Mendeteksi dan Mengurangi Kegagalan Abu-abu

Pola Ketahanan Multi-AZ Tingkat Lanjut

Pola Ketahanan Multi-AZ Tingkat Lanjut: Mendeteksi dan Mengurangi Kegagalan Abu-abu

Table of Contents

Pola Ketahanan Multi-AZ Tingkat Lanjut

Tanggal publikasi:11 Juli 2023(Revisi dokumen)

Banyak pelanggan menjalankan beban kerja mereka dalam konfigurasi Multi-Availability Zone (AZ) yang sangat tersedia. Arsitektur ini bekerja dengan baik selama kejadian kegagalan biner, tetapi sering mengalami masalah denganabu-abukegagalan. Manifestasi dari jenis kegagalan ini bisa halus, dan menentang deteksi cepat dan definitif. Tulisan ini memberikan panduan tentang cara instrumen beban kerja untuk mendeteksi dampak dari kegagalan abu-abu yang terisolasi ke Availability Zone tunggal, dan kemudian mengambil tindakan untuk mengurangi dampak tersebut di Availability Zone.

Pengantar

Tujuan dari dokumen ini adalah untuk membantu Anda menerapkan arsitektur Multi-AZ yang tangguh secara lebih efektif. Salah satu praktik terbaik untuk membangun sistem tangguh diCloud Pribadi Virtual AmazonJaringan (VPC) adalah untukmenyebarkan setiap beban kerja ke beberapa Availability Zone.

SebuahZona Ketersediaanadalah satu atau lebih pusat data diskrit dengan daya, jaringan, dan konektivitas yang berlebihan. Menggunakan beberapa Availability Zone memungkinkan Anda mengoperasikan beban kerja yang lebih tersedia, toleran terhadap kesalahan, dan dapat diskalakan daripada yang mungkin dilakukan dari satu pusat data.

BanyakAWSlayanan, seperti[Penskalaan Otomatis Amazon Elastic Compute Cloud (EC2)](#)atau[Layanan Database Relasional Amazon](#)(Amazon RDS), menyediakan konfigurasi Multi-AZ. Layanan ini tidak mengharuskan Anda untuk membangun observability tambahan atau tooling failover. Mereka membuat beban kerja tangguh untuk mode kegagalan biner yang mudah dideteksi dalam[Wilayah AWS](#)yang memengaruhi Availability Zone tunggal. Ini bisa berupa kegagalan perangkat keras fisik lengkap, kehilangan daya, atau bug perangkat lunak laten yang memengaruhi sebagian besar sumber daya.

Tapi ada kategori kegagalan lain yang disebutkegagalan abu-abu, yang manifestasinya halus dan menentang deteksi cepat dan definitif. Hal ini pada gilirannya menghasilkan waktu yang lebih lama untuk mengurangi dampak yang disebabkan oleh kegagalan. Makalah ini berfokus pada dampak kegagalan abu-abu pada arsitektur Multi-AZ, cara mendeteksinya, dan, akhirnya, cara menguranginya.

Panduan yang disediakan dalam whitepaper ini sebagian besar berlaku untuk kelas beban kerja tertentu yang:

- Terutama menggunakan zonalAWSjasa
- Perlu meningkatkan ketahanan Wilayah tunggal
- Bersedia melakukan investasi yang signifikan untuk membangun pola observabilitas dan ketahanan yang diperlukan

Dalam beban kerja ini, Anda mungkin tidak mau membuat beberapa, atau semua, dari pengorbanan yang disajikan???, atau tidak memiliki opsi untuk menggunakan beberapa Wilayah. Jenis beban kerja ini cenderung mewakili sebagian kecil dari keseluruhan portofolio Anda dan karenanya panduan ini harus dipertimbangkan pada tingkat beban kerja versus pada tingkat platform.

Kegagalan abu-abu

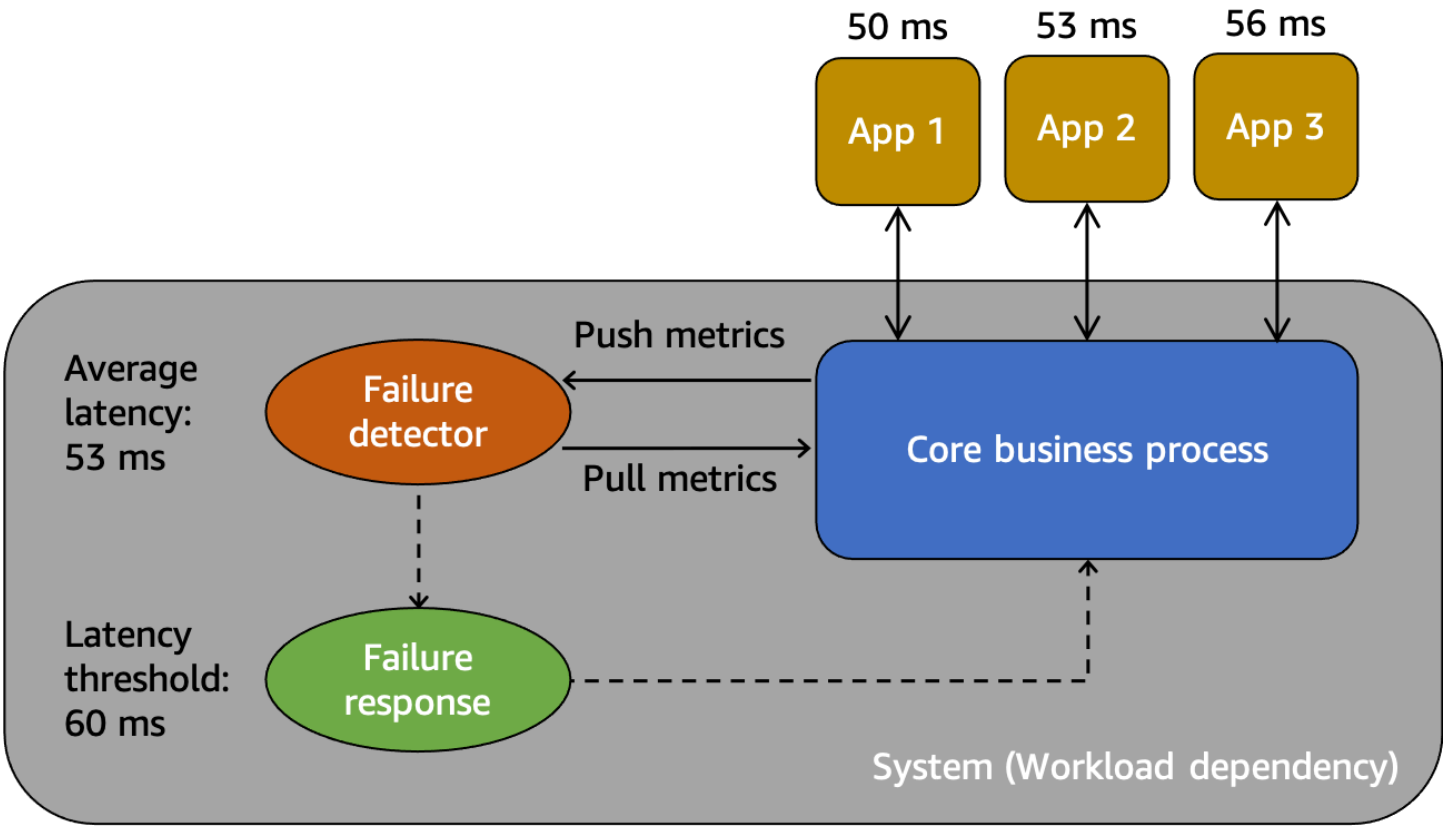
Kegagalan abu-abu didefinisikan oleh karakteristikobservabilitas diferensial, yang berarti bahwa entitas yang berbeda mengamati kegagalan secara berbeda. Mari kita definisikan apa artinya ini.

Observabilitas diferensial

Beban kerja yang Anda operasikan biasanya memiliki dependensi. Misalnya, ini bisa menjadiAWSlayanan cloud yang Anda gunakan untuk membangun beban kerja Anda atau penyedia identitas pihak ketiga (IDP) yang Anda gunakan untuk federasi. Dependensi tersebut hampir selalu menerapkan pengamatan mereka sendiri, merekam metrik tentang kesalahan, ketersediaan, dan latensi antara lain yang dihasilkan oleh penggunaan pelanggan mereka. Ketika ambang batas dilintasi untuk salah satu metrik ini, ketergantungan biasanya mengambil beberapa tindakan untuk memperbaikinya.

Dependensi ini biasanya memiliki banyak konsumen layanan mereka. Konsumen juga menerapkan observabilitas mereka sendiri dan merekam metrik dan log tentang interaksi mereka dengan dependensinya, merekam hal-hal seperti berapa banyak latensi yang ada dalam pembacaan disk, berapa banyak permintaan API yang gagal, atau berapa lama kueri database.

Interaksi dan pengukuran ini digambarkan dalam model abstrak pada gambar berikut.

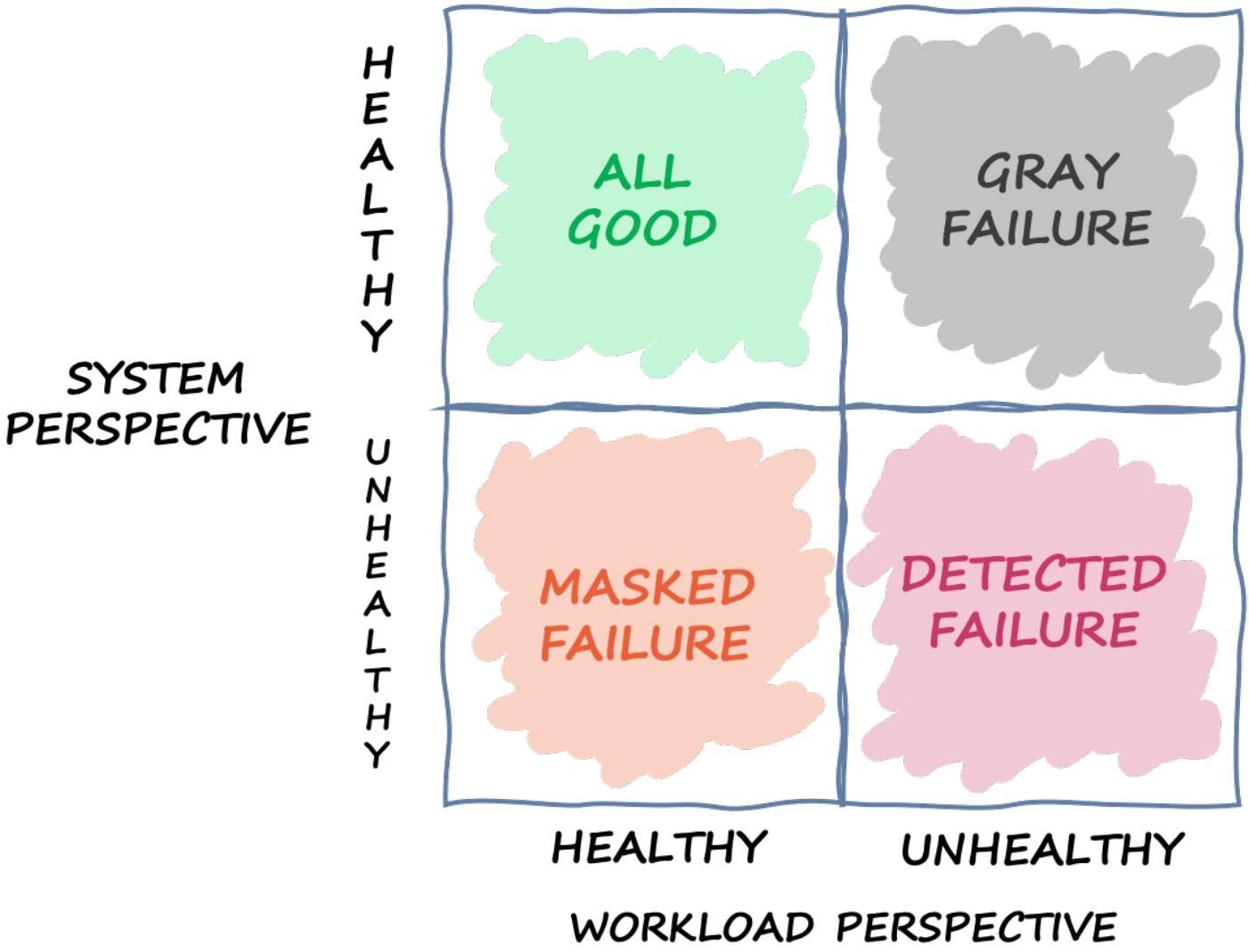


Model abstrak untuk memahami kegagalan abu-abu

Pertama, kita memilikisistem, yang merupakan ketergantungan untuk konsumen App 1, App 2, dan App 3 dalam skenario ini. Sistem ini memiliki detektor kegagalan yang memeriksa metrik yang dibuat dari proses bisnis inti. Ini juga memiliki mekanisme respons kegagalan untuk mengurangi atau memperbaiki masalah yang diamati oleh detektor kegagalan. Sistem melihat latensi rata-rata keseluruhan 53 ms dan telah menetapkan ambang batas untuk memanggil mekanisme respons kegagalan ketika latensi rata-rata melebihi 60 ms. App 1, App 2, dan App 3 juga membuat pengamatan mereka sendiri tentang interaksi mereka dengan sistem, merekam latensi rata-rata 50 ms, 53 ms, dan 56 ms masing-masing.

Observabilitas diferensial adalah situasi di mana salah satu konsumen sistem mendeteksi bahwa sistem tidak sehat, tetapi pemantauan sistem sendiri tidak mendeteksi masalah atau dampaknya tidak melewati ambang alarm. Mari kita bayangkan bahwa App 1 mulai mengalami latensi rata-rata 70 ms, bukan 50ms. Aplikasi 2 dan Aplikasi 3 tidak melihat perubahan latensi rata-rata mereka. Ini meningkatkan latensi rata-rata sistem yang mendasarinya menjadi 59,66 ms, tetapi ini tidak melewati ambang latensi untuk mengaktifkan mekanisme respons kegagalan. Namun, App 1 melihat peningkatan latensi 40%. Hal ini dapat memengaruhi ketersediaannya dengan melampaui batas waktu klien yang dikonfigurasi untuk Aplikasi 1, atau dapat menyebabkan dampak cascading dalam

rantai interaksi yang lebih lama. Dari perspektif App 1, sistem yang mendasarinya tergantung pada tidak sehat, tetapi dari perspektif sistem itu sendiri juga App 2 dan App 3, sistemnya sehat. Gambar berikut merangkum perspektif yang berbeda ini.



Kuadran yang mendefinisikan berbagai negara suatu sistem dapat didasarkan pada perspektif yang berbeda

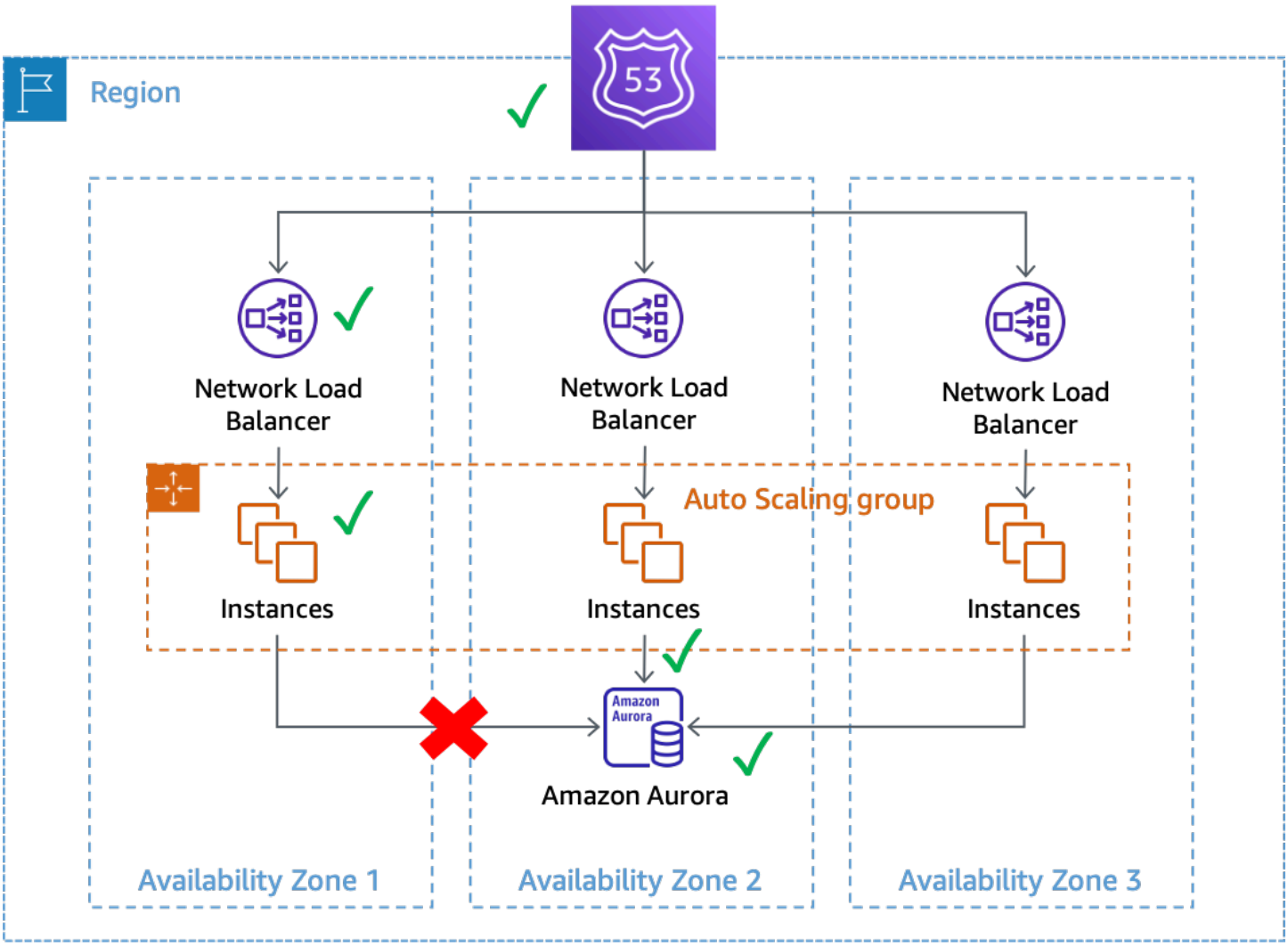
Kegagalan juga dapat melintasi kuadran ini. Suatu peristiwa bisa dimulai sebagai kegagalan abu-abu, kemudian menjadi kegagalan yang terdeteksi, kemudian pindah ke kegagalan bertopeng, dan kemudian mungkin kembali ke kegagalan abu-abu. Tidak ada siklus yang ditentukan, dan hampir selalu ada kemungkinan kegagalan kambuh sampai akar penyebabnya ditangani.

Kesimpulan yang kami ambil dari ini adalah bahwa beban kerja tidak selalu dapat mengandalkan sistem yang mendasarinya untuk mendeteksi dan mengurangi kegagalan. Tidak peduli seberapa canggih dan tangguh sistem yang mendasarinya, akan selalu ada kemungkinan bahwa kegagalan

bisa tidak terdeteksi atau tetap berada di bawah ambang reaksi. Konsumen dari sistem itu, seperti App 1, perlu diperlengkapi untuk mendeteksi dan mengurangi dampak yang disebabkan oleh kegagalan abu-abu. Ini membutuhkan pengamatan bangunan dan mekanisme pemulihan untuk situasi ini.

Contoh kegagalan abu-abu

Kegagalan abu-abu dapat berdampak pada sistem Multi-AZ diAWS. Misalnya, ambil armadaAmazon EC2instans dalam grup Auto Scaling yang diterapkan di tiga Availability Zone. Semuanya terhubung ke database Amazon Aurora dalam satu Availability Zone. Kemudian, terjadi kegagalan abu-abu yang memengaruhi jaringan antara Availability Zone 1 dan Availability Zone 2. Hasil dari gangguan ini adalah persentase koneksi database baru dan yang sudah ada dari instance di Availability Zone 1 gagal. Situasi ini ditunjukkan pada gambar berikut.



Kegagalan abu-abu yang memengaruhi koneksi database dari instans di Availability Zone 1

Dalam contoh ini, Amazon EC2 melihat instans di Availability Zone 1 sehat karena terus lolospemeriksaan status sistem dan instance. Amazon EC2 Auto Scaling juga tidak mendeteksi dampak langsung ke Availability Zone apa pun, dan terus berlanjutmeluncurkan kapasitas di Availability Zone yang dikonfigurasi. Network Load Balancer (NLB) juga melihat instance di belakangnya sehat seperti pemeriksaan kesehatan Route 53 yang dilakukan terhadap titik akhir NLB. Demikian pula, Amazon Relational Database Service (Amazon RDS) melihat klaster database sebagai sehat dan tidakmemicu failover otomatis. Kami memiliki banyak layanan berbeda yang semuanya melihat layanan dan sumber daya mereka sehat, tetapi beban kerja mendeteksi kegagalan yang memengaruhi ketersediaannya. Ini adalah kegagalan abu-abu.

Menanggapi kegagalan abu-abu

Ketika Anda mengalami kegagalan abu-abu diAWSlingkungan, Anda umumnya memiliki tiga pilihan yang tersedia:

- Jangan lakukan apa pun dan tunggu sampai kerusakan berakhir.
- Jika gangguan diisolasi ke Availability Zone tunggal, evakuasi Availability Zone tersebut.
- Failover ke yang lainWilayah AWSdan menggunakan manfaatAWSIsolasi regional untuk mengurangi dampak.

BanyakAWSpelanggan baik-baik saja dengan opsi satu untuk sebagian besar beban kerja mereka. Mereka menerima memiliki kemungkinan diperpanjangTujuan Waktu Pemulihan (RTO)dengan tradeoff bahwa mereka tidak harus membangun observabilitas tambahan atau solusi ketahanan. Pelanggan lain memilih untuk menerapkan opsi ketiga,Pemulihan Bencana Multi Wilayah(DR), sebagai rencana mitigasi mereka untuk berbagai jumlah mode kegagalan. Arsitektur Multi-Region dapat bekerja dengan baik dalam skenario ini. Namun, ada beberapa tradeoff saat menggunakan pendekatan ini (lihatAWSDasar-dasar Multi-Wilayahuntuk diskusi lengkap tentang pertimbangan Multi-wilayah).

Pertama, membangun dan mengoperasikan arsitektur multi-wilayah dapat menjadi usaha yang menantang, kompleks, dan berpotensi mahal. Arsitektur Multi-Region memerlukan pertimbangan yang cermatStrategi DRAnda pilih. Mungkin tidak layak secara fiskal untuk menerapkan solusi DR aktif multi-wilayah hanya untuk menangani gangguan zonal, sementara strategi pencadangan dan pemulihan mungkin tidak memenuhi persyaratan ketahanan Anda. Selain itu, failovers multi-wilayah harus terus dipraktekkan dalam produksi sehingga Anda yakin mereka akan bekerja bila diperlukan. Ini semua membutuhkan banyak waktu dan sumber daya khusus untuk membangun, mengoperasikan, dan menguji.

Kedua, replikasi data diWilayah AWSmemakaiAWSlayanan hari ini semua dilakukan asynchronous. Replikasi asinkron dapat mengakibatkan kehilangan data. Ini berarti bahwa selama failover Regional, ada peluang untuk sejumlah kehilangan data dan inkonsistensi. Toleransi Anda terhadap jumlah kehilangan data didefinisikan sebagai AndaTujuan Titik Pemulihan (RPO). Pelanggan, untuk siapa konsistensi data yang kuat adalah persyaratan, harus membangun sistem rekonsiliasi untuk memperbaiki masalah konsistensi ini ketika Wilayah utama tersedia lagi. Atau, mereka harus membangun replikasi sinkron atau sistem penulisan ganda mereka sendiri, yang dapat berdampak signifikan pada latensi respons, biaya, dan kompleksitas. Mereka juga membuat Region sekunder ketergantungan keras untuk setiap transaksi, yang berpotensi mengurangi ketersediaan sistem secara keseluruhan.

Akhirnya, untuk banyak beban kerja yang menggunakan pendekatan aktif/siaga, ada jumlah waktu yang tidak nol yang diperlukan untuk melakukan failover ke Wilayah lain. Portofolio beban kerja Anda mungkin perlu diturunkan di Wilayah utama dalam urutan tertentu, perlu menguras koneksi, atau menghentikan proses tertentu. Kemudian, layanan mungkin perlu dibawa kembali dalam urutan tertentu. Sumber daya baru mungkin juga perlu disediakan atau memerlukan waktu untuk lulus pemeriksaan kesehatan yang diperlukan sebelum dibawa ke layanan. Proses failover ini dapat dialami sebagai periode tidak tersedianya lengkap. Inilah yang menjadi perhatian RTO.

Di dalam Wilayah, banyakAWSLayanan menawarkan persistensi data yang sangat konsisten. Penggunaan penerapan Multi-AZ Amazon RDSreplikasi sinkron. Layanan Penyimpanan Sederhana Amazon(Amazon S3) menawarkankuatread-after-writekonsistensi. Penyimpanan Blok Elastis Amazon(Amazon EBS) menawarkansnapshot konsisten kecelakaan multi-volume. Amazon DynamoDBbisamelakukan pembacaan yang sangat konsisten. Fitur-fitur ini dapat membantu Anda mencapai RPO yang lebih rendah (dalam kebanyakan kasus nol RPO) dalam satu Wilayah daripada yang Anda bisa dalam arsitektur multi-wilayah.

Mengevakuasi Availability Zone dapat memiliki RTO yang lebih rendah daripada strategi Multi-region, karena infrastruktur dan sumber daya Anda sudah disediakan di Availability Zone. Alih-alih perlu hati-hati memesan layanan yang diturunkan dan dicadangkan, atau menguras koneksi, arsitektur Multi-AZ dapat terus beroperasi secara statis ketika Availability Zone terganggu. Alih-alih periode ketidaktersediaan lengkap yang dapat terjadi selama failover Regional, selama evakuasi Availability Zone, banyak sistem mungkin hanya melihat sedikit degradasi, karena pekerjaan digeser ke Availability Zone yang tersisa. Jika sistem telah dirancang untuk menjadistabil secara statisuntuk kegagalan Availability Zone (dalam hal ini, itu berarti memiliki kapasitas yang disediakan sebelumnya di Availability Zone lainnya untuk menyerap beban), pelanggan beban kerja mungkin tidak melihat dampak sama sekali.

> Ada kemungkinan bahwa gangguan pada Availability Zone tunggal berdampak pada satu atau lebihAWS Layanan regionalselain beban kerja Anda. Jika Anda mengamati dampak Regional, Anda harus memperlakukan acara tersebut sebagai gangguan layanan Regional meskipun sumber dampaknya berasal dari Availability Zone tunggal. Mengevakuasi Availability Zone tidak akan mengurangi jenis masalah ini. Gunakan rencana respons yang Anda miliki untuk menanggapi gangguan layanan Regional saat ini terjadi.

Sisa dokumen ini berfokus pada opsi kedua, mengevakuasi Availability Zone, sebagai cara untuk mencapai RTO dan RPO yang lebih rendah untuk kegagalan abu-abu single-AZ. Pola-pola ini dapat membantu mencapai nilai dan efisiensi arsitektur Multi-AZ yang lebih baik dan, untuk sebagian besar kelas beban kerja, dapat mengurangi kebutuhan untuk membuat arsitektur Multi-wilayah untuk menangani jenis acara ini.

Observabilitas multi-AZ

Untuk dapat mengevakuasi Availability Zone selama peristiwa yang diisolasi ke Availability Zone tunggal, pertama-tama Anda harus dapat mendeteksi bahwa kegagalan tersebut, pada kenyataannya, diisolasi ke satu Availability Zone. Ini membutuhkan visibilitas kesetiaan tinggi tentang bagaimana sistem berperilaku di setiap Availability Zone. Banyak AWS layanan menyediakan out-of-the-box metrik yang memberikan wawasan operasional tentang sumber daya Anda. Misalnya, Amazon EC2 menyediakan banyak metrik seperti CPU pemanfaatan, membaca dan menulis disk, dan lalu lintas jaringan masuk dan keluar.

Namun, saat Anda membangun beban kerja yang menggunakan layanan ini, Anda memerlukan lebih banyak visibilitas daripada hanya metrik standar tersebut. Anda ingin visibilitas ke dalam pengalaman pelanggan yang disediakan oleh beban kerja Anda. Selain itu, metrik Anda harus disejajarkan dengan Availability Zone tempat metrik tersebut diproduksi. Ini adalah wawasan yang Anda butuhkan untuk mendeteksi kegagalan abu-abu yang dapat diamati secara berbeda. Tingkat visibilitas itu membutuhkan instrumentasi.

Instrumentasi membutuhkan penulisan kode eksplisit. Kode ini harus melakukan hal-hal seperti mencatat berapa lama tugas berlangsung, menghitung berapa banyak item yang berhasil atau gagal, mengumpulkan metadata tentang permintaan, dan sebagainya. Anda juga perlu menentukan ambang batas sebelumnya untuk menentukan apa yang dianggap normal dan apa yang tidak. Anda harus menguraikan tujuan dan ambang batas keparahan yang berbeda untuk latensi, ketersediaan, dan jumlah kesalahan dalam beban kerja Anda. Artikel Perpustakaan Pembangun Amazon Instrumentasi sistem terdistribusi untuk visibilitas operasional menyediakan sejumlah praktik terbaik.

Metrik harus dihasilkan dari sisi server maupun sisi klien. Praktik terbaik untuk menghasilkan metrik sisi klien dan memahami pengalaman pelanggan adalah menggunakan kenari, perangkat lunak yang secara teratur memeriksa beban kerja Anda dan mencatat metrik.

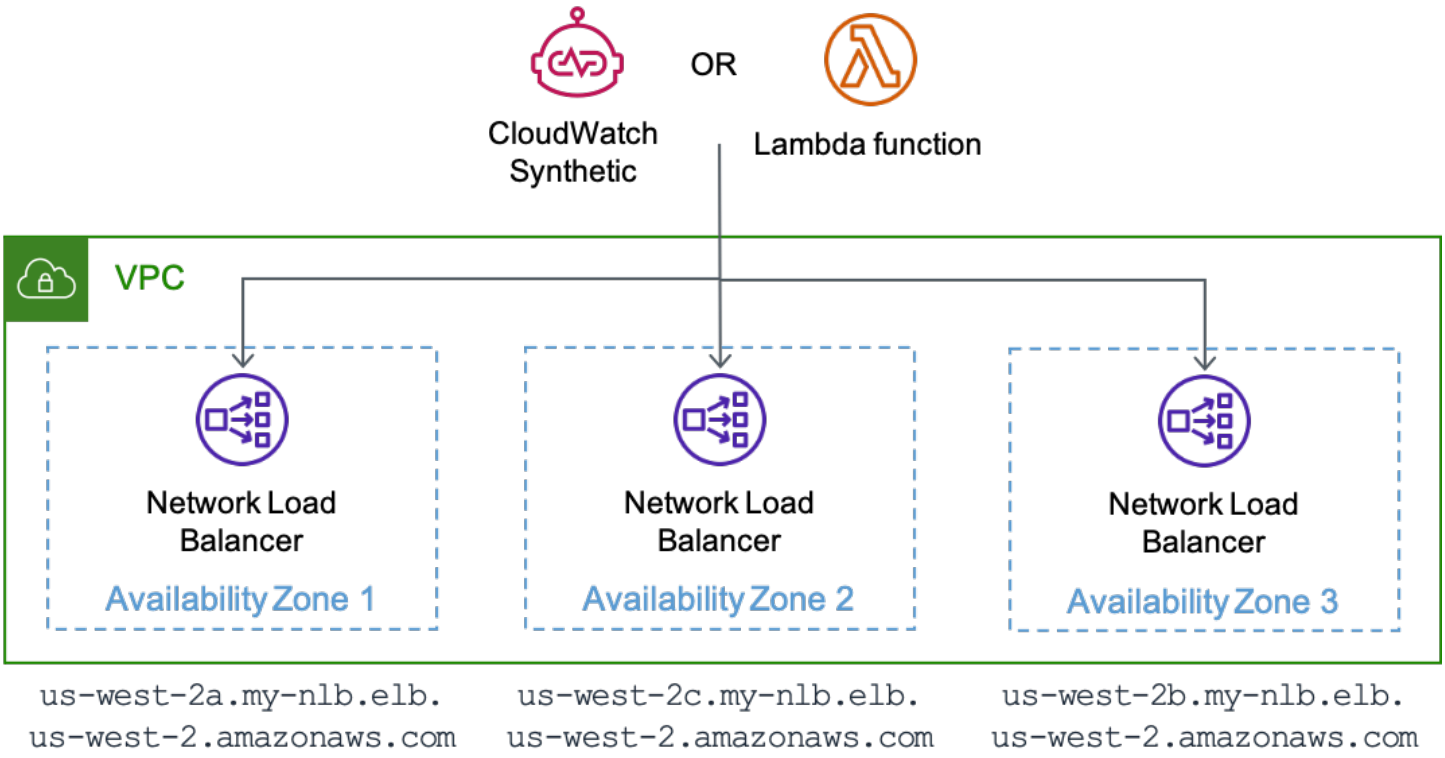
Selain menghasilkan metrik ini, Anda juga perlu memahami konteksnya. Salah satu cara untuk melakukannya adalah dengan menggunakan dimensi. Dimensi memberi metrik identitas unik, dan membantu menjelaskan apa yang dikatakan metrik kepada Anda. Untuk metrik yang digunakan untuk mengidentifikasi kegagalan dalam beban kerja Anda (misalnya, latensi, ketersediaan, atau jumlah kesalahan), Anda perlu menggunakan dimensi yang sejajar dengan batas isolasi kesalahan Anda.

Misalnya, jika Anda menjalankan layanan web di satu Wilayah, di beberapa Availability Zone, menggunakan kerangka web M odel-view-controller (MVC), Anda harus menggunakan `Region`

`Availability Zone ID`,`Controller`,`Action`,, dan `InstanceId` sebagai dimensi untuk kumpulan dimensi Anda (jika Anda menggunakan layanan mikro, Anda mungkin menggunakan nama dan HTTP metode layanan alih-alih nama pengontrol dan tindakan). Ini karena Anda mengharapkan berbagai jenis kegagalan diisolasi oleh batas-batas ini. Anda tidak akan mengharapkan bug dalam kode layanan web Anda yang memengaruhi kemampuannya untuk membuat daftar produk untuk juga memengaruhi halaman beranda. Demikian pula, Anda tidak akan mengharapkan EBS volume penuh pada satu EC2 instance untuk memengaruhi EC2 instance lain dari menyajikan konten web Anda. Dimensi ID Availability Zone adalah hal yang memungkinkan Anda mengidentifikasi dampak terkait Availability Zone secara konsisten. Akun AWS Anda dapat menemukan ID Availability Zone di beban kerja Anda dengan berbagai cara. Lihat Lampiran A — Mendapatkan ID Availability Zone untuk beberapa contoh.

Meskipun dokumen ini terutama menggunakan Amazon EC2 sebagai sumber daya komputasi dalam contoh, `InstanceId` dapat diganti dengan ID penampung untuk Amazon Elastic Container Service (AmazonECS) dan Amazon Elastic Kubernetes Service EKS (Amazon) menghitung sumber daya sebagai komponen dimensi Anda.

Kenari Anda juga dapat menggunakan`Controller`,, `ActionAZ-ID`, dan `Region` sebagai dimensi dalam metriknya jika Anda memiliki titik akhir zona untuk beban kerja Anda. Dalam hal ini, sejajarkan kenari Anda untuk berjalan di Availability Zone yang sedang mereka uji. Ini memastikan bahwa jika peristiwa Availability Zone yang terisolasi memengaruhi Availability Zone tempat kenari Anda berjalan, itu tidak merekam metrik yang membuat Availability Zone berbeda yang sedang diuji tampak tidak sehat. Misalnya, kenari Anda dapat menguji setiap titik akhir zona untuk layanan di belakang Network Load Balancer () atau Application Load Balancer NLB () menggunakan nama zonalnyaALB. DNS



Sebuah kenari yang berjalan pada CloudWatch Synthetics atau fungsi AWS Lambda yang menguji setiap titik akhir zona NLB

Dengan menghasilkan metrik dengan dimensi ini, Anda dapat membuat CloudWatch alarm Amazon yang memberi tahu Anda saat perubahan ketersediaan atau latensi terjadi dalam batas-batas tersebut. Anda juga dapat dengan cepat menganalisis data tersebut menggunakan dasbor. Untuk menggunakan metrik dan log secara efisien, Amazon CloudWatch menawarkan format metrik tertanam (EMF) yang memungkinkan Anda menyematkan metrik kustom dengan data log. CloudWatchsecara otomatis mengekstrak metrik kustom sehingga Anda dapat memvisualisasikan dan alarm pada mereka. AWS menyediakan beberapa pustaka klien untuk berbagai bahasa pemrograman yang membuatnya mudah untuk memulaiEMF. Mereka dapat digunakan dengan AmazonEC2, AmazonECS, Amazon EKS AWS Lambda, dan lingkungan lokal. Dengan metrik yang disematkan ke dalam log, Anda juga dapat menggunakan Amazon CloudWatch Contributor Insights untuk membuat grafik deret waktu yang menampilkan data kontributor. Dalam skenario ini, kita dapat menampilkan data yang dikelompokkan berdasarkan dimensi seperti `AZ-ID`Instance`Id`,, atau `Controller` serta bidang lain di log seperti `SuccessLatency` atau`HttpResponseCode`.

```
{
  "_aws": {
    "Timestamp": 1634319245221,
```

```
    "CloudWatchMetrics": [
      {
        "Namespace": "workloadname/frontend",
        "Metrics": [
          { "Name": "2xx", "Unit": "Count" },
          { "Name": "3xx", "Unit": "Count" },
          { "Name": "4xx", "Unit": "Count" },
          { "Name": "5xx", "Unit": "Count" },
          { "Name": "SuccessLatency", "Unit": "Milliseconds" }
        ],
        "Dimensions": [
          [ "Controller", "Action", "Region", "AZ-ID", "InstanceId"],
          [ "Controller", "Action", "Region", "AZ-ID"],
          [ "Controller", "Action", "Region"]
        ]
      }
    ],
    "LogGroupName": "/loggroupname"
  },
  "CacheRefresh": false,
  "Host": "use1-az2-name.example.com",
  "SourceIp": "34.230.82.196",
  "TraceId": "|e3628548-42e164ee4d1379bf.",
  "Path": "/home",
  "OneBox": false,
  "Controller": "Home",
  "Action": "Index",
  "Region": "us-east-1",
  "AZ-ID": "use1-az2",
  "InstanceId": "i-01ab0b7241214d494",
  "LogGroupName": "/loggroupname",
  "HttpResponseCode": 200,
  "2xx": 1,
  "3xx": 0,
  "4xx": 0,
  "5xx": 0,
  "SuccessLatency": 20
}
```

Log ini memiliki tiga set dimensi. Mereka berkembang dalam urutan perincian, dari instance ke Availability Zone ke Region (`Controller`dan selalu `Action` disertakan dalam contoh ini). Mereka mendukung pembuatan alarm di seluruh beban kerja Anda yang menunjukkan kapan ada dampak pada tindakan pengontrol tertentu dalam satu instance, dalam satu Availability Zone, atau dalam

keseluruhan. Wilayah AWS Dimensi ini digunakan untuk menghitung metrik HTTP respons 2xx, 3xx, 4xx, dan 5xx, serta latensi untuk metrik permintaan yang berhasil (jika permintaan gagal, itu juga akan merekam metrik untuk latensi permintaan yang gagal). Log juga mencatat informasi lain seperti HTTP jalur, IP sumber pemohon, dan apakah permintaan ini memerlukan cache lokal untuk di-refresh. Titik data ini kemudian dapat digunakan untuk menghitung ketersediaan dan latensi dari setiap beban kerja API yang disediakan.

> **Catatan tentang penggunaan kode HTTP respons untuk metrik ketersediaan**
>
> Biasanya, Anda dapat menganggap respons 2xx dan 3xx berhasil, dan 5xx sebagai kegagalan. Kode respons 4xx jatuh di suatu tempat di tengah. Biasanya, mereka diproduksi karena kesalahan klien. Mungkin parameter berada di luar jangkauan yang mengarah ke respons 400, atau mereka meminta sesuatu yang tidak ada, menghasilkan respons 404. Anda tidak akan menghitung tanggapan ini terhadap ketersediaan beban kerja Anda. Namun, ini juga bisa menjadi hasil dari bug dalam perangkat lunak.
> Misalnya, jika Anda telah memperkenalkan validasi masukan yang lebih ketat yang menolak permintaan yang akan berhasil sebelumnya, respons 400 mungkin dihitung sebagai penurunan ketersediaan. Atau mungkin Anda membatasi pelanggan dan mengembalikan respons 429. Sementara membatasi pelanggan melindungi layanan Anda untuk mempertahankan ketersediaannya, dari perspektif pelanggan, layanan tidak tersedia untuk memproses permintaan mereka. Anda harus memutuskan apakah kode respons 4xx adalah bagian dari perhitungan ketersediaan Anda atau tidak.

Meskipun bagian ini telah menguraikan penggunaan CloudWatch sebagai cara untuk mengumpulkan dan menganalisis metrik, itu bukan satu-satunya solusi yang dapat Anda gunakan. Anda dapat memilih untuk juga mengirim metrik ke Amazon Managed Service untuk Prometheus dan Amazon Managed Grafana, tabel Amazon DynamoDB, atau menggunakan solusi pemantauan pihak ketiga. Kuncinya adalah metrik yang dihasilkan oleh beban kerja Anda harus berisi konteks tentang batas isolasi kesalahan beban kerja Anda.

Dengan beban kerja yang menghasilkan metrik dengan dimensi yang disejajarkan dengan batas isolasi kesalahan, Anda dapat membuat observabilitas yang mendeteksi kegagalan terisolasi Availability Zone. Bagian berikut menjelaskan tiga pendekatan gratis untuk mendeteksi kegagalan yang timbul dari penurunan satu Availability Zone.

Topik

- Deteksi kegagalan dengan CloudWatch alarm komposit

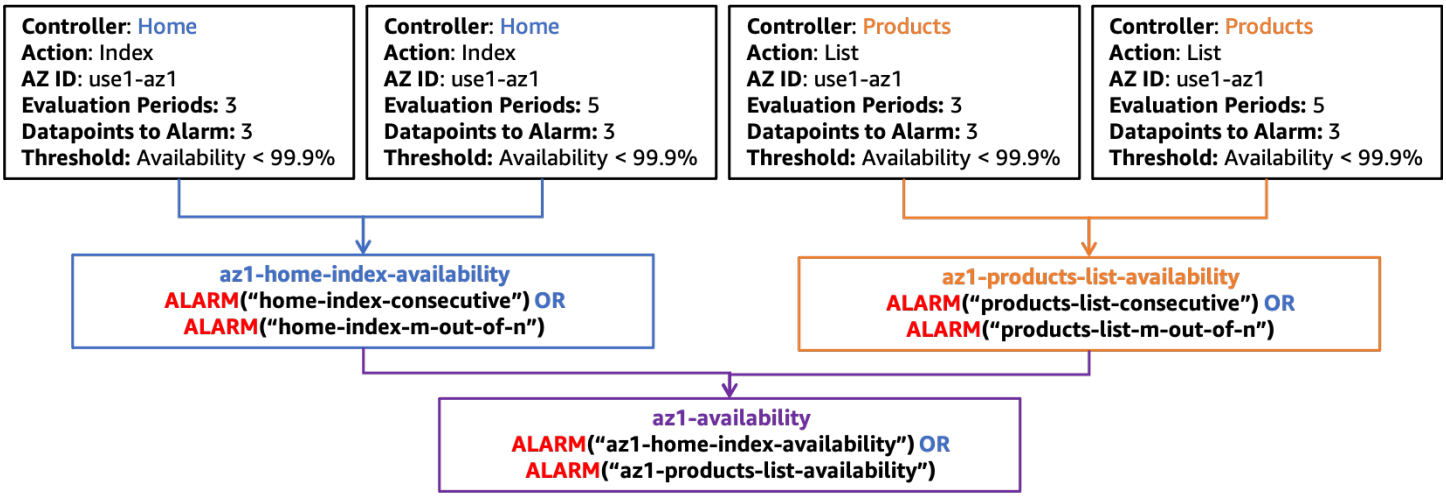
Deteksi kegagalan dengan CloudWatch alarm komposit

Dalam CloudWatch metrik, setiap set dimensi adalah metrik unik, dan Anda dapat membuat CloudWatch alarm pada masing-masing metrik. Anda kemudian dapat membuat alarm CloudWatch komposit Amazon untuk menggabungkan metrik ini.

Untuk mendeteksi dampak secara akurat, contoh-contoh dalam paper ini akan menggunakan dua struktur CloudWatch alarm yang berbeda untuk setiap dimensi yang disetel alarm. Setiap alarm akan menggunakan Periode satu menit, yang berarti metrik dievaluasi sekali per menit. Pendekatan pertama akan menggunakan tiga titik data pelanggaran berturut-turut dengan menetapkan Periode Evaluasi dan Titik Data ke Alarm menjadi tiga, yang berarti dampak total tiga menit. Pendekatan kedua akan menggunakan “M out of N” ketika ada 3 titik data dalam jendela lima menit yang dilanggar dengan mengatur Periode Evaluasi menjadi lima dan Datapoint ke Alarm menjadi tiga. Ini memberikan kemampuan untuk mendeteksi sinyal konstan, serta sinyal yang berfluktuasi dalam waktu singkat. Durasi waktu dan jumlah titik data yang terkandung di sini adalah saran, gunakan nilai yang masuk akal untuk beban kerja Anda.

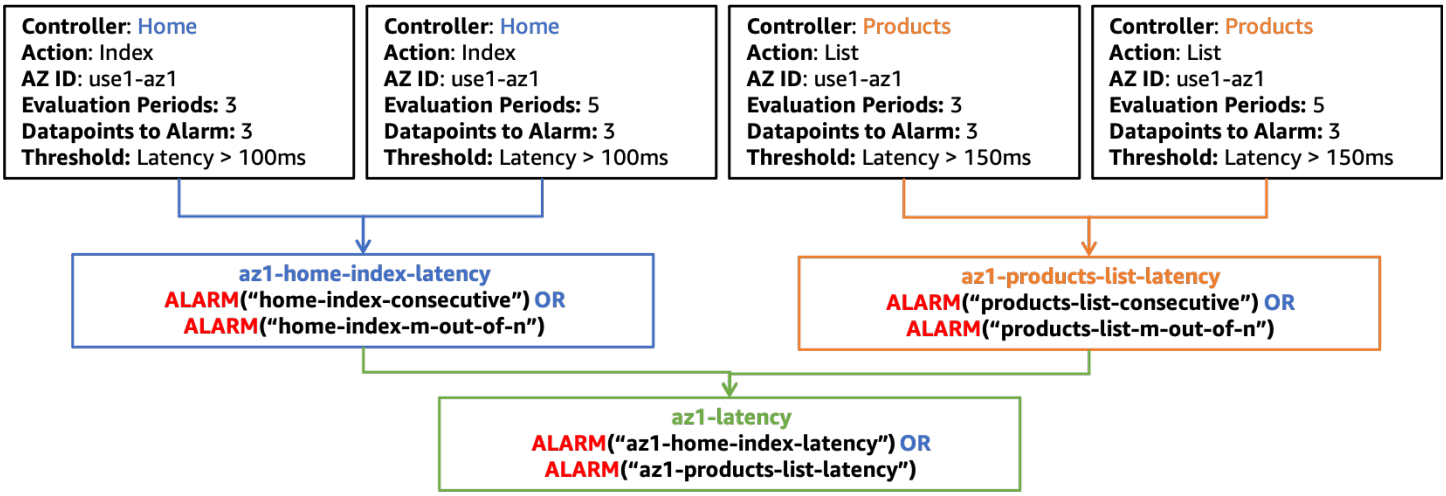
Mendeteksi dampak dalam satu Availability Zone

Dengan menggunakan konstruksi ini, pertimbangkan beban kerja yang menggunakan`Controller`,,, `Action` `InstanceIdAZ-ID`, dan `Region` sebagai dimensi. Beban kerja memiliki dua pengontrol, Produk dan Rumah, dan satu tindakan per pengontrol, Daftar dan Indeks masing-masing. Ini beroperasi di tiga Availability Zone di `us-east-1` Wilayah. Anda akan membuat dua alarm untuk ketersediaan untuk masing-masing `Controller` dan `Action` kombinasi di setiap Availability Zone serta dua alarm untuk latensi untuk masing-masing. Kemudian, Anda dapat memilih untuk membuat alarm komposit untuk ketersediaan untuk masing-masing `Controller` dan `Action` kombinasi. Terakhir, Anda membuat alarm komposit yang menggabungkan semua alarm ketersediaan untuk Availability Zone. Ini ditunjukkan pada gambar berikut untuk Availability Zone tunggal,`use1-az1`, menggunakan alarm komposit opsional untuk masing-masing `Controller` dan `Action` kombinasi (alarm serupa akan ada untuk `use1-az2` dan `use1-az3` Availability Zones juga, tetapi tidak ditampilkan untuk kesederhanaan).



Struktur alarm komposit untuk ketersediaan di *use1-az1*

Anda juga akan membangun struktur alarm serupa untuk latensi juga, yang ditunjukkan pada gambar berikutnya.

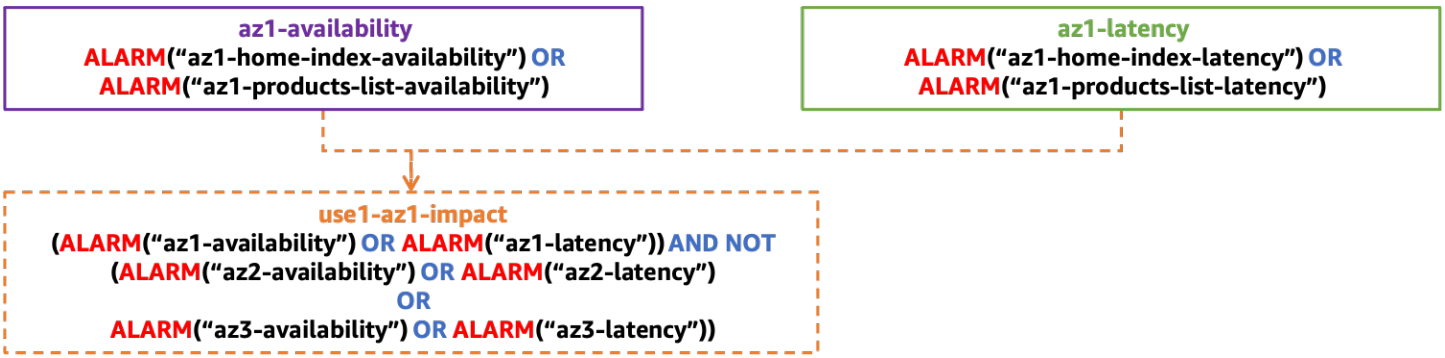


Struktur alarm komposit untuk latensi di *use1-az1*

Untuk sisa angka di bagian ini, hanya alarm `az1-availability` dan `az1-latency` komposit yang akan ditampilkan di tingkat atas. Alarm komposit ini, `az1-availability` dan`az1-latency`, akan memberi tahu Anda jika ketersediaan turun di bawah atau latensi naik di atas ambang batas yang ditentukan di Zona Ketersediaan tertentu untuk bagian mana pun dari beban kerja Anda. Anda mungkin juga ingin mempertimbangkan untuk mengukur throughput untuk mendeteksi dampak yang mencegah beban kerja Anda di satu Availability Zone menerima pekerjaan. Anda dapat mengintegrasikan alarm yang dihasilkan dari metrik yang dipancarkan oleh kenari Anda ke dalam alarm komposit ini juga. Dengan begitu, jika sisi server atau sisi klien melihat dampak ketersediaan atau latensi, alarm akan membuat peringatan.

Pastikan dampaknya tidak Regional

Satu set alarm komposit lainnya dapat digunakan untuk memastikan bahwa hanya peristiwa Availability Zone yang terisolasi yang menyebabkan alarm diaktifkan. Ini dilakukan dengan memastikan bahwa alarm komposit Availability Zone berada dalam `ALARM` status sementara alarm komposit untuk Availability Zone lainnya berada dalam `OK` status. Ini akan menghasilkan satu alarm komposit per Availability Zone yang Anda gunakan. Contoh ditunjukkan pada gambar berikut (ingat bahwa ada alarm untuk latensi dan ketersediaan di `use1-az2` dan`use1-az3`,,,`az2-latency`, dan `az2-availability` `az3-latency`yaz3-availability, yang tidak digambarkan untuk kesederhanaan).



Struktur alarm komposit untuk mendeteksi dampak yang diisolasi ke satu AZ

Pastikan dampaknya tidak disebabkan oleh satu contoh

Satu instance (atau sebagian kecil dari keseluruhan armada Anda) dapat menyebabkan dampak yang tidak proporsional terhadap ketersediaan dan metrik latensi yang dapat membuat seluruh Availability Zone tampak terpengaruh, padahal sebenarnya tidak. Lebih cepat dan sama efektifnya untuk menghapus satu instance bermasalah daripada mengevakuasi Availability Zone.

Instans dan kontainer biasanya diperlakukan sebagai sumber daya sementara, sering diganti dengan layanan seperti. AWS Auto Scaling Sulit untuk membuat CloudWatch alarm baru setiap kali instance baru dibuat (tetapi tentu saja mungkin menggunakan kait siklus hidup Amazon EventBridge atau Amazon EC2 Auto Scaling). Sebagai gantinya, Anda dapat menggunakan CloudWatch Contributor Insights untuk mengidentifikasi jumlah kontributor terhadap ketersediaan dan metrik latensi.

Sebagai contoh, untuk aplikasi HTTP web, Anda dapat membuat aturan untuk mengidentifikasi kontributor teratas untuk HTTP tanggapan 5xx di setiap Availability Zone. Ini akan mengidentifikasi instance mana yang berkontribusi terhadap penurunan ketersediaan (metrik ketersediaan kami yang ditentukan di atas didorong oleh adanya kesalahan 5xx). Dengan menggunakan contoh

EMF log, buat aturan menggunakan kunci dari`InstanceId`. Kemudian, filter log berdasarkan `HttpResponseCode` bidang. Contoh ini adalah aturan untuk `use1-az1` Availability Zone.

```
{
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [
            {
                "Match": "$.InstanceId",
                "IsPresent": true
            },
            {
                "Match": "$.HttpStatusCode",
                "IsPresent": true
            },
            {
                "Match": "$.HttpStatusCode",
                "GreaterThan": 499
            },
            {
                "Match": "$.HttpStatusCode",
                "LessThan": 600
            },
            {
                "Match": "$.AZ-ID",
                "In": ["use1-az1"]
            },
        ],
        "Keys": [
            "$.InstanceId"
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/loggroupname"
    ],
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    }
}
```

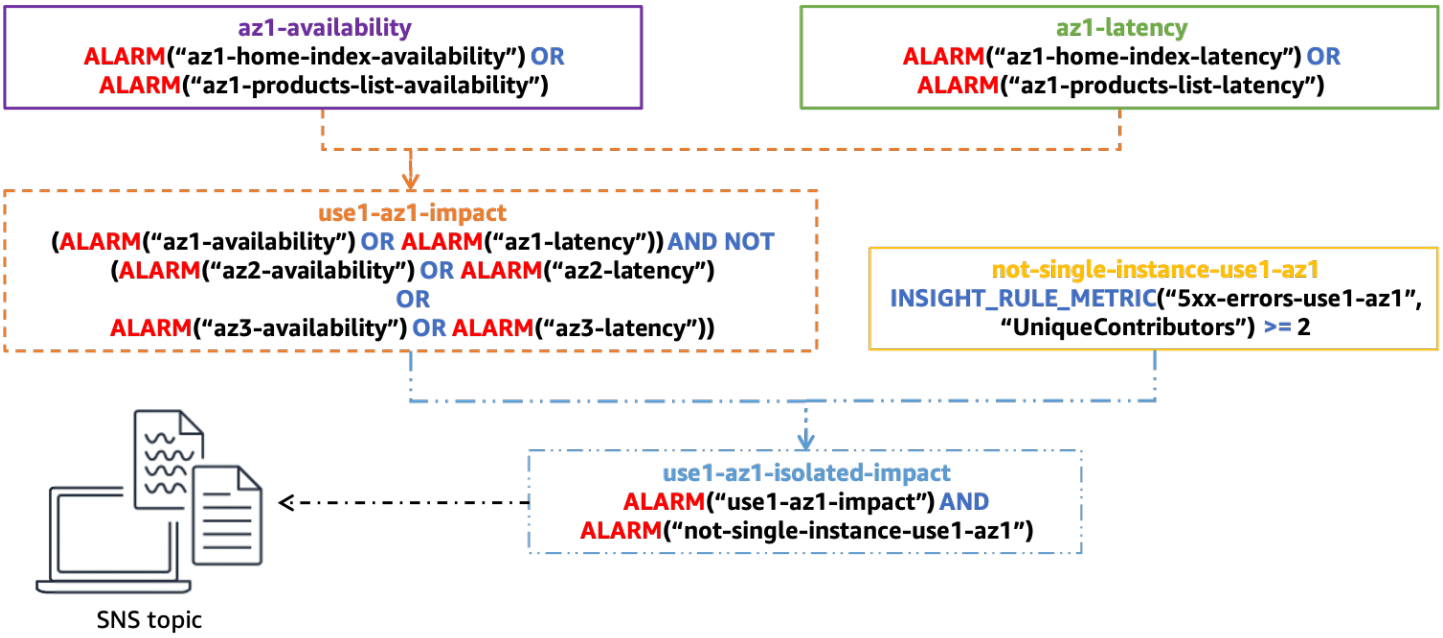
CloudWatch Alarm dapat dibuat berdasarkan aturan-aturan ini juga. Anda dapat membuat alarm berdasarkan aturan Contributor Insights menggunakan matematika metrik dan `INSIGHT_RULE_METRIC` fungsi dengan metrik. `UniqueContributors` Anda juga dapat membuat aturan Contributor Insights tambahan dengan CloudWatch alarm untuk metrik seperti latensi atau jumlah kesalahan selain yang tersedia. Alarm ini dapat digunakan dengan alarm komposit dampak Zona Ketersediaan yang terisolasi untuk memastikan bahwa satu instans tidak mengaktifkan alarm. Metrik untuk aturan wawasan `use1-az1` mungkin terlihat seperti berikut:

```
INSIGHT_RULE_METRIC("5xx-errors-use1-az1", "UniqueContributors")
```

Anda dapat menentukan alarm ketika metrik ini lebih besar dari ambang batas; untuk contoh ini, dua. Ini diaktifkan ketika kontributor unik untuk respons 5xx melampaui ambang batas itu, menunjukkan dampaknya berasal dari lebih dari dua contoh. Alasan alarm ini menggunakan perbandingan yang lebih besar daripada kurang adalah untuk memastikan bahwa nilai nol untuk kontributor unik tidak memicu alarm. Ini memberi tahu Anda bahwa dampaknya bukan dari satu contoh. Sesuaikan ambang batas ini untuk beban kerja pribadi Anda. Panduan umum adalah membuat angka ini 5% atau lebih dari total sumber daya di Availability Zone. Lebih dari 5% sumber daya Anda yang terpengaruh menunjukkan signifikansi statistik, mengingat ukuran sampel yang cukup.

Menyatukan semuanya

Gambar berikut menunjukkan struktur alarm komposit lengkap untuk Availability Zone tunggal:



Struktur alarm komposit lengkap untuk menentukan dampak Single-AZ

Alarm komposit terakhir`use1-az1-isolated-impact`,, diaktifkan ketika alarm komposit yang menunjukkan dampak Zona Ketersediaan terisolasi dari latensi atau ketersediaan`use1-az1-aggregate-alarm`,, dalam `ALARM` keadaan dan saat alarm berdasarkan aturan Wawasan Kontributor untuk Zona Ketersediaan yang sama`not-single-instance-use1-az1`,, juga dalam `ALARM` keadaan (artinya dampaknya lebih dari satu instance). Anda akan membuat tumpukan alarm ini untuk setiap Availability Zone yang digunakan oleh beban kerja Anda.

Anda dapat melampirkan peringatan Amazon Simple Notification Service (AmazonSNS) ke alarm terakhir ini. Semua alarm sebelumnya dikonfigurasi tanpa tindakan. Peringatan dapat memberi tahu operator melalui email untuk memulai penyelidikan manual. Itu juga bisa memulai otomatisasi untuk mengevakuasi Availability Zone. Namun, kata hati-hati dalam membangun otomatisasi untuk menanggapi peringatan ini. Setelah evakuasi Availability Zone terjadi, hasilnya adalah bahwa tingkat kesalahan yang meningkat dikurangi dan alarm kembali ke keadaan. `OK` Jika dampak terjadi di Availability Zone lain, ada kemungkinan bahwa otomatisasi dapat mengevakuasi Availability Zone kedua atau ketiga, berpotensi menghapus semua kapasitas beban kerja yang tersedia. Otomatisasi harus memeriksa untuk melihat apakah evakuasi telah dilakukan sebelum mengambil tindakan apa pun. Anda mungkin juga perlu menskalakan sumber daya di Availability Zone lainnya sebelum evakuasi berhasil.

Saat Anda menambahkan pengontrol atau tindakan baru ke aplikasi MVC web Anda, atau layanan mikro baru, atau secara umum, fungsionalitas tambahan apa pun yang ingin Anda pantau secara terpisah, Anda hanya perlu memodifikasi beberapa alarm dalam pengaturan ini. Anda akan membuat alarm ketersediaan dan latensi baru untuk fungsionalitas baru itu dan kemudian menambahkannya ke ketersediaan selaras Availability Zone dan alarm komposit latensi, `az1-latency` dan `az1-availability` dalam contoh yang telah kami gunakan di sini. Alarm komposit yang tersisa tetap statis setelah dikonfigurasi. Ini membuat orientasi fungsionalitas baru dengan pendekatan ini menjadi proses yang lebih sederhana.

Deteksi kegagalan menggunakan deteksi outlier

Satu celah dengan pendekatan sebelumnya dapat muncul ketika Anda melihat tingkat kesalahan yang meningkat di beberapa Availability Zone yang terjadi karena alasan yang tidak berkorelasi. Bayangkan sebuah skenario di mana Anda memiliki EC2 instance yang diterapkan di tiga Availability Zone dan ambang batas alarm ketersediaan Anda adalah 99%. Kemudian, kerusakan Availability Zone tunggal terjadi, mengisolasi banyak instance dan menyebabkan ketersediaan di zona tersebut turun menjadi 55%. Pada saat yang sama, tetapi di Availability Zone yang berbeda, satu EC2 instance menghabiskan semua penyimpanan pada EBS volumenya, dan tidak dapat lagi

menulis file log. Ini menyebabkannya mulai mengembalikan kesalahan, tetapi masih melewati pemeriksaan kesehatan penyeimbang beban karena itu tidak memicu file log untuk ditulis. Hal ini mengakibatkan ketersediaan turun menjadi 98% di Availability Zone tersebut. Dalam hal ini, alarm dampak Availability Zone tunggal Anda tidak akan aktif karena Anda melihat dampak ketersediaan di beberapa Availability Zone. Namun, Anda masih bisa mengurangi hampir semua dampak dengan mengevakuasi Zona Ketersediaan yang terganggu.

Di beberapa jenis beban kerja, Anda mungkin mengalami kesalahan secara konsisten di semua Availability Zone yang metrik ketersediaan sebelumnya mungkin tidak berguna. Ambil AWS Lambda contoh. AWS memungkinkan pelanggan untuk membuat kode mereka sendiri untuk dijalankan dalam fungsi Lambda. Untuk menggunakan layanan ini, Anda harus mengunggah kode Anda dalam ZIP file, termasuk dependensi, dan menentukan titik masuk ke fungsi tersebut. Tetapi kadang-kadang pelanggan mendapatkan bagian ini salah, misalnya, mereka mungkin lupa ketergantungan kritis dalam ZIP file, atau salah ketik nama metode dalam definisi fungsi Lambda. Hal ini menyebabkan fungsi gagal untuk dipanggil dan menghasilkan kesalahan. AWS Lambda melihat kesalahan semacam ini sepanjang waktu, tetapi itu tidak menunjukkan bahwa ada sesuatu yang tidak sehat. Namun, sesuatu seperti kerusakan Availability Zone juga dapat menyebabkan kesalahan ini muncul.

Untuk menemukan sinyal dalam jenis noise ini, Anda dapat menggunakan deteksi outlier untuk menentukan apakah ada kemiringan yang signifikan secara statistik dalam jumlah kesalahan di antara Availability Zones. Meskipun kami melihat kesalahan di beberapa Availability Zone, jika benar-benar ada kegagalan di salah satunya, kami berharap untuk melihat tingkat kesalahan yang jauh lebih tinggi di Availability Zone tersebut dibandingkan dengan yang lain, atau berpotensi jauh lebih rendah. Tapi seberapa tinggi atau lebih rendah?

Salah satu cara untuk melakukan analisis ini adalah dengan menggunakan uji chi-kuadrat (χ^2) untuk mendeteksi perbedaan yang signifikan secara statistik dalam tingkat kesalahan antara Availability Zones (ada banyak algoritma berbeda untuk melakukan deteksi outlier). Mari kita lihat bagaimana tes chi-kuadrat bekerja.

Tes chi-kuadrat mengevaluasi probabilitas bahwa beberapa distribusi hasil mungkin terjadi. Dalam hal ini, kami tertarik pada distribusi kesalahan di beberapa set yang ditentukanAZs. Untuk contoh ini, untuk mempermudah matematika, pertimbangkan empat Availability Zones.

Pertama, buat hipotesis nol, yang mendefinisikan apa yang Anda yakini sebagai hasil default. Dalam pengujian ini, hipotesis nol adalah bahwa Anda mengharapkan kesalahan didistribusikan secara merata di setiap Availability Zone. Kemudian, hasilkan hipotesis alternatif, yaitu bahwa kesalahan tidak terdistribusi secara merata yang menunjukkan penurunan Zona Ketersediaan. Sekarang

Anda dapat menguji hipotesis ini menggunakan data dari metrik Anda. Untuk tujuan ini, Anda akan mengambil sampel metrik Anda dari jendela lima menit. Misalkan Anda mendapatkan 1000 titik data yang diterbitkan di jendela itu, di mana Anda melihat 100 kesalahan total. Anda berharap bahwa dengan distribusi genap kesalahan akan terjadi 25% dari waktu di masing-masing dari empat Availability Zone. Asumsikan tabel berikut menunjukkan apa yang Anda harapkan dibandingkan dengan apa yang sebenarnya Anda lihat.

Tabel 1: Kesalahan yang diharapkan versus aktual terlihat

AZ	Expected	Aktual
`use1-az1`	25	20
`use1-az2`	25	20
`use1-az3`	25	25
`use1-az4`	25	35

Jadi, Anda melihat bahwa distribusi pada kenyataannya tidak genap. Namun, Anda mungkin percaya bahwa ini terjadi karena beberapa tingkat keacakan dalam titik data yang Anda sampel. Ada beberapa tingkat probabilitas bahwa jenis distribusi ini dapat terjadi dalam set sampel dan masih berasumsi bahwa hipotesis nol benar. Ini mengarah pada pertanyaan berikut: Berapa probabilitas mendapatkan hasil setidaknya ekstrem ini? Jika probabilitas itu di bawah ambang batas yang ditentukan, Anda menolak hipotesis nol. Agar signifikan secara statistik, probabilitas ini harus 5% atau kurang. [1]

[1] Craparo, Robert M. (2007). “Tingkat signifikansi”. Dalam Salkind, Neil J. Ensiklopedia Pengukuran dan Statistik 3. Thousand Oaks, CA: SAGE Publikasi. hal.889—891. ISBN1-412-91611-9.

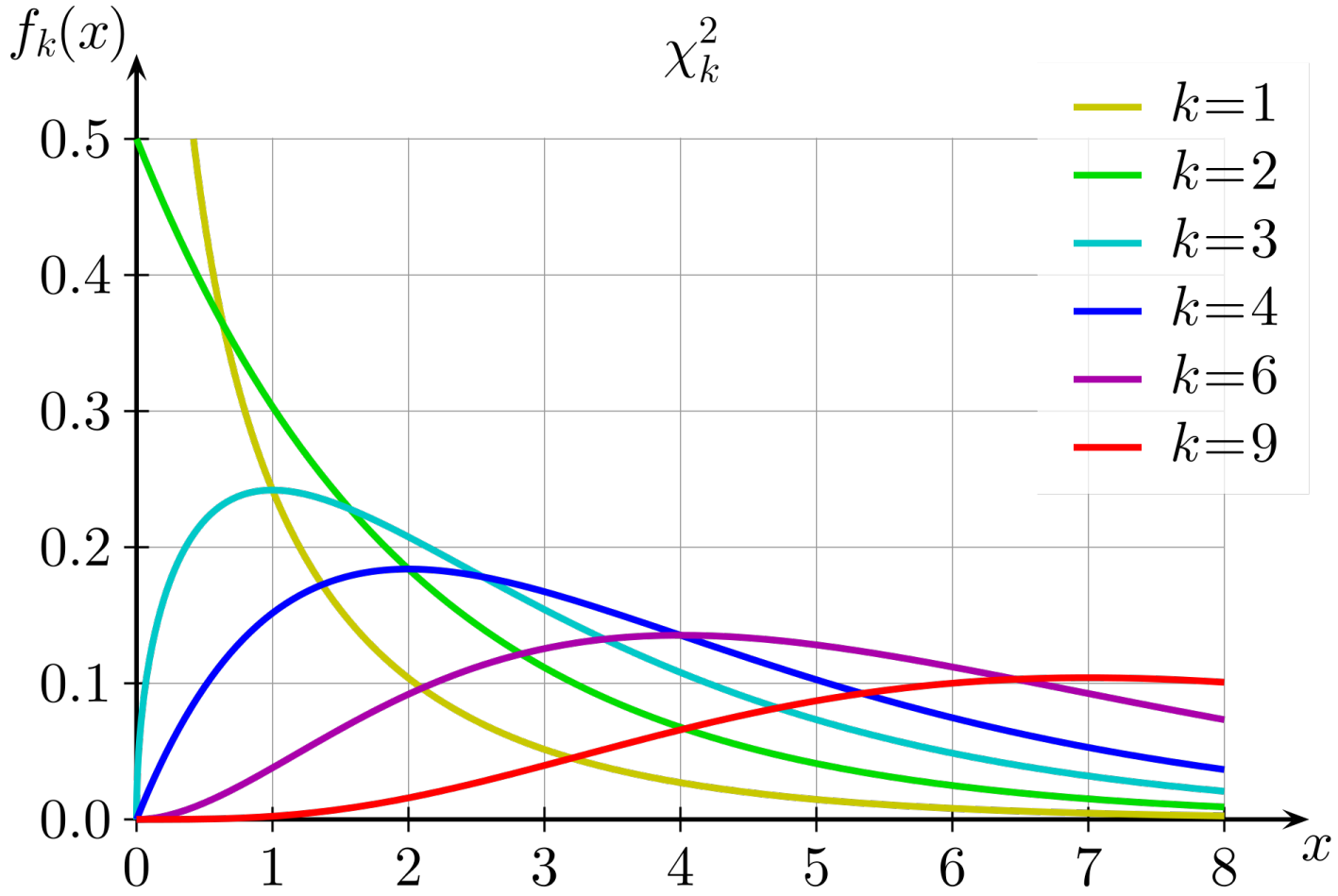
Bagaimana Anda menghitung probabilitas hasil ini? Anda menggunakan statistik χ^2 yang memberikan distribusi yang dipelajari dengan sangat baik dan dapat digunakan untuk menentukan probabilitas mendapatkan hasil yang ekstrem atau lebih ekstrem ini menggunakan rumus ini.

$$E_i = expected\ observations\ of\ type\ i$$
$$O_i = actual\ observations\ of\ type\ i \quad (1)$$
$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Sebagai contoh kami, ini menghasilkan:

$$\chi^2 = \frac{(20-25)^2}{25} + \frac{(20-25)^2}{25} + \frac{(25-25)^2}{25} + \frac{(35-25)^2}{25}$$
$$\chi^2 = \frac{-5^2}{25} + \frac{-5^2}{25} + \frac{0^2}{25} + \frac{10^2}{25} \quad (2)$$
$$\chi^2 = 1 + 1 + 0 + 4$$
$$\chi^2 = 6$$

Jadi, apa 6 artinya dalam hal probabilitas kita? Anda perlu melihat distribusi chi-kuadrat dengan tingkat kebebasan yang sesuai. Gambar berikut menunjukkan beberapa distribusi chi-kuadrat untuk tingkat kebebasan yang berbeda.



Distribusi Chi-kuadrat untuk berbagai tingkat kebebasan

Tingkat kebebasan dihitung sebagai satu kurang dari jumlah pilihan dalam tes. Dalam hal ini, karena ada empat Availability Zone, tingkat kebebasannya adalah tiga. Kemudian, Anda ingin mengetahui luas di bawah kurva (integral) untuk x ≥ 6 pada plot k = 3. Anda juga dapat menggunakan tabel pra-perhitungan dengan nilai yang umum digunakan untuk memperkirakan nilai tersebut.

Tabel 2: Nilai kritis Chi-kuadrat

Derajat kebebasan	Probabilitas kurang dari nilai kritis				
	0,75	0,90	0,95	0,99	0,999
1	1.323	2.706	3.841	6.635	10.828
2	2.773	4.605	5.991	9.210	13.816

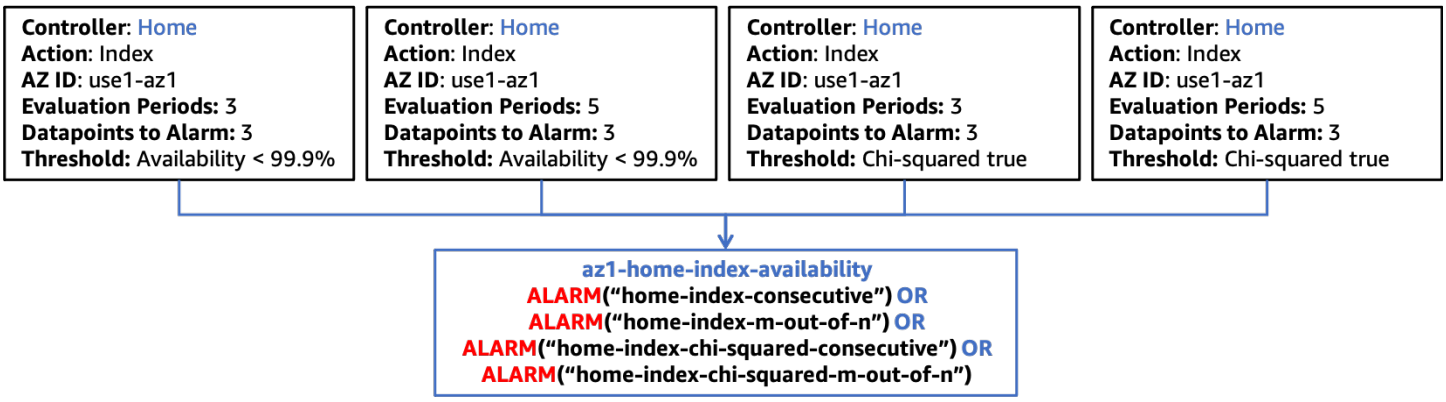
Derajat kebebasan	Probabilitas kurang dari nilai kritis				
	0,75	0,90	0,95	0,99	0,999
3	4.108	6.251	7.815	11.345	16.266
4	5.385	7.779	9.488	13.277	18.467

Untuk tiga derajat kebebasan, nilai chi-kuadrat enam jatuh di antara kolom probabilitas 0,75 dan 0,9. Apa artinya ini adalah ada kemungkinan lebih besar dari 10% dari distribusi ini terjadi, yang tidak kurang dari ambang batas 5%. Oleh karena itu, Anda menerima hipotesis nol dan menentukan tidak ada perbedaan yang signifikan secara statistik dalam tingkat kesalahan di antara Availability Zones.

Melakukan tes statistik chi-kuadrat tidak didukung secara native dalam matematika CloudWatch metrik, jadi Anda perlu mengumpulkan metrik kesalahan yang berlaku dari CloudWatch dan menjalankan pengujian di lingkungan komputasi seperti Lambda. Anda dapat memutuskan untuk melakukan tes ini pada sesuatu seperti MVC Controller/Action atau tingkat layanan mikro individu, atau di tingkat Availability Zone. Anda harus mempertimbangkan apakah penurunan Availability Zone akan memengaruhi setiap Controller/Action atau microservice secara setara, atau apakah sesuatu seperti DNS kegagalan dapat menyebabkan dampak pada layanan throughput rendah dan bukan pada layanan throughput tinggi, yang dapat menutupi dampak saat digabungkan. Dalam kedua kasus, pilih dimensi yang sesuai untuk membuat kueri. Tingkat granularitas juga akan memengaruhi CloudWatch alarm yang dihasilkan yang Anda buat.

Kumpulkan metrik hitungan kesalahan untuk setiap AZ dan Controller/Action di jendela waktu yang ditentukan. Pertama, hitung hasil uji chi-kuadrat sebagai benar (ada kemiringan yang signifikan secara statistik) atau salah (tidak ada, yaitu hipotesis nol berlaku). Jika hasilnya salah, publikasikan titik data 0 ke aliran metrik Anda untuk hasil chi-kuadrat untuk setiap Availability Zone. Jika hasilnya benar, publikasikan 1 titik data untuk Availability Zone dengan kesalahan terjauh dari nilai yang diharapkan dan 0 untuk yang lain (lihat Lampiran B - Contoh perhitungan chi-kuadrat untuk kode contoh yang dapat digunakan dalam fungsi Lambda). Anda dapat mengikuti pendekatan yang sama seperti alarm ketersediaan sebelumnya dengan menggunakan membuat alarm CloudWatch metrik 3 berturut-turut dan alarm CloudWatch metrik 3 dari 5 berdasarkan titik data yang dihasilkan oleh fungsi Lambda. Seperti pada contoh sebelumnya, pendekatan ini dapat dimodifikasi untuk menggunakan lebih banyak atau lebih sedikit titik data dalam jendela yang lebih pendek atau lebih panjang.

Kemudian, tambahkan alarm ini ke alarm ketersediaan Availability Zone yang ada untuk kombinasi Controller dan Action, yang ditunjukkan pada gambar berikut.



Mengintegrasikan uji statistik chi-kuadrat dengan alarm komposit

Seperti disebutkan sebelumnya, saat Anda memasukkan fungsionalitas baru di beban kerja Anda, Anda hanya perlu membuat alarm CloudWatch metrik yang sesuai yang khusus untuk fungsionalitas baru tersebut dan memperbarui tingkat berikutnya dalam hierarki alarm komposit untuk menyertakan alarm tersebut. Sisa struktur alarm tetap statis.

Deteksi kegagalan sumber daya zona instance tunggal

Dalam beberapa kasus, Anda mungkin memiliki satu instance aktif dari sumber daya zona, paling umum sistem yang memerlukan komponen penulis tunggal seperti database relasional (seperti AmazonRDS) atau cache terdistribusi (seperti Amazon ElastiCache (OSSRedis)). Jika satu kerusakan Availability Zone memengaruhi Availability Zone tempat sumber daya utama berada, hal itu dapat berdampak pada setiap Availability Zone yang mengakses sumber daya tersebut. Hal ini dapat menyebabkan ambang batas ketersediaan dilintasi di setiap Availability Zone, yang berarti pendekatan pertama tidak akan mengidentifikasi dengan benar sumber dampak Availability Zone tunggal. Selain itu, Anda mungkin akan melihat tingkat kesalahan serupa di setiap Availability Zone, yang berarti analisis outlier juga tidak akan mendeteksi masalah. Artinya, Anda perlu menerapkan observabilitas tambahan untuk mendeteksi skenario ini secara khusus.

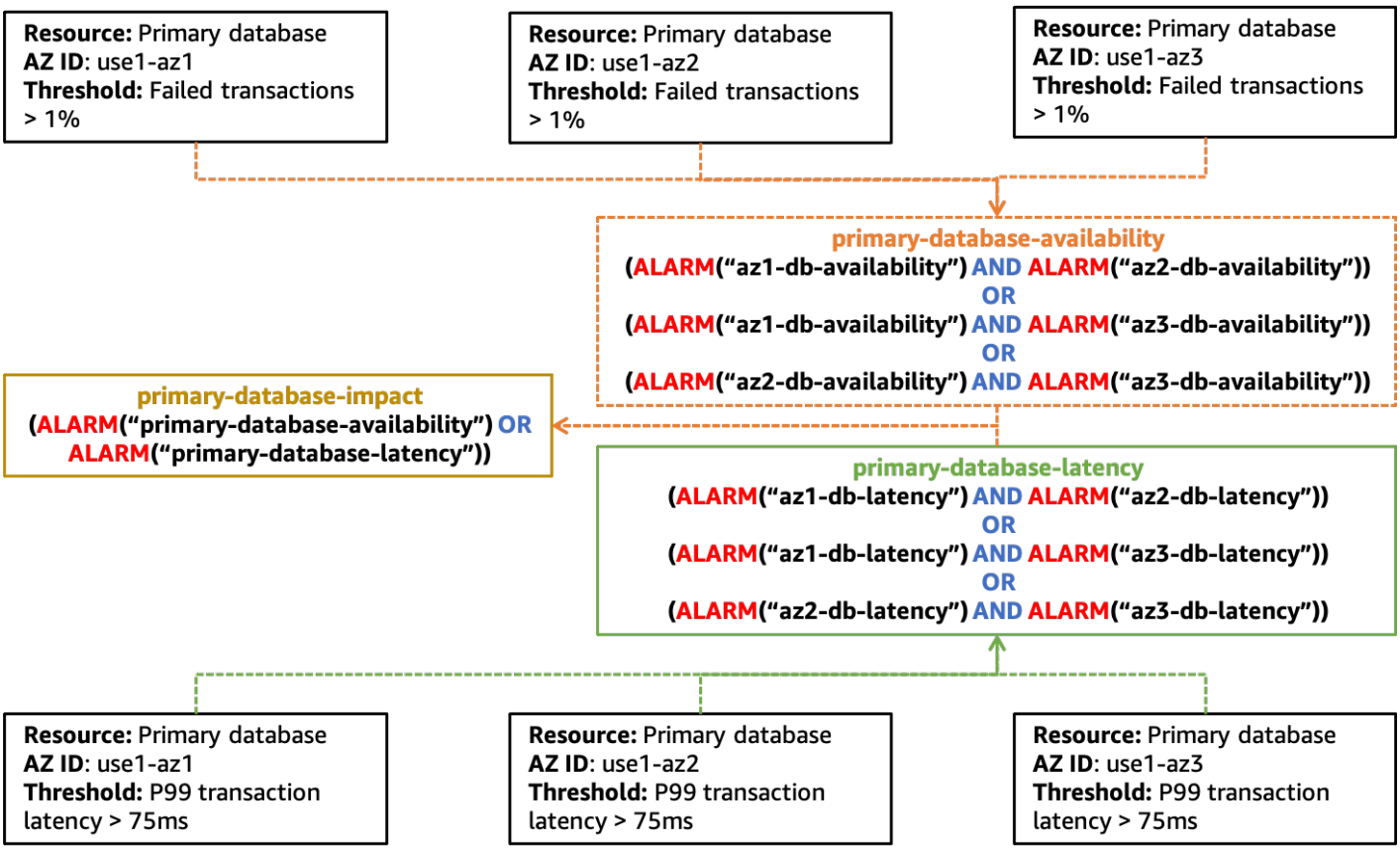
Kemungkinan sumber daya yang Anda khawatirkan akan menghasilkan metriknya sendiri tentang kesehatannya, tetapi selama penurunan Zona Ketersediaan, sumber daya tersebut mungkin tidak dapat memberikan metrik tersebut. Dalam skenario ini, Anda harus membuat atau memperbarui alarm untuk mengetahui kapan Anda terbang buta. Jika ada metrik penting yang sudah Anda pantau dan alarm aktif, Anda dapat mengonfigurasi alarm untuk memperlakukan data yang hilang sebagai pelanggaran. Ini akan membantu Anda mengetahui apakah sumber daya berhenti melaporkan data,

dan dapat dimasukkan dalam hal yang sama dalam satu baris dan m dari n alarm yang digunakan sebelumnya.

Mungkin juga dalam beberapa metrik yang menunjukkan kesehatan sumber daya yang menerbitkan titik data bernilai nol ketika tidak ada aktivitas. Jika gangguan mencegah interaksi dengan sumber daya, Anda tidak dapat menggunakan pendekatan data yang hilang untuk jenis metrik ini. Anda juga mungkin tidak ingin khawatir nilainya nol, karena mungkin ada skenario yang sah di mana itu berada dalam ambang batas normal. Pendekatan terbaik untuk mendeteksi jenis masalah ini adalah dengan metrik yang dihasilkan oleh sumber daya menggunakan ketergantungan ini. Dalam hal ini kami ingin mendeteksi dampak di beberapa Availability Zone menggunakan alarm komposit. Alarm ini harus menggunakan beberapa kategori metrik penting yang terkait dengan sumber daya. Beberapa contoh tercantum di bawah ini:

- Throughput — Tingkat unit kerja yang masuk. Ini bisa berupa transaksi, membaca, menulis, dan sebagainya.
- Ketersediaan — Ukur jumlah unit kerja yang berhasil vs gagal.
- Latensi — Ukur beberapa persentil latensi untuk pekerjaan yang berhasil dilakukan di seluruh operasi kritis.

  Sekali lagi, Anda dapat membuat alarm metrik dalam baris dan m dari n untuk setiap metrik di setiap kategori metrik yang ingin Anda ukur. Seperti sebelumnya, ini dapat digabungkan menjadi alarm komposit untuk menentukan bahwa sumber daya bersama ini adalah sumber dampak di seluruh Availability Zone. Anda ingin dapat mengidentifikasi dampak ke lebih dari satu Availability Zone dengan alarm komposit, tetapi dampaknya tidak harus semua Availability Zone. Struktur alarm komposit tingkat tinggi untuk pendekatan semacam ini ditunjukkan pada gambar berikut.



Contoh pembuatan alarm untuk mendeteksi dampak ke beberapa Availability Zone yang disebabkan oleh satu sumber daya

Anda akan melihat bahwa diagram ini kurang preskriptif tentang jenis alarm metrik apa yang harus digunakan dan hierarki alarm komposit. Ini karena menemukan masalah semacam ini bisa sulit dan akan membutuhkan perhatian yang cermat terhadap sinyal yang tepat untuk sumber daya bersama. Sinyal-sinyal tersebut mungkin juga perlu dievaluasi dengan cara tertentu.

Selain itu, Anda harus memperhatikan bahwa `primary-database-impact` alarm tidak terkait dengan Availability Zone tertentu. Hal ini karena instance database utama dapat ditemukan di Availability Zone yang dikonfigurasi untuk digunakan, dan tidak ada CloudWatch metrik yang menentukan di mana itu berada. Ketika Anda melihat alarm ini diaktifkan, Anda harus menggunakannya sebagai sinyal bahwa mungkin ada masalah dengan sumber daya dan memulai failover ke Availability Zone lain, jika belum dilakukan secara otomatis. Setelah memindahkan sumber daya ke Availability Zone lain, Anda dapat menunggu dan melihat apakah alarm Availability Zone terisolasi diaktifkan, atau Anda dapat memilih untuk memanggil rencana evakuasi Availability Zone terlebih dahulu.

Ringkasan

Bagian ini menjelaskan tiga pendekatan untuk membantu mengidentifikasi gangguan Availability Zone tunggal. Setiap pendekatan harus digunakan bersama untuk memberikan pandangan holistik tentang kesehatan beban kerja Anda.

Pendekatan alarm CloudWatch komposit memungkinkan Anda menemukan masalah di mana kemiringan ketersediaan tidak signifikan secara statistik, katakanlah ketersediaan 98% (Zona Ketersediaan yang terganggu), 100%, dan 99,99%, yang tidak disebabkan oleh satu sumber daya bersama.

Deteksi outlier akan membantu mendeteksi gangguan Availability Zone tunggal di mana Anda memiliki kesalahan tidak berkorelasi yang terjadi di beberapa Availability Zone yang semuanya melampaui ambang alarm Anda.

Terakhir, mengidentifikasi degradasi sumber daya zona instance tunggal membantu mengetahui kapan gangguan Availability Zone memengaruhi sumber daya yang dibagikan di seluruh Availability Zone.

Alarm yang dihasilkan dari masing-masing pola ini dapat digabungkan menjadi hierarki alarm CloudWatch komposit untuk mengetahui kapan gangguan Zona Ketersediaan tunggal terjadi dan berdampak pada ketersediaan atau latensi beban kerja Anda.

Pola evakuasi Zona Ketersediaan

Setelah mendeteksi dampak di Availability Zone tunggal, langkah selanjutnya adalah mengevakuasi Availability Zone tersebut. Ada dua hasil yang perlu dicapai evakuasi.

Pertama, Anda ingin berhenti mengirim pekerjaan ke Availability Zone yang terkena dampak. Ini bisa berarti hal yang berbeda dalam arsitektur yang berbeda. Dalam beban kerja permintaan/respons, ini berarti menghentikan hal-hal seperti permintaan HTTP atau gRPC yang datang dari pelanggan Anda yang dikirim ke penyeimbang muatan atau sumber daya lain di Availability Zone. Dalam pemrosesan batch atau sistem pemrosesan antrian, ini bisa berarti menghentikan sumber daya komputasi dari pekerjaan pemrosesan di Availability Zone yang terkena dampak. Anda juga perlu mencegah sumber daya di Availability Zone yang tidak terpengaruh berinteraksi dengan sumber daya di Availability Zone yang terkena dampak, misalnya, instans EC2 yang mengirimkan lalu lintas keantarmuka VPC endpointdi Availability Zone yang terkena dampak atau menghubungkan ke instance utama database.

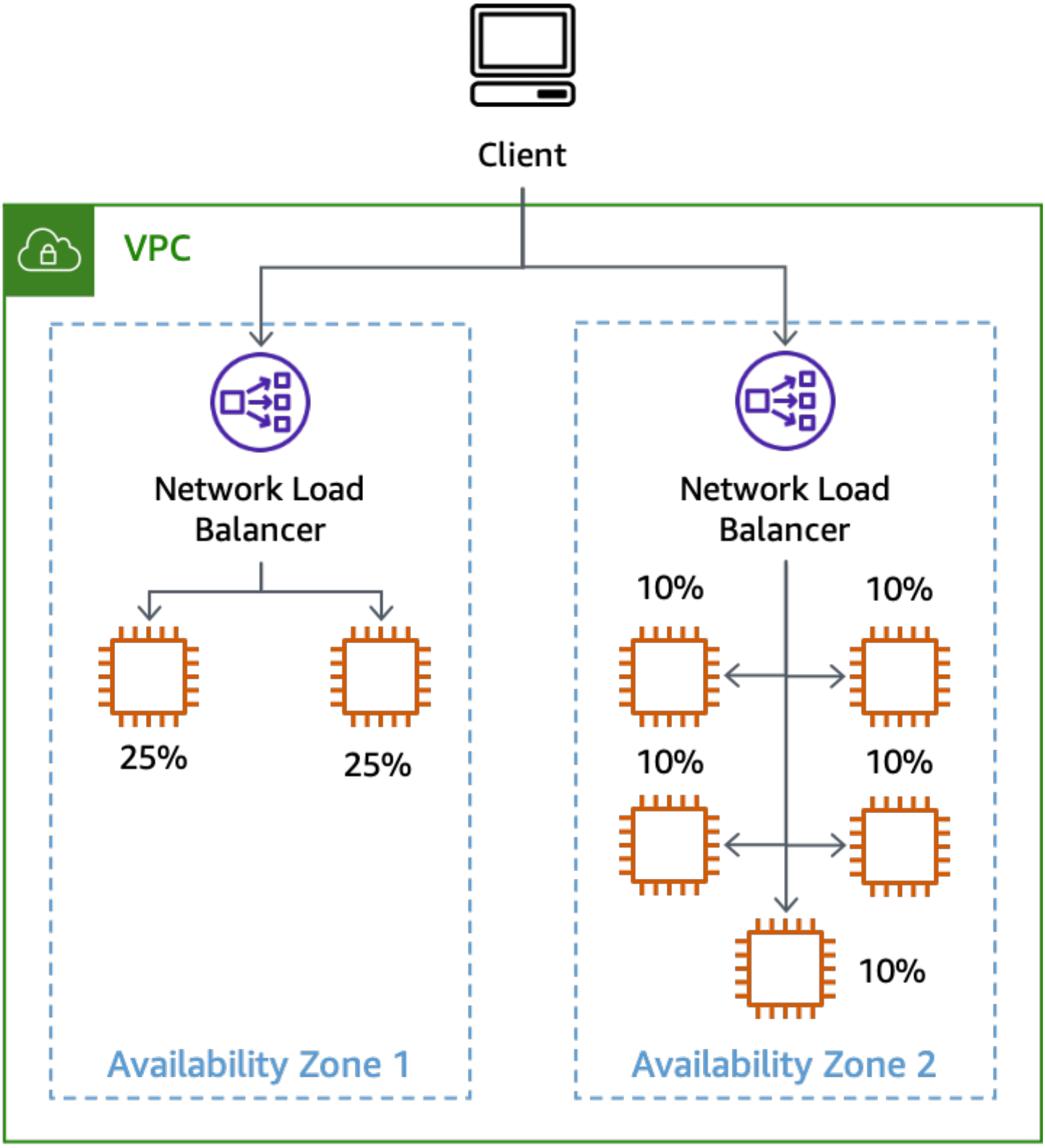
Hasil kedua adalah mencegah kapasitas baru disediakan di Availability Zone yang terkena dampak. Ini penting karena sumber daya baru, seperti instans atau kontainer EC2, yang disediakan di Availability Zone yang terpengaruh kemungkinan akan melihat dampak yang sama dengan sumber daya yang ada. Selain itu, karena hasil pertama mencegah pekerjaan dikirim kepada mereka, mereka tidak dapat menyerap beban yang disediakan untuk ditangani. Hal ini menyebabkan peningkatan beban pada sumber daya yang ada, yang pada akhirnya dapat menyebabkancoklat keluaratau total tidak tersedianya beban kerja. Ada beberapa layanan penskalaan otomatis yang tersedia diAWSdi mana ini berlaku:Penskalaan Otomatis Amazon EC2,Penskalaan Otomatis Aplikasi, danAWS Auto Scaling. Selain itu, layanan seperti Amazon ECS, Amazon EKS, danAWS Batchdapat menjadwalkan pekerjaan pada host di Availability Zone di VPC sebagai bagian dari operasi normal mereka.

Topik

Independensi Zona Ketersediaan

Untuk mencapai hasil pertama, untuk menghentikan pengiriman pekerjaan ke Availability Zone yang terkena dampak, evakuasi mengharuskan Anda menerapkanKetersediaan Zona Kemandirian(AZI), juga kadang-kadang disebutAfinitas Zona Ketersediaan. Pola arsitektur ini mengisolasi sumber daya di dalam Availability Zone dan mencegah interaksi antar sumber daya di Availability Zone yang berbeda kecuali jika benar-benar diperlukan, seperti menghubungkan ke instance database utama di Availability Zone yang berbeda.

Dalam beban kerja tipe permintaan/respons, menerapkan AZI mengharuskan Anda untuknonaktifkan penyeimbangan beban lintas zona untuk Application Load Balancers(ALB),Load Balancers Klasik(CLB), danPenyeimbang Beban Jaringan(NLB) (penyeimbangan beban lintas zona dinonaktifkan secara default untuk NLB). Menonaktifkan load balancing lintas zona memiliki beberapa tradeoff. Saat Anda menonaktifkan penyeimbangan beban lintas zona,lalu lintas dibagi secara merata antara setiap Availability Zoneterlepas dari berapa banyak contoh di masing-masing. Jika Anda memiliki sumber daya yang tidak seimbang atau grup Penskalaan Otomatis, ini dapat memuat sumber daya tambahan di Availability Zone yang memiliki sumber daya lebih sedikit daripada yang lain. Ini ditunjukkan pada gambar berikut di mana dua instance di Availability Zone 1 masing-masing menerima 25% dari beban dan lima instance di Availability Zone 2 masing-masing menerima 10% dari beban.



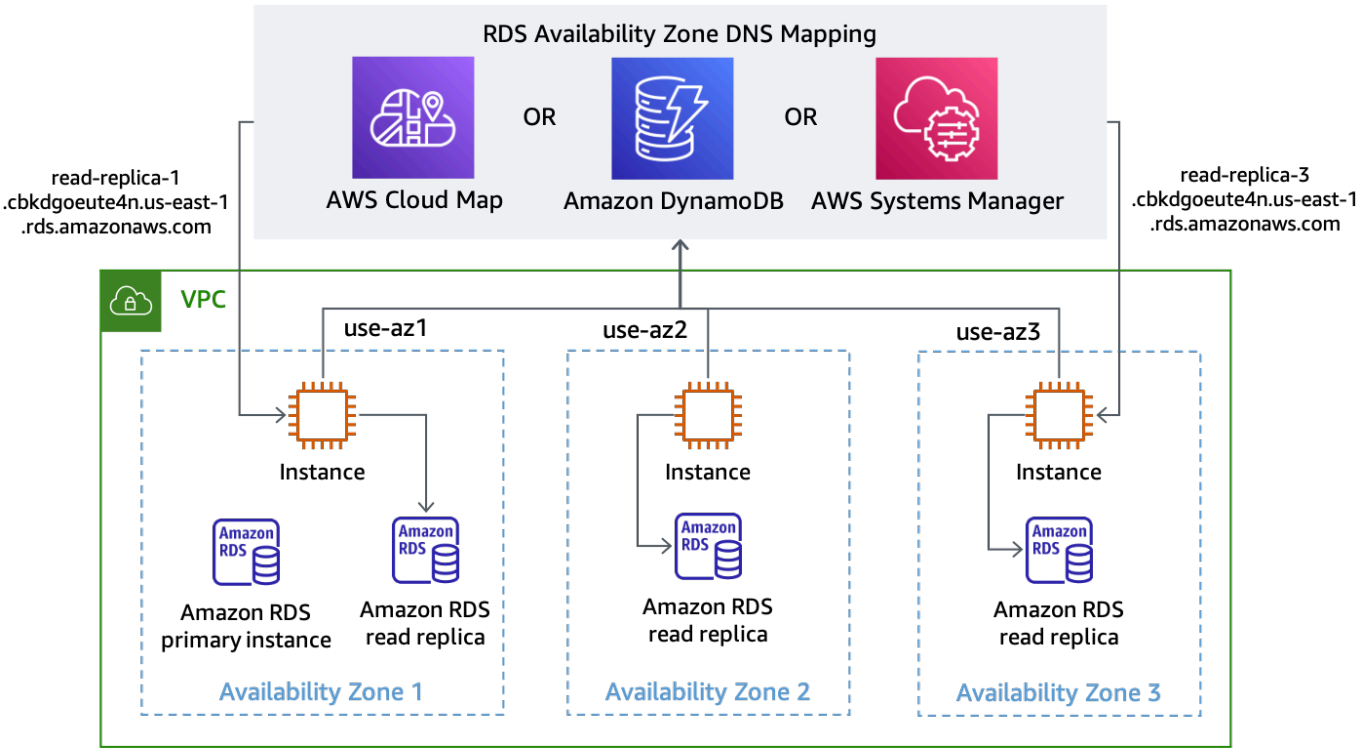
Efek menonaktifkan penyeimbangan beban lintas zona dengan instance yang tidak seimbang

Layanan zonal lain yang Anda gunakan juga perlu diimplementasikan menggunakan pola AZI untuk mendukung evakuasi Availability Zone yang efektif. Misalnya, antarmuka VPC endpoint menyediakannama DNS spesifik untuk setiap Availability Zoneendpoint antarmuka dibuat tersedia di.

Salah satu tantangan dengan menerapkan AZI adalah dengan database, terutama karena sebagian besar database relasional hanya mendukung penulis utama tunggal setiap saat. Saat berkomunikasi dengan instans utama, Anda mungkin perlu melewati batas Availability Zone. BanyakAWSlayanan database mendukung konfigurasi Multi-AZ yang ditentukan pengguna dan memiliki fitur failover Multi-AZ bawaan, sepertiAmazon RDSatauAurora. Dalam banyak skenario kegagalan, layanan dapat mendeteksi dampak dan secara otomatis failover database ke Availability Zone yang berbeda ketika masalah terjadi. Namun, selama kegagalan abu-abu, layanan mungkin tidak mendeteksi dampak yang memengaruhi beban kerja Anda, atau dampaknya mungkin tidak terkait dengan database sama sekali. Dalam kasus ini, setelah Anda mendeteksi dampak di Availability Zone, Anda dapat memanggil failover secara manual untuk memindahkan database utama. Hal ini memungkinkan Anda untuk secara efektif bereaksi terhadap gangguan Availability Zone tunggal.

Jika Anda menggunakan replika baca dengan database tersebut, Anda mungkin juga ingin menerapkan AZI untuk mereka karena Anda tidak dapat failover replika baca ke Availability Zone yang berbeda seperti Anda dapat database utama. Jika Anda memiliki replika baca tunggal di Availability Zone 1, dan instans di tiga Availability Zone dikonfigurasi untuk menggunakannya, gangguan yang memengaruhi Availability Zone 1 juga akan memengaruhi operasi di dua Availability Zone lainnya. Itulah dampak yang ingin Anda cegah.

Untuk instans RDS, Anda menerima endpoint DNS untuk mengakses replika di Availability Zone tertentu. Untuk mencapai AZI, Anda memerlukan replika baca per Availability Zone dan cara agar aplikasi Anda mengetahui titik akhir replika mana yang akan digunakan untuk Availability Zone yang ada. Salah satu pendekatan yang dapat Anda ambil adalah dengan menggunakan Availability Zone ID sebagai bagian dari pengenal database, sesuatu seperti`use1-az1-read-replica.cbkdgoeute4n.us-east-1.rds.amazonaws.com`. Anda juga dapat melakukan ini menggunakan penemuan layanan (seperti denganAWS Cloud Map) atau mencari peta sederhana yang disimpan diAWSToko Parameter Manajer Sistematau tabel DynamoDB. Konsep ini ditunjukkan pada gambar berikut.

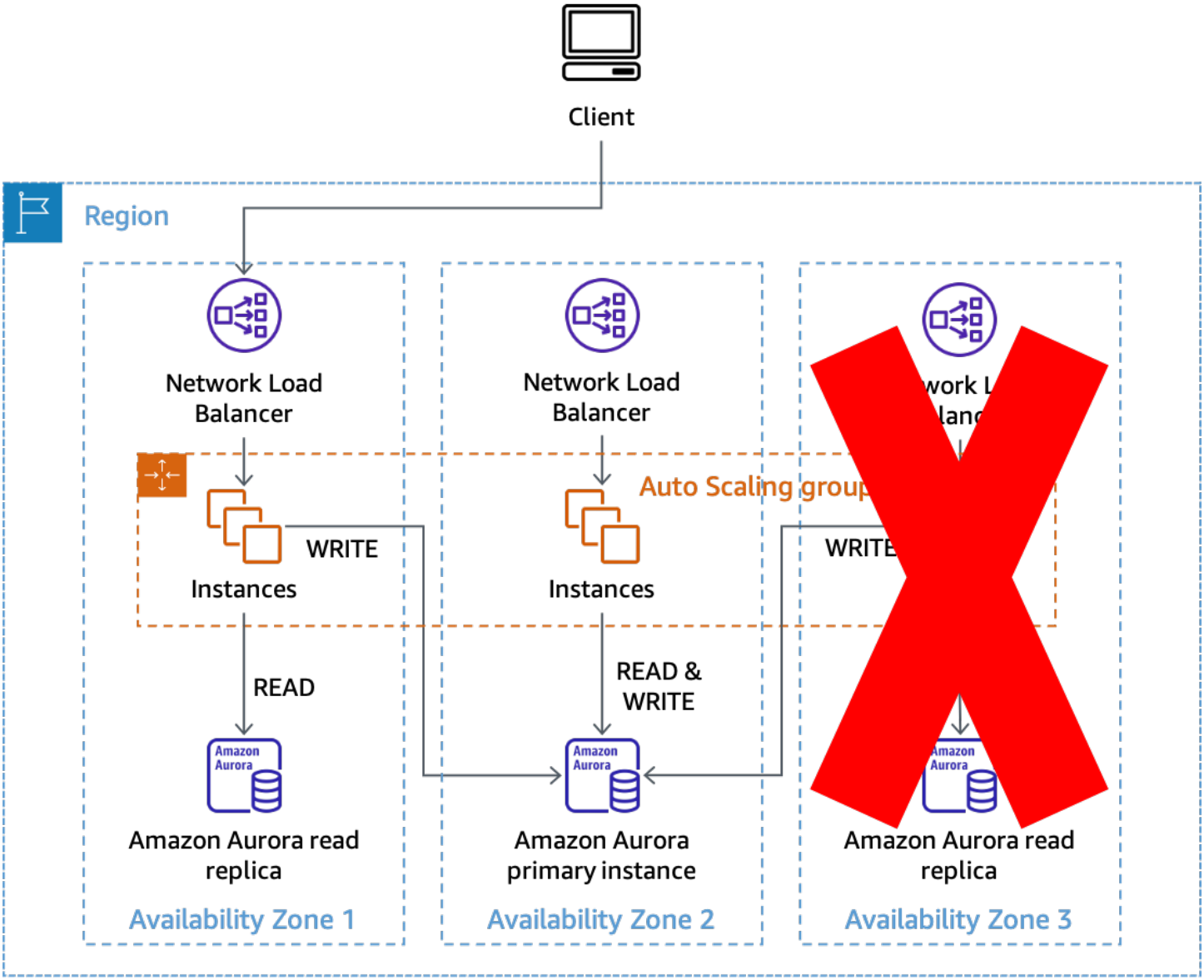


Menemukan nama DNS endpoint RDS untuk setiap Availability Zone

Konfigurasi default Amazon Aurora adalah menyediakanendpoint pembaca tunggalyang memuat permintaan saldo di seluruh replika baca yang tersedia. Untuk mengimplementasikan AZI menggunakan Aurora, Anda dapat menggunakanendpoint kustomuntuk setiap replika baca menggunakanANYketik (sehingga Anda dapat mempromosikan replika baca jika diperlukan). Beri nama titik akhir kustom berdasarkan ID Availability Zone tempat replika diterapkan. Kemudian, Anda dapat menggunakan nama DNS yang disediakan oleh titik akhir kustom untuk menyambung ke replika baca tertentu di Availability Zone tertentu, yang ditunjukkan pada gambar berikut.

Menggunakan titik akhir khusus untuk replika baca Aurora

Ketika sistem Anda dirancang dengan cara ini, itu membuat evakuasi Availability Zone menjadi tugas yang jauh lebih sederhana. Misalnya, pada gambar berikut ketika ada gangguan yang mempengaruhi Availability Zone 3, operasi baca dan tulis di Availability Zone 1 dan 2 tidak terpengaruh.



Menggunakan AZI untuk mencegah dampak dengan replika baca Amazon Aurora

Atau, jika Availability Zone 2 terkena dampak, operasi baca akan tetap berhasil di Availability Zone 1 dan 3. Kemudian, jika Amazon Aurora tidak secara otomatis gagal melalui database utama, Anda dapat memanggil failover secara manual ke Availability Zone yang berbeda untuk memulihkan kemampuan pemrosesan penulisan. Pendekatan ini mencegah kebutuhan untuk membuat perubahan konfigurasi dalam koneksi database Anda ketika Anda perlu untuk mengevakuasi Availability Zone. Meminimalkan perubahan yang diperlukan dan menjaga proses sesederhana mungkin akan membuatnya lebih dapat diandalkan.

Mengontrol pesawat dan bidang data

Sebelum kita mencapai pola aktual yang dapat Anda gunakan untuk melakukan evakuasi Availability Zone, kita perlu membahas konsep bidang kontrol dan bidang data.AWSmembuat perbedaan antara bidang kontrol dan pesawat data dalam layanan kami. Pesawat kontrol adalah mesin yang terlibat dalam membuat perubahan pada sistem—menambahkan sumber daya, menghapus sumber daya, memodifikasi sumber daya—dan mendapatkan perubahan tersebut disebarkan ke mana pun mereka perlu pergi untuk berlaku, seperti memperbarui konfigurasi jaringan untuk ALB atau membuatAWS Lambdafungsi.

Bidang data adalah fungsi utama dari sumber daya tersebut, hal-hal seperti instans EC2 yang sedang berjalan, atau mendapatkan item dari atau memasukkan item ke dalam tabel Amazon DynamoDB. Untuk diskusi yang lebih rinci tentang bidang kontrol dan bidang data, lihatStabilitas statis menggunakan Availability ZonedanAWSBatas Isolasi Kesalahan.

Untuk keperluan dokumen ini, pertimbangkan bahwa bidang kontrol cenderung memiliki lebih banyak bagian dan dependensi yang bergerak daripada bidang data. Ini membuatnya secara statistik lebih mungkin bahwa bidang kontrol menjadi terganggu dibandingkan dengan bidang data. Ini sangat relevan untuk layanan yang menyediakan AZI, seperti Amazon EC2 dan EBS, karena bagian dari layanan tersebut memiliki bidang kontrol yang juga independen secara zonal dan dapat terkena dampak selama peristiwa Single-AZ.

Sementara tindakan pesawat kontrol dapat digunakan untuk melakukan evakuasi AZ, berdasarkan informasi sebelumnya, mereka mungkin memiliki probabilitas keberhasilan yang lebih rendah, terutama selama peristiwa kegagalan. Untuk meningkatkan kemungkinan dampak mitigasi yang berhasil, Anda dapat menggunakan dua pola yang berbeda. Pola pertama hanya bergantung pada tindakan bidang data untuk awalnya mengurangi dampak dengan mencegah pekerjaan dialihkan ke atau menghentikan pekerjaan agar tidak dilakukan di Availability Zone yang terkena dampak. Kemudian, pola kedua dapat dicoba untuk memperbarui konfigurasi sumber daya dengan tindakan bidang kontrol untuk mencegah kapasitas disediakan di Availability Zone yang terkena dampak serta menghentikan komunikasi Inter-availability Zone dengan Availability Zone tersebut.

Pola pemulihan yang dibahas di bagian ini adalahtombol merah besar. Mereka adalah mekanisme yang Anda gunakan untuk mengambil tindakan skala besar, cepat, mirip dengan menarikKabel Andon pada jalur perakitan. Mereka berasumsi bahwa beban kerja telah mencoba strategi seperticoba lagi dengan backoff eksponensial dengan jitterdalam kode mereka untuk mengatasi kesalahan sementara. Ini berarti bahwa ketika dampak Availability Zone terisolasi terdeteksi, efeknya

pada ketersediaan atau latensi cukup parah sehingga memerlukan evakuasi Availability Zone untuk mengurangi secara efektif.

Evakuasi data yang dikendalikan pesawat

Ada beberapa solusi yang dapat Anda terapkan untuk melakukan evakuasi Availability Zone menggunakan tindakan khusus bidang data. Bagian ini akan menjelaskan tiga dari mereka dan kasus penggunaan di mana Anda mungkin ingin memilih satu dari yang lain.

Saat menggunakan salah satu solusi ini, Anda perlu memastikan bahwa Anda memiliki kapasitas yang cukup di Availability Zone yang tersisa untuk menangani beban Availability Zone tempat Anda beralih. Yang paling tangguh adalah melakukan ini adalah dengan memiliki kapasitas yang diperlukan pra-penyediaan di setiap Availability Zone. Jika Anda menggunakan tiga Availability Zone, Anda akan memiliki 50% dari kapasitas yang diperlukan untuk menangani beban puncak yang digunakan di masing-masing Availability Zone, sehingga hilangnya Availability Zone tunggal akan tetap membuat Anda 100% dari kapasitas yang diperlukan tanpa harus bergantung pada bidang kontrol untuk menyediakan lebih banyak.

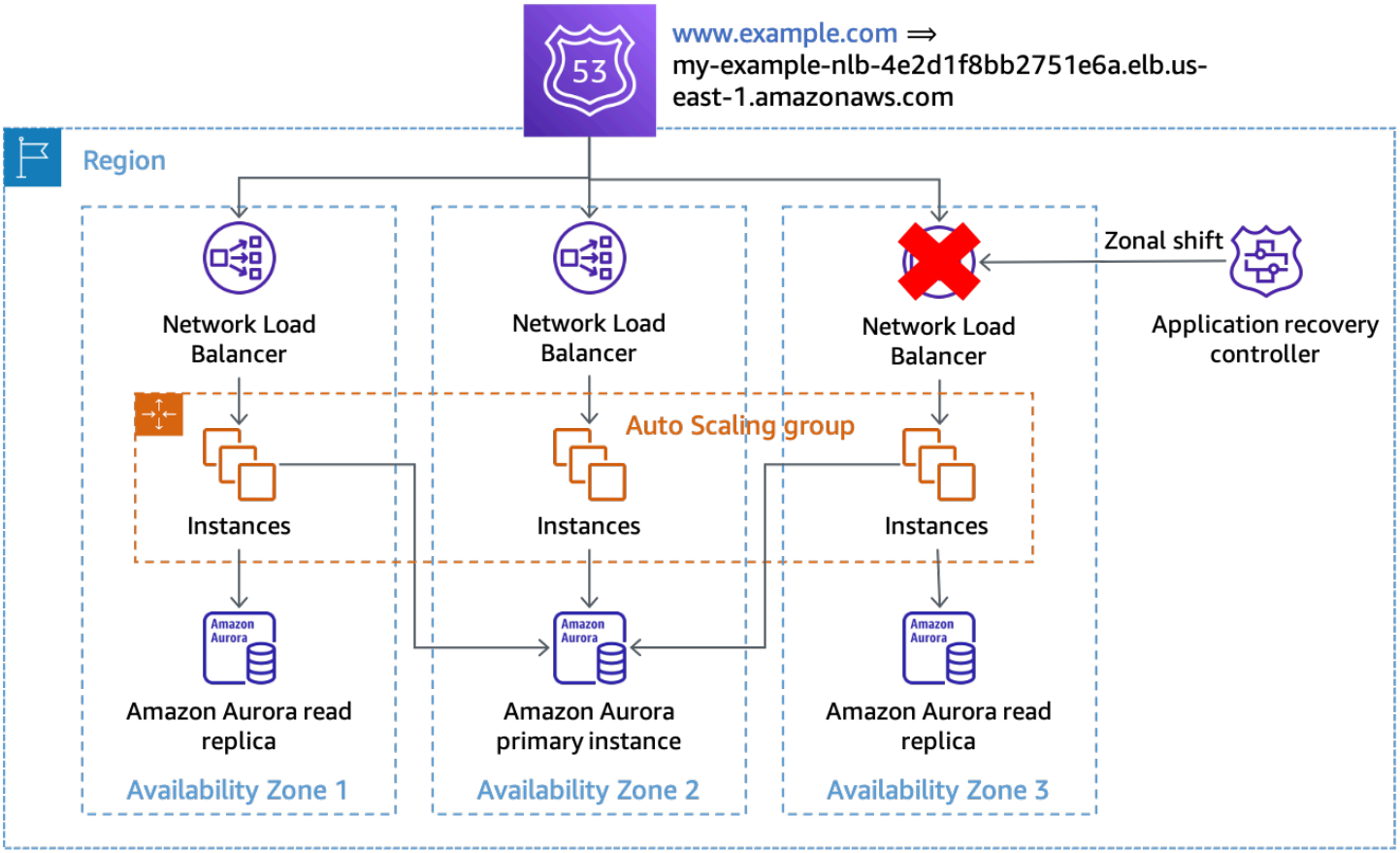
Selain itu, jika Anda menggunakan EC2 Auto Scaling, pastikan grup Auto Scaling (ASG) Anda tidak menskalakan selama shift, sehingga ketika shift berakhir, Anda masih memiliki kapasitas yang cukup dalam grup untuk menangani lalu lintas pelanggan Anda. Anda dapat melakukan ini dengan memastikan bahwa kapasitas minimum yang diinginkan ASG Anda dapat menangani beban pelanggan Anda saat ini. Anda juga dapat membantu memastikan bahwa ASG Anda tidak secara tidak sengaja menskalakan dengan menggunakan rata-rata dalam metrik Anda dibandingkan dengan metrik persentil outlier seperti P90 atau P99.

Selama shift, sumber daya yang tidak lagi melayani lalu lintas harus memiliki pemanfaatan yang sangat rendah, tetapi sumber daya lainnya akan meningkatkan pemanfaatannya dengan lalu lintas baru, menjaga rata-rata cukup konsisten, yang akan mencegah tindakan skala. Akhirnya, Anda juga dapat menggunakan pengaturan kesehatan grup target untukALBdanNLBuntuk menentukan failover DNS dengan persentase atau jumlah host sehat. Ini mencegah lalu lintas dialihkan ke Availability Zone yang tidak memiliki cukup host sehat.

Zonal Shift di Route 53 Aplikasi Recovery Controller (ARC)

Solusi pertama untuk penggunaan evakuasi Availability Zonepergeseran zona di Route 53 ARC. Solusi ini dapat digunakan untuk beban kerja permintaan/respons yang menggunakan NLB atau ALB sebagai titik masuknya lalu lintas pelanggan.

Ketika Anda mendeteksi bahwa Availability Zone telah mengalami gangguan, Anda dapat memulai pergeseran zonal dengan Route 53 ARC. Setelah operasi ini selesai dan respons DNS cache yang ada kedaluwarsa, semua permintaan baru hanya dialihkan ke sumber daya di Availability Zone yang tersisa. Gambar berikut menunjukkan bagaimana pergeseran zonal bekerja. Pada gambar berikut kita memiliki Route 53 alias record untuk`www.example.com`yang menunjuk ke`my-example-nlb-4e2d1f8bb2751e6a.elb.us-east-1.amazonaws.com`. Pergeseran zonal dilakukan untuk Availability Zone 3.



Pergeseran zonal

Dalam contoh, jika instance database utama tidak berada di Availability Zone 3, maka melakukan pergeseran zonal adalah satu-satunya tindakan yang diperlukan untuk mencapai hasil pertama untuk evakuasi, mencegah pekerjaan diproses di Availability Zone yang terkena dampak. Jika node utama berada di Availability Zone 3, maka Anda dapat melakukan failover yang dimulai secara manual (yang bergantung pada bidang kontrol Amazon RDS) dalam koordinasi dengan pergeseran zonal, jika Amazon RDS belum melakukan failover secara otomatis. Ini akan berlaku untuk semua solusi data yang dikendalikan pesawat di bagian ini.

Anda harus memulai pergeseran zonal menggunakan perintah CLI atau API untuk meminimalkan dependensi yang diperlukan untuk memulai evakuasi. Semakin sederhana proses evakuasi, semakin dapat diandalkan. Perintah spesifik dapat disimpan dalam runbook lokal yang dapat diakses oleh teknisi panggilan dengan mudah. Pergeseran zona adalah solusi yang paling disukai dan paling sederhana untuk mengevakuasi Availability Zone.

Rute 53 ARC

Solusi kedua menggunakan kemampuan Route 53 ARC untuk secara manual menentukan kesehatan catatan DNS tertentu. Solusi ini memiliki manfaat menggunakan pesawat data cluster Route 53 ARC yang sangat tersedia, sehingga tahan terhadap gangguan hingga dua yang berbedaWilayah AWS. Ini memiliki tradeoff biaya tambahan dan memerlukan beberapa konfigurasi tambahan catatan DNS. Untuk menerapkan pola ini, Anda perlu membuat catatan alias untukNama DNS khusus Zona Ketersediaandisediakan oleh load balancer (ALB atau NLB). Hal ini ditunjukkan pada tabel berikut.

Tabel 3: Route 53 catatan alias dikonfigurasi untuk nama DNS zonal load balancer

Kebijakan Routing: tertimbang	Kebijakan Routing:berbobot	Kebijakan Routing:berbobot
Nama: `www.example.com`	Nama: `www.example.com`	Nama: `www.example.com`
Jenis:`A`(alias)	Jenis: `A`(alias)	Jenis: `A`(alias)
Nilai:`us-east-1b.load-balancer-name.elb.us-east-1.amazonaws.com` Berat:`100` Evaluasi Target Kesehatan: benar	Nilai `us-east-1a.load-balancer-name.elb.us-east-1.amazonaws.com` : Berat: `100` Evaluasi Kesehatan Target: `true`	Nilai `us-east-1c.load-balancer-name.elb.us-east-1.amazonaws.com` : Berat: `100` Evaluasi Kesehatan Target: `true`

Untuk masing-masing data DNS ini, Anda akan mengkonfigurasi pemeriksaan kesehatan Route 53 yang terkait dengan Route 53 ARCkontrol perutean. Saat Anda ingin memulai evakuasi Availability Zone, atur status kontrol perutean ke`Off`.AWSmenyarankan Anda melakukan ini menggunakan CLI atau API untuk meminimalkan dependensi yang diperlukan untuk memulai evakuasi Availability

Zone. Sebagaipraktek terbaik, Anda harus menyimpan salinan lokal dari titik akhir cluster Route 53 ARC sehingga Anda tidak perlu mengambilnya dari bidang kontrol ARC saat Anda perlu melakukan evakuasi.

Untuk meminimalkan biaya saat menggunakan pendekatan ini, Anda dapat membuat satu klaster Route 53 ARC dan pemeriksaan kesehatan dalam satuAkun AWSdanbagikan pemeriksaan kesehatan dengan yang lainAkun AWSdi organisasi Anda. Bila Anda mengambil pendekatan ini, Anda harus menggunakanID Zona Ketersediaan(AZ-ID) (misalnya,`use1-az1`) alih-alih nama Availability Zone (misalnya,`us-east-1a`) untuk kontrol routing Anda. KarenaAWSmemetakan Availability Zone fisik secara acak ke nama Availability Zone untuk masing-masingAkun AWS, menggunakan AZ-ID menyediakan cara yang konsisten untuk merujuk ke lokasi fisik yang sama. Saat Anda memulai evakuasi Availability Zone, katakan untuk`use1-az2`, set rekaman Route 53 di masing-masingAkun AWSharus memastikan mereka menggunakan pemetaan AZ-ID untuk mengkonfigurasi pemeriksaan kesehatan yang tepat untuk setiap catatan NLB.

Misalnya, kita memiliki pemeriksaan kesehatan Route 53 yang terkait dengan kontrol routing Route 53 ARC untuk`use1-az2`, dengan ID`0385ed2d-d65c-4f63-a19b-2412a31ef431`. Jika berbedaAkun AWSyang ingin menggunakan pemeriksaan kesehatan ini,`us-east-1c`dipetakan ke`use1-az2`, Anda perlu menggunakan`use1-az2`pemeriksaan kesehatan untuk catatan`us-east-1c.load-balancer-name.elb.us-east-1.amazonaws.com`. Anda akan menggunakan ID pemeriksaan kesehatan`0385ed2d-d65c-4f63-a19b-2412a31ef431`dengan catatan sumber daya yang ditetapkan.

Menggunakan endpoint HTTP yang dikelola sendiri

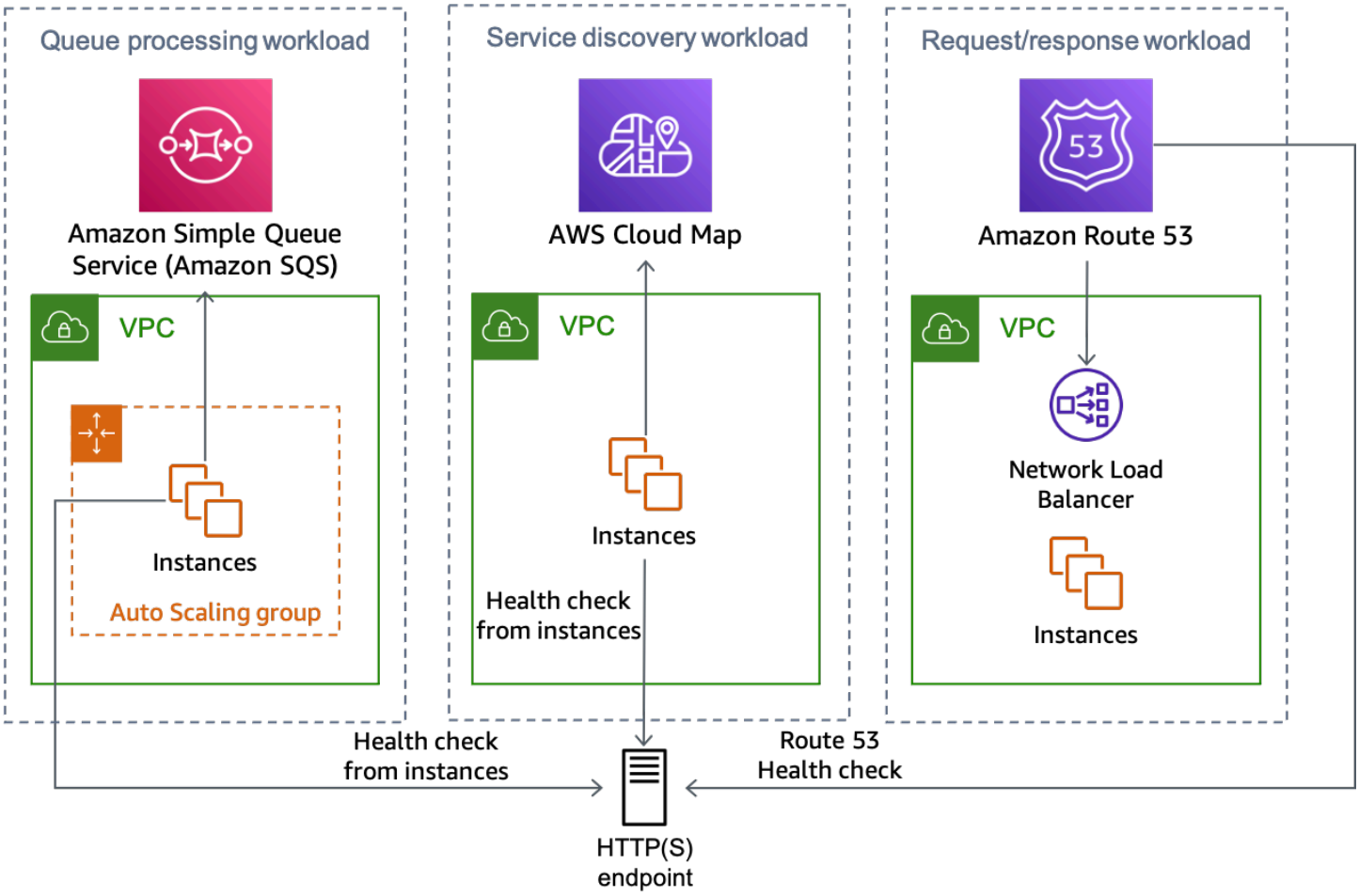
Anda juga dapat menerapkan solusi ini dengan mengelola endpoint HTTP Anda sendiri yang menunjukkan status Availability Zone tertentu. Ini memungkinkan Anda untuk menentukan secara manual kapan Availability Zone tidak sehat berdasarkan respons dari titik akhir HTTP. Solusi ini harganya lebih murah daripada menggunakan Route 53 ARC, tetapi lebih mahal daripada pergeseran zonal dan membutuhkan pengelolaan infrastruktur tambahan. Ini memiliki manfaat menjadi jauh lebih fleksibel untuk skenario yang berbeda.

Pola ini dapat digunakan dengan arsitektur NLB atau ALB dan pemeriksaan kesehatan Route 53. Ini juga dapat digunakan dalam arsitektur seimbang non-beban, seperti penemuan layanan atau sistem pemrosesan antrian di mana node pekerja melakukan pemeriksaan kesehatan mereka sendiri. Dalam skenario tersebut, host dapat menggunakan thread latar belakang di mana mereka secara berkala membuat permintaan ke endpoint HTTP dengan AZ-ID mereka (lihatLampiran A —

Mendapatkan ID Availability Zone untuk rincian tentang bagaimana menemukan ini) dan menerima kembali respon tentang kesehatan Availability Zone.

Jika Availability Zone dinyatakan tidak sehat, mereka memiliki beberapa opsi tentang cara merespons. Mereka dapat memilih untuk gagal pemeriksaan kesehatan eksternal dari sumber seperti ELB, Route 53, atau pemeriksaan kesehatan khusus dalam arsitektur penemuan layanan sehingga mereka tampak tidak sehat untuk layanan tersebut. Mereka juga dapat segera merespons dengan kesalahan jika mereka menerima permintaan, memungkinkan klien untuk mundur dan mencoba lagi. Dalam arsitektur yang digerakkan oleh peristiwa, node dapat dengan sengaja gagal memproses pekerjaan, seperti dengan sengaja mengembalikan pesan SQS ke antrian. Dalam bekerja arsitektur router di mana jadwal layanan pusat bekerja pada host tertentu Anda juga dapat menggunakan pola ini. Router dapat memeriksa status Availability Zone sebelum memilih pekerja, titik akhir, atau sel. Dalam arsitektur penemuan layanan yang menggunakanAWS Cloud Map, kamu bisatemukan titik akhir dengan menyediakan filter dalam permintaan Anda, seperti AZ-ID.

Gambar berikut menunjukkan bagaimana pendekatan ini dapat digunakan untuk beberapa jenis beban kerja.



Beberapa jenis beban kerja semuanya dapat menggunakan solusi endpoint HTTP

Ada beberapa cara untuk menerapkan pendekatan endpoint HTTP, dua di antaranya diuraikan berikutnya.

Menggunakan Amazon S3

Pola ini awalnya disajikan dalam hal iniposting bloguntuk pemulihan bencana multi-wilayah. Anda dapat menggunakan pola yang sama untuk evakuasi Availability Zone.

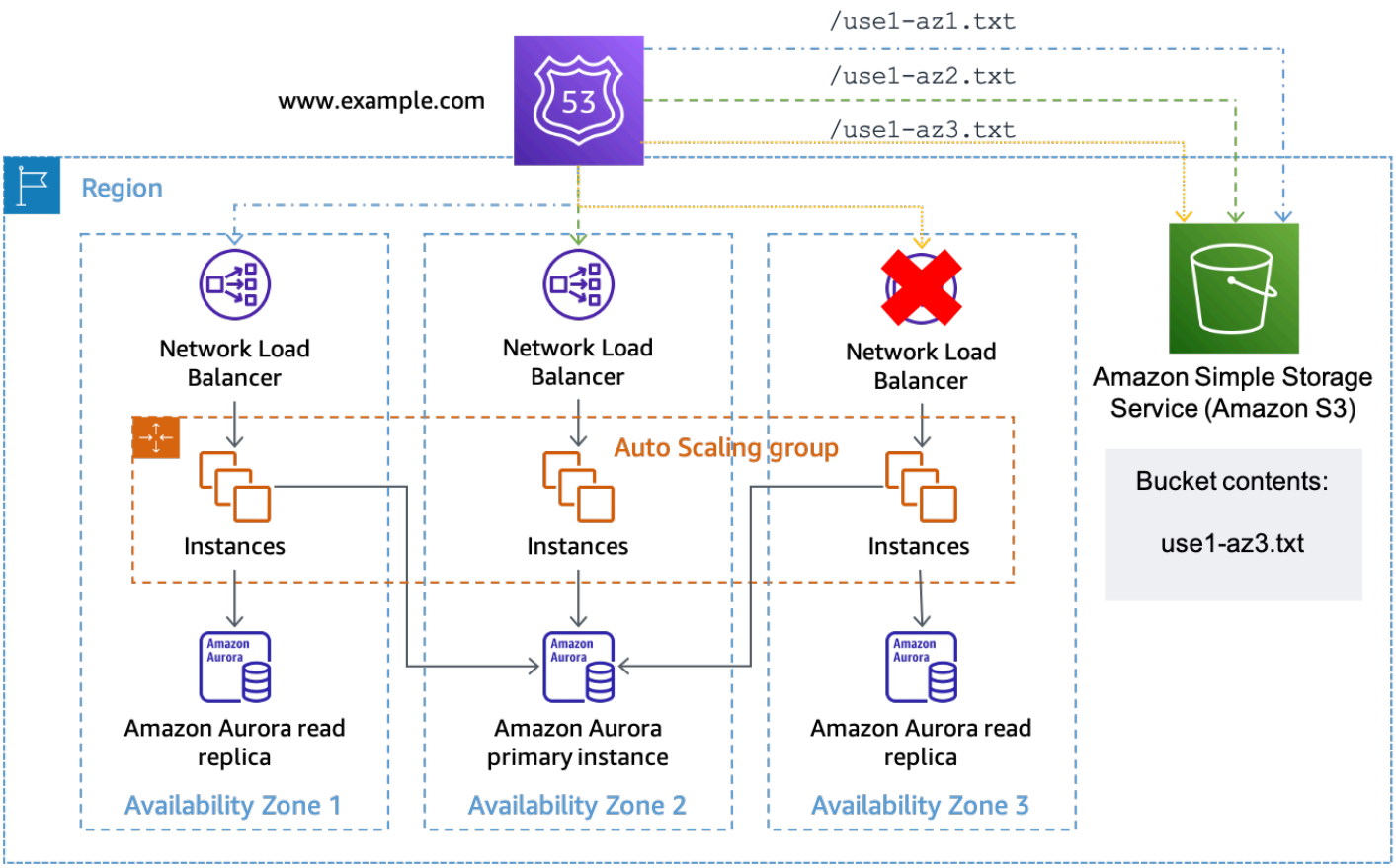
Dalam skenario ini Anda akan membuat kumpulan data sumber daya Route 53 DNS untuk setiap catatan DNS zona sepertiRute 53 ARCskenario di atas serta pemeriksaan kesehatan terkait. Namun, untuk implementasi ini, alih-alih mengaitkan pemeriksaan kesehatan dengan kontrol routing Route 53 ARC, mereka dikonfigurasi untuk menggunakanTitik akhir HTTPdan terbalik untuk melindungi terhadap gangguan di Amazon S3 secara tidak sengaja memicu evakuasi. Pemeriksaan kesehatan dipertimbangkansehatketika objek tidak ada dantidak sehatketika objek hadir. Pengaturan ini ditunjukkan pada tabel berikut.

Tabel 4: Konfigurasi catatan DNS untuk menggunakan pemeriksaan kesehatan Route 53 per Availability Zone

Jenis pemeriksaan kesehatan: memonitor titik akhir Protokol: HTTPS ID:dddd-4444 URL:https://*bucketname*.s3.us-east-1.amazonaws.com/use1-az1.txt	Jenis pemeriksaan kesehatan: memonitor titik akhir Protokol:HTTPS ID:eeee-5555 URL: https://*bucketname*.s3.us-east-1.amazonaws.com/use1-az3.txt	Jenis pemeriksaan kesehatan: memonitor titik akhir Protokol:HTTPS ID:ffff-6666 URL:https://*bucketname*.s3.us-east-1.amazonaws.com/use1-az2.txt	←	Pemeriksaan kesehatan
↑	↑	↑		

Kebijakan Routing: tertimbang Nama: `www.example.com` Jenis:A(alias) Nilai:`us-east-1b.load-balancer-name.elb.us-east-1.amazonaws.com` Berat:`100` Evaluasi Target Kesehatan:`true`	Kebijakan Routing:berbobot Nama: `www.example.com` Jenis: A(alias) Nilai: `us-east-1a.load-balancer-name.elb.us-east-1.amazonaws.com` Berat: `100` Evaluasi Kesehatan Target: `true`	Kebijakan Routing:berbobot Nama: `www.example.com` Jenis: A(alias) Nilai: `us-east-1c.load-balancer-name.elb.us-east-1.amazonaws.com` Berat: `100` Evaluasi Kesehatan Target: `true`	←	Tingkat atas, alias tertimbang merata Catatan A menunjuk ke titik akhir spesifik NLB AZ

Mari kita asumsikan bahwa Availability Zone`us-east-1a`dipetakan ke`use1-az3`di akun di mana kita memiliki beban kerja di mana kita ingin melakukan evakuasi Availability Zone. Untuk kumpulan catatan sumber daya yang dibuat untuk`us-east-1a.load-balancer-name.elb.us-east-1.amazonaws.com`akan mengaitkan pemeriksaan kesehatan yang menguji URL`https://bucket-name.s3.us-east-1.amazonaws.com/use1-az3.txt`. Saat Anda ingin memulai evakuasi Availability Zone untuk`use1-az3`, unggah file bernama`use1-az3.txt`ke bucket menggunakan CLI atau API. File tidak perlu berisi konten apa pun, tetapi harus bersifat publik sehingga pemeriksaan kesehatan Route 53 dapat mengaksesnya. Gambar berikut menunjukkan implementasi ini digunakan untuk mengungsi`use1-az3`.



Menggunakan Amazon S3 sebagai target untuk pemeriksaan kesehatan Route 53

Menggunakan API Gateway dan DynamoDB

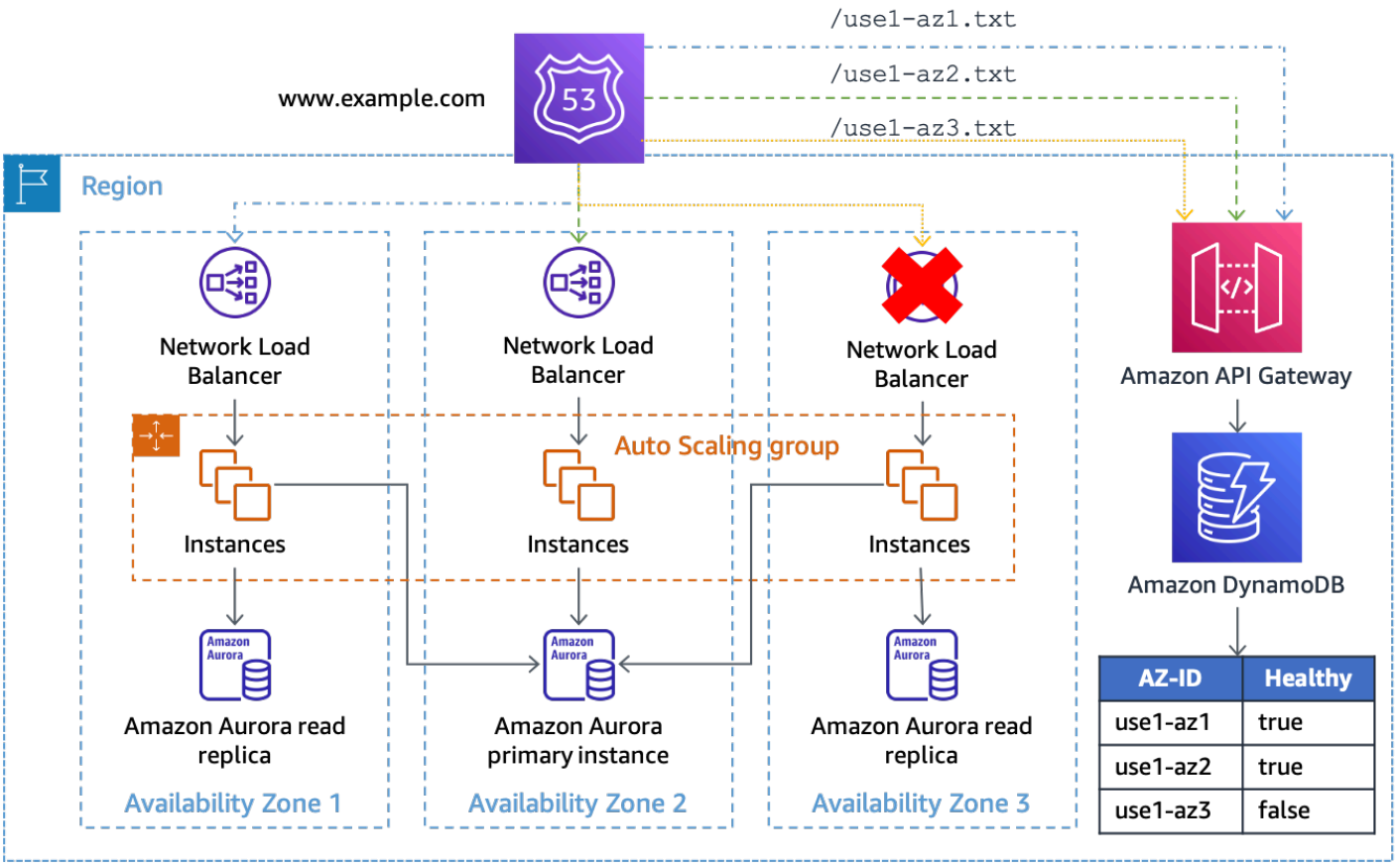
Implementasi kedua dari pola ini menggunakanGerbang API Amazon SISANYA API. API dikonfigurasi denganintegrasi layananke Amazon DynamoDB tempat status untuk setiap Availability Zone yang sedang digunakan disimpan. Implementasi ini lebih fleksibel daripada pendekatan Amazon S3, tetapi membutuhkan pembangunan, pengoperasian, dan pemantauan lebih banyak infrastruktur. Ini juga dapat digunakan dengan pemeriksaan kesehatan Route 53 serta pemeriksaan kesehatan yang dilakukan oleh masing-masing host.

Jika Anda menggunakan solusi ini dengan arsitektur NLB atau ALB, siapkan data DNS Anda dengan cara yang sama seperti contoh Amazon S3 di atas, kecuali ubah jalur pemeriksaan kesehatan untuk menggunakan titik akhir API Gateway dan menyediakan`AZ-ID`di jalur URL. Misalnya, jika API Gateway dikonfigurasi dengan domain kustom`az-status.example.com`, permintaan penuh untuk`use1-az1`akan terlihat seperti`https://az-status.example.com/status/use1-az1`. Saat Anda ingin memulai evakuasi Availability Zone, Anda dapat membuat atau memperbarui item DynamoDB menggunakan CLI atau API. Item menggunakan`AZ-ID`sebagai kunci utama dan

kemudian memiliki atribut Boolean disebut`Healthy`yang digunakan menunjukkan bagaimana API Gateway merespons. Berikut ini adalah contoh kode yang digunakan dalam konfigurasi API Gateway untuk membuat penentuan ini.

```
#set($inputRoot = $input.path('$'))
#if ($inputRoot.Item.Healthy['BOOL'] == (false))
    #set($context.responseOverride.status = 500)
#end
```

Jika atributnya adalah`true`(atau tidak ada), API Gateway merespons pemeriksaan kesehatan dengan HTTP 200, jika salah, itu merespons dengan HTTP 500. Implementasi ini ditunjukkan pada gambar berikut.



Menggunakan API Gateway dan DynamoDB sebagai target pemeriksaan kesehatan Route 53

Dalam solusi ini Anda perlu menggunakan API Gateway di depan DynamoDB sehingga Anda dapat membuat endpoint dapat diakses publik serta memanipulasi URL permintaan menjadi`GetItem`permintaan untuk DynamoDB. Solusinya juga memberikan fleksibilitas jika Anda ingin menyertakan data tambahan dalam permintaan. Misalnya, jika Anda ingin membuat status yang

lebih terperinci, seperti per aplikasi, Anda dapat mengonfigurasi URL pemeriksaan kesehatan untuk memberikan ID aplikasi di jalur atau string kueri yang juga cocok dengan item DynamoDB.

Endpoint status Availability Zone dapat diterapkan secara terpusat sehingga beberapa sumber daya pemeriksaan kesehatan di seluruhAkun AWSsemua dapat menggunakan tampilan konsisten yang sama dari kesehatan Availability Zone (memastikan bahwa API Gateway REST API dan tabel DynamoDB Anda diskalakan untuk menangani beban) dan menghilangkan kebutuhan untuk berbagi pemeriksaan kesehatan Route 53.

Solusinya juga bisa diskalakan di beberapaWilayah AWSmenggunakanTabel global Amazon DynamoDBdan salinan API REST API Gateway di setiap Region. Ini mencegah solusi ini memiliki ketergantungan pada satu Wilayah dan meningkatkan ketersediaannya. Anda dapat menerapkan solusi di tiga atau lima Wilayah dan meminta masing-masing untuk kesehatan Availability Zone, menggunakan hasil mayoritas titik akhir untuk memastikan kuorum. Hal ini memungkinkan replikasi pembaruan yang konsisten pada akhirnya di seluruh tabel global serta mengurangi gangguan yang dapat mencegah titik akhir merespons. Misalnya, jika Anda menggunakan lima Wilayah, dan tiga titik akhir melaporkan Availability Zone sebagai tidak sehat, satu titik akhir melaporkan Availability Zone sebagai sehat, dan satu titik akhir tidak merespons, Anda akan memilih untuk memperlakukan Availability Zone sebagai tidak sehat. Anda juga bisa membuatPemeriksaan kesehatan yang dihitung Route 53menggunakanm dari nkalkulasiuntuk melakukan logika ini untuk menentukan kesehatan Availability Zone.

Jika Anda membangun solusi untuk masing-masing host untuk digunakan sebagai mekanisme untuk menentukan kesehatan AZ mereka, sebagai alternatif, alih-alih menyediakan mekanisme tarik untuk pemeriksaan kesehatan, Anda dapat menggunakan pemberitahuan push. Salah satu cara untuk melakukannya adalah dengan topik SNS yang berlangganan konsumen Anda. Saat Anda ingin memicu pemutus arus, publikasikan pesan ke topik SNS yang menunjukkan Availability Zone mana yang terganggu. Pendekatan ini membuat tradeoff dengan mantan. Ini menghilangkan kebutuhan untuk membuat dan mengoperasikan infrastruktur API Gateway dan melakukan manajemen kapasitas. Ini juga berpotensi memberikan konvergensi yang lebih cepat dari status Availability Zone. Namun, ini menghilangkan kemampuan untuk melakukan kueri ad hoc dan bergantung padaKebijakan percobaan ulang pengiriman SNSuntuk memastikan setiap endpoint menerima notifikasi. Hal ini juga memerlukan setiap beban kerja atau layanan untuk membangun cara untuk menerima pemberitahuan SNS dan mengambil tindakan di atasnya.

Misalnya, setiap instans atau wadah EC2 baru yang diluncurkan perlu berlangganan topik dengan endpoint HTTP selama bootstrap. Kemudian, setiap instance perlu mengimplementasikan perangkat lunak yang mendengarkan titik akhir ini di mana notifikasi dikirimkan. Selain itu, jika instans

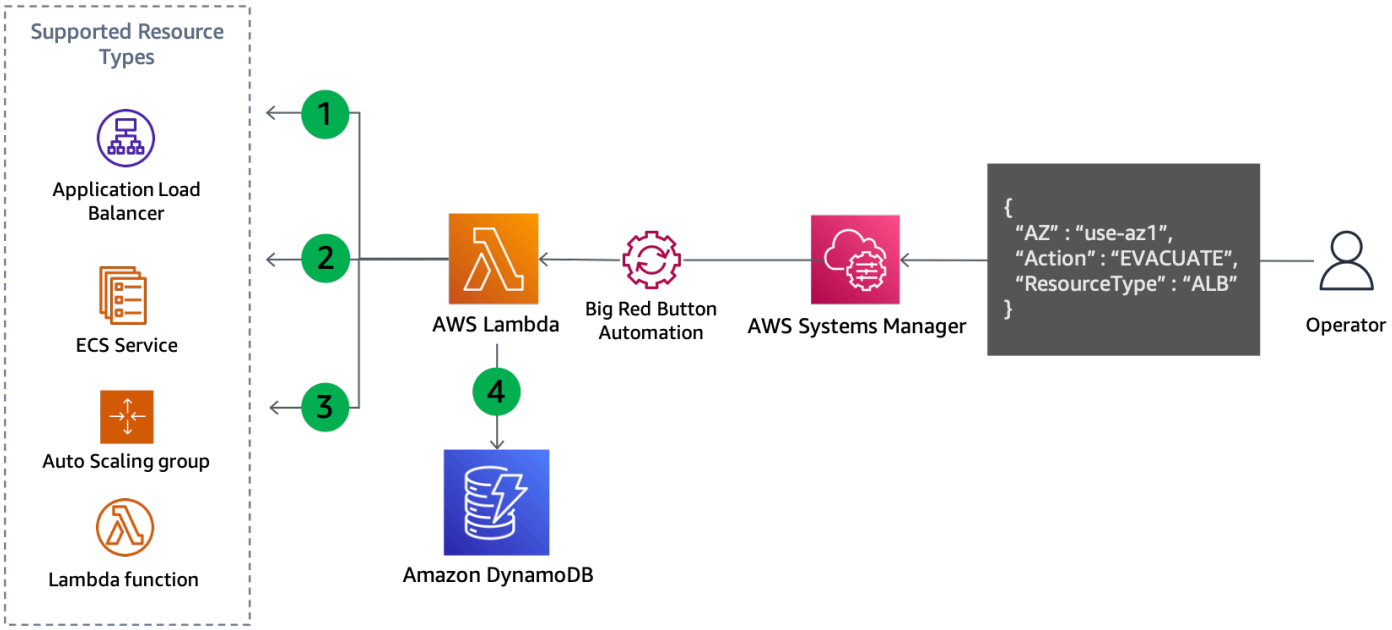
dipengaruhi oleh peristiwa, instans mungkin tidak menerima notifikasi push dan terus berfungsi. Sedangkan, dengan pull notification, instance akan tahu apakah pull request gagal dan dapat memilih tindakan apa yang akan diambil sebagai respons.

Cara kedua untuk mengirim notifikasi push adalah dengan umur panjangWebSocketkoneksi. Amazon API Gateway dapat digunakan untuk menyediakanWebSocketAPIbahwa konsumen dapat terhubung dan menerima pesan kapandikirim oleh backend. DenganWebSocket, instans dapat melakukan penarikan berkala untuk memastikan koneksi mereka sehat dan juga menerima pemberitahuan push latensi rendah.

Kontrol evakuasi yang dikendalikan pesawat

Pola pertama menggunakan operasi bidang data untuk mencegah melakukan pekerjaan di Availability Zone yang terkena dampak untuk mengurangi dampak suatu peristiwa. Namun, Anda mungkin menggunakan arsitektur yang tidak menggunakan penyeimbang muatan atau jika mengonfigurasi pemeriksaan kesehatan per host tidak layak dilakukan. Atau, Anda mungkin ingin mencegah kapasitas baru diterapkan ke Availability Zone yang terkena dampak melalui Penskalaan Otomatis atau penjadwalan pekerjaan normal.

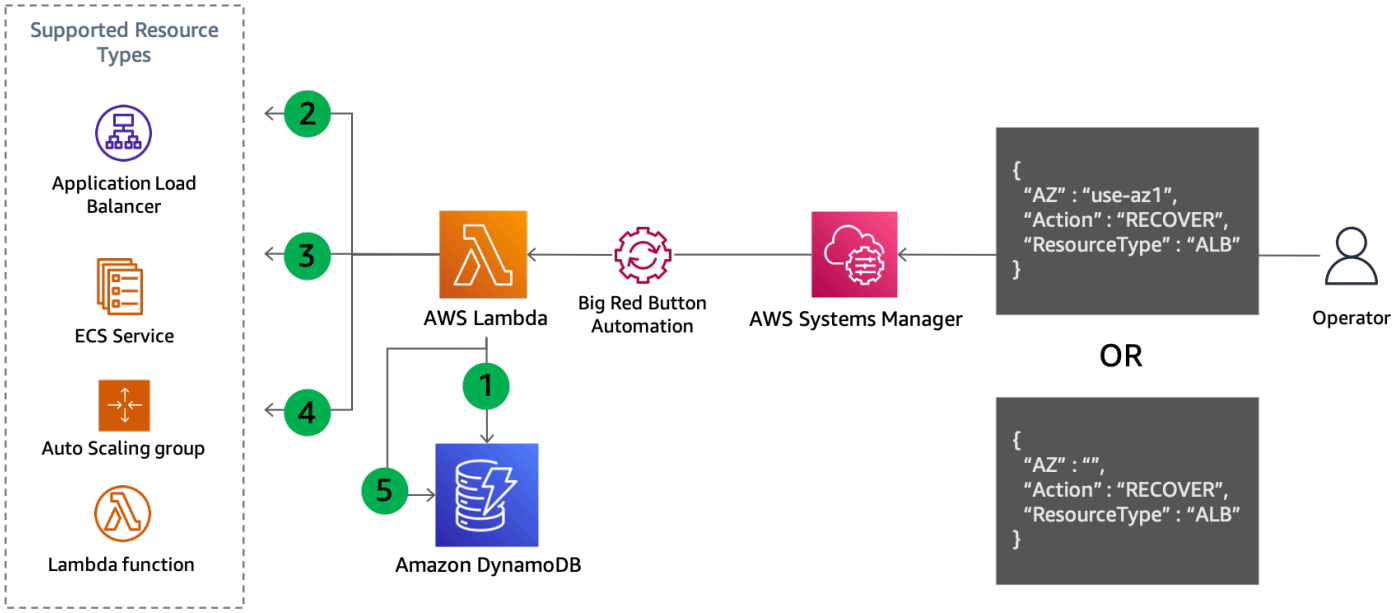
Untuk mengatasi kedua situasi, tindakan bidang kontrol diperlukan untuk memperbarui konfigurasi sumber daya. Pola ini akan berfungsi untuk layanan apa pun yang konfigurasi jaringannya dapat diperbarui, misalnya, EC2 Auto Scaling, Amazon ECS, Lambda, dan lainnya. Ini membutuhkan kode penulisan untuk setiap layanan, tetapi logika bisnis mengikuti pola standar. Kode harus dijalankan secara lokal oleh operator menanggapi acara untuk meminimalkan dependensi yang diperlukan. Aliran dasar logika skrip ditunjukkan pada gambar berikut.



Kontrol pembaruan pesawat untuk mengevakuasi Availability Zone

1. Script mencantumkan semua sumber daya dari jenis yang ditentukan, seperti grup Auto Scaling, layanan ECS, atau fungsi Lambda, dan mengambil subnet mereka dari informasi sumber daya. Sumber daya yang didukung bergantung pada apa skrip telah dikonfigurasi untuk mendukung.
2. Ini menentukan subnet mana yang harus dihapus dengan membandingkan nama Availability Zone setiap subnet dengan ID Availability Zone yang dipetakan yang disediakan sebagai parameter input.
3. Konfigurasi jaringan sumber daya diperbarui untuk menghapus subnet yang diidentifikasi.
4. Rincian pembaruan dicatat dalam tabel DynamoDB. ID Availability Zone disimpan sebagaikunci partisidan ARN sumber daya atau nama disimpan sebagaimenyortir kunci. Subnet yang dihapus disimpan sebagai array string. Akhirnya, jenis sumber daya juga disimpan dan digunakan sebagai kunci hash untukIndeks Sekunder Global(GSI).

Karena langkah keempat mencatat pembaruan yang dibuat, pendekatan ini juga cocok untuk mudah dibalik ketika Anda siap untuk pulih, seperti yang ditunjukkan pada gambar berikut.



Kontrol pembaruan pesawat untuk pulih dari evakuasi Availability Zone

Langkah pemulihan:

1. Kueri GSI agar subnet dihapus untuk setiap sumber daya dari jenis yang ditentukan di Availability Zone yang ditentukan (atau semua Availability Zone jika tidak ditentukan).
2. Jelaskan setiap sumber daya yang ditemukan dalam kueri DynamoDB untuk mendapatkan konfigurasi jaringan saat ini.
3. Gabungkan subnet dari konfigurasi jaringan saat ini dengan yang diambil dari kueri DynamoDB.
4. Perbarui konfigurasi jaringan sumber daya dengan set subnet baru.
5. Hapus catatan dari tabel DynamoDB setelah pembaruan selesai dengan sukses.

Pola umum ini mencegah pekerjaan perutean ke Availability Zone yang terkena dampak dan mencegah kapasitas baru diterapkan di sana. Berikut ini adalah contoh bagaimana hal ini dicapai untuk layanan yang berbeda.

- Lambda- Perbarui fungsiKonfigurasi VPCuntuk menghapus subnet di Availability Zone yang ditentukan.
- Grup Penskalaan Otomatis—Hapus subnet dari konfigurasi ASGyang akan menggantikan kapasitas itu di Availability Zone yang tersisa.
- Amazon ECS—Perbarui konfigurasi VPC layanan ECSuntuk menghapus subnet.

- Amazon— Terapkannodake node di Availability Zone yang terkena dampak untuk menggusur Pod yang ada dan mencegah Pod tambahan agar tidak dijadwalkan di sana.

Setiap layanan akan bereaksi berbeda terhadap pembaruan konfigurasi. Misalnya, Amazon ECS akan mengikutikonfigurasi penyebaran layanan setelah pembaruandan memicu penyebaran bergulir atau penyebaran biru/hijau tugas baru.

Pembaruan ini dapat mengalihkan pekerjaan ke Availability Zone yang sehat terlalu cepat untuk beberapa beban kerja. Saat dikonfigurasi agar stabil secara statis terhadap kegagalan (memiliki kapasitas yang cukup disediakan sebelumnya di Availability Zone yang tersisa untuk menangani pekerjaan Availability Zone yang terkena dampak), Anda mungkin juga ingin menghapus kapasitas secara bertahap dari Availability Zone yang terkena dampak.

Jika Anda berencana untuk memperbarui konfigurasi jaringan grup Auto Scaling yang merupakan grup target untuk penyeimbang muatan dengan penyeimbang muatan lintas zonadinonaktifkan, ikuti panduan ini.

Auto Scaling bereaksi terhadap perubahan ini menggunakanLogika penyeimbangan kembali Zona Ketersediaan. Ini akan meluncurkan instance di Availability Zone lain untuk memenuhi kapasitas yang Anda inginkan dan menghentikan instans di Availability Zone yang Anda hapus. Namun, load balancer akan terus membagi lalu lintas secara merata di setiap Availability Zone, termasuk yang Anda hapus dari ASG, sementara instans sedang dihentikan. Ini dapat menyebabkan kecoklatan dari kapasitas yang tersisa di Availability Zone tersebut hingga semua instance berhasil dihentikan di sana. Ini adalah masalah yang sama yang dijelaskan dalamIndependensi Zona Ketersediaantentang Ketidakseimbangan Zona Ketersediaan saat penyeimbangan beban lintas zona dinonaktifkan. Untuk mencegah hal ini terjadi, Anda dapat:

- Selalu lakukan evakuasi Availability Zone Anda terlebih dahulu sehingga lalu lintas hanya dibagi di antara Availability Zone yang tersisa
- Tentukanjumlah target sehat minimum dengan failover DNSuntuk mencocokkan jumlah target minimum yang diperlukan untuk Availability Zone tersebut.

Ini akan membantu memastikan lalu lintas tidak dikirim ke Availability Zone yang Anda hapus setelah instans mulai dihentikan.

Ringkasan

Tabel berikut merangkum pro dan kontra dari pola evakuasi yang dijelaskan.

Tabel 5: Pro dan kontra pola evakuasi

Pendekatan	Pro	Kontra
Evakuasi data yang dikendalikan pesawat	Hanya mengandalkan tindakan bidang data Dengan cepat mencegah pekerjaan dilakukan di Availability Zone yang terkena dampak Pendekatan fleksibel untuk pandangan terpusat kesehatan Availability Zone	Tidak mencegah kapasitas digunakan di Availability Zone yang terkena dampak Tidak semua jenis beban kerja dapat menggunakan pendekatan ini dengan mudah
Kontrol evakuasi yang dikendalikan pesawat	Mencegah kapasitas baru agar tidak digunakan di Availability Zone yang terkena dampak Menghapus kapasitas yang ada dari Availability Zone yang terkena dampak	Mengandalkan setiap bidang kontrol layanan Membutuhkan kode yang akan ditulis untuk setiap layanan Harus dilengkapi layanan dengan layanan Perlu berhati-hati untuk tidak membanjiri kapasitas selama pembaruan

Anda mungkin akan menggunakan kedua pendekatan bersama-sama sebagai bagian dari rencana evakuasi Availability Zone. Mulailah dengan tindakan evakuasi yang dikendalikan oleh data yang lebih mungkin berhasil menghentikan pekerjaan pemrosesan dengan cepat di Availability Zone yang terkena dampak. Kemudian, setelah dampak awal dikurangi, tindak lanjut dengan tindakan evakuasi yang dikendalikan pesawat kontrol, jika Anda anggap perlu.

Kesimpulan

Makalah ini memberikan ikhtisar kegagalan abu-abu, bagaimana mereka memanifestasikan, dan menguraikan mengapa Anda perlu membangun observability dan evakuasi tooling untuk mengurangi jenis-jenis peristiwa ketika mereka terjadi. Di bagian selanjutnya, Anda meninjau observabilitas Multi-AZ dan tiga pendekatan yang dapat Anda terapkan untuk mendeteksi dampak Availability Zone tunggal. Pada bagian terakhir, makalah ini mempresentasikan dua pendekatan umum untuk melakukan evakuasi Availability Zone. Pendekatan pertama menggunakan tindakan bidang data untuk mencegah pekerjaan dialihkan ke Availability Zone yang terkena dampak sementara pendekatan kedua menggunakan tindakan bidang kontrol untuk mencegah kapasitas disediakan di Availability Zone yang terkena dampak. Bersama-sama, kedua pendekatan ini mencapai dua hasil yang dimaksudkan evakuasi Availability Zone.

Pola pemulihan yang dijelaskan dalam makalah ini kemungkinan akan menjadi bagian dari solusi pemantauan dan pemulihan kesalahan yang lebih besar. Pendekatan untuk menangani kegagalan abu-abu Zona Ketersediaan Tunggal ini membutuhkan pekerjaan teknik untuk membangun instrumentasi yang diperlukan untuk mendeteksi mereka serta perkakas untuk meresponsnya. Namun, untuk banyak beban kerja, pendekatan ini bisa menjadi alternatif yang lebih sederhana dan lebih murah untuk membangun arsitektur multi-wilayah. Selain itu, ini dapat membantu mencapai RPO dan RTO yang lebih kecil (yang meningkatkan ketersediaan beban kerja) jika dibandingkan dengan DR.

Lampiran A — Mendapatkan ID Availability Zone

Jika Anda menggunakanAWS.NET SDK (serta beberapa lainnya sepertiJavaScript) atau menjalankan sistem Anda pada instans EC2 (termasuk Amazon ECS dan Amazon EKS), Anda bisa mendapatkan ID Availability Zone secara langsung.

- AWS.NET

```
Amazon.Util.EC2InstanceMetadata.GetData(“/placement/availability-zone-id”)
```

- Layanan Metadata Instans EC2

```
curl http://169.254.169.254/latest/meta-data/placement/availability-zone-id
```

  Di platform lain, seperti Lambda dan Fargate, Anda perlu mengambil nama Availability Zone dan kemudian menemukan pemetaan ke Availability Zone ID. Dengan nama Availability Zone Anda dapat menemukan ID Availability Zone seperti ini:

```
aws ec2 describe-availability-zones --zone-names $AZ --output json
    --query ‘AvailabilityZones[0].ZoneId’
```

  Contoh berikut untuk menemukan nama Availability Zone yang akan digunakan dalam contoh di atas ditulis dalam bash menggunakanAWS CLIdan paketnyajq. Mereka perlu dikonversi ke bahasa pemrograman yang digunakan untuk beban kerja Anda.

  - Amazon ECS- Jika Instance Metadata Service (IMDS) diblokir oleh host, Anda dapat menggunakan file metadata kontainer sebagai gantinya.

```
AZ=$(cat $ECS_CONTAINER_METADATA_FILE | jq --raw-output
    .AvailabilityZone)
```

  - Fargate(versi platform 1.4 atau yang lebih baru)

```
AZ=$(curl $ECS_CONTAINER_METADATA_URI_V4/task | jq --raw-output
    .AvailabilityZone)
```

- Lambda- Availability Zone tidak terpapar langsung ke fungsi. Untuk menemukannya, Anda perlu menyelesaikan beberapa langkah. Untuk melakukan ini, Anda perlu membuat endpoint REST API Gateway pribadi yang mengembalikan alamat IP pemohon. Ini akan mengidentifikasi IP pribadi yang ditetapkan ke antarmuka jaringan elastis yang digunakan oleh fungsi.
  - Panggil Lambda`GetFunction`API untuk menemukan ID VPC fungsi.
  - Panggil layanan API Gateway untuk mendapatkan IP fungsi.
  - Menggunakan IP dan ID VPC, temukan antarmuka jaringan terkait dan ekstrak Availability Zone.

```
VPC_ID=$(aws lambda get-function --function-name $ AWS_LAMBDA_FUNCTION_NAME --
region $AWS_REGION --output json --query ‘Configuration.VpcConfig.VpcId’)

MY_IP=$(curl http://whats-my-private-ip.internal)

AZ=$(aws ec2 describe-network-interfaces --filters Name=private-ip-address,Values=
$MY_IP Name=vpc-id,Values=$VPC_ID --region $AWS_REGION --output json –query
 ‘NetworkInterfaces[0].AvailabilityZone’)
```

Lampiran B - Contoh perhitungan chi-kuadrat

Berikut ini adalah contoh pengumpulan metrik kesalahan dan melakukan tes chi-kuadrat pada data. Kode ini tidak siap produksi dan tidak melakukan penanganan kesalahan yang diperlukan, tetapi memberikan bukti konsep tentang bagaimana logika bekerja. Anda harus memperbarui contoh ini agar sesuai dengan kebutuhan Anda.

Pertama, fungsi Lambda dipanggil setiap menit oleh AmazonEventBridgeacara yang dijadwalkan. Isi acara dikonfigurasi dengan data berikut:

```
{
  "timestamp": "2023-03-15T15:26:37.527Z",
  "namespace": "multi-az/frontend",
  "metricName": "5xx",
  "dimensions": [
    { "Name": "Region", "Value": "us-east-1" },
    { "Name": "Controller", "Value": "Home" },
    { "Name": "Action", "Value": "Index" }
  ],
  "period": 60,
  "stat": "Sum",
  "unit": "Count",
  "chiSquareMetricName": "multi-az/chi-squared",
  "azs": [ "use1-az2", "use1-az4", "use1-az6" ]
}
```

Data digunakan untuk menentukan data umum yang diperlukan untuk mengambil yang sesuaiCloudWatchmetrik (seperti namespace, nama metrik, dan dimensi) dan kemudian mempublikasikan hasil chi-squared untuk setiap Availability Zone. Kode dalam fungsi Lambda terlihat seperti berikut menggunakan Python 3.9. Pada tingkat tinggi, ia mengumpulkan yang ditentukanCloudWatchmetrik untuk menit sebelumnya, menjalankan uji chi-squared pada data itu, dan kemudian menerbitkanCloudWatchmetrik tentang hasil pengujian untuk setiap Availability Zone yang ditentukan.

```
import os
import boto3
import datetime
import copy
import json
```

```
from datetime import timedelta
from scipy.stats import chisquare
from aws_embedded_metrics import metric_scope

cw_client = boto3.client("cloudwatch", os.environ.get("AWS_REGION", "us-east-1"))

@metric_scope
def handler(event, context, metrics):
    metrics.set_property("Event", json.loads(json.dumps(event, default = str)))
    time = datetime.datetime.strptime(event["timestamp"], "%Y-%m-%dT%H:%M:%S.%fZ")

    # Round down to the previous minute
    end: datetime = roundTime(time)

    # Subtract a minute for the start
    start: datetime = end - timedelta(minutes = 1)

    # Get all the metrics that match the query
    results = get_all_metrics(event, start, end, metrics)
    metrics.set_property("MetricCounts", results)

    # Calculate the chi squared result
    chi_sq_result = chisquare(list(results.values()))
    expected = sum(list(results.values())) / len(results.values())
    metrics.set_property("ChiSquaredResult", chi_sq_result)

    # Put the chi square metrics into CloudWatch
    put_all_metrics(event, results, chi_sq_result[1], expected, start, metrics)

def get_all_metrics(detail: dict, start: datetime, end: datetime, metrics):
    """
    Gets all of the error metrics for each AZ specified
    """
    metric_query = {
        "MetricDataQueries": [
        ],
        "StartTime": start,
        "EndTime": end
    }

    for az in detail["azs"]:

        dim = copy.deepcopy(detail["dimensions"])
        dim.append({"Name": "AZ-ID", "Value": az})
```

```
        query = {
            "Id": az.replace("-", "_"),
            "MetricStat": {
                "Metric": {
                    "Namespace": detail["namespace"],
                    "MetricName": detail["metricName"],
                    "Dimensions": dim
                },
                "Period": int(detail["period"]),
                "Stat": detail["stat"],
                "Unit": detail["unit"]
            },
            "Label": az,
            "ReturnData": True
        }

        metric_query["MetricDataQueries"].append(query)

    metrics.set_property("GetMetricRequest", json.loads(json.dumps(metric_query,
default=str)))
    next_token: str = None
    results = {}

    while True:
        if next_token is not None:
            metric_query["NextToken"] = next_token

        data = cw_client.get_metric_data(**metric_query)

        if next_token is not None:
            metrics.set_property("GetMetricResult::" + next_token,
json.loads(json.dumps(data, default = str)))
        else:
            metrics.set_property("GetMetricResult", json.loads(json.dumps(data, default
= str)))

        for item in data["MetricDataResults"]:
            key = item["Id"].replace("_", "-")
            if key not in results:
              results[key] = 0

            results[key] += sum(item["Values"])
```

```
        if "NextToken" in data:
            next_token = data["NextToken"]

        if next_token is None:
            break

    return results

def put_all_metrics(detail: dict, results: dict, chi_sq_value: float, expected: float,
 timestamp: datetime, metrics):
    """
    Adds the chi squared metric for all AZs to CloudWatch
    """
    farthest_from_expected = None
    if len(results) > 0:
        keys = list(results.keys())
        farthest_from_expected = keys[0]

        for key in keys:
            if abs(results[key] - expected) > abs(results[farthest_from_expected] -
 expected):
                farthest_from_expected = key

    metric_query = {
        "Namespace": detail["namespace"],
        "MetricData": []
    }

    for az in detail["azs"]:
        dim = copy.deepcopy(detail["dimensions"])
        dim.append({"Name": "AZ-ID", "Value": az})

        query = {
            "MetricName": detail["chiSquareMetricName"],
            "Dimensions": dim,
            "Timestamp": timestamp,
        }

        if chi_sq_value <= 0.05 and az == farthest_from_expected:
            query["Value"] = 1
        else:
            query["Value"] = 0

        metric_query["MetricData"].append(query)
```

```
    metrics.set_property("PutMetricRequest", json.loads(json.dumps(metric_query,
 default = str)))

    cw_client.put_metric_data(**metric_query)

def roundTime(dt=None, roundTo=60):
   """Round a datetime object to any time lapse in seconds
   dt : datetime.datetime object, default now.
   roundTo : Closest number of seconds to round to, default 1 minute.
   """
   if dt == None : dt = datetime.datetime.now()
   seconds = (dt.replace(tzinfo=None) - dt.min).seconds
   rounding = (seconds+roundTo/2) // roundTo * roundTo
   return dt + datetime.timedelta(0,rounding-seconds,-dt.microsecond)
```

Anda kemudian dapat membuat alarm per AZ. Contoh berikut adalah untuk`use1-az2`dan alarm untuk tiga, titik data satu menit berturut-turut yang memiliki nilai maksimum sama dengan 1 (1 adalah metrik yang diterbitkan ketika uji chi-kuadrat menentukan kemiringan signifikan secara statistik dalam tingkat kesalahan).

```
{
    "Type": "AWS::CloudWatch::Alarm",
    "Properties": {
        "AlarmName": "use1-az2-chi-squared",
        "ActionsEnabled": true,
        "OKActions": [],
        "AlarmActions": [],
        "InsufficientDataActions": [],
        "MetricName": "multi-az/chi-squared",
        "Namespace": "multi-az/frontend",
        "Statistic": "Maximum",
        "Dimensions": [
            {
                "Name": "AZ-ID",
                "Value": "use1-az2"
            },
            {
                "Name": "Action",
                "Value": "Index"
            },
```

```
            {
                "Name": "Region",
                "Value": "us-east-1"
            },
            {
                "Name": "Controller",
                "Value": "Home"
            }
        ],
        "Period": 60,
        "EvaluationPeriods": 3,
        "DatapointsToAlarm": 3,
        "Threshold": 1,
        "ComparisonOperator": "GreaterThanOrEqualToThreshold",
        "TreatMissingData": "missing"
    }
}
```

Anda juga dapat membuatm-of-nalarm dan menggabungkan kedua alarm ini bersama-sama dengan alarm komposit. Anda juga perlu membuat alarm yang sama untuk setiap kombinasi Controller/Action atau microservice yang Anda miliki di setiap Availability Zone. Terakhir, Anda dapat menambahkan alarm komposit chi-squared ke alarm khusus Availability Zone untuk setiap kombinasi Controller/Action seperti yang ditunjukkan padaDeteksi kegagalan menggunakan deteksi outlier.

Kontributor

Kontributor untuk dokumen ini meliputi:

- Michael Haken, Arsitek Solusi Utama, Layanan Web Amazon

Revisi dokumen

Untuk diberitahu tentang pembaruan whitepaper ini, berlangganan RSS feed.

Perubahan	Deskripsi	Tanggal
Whitepaper diperbarui	Diperbarui dengan panduan observabilitas tambahan dan menggunakan fitur pergeseran zonal baru.	11 Juli 2023
Publikasi awal	Whitepaper pertama kali diterbitkan.	2 Maret 2022

> **Note**
>
> Untuk berlangganan pembaruan RSS, Anda harus mengaktifkan plug-in RSS untuk browser yang Anda gunakan.

Pemberitahuan

Pelanggan bertanggung jawab untuk membuat penilaian independen mereka sendiri terhadap informasi dalam dokumen ini. Dokumen ini: (a) hanya untuk tujuan informasi, (b) mewakili arusAWSpenawaran dan praktik produk, yang dapat berubah tanpa pemberitahuan, dan (c) tidak menciptakan komitmen atau jaminan apa pun dariAWSdan afiliasinya, pemasok atau pemberi lisensinya.AWSproduk atau layanan disediakan “sebagaimana adanya” tanpa jaminan, pernyataan, atau kondisi apa pun, baik tersurat maupun tersirat. Tanggung jawab dan kewajibanAWSkepada pelanggannya dikendalikan olehAWSperjanjian, dan dokumen ini bukan bagian dari, juga tidak memodifikasi, perjanjian apa pun antaraAWSdan pelanggannya.

Glosarium AWS

Lihat terminologi AWS terbaru di AWS glosarium dalam Referensi Glosarium AWS.

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