi dilayani dari layanan eksperimental, dengan tombol fitur yang dapat menyembunyikan layanan jika perlu.

Beban kerja ini dapat di-deploy dengan satu Wilayah dan satu Zona Ketersediaan.

Pantau sumber daya

Kami akan memiliki pemantauan sederhana, yang menunjukkan apakah halaman utama layanan mengembalikan status OK HTTP 200. Ketika masalah timbul, playbook kami akan menunjukkan bahwa logging dari instans akan digunakan untuk menetapkan akar masalahnya.

Beradaptasi dengan perubahan dalam permintaan

Kami akan memiliki playbook untuk kegagalan perangkat keras yang umum, pembaruan perangkat lunak mendesak, dan perubahan lain yang mengganggu.

Implementasikan perubahan

Kami akan menggunakan AWS CloudFormation untuk menetapkan infrastruktur kami sebagai kode, dan secara khusus untuk mempercepat rekonstruksi jika terjadi kegagalan.

Pembaruan perangkat lunak dilakukan secara manual menggunakan runbook, dengan waktu henti yang diperlukan untuk penginstalan dan pemulaian ulang layanan. Jika masalah timbul selama deployment, runbook menjelaskan cara untuk kembali ke versi sebelumnya.

Setiap koreksi kesalahan dilakukan menggunakan analisis log oleh tim operasi dan pengembangan, dan koreksinya di-deploy setelah perbaikan diprioritaskan dan diselesaikan.

Cadangkan data

Kami akan menggunakan vendor atau solusi cadangan yang dibuat khusus untuk mengirimkan data cadangan terenkripsi ke Amazon S3 menggunakan runbook. Kami akan menguji apakah cadangan berfungsi dengan memulihkan data dan memastikan kemampuan untuk menggunakannya secara teratur dengan runbook. Kami mengonfigurasikan versioning di objek Amazon S3 kami dan menyingkirkan izin untuk penghapusan cadangan. Kami menggunakan kebijakan siklus hidup bucket Amazon S3 untuk mengarsipkan atau secara permanen menghapus sesuai dengan persyaratan kami.

Rancang untuk ketangguhan

Beban kerja di-deploy dengan satu Wilayah dan satu Zona Ketersediaan. Kami melakukan deploy aplikasi, termasuk basis data, ke instans tunggal.

Uji ketangguhan

Alur deployment perangkat lunak baru dijadwalkan, dengan beberapa pengujian unit, tetapi sebagian besar pengujian kotak putih/kotak hitam dari beban kerja yang tersusun.

Rencanakan pemulihan bencana (DR)

Selama kegagalan terjadi, kami menunggu sampai kegagalan selesai, dengan pilihan merutekan permintaan ke situs web statis menggunakan modifikasi DNS lewat runbook. Waktu pemulihan untuk ini akan ditentukan oleh kecepatan infrastruktur dapat di-deploy dan basis data dipulihkan ke cadangan terbaru. Deployment ini bisa ke Zona Ketersediaan yang sama, atau ke Zona Ketersediaan yang berbeda, jika terjadi kegagalan Zona Ketersediaan, menggunakan runbook.

Tujuan desain ketersediaan

Kami meluangkan waktu 30 menit memahami dan memutuskan untuk melaksanakan pemulihan, melakukan deploy seluruh tumpukan di AWS CloudFormation dalam 10 menit, berasumsi bahwa kami melakukan deploy ke Zona Ketersediaan baru, dan berasumsi bahwa basis data dapat

dipulihkan dalam waktu 30 menit. Ini menyiratkan bahwa diperlukan sekitar 70 menit untuk pulih dari kegagalan. Dengan asumsi satu kegagalan per kuartal, waktu dampak yang kami perkirakan untuk setahun yakni 280 menit, atau empat jam 40 menit.

Ini berarti batas atas ketersediaan yakni 99,9%. Ketersediaan aktual juga bergantung pada tingkat kegagalan yang sesungguhnya, durasi kegagalan, dan seberapa cepat setiap kegagalan pulih secara aktual. Untuk arsitektur ini, kami memerlukan aplikasi dalam keadaan offline untuk pembaruan (dengan perkiraan 24 jam per tahun; empat jam per perubahan, enam kali per tahun), ditambah peristiwa aktual. Jadi, dengan merujuk ke tabel ketersediaan aplikasi sebelumnya dalam laporan resmi, maka terlihat bahwa tujuan desain ketersediaan 99%.

Ringkasan

Topik	Implementasi
Pantau sumber daya	Pemeriksaan kondisi situs saja; tanpa peringatan.
Beradaptasi dengan perubahan dalam permintaan	Penskalaan vertikal lewat deployment ulang.
Implementasikan perubahan	Runbook untuk deployment dan pengembalian ke sebelumnya.
Cadangkan data	Runbook untuk cadangan dan pemulihan.
Rancang untuk ketangguhan	Pembangunan ulang lengkap; pemulihan dari cadangan.
Uji ketangguhan	Pembangunan ulang lengkap; pemulihan dari cadangan.
Rencanakan pemulihan bencana (DR)	Cadangan terenkripsi, pemulihan ke Zona Ketersediaan yang berbeda jika perlu.

Skenario 3 angka 9 (99,9%)

Tujuan ketersediaan berikutnya adalah untuk aplikasi yang penting memiliki ketersediaan tinggi, tetapi aplikasi tersebut dapat menoleransi ketidaktersediaan dalam jangka waktu. Jenis beban kerja

ini biasanya digunakan untuk operasi internal yang memiliki efek atas karyawan ketika operasi tidak berfungsi. Jenis beban kerja ini juga dapat dilihat oleh pelanggan, tetapi pendapatannya tidak cukup tinggi bagi bisnis, dan dapat menoleransi waktu pemulihan atau titik pemulihan lebih lama. Beban kerja tersebut mencakup aplikasi administrasi untuk akun atau manajemen informasi.

Kami dapat meningkatkan ketersediaan untuk beban kerja dengan menggunakan dua Zona Ketersediaan untuk deployment kami dan dengan memisahkan aplikasi ke tingkat terpisah.

Pantau sumber daya

Pemantauan akan diperluas untuk memberikan peringatan tentang ketersediaan situs web secara keseluruhan dengan memeriksa status OK HTTP 200 di halaman utama. Selain itu, akan ada peringatan tentang setiap penggantian server web dan ketika terjadi failover basis data. Kami juga akan memantau konten statis di Amazon S3 untuk mengetahui ketersediaan dan memberikan peringatan jika tidak tersedia. Logging akan diagregatkan untuk memudahkan pengelolaan dan untuk membantu analisis akar masalah.

Beradaptasi dengan perubahan dalam permintaan

Penskalaan otomatis dikonfigurasikan untuk memantau penggunaan CPU di instans EC2, dan menambahkan atau menyingkirkan instans guna mempertahankan CPU target di 70%, tetapi dengan tidak kurang dari satu instans EC2 per Zona Ketersediaan. Jika pola beban di instans RDS kami menunjukkan diperlukannya kenaikan skala, kami akan mengubah jenis instans selama jangka waktu pemeliharaan.

Implementasikan perubahan

Teknologi deployment infrastruktur tetap sama seperti skenario sebelumnya.

Penyampaian perangkat lunak baru berdasarkan jadwal tetap, setiap dua hingga empat minggu. Pembaruan perangkat lunak akan otomatis, tidak menggunakan pola deployment blue/green atau canary, tetapi menggunakan penggantian. Keputusan untuk kembali ke sebelumnya akan dibuat menggunakan runbook.

Kami akan memiliki playbook untuk menetapkan akar masalah. Setelah akar masalah diidentifikasi, koreksi kesalahan akan diidentifikasi melalui gabungan antara tim operasi dan pengembangan. Koreksi akan dilakukan deploy setelah perbaikan dikembangkan.

Cadangkan data

Pencadangan dan pemulihan dapat dilakukan menggunakan Amazon RDS. Ini akan dilaksanakan secara teratur menggunakan runbook untuk memastikan bahwa kami dapat memenuhi persyaratan pemulihan.

Rancang untuk ketangguhan

Kami dapat meningkatkan ketersediaan untuk aplikasi dengan menggunakan dua Zona Ketersediaan untuk deployment kami dan dengan memisahkan aplikasi ke tingkat terpisah. Kami akan menggunakan layanan yang berfungsi di beberapa Zona Ketersediaan, seperti Elastic Load Balancing, Auto Scaling, dan Multi-AZ Amazon RDS dengan penyimpanan terenkripsi lewat AWS Key Management Service. Ini akan memastikan toleransi terhadap kegagalan di tingkat sumber daya dan di tingkat Zona Ketersediaan.

Penyeimbang beban hanya akan merutekan lalu lintas ke instans aplikasi berkondisi bagus. Pemeriksaan kondisi harus dilakukan di lapisan bidang data/aplikasi yang menunjukkan kemampuan aplikasi pada instans. Pemeriksaan ini tidak boleh dilakukan terhadap bidang kendali. Akan ada URL pemeriksaan kondisi untuk aplikasi web yang dikonfigurasikan untuk digunakan oleh penyeimbang beban dan Auto Scaling, sehingga instans yang gagal akan disingkirkan dan diganti. Amazon RDS akan mengelola mesin basis data aktif agar tersedia di Zona Ketersediaan kedua jika instans gagal di Zona Ketersediaan utama, lalu memperbaiki untuk memulihkan ke ketangguhan yang sama.

Setelah kami memisahkan tingkat, kami dapat menggunakan pola ketangguhan sistem terdistribusi untuk meningkatkan keandalan aplikasi sehingga aplikasi masih tersedia meskipun basis data tidak tersedia untuk sementara selama failover Zona Ketersediaan.

Uji ketangguhan

Kami melakukan uji fungsional, sama seperti dalam skenario sebelumnya. Kami tidak menguji kemampuan pemulihan mandiri ELB, penskalaan otomatis, atau failover RDS.

Kami akan memiliki playbook untuk masalah basis data umum, insiden terkait keamanan, dan deployment yang gagal.

Rencanakan pemulihan bencana (DR)

Ada runbook untuk pemulihan beban kerja total dan pelaporan umum. Pemulihan menggunakan cadangan yang disimpan di wilayah yang sama seperti beban kerja.

Tujuan desain ketersediaan

Kami berasumsi bahwa minimal beberapa kegagalan akan memerlukan keputusan manual untuk melaksanakan pemulihan. Tetapi, dengan otomatisasi lebih besar dalam skenario ini, kami berasumsi bahwa hanya dua peristiwa per tahun yang akan memerlukan keputusan ini. Kami meluangkan waktu 30 menit memutuskan untuk melaksanakan pemulihan, dan berasumsi bahwa pemulihan diselesaikan dalam 30 menit. Ini menyiratkan 60 menit untuk pulih dari kegagalan. Dengan asumsi dua insiden per tahun, waktu dampak yang kami perkirakan untuk setahun yakni 120 menit.

Ini berarti batas atas ketersediaan yakni 99,95%. Ketersediaan aktual juga bergantung pada tingkat kegagalan yang sesungguhnya, durasi kegagalan, dan seberapa cepat setiap kegagalan pulih secara aktual. Untuk arsitektur ini, kami memerlukan aplikasi dalam keadaan offline sebentar untuk pembaruan, tetapi pembaruan ini bersifat otomatis. Kami memperkirakan 150 menit per tahun untuk ini: 15 menit per perubahan, 10 kali per tahun. Ini totalnya 270 menit per tahun ketika layanan tidak tersedia, jadi tujuan desain ketersediaan 99,9%.

Ringkasan

Topik	Implementasi
Pantau sumber daya	Pemeriksaan kondisi situs saja; peringatan dikirimkan ketika tidak berfungsi.
Beradaptasi dengan perubahan dalam permintaan	ELB untuk tingkat aplikasi penskalaan otomatis dan web; pengubahan ukuran RDS Multi-AZ.
Implementasikan perubahan	Terdapat deployment otomatis dan runbook untuk pengembalian ke sebelumnya.
Cadangkan data	Cadangan otomatis lewat RDS untuk memenuhi RPO dan runbook untuk pemulihan.
Rancang untuk ketangguhan	Penskalaan otomatis untuk memberikan tingkat aplikasi dan web dengan pemulihan mandiri; RDS bersifat Multi-AZ.
Uji ketangguhan	ELB dan aplikasi dapat pulih secara mandiri; RDS bersifat Multi-AZ; tanpa pengujian eksplisit .

Topik	Implementasi
Rencanakan pemulihan bencana (DR)	Cadangan terenkripsi lewat RDS ke Wilayah AWS yang sama.

Skenario 4 angka 9 (99,99%)

Tujuan ketersediaan untuk aplikasi ini memerlukan aplikasi yang memiliki ketersediaan tinggi dan toleran terhadap kegagalan komponen. Aplikasi harus dapat menyerap kegagalan tanpa memerlukan sumber daya tambahan. Tujuan ketersediaan ini adalah untuk aplikasi yang sangat penting yang merupakan pendorong pendapatan utama atau signifikan bagi korporasi, seperti situs perdagangan elektronik, layanan web bisnis ke bisnis, atau situs konten/media dengan lalu lintas tinggi.

Kami dapat meningkatkan ketersediaan lebih lanjut dengan menggunakan arsitektur yang akan stabil secara statistik dalam Wilayah. Tujuan ketersediaan ini tidak memerlukan perubahan bidang kendali dalam perilaku beban kerja kami untuk menoleransi kegagalan. Contohnya, seharusnya ada kapasitas yang cukup untuk bertahan dari kehilangan salah satu Zona Ketersediaan. Seharusnya kami tidak memerlukan pembaruan ke DNS Amazon Route 53. Seharusnya kami tidak perlu menciptakan infrastruktur baru apa pun, baik itu menciptakan atau memodifikasi bucket S3, membuat kebijakan IAM baru (atau modifikasi kebijakan), atau memodifikasi konfigurasi tugas Amazon ECS.

Pantau sumber daya

Pemantauan akan mencakup metrik kesuksesan serta memberikan peringatan ketika masalah timbul. Selain itu, akan ada peringatan tentang setiap penggantian server web yang gagal, ketika terjadi failover basis data, dan ketika AZ gagal.

Beradaptasi dengan perubahan dalam permintaan

Kami akan menggunakan Amazon Aurora sebagai RDS kami, yang memampukan penskalaan otomatis replika baca. Untuk aplikasi-aplikasi ini, rekayasa untuk ketersediaan baca dibandingkan ketersediaan tulis konten utama juga merupakan keputusan arsitektur utama. Aurora juga dapat mengembangkan penyimpanan sesuai keperluan secara otomatis, dalam kenaikan 10 GB hingga 64 TB.

Implementasikan perubahan

Kami akan melakukan deploy pembaruan menggunakan canary atau deployment blue/green ke setiap zona isolasi secara terpisah. Deployment bersifat sepenuhnya otomatis, termasuk pengembalian ke yang sebelumnya jika KPI menunjukkan ada masalah.

Akan ada runbook untuk pelacakan performa dan persyaratan pelaporan yang ketat. Jika operasi yang sukses memiliki kecenderungan untuk gagal agar memenuhi tujuan ketersediaan atau performa, playbook akan digunakan untuk menetapkan apa yang menyebabkan kecenderungan tersebut. Akan ada playbook untuk insiden keamanan dan mode kegagalan yang tidak ditemukan. Juga akan ada playbook untuk menetapkan akar masalah dari kegagalan. Kami juga akan melibatkan AWS Support untuk penawaran Manajemen Peristiwa Infrastruktur.

Tim yang membangun dan mengoperasikan situs web akan mengidentifikasi koreksi kesalahan setiap kegagalan yang tidak terduga dan memprioritaskan deployment perbaikan setelah perbaikan diimplementasikan.

Cadangkan data

Pencadangan dan pemulihan dapat dilakukan menggunakan Amazon RDS. Ini akan dilaksanakan secara teratur menggunakan runbook untuk memastikan bahwa kami dapat memenuhi persyaratan pemulihan.

Rancang untuk ketangguhan

Kami menyarankan tiga Zona Ketersediaan untuk pendekatan ini. Menggunakan deployment tiga Zona Ketersediaan, setiap AZ memiliki kapasitas statis 50% dari kapasitas puncak. Dua Zona Ketersediaan dapat digunakan, tetapi biaya kapasitas yang stabil secara statistik akan lebih besar karena kedua zona harus memiliki 100% kapasitas puncak. Kami akan menambahkan Amazon CloudFront untuk memberikan caching geografis, serta pengurangan permintaan di bidang data aplikasi kami.

Kami akan menggunakan Amazon Aurora sebagai RDS kami dan melakukan deploy replika baca di ketiga zona.

Aplikasi akan dibangun menggunakan pola ketangguhan perangkat lunak/aplikasi di semua lapisan.

Uji ketangguhan

Alur deployment akan memiliki kumpulan pengujian penuh, termasuk uji performa, beban, dan injeksi kegagalan.

Kami akan secara terus-menerus mempraktikkan prosedur pemulihan kegagalan kami melalui aktivitas game, menggunakan runbook untuk memastikan kami dapat menjalankan tugas dan tidak menyimpang dari prosedur. Tim yang membangun situs web juga mengoperasikan situs web.

Rencanakan pemulihan bencana (DR)

Ada runbook untuk pemulihan beban kerja total dan pelaporan umum. Pemulihan menggunakan cadangan yang disimpan di wilayah yang sama seperti beban kerja. Prosedur pemulihan dilatih secara teratur sebagai bagian dari aktivitas game.

Tujuan desain ketersediaan

Kami berasumsi bahwa minimal beberapa kegagalan akan memerlukan keputusan manual untuk melaksanakan pemulihan, tetapi dengan otomatisasi lebih besar dalam skenario ini, kami berasumsi bahwa hanya dua peristiwa per tahun yang akan memerlukan keputusan ini dan tindakan pemulihannya akan cepat. Kami meluangkan waktu 10 menit memutuskan untuk melaksanakan pemulihan, dan berasumsi bahwa pemulihan diselesaikan dalam lima menit. Ini menyiratkan 15 menit untuk pulih dari kegagalan. Dengan asumsi dua kegagalan per tahun, waktu dampak yang kami perkirakan untuk setahun yakni 30 menit.

Ini berarti batas atas ketersediaan yakni 99,99%. Ketersediaan aktual juga akan bergantung pada tingkat kegagalan yang sesungguhnya, durasi kegagalan, dan seberapa cepat setiap kegagalan pulih secara aktual. Untuk arsitektur ini, kami berasumsi bahwa aplikasinya online terus selama pembaruan. Berdasarkan ini, tujuan desain ketersediaan 99,99%.

Ringkasan

Topik	Implementasi
Pantau sumber daya	Pemeriksaan kondisi di semua lapisan dan di KPI; peringatan dikirimkan ketika alarm yang dikonfigurasi terpicu; yang memberikan peringatan tentang semua kegagalan. Rapat operasional yang sangat teliti untuk mendeteksi tren dan mendesain tujuan dengan sukses.
Beradaptasi dengan perubahan dalam permintaan	ELB untuk tingkat aplikasi penskalaan otomatis dan web; penyimpanan penskalaan otomatis

Topik	Implementasi
	dan replika baca di beberapa zona untuk RDS Aurora.
Implementasikan perubahan	Deployment otomatis lewat canary atau pengembalian ke sebelumnya yang otomatis dan blue/green ketika KPI atau peringatan menunjukkan masalah yang tak terdeteksi dalam aplikasi. Deployment dibuat oleh zona isolasi.
Cadangkan data	Cadangan otomatis lewat RDS untuk memenuhi restorasi otomatis dan RPO yang dipraktikkan secara teratur dalam aktivitas game.
Rancang untuk ketangguhan	Zona isolasi kegagalan yang diimpleme ntasikan untuk aplikasi; penskalaan otomatis untuk memberikan tingkat aplikasi dan web dengan pemulihan mandiri; RDS bersifat Multi-AZ.
Uji ketangguhan	Pengujian kesalahan zona isolasi dan komponen sudah dalam alur dan dipraktikkan dengan staf operasional secara teratur dalam aktivitas game; ada playbook untuk mendiagno sis masalah yang tak diketahui; dan ada proses Analisis Akar Masalah.
Rencanakan pemulihan bencana (DR)	Cadangan terenkripsi lewat RDS ke Wilayah AWS yang sama yang dipraktikkan dalam aktivitas game.

Skenario Multi-Wilayah

Mengimplementasikan aplikasi kami dalam beberapa Wilayah AWS akan meningkatkan biaya operasi, sebagian karena kami mengisolasikan wilayah agar wilayah tetap otonom. Ini harus

merupakan keputusan yang sangat bijaksana untuk mengikuti jalur ini. Meskipun demikian, wilayah memberikan batas isolasi yang kuat dan kami berusaha keras untuk menghindari kegagalan yang berkorelasi lintas wilayah. Menggunakan beberapa wilayah akan memberikan kepada Anda kontrol lebih besar atas waktu pemulihan Anda jika terjadi kegagalan ketergantungan yang sulit di layanan AWS regional. Di bagian ini, kami akan membahas berbagai pola implementasi dan ketersediaan umumnya.

Topik

3½ angka 9 (99,95%) dengan Waktu Pemulihan antara 5 dan 30 Menit

Tujuan ketersediaan untuk aplikasi ini memerlukan waktu henti yang sangat singkat dan sangat sedikit kehilangan data selama waktu tertentu. Aplikasi dengan tujuan ketersediaan ini mencakup aplikasi dalam bidang: perbankan, investasi, layanan darurat, dan pengambilan data. Aplikasi-aplikasi ini memiliki titik pemulihan dan waktu pemulihan yang sangat pendek.

Kami dapat meningkatkan waktu pemulihan lebih lanjut dengan menggunakan pendekatan Warm Standby di dua Wilayah AWS. Kami akan melakukan deploy seluruh beban kerja ke kedua Wilayah, dengan menurunkan skala situs pasif kami dan semua data dipertahankan agar pada akhirnya konsisten. Kedua deployment akan stabil secara statistik dalam wilayah masing-masing. Aplikasi harus dibangun menggunakan pola ketangguhan sistem terdistribusi. Kami harus membuat komponen perutean ringan yang memantau kondisi beban kerja, dan yang dapat dikonfigurasikan untuk merutekan lalu lintas ke wilayah pasif jika perlu.

Pantau sumber daya

Akan ada peringatan tentang setiap penggantian server web, ketika terjadi failover basis data dan ketika terjadi failover Wilayah. Kami juga akan memantau konten statis di Amazon S3 untuk mengetahui ketersediaan dan memberikan peringatan jika tidak tersedia. Logging akan diagregatkan untuk memudahkan pengelolaan dan untuk membantu analisis akar masalah di setiap Wilayah.

Komponen perutean memantau kondisi aplikasi dan ketergantungan regional sulit yang kami miliki.

Beradaptasi dengan perubahan dalam permintaan

Sama seperti skenario 4 angka 9.

Implementasikan perubahan

Penyampaian perangkat lunak baru berdasarkan jadwal tetap, setiap dua hingga empat minggu. Pembaruan perangkat lunak akan diotomatiskan menggunakan pola deployment blue/green atau canary.

Ada runbook untuk ketika terjadi failover Wilayah, untuk masalah umum pelanggan yang terjadi selama peristiwa tersebut, dan untuk pelaporan umum.

Kami akan memiliki playbook untuk masalah umum basis data, insiden yang terkait dengan keamanan, deployment yang gagal, masalah pelanggan yang tak terduga di failover Wilayah, dan penetapan akar masalah. Setelah akar masalah diidentifikasi, koreksi kesalahan akan diidentifikasi melalui gabungan antara tim operasi dan pengembangan dan di-deploy ketika perbaikan dikembangkan.

Kami juga akan melibatkan AWS Support untuk Manajemen Peristiwa Infrastruktur.

Cadangkan data

Seperti skenario 4 angka 9, kami menggunakan cadangan RDS otomatis dan versioning S3. Data secara otomatis dan asinkron direplikasi dari klaster RDS Aurora di wilayah aktif ke replika baca lintas wilayah di wilayah pasif. Replikasi lintas wilayah S3 digunakan untuk secara otomatis dan asinkron memindahkan data dari wilayah aktif ke wilayah pasif.

Rancang untuk ketangguhan

Sama seperti skenario 4 angka 9, ditambah failover regional dapat terjadi. Ini akan dikelola secara manual. Selama failover, kami akan merutekan permintaan ke situs web statis menggunakan failover DNS sampai pemulihan di Wilayah kedua.

Uji ketangguhan

Sama seperti skenario 4 angka 9, ditambah kami akan memvalidasi arsitektur melalui aktivitas game menggunakan runbook. Selain itu, koreksi RCA diprioritaskan melebihi rilis fitur untuk deployment dan implementasi dengan segera

Rencanakan pemulihan bencana (DR)

Failover regional dikelola secara manual. Semua data direplikasi secara asinkron. Infrastruktur di warm standby diskalakan ke luar. Ini dapat diotomatiskan menggunakan alur kerja yang dilaksanakan di AWS Step Functions. AWS Systems Manager (SSM) juga dapat membantu otomatisasi ini, karena

Anda dapat membuat dokumen SSM yang memperbarui grup Auto Scaling dan mengubah ukuran instans.

Tujuan desain ketersediaan

Kami berasumsi bahwa minimal beberapa kegagalan akan memerlukan keputusan manual untuk melaksanakan pemulihan, tetapi dengan otomatisasi yang bagus dalam skenario ini, kami berasumsi bahwa hanya dua peristiwa per tahun yang akan memerlukan keputusan ini. Kami meluangkan waktu 20 menit memutuskan untuk melaksanakan pemulihan, dan berasumsi bahwa pemulihan diselesaikan dalam 10 menit. Ini menyiratkan bahwa diperlukan 30 menit untuk pulih dari kegagalan. Dengan asumsi dua kegagalan per tahun, waktu dampak yang kami perkirakan untuk setahun yakni 60 menit.

Ini berarti batas atas ketersediaan yakni 99,95%. Ketersediaan aktual juga akan bergantung pada tingkat kegagalan yang sesungguhnya, durasi kegagalan, dan seberapa cepat setiap kegagalan pulih secara aktual. Untuk arsitektur ini, kami berasumsi bahwa aplikasinya online terus selama pembaruan. Berdasarkan ini, tujuan desain ketersediaan 99,95%.

Ringkasan

Topik	Implementasi
Pantau sumber daya	Pemeriksaan kondisi di semua lapisan, termasuk kondisi DNS di tingkat Wilayah AWS, dan di KPI; peringatan dikirimkan ketika alarm yang dikonfigurasi terpicu; yang memberikan peringatan tentang semua kegagalan. Rapat operasional yang sangat teliti untuk mendeteksi tren dan mendesain tujuan dengan sukses.
Beradaptasi dengan perubahan dalam permintaan	ELB untuk tingkat aplikasi penskalaan otomatis dan web; penyimpanan penskalaan otomatis dan replika baca di beberapa zona di wilayah aktif dan pasif untuk RDS Aurora. Data dan infrastruktur disinkronkan antara Wilayah AWS untuk stabilitas statis.
Implementasikan perubahan	Deployment otomatis lewat canary atau pengembalian ke sebelumnya yang otomatis

Topik	Implementasi
	dan blue/green ketika KPI atau peringatan menunjukkan masalah yang tak terdeteksi dalam aplikasi, deployment dibuat ke satu zona isolasi dalam satu Wilayah AWS setiap kali.
Cadangkan data	Cadangan otomatis di setiap Wilayah AWS lewat RDS untuk memenuhi restorasi otomatis dan RPO yang dipraktikkan secara teratur dalam aktivitas game. Data S3 dan RDS Aurora secara otomatis dan asinkron direplikasi dari wilayah aktif ke wilayah pasif.
Rancang untuk ketangguhan	Penskalaan otomatis untuk memberikan tingkat aplikasi dan web dengan pemulihan mandiri; RDS bersifat Multi-AZ; failover regional dikelola secara manual dengan situs statis yang dipresentasikan ketika failover terjadi.
Uji ketangguhan	Pengujian kesalahan zona isolasi dan komponen sudah dalam alur dan dipraktik kan dengan staf operasional secara teratur dalam aktivitas game; ada playbook untuk mendiagnosis masalah yang tak diketahui; dan ada proses Analisis Akar Masalah, dengan jalur komunikasi untuk apa masalahnya, dan bagaimana masalah dikoreksi atau dicegah. Koreksi RCA diprioritaskan melebihi rilis fitur untuk deployment dan implementasi dengan segera.

Topik	Implementasi
Rencanakan pemulihan bencana (DR)	Warm Standby di-deploy di wilayah lain. Infrastruktur diskalakan ke luar menggunak an alur kerja yang dilaksanakan menggunak an AWS Step Functions atau Dokumen AWS Systems Manager. Cadangan terenkripsi lewat RDS. Replika baca lintas wilayah antara dua Wilayah AWS. Replikasi aset statis lintas wilayah di Amazon S3. Restorasi ke Wilayah AWS yang saat ini aktif, dipraktikkan dalam aktivitas game, dan dikoordinasikan dengan AWS.

Skenario 5 angka 9 (99,999%) atau lebih tinggi dengan waktu pemulihan kurang dari satu menit

Tujuan ketersediaan untuk aplikasi ini hampir tidak memerlukan waktu henti atau kehilangan data untuk waktu tertentu. Aplikasi yang dapat memiliki tujuan ketersediaan ini antara lain, aplikasi perbankan, investasi, keuangan, pemerintah, dan aplikasi penting bagi bisnis yang merupakan bisnis inti dari usaha yang menghasilkan pendapatan sangat besar. Keinginan untuk memiliki penyimpanan data yang sangat konsisten dan redundansi lengkap di semua lapisan. Kami telah memilih penyimpanan data berbasis SQL. Tetapi, dalam beberapa skenario, kami akan kesulitan untuk mencapai RPO yang sangat kecil. Jika Anda dapat mempartisi data Anda, mungkin tidak akan ada kehilangan data. Ini mungkin mengharuskan Anda untuk menambahkan latensi dan logika aplikasi untuk memastikan Anda memiliki data yang konsisten antara lokasi geografis, serta kemampuan untuk memindahkan atau menyalin data antara partisi. Melakukan partisi ini mungkin akan lebih mudah jika Anda menggunakan basis data NoSQL.

Kami dapat meningkatkan ketersediaan lebih lanjut dengan menggunakan pendekatan Active-Active di beberapa Wilayah AWS. Beban kerja ini akan di-deploy di semua Wilayah yang diinginkan yang stabil secara statistik di seluruh wilayah (sehingga wilayah yang tersisa dapat menangani beban dengan kehilangan satu wilayah). Satu perutean mengarahkan lalu lintas ke lokasi geografis dengan kondisi bagus dan otomatis mengubah tujuan ketika lokasinya tidak berkondisi bagus, serta

menghentikan sementara lapisan replikasi data. Amazon Route 53 menawarkan pemeriksaan kondisi dengan interval 10 detik serta menawarkan TTL pada set catatan Anda hingga sesingkat satu detik.

Pantau sumber daya

Sama seperti skenario 3½ angka 9, ditambah peringatan ketika Wilayah dideteksi tidak berkondisi bagus, dan lalu lintas dirutekan menjauh darinya.

Beradaptasi dengan perubahan dalam permintaan

Sama seperti skenario 3½ angka 9.

Implementasikan perubahan

Alur deployment akan memiliki kumpulan pengujian penuh, termasuk uji performa, beban, dan injeksi kegagalan. Kami akan melakukan deploy pembaruan menggunakan canary atau deployment blue/green ke satu zona isolasi setiap kali, di satu Wilayah sebelum memulai di yang lain. Selama deployment, versi lama kasih akan dijalankan pada instans untuk memfasilitasi pengembalian lebih cepat ke versi sebelumnya. Deployment bersifat sepenuhnya otomatis, termasuk pengembalian ke yang sebelumnya jika KPI menunjukkan ada masalah. Pemantauan akan mencakup metrik kesuksesan serta memberikan peringatan ketika masalah timbul.

Akan ada runbook untuk pelacakan performa dan persyaratan pelaporan yang ketat. Jika operasi yang sukses memiliki kecenderungan untuk gagal agar memenuhi tujuan ketersediaan atau performa, playbook akan digunakan untuk menetapkan apa yang menyebabkan kecenderungan tersebut. Akan ada playbook untuk insiden keamanan dan mode kegagalan yang tidak ditemukan. Juga akan ada playbook untuk menetapkan akar masalah dari kegagalan.

Tim yang membangun situs web juga mengoperasikan situs web. Tim tersebut akan mengidentifikasi koreksi kesalahan setiap kegagalan yang tidak terduga dan memprioritaskan deployment perbaikan setelah perbaikan diimplementasikan. Kami juga akan melibatkan AWS Support untuk Manajemen Peristiwa Infrastruktur.

Cadangkan data

Sama seperti skenario 3½ angka 9.

Rancang untuk ketangguhan

Aplikasi harus dibangun menggunakan pola ketangguhan perangkat lunak/aplikasi. Mungkin saja banyak lapisan perutean lain dapat diperlukan untuk mengimplementasikan ketersediaan

yang dibutuhkan. Kompleksitas implementasi tambahan ini tidak boleh diremehkan. Aplikasi akan diimplementasikan dalam zona isolasi kesalahan deployment, dan dipartisi dan di-deploy sedemikian sehingga bahkan peristiwa di semua Wilayah tidak akan memengaruhi semua pelanggan.

Uji ketangguhan

Kami akan memvalidasi arsitektur melalui aktivitas game menggunakan runbook untuk memastikan kami dapat menjalankan tugas dan tidak menyimpang dari prosedur.

Rencanakan pemulihan bencana (DR)

Deployment Active-Active multi-wilayah dengan data dan infrastruktur beban kerja lengkap di beberapa wilayah. Menggunakan strategi lokal baca, global tulis, satu wilayah merupakan basis data utama untuk semua penulisan, dan data direplikasi untuk pembacaan ke wilayah lain. Jika wilayah DB utama gagal, DB baru harus digalakkan. Lokal baca, global tulis memiliki pengguna yang ditetapkan ke wilayah beranda di mana penulisan DB ditangani. Ini memungkinkan pengguna membaca atau menulis dari wilayah mana pun, tetapi memerlukan logika kompleks untuk mengelola potensi konflik data di seluruh penulisan di wilayah yang berbeda.

Ketika wilayah dideteksi tidak berkondisi bagus, lapisan perutean otomatis merutekan lalu lintas ke wilayah tersisa dengan kondisi bagus. Tidak diperlukan intervensi manual.

Penyimpanan data harus direplikasi antara Wilayah dengan cara yang dapat mengatasi potensi konflik. Alat dan proses otomatis akan harus dibuat untuk menyalin atau memindahkan data antara partisi karena alasan latensi dan untuk menyeimbangkan permintaan atau jumlah data dalam setiap partisi. Perbaikan resolusi konflik data juga akan memerlukan runbook operasional tambahan.

Tujuan desain ketersediaan

Kami berasumsi bahwa investasi besar dilakukan untuk mengotomatiskan semua pemulihan, dan pemulihan dapat diselesaikan dalam satu menit. Kami berasumsi tidak ada pemulihan yang dipicu secara manual, tetapi hingga satu tindakan pemulihan otomatis per kuartal. Ini menyiratkan empat menit per tahun untuk pemulihan. Kami berasumsi bahwa aplikasinya online terus selama pembaruan. Berdasarkan ini, tujuan desain ketersediaan 99,999%.

Ringkasan

Topik	Implementasi
Pantau sumber daya	Pemeriksaan kondisi di semua lapisan, termasuk kondisi DNS di tingkat Wilayah AWS, dan di KPI; peringatan dikirimkan ketika alarm yang dikonfigurasi terpicu; yang memberikan peringatan tentang semua kegagalan. Rapat operasional yang sangat teliti untuk mendeteksi tren dan mendesain tujuan dengan sukses.
Beradaptasi dengan perubahan dalam permintaan	ELB untuk tingkat aplikasi penskalaan otomatis dan web; penyimpanan penskalaan otomatis dan replika baca di beberapa zona di wilayah aktif dan pasif untuk Aurora RDS. Data dan infrastruktur disinkronkan antara Wilayah AWS untuk stabilitas statis.
Implementasikan perubahan	Deployment otomatis lewat canary atau pengembalian ke sebelumnya yang otomatis dan blue/green ketika KPI atau peringatan menunjukkan masalah yang tak terdeteksi dalam aplikasi, deployment dibuat ke satu zona isolasi dalam satu Wilayah AWS setiap kali.
Cadangkan data	Cadangan otomatis di setiap Wilayah AWS lewat RDS untuk memenuhi restorasi otomatis dan RPO yang dipraktikkan secara teratur dalam aktivitas game. Data S3 dan RDS Aurora secara otomatis dan asinkron direplikasi dari wilayah aktif ke wilayah pasif.
Rancang untuk ketangguhan	Zona isolasi kegagalan yang diimpleme ntasikan untuk aplikasi; penskalaan otomatis untuk memberikan tingkat aplikasi dan web dengan pemulihan mandiri; RDS bersifat Multi-AZ; failover regional diotomatiskan.

Topik	Implementasi
Uji ketangguhan	Pengujian kesalahan zona isolasi dan komponen sudah dalam alur dan dipraktik kan dengan staf operasional secara teratur dalam aktivitas game; ada playbook untuk mendiagnosis masalah yang tak diketahui; dan ada proses Analisis Akar Masalah dengan jalur komunikasi untuk apa masalahnya, dan bagaimana masalah dikoreksi atau dicegah. Koreksi RCA diprioritaskan melebihi rilis fitur untuk deployment dan implementasi dengan segera.
Rencanakan pemulihan bencana (DR)	Active-Active di-deploy di minimal dua wilayah. Infrastruktur diskalakan sepenuhnya dan stabil secara statistik di seluruh wilayah. Data dipartisi dan disinkronkan di seluruh wilayah. Cadangan terenkripsi lewat RDS. Kegagalan wilayah dipraktikkan dalam aktivitas game, dan dikoordinasikan dengan AWS. Selama restorasi, basis data utama baru mungkin harus digalakkan.

Sumber daya

Dokumentasi

- Amazon Builders' Library - Cara Amazon membangun dan mengoperasikan perangkat lunak
- Pusat Arsitektur AWS

Lab

- Lab Keandalan AWS Well-Architected

Tautan Eksternal

- Pola Antre Adaptif: Gagal dalam Skala Besar
- Ketersediaan dan Lainnya: Memahami dan meningkatkan ketangguhan sistem terdistribusi di AWS

Buku

- Robert S. Hammer "Pola untuk Perangkat Lunak yang Toleran Terhadap Kesalahan (Patterns for Fault Tolerant Software)"
- Andrew Tanenbaum dan Marten van Steen "Sistem Terdistribusi: Prinsip dan Paradigma (Distributed Systems: Principles and Paradigms)"

Kesimpulan

Kami harap laporan resmi ini dapat mengasah cara berpikir Anda, menawarkan ide baru, atau menimbulkan serangkaian pertanyaan yang mengarahkan pada argumen logis untuk Anda, baik Anda belum familier dengan topik ketersediaan dan keandalan, maupun sudah berpengalaman dan sedang mencari wawasan untuk memaksimalkan ketersediaan beban kerja misi kritis Anda. Kami harap ini memberi Anda pemahaman mendalam tentang tingkat ketersediaan yang tepat berdasarkan kebutuhan bisnis, dan cara mendesain serta mencapai keandalan. Sebaiknya manfaatkan desain, operasional, dan rekomendasi berorientasi pemulihan yang ditawarkan di sini, juga pengetahuan dan pengalaman dari Arsitek Solusi AWS kami. Kami menantikan testimoni dari Anda–terutama tentang kisah sukses Anda dalam mencapai ketersediaan tingkat tinggi di AWS. Hubungi tim akun Anda atau gunakan Hubungi KAMI di situs web kami.

Kontributor

Kontributor dokumen ini antara lain:

- Seth Eliot, Principal Developer Advokat, Amazon Web Services
- Mahanth Jayadeva, Solutions Architect – Well-Architected, Amazon Web Services
- Amulya Sharma, Principal Solutions Architect, Amazon Web Services
- Jason DiDomenico, Senior Solutions Architect – Cloud Foundations, Amazon Web Services
- Marcin Bednarz, Principal Solutions Architect, Amazon Web Services
- Tyler Applebaum, Senior Solutions Architect, Amazon Web Services
- Rodney Lester, Principal Solutions Architect – App Modernization, Amazon Web Services
- Joe Chapman, Senior Solutions Architect, Amazon Web Services
- Adrian Hornsby, Principal System Development Engineer, Amazon Web Services
- Kevin Miller, Vice President – S3, Amazon Web Services
- Shannon Richards, Principal Technical Program Manager, Amazon Web Services
- Laurent Domb, Chief Technologist - Fed Fin, Amazon Web Services
- Kevin Schwarz, Sr. Solutions Architect, Amazon Web Services
- Rob Martell, Principal Cloud Resilience Architect, Amazon Web Services
- Priyam Reddy, Senior Solutions Architect Manager DR, Amazon Web Services
- Jeff Ferris, Principal Technologist, Amazon Web Services
- Matias Battaglia, Senior Solutions Architect, Amazon Web Services

Bacaan lebih lanjut

Untuk informasi tambahan, buka:

- AWS Kerangka Kerja Well-Architected
- AWS Pusat Arsitektur

Revisi Dokumen

Berlangganan umpan RSS untuk memperoleh pemberitahuan tentang pembaruan laporan resmi ini.

Perubahan	Deskripsi	Tanggal
Laporan resmi diperbarui	Praktik terbaik diperbarui dengan panduan implementasi baru.	June 27, 2024
Panduan praktik terbaik yang diperbarui	Praktik terbaik diperbarui dengan panduan baru di area-area berikut: Merancang interaksi di dalam sistem terdistribusi untuk mencegah kegagalan, Merancang interaksi dalam sistem terdistribusi untuk mitigasi atau bertahan dari kegagalan, Memantau sumber daya beban kerja, Merancang beban kerja Anda agar dapat beradaptasi dengan perubahan dalam permintaan, Mengimplementasikan perubahan, dan Menguji keandalan.	December 6, 2023
Panduan praktik terbaik yang diperbarui	Praktik terbaik diperbarui dengan panduan baru di area-area berikut: Memantau sumber daya beban kerja dan Merancang beban kerja Anda agar bertahan dalam kegagalan komponen.	October 3, 2023

Panduan praktik terbaik yang diperbarui	Praktik terbaik diperbarui dengan panduan baru di area-area berikut: Merancang arsitektur layanan beban kerja Anda, Merancang interaksi dalam sistem terdistribusi untuk mitigasi atau bertahan dari kegagalan, dan Memantau sumber daya beban kerja.	July 13, 2023
Pembaruan kecil	Bahasa non-inklusif dihilangkan.	April 13, 2023
Pembaruan untuk Kerangka Kerja baru	Praktik terbaik diperbarui dengan panduan preskriptif dan praktik terbaik baru ditambahkan.	April 10, 2023
Laporan resmi diperbarui	Praktik terbaik diperbarui dengan panduan implementasi baru.	December 15, 2022
Pembaruan kecil	Koreksi angka dan sejumlah perubahan kecil lainnya.	November 17, 2022
Laporan resmi diperbarui	Praktik terbaik diperluas dan rencana pengembangan ditambahkan.	October 20, 2022
Laporan resmi diperbarui	Penambahan dua praktik terbaik baru pada Pilar Keandalan di bagian Gunakan Isolasi Kesalahan untuk Melindungi Beban Kerja Anda dan Rancang Beban Kerja Anda agar Bertahan dalam Kegagalan Komponen.	May 5, 2022

Pembaruan kecil	Penambahan Pilar Pelestarian Lingkungan ke pengantar.	December 2, 2021
Laporan resmi diperbarui	Perbarui panduan Pemulihan Bencana untuk mencakup Pengontrol Pemulihan Aplikasi Route 53. Tambahkan referensi ke DevOps Guru. Perbarui beberapa tautan Sumber daya, dan perubahan editorial kecil lainnya.	October 26, 2021
Pembaruan kecil	Informasi tentang AWS Fault Injection Service (AWS FIS) ditambahkan.	March 15, 2021
Pembaruan kecil	Pembaruan teks kecil.	January 4, 2021

Laporan resmi diperbarui	Lampiran A diperbarui untuk memperbarui Tujuan Desain Ketersediaan untuk Amazon SQS, Amazon SNS, dan Amazon MQ; Susun ulang baris dalam tabel untuk pencarian yang lebih mudah; Tingkatkan penjelasan tentang perbedaan antara ketersedi aan dan pemulihan bencana dan bagaimana keduanya berkontribusi pada keandalan ; Perluas cakupan arsitektur multiwilayah (untuk ketersedi aan) dan strategi multiwilayah (untuk pemulihan bencana); Perbarui buku referensi ke versi terbaru; Perluas penghitungan ketersediaan untuk menyertakan penghitun gan berbasis permintaan, dan penghitungan pintasan; Tingkatkan deskripsi untuk Game Days	December 7, 2020
Pembaruan kecil	Lampiran A diperbarui untuk memperbarui Tujuan Desain Ketersediaan untuk AWS Lambda	October 27, 2020
Pembaruan kecil	Lampiran A diperbarui untuk memperbarui Tujuan Desain Ketersediaan untuk AWS Lambda	July 24, 2020

Pembaruan untuk Kerangka Kerja baru	Pembaruan substansial dan konten baru/yang direvisi, termasuk: Penambahan bagian praktik terbaik “Arsitektur Beban Kerja”, penataan ulang praktik terbaik ke dalam bagian Manajemen Perubahan dan Manajemen Kegagalan, pembaruan Sumber Daya, pembaruan untuk menyertakan sumber daya dan layanan AWS terbaru seperti AWS Global Accelerator, AWS Service Quotas, dan AWS Transit Gateway, penambahan/pembaruan definisi untuk Keandalan, Ketersediaan, Ketangguhan, laporan resmi yang lebih selaras dengan AWS Well-Architected Tool (pertanyaan dan praktik terbaik) yang digunakan untuk Peninjauan Well-Architected, menyusun ulang prinsip-prinsip desain, memindahkan Pulihkan secara otomatis dari kegagalan menjadi sebelum Uji prosedur pemulihan, pembaruan diagram dan format untuk persamaan, penghapusan bagian Layanan Utama yang diganti dengan integrasi referensi ke layanan AWS utama ke dalam praktik terbaik.	July 8, 2020

Pembaruan kecil	Perbaikan tautan yang bermasalah	October 1, 2019
Laporan resmi diperbarui	Lampiran A diperbarui	April 1, 2019
Laporan resmi diperbarui	Penambahan rekomendasi jaringan AWS Direct Connect spesifik dan tujuan desain layanan tambahan	September 1, 2018
Laporan resmi diperbarui	Penambahan bagian Prinsip Desain dan Manajemen Batas. Pembaruan tautan, penghapusan ambiguitas terminologi upstream/downstream, dan penambahan referensi eksplisit ke topik Pilar Keandalan di dalam skenario ketersediaan.	June 1, 2018
Laporan resmi diperbarui	Solusi Lintas Wilayah DynamoDB diganti menjadi Tabel Global DynamoDB. Tujuan desain layanan ditambahkan	March 1, 2018
Pembaruan kecil	Koreksi kecil pada perhitungan ketersediaan agar menyertakan ketersediaan aplikasi	December 1, 2017
Laporan resmi diperbarui	Pembaruan untuk memberikan panduan pada desain dengan ketersediaan tinggi, termasuk konsep, praktik terbaik, dan implementasi contoh.	November 1, 2017

Publikasi awal	Pilar Keamanan - Kerangka Kerja AWS Well-Architected diterbitkan.	November 1, 2016