Chapter 2

The AWS Well-Architected Framework

This chapter covers the following topics:

- **The Well-Architected Framework**
- **Designing a Workload SLA**
- **Deployment Methodologies**

This chapter covers content that's important to the following exam domain and task statements:

Domain 2: Design Resilient Architectures

Task Statement 1: Design scalable and loosely coupled architectures

Task Statement 2: Design highly available and/or fault-tolerant architectures

Your organization may be developing applications to be hosted in the cloud, or they may want to move some or all of current IT operations to the cloud. Regardless of the reason or scenario, your organization's applications/workloads that are moved to the cloud will be hosted on a variety of cloud services maintained and provided by the cloud provider. The most popular public cloud provider competitors to AWS, Microsoft Azure and Google Cloud, have been in operation for well over a decade. During this time, there have been many lessons learned by all cloud providers and customers; what works and what needs to be refined. This learned experience has resulted in many best practices that have been tested and refined for a variety of development and deployment scenarios. Each customer, before moving to the public cloud should take advantage of this documented experience, called the Well-Architected Framework. Microsoft Azure has released the Microsoft Azure Well-Architected Framework (**https://learn.microsoft.com/en-us/azure/architecture/framework/**), and Google has a Google Cloud Architecture Framework (**https://cloud.google.com/architecture/framework**).

The focus of this chapter, and indeed the book, is the pillars of the AWS Well-Architected Framework. The AWS Certified Solutions Architect – Associate website states that “The focus of this certification is on the design of cost and performance optimized solutions, demonstrating a strong understanding of the AWS Well-Architected Framework.”

Selecting the best architectural design for a workload by considering the relevant best practices that have been tested in production by thousands of customers allows organizations to successfully plan for success with a very high degree of confidence. Instead of reacting to a never-ending series of short-term issues and fixes, customers can plan for engineering and operating workloads in the AWS cloud with a proven long-term approach. Whether you have existing systems that you are trying to migrate to the cloud or are building a new project from the ground up, using the AWS Well-Architected Framework will ultimately save your organization a great deal of time by considering many scenarios and ideas for design and deployment that you might not have fully considered.

Keep in mind that you are engineering your workloads for people—your customers. Employees and contractors are building and operating your cloud-based or hybrid deployments, choosing when, where, and how to use the technologies that make sense for each workload deployment. The goal is architecting your operations and workloads to operate successfully in the cloud, meeting and exceeding your business needs and requirements.

“Do I Know This Already?”

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. **Table 2-1** lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in **Appendix A**, “**Answers to the ‘Do I Know This Already?’ Quizzes and Q&A Sections.**”

Table 2-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section	Questions
The Well-Architected Framework	1, 2
Designing a Workload SLA	3, 4
Deployment Methodologies	5, 6

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of the self-assessment. Giving yourself credit for an answer you correctly guess skews your self-assessment results and might provide you with a false sense of security.

---

1. The AWS Well-Architected Framework Sustainability pillar provides guidance on what types of impacts?

1. Reliability
2. Sizing compute and storage to avoid waste
3. Compute performance
4. Storage sizing

2. Which other AWS Well-Architected Framework pillars does workload reliability also affect?

1. Cost Optimization and Sustainability
2. Operation Excellence and Sustainability
3. Scale and Performance
4. Security and Cost Optimization

3. A workload SLA is designed to meet what criteria?

1. Cloud service SLA
2. Service-level objective
3. Mean time between failures
4. Restore point objective

4. What are the metrics used in determining a workload SLA called?

1. Service-level indicators
2. CloudWatch metrics
3. Rules and alerts
4. AWS defined SLAs

5. Agile development means focusing on what processes at the same time?

1. Design, coding, and testing
2. Planning and testing
3. Planning and coding
4. Planning, design, coding, and testing

6. What companion process can also be used with the AWS Well-Architected Framework?

1. Big Bang development
2. Waterfall development
3. Agile development
4. Twelve-Factor App Methodology

Foundation Topics

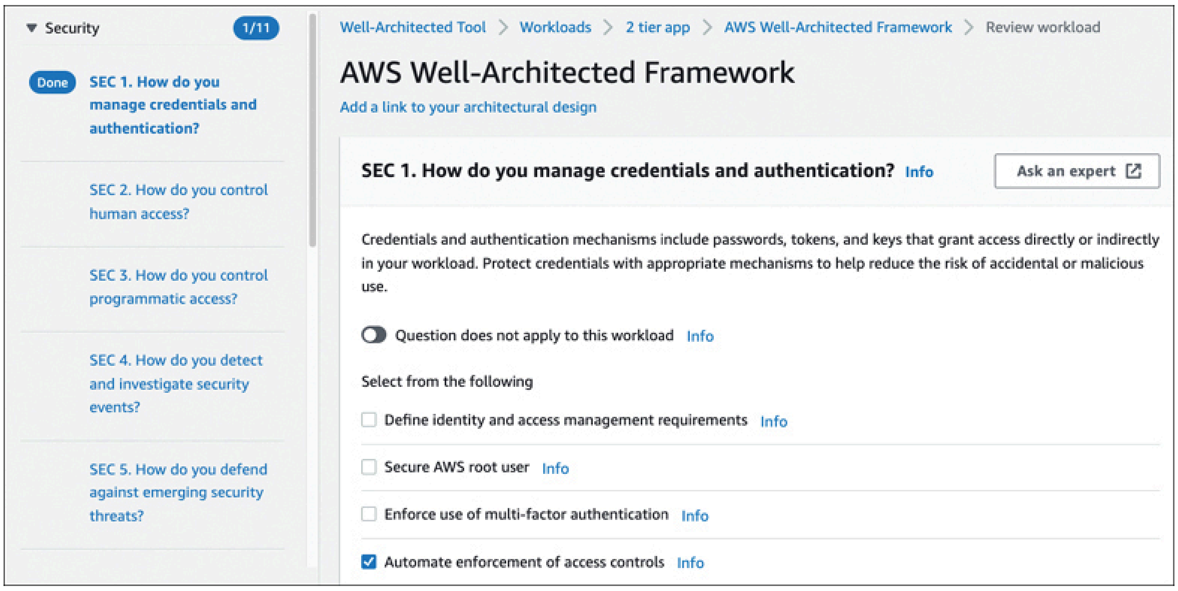
The Well-Architected Framework

As previously mentioned, AWS, Microsoft Azure, and Google Cloud all have their own well-architected frameworks (WAFs) as guidance, broken up into essential categories. I strongly recommend AWS's guidance for workload deployment (and for preparing for the AWS Certified Solutions Architect – Associate exam; exam questions are based on the Reliability, Security, Performance Efficiency, and Cost Optimization pillars). AWS is in constant contact with its customers to evaluate what they are currently doing in the cloud, and how they are doing it. Customers are always asking for features and changes to be added, and AWS and other public cloud providers are happy to oblige; after all, they want to retain their customers and keep them happy.

New products and services may offer improvements, but only if they are the right fit for your workload and your business needs and requirements. Decisions should be based on the six pillars of the AWS Well-Architected

Framework: Security, Reliability, Performance Efficiency, Cost Optimization, Operational Excellence, and Sustainability. Evaluating these pillars is necessary for both pre-deployment architecture design *and* during your workload solution's lifecycle. Each of the WAF pillars is a subdiscipline of systems engineering in itself. Security engineering, reliability engineering, operational engineering, performance engineering, and cost and sustainability engineering are all areas of concern that customers need to keep on top of.

The AWS Well-Architected Framework has a number of relevant questions for each pillar that each customer should consider (see **Figure 2-1**). The end goal is to help you understand both the pros and cons of any decisions that you make when architecting workload successfully in the AWS cloud. The Well-Architected Framework contains best practices for designing reliable, secure, efficient, and cost-effective systems; however, you must carefully consider each best practice offered to see whether it applies. Listed best practices are suggestions, not decrees; final decisions are always left to each organization.


▼ Security 1/11

Done SEC 1. How do you manage credentials and authentication?

SEC 2. How do you control human access?

SEC 3. How do you control programmatic access?

SEC 4. How do you detect and investigate security events?

SEC 5. How do you defend against emerging security threats?

Well-Architected Tool > Workloads > 2 tier app > AWS Well-Architected Framework > Review workload

AWS Well-Architected Framework

Add a link to your architectural design

SEC 1. How do you manage credentials and authentication? Info

Ask an expert

Credentials and authentication mechanisms include passwords, tokens, and keys that grant access directly or indirectly in your workload. Protect credentials with appropriate mechanisms to help reduce the risk of accidental or malicious use.

Question does not apply to this workload Info

Select from the following

- [ ] Define identity and access management requirements Info
- [ ] Secure AWS root user Info
- [ ] Enforce use of multi-factor authentication Info
- [x] Automate enforcement of access controls Info

Figure 2-1 AWS Well-Architected Framework Security Pillar Questions

Before AWS launches a new cloud service, a focus on the desired operational excellence of the proposed service is discussed based on the overall requirements and priorities for the new service and the required business outcomes. There are certain aspects of operational excellence that are considered at the very start of the prepare phase, and there are continual tasks performed during the operation and management of the workload during its lifetime. However, one common thread is woven throughout the operational excellence pillar: performing detailed monitoring of all aspects of each workload during the prepare, operate, and evolve phases. There are four best practice areas for achieving and maintaining operational excellence:

- **Organization:** Completely understand the entire workload, team responsibilities, and defined business goals for business success for the new workload.
- **Prepare:** Understand the workload to be built, and closely monitor expected behaviors of the workload's in-

ternal state and external components.

- **Operate:** Each workload's success is measured by achieving both business and customer outcomes.
- **Evolve:** Continuously improve operations over time through learning and sharing the knowledge learned across all technical teams.

After operation excellence has been addressed, the next goal is to make each workload as secure as possible. Once security at all layers has been considered and planned for, workload reliability is next addressed. Next up is the performance efficiency of the workload; performance should be as fast as required, based on the business requirements. How do you know when there is a security, reliability, performance, or operational issue? To paraphrase: "There are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns—the ones we don't know we don't know." Ultimately, the answer to what we don't know always comes back to proactive monitoring. Without monitoring at every level of the workload, technical teams will not be able to monitor the operational health of each workload and won't have enough information to be able to solve existing and unexpected problems.

Let's look at the goals of each pillar of the AWS Well-Architected Framework in brief.

Key Topic

Operational Excellence Pillar

Operational excellence isn’t a one-time effort that you achieve and are done with. Operational excellence is the repeated attempts to achieve greater outcomes from the technologies you choose and to improve your workload’s operational models, procedures, principles, patterns, and practices.

Once security, reliability, performance, and cost have been addressed, you will have hopefully achieved a measure of operational excellence for a period of time, perhaps six months, perhaps longer. Then AWS will introduce a new feature, or a new service that looks interesting; this will send you back into the testing and development phase once again. Perhaps you will find that one of the recently introduced new features will vastly improve how a current workload could be improved. This cycle of change and improvement will continue, forever. Take any new or improved features into account. Operational excellence guidance helps workloads successfully operate in the AWS cloud for the long term.

Operational excellence has been achieved when your workload is operating with just the right amount of security and reliability; the required performance is perfect for your current needs; there is no waste or underutilized components in your application stack; and the cost of running your application is exactly right. Achieving this rarefied level of operation might seem like a fantasy, but in fact it’s the ultimate goal of governance processes that have been designed by a Cloud Center of Excellence (Cloud CoE) team within an organization that is tasked with overseeing cloud operations for the organization. The Cloud CoE, and the culture of operational excellence it implements, is the driver that propals improvements and value throughout your organization. But getting there takes work, planning, analysis, and a willingness to make changes to continuously improve and refine operations as a whole.

Changes and improvements to security, reliability, performance efficiency, and cost optimization within your cloud architectures are best communicated through a Cloud CoE. It's also important to realize that changes that affect one pillar might have side effects in other pillars. Changes in security will affect reliability. Changes in improving reliability will affect cost. Operational excellence is where you can review the entire workload and achieve the desired balance.

Note

Operational Excellence design principles include organize, prepare, operate, and evolve. These best practices are achieved through automated processes for operations and deployments, daily and weekly maintenance, and mitigating security incidents when they occur.

Security Pillar

After the initial operational excellence design discussions, implementing a strong security foundation is next. After a workload has been deployed, the management of workload security is paramount. Organizations must design security into their cloud solution architectures to protect the workload and ensure its survival from attacks and catastrophic events.

If your online store or customer portal is knocked offline by hackers or corrupted with false or damaging information, your organization's reputation and customers could be maligned. What if customers' financial, medical, or other personal information is leaked from your systems? Organizations must design, deploy, test, monitor, and continuously improve security controls from the beginning. Security requires planning, effort, and expense, as nothing is more valuable than your customers and your ability to securely deliver applications and services meeting their needs. Strategies need to be employed to help achieve the security, privacy, and compliance required by each application and its associated components. These strategies are discussed next.

Defense in Depth

Defense in depth can be divided into three areas: physical controls, technical controls, and administrative controls. AWS as the cloud provider is responsible for security of the cloud; therefore, AWS is responsible for securing the physical resources using a variety of methods and controls. AWS also is responsible for providing technical and administrative controls for the cloud services its customers are using, so that they may secure and protect their application stacks and resources that are hosted in the cloud.

Each component of your application stack should have relevant security controls enabled to limit access. For example, consider a two-tier application consisting of web servers and an associated relational database. Both the web and database servers should be hosted on private subnets with no direct access to the Internet. Access to the Internet for updates and licensing should be controlled by using network address translation (NAT) services that allow indirect access from private subnets to the Internet for server updates. For public-facing applications, the load balancer should be placed on a public subnet accepting and directing requests to the web servers hosted on private subnets. Firewalls should be in place at each tier: To protect incoming traffic from Internet attacks, web application firewalls filter out undesirable traffic (see **Figure 2-2**). Each web and database server should be also protected by firewalls that allow only the required traffic through.

Each subnet should be secured with network access controls that allow the required traffic and deny all other requests. Encryption should be deployed for both data in transit and data at rest.

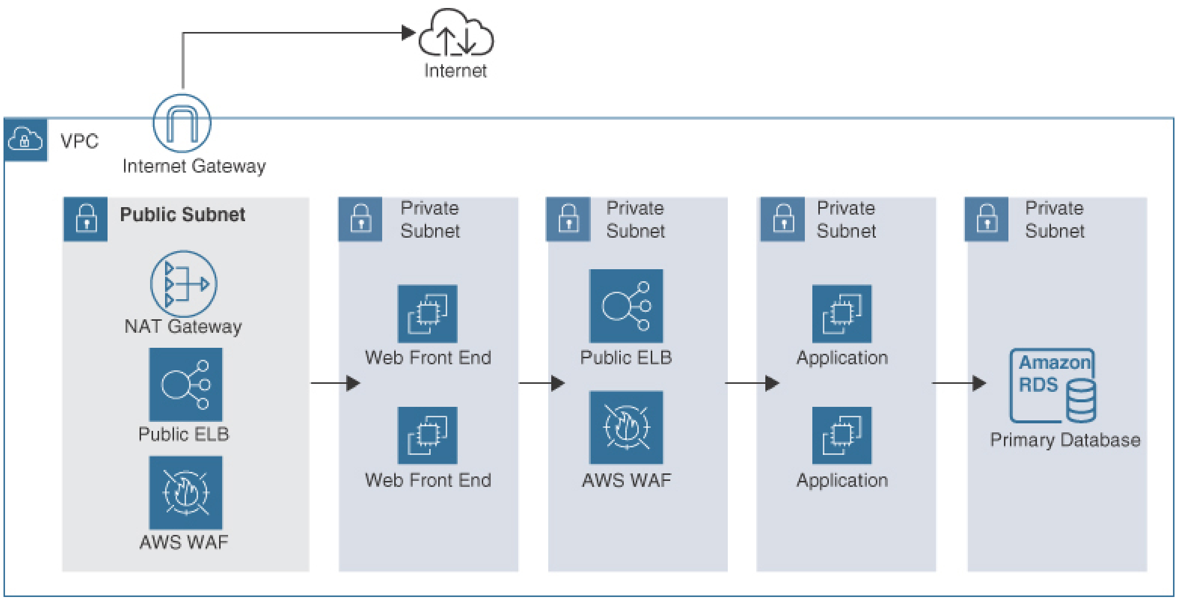
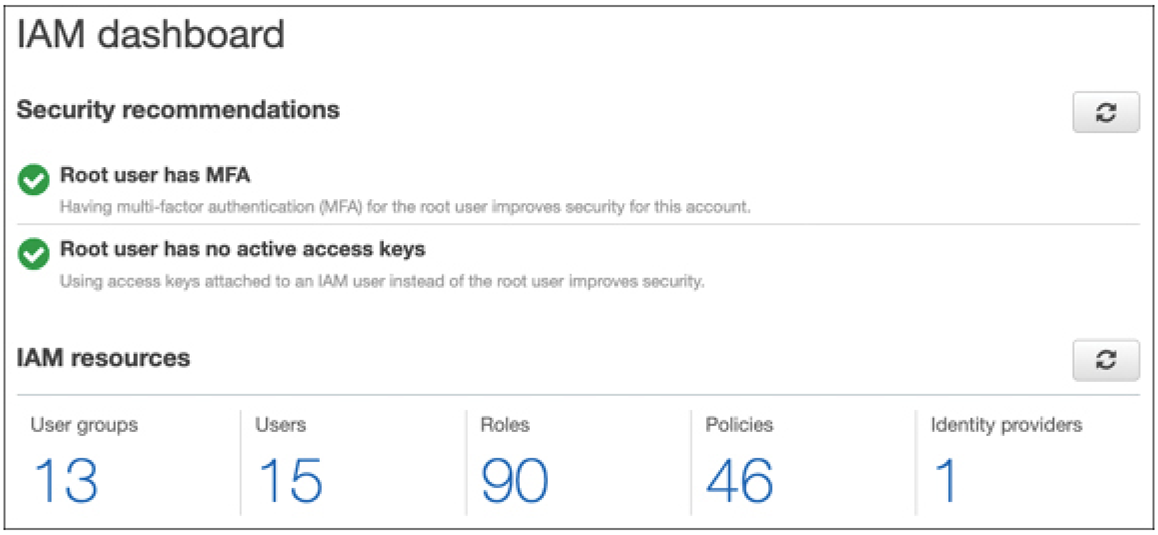


Figure 2-2 Defense in Depth Using AWS Services

Other security strategies include implementing the principle of least privilege using identity and authorization controls. AWS Identity and Access Management (IAM) allows customers to create permission policies for users, groups, and roles for cloud administrators, cloud services, and end users that access cloud resources from their mobile devices (see **Figure 2-3**).


IAM dashboard

Security recommendations

Root user has MFA

Having multi-factor authentication (MFA) for the root user improves security for this account.

Root user has no active access keys

Using access keys attached to an IAM user instead of the root user improves security.

IAM resources

User groups	Users	Roles	Policies	Identity providers
13	15	90	46	1

Figure 2-3 Identity and Access Management Security Controls

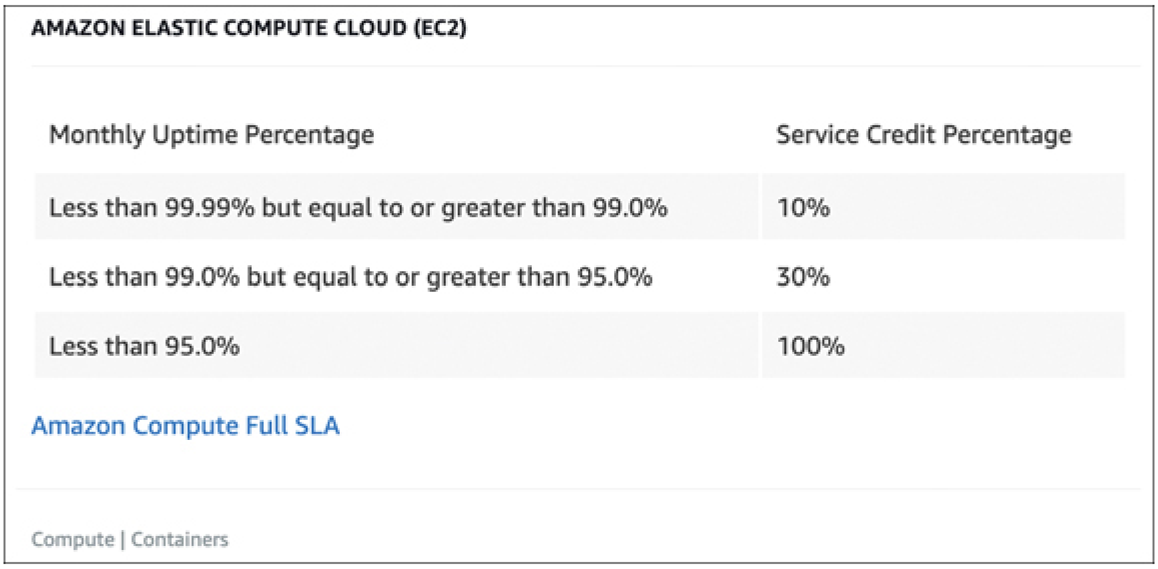
There are many security services you can enable in the cloud, allowing organizations to reap the benefits. For example, Amazon GuardDuty provides intelligent threat detection using machine learning to provide continuous monitoring of network traffic, DNS queries, and API calls, and protect access to RDS databases and Kubernetes deployments.

Key Topic

Reliability Pillar

Reliability is the most important requirement; without reliability, end users will eventually stop using the application. Each workload should be designed to minimize failure, keeping in mind there are many components in each application stack to consider. Over the past decade, many best practices have been published as to how to best deploy and manage workloads and associated AWS cloud services with a high degree of reliability.

Organizations must define the required level of reliability for each workload deployed in the AWS cloud. Cloud reliability is commonly defined as cloud service availability. Before the cloud, a **service-level agreement (SLA)** was an explicit contract with the provider that included consequences for missing the provider's service-level objectives. AWS SLAs indicate that they will do their best to keep their infrastructure and managed services up and available most of the time. Each AWS cloud service typically has a defined SLA that defines an operational uptime goal which AWS attempts to meet, and usually exceeds (see **Figure 2-4**). If failures occur, and they do occur from time to time, and you can prove that a workload was down because of AWS's failures, AWS will provide you with a credit on your bill. AWS SLAs can be found here: **https://aws.amazon.com/legal/service-level-agreements/**.



AMAZON ELASTIC COMPUTE CLOUD (EC2)

Monthly Uptime Percentage	Service Credit Percentage
Less than 99.99% but equal to or greater than 99.0%	10%
Less than 99.0% but equal to or greater than 95.0%	30%
Less than 95.0%	100%

Amazon Compute Full SLA

Compute | Containers

Figure 2-4 AWS Service-Level Agreements

SLA numbers that define cloud service availability look better than they really are. For example, demanding a desired application availability of 99.99% is designing for the potential unavailability over a calendar year of roughly 52 minutes of downtime. Because potential downtime does not include scheduled maintenance, when is this 52 minutes of downtime going to occur? That is the big question to which there is no guaranteed answer. If your workload is streaming video delivery, 99.99% is the recommended availability target to shoot for. If your workload processes ATM transactions, a maximum unavailability of 5 minutes per year, or five nines (99.999%), is the recommended availability target to shoot for. For an online software as a service (SaaS) application involving point-of-sale transactions, the recommendation is 99.95%, or roughly 4 hours and 22 minutes of downtime per year. Other considerations when calculating workload availability include workload dependencies and availability with redundant components.

- **Workload dependencies:** On-premises workloads will have hard dependencies on other locally installed services; for example, an associated database. Operating in the AWS cloud, if a dependent database fails, a backup or standby database service can be available for automatic failover. In the AWS cloud, workloads can more easily be designed with a reliance on what are defined as *soft dependencie*s. With each AWS region containing multiple availability zones, web and application servers deployed across multiple availability zones fronted by a load balancer provide a highly available design. Databases are also deployed across at least two availability zones, and the primary and secondary database servers are kept up to date with synchronous replication.

- **Availability using redundant components:** Designing with independent and redundant cloud services, availability can be calculated by subtracting the availability of the independent cloud services utilized by your workload from 100%. Operating a workload across two availability zones, each independent availability zone has a defined availability of 99.95%. Multiplied together and subtracted from 100%, availability is six nines, or 99.9999% (see **Figure 2-5**).

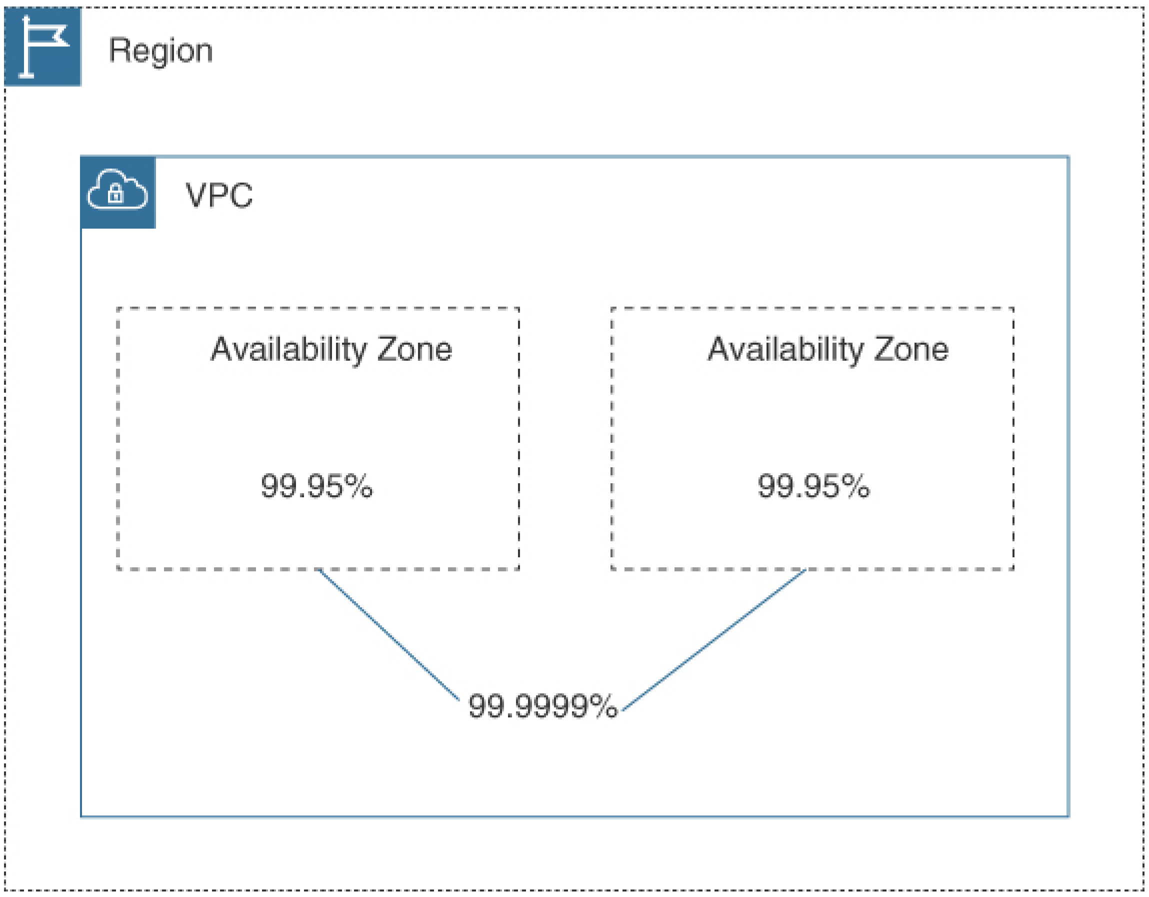


Figure 2-5 Multi-AZ Service-Level Agreements

Note

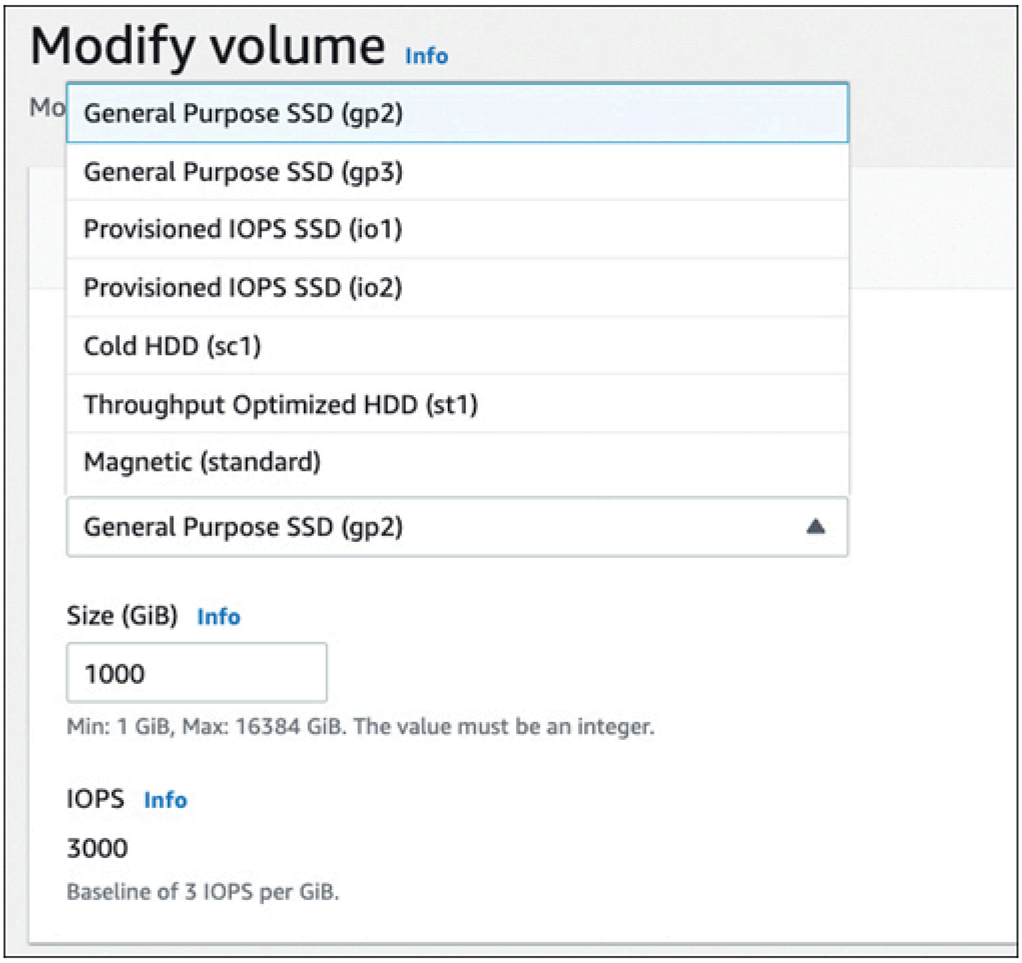
Applications that are designed for higher levels of availability will also have increased costs. Workloads designed with high availability will have multiple web, application, and database instances across multiple availability zones.

Performance Efficiency Pillar

Key Topic

In information technology systems engineering, design specialists characterize workloads as compute oriented, storage focused, or memory driven, designing solutions tuned to efficiently meet the design requirements. To achieve your performance goals, customers must address how the design of workload components can affect the performance efficiency of the entire application stack.

How can you design around a compute bottleneck? How do you get around a limit of the read or write rate of your storage? Operating in the AWS cloud, customers can change compute and network performance as simply as turning off their EC2 instances and resizing the EC2 instance, changing the memory, processor, and network specs. Amazon Elastic Block Store (EBS) storage volumes can be changed in size, type, and speed at a moment's notice, increasing volume size and performance as needs change (see Figure 2-6).


Modify volume Info

Mo

General Purpose SSD (gp2)
General Purpose SSD (gp3)
Provisioned IOPS SSD (io1)
Provisioned IOPS SSD (io2)
Cold HDD (sc1)
Throughput Optimized HDD (st1)
Magnetic (standard)

General Purpose SSD (gp2)

Size (GiB) Info

1000

Min: 1 GiB, Max: 16384 GiB. The value must be an integer.

IOPS Info

3000

Baseline of 3 IOPS per GiB.

Figure 2-6 Resize EBS Volumes

Maximizing performance efficiency depends on knowing your workload requirements over time. Measure workload limitations and overall operation by closely monitoring all aspects of the associated cloud services. After several days, weeks, and months of analyzing monitoring data for the computer, networking, and storage services utilized in your application stack, developers and operations teams will be able to make informed decisions about required changes and improvements to the current workload design. Learn where your bottlenecks are by carefully monitoring all aspects of your application stack.

Performance efficiency can also be improved with parallelism, scalability, and improved network speeds:

- Parallelism can be designed into many systems so that you can have many instances running simultaneously. Workload transactions can be serialized using SQS caches or database read replicas.
- Scalability in the cloud is typically horizontal. But scale can also be vertical by increasing the size, compute power, or transaction capacity on web, app, or database processing instances or containers. Customers may find success by increasing the sizes of both the compute and networking speeds of database instances without having to rebuild from scratch, if this is a solution that can be carried out relatively quickly.
- Networking is critical to performance engineering in the cloud, because the host architecture that AWS offers its cloud services on, and the EC2 instances and storage arrays used for all workloads, are all networked. Networking speeds across private networks of AWS can reach 200 Gbps. Connecting to the AWS cloud from an on-premises location privately using VPN connections max out at 1.25 Gbps. Utilizing high-speed fiber AWS Direct Connections to the AWS cloud range from 1 to 100 Gbps.

Key Topic

Cost Optimization Pillar

Customers want to optimize their returns on investments in AWS cloud technologies. Successfully reducing cloud costs starts with assessing your true needs. Monitoring is essential to continually improving your performance efficiency, and it's also the key to unlocking cost benefits over time. AWS provides many cost tools for monitoring costs, including cost estimators, trackers, and machine learning–based advisors. Many of these services are free, such as AWS Cost Explorer or AWS Cost and Usage Report.

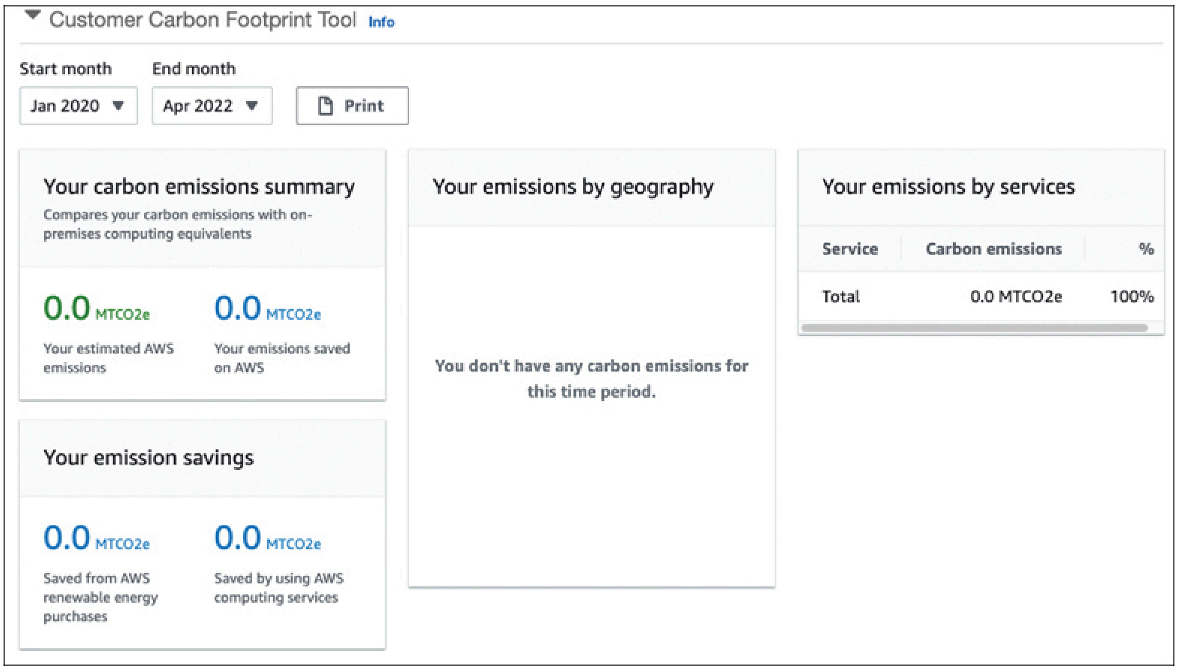
When analyzing the daily operation of services such as compute and storage, autoscaling and true consumption spending are not the default configuration. Insights into spending trends and rates allow you to control what you spend in the cloud.

Sustainability Pillar

The Sustainability pillar addresses the impact of your workload deployments against long-term environmental issues such as indirect carbon emissions (see Figure 2-7) or environmental damage caused by cloud services. Because workload resources rely on cloud services that are virtual compute and storage services, areas where improvements in sustainability include energy consumption and workload efficiency in the following areas:

- Utilizing indirect electricity to power workload resources. Consider deploying resources in AWS regions where the grid has a published carbon intensity lower than other AWS regions.
- Optimizing workloads for economical operations. Consider scaling infrastructure matching user demand

and ensuring only the minimum number of resources required are deployed.
- Minimizing the total number of storage resources used by each workload.
- Utilizing managed AWS services instead of existing infrastructure.


Customer Carbon Footprint Tool Info

Start month: Jan 2020
End month: Apr 2022
Print

Your carbon emissions summary
Compares your carbon emissions with on-premises computing equivalents

0.0 MTCO2e Your estimated AWS emissions
0.0 MTCO2e Your emissions saved on AWS

Your emission savings

0.0 MTCO2e Saved from AWS renewable energy purchases
0.0 MTCO2e Saved by using AWS computing services

Your emissions by geography

You don't have any carbon emissions for this time period.

Your emissions by services

Service	Carbon emissions	%
Total	0.0 MTCO2e	100%

Figure 2-7 Customer Carbon Footprint Tool

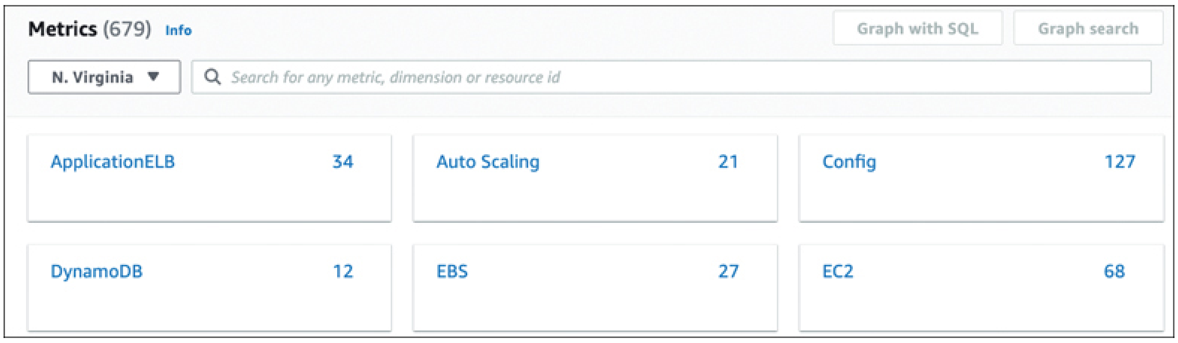
Designing a Workload SLA

Organizations must design workload SLAs that define the level of workload reliability they require and are willing to pay for. It should be noted that AWS cloud services are on-line, very reliable, and up most of the time. The onus is on each AWS customer to design workloads to be able to minimize the effects of any failures of the AWS cloud services used to create application stacks. Note that each cloud service, such as storage and compute, that is part of your workload application stacks has its own separately defined SLA.

The AWS cloud services that are included in our workloads need to operate at our defined acceptable level of reliability; in the cloud industry this is defined as a **service-level objective (SLO)** and is measured by a metric called a **service-level indicator (SLI)**. For example, web servers must operate at a target of between 55% and 65% utilization. By monitoring the CPU utilization of the web servers, you can be alerted using an Amazon CloudWatch metric linked to an event-driven alarm when the utilization exceeds 65%. You could manually add additional web servers, or EC2 Auto Scaling could be used to automatically add and remove additional web servers as required. There are numerous CloudWatch metrics that can be utilized for more than 70 AWS cloud services providing specific operational details and alert when issues occur.

Key Topic

Ongoing CloudWatch monitoring should be used to monitor each integrated cloud service for calculating the reliability of the workload as a whole using the cloud service metrics (see **Figure 2-8**). Each metric can be monitored over a defined time period, which can range from seconds to weeks; the default time period is typically 5 minutes. Every month the average amount of workload availability can be calculated by dividing the successful responses against all requests. By monitoring all integrated cloud services of a workload, valuable knowledge will be gathered regarding reliability issues, potential security threats, and workload performance.



ApplicationELB	34
Auto Scaling	21
Config	127
DynamoDB	12
EBS	27
EC2	68

Figure 2-8 CloudWatch Metrics

Service-level indicators are invaluable for all cloud services; here are a few examples to consider:

- **Availability:** The amount of time that the service is available and usable
- **Latency:** How quickly requests can be fulfilled
- **Throughput:** How much data is being processed; input/output operations per second (IOPS)
- **Durability:** The likelihood data written to storage can be retrieved in the future

Reliability and Performance Are Linked

Safety, testability, quality, maintainability, stability, durability, and availability are all aspects of a workload's overall reliability. Reliability is the critical design consideration. For example, if a workload crashes periodically but reboots and carries on, persistent end users who retry their requests might get their results eventually. But besides the obvious reliability issue, there is also a performance issue. Your workload's effective performance is lower because of the required retries.

An unresponsive website will eventually cause customer dissatisfaction, which negatively impacts trust and the reputation of the application. If it leads prospective or established customers to shop elsewhere, that could result in lost potential business. Maintaining redundant cloud services to improve workload reliability and performance will result in additional operational expense for some cloud services, such as multiple EC2 instances and multiple database instances providing redundant storage.

When designing for reliability, it's important to realize that not all workload dependencies will have the same impact when they fail. An outage for an application stack designed with some hard dependencies, such as a single primary database with no alternate database as a backup, will obviously cause problems that cannot be ignored when failure occurs. An outage with a soft dependency, such as an alternate database read-replica, will hopefully have no short-term impact on regular workload operation. Workload reliability can also positively affect overall performance. With the use of multiple availability zones utilizing separate physical data centers separated by miles, workloads can easily achieve a level of reliability and high availability as the web servers, and primary and alternate database servers, are hosted in separate physical locations. Database records can be kept up to date using synchronous replication between the primary and alternate database instances or storage locations.

Disaster Recovery

In addition to defining your availability objectives, you should consider disaster recovery (DR) objectives. How is each workload recovered when disaster occurs? How much data can you afford to lose, and how quickly must you recover? The application's acceptable **recovery time objective (RTO)** and **recovery point objective (RPO)** must be defined and then tested to ensure that the application meets and possibly exceeds the desired service-level objectives. Both the RTO and RPO for each workload need to be defined by your organization. RTO is the maximum acceptable delay between the interruption of an application and the restoration of service. RPO is the maximum acceptable amount of data loss.

Placing Cloud Services

It's critical that you choose where each workload component resides and operates. Some cloud services, such as DNS name resolution and traffic routing and content delivery networks (CDNs), are globally distributed across the world. But most AWS cloud services are regional in design. That is, they are *hosted* in one particular geographical location, even if they might be *accessible* globally. Techniques such as replication, redirection, and load balancing allow you to deploy workload cloud services as multi-region architecture.

Exam questions will ask you to consider several options when deciding where each workload and associated cloud services should be located to best meet the needs of the question's scenario: host location, data caching, data replication, load balancing, and failover architecture that is required.

Data Residency and Compute Locations

Running workloads in the cloud is essentially leasing time on storage and compute power in a cloud provider's data centers. Each cloud provider hosts services in regions throughout the world. For example, Amazon, Google, and Microsoft each host their cloud services somewhere in the state of Virginia, in a region near Tokyo, and in dozens of other regions around the globe.

How do you choose a region to host your services in? The first suggestion is to consider data residency. Do you have compliance guidelines, laws, or underwriters that suggest or dictate that you store your data within a certain country, state, or province? That might be your sole consideration. If data residency isn't strictly mandated or multiple AWS regions don't meet your criteria, you could instead place your data close to your customers or to your own facilities. Those regions might not be the same geography, depending on the nature of your business and markets you serve.

Let's use an example of a fictitious business called Terra Firma based in Winnipeg, Ontario, which is in the middle of Canada. Let’s assume that Terra Firma has deployed its customer portal website in the AWS cloud somewhere in the central Canada region, near its offices and the majority of its users. If customers in the central Canada region have sufficiently fast, low-latency Internet connectivity to this AWS cloud region, their user experience could be adequate with the hosted website portal. Let’s look next at whether caching could be a benefit to the Terra Firma website portal.

Caching Data with CDNs

Key Topic

CDNs serve up temporary copies of data to the client in order to improve effective network performance. Architecturally, the CDN servers are distributed around many global service areas. In the case of modern cloud CDNs, the service area is global; AWS hosts a global CDN world-wide *cache* called Amazon CloudWatch. How can a CDN cache benefit the web portal users?

Without a CDN cache, the web browsers of Terra Firma’s central Canadian customers would send requests to the website address hosted at the cloud region hundreds of miles away. The speed of the website hosting and backend storage and databases for the site are a factor in the end-user experience. But the Internet latency from each user’s Internet connection to the AWS cloud region chosen by

Terra Firma can significantly contribute to the performance and overall experience.

Let’s assume that Terra Firma’s AWS cloud provider CloudFront has a CDN point of presence (POP) located in downtown Winnipeg. If Terra Firma’s cloud architects and operations staff configured their website to use a CDN, their customers could benefit from a faster user experience. Customer Julian’s web browser queries for Terra Firma’s web address and receives a response from the CDN POP location in Winnipeg. Julian’s browser next sends a request to load the website to that local POP in Winnipeg, which is a few miles away through the local Internet gateway instead of hundreds of miles and several network hops away to the location of the web server.

If Julian is the first user in the last 24 hours to load any of the requested files, the Winnipeg CDN POP won’t find the files in its temporary cache. The POP would then relay a web request to the website origin address hosted in the cloud region hundreds of miles away.

When another customer in the Winnipeg region, Jan, visits the web portal site, her browser resolves the site name to the Winnipeg CDN POP just as Julian’s had done recently. For each file that is still cached locally, the POP will send back an immediate local response; without consulting the web server hundreds of miles away, Jan gets a super-fast experience. And the hundreds, thousands, or millions of other customers who visit the Terra Firma web portal can get this performance benefit as well. AWS CloudFront, AWS's CDN, provides hundreds of POPs around the world (see **Figure 2-9**). Wherever users are located, there is likely a POP locally or within the same geographical area. Users close to the actual website AWS cloud region can also be serviced by a POP in that cloud region. People far away will get the same benefits once another end user in their area visits the Terra Firma website, which will automatically populate their local POP's cache.

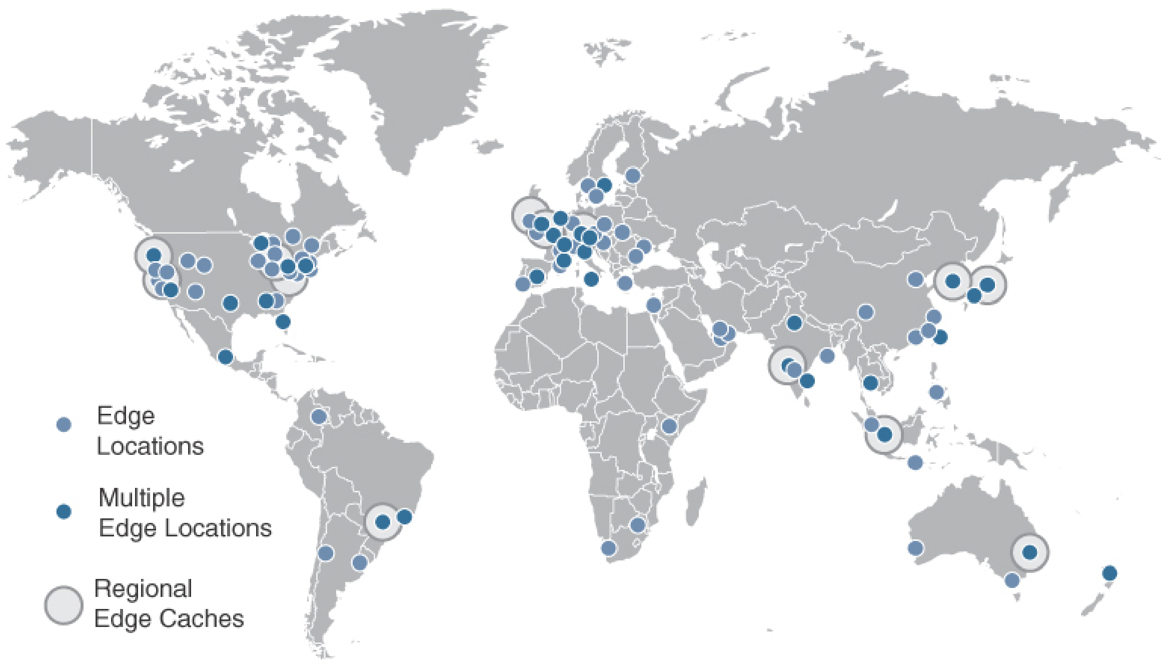


Figure 2-9 Amazon CloudWatch Edge Locations

Note

CDN caching techniques must be explicitly programmed and configured in your application code and in the Amazon CloudWatch distribution. Deploying a CDN is designed to improve performance and alleviate the load on the origin services, as in our example of a web portal.

Data Replication

Although CloudFront focuses on performance, a CDN provides substantial security, reliability, cost, and operations benefits. But what if your workload design also stores persistent replica copies of your data, resulting in multiple copies of data? Let’s look at the benefits of cloud storage and replicated data records.

The applications, cloud services, and data records that make up your workload solutions must be reliable. A single database or website is a single point of failure that could bring down the entire workload. Organizations don’t want their business to fail, so they need to architect and engineer systems to avoid any single points of failure. To improve reliability in cloud solutions architectures, customers need to make choices about redundancy, replication, scaling, and failover.

AWS provides multiple replicas of data records that are stored in the cloud. Three copies is a common default, with options for greater or lesser redundancy depending on the storage service chosen. There are additional options for expanding or reducing additional data replication. Storage and replication are not free; customers must pay for the used storage infrastructure consumed per month. Network transfer costs are billed relative to the rate of change and duplicate copies are billed based on the stored capacity used. Always check with AWS for up-to-date and region-specific pricing of storage, databases, and replication for these services. Deploy the required configurations at AWS to enable the desired redundancy for every database, data lake, and storage requirement:

- Replicate within an availability zone if necessary
- Replicate between availability zones within your primary AWS region
- Replicate between your chosen primary AWS cloud region and one or more secondary regions

Operating in the AWS cloud requires security and replication at all levels. Customers should also consider deploying compute containers, virtual machine instances, or clusters of either to multiple availability zones and across regions to meet their needs.

Load Balancing Within and Between Regions

When we have more than one replica of an EC2 instance or containerized application, we can load balance requests to any one of those replicas, thereby balancing the load across the replicas (see **Figure 2-10**). Read operations, queries, database select statements, and many compute requests can be load balanced, as can writes to multi-master database systems.

- Network load balancers use Internet protocol addresses (IPv4 or IPv6), with a choice of TCP, UDP, or both on top of IP and with port numbers to identify the application or service running. Rules stating how to block or relay network traffic matching the source and target IP address, protocol, and port patterns dictate the load-balancing process.

- Application load balancers indicate that HTTP or HTTPS traffic is being load balanced, with balancing rules based on web URLs with rule choices of HTTP or HTTPS, domain name, folder and file name, web path, and query strings.

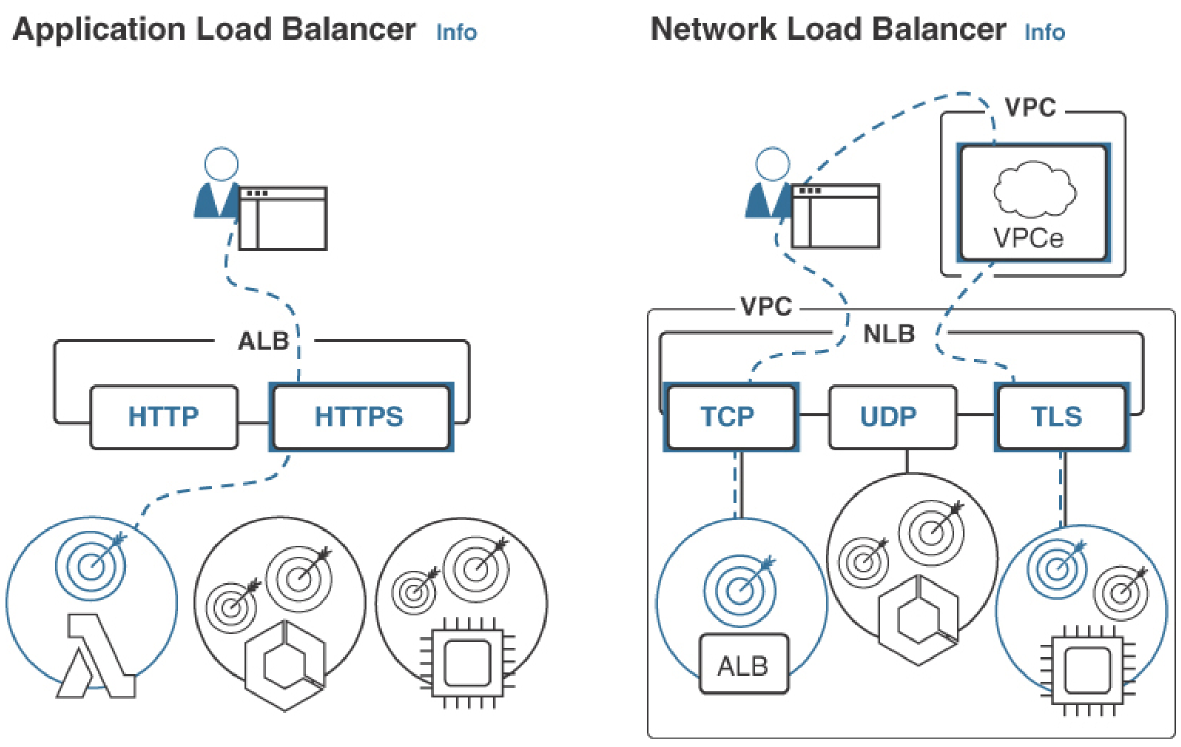


Figure 2-10 Load Balancer Options

Before load-balancing associations and connections can be established at load balancers, network communications using Internet technologies must perform name resolution. The Domain Name System (DNS) Amazon Route 53 service is used to *resolve* a domain name like www.pearson.com into other domain names (canonically), or into the IPv4 or IPv6 addresses needed to communicate. Rules and policies can be associated with a domain name so that round-robin, priority, failover, weighted, or geolocation will select the results for one end user versus another. Amazon Route 53 also provides *inter-region* load balancing across AWS regions.

One of the key advantages of the Network, Application, and Route 53 load balancers is that they can monitor the health of the targets they have been configured to distribute queries to. This feedback enables the load balancers to provide actual *balancing* of the load based on health checks, affinity, stickiness, and security filtering in order to better truly balance the load on your replicas, which allows you to better achieve cost optimization and performance efficiency. Load balancers are also key to application reliability as well.

Failover Architecture

Failover is a critical aspect of load-balancing resources. The ability to have your solutions *detect* failures and *automatically* divert to an alternate replica is essential. With *automatic failover*, you can quickly switchover and reassign requests to the surviving replicas. For database systems, primary–secondary relationships should be designed and deployed.

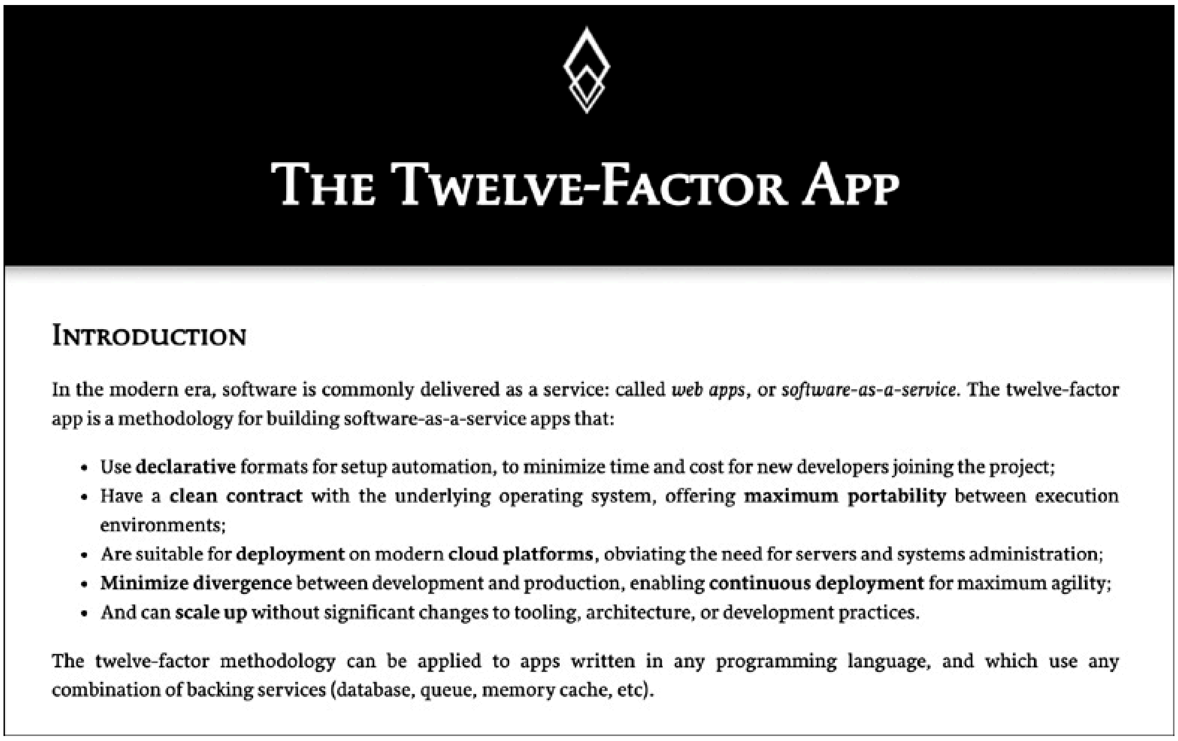
Failover allows for business continuity in the event of cloud service failure. If there were only two replicas of the application in question and one replica failed, in the failover state you are running without any redundancy. Customers must restore redundancy by either repairing the failed component and bringing it back online or by replacing it. If you deploy triple or greater degrees of redundancy in the first place, recovery from a failure state is not as immediate a concern.

Failover is a general topic that is not just limited to load-balancing technologies. Additional strategies of failover will be addressed in later chapters. It's important to note that replication, load balancing, and failover are related features that must be orchestrated so that they work in concert with one another. When configured properly, customers can achieve and maintain advantages in security, reliability, and workload performance.

Deployment Methodologies

Developers getting ready to create their first application in the cloud can look to a number of rules that are generally accepted for successfully creating applications that run exclusively in the public cloud.

Several years ago, Heroku cofounder Adam Wiggins released a suggested blueprint for creating native SaaS applications hosted in the public cloud, called the Twelve-Factor App Methodology. Heroku (**https://www.heroku.com/**) is a platform as a service provider (PaaS) owned by Salesforce and hosted at AWS. Heroku was attempting to provide guidance for SaaS applications created in the public cloud based on their real-world experience. Additional details on this methodology can be found at **https://12factor.net/** (see **Figure 2-11**).


THE TWELVE-FACTOR APP

INTRODUCTION

In the modern era, software is commonly delivered as a service: called *web apps*, or *software-as-a-service*. The twelve-factor app is a methodology for building software-as-a-service apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).

Figure 2-11 The 12 Factor App Methodology

These guidelines can be viewed as a set of best practices to consider using when deploying applications at AWS that align with the AWS Well-Architected Framework. Depending on your deployment methods, you may quibble with some of the factors—and that's okay. There are many complementary management services hosted at AWS that greatly speed up the development and deployment process of workloads that are hosted at AWS. The development and operational model that you choose to embrace will fol-low one of these development and deployment paths:

- **Waterfall:** In this model, deployment is broken down into phases, including proper analysis, system design, implementation and testing, deployment, and ongoing maintenance. In the waterfall model, each of these phases must be completed before the next phase can be-gin. If your timeline is short, and all the technologies to be used in hosting and managing your workload are fully understood, then perhaps this model can still work in the cloud. However, cloud providers are introducing many cloud services that free you from having to know every technical detail as to how the service works; instead, you can just use the cloud service as part of your workload deployment. For example, an Amazon S3 bucket is used for unlimited cloud storage, without customers without customers needing to know the technical details of the S3 storage array. In the AWS cloud, all the infrastructure components for storage and compute are already online and functional; you don't have to build storage arrays or even databases from scratch; you can merely order or quickly configure the service, and you are off and running. When developing in the cloud, if your timeline for development is longer than six months, most hosted cloud services will have changed and improved in that time frame, forcing you to take another look at your design and deployment options.

- **Agile:** In this model, the focus is on process adaptability, and teams can be working simultaneously on the planning, design, coding, and testing processes. The entire process cycle is divided into a relatively shorter time frame, such as a 1-month duration. At the end of the first month, the first build of the product is presented to the potential customers, feedback is provided, and is incorporated into the next process cycle and the second version of the product. This process continues until a final production version of the product is delivered and accepted. This process might continue indefinitely if an application has continual changes and updates. Think of any cloud application installed on your personal devices and consider how many times that application gets updated. It's probably updated every few months at a minimum; for example, the Google Chrome browser updates itself at least every couple of weeks. AWS has a number of cloud services that can help with Agile deployments, including AWS Cloud9, AWS CloudFormation, AWS CodeCommit, AWS CodeBuild, and AWS CodePipeline.

- **Big Bang:** In this model, there is no specific process flow; when money is available in the budget, development starts, and eventually software is developed. This model can work in the cloud because there are no capital costs to worry about; work can proceed when there is a budget. But without proper planning and a full understanding of requirements of the application, long-term projects may have to be constantly revised over time due to changes in the cloud and changes from the customer.

Before deploying applications in the AWS cloud, you should carefully review your current development process and perhaps consider taking some of the steps proposed in the Twelve-Factor App Methodology described in the following sections. Applications that are hosted in the AWS cloud need the correct infrastructure; as a result, the rules for application deployment in the AWS cloud don't stand alone. Cloud infrastructure adhering to the principles of the AWS Well-Architected Framework is also a necessary part of the rules. The following sections look at the applicable factors of the Twelve-Factor App Methodology from the infrastructure point of view and also identify the AWS services that can help with adhering to each factor. This discussion can help you understand both the factors and the AWS services that are useful in application development and deployment and help you prepare for the exam with the right mindset.

Factor 1: Use One Codebase That Is Tracked with Version Control to Allow Many Deployments

In development circles, this factor is nonnegotiable; it must be followed. Creating an application usually involves three separate environments: development, testing, and production (see **Figure 2-12**). The same codebase should be used in each environment, whether it's the developer's laptop, a set of EC2 instances in the test environments, or the production EC2 instances. Each version of application code needs to be stored separately and securely in a safe location. Multiple AWS environments can take advantage of multiple availability zones and multiple VPCs to create dev, test, and production environments.

Key Topic

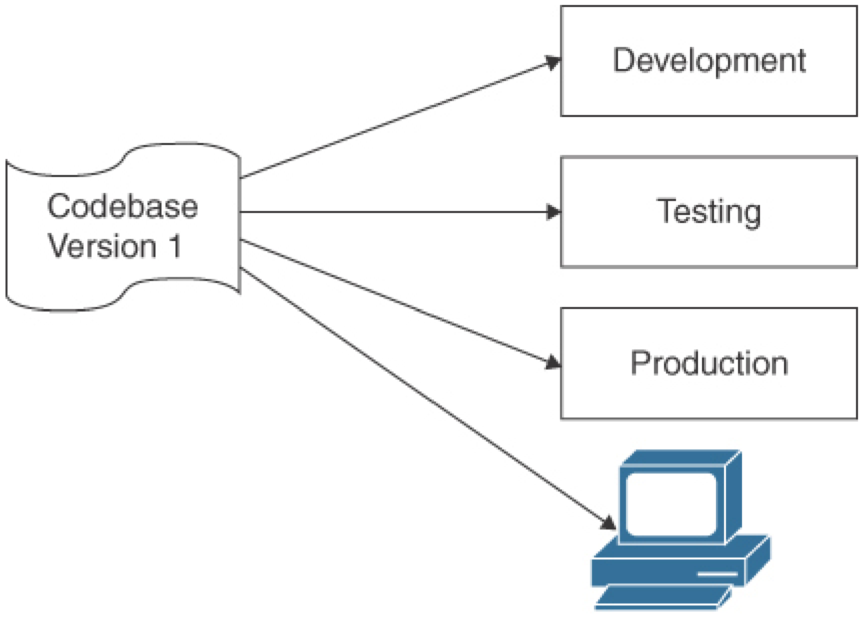


Figure 2-12 One Codebase, Regardless of Location

Developers typically use code repositories such as GitHub to store their code. Operating systems, off-the-shelf software, dynamic link libraries (DLLs), development environments, and application code are always defined by a specific version number. As your codebase undergoes revisions, each revision of each component needs to be tracked; after all, a single codebase might be responsible for thousands of deployments, and documenting and controlling the separate versions of the codebase just makes sense. Amazon has a code repository, called AWS CodeCommit, for applications developed and hosted at AWS.

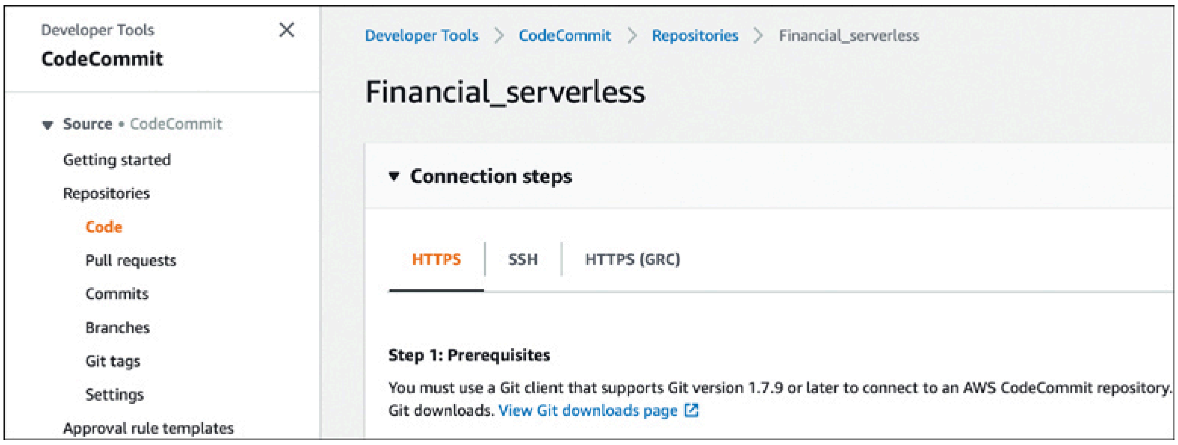
At the infrastructure level at Amazon, it is important to consider all dependencies. The AWS infrastructure components to keep track of include the following:

- **AMIs:** Images for web, application, database, and appliance instances. Amazon Machine Images (AMI) should be version controlled and immutable.

- **Amazon EBS volumes:** Boot volumes and data volumes should be tagged by version number for proper identification and control.
- **Amazon EBS snapshots:** Snapshots used to create virtual server boot volumes are also part of each AMI.
- **Container images:** Each private container image can be stored in the Amazon Elastic Container Registry (ECR) and protected with Identity and Access Management (IAM) permission policies. Container images could be stored in the Amazon Elastic Container Registry.
- **Serverless applications:** The AWS Serverless Application Repository can be used to store, share, and assemble serverless architectures.

AWS CodeCommit

CodeCommit is a hosted AWS version control service with no storage size limits (see Figure 2-13). It allows AWS customers to privately store their source code and binary code, which are automatically encrypted at rest and at transit, at AWS. CodeCommit allows customers to store code versions at AWS rather than at Git without worrying about running out of storage space. CodeCommit is also HIPAA eligible and supports PCI DSS and ISO/IEC 27001 standards.



Step 1: Prerequisites

You must use a Git client that supports Git version 1.7.9 or later to connect to an AWS CodeCommit repository. Git downloads. View Git downloads page

Figure 2-13 A CodeCommit Repository

CodeCommit supports common Git commands and, as mentioned earlier, there are no limits on file size, type, and repository size. CodeCommit is designed for collaborative software development environments. When developers make multiple file changes, CodeCommit manages the changes across multiple files. Amazon S3 buckets also support file versioning, but S3 versioning is meant for recovery of older versions of files; it is not designed for collaborative software development environments; as a result, S3 buckets are better suited for storing files that are not source code.

Factor 2: Explicitly Declare and Isolate Dependencies

A workload deployed in development, testing, and production VPC networks at AWS may require specific components, such as a MySQL database, a specific operating system version, and a particular utility, and monitoring agent. All dependencies for each workload must be documented so that developers are aware of the version of each component required by the application stack. Deployed applications should never rely on the assumed existence of required system components; dependencies need to be declared and managed by a dependency manager, ensuring that the required dependencies are installed with the codebase. Examples of dependency managers are Composer, which is used with PHP projects, and Maven, which can be used with Java projects. Dependency managers use a configuration database to keep track of the required version of each component, and what repository to retrieve it from. If there is a specific version of system tools that the codebase always requires, perhaps the system tools could be added to the operating system that the codebase will be installed on. However, over time, software versions for every component will change. The benefit of using a dependency manager is that the versions of your dependencies will be the same versions used in the development, testing, and production environments.

If multiple operating system versions are deployed, the operating system and its feature set can also be controlled by AMI versions. Several AWS services work with versions of AMIs, application code, and deployment of AWS infrastructure stacks:

- **AWS EC2 Image Builder:** Simplify the building, testing, and deployment of virtual machines and container images at AWS and on premises.
- **AWS CodeCommit:** Can be used to host different versions of the application code.
- **AWS CloudFormation:** Includes several helper scripts to automatically install and configure applications,

packages, and operating system services that execute on EC2 Linux and Windows instances. The following are a few examples of these helper scripts:

- **cfn-init:** This script can install packages, create files, and start operating system services.
- **cfn-signal:** This script can be used with a wait condition to synchronize installation timings only when the required resources are installed and available.
- **cdn-get-metadata:** This script can be used to retrieve metadata from the EC2 instance's memory.

Factor 3: Store Configuration in the Environment

Your codebase should be the same when it is running in the development, testing, and production network environments. However, your database instances will have different paths, or URLs, when connecting to testing or development environments. Other configuration components, such as API keys, plus database credentials for access and authentication, should never be hard-coded per environment. Use AWS Secrets Manager to store database credentials and secrets. Create IAM roles to access data resources at AWS, including S3 buckets, DynamoDB tables, and RDS databases. You can use Amazon API Gateway to host your APIs.

Development frameworks define environment variables through the use of configuration files. Separating your application components from the application code allows you to reuse your backing services in different environments, using environment variables to point to the desired resource from the development, testing, or production environment. Amazon has a few services that can help centrally store application configurations:

- **AWS Secrets Manager:** This service allows you to store application secrets such as database credentials, API keys, and OAuth tokens.
- **AWS Certificate Manager (ACM):** This service allows you to create and manage public Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates used for any hosted AWS websites or applications. ACM also enables you to create a private certificate authority and issue X.509 certificates for identification of IAM users, EC2 instances, and AWS services.
- **AWS Key Management Service (AWS KMS):** This service can be used to create and manage encryption keys.
- **AWS CloudHSM:** This service provides single-tenant hardware security modules allowing organizations to generate and manage their own encryption keys at AWS.
- **AWS Systems Manager Parameter Store:** This service stores configuration data and secrets for EC2 instances, including passwords, database strings, and license codes.

Factor 4: Treat Backing Services as Attached Resources

Many cloud infrastructure and platform services at AWS can be defined as backing services accessed by HTTPS private endpoints connected over the AWS private network.

These include databases (for example, Amazon Relational Database Service [RDS], DynamoDB), shared storage (for example, Amazon S3 buckets, Amazon Elastic File System [EFS]), Simple Mail Transfer Protocol (SMTP) services, queues (for example, Amazon Simple Queue Service [SQS]), caching systems (such as Amazon ElastiCache, which manages Memcached or Redis in-memory queues or in-memory databases), and monitoring services (for example, Amazon CloudWatch, AWS Config, AWS CloudTrail).

Backing services should be completely swappable; for example, a MySQL database hosted on premises should be able to be swapped with a hosted copy of the database at AWS without requiring any changes to application code; for this example, the only variable that would change is the resource handle in the configuration file pointing to the database location.

Factor 5: Separate Build and Run Stages

Applications that will be updated on a defined schedule or at unpredictable times require defined stages during which testing can be carried out on the application state before it is approved and moved into production. AWS Elastic Beanstalk allows you to upload and deploy your application code combined with a configuration file that builds the AWS environment required for the application.

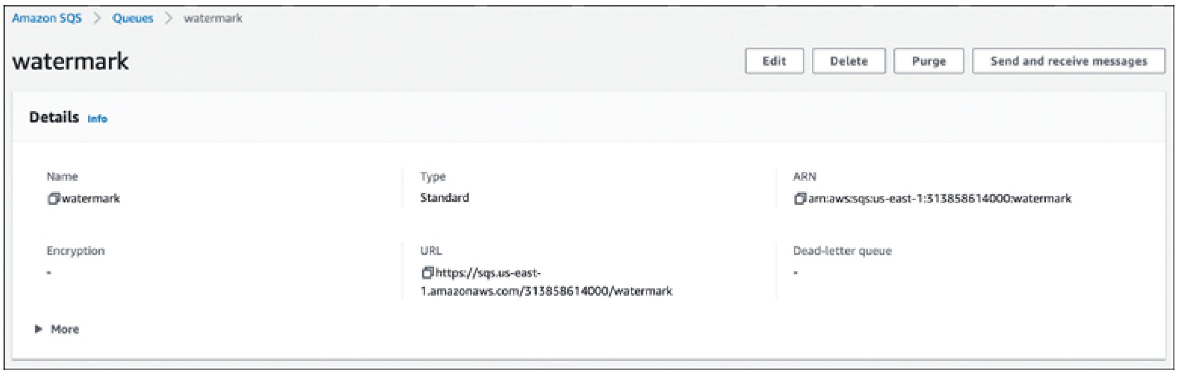
The Elastic Beanstalk build stage could retrieve your application code from a defined repo storage location, such as an Amazon S3 bucket. Developers could also use the Elastic Beanstalk CLI to push application code commits to AWS CodeCommit. When you run the CLI command **EB create** or **EB deploy** to create or update an EBS environment, the selected application version is pulled from the defined AWS CodeCommit repository and the application and required environment are uploaded to Elastic Beanstalk. Other AWS services that work with deployment stages include the following:

- **AWS CodePipeline:** This service provides a continuous delivery service for automating deployment of applications using multiple staging environments.
- **AWS CodeDeploy:** This service helps automate application deployments to EC2 instances hosted at AWS or on premises.
- **AWS CodeBuild:** This service compiles source code, runs tests on prebuilt environments, and produces code ready to deploy without having to manually build the test server environment.

Factor 6: Execute an App as One or More Stateless Processes

Stateless processes provide fault tolerance for the EC2 instances or containers running applications by separating the application data records and storing them in a centralized storage location such as an Amazon SQS message queue. An example of a stateless design is using an SQS message queue (see Figure 2-14). EC2 instances that are subscribed to the watermark SQS queue poll the queue, for any updates; when an update message is received, the server carries out the work of adding a watermark to the video and storing the modified video in S3 storage.

Key Topic



watermark

Details

Name	Type	ARN
watermark	Standard	arn:aws:sqs:us-east-1:313858614000:watermark
Encryption	URL	Dead-letter queue
-	https://sqs.us-east-1.amazonaws.com/313858614000/watermark	-

Figure 2-14 Using SQS Queues to Provide Stateless Memory-Resident Storage for Applications

Other stateless options available at AWS include the following:

- **AWS Simple Notification Service (SNS):** This hosted messaging service allows applications to deliver push-based notifications to subscribers such as SQS queues or Lambda.
- **Amazon MQ:** This hosted managed message broker service, specifically designed for Apache Active MQ, is an open-source message broker service that provides functionality similar to that of AWS SQS queues.
- **Amazon Simple Email Service (SES):** This hosted email-sending service includes an SMTP interface that allows you to integrate the email service into your application for communicating with an end user.

- **AWS Lambda:** This service is used for executing custom functions written by each customer for a vast variety of event-driven tasks. Examples include AWS Config custom rules for resolving infrastructure resources that fall out of compliance, and automated responses for Amazon Simple Notification Services (SNS), and Amazon EventBridge and CloudWatch alarms.
- **Amazon AppFlow:** This service enables you to exchange data between SaaS applications and storage services such as Amazon S3 or Amazon Redshift.
- **AWS Step Functions:** This service enables you to build and run workflows that coordinate the execution of multiple AWS services.

Factor 7: Export Services via Port Binding

Instead of using a local web server installed on a local server host and accessible only from a local port, you should make services accessible by binding to external ports where the services are located and accessible, using an external URL. For example, all web requests can be carried out by binding to an external port, where the web service is hosted and from which it is accessed. The service port that the application needs to connect to is defined by the development environment's configuration file (see the section "**Factor 3: Store Configuration in the Environment**," earlier in this chapter. The associated web service can be used multiple times by different applications and the different development, testing, and production environments.

Key Topic

Factor 8: Scale Out via the Process Model

If your application can't scale horizontally, it's not designed for dynamic cloud operation. Many AWS services are designed to automatically scale horizontally:

- **EC2 instances:** Instances and containers can be scaled with EC2 Auto Scaling and CloudWatch metric alarms.
- **Load balancers:** The Elastic Load Balancing (ELB) load balancer infrastructure horizontally scales to handle demand.
  - **Amazon S3 storage:** The S3 storage array infrastructure horizontally scales in the background to handle reads.
  - **Amazon DynamoDB:** The DynamoDB service scales tables within an AWS region. Tables can also be designed as global tables that are asynchronously replicated across multiple AWS regions. Each region table copy is horizontally scaled within the region as required. Each copy of the regional table in each region has a copy of all table data.
  - **Amazon Aurora Serverless:** The Amazon Aurora v2 serverless deployment of PostgreSQL or MySQL can be deployed across three availability zones per AWS region, or as a global datastore across multiple AWS regions supporting highly variable workloads.

Factor 9: Maximize Robustness with Fast Startup and Graceful Shutdown

User session information can be stored in Amazon ElastiCache or in in-memory queues, and application state can be stored in SQS message queues. Application configuration and bindings, source code, and backing services can be hosted by many AWS-managed services, each with its own levels of redundancy and durability. Data is stored in a persistent storage location such as S3 buckets, RDS databases, DynamoDB databases, or EFS or FSx for Windows File Server shared storage arrays. Workloads with no local dependencies and integrated cloud services can be managed and controlled by a number of AWS management services.

- **Elastic Load Balancer Service:** Load balancers targeting web application hosted on an EC2 instance stop sending requests when ELB health checks fail.
- **Amazon Route 53:** Regional workload failures can be redirected using Route 53 alias records to another region using defined traffic policies.
- **Amazon Relational Database Service (RDS):** When failure occurs, the RDS relational database instances automatically fail over to either the alternate or primary database instance. The failed database instance is automatically rebuilt and brought back online.
- **Amazon DynamoDB:** Tables are replicated across three availability zones throughout each AWS region.

- **EC2 Spot instances:** Spot instances can be configured to automatically hibernate when resources are taken back.
- **Amazon SQS:** SQS messages being processed by EC2 instances that fail are returned to the SQS work queue for reprocessing.
- **AWS Lambda:** Custom function can shut down tagged resources on demand.

Factor 10: Keep Development, Staging, and Production as Similar as Possible

With this factor, *similar* does not refer to the number of instances or the size of database instances and supporting infrastructure. Your development environment must be exact in the codebase being used but can be dissimilar in the number of instances or database servers being used. Aside from the infrastructure components, everything else in the codebase must remain the same.

- **AWS CloudFormation:** JSON or YAML template files can be used to automatically build AWS infrastructure with conditions that define what infrastructure resources to build for specific development, testing, and production environments.

Factor 11: Treat Logs as Event Streams

In development, testing, and production environments, each running process log stream must be stored externally. At AWS, logging is designed as event streams.

- **Amazon CloudWatch:** CloudWatch log groups or S3 buckets store EC2 instances' application logs. AWS CloudTrail event logs, which track all API calls to the AWS account, can also be streamed to CloudWatch logs for further analysis.

Factor 12: Run Admin/Management Tasks as One-Off Processes

Administrative processes should be executed using the same method, regardless of the environment in which the administrative task is executed. For example, an application might require a manual process to be carried out; the steps to carry out the manual process must remain the same, whether they are executed in the development, testing, or production environment.

Several AWS utilities can be used to execute administrative tasks:

- **AWS CLI:** Use the CLI to carry out administrative tasks with scripts.
- **AWS Systems Manager:** Apply OS patches and configure Linux and Windows systems.

Exam Preparation Tasks

As mentioned in the section "**How to Use This Book**" in the Introduction, you have a couple of choices for exam preparation: the exercises here, **Chapter 16**, "**Final**

Preparation," and the exam simulation questions in the Pearson Test Prep Software Online.

Review All Key Topics

Review the most important topics in the chapter, noted with the Key Topic icon in the outer margin of the page. Table 2-2 lists these key topics and the page number on which each is found.

Table 2-2 Chapter 2 Key Topics

Key Topic Element	Description	Page Number
Section	Operational Excellence Pillar	44
Section	Security Pillar	45
Section	Defense in Depth	45
Section	Reliability Pillar	47
Section	Performance Efficiency Pillar	49

Key Topic Element	Description	Page Number
Section	Cost Optimization Pillar	51
Paragraph	Regular CloudWatch monitoring	53
Section	Caching Data with CDNs	56
Figure 2-12	One Codebase, Regardless of Location	63
Figure 2-14	Using SQS Queues to Provide Stateless Memory-Resident Storage for Applications	68
Section	Factor 8: Scale Out via the Process Model	69

Define Key Terms

Define the following key terms from this chapter and check your answers in the glossary:

defense in depth

service-level agreement (SLA)

service-level objective (SLO)

service-level indicator (SLI)

recovery time objective (RTO)

recovery point objective (RPO)

dependencies

codebase

Q&A

The answers to these questions appear in **Appendix A**. For more practice with exam format questions, use the Pearson Test Prep Software Online.

1. Defense in depth can be divided into three areas: physical, technical, and _______________.

2. Application availability of 99.99% means designing for the potential unavailability of roughly 52 minutes of _______________.

3. Determine workload limits by _______________ all aspects of the application stack.

4. Changes in security can affect _______________.

5. Changes in reliability can affect _______________.

6. Availability is defined as the _______________ of time a cloud service is available and _______________.

7. If your application can't scale _______________, it's not designed for _______________ cloud operation.

8. AWS CloudFormation can be used to automatically build infrastructure using a single _______________.