**CHAPTER 9**

AWS Well-Architected Framework and Best Practices

In this chapter, you will

•   Learn how to secure your environment

•   Learn how to make a reliable architecture

•   Learn how to make sure the architecture is performing efficiently

•   Learn how to make sure the architecture is cost-effective

•   Learn how to make sure the architecture is excellent in terms of operation

When you use the AWS Well-Architected Framework for building any new architecture, you get these immediate benefits:

•   **Build and deploy faster** By reducing firefighting, implementing capacity management, and using automation, you can experiment and increase value of running into cloud more often.

•   **Lower or mitigate risks** Understand where you have risks in your architecture and address them before they impact your business and distract your team.

•   **Make informed decisions** Ensure you have made active architectural decisions that highlight how they might impact your business outcomes.

•   **Implement AWS best practices** Since you will be leveraging the AWS Well-Architected Framework, the architecture you will come up with will have all the best practices inherited in it.

In this chapter, you will learn about the AWS Well-Architected Framework (WAF). (Note that WAF also refers to the AWS product Web Application Firewall, but in this chapter it means the documentation and architecture called the AWS Well-Architected Framework.) You’ll also learn about AWS best practices and how to implement them. Whenever you create an architecture in AWS or deploy an application in AWS, it is important that your architecture follows all the AWS best practices. You want the architecture to be secure, efficient, scalable, reliable, and cost-effective. Designing an architecture using AWS best practices can help you achieve these business goals and make your organization successful.

When you’re constructing a building, if the foundation is not done properly, there a chance is that the building may collapse or end up damaged in some way over time. In the same way, wherever you are defining an architecture in AWS, you must construct the foundation carefully, which means embedding the principles of the AWS Well-Architected Framework into the design principles of the architecture. By using the AWS Well-Architected Framework, you can make sure that your architecture has all the best practices built in.

This chapter focuses on AWS WAF and the various principles behind the WAF. It also talks about the AWS best practices and what you should be doing when deploying an application to the cloud.

The AWS WAF has these five pillars:

•   Operational excellence

•   Security

•   Performance

•   Reliability

•   Cost optimization

All these pillars follow a design principle followed by best practices for each pillar.

OPERATIONAL EXCELLENCE

Operational excellence is measured in terms of how you are able to support the business. If you have aligned your operations team to support the business SLAs, you are in good shape. It is important that the operations team understands the business’s goals, priorities, and metrics so that it delivers according to the needs of the business.

Businesses may run several kinds of applications in the cloud. Some of those applications might be mission critical, and some of them won’t be. The operations team should be able to prioritize critical applications over noncritical applications and should be able to support them accordingly.

These are the design principles for achieving operational excellence in the cloud:

•   **Perform operations as code** In the cloud, it is possible to lay down the entire infrastructure as code and update it with code. You can script most of the tasks and try to automate as much as possible. For example, you should be able to automatically trigger operating procedures in response to events; if your CPU usage goes up, Auto Scaling can automatically start a new server.

•   **Document everything** Everything should be documented for all the operations in the cloud. It does not matter if you are making small changes or big changes in the system or in the application; you should annotate the documentation.

•   **Push small changes instead of big** Instead of pushing one big change in the system, it is recommended that you push small changes that can be reversible. The damage caused by a bigger change going wrong will be much bigger compared to the damage caused by small changes. Also, if the changes are reversible, you can roll back at any point of time if it does not go well.

•   **Refine operating procedures often** The architecture keeps on evolving, and therefore you need to keep updating your operating procedures. For example, say today you are using only one web server to host an application. Whenever there is a maintenance activity, you apply the operating system bug fixes on the one server. But tomorrow if you expand the web server footprint to four servers, you need to refine your operating procedures to make sure you will apply the operating system bug fixes on four different servers and not one. Set up regular days to review and validate your operating procedures.

•   **Anticipate failure** You should not wait for an actual failure to happen. You should assume failures can happen at any point in time and proactively simulate them. For example, in a multinode fleet of web servers, shut down one or two nodes randomly and see what the impact on the application is. Is the application able to resolve the failures automatically? You should be doing all kinds of destruction testing proactively so that when a real failure happens, your application is prepared to handle it.

•   **Learn from the operational failures** You should always learn from your operational failures and make sure that the same failure does not happen twice. You should share what you have learned with other teams, as well as learn from the failures of other teams.

Operational excellence in the cloud is composed of three areas: prepare, operate, and evolve. Each one is described in the following sections.

Prepare

Your operations team should be prepared to support the business. To do so, the operations team should understand the needs of the business. Since the operations team needs to support multiple business units, the team should know what each business unit needs. The priorities for every business unit might be different. Some business units may be running mission-critical applications, and other units might be running low-priority applications. The operations team must have a baseline of performance needed by business applications, and it should be able to support it. For example, say the business needs an order management system, and it is expecting an average of 100,000 orders per day from that system. The operations team should be prepared to provide the infrastructure that not only can host the order management system but also support 100,000 orders per day. Similarly, if the business is running a report and there is a performance degradation while running that report, the operations team should be able to handle it. In addition, the operations team should be prepared to handle planned and unplanned downtime. If you want your operations team to be successful, you should anticipate failures, as described earlier. This will make sure you are prepared to handle any kind of unplanned downtime.

Operate

When you are better prepared, you can handle the operations in a much more efficient way. Operational success is measured by the outcomes and metrics you define. These metrics can be based on the baseline performance for a certain application, or they can support your business in a certain way. To operate successfully, you must constantly meet the business goals and their SLAs, and you should be able to respond to events and take actions accordingly. One of the keys for the operations team’s success is to have proper communication with the business. The operations team should have a dashboard that provides a bird’s-eye view of the status of all the applications’ health checks. Consider the following four services when creating the dashboard:

•   **Amazon CloudWatch logs** Logs allow you to monitor and store logs from EC2 instances, AWS CloudTrail, and other sources.

•   **Amazon ES** Amazon ES makes it easy to deploy, secure, operate, and scale Elasticsearch for log analytics and application monitoring.

•   **Personal Health Dashboard** This dashboard provides alerts and remediation guidance when AWS is experiencing events that may impact you.

•   **Service Health Dashboard** This dashboard provides up-to-the-minute information on AWS service availability.

Automation can be your friend. To operate efficiently, you must automate as much as possible. If you are able to take care of automating the day-to-day operations and other pieces such as responding to certain events, you can focus on important and mission-critical activities.

Evolve

We all learn something new every day; similarly, you should always raise the operations team’s efficiency by taking it to the next level. You should learn from your own experience as well as from other people’s experience. You will often see that some people like to start with minimal viable products and then keep on adding more functionality on top of them. In the same way, regarding the infrastructure, they like to start small and keep evolving depending on how critical the infrastructure becomes. Thus evolve means start with small and continuously keep on adding new and new functionality or keep enhancing your architecture. An example of evolving architecture is given in the “AWS Best Practices” section of this chapter.

SECURITY

The next pillar of the WAF is security. Needless to say, security is the heart of everything; therefore, it must be your top priority. The security pillar contains design principles, which are discussed in the following sections.

Have a Strong Identity Foundation

Use IAM to manage the accounts in AWS. Use the principle of least privilege and don’t grant anyone access unless needed. There should be a central team of users responsible for granting access across the organization. This will make sure that access control is handled by only one set of people and others won’t be able to override each other. The principle of least privilege means that by default everyone should be denied access to the system. The access should be given only when someone explicitly requests it. This way, you will minimize unauthorized access to the system. In addition, you should be using either IAM users or federate users. You can use federation via SAML 2.0 or web identities. By using federation, you can leverage the existing identities, and you won’t have to re-create the users in IAM. It is important to define the roles and access for each user, and employee life cycle policies should be strictly enforced. For example, the day an employee is terminated, he should lose all access to the cloud. You should also enforce a strong password policy with a combination of uppercase, lowercase, and special characters, and users should change passwords after a specified time and not be allowed to repeat any of their last ten passwords. You can even enforce MFA when IAM users log in from the console. In many cases, IAM users may require access to AWS APIs via the Command Line Interface (CLI) or Software Development Kit (SDK). In that case, sometimes federation may not work properly. In those cases, you can use an access key and secret key in addition to or in place of a username and password. In some cases, you might notice that IAM roles may not be practical. For example, when you are switching from one service to another, you should leverage AWS Security Token Service to generate the temporary credentials.

Enable Traceability

You should be able to trace, audit, monitor, and log everything happening in your environment in real time and should have a mechanism to get an alert for any changes that are happening. You should also automate some of the actions by integrating the log and alert system with the system to automatically respond and take action in real time. It is important to enable auditing and traceability so that if anything happens, you will be able to quickly figure out who has logged in to the system and what action has been taken that has caused the issue. Make sure all the changes in the system are audited and you can account for who has made changes. You should have a strong system of control for making the changes. You can use AWS Config to track AWS resource inventory, configuration history, and configuration change notifications to enable security and governance. Without proper approvals, no change should be permissible in the system. You can also create rules that automatically check the configuration of AWS resources recorded by AWS Config using AWS Config rules. You can capture the key activities via AWS CloudTrail; it provides details about the API calls made in your account. You can also direct the CloudTrail logs to Amazon CloudWatch logs and can view whatever is happening across compute, storage, and applications under a single pane of glass.

Implement Security at All Layers

The security should be applied at all layers across the stack. Say you have EC2 servers running in both private and public subnets. In this case, you should have layers of security across the subnets by leveraging NACL, layers of security across EC2, and a load balancer by leveraging security groups; you also should secure the operating system, storage, and the applications running. In short you should be able to isolate every component of your infrastructure and secure each part. Let’s look at an example to understand this. Say you have a three-tier architecture with a web tier, app tier, and database tier. You should have separate security groups for each tier, and only authorized users can access the web tier or app tier or database tier. You should also put the Internet-facing web tier in the public subnet and put the internal-facing database and app tiers in the private subnet. Similarly, if you want to have a firewall, you should apply one to all the layers; in other words, use a separate firewall for the database tier and a separate firewall for the application tier. Also, you can use a separate set of ACLs for a different tier. Thus, you have a security control or firewall at every virtual server, every load balancer, and every network subnet. Focus on securing all your systems. Since AWS provides the shared responsibility model, as a result half of the burden of securing the data center, physical facilities, and networking is taken care by AWS. You just need to focus on securing your application, data, and operating systems. Whenever possible, leverage the managed services since they take the burden of managing the infrastructure from you. Similarly, whenever you are designing an architecture, you should make sure that you have leveraged all the avenues for securing the design. For example, within VPC, use the public and private subnets to segregate the workload depending on who can have Internet or external access. Use a bastion host to log in to instances running on the private subnet. Always use a NAT gateway when you want to update the servers running on a private subnet, use different security groups depending on the workload, and use NACL to filter the traffic at the subnet level. Use different VPCs for different workloads. For example, create a separate VPC for the production workload, a separate VPC for the development workload, and so on.

Secure the Data

Along with security at all layers, it is equally important to protect the data. You should secure the data both at rest and in transit. Use all the different technologies to encrypt the data depending on sensitivity. When the data moves from the web tier to the app tier or from the app tier to the database tier, make sure it is encrypted. You can use SSL or TLS to encrypt the data in transit. If you are using APIs, make sure they are SSL/TLS enabled. Similarly, for all communications, you should use SSL or TLS; you can also use a VPN-based solution or Direct Connect to make sure that the communication path is also secure. For data at rest, you can use technologies such as Transparent Data Encryption (TDE) to encrypt the data at rest. When you are using AWS services, you can use Amazon S3 server-side encryption, and you can encrypt the EBS volumes. If using client-side technologies, then you can use a supported SDK or OS to make sure it meets all the standards for security. Whenever you have been given the task of securing the data, you need to think about the path of data flow and secure all the points to make sure your data is secured everywhere. One of the most common examples of data flow is from ELB to EC2 to EBS to RDS to S3. When you know your data is going to touch these components, you can secure every component plus ensure that the data in transit is secure, and thus you will have an end-to-end secured solution. If you are using keys for encryption, then you should look at the AWS Key Management Service (KMS) for creating and controlling the keys. If you are using SSL, then your content is delivered via HTTPS for which you can leverage Amazon CloudFront. Using Amazon CloudFront provides lots of advantages. You can use your own domain name and SSL certificates, or you can use a Server Name Indication (SNI) custom SSL (older versions of browsers do not support SNI’s custom SSL), or you can use the dedicated IP custom SSL if your browser does not support SNI’s custom SSL. Amazon CloudFront supports all of them.

Automate for Security

Automation can be your best friend. You can have a software-based security mechanism to securely scale more rapidly and cost-effectively. You should set up alerts for all important actions so that if something goes wrong, you are immediately notified, and at the same time you should have automation so that the system can act upon it promptly. You can also set up some automated triggered responses to event-driven conditions. Also, it is important to monitor and go through the logs once in a while to make sure there are no anomalies. It is important to implement automation as a core tenet for security best practices. You can automate a lot of things to minimize risk and any errors. For example, you can install all the security patches and bug fixes into a virtual machine, save that as a gold image, and deploy this image to any server that you are going to launch. You can see by doing a small automation that you are able to implement the security fixes in all the VMs that you will be launching; it does not matter if it is hundreds of VMs or a few thousand.

Plan for Security Events

Always plan for security events well in advance. Run some simulations proactively to find gaps in your architecture and fix them before any incident can happen. Run the testing to simulate real-life attacks and learn from the outcome. Learn from other teams or different business units about how they are handling the security events. In a nutshell, your system should be ready against all kinds of attacks such as DDoS attacks and so on.

Best Practices

There are five best practices for security in the cloud: use identity and access management, use detective controls, use infrastructure protection, use data protection, and use incident response.

USE IDENTITY AND ACCESS MANAGEMENT

IAM makes sure that only those who are authorized can access the system. IAM can help in protecting the AWS account credentials as well as providing fine-grained authorization. You should use this service as a best practice. You already studied this in detail in [Chapter 5](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch5.xhtml).

USE DETECTIVE CONTROLS

You can use detective controls to identify a threat or incident. One type of detective control is to capture and analyze logs. If you want to do this in the on-premise world, you need to install some kind of agent on all the servers that will capture the logs and then analyze the agent. In the cloud, capturing logs is easy since assets and instances can be described without depending on the agent’s health. You can also use native API-driven services to collect the logs and then analyze them directly in the AWS cloud. In AWS, you can direct AWS CloudTrail logs to Amazon CloudWatch logs or other endpoints so you can get details of all the events. For EC2 instances, you will still use traditional methods involving agents to collect and route events.

Another way to use a detective control is to integrate auditing controls with notification and workflow. A search on the logs collected can be used to discover potential events of interest, including unauthorized access or a certain change or activity. A best practice for building a mature security operations team is to deeply integrate the flow of security events and findings into a notification and workflow system such as a ticketing system, thereby allowing you to route, escalate, and manage events or findings.

These are some of the services that help you when implementing detective controls:

•   **AWS Config** This is a fully managed service that provides you with an AWS resource inventory, configuration history, and configuration change notifications to enable security and governance. With AWS Config, you can discover existing AWS resources, export a complete inventory of your AWS resources with all the configuration details, and determine how a resource was configured at any point in time. These capabilities enable compliance auditing, security analysis, resource change tracking, and troubleshooting.

•   **AWS Config rule** An AWS Config rule represents the desired configurations for a resource and is evaluated against configuration changes on the relevant resources, as recoded by AWS Config. The results of evaluating a rule against the configuration of a resource are available on a dashboard. Using AWS Config rules, you can assess your overall compliance and risk status from a configuration perspective, view compliance trends over time, and pinpoint which configuration change caused a resource to drift out of compliance with a rule.

•   **AWS CloudTrail** This is a web service that records AWS API calls for your account and delivers logs. It can be useful in answering these questions: Who made the API call? When was the API call made? What was the API call? Which resources were acted upon in the API call? Where was the API call made from, and who was it made to?

•   **Amazon CloudWatch** You can use Amazon CloudWatch to gain systemwide visibility into resource utilization, application performance, and operational health. You can use these insights to keep your application running smoothly. The Amazon CloudWatch API and AWS SDKs can be used to create custom events in your own applications and inject them into CloudWatch events for rule-based processing and routing.

•   **VPC flow logs to help with network monitoring** Once enabled for a particular VPC, VPC subnet, or Elastic Network Interface (ENI), relevant network traffic will be logged to CloudWatch logs.

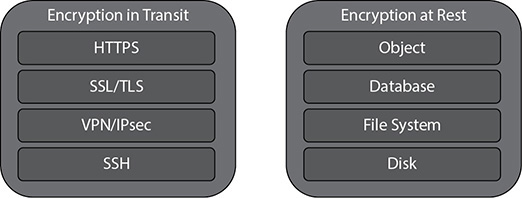
•   **Amazon Inspector** This tool offers a programmatic way to find security defects or misconfigurations in your operating systems and applications. It can be easily integrated with CI/CD tools and can be automated via APIs. It has the ability to generate findings.

USE INFRASTRUCTURE PROTECTION

Infrastructure protection consists of protecting your entire infrastructure. It ensures that systems and services within your solution are protected against unintended and unauthorized access and potential vulnerabilities. You can protect network and host-level boundaries by applying appropriate configurations to your virtual private cloud, subnets, routing tables, network access control lists (NACLs), gateways, and security groups to achieve the network routing as well as host-level protection. You can protect system security configuration and maintenance by using AWS Systems Manager. This gives you visibility and control of your infrastructure on AWS. With Systems Manager, you can view detailed system configurations, operating system patch levels, software installations, application configurations, and other details about your environment through the Systems Manager dashboard. The last thing in infrastructure protection is to enforce service-level protection. The security configurations of service endpoints form the foundation of how you will maintain secure and authorized access to these endpoints. You can protect AWS service endpoints by defining policies using IAM.

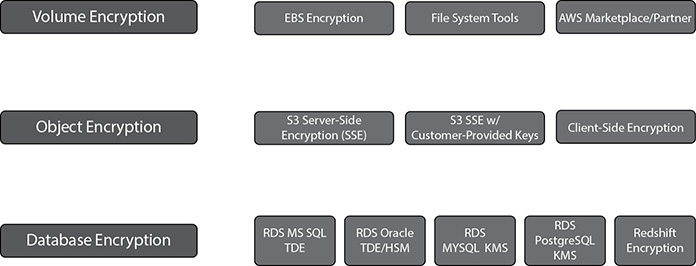
USE DATA PROTECTION

The data first needs to be classified according to the level of sensitivity. Depending on the type of data, you control the level of access/protection appropriate to the data classification. Once the data has been classified, you can either encrypt it or tokenize it. Encryption is a way of transforming content in a manner that makes it unreadable without a secret key necessary to decrypt the content back into plain text. Tokenization is a process that allows you to define a token to represent an otherwise sensitive piece of information. For example, you can have a token to represent an SSN. You can define your own tokenization scheme by creating a lookup table in an encrypted Amazon Relational Database Service (Amazon RDS) database instance. The next step would be to protect the data at rest as well as in transit. Data at rest represents any data that you persist for any duration. This includes block storage, object storage, databases, archives, and any other storage medium on which data is persisted. Data in transit is any data that gets transmitted from one system to another. The encryption of data in transit can be done by using SSL or TLS, HTTPS, VPN/IPsec, or SSH, as shown in [Figure 9-1](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#fig9_1).



**Figure 9-1** Encryption at rest and in transit

The encryption at rest can be done at the volume level, object level, and database level; the various methods are shown in [Figure 9-2](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#fig9_2). The last step in data protection is to have backups, a disaster strategy, and replication in place.



**Figure 9-2** Encryption at rest

USE INCIDENT RESPONSE

The last area of best practices for the security of the cloud is incident response. Whenever an incident happens, you should be able to respond to it quickly and act on it. Putting in place the tools and access ahead of a security incident and then routinely practicing incident response will help you ensure that your architecture can accommodate timely investigation and recovery. Products such as AWS Web Application Firewall and Shield can be leveraged to protect against SQL injection (SQLi) and cross-site scripting (XSS); prevent web site scraping, crawlers, and bots; and mitigate DDoS attacks (HTTP/HTTPS floods).

PERFORMANCE

This pillar is all about the need for speed. It focuses on performance efficiency and how your business can benefit from it. In this age, every business wants to run faster, so whatever speedup you can provide will always be helpful for running the business. In this section, you will look at the design principles of this pillar and the best practices associated with it.

It has the following design principles:

•   **Go global in a few clicks** AWS allows you to lay out a global infrastructure in minutes. Depending on the business needs, you should be able to design a highly scalable and available architecture, which can fail over to a different region in the case of disaster recovery. Leverage multiple AZs in your design pattern.

•   **Leverage new technologies such as serverless** The biggest advantage of the cloud is that for some use cases you have the opportunity of not managing any servers but still supporting the business. Try to leverage serverless architecture as much as possible so that you don’t have to manage the infrastructure.

•   **Consume advanced technology** If you are leveraging an advanced technology, you don’t have to learn it or become an expert in it. Leverage the managed services and just be a consumer. For example, say on your premise you have an Oracle database and you have migrated to Aurora in AWS; you don’t have to learn all the details of Aurora because it is a managed service and all the DBA activities are taken care of by Amazon.

•   **Leverage multiple technologies** As they say, one size does not fit all when you move to the cloud. Consider a technology approach that aligns best to what you are trying to achieve. Say you have a relational database running on-premise; when you move to the cloud, evaluate how much can be offloaded to a NoSQL system, how much can go in RDS, how much can go to Redshift, and so on. Thus, instead of moving the relational database as it is in the cloud, you can choose the right technology for the right data patterns and save a lot of money.

•   **Experiment more often** The cloud gives you the opportunity to experiment more often, which can help the business to innovate faster. You can quickly deploy resources in the cloud and start executing your idea. You don’t have to wait for months to procure new infrastructure. Even if your idea fails, it does not matter since you pay only for the resources used. You can quickly jump to the next idea.

Performance Efficiency

Performance efficiency in the cloud is composed of three steps: selection, review, and monitoring.

SELECTION

The cloud gives you a variety of choices. Choosing the right services for running your workload is the key. To select the right service, you need to have a good idea about your workload. If you know the nature of your workload, it becomes easier to choose the right solutions. Say your application needs a lot of memory; in that case, you should go with memory-intense EC2 instances. Similarly, if your application is I/O bound, you should provision PIOPS-based EBS volumes along with the EC2 server. When you are selecting a complete solution, you should focus on the following four areas:

•   **Compute** The ideal compute for an application depends on its design, usage patterns, and configuration settings. In AWS, compute is available in three forms: instances, containers, and functions. Instances are virtualized servers (EC2), and you can change their capabilities with just a few clicks or an API call. There are different types of EC2 instances available to choose from (general purpose, compute, memory, storage, GPU optimized). The key is choosing the right one for your workload. Containers are a method of operating system virtualization that allow you to run an application and its dependencies in resource-isolated processes. You can even choose containers to run your workload if you are running a microservice architecture. Functions abstract the execution environment from the code you want to execute. For example, AWS Lambda allows you to execute code without running an instance. When building new applications in the cloud, you should evaluate how you can leverage API Gateway and Lambda more instead of designing the application in the traditional way.

•   **Storage** AWS offers you various types of storage. You don’t have to stick with just one kind of storage; rather, you should leverage various types of storage. Even for the same application, you don’t have to use only one kind of storage. You can tier the storage as per the usage. The ideal storage solution will be based on the kind of access method (block, file, or object), patterns of access (random or sequential), frequency of access (online, offline, archival), frequency of update (WORM, dynamic), throughput required, and availability and durability constraints. This is discussed in more detail in the “Leverage Multiple Storage Options” section of this chapter.

•   **Network** The optimal network solution for a particular system will vary based on latency, throughput requirements, and so on. Physical constraints such as user or on-premises resources will drive location options, which can be offset using edge techniques or resource placement. In AWS, networking is virtualized and is available in a number of different types and configurations. This makes it easier to match your networking methods more closely with your needs. AWS provides lots of ways to optimize networks, and you should have a reworking solution that can support your business needs. For example, if you need faster networking between the EC2 instance and the EBS volume, you can use EBS-optimized instances, and Amazon EC2 provides placement groups for networking. A placement group is a logical grouping of instances within a single availability zone. Using placement groups with supported instance types enables applications to participate in a low-latency, 20Gbps network. Placement groups are recommended for applications that benefit from low network latency, high network throughput, or both. Similarly, S3 transfer acceleration and Amazon CloudFront speed things up, Direct Connect helps quickly to move data back and forth between your data center, and Amazon VPC endpoints provide connectivity to AWS services such as Amazon S3 without requiring an Internet gateway or a Network Address Translation (NAT) instance. Latency-based routing (LBR) for Route 53 helps you improve your application’s performance for a global audience.

•   **Database** Until a few years back, all data used to go to the relational database by default. Every organization used to have multiple big relational databases running with every kind of data in them. Now you don’t have to put everything in the relational database. Only part can remain in a relational database, and the rest can be moved to various other types of databases such as NoSQL, a data warehouse, and so on. Only the part of the data that needs relational access to data sets involving complex queries that will join and aggregate data from multiple entities can stay in a relational database, and the rest can be moved to a different type of database. If you are putting data in a relational database, you can either choose to host it in EC2 servers or use an RDS service to host it. Similarly, for NoSQL, you can use DynamoDB, and for data warehouses, you can use Redshift. In many cases, you may need to index, search, and consolidate information on a large scale. Technologies such as search engines are particularly efficient for these use cases. In these cases, you can use Elasticsearch in conjunction with Kibana and document or log aggregation with LogStash. Elasticsearch provides an easy-to-use platform that can automatically discover and index documents at a really large scale, and Kibana provides a simple solution for creating dashboards and analysis on indexed data. Elasticsearch can be deployed in EC2 instances, or you can use the Amazon Elasticsearch service, which is the managed service of AWS.

REVIEW

Once you make the selection, it is equally important to periodically review the architecture and make continuous changes to the architecture as per the business needs. The business needs can change at any point of time, and you may have to change the underlying infrastructure. For example, the business may want to open an internal ordering tool to all its suppliers. Now all the performance metrics are going to change, and to support the business, you have to make some architecture changes. Also, AWS keeps on innovating, releasing new services, dropping the price, and adding new geographies and edge locations. You should take advantage of them since it is going to improve the performance efficiency of your architecture. To reiterate, you should focus on the following from day 1:

•   Define your infrastructure as code using approaches such as AWS CloudFormation templates. This enables you to apply the same practices you use to develop software to your infrastructure so you can iterate rapidly.

•   Use a continuous integration/continuous deployment (CI/CD) pipeline to deploy your infrastructure. You should have well-defined metrics, both technical and business metrics, of monitoring to capture key performance indicators.

•   Run benchmarks and do performance tests regularly to make sure that the performance tests have passed successfully and meet the performance metrics. During the testing, you should create a series of test scripts that can replicate or simulate the life workload so that you can compare apples to apples (load testing).

MONITORING

Once you deploy the application/workload, you should constantly monitor it so that you can remediate any issues before your customers are aware of them. You should have a way of alerting whenever there is an anomaly. These alerts can be customized and should be able to automate action to work around any badly performing components. CloudWatch provides the ability to monitor and send notification alarms. You can then use these alarms to make automation work around performance issues by triggering actions through Amazon Kinesis, Amazon Simple Queue Service (Amazon SQS), and AWS Lambda.

RELIABILITY

Using this pillar you are going design the architecture as per the service level agreements. You need to consider how the solution can recover from various failures, how much outage a business can handle, how to work backward from the RTO and RPO levels, and so on. The goal is to keep the impact of any type of failure to a minimum. By preparing your system for the worst, you can implement a variety of mitigation strategies for the different components of your infrastructure and applications. When designing the system you can think about various failure scenarios and work backward. These requirements can cause long lead times because of dependencies and therefore must be incorporated during initial planning. With AWS, most of these foundational requirements are already incorporated or may be addressed as needed. At the same time, you need to make sure that your application or workload is leveraging all the aspects of reliability. For example AWS provides multiple AZs as a foundation for high availability. If you design your application to make use of multiple AZs, your application is going to leverage HA in the context of reliability automatically.

There are five design principles for reliability in the cloud:

•   **Test recovery procedures** Simulate all the failures that you might be encountering and try to see whether you are able to recover from those failures. Try to automate as much as possible to simulate the failure. You should be doing this activity often to make sure your system can handle failures and they are always up and running.

•   **Automate recovery from failure** You should automate the recovery from failures as much as possible. You should proactively monitor the system, as well as the response to an alert. By doing this, you should always be ready to handle a failure.

•   **Scale horizontally** Instead of scaling vertically, you should try to scale horizontally and leverage multiple servers to minimize the impact. For example, instead of running the workload on a 16-core server, you can run the same workload on four different servers of 4 cores each. You can even run these servers across different AZs within a region to provide HA within your application.

•   **Stop guessing capacity** In the cloud, the resources are almost infinite. In an on-premise environment, the resources are always provisioned for peak loads; despite that, if the resources saturate, the performance will degrade. In the cloud, there is no need to guess the right capacity. You just need to make sure you have proper automation in place that is going to spin off new resources for you as per the alerts.

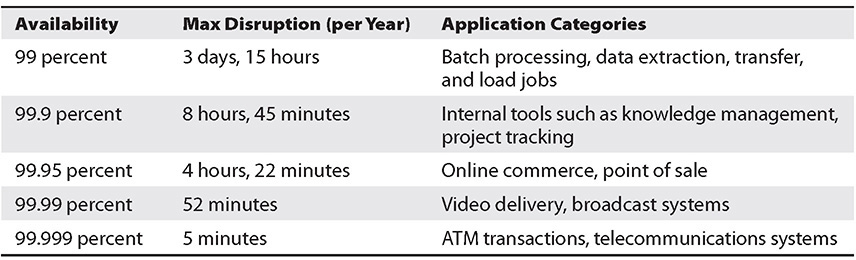
•   **Automate changes to the system** All the changes to the system should be done via automation. This minimizes the chances for error.

Best Practices

There are three best practice areas for reliability in the cloud: lay the foundations, implement change management, and implement failure management.

LAY THE FOUNDATION

Before architecting any system, you must lay down the foundation. The foundation should be laid out as per the reliability needs. Find out from the business what exactly it is looking for. For example, if the business says it needs 99.99 percent availability, find out what exactly it is trying to do and then work backward. Designing applications for higher levels of availability often comes with increased costs, so it makes sense to identify the true availability needs before starting the application design. [Table 9-1](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#tab9_1) shows availability and its corresponding downtime.



**Table 9-1** Availability vs. Outage

When you are building the foundation, you can start it right from your data center to the cloud. Do you need very large network bandwidth to your data center from the cloud? If yes, then you can start setting up Direct Connect instead of VPN. You can set up multiple Direct Connect options depending on your needs. The network topology needs to be planned well in advance, and you should also envision future growth and integration with other systems. You should keep in mind the following:

•   Allow IP address space for more than one VPC per region.

•   Within a VPC, keep space for multiple subnets that span multiple availability zones.

•   Leave some unused CIDR block space within a VPC, which will take care of any needs for the future.

•   Depending on business needs, you can have cross-account connections; in other words, each business can have their unique account and VPC that can connect back to the shared services.

•   Start thinking about all the failures and work backward. Say you have Direct Connect. If that fails, then what happens? You can have a second Direct Connect. If you don’t have the budget for a second Direct Connect, you can start with a VPN for failover.

There are a few areas you should focus on when designing an application for availability:

•   **Fault isolation zones** AWS has the fault isolation construct of availability zones; within a region there are two or more AZs. Whenever you are designing your workload/application, you should make sure that you are using at least two AZs. The more AZs you use in the architecture, the less chance there is of your application going down. Say you deploy your application across only one AZ. If that AZ goes down, there is 100 percent impact to your application. If you design the application using two AZs, it will have a 50 percent impact on your application if that goes down.

•   **Redundant components** One of the design principles of AWS is to avoid single points of failure in the underlying physical infrastructure. All the underlying physical infrastructure is built with redundant systems, so when you are designing the applications, you must make sure you plan for redundant components. For example, say you want to host your database on EC2 servers. Since data is the most critical part of the database, you can take multiple EBS volumes and use RAID to get HA in the storage layer.

•   **Leveraging managed services** Try to leverage managed services as much as possible because most of them come with built-in high availability. For example, RDS provides a multi-AZ architecture. If the AZ where you are hosting your database goes down, it automatically fails over to a different AZ. Similarly, if you choose to host your data in Amazon Aurora, six copies of the data are automatically written across three AZs.

IMPLEMENT CHANGE MANAGEMENT

Change management can make or break things. If you accidentally push the wrong code into the system, it can take down the entire system. In an on-premise environment, often the change control is a manual process and carefully coordinated with auditing to effectively control who makes changes and when they are made. In AWS, you can automate this whole part. You can also automate change management in response to key performance indicators, including fault management. When you are deploying changes in the system, you should consider these deployment patterns that minimize risk:

•   **Blue-green deployments** In this case, you have two stacks of deployment running in parallel: one stack running the old version and the other running the new version. You start with sending small traffic to the new deployment stack and watch out for failures, errors, and so on, and you send the rest of the traffic to the old stack (say 10 percent to the new stack and 90 percent to the old). If there are failures, you redirect the 10 percent traffic again to the old stack and work on fixing the issues. If things look good, you slowly keep on increasing the percentage of traffic to the new stack until you reach 100 percent.

•   **Canary deployment** This is the practice of directing a small number of your customers to the new version and scrutinizing deeply any behavior changes or errors that are generated. This is similar to blue-green deployment. If things look good, you can redirect more users to the new version until you are fully deployed, and if things keep on failing, you revert to the old version.

•   **Feature toggle** You can deploy the software with a feature turned off so that customers don’t see the feature. If the deployment goes fine, you can turn on the feature so that customers can start using it. If the deployment has problems, you keep the feature turned off without rolling back till the next deployment.

IMPLEMENT FAILURE MANAGEMENT

Your application should be able to handle the failure at every layer of the stack. You should evaluate every stack and think, if this stack fails or if this component goes down, how is my application going to handle it? You should ask questions like, what if the AZ fails, what if Direct Connect fails, what if the EC2 servers fail, or what if one of the hard drives goes down? Once you start thinking about all the possible scenarios, you should be able to architect your architecture in such a way that it can mitigate all types of failures. At the same time, it is important to make sure you have the proper backup and DR strategy in place. You should be regularly backing up your data and test your backup files to make sure you are able to recover from the backup. You should be running a DR simulation test to see how quickly you can spin up your infrastructure in a different region. You should be constantly shipping your data to a different region and automating the deployment of infrastructure in a different region using CloudFormation.

COST OPTIMIZATION PILLAR

This pillar helps you cut down on costs and provides you with the ability to avoid or eliminate unneeded cost or suboptimal resources. You might have noticed that AWS regularly decreases the prices of its products and services and encourages its customers to optimize their resources so that they have to pay less. AWS provides lots of ways to do this, which helps a business keep its budget in check. Who does not love lower costs?

In the cloud, you can follow several principles that help you save money. If you follow cost optimization best practices, you should have a good cost comparison with on-premises, but it’s always possible to reduce your costs in the cloud as your applications and environments are migrated and mature over time. Cost optimization should never end until the cost of identifying money-saving opportunities is more than the amount of money you are actually going to save.

The cost optimization pillar consists of the following design principles:

•   **Choose the best consumption model** Pay using the consumption model that makes the most sense for your business. If you are going with an on-demand or pay-as-you-go pricing model, you can shut down the environments when not in use to save costs. For example, if your developers leave for home after 5 p.m. and come to the office the next day at 9 a.m., you can shut down the development environment during that time. Similarly, for production workloads, if you know that your application will need a certain number of cores for the whole year, you can go with the reserved pricing model to save money.

•   **Use managed services** Leverage managed services and serverless services as much as you can so that you don’t have to deal with the infrastructure and can focus on running your business.

•   **Measure the overall efficiency** If you can measure the business output with the cost associated with it for delivery, you should be able to figure out in the long run if you are increasing costs or decreasing costs.

•   **Analyze the expenditure** The cloud provides you with all the tools to analyze the costs for running the business. You can find out which business unit is incurring which costs for running the system and can tie these back to the business owner. This way, you can find out whether you are getting enough return on investments.

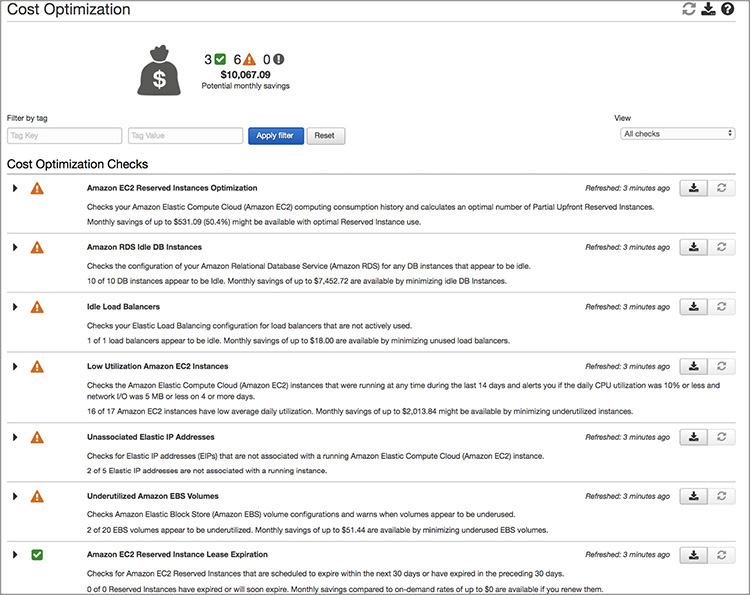
•   **Stop spending on a data center** Since AWS take cares of the heavy lifting, you don’t have to spend money on your data center. You can just focus on innovation and on running your business rather than on IT infrastructure.

Cost optimization in the cloud is composed of four areas: finding cost-effective resources, matching supply with demand, being aware of expenditures, and optimizing over time. The following sections describe each in detail.

Finding Cost-Effective Resources

You can do lots of things to make sure the resources you are utilizing are cost-effective. You can start with minimum resources, and after running your workload for some time, you can modify your resources to scale up or down to size them correctly. If you start with minimum resources, then sometimes you may not have to scale further down. Depending on your business needs, you can choose between various purchasing options such as on-demand resources, spot instances, and reserved instances. You should choose a region that is near to your end users and that meets the business needs. For example, reduced latency can be a key factor in improving the usage of your e-commerce or other web sites. When you architect your solutions, a best practice is to seek to place computing resources closer to users to provide lower latency and strong data sovereignty. As discussed earlier, managed services or serverless services can become your best friends to avoid unnecessary IT management costs.

AWS provides a tool called Trusted Advisor, shown in [Figure 9-3](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#fig9_3), that can help you control costs. Trusted Advisor looks at various AWS resources and provides guidance in terms of what you can do to get additional cost savings.



**Figure 9-3** Trusted Advisor

Matching Supply with Demand

In an on-premise environment, you always need to over-provision to meet the peak demand. Demand can be fixed or variable. In the cloud, you can automatically provision resources to match the demand. For example, by leveraging Auto Scaling, you can automatically scale up or down to meet the demand. Similarly, you can start with a small instance class of RDS and move to a higher instance class when the demand goes up. Your capacity needs to match your needs but not substantially exceed what you need.

Being Aware of Expenditures

It is important that business is aware of underlying costs. This will help the business to make better decisions. If the costs of operations are higher than the revenue from the business, then it does not make sense to invest more in infrastructure. On the other hand, if the costs of operations are much lower, it makes sense to invest more infrastructure for the growth of business. While determining the cost, make sure you capture all the small details such as data transfer charges or Direct Connect charges.

Optimizing Over Time

To optimize over time, you need to constantly measure, monitor, and improve. Measure and monitor your users and applications, and combine the data you collect with data from monitoring. You can perform a gap analysis that tells you how closely aligned your system utilization is to your requirements. Say you start with an on-demand instance model. After a few months, you realize that you are using this instance 24/7, and you are going to use this instance for another year. At that time, you can switch to a reserved instance plan instead of sticking with on-demand. If you are not sure whether you might need a bigger class of instance after six months, in that case you can choose a convertible reserved instance.

AWS BEST PRACTICES

If you want to deploy applications to the cloud, there are three different ways you can do so:

•   **Lift and shift** There are many applications that cannot be changed or modified. For instance, it is difficult to modify older applications. In that case, you have to “lift and shift” those applications to the cloud. Say you want to deploy Oracle ERP on the cloud; however, the application cannot be re-architected. In this case, it is important that you use the AWS core services. For example, you can use a VPC and subnets as a foundation and then use EC2 instances, EBS storage, and so on.

•   **Cloud optimized** Even if it is not possible to re-architect the application, you can still get some benefits from the cloud by optimizing the architecture for the cloud. For example, you can use native services such as RDS for provisioning the database and not EC2 servers, you can use SNS for sending the notifications, and so on.

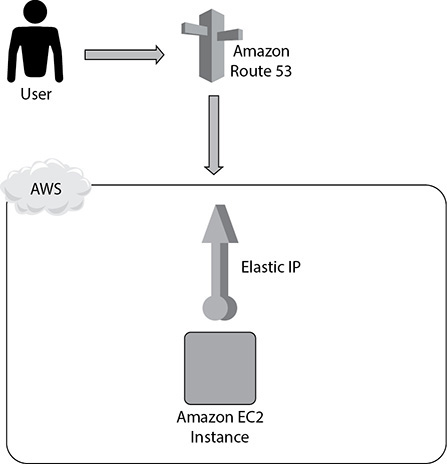
•   **Cloud-native architecture** In this case, you are starting everything from the cloud. You can leverage the full AWS portfolio and thus truly gain all the benefits of AWS in terms of security, scalability, cost, reliability, low operational cost, and so on.

Therefore, whenever you are architecting for the cloud, you should make sure that your architecture inherits all the best practices built in it. The following are the best practices you should follow for designing your architecture for the cloud.

Design for Failures

It is important that you design for failures at all layers of the stack. For the storage layer, you should mirror the data using RAID or some other technology. For the database tier, you should have high availability, and for the application tier, you should have multiple EC2 instances so that if one of them goes down, you can fail over to the other.

Let’s look at a simple example to understand this. Say you have an EC2 server that is hosting the database and the application, as shown in [Figure 9-4](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#fig9_4). You are also using an Elastic IP address for this server, and the end users are using Route 53 to reach the application.



**Figure 9-4** Hosting a database and application in an EC2 server

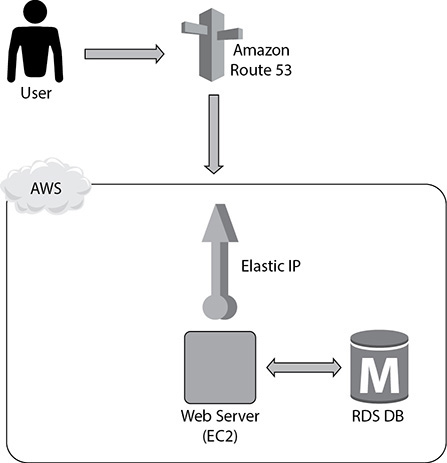
If you look at [Figure 9-4](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#fig9_4), do you see any problems? Yes, you have put all the components on a single EC2 server. There is no redundancy or failover. If this EC2 server goes down, what happens? Everything goes down, including your database, your applications, and everything else.

Amazon.com’s CTO, Werner Vogels, has said, “Everything fails, all the time.” If you design your architecture around that premise—specifically, assuming that any component will eventually fail—then your application won’t fail when an individual component does.

In this case, when designing for failure, your goal is to see your application survive when the underlying physical hardware fails on one of your servers.

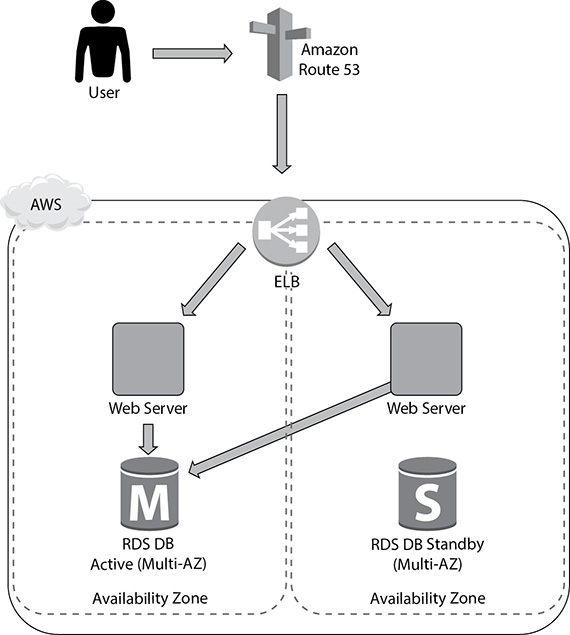
Let’s try to work backward from this goal. In real life, you will often see that you won’t be able to implement all the steps needed to make your architecture well architected in one day. Sometimes, as discussed previously, you have to evolve the design. Or you have to take small steps to reach a final architecture.

So, in this case, you can start by separating the database and the app tier. Thus, you are going to host the web tier in the EC2 server and put the database in RDS, as shown in [Figure 9-5](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#fig9_5). Still, the application is vulnerable because if the database goes down, the application goes down, and if the web server goes down, the application goes down. But at least now you have segregated the database and the web tier.



**Figure 9-5** Web server in EC2 and database in RDS

In the next phase, you can address the lack of failover and redundancy in the infrastructure by adding another web tier EC2 instance and enabling the multi-AZ feature of RDS ([Figure 9-6](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#fig9_6)), which will give you a standby instance in a different AZ from the primary. You’ll also replace your EIP with an elastic load balancer to share the load between your two web instances. Now the app has redundancy and high availability built in.



**Figure 9-6** High availability across web and database tiers

Therefore, the key takeaway from this is to avoid a single point of failure. You should always assume everything fails and design backward. Your goal should be that applications should continue to function even if the underlying physical hardware fails or is removed/replaced. If an individual component fails, the application should not have any impact. You can use the following AWS tools and technologies to do so:

•   Use multiple availability zones

•   Use elastic load balancing

•   Use elastic IP addresses

•   Do real-time monitoring with CloudWatch

•   Use Simple Notification Service (SNS) for real-time alarms based on CloudWatch metrics

•   Create database slaves across availability zones

Build Security in Every Layer

This was already discussed previously, but since this is such an important aspect, it is worth recapping. You should build security in every layer. For all important layers, you should encrypt data in transit and at rest. In addition, you should use the Key Management Service to create and control encryption keys used to encrypt your data. You should enforce the principle of least privilege with IAM users, groups, roles, and policies. You should use multifactor authentication to gain an additional layer of security. You should use both security groups to restrict the access to the EC2 servers and network ACLs to restrict traffic at the subnet level.

Leverage Multiple Storage Options

Since Amazon provides a wide range of storage options and one size does not fit all, you should leverage multiple storage options for your application to optimize. These are the storage services provided by Amazon:

•   **Amazon S3** Large objects

•   **Amazon Glacier** Archive data

•   **Amazon CloudFront** Content distribution

•   **Amazon DynamoDB** Simple nonrelational data

•   **Amazon EC2 Ephemeral Storage** Transient data

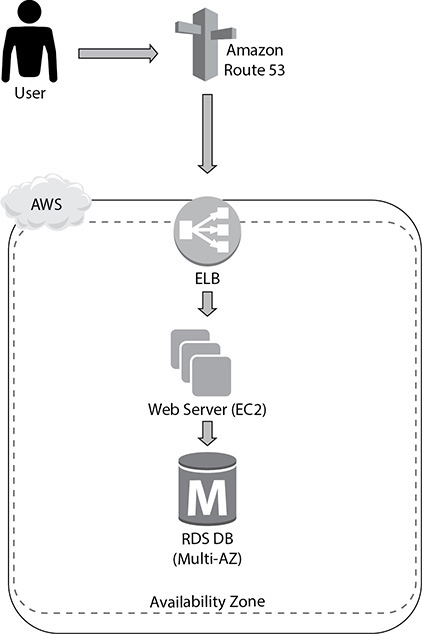
•   **Amazon EBS** Persistent block storage with snapshots

•   **Amazon RDS** Automated, managed MySQL, PostgreSQL, Oracle, Maria DB, SQL Server

•   **Amazon Aurora** Cloud-optimized flavors of MySQL and PostgreSQL

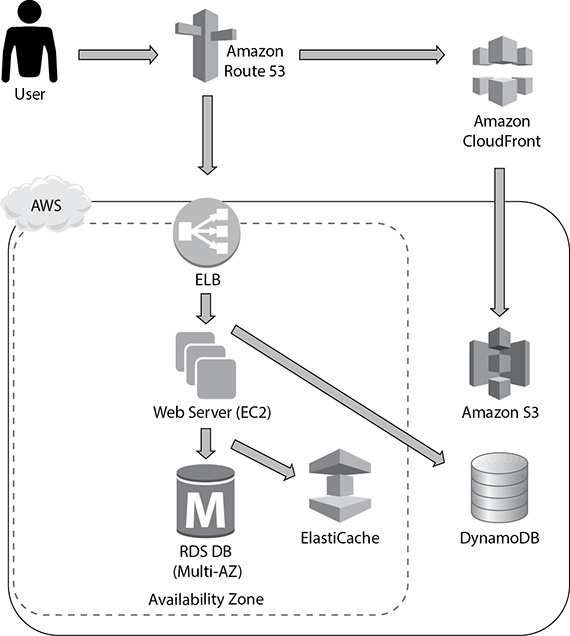
•   **Amazon Redshift** Data warehouse workloads

Let’s look at an example to understand this. Say you have built a web site on top of EC2 servers and are leveraging EBS volumes (let’s say multiple EC2 servers) to host your web site. The web server is running on EC2. You are also using an RDS database instance to host the database, as shown in [Figure 9-7](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#fig9_7). In this example, you are only using the EBS volume to store the data. To leverage multiple storage options and optimize the architecture, you can do lots of things.



**Figure 9-7** Web site running on top of EC2 and RDS

You can start by moving any static objects (images, videos, CSS, JavaScript) from the web server (EC2 servers) to S3 and then serving those objects via CloudFront. These files can be served via an S3 origin and then globally cached and distributed via CloudFront. This will take some load off your web servers and allow you to reduce your footprint in the web tier. The next thing you can do is move the session information to a NoSQL database like DynamoDB. If you do this, your session information won’t be stored in the EC2 servers. Therefore, when you scale up or down the EC2 servers, you won’t lose session information, and thus it won’t impact the users. This is called making the tier *stateless*. You can also use ElastiCache to store some of the common database query results, which will prevent hitting the database too much. [Figure 9-8](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#fig9_8) displays how you can leverage multiple storage options in this case.



**Figure 9-8** Leveraging multiple storage options

Implement Elasticity

Elasticity is one of the major benefits that you get by moving to the cloud. Gone are the days when you have to over-provision the hardware to meet the peak demand. With the AWS cloud, you just need to provision what is needed for now without worrying about the peak since you know that AWS provides elasticity, and you should be able to scale up and down any time to meet your demand. These are your best friends when implementing elasticity:

•   **Auto Scaling** Use Auto Scaling to automatically scale your infrastructure by adding or removing EC2 instances to your environment.

•   **Elastic load balancing** Use ELB to distribute the load across EC2 instances in multiple availability zones.

•   **DynamoDB** Use DynamoDB to maintain the user state information.

Think Parallel

AWS gives you the ability to experiment with different parallel architectures, and you should leverage all of them to make sure whatever you are doing will go quickly. Use SNS and SQS to build components that run in parallel and communicate (state, workflow, job status, and so on) by sending messages and notifications.

For example, you could publish a message to a topic every time a new image is uploaded. Independent processes, each reading from a separate SQS queue, could generate thumbnails, perform image recognition, and store metadata about the image.

•   **Use multithreading and concurrent requests to the cloud service** Submitting multiple requests to cloud services shouldn’t be done sequentially. Use threads to spawn simultaneous requests; a thread waiting for a response from a cloud service is not consuming CPU cycles, so you can spawn more threads than you have CPU cores to parallelize and speed up a job.

•   **Run parallel MapReduce jobs** Amazon Elastic MapReduce automatically spins up a Hadoop implementation of the MapReduce framework on Amazon EC2 instances, subdividing the data in a job flow into smaller chunks so that they can be processed (the “map” function) in parallel and eventually recombining the processed data into the final solution (the “reduce” function). Amazon S3 serves as the source for the data being analyzed and as the output destination for the end results. Say you run the EMR job using just one node and that takes ten hours. Let’s assume that the cost of running the EMR node is $1 per hour, so if your job runs for ten hours, you are paying $10. Now instead of running this job in one EMR node for ten hours, you can spin ten nodes in the EMR cluster. Since you have ten times more resources to process the job, your job will be finished in one hour instead of ten. You will pay $1 for each server for every hour, so that’s ten servers for one hour each, making a total cost of $10. If you compare the prices, you will notice that they both cost $10, but the job took ten hours previously; with MapReduce you are able to finish the same job in just one hour. The only difference this time is you have leveraged parallelism.

•   **Use elastic load balancing to distribute load** Use ELB to distribute incoming traffic across your Amazon EC2 instances in a single availability zone or multiple availability zones. Elastic load balancing automatically scales its request-handling capacity in response to incoming application traffic.

•   **Use Amazon Kinesis for concurrent processing of data** Using Kinesis, you can have multiple applications process a stream of data concurrently. Have one Amazon Kinesis application running real-time analytics and the other sending data to Amazon S3 from the same Amazon Kinesis stream.

Lambda lets you run thousands of functions in parallel, and performance remains consistently high regardless of the frequency of events. Use Lambda for back-end services that perform at scale.

When uploading files in S3, you can use multipart upload and ranged gets.

Loosely Couple Your Architecture

Always design architectures with independent components; the looser they’re coupled, the larger they scale. Design every component as a black box. Build separate services instead of something that is tightly interacting with something else. Use common interfaces or common APIs between the components. For example, if you are building an application for transcoding the video, you can do everything in one shot or can split the steps into multiple small processes. Don’t try to do all these tasks in one step: upload the file, transcode the video, and send the user a mail notification. If the transcode fails, you need to restart from the upload. Instead of that, if you split the process into three independent steps (upload video, transcode, and send mail notification), then even if the second step fails, you won’t have to start from scratch.

There Are No Constraints in the AWS Cloud

If you are coming from an on-premise ecosystem, often the resources are constrained. If your instance needs more CPU or more RAM or if your application needs a finely tuned I/O system, it takes a long time to get that. In AWS you don’t really have to worry about any type of constraint. Everything is available in just a few clicks. In the cloud, since there are new technologies available, you may try something out of the box to make the best use of it. For example, in an on-premises environment, you can host your web server in a bigger box, say, of 32 cores, in the cloud. Instead of hosting your web server on one server with 32 cores, you can use four servers of 8 cores each to host the web server. You can spread these web servers across different AZs to minimize the risk. You should try to use more AZs whenever possible. Here’s an example to explain this:

•   **First case using two AZs** Say you have four web servers, and you have deployed two in each AZ. The probability of one of the AZs going down is 50 percent, so if one of the AZs goes down, you will immediately lose 50 percent of your resource.

•   **Second case using three AZs** If you use three AZs, then the probability of one AZ going down is 33.3 percent. When this happens, you will lose 33.3 percent of your resource, which is always better than losing 50 percent of your resource. Similarly, if you use four AZs, you will lose 25 percent of your resource, and if you use five AZs, you will lose 20 percent of your resource.

In an on-premise world, whenever there is an issue with hardware, you try to troubleshoot and fix it. For example, the memory can go bad or there might be an issue with the disk or processor. Instead of wasting valuable time and resources diagnosing problems and replacing components, favor a “rip-and-replace” approach. Simply decommission the entire component and spin up a fully functional replacement. By doing this, you don’t have to spend any time troubleshooting the faulty part.

CHAPTER REVIEW

In this chapter, you learned about the five pillars of the AWS Well-Architected Framework.

Operations excellence is measured in terms of how you are able to support the business. If you have aligned your operations teams to support the business SLAs, you are in good shape. The design principles for operational excellence in the cloud are to perform operations as code, document everything, push small changes instead of big, refine operating procedures often, and anticipate failure.

The security pillar makes sure that your environment is secure from all aspects. These are the design principles for security: have a strong identity foundation, enable traceability, implement security at all the layers, secure the data, automate for security, and plan for security events.

The performance pillar is all about need for speed. This pillar focuses on performance efficiency and how your business can benefit from it. It has the following design principles: go global in a few clicks, leverage new technologies such as serverless, consume advanced technology, leverage multiple technologies, and experiment more often.

The reliability pillar makes sure that the solution can recover from various failures, determines how much outage a business can handle, determines how to work backward from the RTO and RPO levels, and so on. These are the design principles for reliability in the cloud: test recovery procedures, automate recovery from failure, scale horizontally, stop guessing capacity, and automate changes to the system.

The cost optimization pillar helps you cut down on costs and provides you with the ability to avoid or eliminate unneeded costs or suboptimal resources. The cost optimization pillar consists of the following design principles: choose the best consumption model, use managed services, measure the overall efficiency, analyze the expenditure, and stop spending on a data center.

These are the best practices in the cloud:

•   Design for failures.

•   Build security in every layer.

•   Leverage multiple storage options.

•   Implement elasticity.

•   Think parallel.

•   Loosely couple your architecture.

There are no constraints in the AWS cloud.

Questions

[**1**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#ch9_1)**.**   How do you protect access to and the use of the AWS account’s root user credentials? (Choose two.)

**A.**   Never use the root user

**B.**   Use multifactor authentication (MFA) along with the root user

**C.**   Use the root user only for important operations

**D.**   Lock the root user

[**2**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#ch9_2)**.**   What AWS service can you use to manage multiple accounts?

**A.**   Use QuickSight

**B.**   Use Organization

**C.**   Use IAM

**D.**   Use roles

[**3**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#ch9_3)**.**   What is an important criterion when planning your network topology in AWS?

**A.**   Use both IPv4 and IPv6 IP addresses.

**B.**   Use nonoverlapping IP addresses.

**C.**   You should have the same IP address that you have on-premise.

**D.**   Reserve as many EIP addresses as you can since IPv4 IP addresses are limited.

[**4**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#ch9_4)**.**   If you want to provision your infrastructure in a different region, what is the quickest way to mimic your current infrastructure in a different region?

**A.**   Use a CloudFormation template

**B.**   Make a blueprint of the current infrastructure and provision the same manually in the other region

**C.**   Use CodeDeploy to deploy the code to the new region

**D.**   Use the VPC Wizard to lay down your infrastructure in a different region

[**5**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#ch9_5)**.**   Amazon Glacier is designed for which of the following? (Choose two.)

**A.**   Active database storage

**B.**   Infrequently accessed data

**C.**   Data archives

**D.**   Frequently accessed data

**E.**   Cached session data

[**6**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#ch9_6)**.**   Which of the following will occur when an EC2 instance in a VPC with an associated elastic IP is stopped and started? (Choose two.)

**A.**   The elastic IP will be dissociated from the instance.

**B.**   All data on instance-store devices will be lost.

**C.**   All data on Elastic Block Store (EBS) devices will be lost.

**D.**   The Elastic Network Interface (ENI) is detached.

**E.**   The underlying host for the instance is changed.

[**7**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#ch9_7)**.**   An instance is launched into the public subnet of a VPC. Which of the following must be done for it to be accessible from the Internet?

**A.**   Attach an elastic IP to the instance.

**B.**   Nothing. The instance is accessible from the Internet.

**C.**   Launch a NAT gateway and route all traffic to it.

**D.**   Make an entry in the route table, passing all traffic going outside the VPC to the NAT instance.

[**8**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#ch9_8)**.**   To protect S3 data from both accidental deletion and accidental overwriting, you should:

**A.**   Enable S3 versioning on the bucket

**B.**   Access S3 data using only signed URLs

**C.**   Disable S3 delete using an IAM bucket policy

**D.**   Enable S3 reduced redundancy storage

**E.**   Enable multifactor authentication (MFA) protected access

[**9**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#ch9_9)**.**   Your web application front end consists of multiple EC2 instances behind an elastic load balancer. You configured an elastic load balancer to perform health checks on these EC2 instances. If an instance fails to pass health checks, which statement will be true?

**A.**   The instance is replaced automatically by the elastic load balancer.

**B.**   The instance gets terminated automatically by the elastic load balancer.

**C.**   The ELB stops sending traffic to the instance that failed its health check.

**D.**   The instance gets quarantined by the elastic load balancer for root-cause analysis.

[**10**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#ch9_10)**.**   You are building a system to distribute confidential training videos to employees. Using CloudFront, what method could be used to serve content that is stored in S3 but not publicly accessible from S3 directly?

**A.**   Create an origin access identity (OAI) for CloudFront and grant access to the objects in your S3 bucket to that OAI

**B.**   Add the CloudFront account security group called “amazon-cf/amazon-cf-sg” to the appropriate S3 bucket policy

**C.**   Create an Identity and Access Management (IAM) user for CloudFront and grant access to the objects in your S3 bucket to that IAM user

**D.**   Create an S3 bucket policy that lists the CloudFront distribution ID as the principal and the target bucket as the Amazon resource name (ARN)

Answers

[**1**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#rch9_1)**.**   **A, B.** It is critical to keep the root user’s credentials protected, and to this end, AWS recommends attaching MFA to the root user and locking the credentials with the MFA in a physically secured location. IAM allows you to create and manage other nonroot user permissions, as well as establish access levels to resources.

[**2**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#rch9_2)**.**   **B.** QuickSight is used for visualization. IAM can be leveraged within accounts, and roles are also within accounts.

[**3**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#rch9_3)**.**   **B.** Using IPv4 or IPv6 depends on what you are trying to do. You can’t have the same IP address, or when you integrate the application on-premise with the cloud, you will end up with overlapping IP addresses, and hence your application in the cloud won’t be able to talk with the on-premise application. You should allocate only the number of EIPs you need. If you don’t use an EIP and allocate it, you are going to incur a charge on it.

[**4**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#rch9_4)**.**   **A.** Creating a blueprint and working backward from there is going to be too much effort. Why you would do that when CloudFormation can do it for you? CodeDeploy is used for deploying code, and the VPC Wizard is used to create VPCs.

[**5**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#rch9_5)**.**   **B, C.** Amazon Glacier is used for archival storage and for archival purposes.

[**6**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#rch9_6)**.**   **A, D.** If you have any data in the instance store, that will also be lost, but you should not choose this option since the question is regarding elastic IP.

[**7**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#rch9_7)**.**   **B.** Since the instance is created in the public subnet and an Internet gateway is already attached with a public subnet, you don’t have to do anything explicitly.

[**8**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#rch9_8)**.**   **A.** Signed URLs won’t help, even if you disable the ability to delete.

[**9**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#rch9_9)**.**   **C.** The ELB stops sending traffic to the instance that failed its health check.

[**10**](https://learning.oreilly.com/library/view/aws-certified-solutions/9781260470192/ch9.xhtml#rch9_10)**.**   **A.** Create an OAI for CloudFront and grant access to the objects in your S3 bucket to that OAI.