**9**

**Architectural Reliability Considerations**

Application reliability is one of the essential aspects of architecture design. A reliable application helps to win customer trust by making it available whenever the customer needs it. As all kinds of businesses are now online, high availability has become one of the mandatory criteria for online applications. Users want to browse your application anytime and complete tasks such as shopping and banking at their convenience. Reliability is one of the essential recipes for any business to be successful.

**Reliability** means the ability of the system to recover from failure. It's about making your application fault-tolerant so that it can recover without impacting the customer experience. A reliable system should be able to recover from any infrastructure failure or server failure. Your system should be prepared to handle any situation that could cause disruption.

In this chapter, you will learn various design principles applicable to making your solution reliable. When assessing reliability, you need to consider every component of the architecture. You will understand how to choose the right technology to ensure your architecture's reliability at every layer. You will learn the following best practices for reliability in this chapter:

* Design principles for architectural reliability
* Technology selection for architectural reliability
* Improving reliability with the cloud

By the end of this chapter, you will have learned about various disaster recovery techniques and data replication methods to ensure the high availability of your application and the continuation of business processes.

**Design principles for architectural reliability**

The goal of reliability is to contain the impact of any failure in the smallest area possible. By preparing your system for the worst-case scenarios, you can implement various mitigation strategies for the different components of your infrastructure and applications.

Before a failure occurs, you should thoroughly test your recovery procedures.

The following are the standard design principles that help you to strengthen your system's reliability. You will find that all reliability design principles are closely related and complement each other.

**Making systems self-healing**

System failure needs to be predicted in advance, and in the case of failure incidence, you should have an automated response for system recovery, called system self-healing. Self-healing is the ability of the solution to recover from failure automatically. A self-healing system detects failure proactively and responds to it gracefully with minimal customer impact. Failure can happen in any layer of your entire system, including hardware failure, network failure, or software failure. Usually, data center failure is not an everyday event, and more granular monitoring is required for frequent failures such as database connection and network connection failures. The system needs to monitor the failure and act to recover.

To make system self-healing, first, you need to identify **Key Performance Indicators** (**KPIs**) for your application and business. These KPIs may include the number of requests served per second or page load latency for your website at the user level. You can define maximum CPU utilization at the infrastructure level, such as it should not go above 60% and memory utilization *should not go beyond 50%* of the total available **Random-Access Memory** (**RAM**).

As you defined your KPIs, you should put the monitoring system to track failures and notify you as your KPIs reach the threshold. You should apply automation around monitoring so that the system can self-heal in the event of any incidents. For example, add more servers when CPU utilization reaches near 50%—proactive monitoring helps prevent failures.

**Applying automation**

Automation is the key to improving your application's reliability. Try to automate everything from application deployment and configuration to the overall infrastructure. Automation provides you with agility where your team can move fast and experiment more often. You can replicate the entire system infrastructure and the environment with a single click to try a new feature.

You can plan the auto-scaling of your application based on a schedule, for example, an e-commerce website may have more user traffic on weekends. You can also automate scaling based on the user request volume to handle the unpredictable workload. Use automation to launch independent and parallel jobs that will provide greater accuracy when combined with the results from the individual job.

Frequently, you need to apply the same configuration that you have on your development environment to **Quality Assurance** (**QA**) environments. There may be multiple QA environments for each testing stage, which includes functional testing, UAT, and stress testing environments. Often, a QA tester discovers a defect caused by wrongly configured resources, which could introduce a further delay in the test schedule. Most importantly, you cannot afford to have a configuration error in production servers.

To reproduce precisely the same configuration, you may need to document step-by-step configuration instructions. Repeating the same steps to configure each environment manually can be error-prone. There is always a chance of human error, such as a typo, for example, in a database name. The solution to this challenge is to automate these steps by creating a script. The automation script itself can be the documentation.

As long as the script is correct, it is more reliable than manual configuration. It is undoubtedly reproducible. Detecting unhealthy resources and launching replacement resources can be automated, and you can notify the IT operations team when resources are changed. Automation is a fundamental design principle that needs to apply everywhere in your system.

**Creating a distributed system**

Monolithic applications have low reliability when it comes to system uptime, as a tiny issue in a particular module can bring down the entire system. Dividing your application into multiple small services reduces the *impact area*. One part of the application shouldn't impact the whole system, and the application can continue to serve critical functionality. For example, in an e-commerce website, an issue with the payment service should not affect the customer's ability to place orders, as payment can be processed later.

At the service level, scale your application horizontally to increase system availability. Design a system to use multiple smaller components working together rather than a single monolithic system to reduce the impact area. In a distributed design, requests are handled by different system components, and the failure of one component doesn't impact the functioning of other parts of the system. For example, on an e-commerce website, the failure of warehouse management components will not impact the customer placing the order.

However, the communication mechanism can be complicated in a distributed system. You need to take care of system dependencies by utilizing the circuit breaker pattern. As you learned regarding the circuit breaker pattern in *Chapter 6*, *Solution Architecture Design Patterns*, the basic idea is simple. You wrap a protected function call in a circuit breaker object, which monitors for failures and takes automated action to mitigate it.

**Monitoring and adding capacity**

Resource saturation is the most common reason for application failure. Often, you will encounter the issue where your applications start rejecting requests due to CPU, memory, or hard disk overload. Adding more resources is not always a straightforward task, as you should have additional capacity available when needed.

In a traditional on-premises environment, you need to calculate server capacity based on the assumption in advance. Workload capacity prediction becomes more challenging for a business such as a shopping website and any online business. Online traffic is very unpredictable and fluctuates heavily driven by global trends. Usually, procuring hardware can take anywhere between 3 to 6 months, and it's tough to guess capacity in advance. Ordering excess hardware will incur an extra cost as a resource is sitting idle, and a lack of resources will cause the loss of business due to application unreliability.

You need an environment where you don't need to guess capacity, and your application can scale on demand.

A public cloud provider such as **Amazon Web Services** (**AWS**) provides **Infrastructure as a Service** (**IaaS**), facilitating the on-demand availability of resources.

In the cloud, you can monitor system supply and demand. You can automate the addition or removal of resources as needed. It allows you to maintain the level of resources that will satisfy demand without over-provisioning or under-provisioning.

**Performing recovery validation**

When it comes to infrastructure validation, most of the time, organizations focus on validating a happy path where everything is working. Instead, you should validate how your system fails and how well your recovery procedures work. Validate your application, assuming everything fails all the time. Don't just expect that your recovery and failover strategies will work. Make sure to test them regularly, so you're not surprised if something does go wrong.

A simulation-based validation helps you to uncover any potential risks. You can automate a possible scenario that could cause your system to fail and prepare an incident response accordingly. Your validation should improve application reliability in such a way that nothing will fail in production.

Recoverability is sometimes overlooked as a component of availability. To improve the system's **Recovery Point Objective** (**RPO**) and **Recovery Time Objective** (**RTO**), you should back up data and applications along with their configuration as a machine image. You will learn more about RTO and RPO in the next section. Suppose a natural disaster makes one or more of your components unavailable or destroys your primary data source. In that case, you should be able to restore the service quickly and without losing data. Let's talk more about specific disaster recovery strategies to improve application reliability and associated technology choices.

**Technology selection for architectural reliability**

Application reliability often looks at the availability of the application to serve users. Several factors go into making your application highly available. However, **fault tolerance** refers to the built-in redundancy of an application's components. Your application may be highly available but not be 100% fault-tolerant. For example, if your application needs four servers to handle the user request, you divided them between two data centers for high availability. If one site goes down, your system is still highly available at 50% capacity, but it may impact user performance expectations. However, if you create equal redundancy in both sites with four servers each, your application will not be only highly available but will be 100% fault-tolerant.

Suppose your application is not 100% fault-tolerant. In that case, you want to add automated scalability, defining how your application's infrastructure will respond to increased capacity needs to ensure your application is available and performing within your required standards. To make your application reliable, you should be able to restore services quickly and without losing data. Going forward, we are going to address the recovery process as disaster recovery. Before going into various disaster recovery scenarios, let's learn more about the RTO/RPO and data replication.

**Planning the Recovery Time Objective and Recovery Point Objective**

Business applications need to define service availability in the form of a **Service-Level Agreement** (**SLA**). Organizations define SLAs to ensure application availability and reliability for their users. You may want to define an SLA, saying your application should be 99.9% available in a given year, or that the organization can tolerate downtime of 43 minutes per month, and so on. The defined SLA primarily drives the RPO and RTO for an application.

The RPO is the amount of data loss an organization can tolerate in a given period of time. For example, my application is acceptable if it loses 15 minutes' worth of data. For example, if you are processing customer orders every 15 minutes for fulfillment, then you can tolerate reprocessing that data in case of any system failure at order fulfillment application. The RPO helps to define a data backup strategy. The RTO is about application downtime and how much time the application should take to recover and function normally after an incidence of failure. The following diagram illustrates the difference between the RTO and RPO:

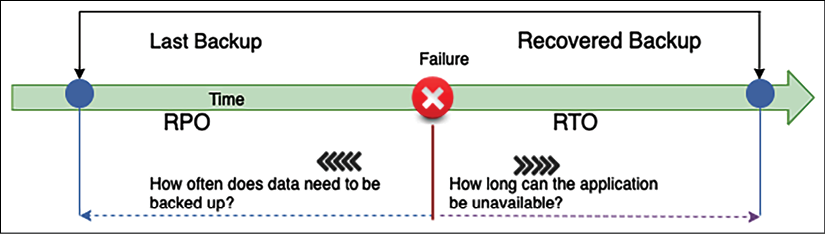


Figure 9.1: RTO and RPO

In the preceding diagram, suppose the failure occurs at 10 A.M., and you took the last backup at 9 A.M.; in the event of a system crash, you would lose 1 hour of data. When you restore your system, there is an hour's worth of data loss, as you have been taking data backups every hour.

In this case, your system RPO is 1 hour, as it can tolerate living with an hour's worth of data loss. In this case, the RPO indicates that the maximum data loss that can be tolerated is 1 hour.

If your system takes 30 minutes to restore to the backup and bring up the system, it defines your RTO as half an hour. This means the maximum downtime that can be tolerated is 30 minutes. The RTO is the time it takes to restore the entire system after a failure that causes downtime, which is 30 minutes in this case.

An organization typically decides on an acceptable RPO and RTO based on the user experience and financial impact on the business in the event of system unavailability. Organizations consider various factors when determining the RTO/RPO, including the loss of business revenue and damage to their reputation due to downtime. IT organizations plan solutions to provide effective system recovery as per the defined RTO and RPO. You can see now how data is the key to system recovery, so let's learn some methods to minimize data loss.

**Replicating data**

Data replication and snapshots are the key to disaster recovery and making your system reliable. Replication creates a copy of the primary data site on the secondary site. In the event of primary system failure, the system can fail over to the secondary system and keep working reliably. This data could be your file data stored in a **NAS drive**, **database snapshot**, or **machine image snapshot**. Sites could be two geo-separated on-premises systems, two separate devices on the same premises, or a physically separated public cloud.

Data replication is not only helpful for disaster recovery, but it can speed up an organization's agility by quickly creating a new environment for testing and development. Data replication can be synchronous or asynchronous.

**Synchronous versus asynchronous replication**

Synchronous replication creates a data copy in real time. Real-time data replication helps to reduce the RPO and increase reliability in the event of a disaster. However, it is expensive as it requires additional resources in the primary system for continuous data replication.

Asynchronous replication creates copies of data with some lag or as per the defined schedule. However, asynchronous replication is less expensive as it uses fewer resources compared to synchronous replication. You may choose asynchronous replication if your system can work with a longer RPO.

In terms of database technology such as Amazon RDS, synchronous replication is applied if we create an RDS with multiple Availability Zone failover. For read replicas, there is asynchronous replication, and you can use that to serve report and read requests.

As illustrated in the following architecture diagram, in synchronous replication, there is no lag of data replication between the master and standby instance of the database, while, in the case of asynchronous replication, there could be some lag while replicating the data between the master and replication instance:

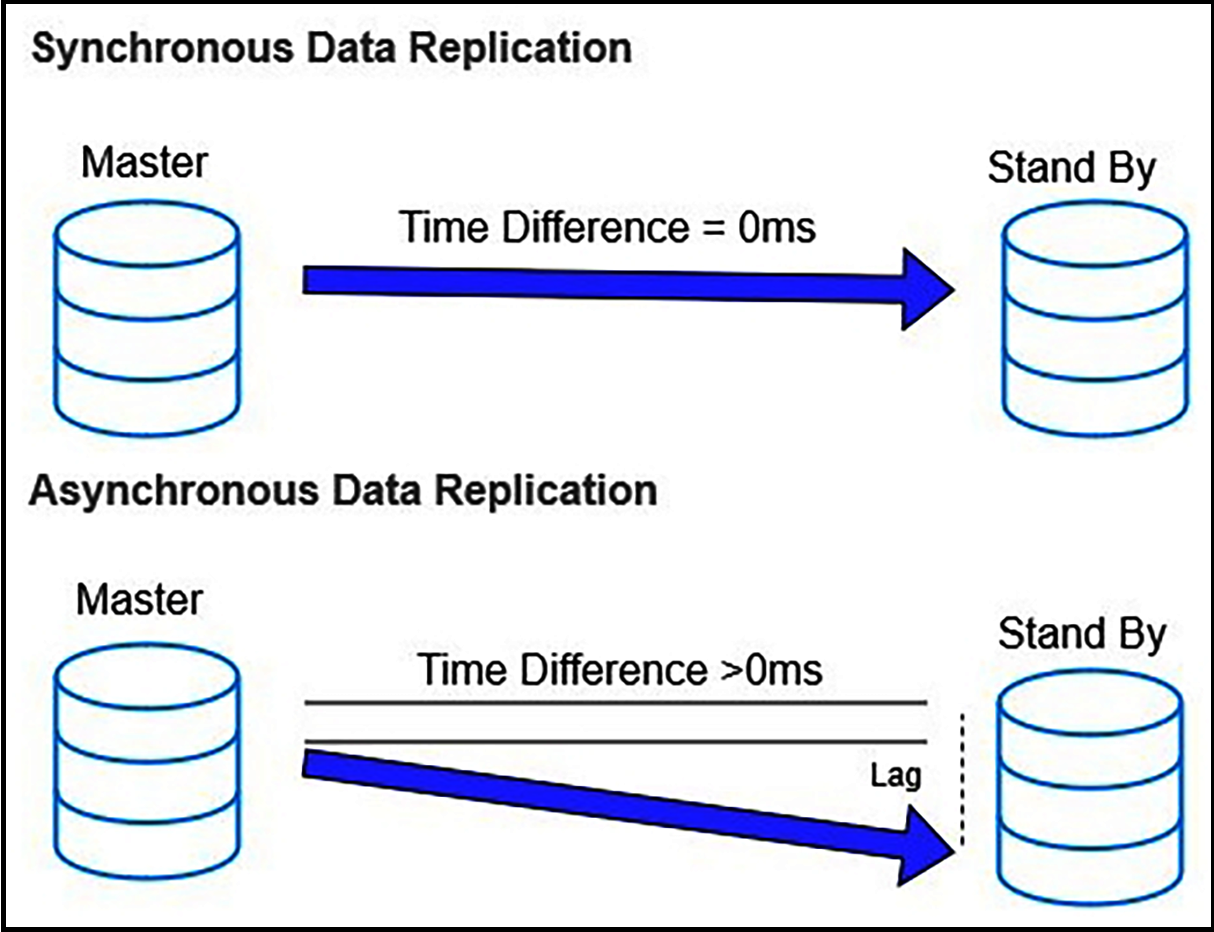


Figure 9.2: Synchronous and asynchronous data replication

Let's explore some methods of data replication for the synchronous and asynchronous approaches.

**Replication methods**

The replication method is an approach to extract data from the source system and create a copy for data recovery purposes. Different replication methods are available to store a copy of data as per the storage type for business process continuation. Replications can be implemented in the following ways:

* **Array-based replication**: In this, built-in software automatically replicates data. However, both the source and destination storage arrays should be compatible and homogeneous to replicate data. A storage array contains multiple storage disks in a rack.

Large enterprises use array-based replication due to the ease of deployment and the reduction in the compute power host system. You can choose array-based replication products such as HP Storage, EMC SAN Copy, and NetApp SnapMirror.

* **Network-based replication**: This can copy data between a different kind of heterogeneous storage array. It uses an additional switch or appliance between incompatible storage arrays to replicate data. In network-based replication, the cost of replication could be higher as multiple players come into the picture. You can choose from networked-based replication products such as NetApp Replication X and EMC RecoverPoint.
* **Host-based replication**: In this, you install a software agent on your host that can replicate data to any storage system such as NAS, SAN, or DAS. You can use a host-based software vendor, for example, Symantec, Commvault, CA, or Vision Solution. It's highly popular in **Small and Medium-Sized Businesses** (**SMBs**) due to lower upfront costs and heterogeneous device compatibility. However, it consumes more compute power as the agent needs to be installed on the host operating system.
* **Hypervisor-based replication**: This is VM-aware, which means copying the entire virtual machine from one host to another. As organizations mostly use virtual machines, it provides a very efficient disaster recovery approach to reduce the RTO. Hypervisor-based replication is highly scalable and consumes fewer resources than host-based replication. It can be carried out by native systems built into VMware and Microsoft Windows. You can choose a product such as Zerto to perform hypervisor-based replication or another product from various vendors.

Previously, in *Chapter 3*, *Attributes of the Solution Architecture*, you learned about scalability and fault tolerance. In *Chapter 6*, *Solution Architecture Design Patterns*, you learned about various design patterns to make your architecture highly available. Now, you will discover multiple ways to recover your system from failure and make it highly reliable.

**Planning disaster recovery**

**Disaster recovery** (**DR**) is about maintaining business continuation in the event of system failure. It's about preparing the organization for any possible system downtime and the ability to recover from it. DR planning covers multiple dimensions, including hardware and software failure. While planning for DR, always ensure you consider other operational losses, including power outages, network outages, heating and cooling system failures, physical security breaches, and other incidents, such as fires, floods, or human error.

Organizations invest effort and money in DR planning as per system criticality and impact. A revenue-generating application needs to be up all of the time as it significantly impacts company image and profitability. Such an organization invests lots of effort in creating their infrastructure and training their employees for a DR situation. DR is like an insurance policy that you have to invest in and maintain even when you don't utilize it, as in the case of unforeseen events, a DR plan will be a lifesaver for your business.

Bases of business criticality, such as software applications, can be placed on a *spectrum of complexity*. There are four DR scenarios, sorted from highest to lowest RTO/RPO as follows:

* Backup and restore
* Pilot light
* Warm standby
* Multi-site

As shown in the following diagram, in DR planning, as you progress with each option, your RTO and RPO will reduce while the cost of implementation increases. You need to make the right trade-off between RTO/RPO requirements and cost per your application reliability requirements:

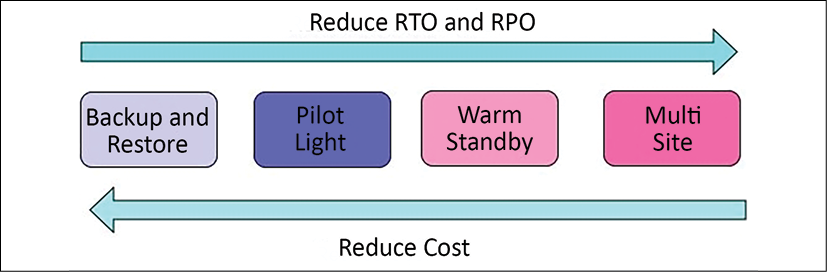


Figure 9.3: The spectrum of DR options

Let's explore each of the options mentioned above in detail with the technology choices involved. Note that public clouds such as AWS enable you to operate each of the preceding DR strategies cost-effectively and efficiently.

Business continuity is about ensuring critical business functions continue to operate or function quickly in the event of disasters. As organizations opt to use the cloud for DR plans, let's learn about the various DR strategies between on-premises environments and the cloud.

**Backup and restore**

Backup and restore is the lowest cost option but it results in a higher RPO and RTO. This method is simple to get started and highly cost-effective as you only need backup storage space. This backup storage could be a tape drive, hard disk drive, or network access drive. As your storage needs increase, adding and maintaining more hardware across regions could be a daunting task. One of the most cost-effective and straightforward options is to use the cloud as backup storage. Amazon S3 provides unlimited storage capacity at a low cost and with a pay-as-you-go model.

The following diagram shows a basic DR system. In this diagram, the data is in a traditional data center, with backups stored in AWS. AWS Import/Export or Snowball are used to get the data into AWS, and the information is later stored in Amazon S3:

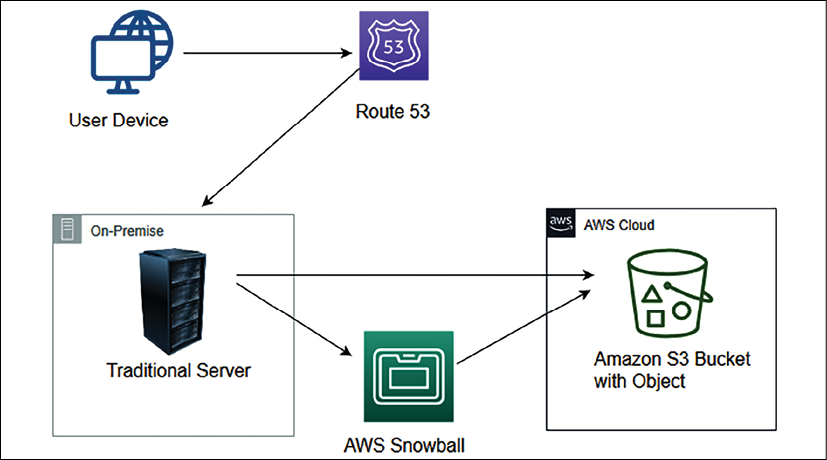


Figure 9.4: Data backup to Amazon S3 from on-premises infrastructure

You can use other third-party solutions available for backup and recovery. Some of the most popular choices are NetApp, VMware, Tivoli, Commvault, and CloudEndure. You need to take backups of the current system and store them in Amazon S3 using a backup software solution. Make sure to list the procedure to restore the system from a backup on the cloud, which includes the following:

1. Understand which **Amazon Machine Image** (**AMI**) to use or build your own machine image as required with pre-installed software and security patches.
2. Document the steps to restore your system from a backup.
3. Document the steps to route traffic from the primary site to the new site in the cloud.
4. Create a run book for deployment configuration and possible issues with their resolutions.

If the primary site located on-premises goes down, you will need to start the recovery process. As shown in the following diagram, in the preparation phase, create a custom **Amazon Machine Image** (**AMI**), which is pre-configured with the operating system and the required software, and store it as a backup in Amazon S3. Store any other data such as database snapshots, storage volume snapshots, and files in Amazon S3:

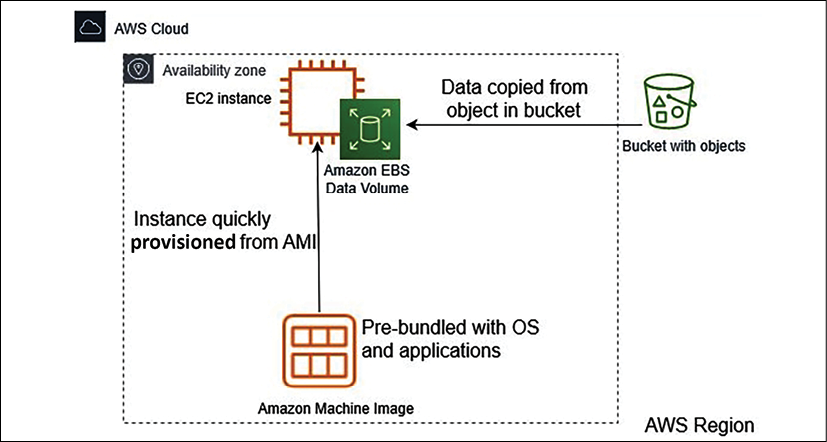


Figure 9.5: Restoring systems from Amazon S3 backups in the cloud

If the primary site goes down, you need to perform the following recovery steps:

1. Bring up the required infrastructure by spinning up Amazon EC2 server instances using machine images with all security patches and required software and put them behind a load balancer with an auto-scaling configuration as required.
2. Once your servers are up and running, you need to restore data from the backup stored in Amazon S3.
3. The last task is to switch over traffic to the new system by adjusting the DNS records to point to AWS.

It would be a better approach to automate elements of your infrastructure, such as networking, server, and database deployment, and bring it up by running the **AWS CloudFormation template**.

This DR pattern is easy to set up and relatively inexpensive. However, in this scenario, both the RPO and RTO will be high; the RTO will be the downtime until the system gets restored from the backup and starts functioning, while the RPO that is lost depends upon the system's backup frequency. Let's explore the next approach, pilot light, which provides improvements to your RTOs and RPOs.

**Pilot light**

The pilot light is the next lowest-cost DR method after backup and restore. As the name suggests, you need to keep the minimum number of core services up and running in different regions. You can spin up additional resources quickly in the event of a disaster.

You would probably actively replicate the database tier, then spin up instances from a VM image or build out infrastructure using infrastructure as code such as **CloudFormation**. Just like the pilot light in your gas heater, a tiny flame that is always on can quickly light the entire furnace to heat the house.

The following diagram shows a pilot light DR pattern. In this case, the database is replicated into AWS, with Amazon EC2 instances of the web servers and application servers ready to go, but not currently running:

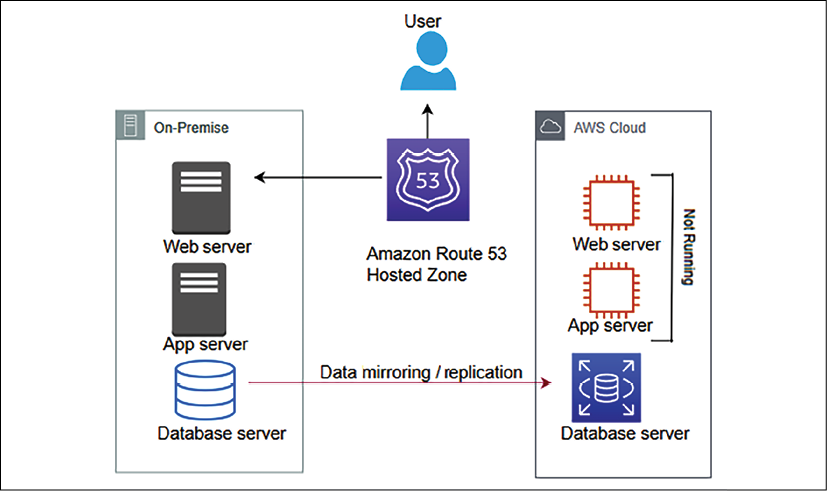


Figure 9.6: The pilot light data replication to DR site scenario

A pilot light scenario is pretty much similar to backup and restore, where you take a backup of most of the components and store them passively. However, you maintain active instances with a lower capacity for critical components such as a database or authentication server, which can take a significant time to come up. You need to automatically start all required resources, including network settings, load balancers, and virtual machine images as needed. As the core pieces are already running, recovery time is faster than the backup and restore method.

The pilot light is very cost effective as you are not running all of the resources at full capacity. You need to enable the replication of all critical data to the DR site—in this case, the AWS cloud. You can use the **AWS Data Migration Service** to replicate data between on-premises and cloud databases. For file-based data, you can use **Amazon File Gateway**. Many third-party-managed tools provide data replication solutions efficiently, such as Attunity, Quest, Syncsort, Alooma, and JumpMind.

If the primary system fails, as shown in the following diagram, you start up the Amazon EC2 instances with the latest copy of the data. Then, you redirect Amazon Route 53 to point to the new web server:

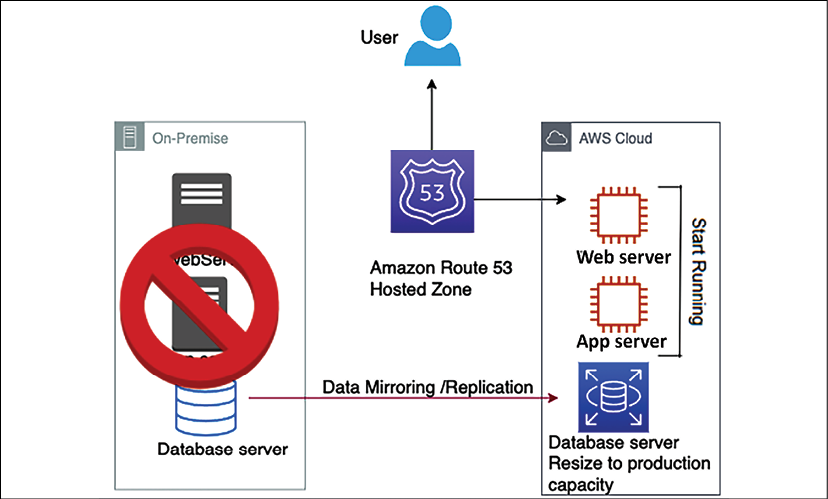


Figure 9.7: Recovery in the pilot light method

For the pilot light method, in the case of a disaster environment, you need to perform the following steps:

1. Start the application and web servers that were in standby mode. Further, scale-out application servers with horizontal scaling using a load balancer.
2. Vertically scale up the database instance that was running in low capacity.
3. Finally, update the DNS record in your router to point to the new site.

In pilot light, you bring up the resources around the replicated core dataset automatically and scale the system as required to handle the current traffic. A pilot light DR pattern is relatively easy to set up and inexpensive. However, in this scenario, the RTO takes longer to automatically bring up a replacement system, while the RPO largely depends on the replication type. Let's explore the next approach, warm standby, which further improves your RTOs and RPOs.

**Warm standby**

Warm standby, also known as **fully working low-capacity standby**, is like the enhanced version of the pilot light. It is the option where you use the agility of the cloud to provide low-cost DR. It saves the server cost and allows data to recover more quickly by having a small subset of services already running.

You can decide whether your DR environment should be enough to accommodate 30% or 50% of production traffic. Alternatively, you can also use this for non-production testing.

As shown in the following diagram, two systems are running in the warm standby method—the central system and a low-capacity system—on a cloud such as AWS.

You can use a router such as Amazon Route 53 to distribute requests between the central system and the cloud system:

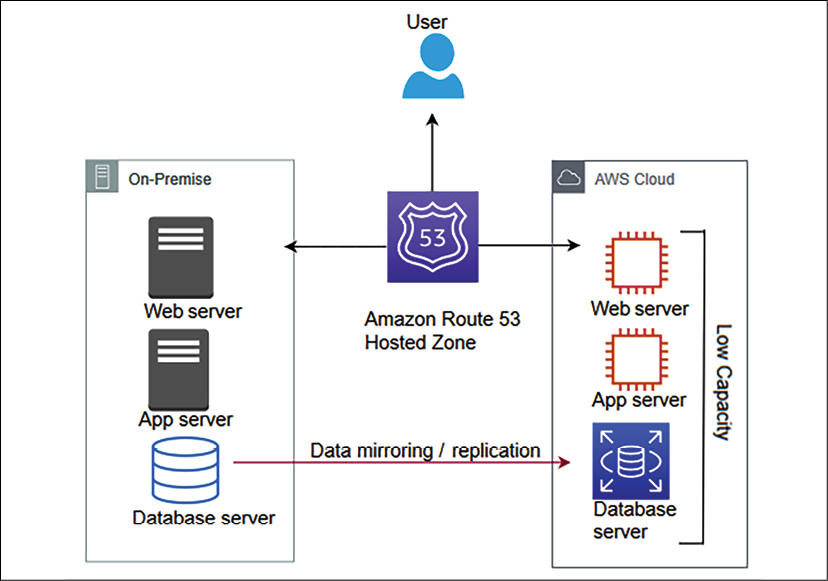


Figure 9.8: Warm standby scenario running an active-active workload with a low capacity

When it comes to databases, warm standby takes a similar approach to the pilot light, where data is continuously replicating from the main site to the DR site. However, in warm standby, you are running all necessary components 24/7; but they do not scale up for production traffic.

Often, the organization chooses a warm standby strategy for more critical workloads, so you need to make sure there are no issues in the DR site using continuous testing. The best approach to take is A/B testing, where the leading site will handle significant traffic. A small amount of traffic, approximately 1% to 5%, is routed to the DR site. This will make sure that the DR site is able to serve traffic when the primary site is down. Also, make sure to patch and update the software on the DR site regularly.

As shown in the following diagram, during the unavailability of the primary environment, your router switches over to the secondary system, which is designed to automatically scale its capacity up in the event of a failover from the primary system:

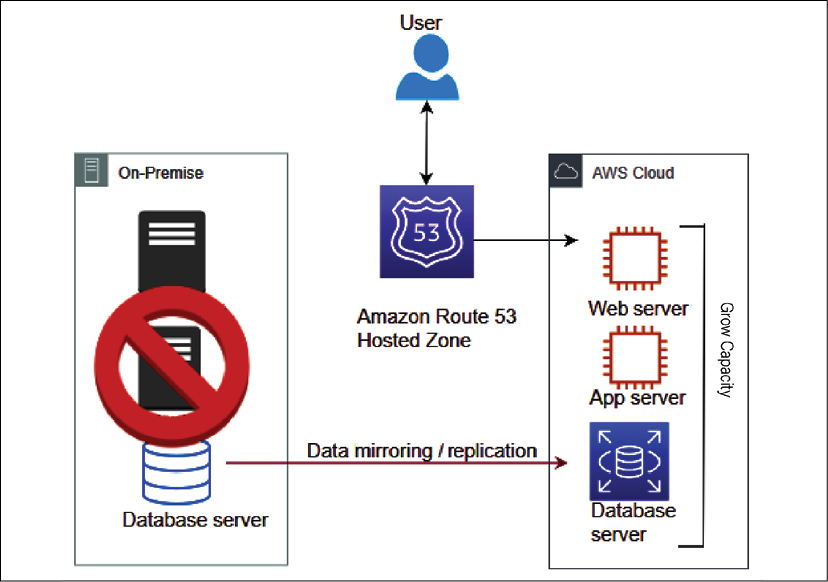


Figure 9.9: Recovery phase in the warm standby scenario

Suppose a failure occurs in the primary site. In that case, you can take the following approach:

1. Perform an immediate transfer of the critical production workload traffic to the DR site. Increase traffic routing from 5% to 100% in the secondary site. For example, in an e-commerce business, you first need to bring up your customer-facing website to keep it functioning.
2. Scale up the environment that was running on low capacity. You can apply vertical scaling for databases and horizontal scaling for servers.
3. As you scale up the environment, other non-critical workloads working in the background can now be transferred, such as warehouse management and shipping.

Your DR process becomes more efficient if your application is an all-in cloud, where entire infrastructures and applications are hosted in the public cloud, such as AWS.

The AWS cloud allows you to use cloud native tools efficiently; for example, you can enable a multi-AZ failover feature in the Amazon RDS database to create a standby instance in another Availability Zone with continuous replication.

In the case of the primary database, when an instance goes down, an in-built automatic failover takes care of switching the application to the standby database without any application configuration changes. Similarly, you can use automatic backup and replication options for all kinds of data protection.

A warm standby DR pattern is relatively complex to set up and expensive. The RTO is much quicker than the pilot light for the critical workload. However, for non-critical workloads, it depends upon how quickly you can scale up the system, while the RPO largely depends upon the replication type. Let's explore the next approach, multi-site, which provides near-zero RTOs and RPOs.

**Multi-site**

Lastly, the multi-site strategy, also known as a hot standby, helps you achieve near-zero RTO and RPO. Your DR site is a replica of the primary site with continuous data replication and traffic flow between sites. It is known as multi-site architecture due to the automated load balancing of traffic across regions or between on-premise and the cloud.

As shown in the following diagram, multi-site is the next level of DR, having a fully functional system running in the cloud at the same time as on-premises systems:

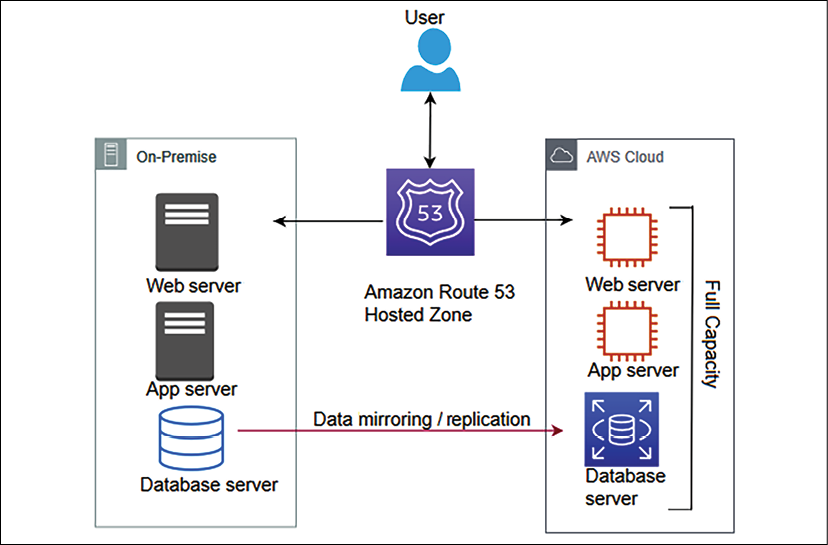


Figure 9.10: Multi-site scenario running an active-active workload with full capacity

The advantage of the multi-site approach is that it is ready to take a full production load at any moment. It's similar to warm standby but runs at full capacity on the DR site. If the primary site goes down, all traffic can be immediately failed over to the DR site, which is an improvement over the loss in performance and time when switching over and scaling up the DR site in the case of a warm standby.

A multi-site DR pattern is most expensive as it requires redundancy to be built for all components. However, the RTO is much quicker for all workloads in this scenario, while the RPO largely depends upon the replication type. Let's explore some best practices around DR to make sure your system is running reliably.

**Applying best practices for DR**

As you start thinking about DR, here are some important considerations:

* **Start small and build as needed**: Make sure you first bring up the critical workloads that have the most business impact and build upon this to bring up less critical loads. Streamline the first step of taking a backup, as often organizations lose data as they didn't have an efficient backup strategy. Take backups of everything, whether it is your file server, machine image, or databases.
* **Apply the data backup life cycle**: Keeping lots of active backups could increase costs, so make sure to apply a life cycle policy to archive and delete data as per your business needs. For example, you can choose to keep a 90-day active backup and, after that period, store it in low-cost archive storage such as a tape drive or Amazon Glacier. After 1 or 2 years, you may want to set a life cycle policy to delete the data. Compliance with standards such as PCI-DSS may require users to store data for 7 years, and in that case, you must opt for archival data storage to reduce costs.
* **Check your software licenses**: Managing software licenses can be a daunting task, especially in the current microservice architecture environment, where you have several services running independently on their virtual machines and databases. Software licenses could be associated with several installations, a number of CPUs, and several users. It becomes tricky when you go for scaling. You need to have enough licenses to support your scaling needs.
* **For horizontal scaling, you need to add more instances with software installed, and in vertical scaling, you need to add more CPU or memory**: You need to understand your software licensing agreement and ensure you have the appropriate license to fulfill system scaling. Also, make sure you don't buy excessive licenses, which you may not utilize and will cost you more money. Overall, make sure to manage your license inventory like your infrastructure or software.
* **Test your solutions often**: DR sites are created for rare DR events and are often overlooked. You need to make sure your DR solution is working as expected to achieve high reliability in case of an incident. Compromising a defined SLA can violate contractual obligations and result in the loss of money and customer trust.
* **Play gameday**: One way to test your solution often is by playing *gameday*. To play gameday, you choose a day when the production workload is small and gather all of the team responsible for maintaining the production environment. You can simulate a disaster event by bringing down a portion of the production environment and let the team handle the situation to keep the environment up and running. These events make sure you have working backups, snapshots, and machine images to handle disaster events.
* **Always monitor resources**: Put a monitoring system to make sure automated failover to the DR site occurs if an event occurs. Monitoring helps you to take a proactive approach and improves system reliability by applying automation. Monitoring capacity saves you from resource saturation issues, which can impact your application's reliability.

Creating a DR plan and performing regular recovery validation helps to achieve the desired application reliability. Let's learn more about improving reliability through the use of the public cloud.

**Improving reliability with the cloud**

In previous sections, you have seen examples of a cloud workload for the DR site. Many organizations have started to choose the cloud for DR sites to improve application reliability, as the cloud provides various building blocks. Also, cloud providers such as AWS have a marketplace where you can purchase multiple ready-to-use solutions from providers.

The cloud provides data centers that are available across geographic locations at your fingertips. You can choose to create a reliability site on another continent without any hassle. With the cloud, you can easily create and track the availability of your infrastructure, such as backups and machine images.

In the cloud, easy monitoring and tracking help make sure your application is highly available as per the business-defined SLA. The cloud enables you to have fine control over IT resources, cost, and handling trade-offs for RPO/RTO requirements. Data recovery is critical for application reliability. Data resources and locations must align with RTOs and RPOs.

The cloud provides easy and effective testing of your DR plan. You inherit features available in the cloud, such as the logs and metrics for various cloud services. Built-in metrics are a powerful tool for gaining insight into the health of your system.

With all available monitoring capabilities, you can notify the team of any threshold breach or trigger automation for system self-healing. For example, AWS provides CloudWatch, which collects logs and generates metrics while monitoring different applications and infrastructure components. It can trigger various automations to scale your application.

The cloud provides a built-in change management mechanism that helps to track provisioned resources. Cloud providers extend out-of-the-box capabilities to ensure applications and operating environments are running known software and can be patched or replaced in a controlled manner. For example, AWS provides AWS System Manager, which has the capability of patching and updating cloud servers in bulk. The cloud has tools to back up data, applications, and operating environments to meet requirements for RTOs and RPOs. Customers can leverage cloud support or a cloud partner for their workload handling needs.

With the cloud, you can design a scalable system, which can provide flexibility to add and remove resources automatically to match the current demand. Data is one of the essential aspects of any application's reliability. The cloud offers out-of-the-box data backup and replication tools, including machine images, databases, and files. In a disaster, all of your data is backed up and appropriately saved in the cloud, which helps the system recover quickly.

Regular interaction across the application development and operation team will help address and prevent known issues and design gaps, thereby reducing the risk of failures and outages. Continually architect your applications to achieve resiliency and distribute them to handle any outages. Distribution should span different physical locations to achieve high levels of availability.

**Summary**

In this chapter, you learned about various principles to make your system reliable. These principles include making your system self-healing by applying automation rules and reducing the impact in the event of failure by designing a distributed system where the workload spans multiple resources.

Overall system reliability heavily depends on your system's availability and its ability to recover from disaster events. You learned about synchronous and asynchronous data replication types and how they affect your system reliability. You learned about various data replication methods, including array-based, network-based, host-based, and hypervisor-based methods. Each replication method has its pros and cons. There are multiple vendors' products available to achieve the desired data replication.

You learned about various disaster planning methods depending on the organization's needs and the RTO and RPO. You learned about the backup and restore method, which has high RTO and RPO, and it is easy to implement. The pilot light method improves your RTO/RPO by keeping critical resources such as databases active in the DR site. The warm standby and multi-site methods maintain an active copy of a DR site's workload and help achieve a better RTO/RPO as you increase application reliability by lowering the system's RTO/RTO, the system's complexity, and costs. You learned about utilizing the cloud's built-in capability to ensure application reliability.

Solution design and launch may not happen too often, but operational maintenance is an everyday task. In the next chapter, you will learn about the alerting and monitoring aspects of solution architecture. You will learn about various design principles and technology choices to make your application operationally efficient and apply operational excellence.