Networking & Content Delivery

Creating Disaster Recovery Mechanisms Using Amazon Route 53

by Joe Chapman and Gavin McCullagh | on 12 OCT 2022 | in Advanced (300), Amazon Route 53, Architecture, Networking & Content Delivery | Permalink | Comments | Share

For many applications, deploying to multiple Availability Zones in a single AWS Region and aligning with the AWS Well-Architected Framework offers the right balance of availability, simplicity, and affordability. However, some customers may have regulatory or other reasons that require expanding to additional Regions in an active-active or active-standby disaster recovery (DR) configuration.

In a DR scenario, you need a simple and effective approach to mitigate failures and then return to normal operations. To be fast and reliable, this process must be straightforward, require minimal steps, and be practiced regularly. To failover, many companies change their DNS records. This blog post describes how to use updates to DNS for effective disaster recovery, and highlights principles and best practices you should adopt when following this approach.

Modern DNS services, like Amazon Route 53, offer health checks and failover records that you can use to simplify and strengthen your DR plan. We'll start by outlining how AWS services provide reliability using control planes and data planes, then share high-level design principles for creating a failover mechanism. Finally, we'll explain the features of Route 53 that can make your DR approach more effective.

Control planes and data planes

Most AWS services include a *data plane*, which provides the service's core functionality, and a *control plane*, which enables you to create, update, and delete resources. Regional AWS services, such as Amazon Elastic Compute Cloud (Amazon EC2), have Regional service endpoints that connect you to the service's control plane in a specific AWS Region. However, some AWS services are distributed across many locations around the globe, such as Amazon CloudFront, AWS Identity and Access Management (IAM), and Route 53. The control plane for these global services are only available in the US East (N. Virginia) Region (us-east-1). Similarly, AWS Global Accelerator and Amazon Route 53 Application Recovery Controller (Route 53 ARC) are global services with control planes located in the US West (Oregon) Region (us-west-2). To create and configure resources on these global services, you must use the API endpoint in the Region where their control plane is located.

The Route 53 data planes answer DNS queries, and perform and evaluate health checks. They are globally distributed and designed for a 100% availability service level agreement (SLA). The Route 53 management APIs and consoles where you create, update, and delete Route 53 resources run on control planes that are designed to prioritize the strong consistency and durability that you need when managing DNS. To achieve this, the control planes are located in a single Region, US East (N. Virginia). While both systems are built to be very reliable, the control planes are not included in the SLA. There could be rare events in which the data plane's resilient design allows it to maintain availability while the control planes do not. For this reason, we recommend disaster recovery and failover mechanisms use data plane functions to provide the best possible reliability.

For more information about data planes, control planes, and how AWS builds services to meet high availability targets, see the Static stability using Availability Zones paper and the Amazon Builders' Library.

Since the Route 53 has a data plane designed for a 100% availability SLA, good failover mechanisms should rely on the data plane and not the control plane. Later in this post, we'll review ways that you can make targeted changes to Route 53 that don't involve the control plane. But first let's look at some best practices to follow when creating an effective DR failover mechanism.

Designing a reliable failover mechanism

When you design a disaster recovery plan, it's important to consider how you will make sure that the failover mechanism is resilient and available when you need it. These mechanisms should be simple, have the fewest dependencies possible, and be thoroughly tested. While it may seem obvious, it's easy to overlook dependencies that work during a test, but fall into a failed state during a disaster.

To create an analogy from the physical world, you need a backup power generator most when utility power isn't available. Therefore, a reliable failover mechanism must not depend on utility power to start up nor to fail over power to itself. In addition, because utility power is typically available when you run tests with your failover setup, you must be sure that you don't have a hidden dependency on that utility power.

Consider the following principles as you build a mechanism for failover. Then in the next sections, we'll look at Route 53-based solutions that incorporate these principles.

- **Use data plane functions.** Design your failover mechanisms to use data plane functions for the greatest reliability and fault tolerance. For example, choose mechanisms that use Route 53 health checks (data plane functions) rather than updating DNS records in hosted zones or health check configurations (control plane functions).
- **Control failover from your standby Region.** Multi-Region failover is powerful because we build each AWS Region to be independent and isolated from other Regions. When you design mechanisms to fail over from an impaired application in one Region, make sure that you do not need to make changes in the impaired Region. Instead, initiate failover by making changes in the healthy standby Region.
- **Understand and reduce dependencies to make failover more reliable**. Map out all internal and external dependencies that you'll require to initiate and coordinate a successful failover. Minimizing the number of dependencies in a failover process reduces potential points of error, making it more reliable. For example, use AWS CLI or SDK calls in your failover process instead of the AWS Management Console, as the console is an added UI component that introduces further dependencies.
- **Pre-provision critical components**. Avoid control plane dependencies for creating critical components during an event by pre-provisioning resources to handle full or partial loads. Doing so helps achieve disaster recovery objectives (RTO and RPO), but requires balancing cost against recovery objectives. Whenever possible, creating and scaling resources beforehand also helps prevent capacity constraints in the standby Region.
- **Review authentication methods**. Think through how you authenticate for managing your AWS services. For example, an AWS Organization can only support AWS Single Sign-On (AWS SSO) in one Region at a time. If AWS SSO is impaired in that Region, then you'll have to use separate credentials that bypass your typical process and don't rely on AWS SSO. A break glass IAM user would meet this requirement, as IAM authentication relies on the globally-available IAM data plane.

- **Test regularly**. To help instill confidence that your failover mechanism will work when there's a disaster, it's important that you test the process regularly. Testing verifies that your DR plans are up-to-date and can be carried out successfully when needed.

Now let's look at three ways that you can use different Route 53 features to follow these principles and create a failover mechanism as part of a disaster recovery plan.

Route 53 health checks

Route 53 health checks make requests to your resources from eight different AWS Regions to verify their availability. We have integrated health checks into the globally distributed Route 53 data plane for extreme reliability. A Route 53 health check can monitor the following:

- A TCP, HTTP, or HTTPS endpoint
- The status of other Route 53 health checks
- The status of an Amazon CloudWatch alarm
- The state of a Route 53 Application Recovery Controller routing control

Route 53 health checks return as healthy as long as at least 18% of global health checkers report that the endpoint is responding healthy. By default, at least 3 out of 16 checkers (18.75%) must report as healthy for Route 53 to consider the endpoint healthy.

These safeguards help to prevent erroneous failures. There must be consistent, repeated failures to turn a health check unhealthy. A health check won't fail if it is flapping between healthy and unhealthy, or if a small subset of the checkers report as unhealthy, for example, because of local network issues. You can set both the health checker Regions and check failure thresholds.

You can configure DNS to automatically fail over by using a failover routing policy with health checks. In this scenario, Route 53 failover records return the primary resource for a DNS query when the resource is healthy. If the resource becomes unhealthy and the secondary is healthy, Route 53 automatically updates and responds with the secondary record (that is, it fails over). If both the primary and secondary records are unhealthy, it returns the primary record.

Figure 1 shows an active-passive setup with Route 53 DNS failover. An issue in the primary Region resulting in a failed health check (HC-Primary) initiates an automatic failover to the secondary record.

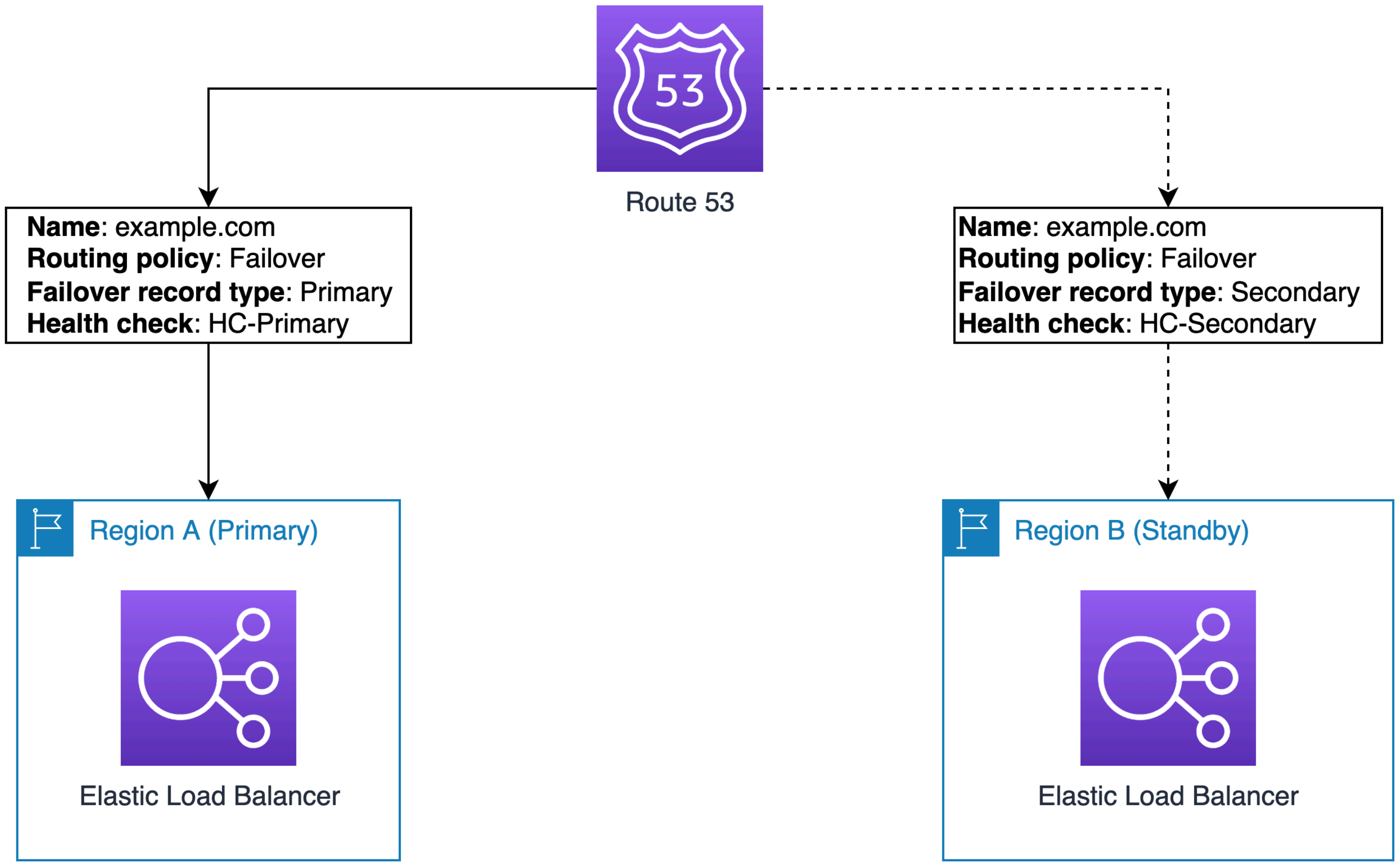
Route 53
Name: example.com
Routing policy: Failover
Failover record type: Primary
Health check: HC-Primary
Name: example.com
Routing policy: Failover
Failover record type: Secondary
Health check: HC-Secondary
Region A (Primary)
Elastic Load Balancer
Region B (Standby)
Elastic Load Balancer

Figure 1. Simple automated DNS failover with Route 53 health checks.

Automatic recovery through health checks is simple to configure and highly reliable, with no dependencies on the Route 53 control plane. When a health check changes status, Route 53 updates the record associated with the health check using the Route 53 data plane.

However, if you use health checks to fail over automatically, you won't be able to force a failover if there are intermittent issues. In addition, you should not depend on the Route 53 heath check APIs to force a failover as that would add a control plane dependency.

The direct health checking of resources described here fits best in active-active architectures. In these scenarios, the job of the health checks is simply to remove one failing endpoint from a group of active endpoints. If the endpoint later begins passing health checks, it should be safe to bring it back into service automatically. In this setup, failover and reversion are both automatic and require no user intervention.

However, for most disaster recovery plans, an active-standby architecture is more common. A successful failover is often part of a sequence of steps, such as a database failover and scaling up standby capacity. In a multi-step recovery process, you need more direct control over failover and fail back than automated health checking provides.

While Route 53 health checks are often used to directly probe the health of resources, you can also use them as a signaling system. By configuring health checks to check other web services that you control, such as the existence of files in S3 buckets, you can reliably signal the Route 53 DNS data plane to initiate failover. The next two sections explain how to take advantage of this so you can control the failover directly.

Route 53 Application Recovery Controller

Amazon Route 53 Application Recovery Controller (Route 53 ARC) gives you full control of when highly available applications fail over. It provides two features to manage and successfully fail over: readiness checks and routing controls. Readiness checks help make sure your standby systems are ready to fail over to, and routing controls enable you to initiate a failover, manually or programmatically, using a highly available data plane API without any dependency on the Route 53 control plane.

A Route 53 ARC routing control is an on/off switch that changes the state of a Route 53 health check, which updates a DNS entry and redirects traffic, for example, from a primary to a standby deployment.

We host your routing controls on a dedicated cluster that creates a data plane across five geographically-isolated AWS Regions. The cluster operates on a quorum model, which means that the cluster and routing controls continue to function and provide failover control even if up to two Regional endpoints are unavailable. Each cluster includes five independent API endpoints, one in each Region, and you can update a routing control state by using any of the five endpoints.

Route 53 ARC routing controls have the least number of dependencies, and each dependency is extremely reliable on its own. When you make changes to routing control states, you rely on the three criteria below that are highly unlikely to fail:

- At least three of the five endpoints are available and take part in the quorum.
- You have working IAM credentials and can authenticate against a working Regional cluster endpoint.
- The Route 53 data plane is healthy (this data plane is designed to meet a 100% availability SLA).

When you update a routing control state, the change propagates through Route 53 health checks and then to your DNS record in Route 53. Since Route 53 ARC's routing controls are designed to be highly available and there are no control plane API operations involved, the change is executed reliably.

Routing controls also include configurable safety rules that enable you to define guard rails for failover operations. With safety rules, you can define guard rails explicitly so that the cluster enforces them and refuses any state change that breaks a rule. Some examples are the following:

- In an *active-standby* architecture, you might have a rule that only one endpoint (either active or standby) can be enabled and in service at a time.
- In an *active-active* architecture, you might have a rule that limits the number of application replicas that are taken offline at a time. This helps prevent different operators or automation components from reducing capacity below safe limits. Changes are processed in the order that they are requested, with checks to make sure that the required number of replicas are still in service.

Route 53 ARC readiness checks audit resource quotas, capacity, and other configurations on your application replicas. These checks help keep your application configurations aligned, in case you need to fail over to a replica. For example, in active-passive architectures, you might find that your primary application must be scaled up or otherwise changed. Readiness checks provide a reliable mechanism to make sure that when changes are made to your primary, your standby is also scaled up.

Figure 2 illustrates a Route 53 ARC deployment with an application in an active-standby configuration:

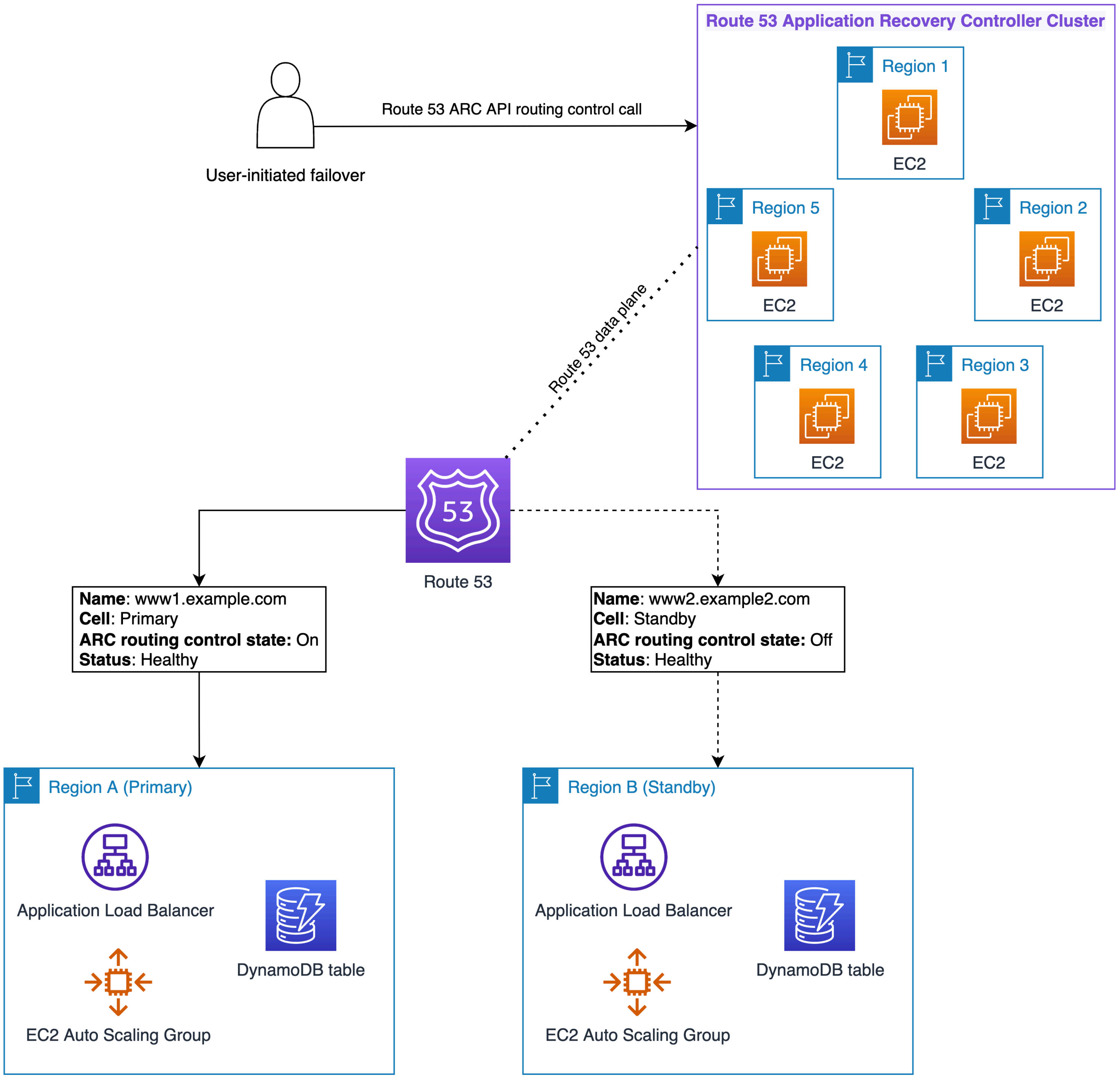

Figure 2. Route 53 Application Recovery Controller setup with two application replicas

You can see that Region A (Primary) and Region B (Standby) have identical applications with an Application Load Balancer, EC2 Auto Scaling group, and a DynamoDB table. In addition, note the following:

- We've created readiness checks that audit the failover replica in the standby Region on an ongoing basis.
- We've created EventBridge rules to notify us if individual readiness checks return NOT READY values.

- If an issue arises with the application in Region A, we can make a routing control API call to any of the five Route 53 ARC cluster data plane endpoints in Region 1-5. When we update the routing control state, that changes the health check status on the Route 53 resource record for our application, and moves traffic to Region B.

After you've set up your application with Route 53 ARC routing controls for failover, make sure that you're ready when there's an unexpected event. For example, you should plan to use routing control API operations with the highly available Route 53 ARC data plane to get and update routing control states for production updates. You should also avoid dependencies on control planes, such as the Route 53 ARC console or the recovery control configuration API operations, which are only available in the US West (Oregon) Region (us-west-2).

For full guidance about preparing to use Route 53 ARC in a disaster scenario, see Best practices in the Route 53 ARC Developer Guide.

Route 53 ARC offers the most feature-rich failover solution of those discussed in this post. However, in some cases it may be more complex to set up, and the multi-Region Route 53 ARC cluster might cost more than other solutions.

Let's next look at a simple and cost-effective way of using independent Route 53 health checks to initiate a failover from a standby Region.

Standby takes over primary

Our third recommendation for failover using Route 53 features is a process we call "standby takes over primary" (STOP). This strategy depends on a healthy standby Region and application. With this solution, you use a resource (a health check) in the standby Region to control the failover. This allows you to initiate a failover without depending on any resources in the primary Region.

At a minimum, this solution uses one health check, which checks the status of a resource in the standby Region. Let's look at an example of this architecture.

We start with a Route 53 health check, *HC-Initiate-Disaster-Recovery*, that is associated with the Route 53 DNS record for our application in the primary Region (Region A). This health check looks for a condition that enables an operator (or automation) to declare the primary application "unhealthy" and initiates failover. The condition is something simple; in our case, we use the presence or absence of a file that we create or delete in the standby Region.

Specifically, the health check looks for a file on a public S3 website:
`https://<FAILOVER-BUCKET>.s3.<STANDBY-REGION>.amazonaws.com/initiate-failover.html`
If the file doesn't exist, the (inverted) health check result is "healthy" and the failover doesn't happen.

We use an inverted health check to safeguard against accidentally initiating a failover if the resource in the standby Region fails. The health check is associated with a DNS record in the *primary* Region (the application), but we configure the inverted health check to probe the health of a resource in the *standby* Region (the file), which we use to control the health check's state.

To initiate a failover, we create the S3 file that the health check looks for: “initiate-failover.html”. This tells the health check that the primary Region is “unhealthy”, causing it to switch traffic from the primary Region (Region A) to the secondary Region (Region B).

Figure 3 illustrates the configuration of Route 53 health checks, routing policies, and failover records for this “standby takes over primary” failover strategy:

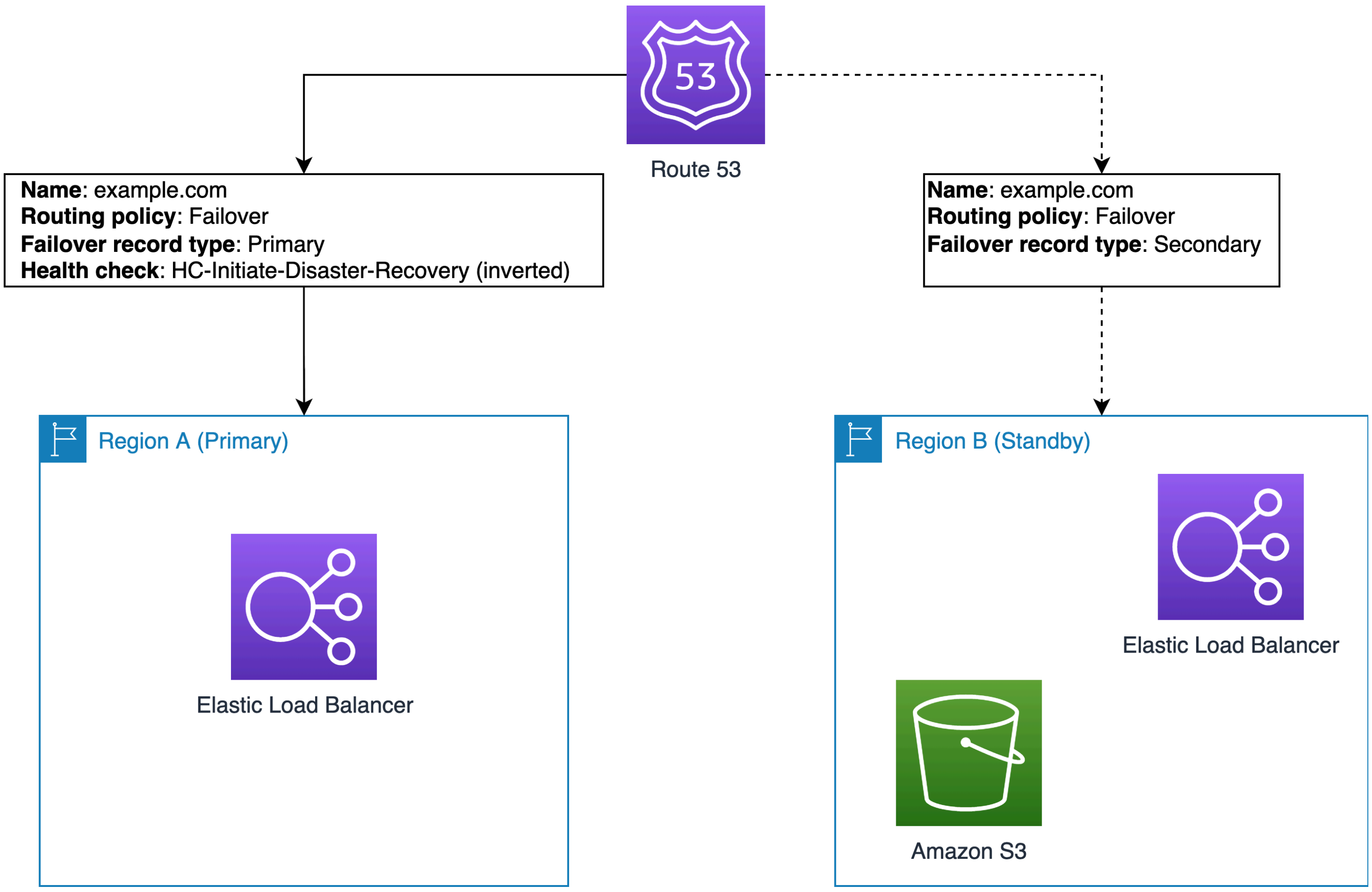
Route 53
Name: example.com
Routing policy: Failover
Failover record type: Primary
Health check: HC-Initiate-Disaster-Recovery (inverted)
Name: example.com
Routing policy: Failover
Failover record type: Secondary
Region A (Primary)
Elastic Load Balancer
Region B (Standby)
Elastic Load Balancer
Amazon S3

Figure 3. Standby takes over primary failover strategy

The health check has the following configuration:

- **Health check: HC-Initiate-Disaster-Recovery**
  - Associated with the DNS record of the application in the primary Region.
  - Checks a resource located in the standby Region.
  - Inverted health check is always healthy if the response is a HTTP 4xx, 5xx, or timeout. This makes sure that traffic continues to go toward the primary Region.
  - An operator-controlled initiation activity (such as creation of a file) updates the resource to return HTTP 2xx or 3xx to initiate a failover.

To fail over within a private context, one alternative to the S3 file trigger is to use an Amazon CloudWatch metric alarm. With this option, we create an alarm that watches a specific CloudWatch metric in the standby Region, and then create a Route 53 health check that monitors the CloudWatch alarm. As with the S3 option, we use an inverted health check to ensure that the standby Region only takes over when the standby is healthy and posts a specific metric value.

Using a CloudWatch alarm as the failover trigger enables us to fail over completely within an Amazon Virtual Private Cloud (VPC). For example, we can call CloudWatch on a VPC endpoint, create DNS entries for health checks in a Route 53 private hosted zone, and then perform failover, in private, within the VPC.

We recommend that, if possible, you align your failover initiation activity with existing dependencies for your standby application. For example, if your standby application already depends on read/write access to S3, you won't be adding a new dependency by using an initiation activity that puts a file that you create in an S3 bucket. Alternatively, if your standby application depends on DynamoDB, you might choose to create a health check handler in your application that responds based on a record that you update in a DynamoDB table.

You can easily integrate your failover initiation activity into your existing failover automation if, for example, you already use an AWS Systems Manager automation document to prepare resources for failover. Add a new action to the end of the document that invokes the failover initiation activity, and then an operator can run the automation to both prepare the resources and to fail over.

When your primary stack has recovered, you have two options to revert. If you want to fail back to the primary Region, you revert the change you made to cause Route 53 to switch back. Alternatively, if you prefer to avoid the impact of a second failover, you can update the Route 53 record sets to swap the failover record set roles of primary and standby.

Because this solution uses Route 53 health checks and the failover routing type, it includes the controls and safety checks of those Route 53 features. In addition, using this strategy means minimal costs, because it primarily relies on Route 53 health check pricing. If you need to audit your readiness for failover regularly (which we recommend), Route 53 ARC readiness checks also work with this failover pattern.

Summary

In this blog post, we described three disaster recovery strategies that you can consider by using Amazon Route 53 features to fail over quickly and reliably, to minimize downtime from an event.

We discussed the tradeoffs that come with each strategy. Route 53 health checks are globally distributed and work reliably even during significant service events. Route 53 Application Recovery Controller's data plane is multi-Region and can tolerate the failure of up to two of its Regional endpoints. Finally, the "standby takes over primary" approach ensures that you can fail over without any dependency on your primary Region.

Each of these strategies give you options to consider as you create disaster recovery mechanisms that are resilient and limit dependencies for the systems that you rely on during failures.

Related Posts

- Disaster Recovery of Workloads on AWS: Recovery in the Cloud AWS Whitepaper
- Reliability Pillar AWS Well-Architected Framework
- Building highly resilient applications using Amazon Route 53 Application Recovery Controller, Part 1: Single-Region stack
- Building highly resilient applications using Amazon Route 53 Application Recovery Controller, Part 2: Multi-Region stack

TAGS: Amazon Route 53, Disaster Recovery, Multi-Region

Comments

0 Comments 4 **Prashanth ▾**

Start the discussion…

♡ 1 **Share** **Best** **Newest** **Oldest**

Be the first to comment.