Chapter 14
Architecture Best Practices

THE AWS CERTIFIED SOLUTIONS ARCHITECT ASSOCIATE EXAM OBJECTIVES COVERED IN THIS CHAPTER MAY INCLUDE, BUT ARE NOT LIMITED TO, THE FOLLOWING:

Domain 1.0: Designing highly available, cost-efficient, fault-tolerant, and scalable systems

✓ **1.1 Identify and recognize cloud architecture considerations, such as fundamental components and effective designs.**

Content may include the following:

- How to design cloud services
- Planning and design
- Familiarity with:
  - Best practices for AWS architecture
  - Hybrid IT architectures (e.g., AWS Direct Connect, AWS Storage Gateway, Amazon Virtual Private Cloud [Amazon VPC], AWS Directory Service)
  - Elasticity and scalability (e.g., Auto Scaling, Amazon Simple Queue Service [Amazon SQS], Elastic Load Balancing, Amazon CloudFront)

Introduction

For several years, software architects have created and implemented patterns and best practices to build highly scalable applications. Whether migrating existing applications to the cloud or building new applications on the cloud, these concepts are even more important because of ever-growing datasets, unpredictable traffic patterns, and the demand for faster response times.

Migrating applications to AWS, even without significant changes, provides organizations with the benefits of a secured and cost-efficient infrastructure. To make the most of the elasticity and agility possible with cloud computing, however, Solutions Architects need to evolve their architectures to take full advantage of AWS capabilities.

For new applications, AWS customers have been discovering cloud-specific IT architecture patterns that drive even more efficiency and scalability for their solutions. Those new architectures can support anything from real-time analytics of Internet-scale data to applications with unpredictable traffic from thousands of connected *Internet of Things (IoT)* or mobile devices. This leaves endless possibilities for applications architected using AWS best practices.

This chapter highlights the tenets of architecture best practices to consider whether you are migrating existing applications to AWS or designing new applications for the cloud. These tenets include:

- Design for failure and nothing will fail.
- Implement elasticity.
- Leverage different storage options.
- Build security in every layer.
- Think parallel.
- Loose coupling sets you free.
- Don’t fear constraints.

Understanding the services covered in this book in the context of these practices is key to succeeding on the exam.

Design for Failure and Nothing Fails

The first architecture best practice for AWS is the fundamental principle of designing for failure.

> *Everything fails, all the time*
>
> —*Werner Vogels, CTO, AWS*

Typically, production systems come with defined or implicit requirements in terms of uptime. A system is *highly available* when it can withstand the failure of an individual or multiple components. If you design architectures around the assumption that any component will eventually fail, systems won't fail when an individual component does. As an example, one goal when designing for failure would be to ensure an application survives when the underlying physical hardware for one of the servers fails.

Let's take a look at the simple web application illustrated in Figure 14.1. This application has some fundamental design issues for protecting against component failures. To start, there is no redundancy or failover, which results in single points of failure.

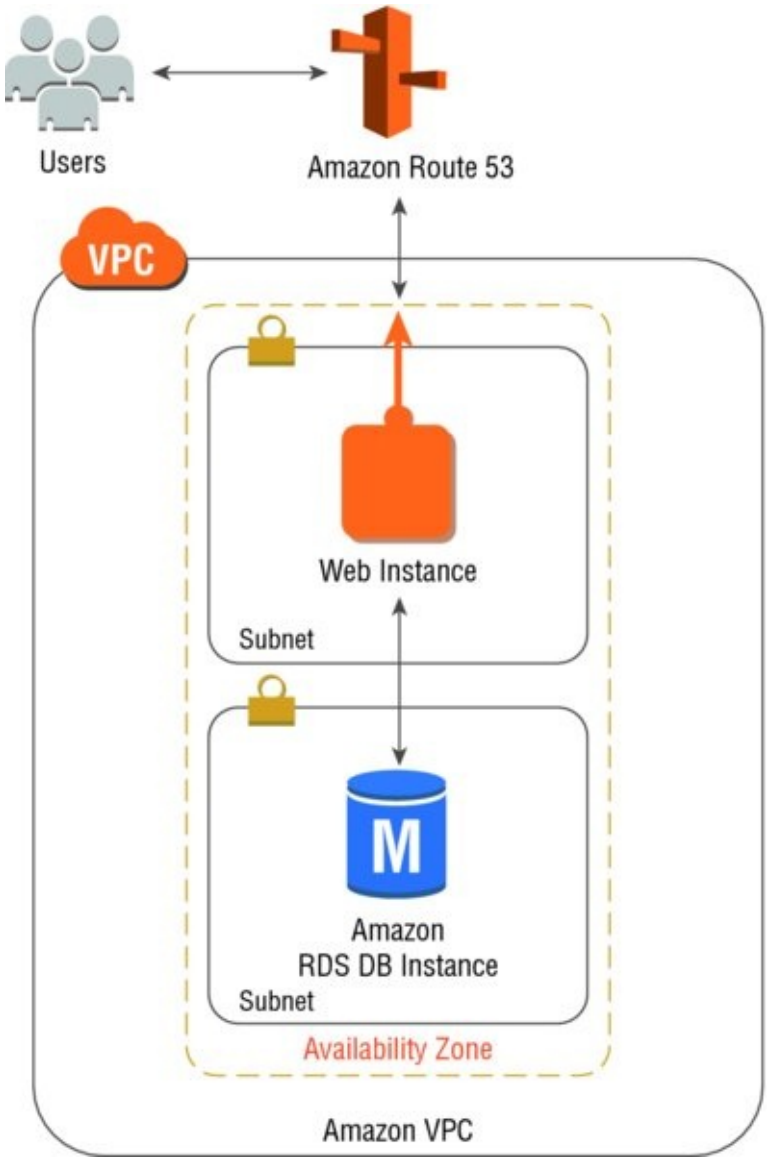

FIGURE 14.1 Simple web application architecture

- If the single web server fails, the system fails.
- If the single database fails, the system fails.
- If the Availability Zone (AZ) fails, the system fails.

Bottom line, there are too many eggs in one basket.

Now let's walk through transforming this simple application into a more resilient architecture. To begin, we are going to address the single points of failure in the current architecture. Single points of failure can be removed by introducing *redundancy*, which is having multiple resources for the same task. Redundancy can be implemented in either standby or active mode.

In standby redundancy when a resource fails, functionality is recovered on a secondary resource using a process called *failover*. The failover will typically require some time before it is completed, and during that period the resource remains unavailable. The secondary resource can either be launched automatically only when needed (to reduce cost), or it can be already running idle (to accelerate failover and minimize disruption). Standby redundancy is often used for stateful components such as relational databases.

In active redundancy, requests are distributed to multiple redundant compute resources, and when one of them fails, the rest can simply absorb a larger share of the workload. Compared to standby redundancy, it can achieve better utilization and affect a smaller population when there is a failure.

To address the redundancy issues, we will add another web instance and add a standby instance for Amazon Relational Database Service (Amazon RDS) to provide high availability and automatic failover. The key is that we are going to add the new resources in another AZ. An AZ consists of one or more discrete data centers. AZs within a region provide inexpensive, low-latency network connectivity to other AZs in the same region. This allows our application to replicate data across data centers in a synchronous manner so that failover can be automated and be transparent for the users.

Additionally, we are going to implement active redundancy by swapping out the Elastic IP Address (EIP) on our web instance with an Elastic Load Balancer (ELB). The ELB allows inbound requests to be distributed between the web instances. Not only will the ELB help with distributing load between multiple instances, it will also stop sending traffic to the affected web node if an instance fails its health checks. Figure 14.2 shows the updated architecture with redundancy for the web application.

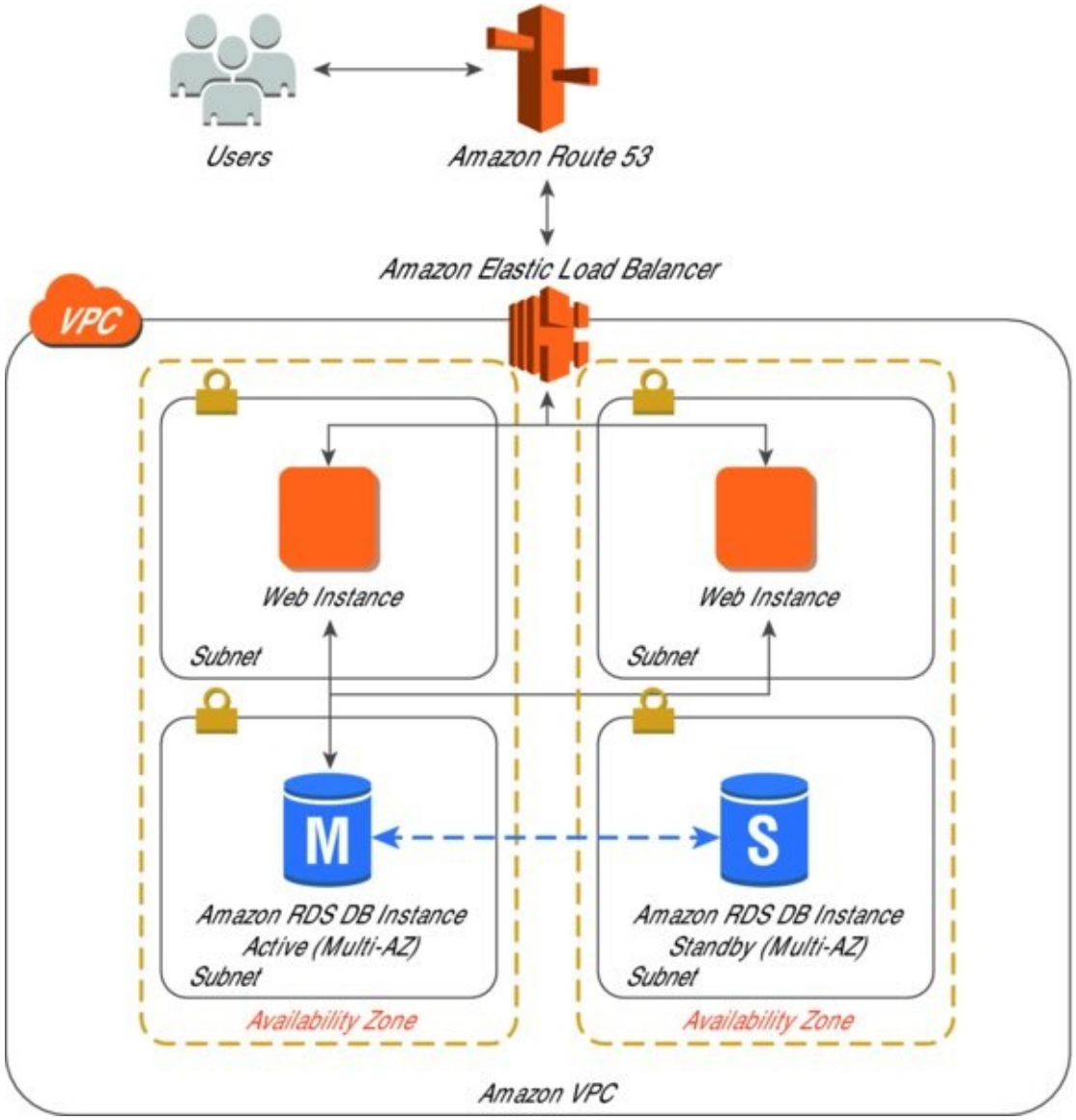

FIGURE 14.2 Updated web application architecture with redundancy

This *Multi-AZ* architecture helps to ensure that the application is isolated from failures in a single Availability Zone. In fact, many of the higher level services on AWS are inherently designed according to the Multi-AZ principle. For example, Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB ensure that data is redundantly stored across multiple facilities.

One rule of thumb to keep in mind when designing architectures in the cloud is to be a pessimist; that is, assume things will fail. In other words, always design, implement, and deploy for automated recovery from failure.

Implement Elasticity

Elasticity is the ability of a system to grow to handle increased load, whether gradually over time or in response to a sudden change in business needs. To achieve elasticity, it is important that the system be built on a scalable architecture. Such architectures can support growth in users, traffic, or data size with no drop in performance. These architectures should provide scale in a linear manner, where adding extra resources results in at least a proportional increase in ability to serve additional system load. The growth in resources should introduce economies of scale, and cost should follow the same dimension that generates business value out of that system. While cloud computing provides virtually unlimited on-demand capacity, system architectures need to be able to take advantage of those resources seamlessly. There are generally two ways to scale an IT architecture: vertically and horizontally.

Scaling Vertically

Vertical scaling takes place through an increase in the specifications of an individual resource (for example, upgrading a server with a larger hard drive, more memory, or a faster CPU). On Amazon Elastic Compute Cloud (Amazon EC2), this can easily be achieved by stopping an instance and resizing it to an instance type that has more RAM, CPU, I/O, or networking capabilities. Vertical scaling will eventually hit a limit, and it is not always a cost-efficient or highly available approach. Even so, it is very easy to implement and can be sufficient for many use cases, especially in the short term.

Scaling Horizontally

Horizontal scaling takes place through an increase in the number of resources (for example, adding more hard drives to a storage array or adding more servers to support an application). This is a great way to build Internet-scale applications that leverage the elasticity of cloud computing. Not all architectures are designed to distribute their workload to multiple resources, and it is important to understand system characteristics that can affect a system's ability to scale horizontally. One key characteristic is the impact of stateless and stateful architectures.

Stateless Applications

When users or services interact with an application, they will often perform a series of interactions that form a session. A *stateless application* needs no knowledge of the previous interactions and stores no session information. A stateless application can scale horizontally, because any request can be serviced by any of the available system compute resources. Because no session data needs to be shared between system resources, compute resources can be added as needed. When excess capacity is no longer required, any individual resource can be safely terminated. Those resources do not need to be aware of the presence of their peers; all that is required is a way to distribute the workload to them.

Let's assume that the web application we used in the previous section is a stateless application with unpredictable demand. In order for our web instances to meet the peaks and valleys associated with our demand profile, we need to scale elastically. A great way to

introduce elasticity and horizontal scaling is by leveraging Auto Scaling for web instances. An Auto Scaling group can automatically add Amazon EC2 instances to an application in response to heavy traffic and remove them when traffic slows. Figure 14.3 shows our web application architecture after the introduction of an Auto Scaling group.

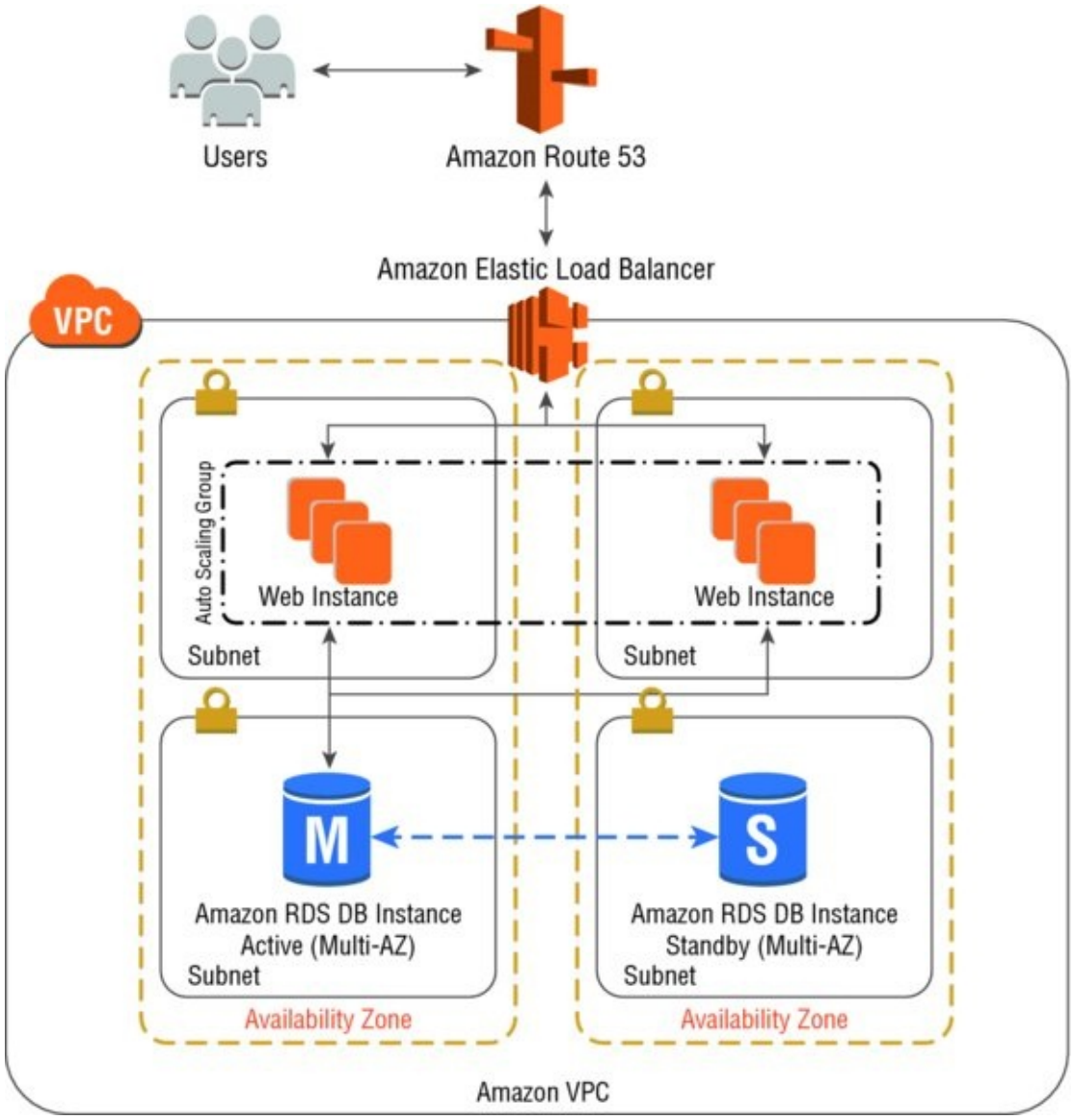

FIGURE 14.3 Updated web application architecture with auto scaling

Stateless Components

In practice, most applications need to maintain some kind of state information. For example, web applications need to track whether a user is signed in, or else they might present personalized content based on previous actions. You can still make a portion of these architectures stateless by not storing state information locally on a horizontally-scaling resource, as those resources can appear and disappear as the system scales up and down.

For example, web applications can use HTTP cookies to store information about a session at the client's browser (such as items in the shopping cart). The browser passes that information back to the server at each subsequent request so that the application does not need to store it. However, there are two drawbacks with this approach. First, the content of the HTTP cookies can be tampered with at the client side, so you should always treat them as untrusted data that needs to be validated. Second, HTTP cookies are transmitted with every request, which means that you should keep their size to a minimum to avoid unnecessary

latency.

Consider only storing a unique session identifier in a HTTP cookie and storing more detailed user session information server-side. Most programming platforms provide a native session management mechanism that works this way; however, these management mechanisms often store the session information locally by default. This would result in a stateful architecture. A common solution to this problem is to store user session information in a database. Amazon DynamoDB is a great choice due to its scalability, high availability, and durability characteristics. For many platforms, there are open source, drop-in replacement libraries that allow you to store native sessions in Amazon DynamoDB.

Stateful Components

Inevitably, there will be layers of your architecture that you won't turn into stateless components. First, by definition, databases are stateful. In addition, many legacy applications were designed to run on a single server by relying on local compute resources. Other use cases might require client devices to maintain a connection to a specific server for prolonged periods of time. For example, real-time multiplayer gaming must offer multiple players a consistent view of the game world with very low latency. This is much simpler to achieve in a non-distributed implementation where participants are connected to the same server.

Deployment Automation

Whether you are deploying a new environment for testing or increasing capacity of an existing system to cope with extra load, you will not want to set up new resources manually with their configuration and code. It is important that you make this an automated and repeatable process that avoids long lead times and is not prone to human error. Automating the deployment process and streamlining the configuration and build process is key to implementing elasticity. This will ensure that the system can scale without any human intervention.

Automate Your Infrastructure

One of the most important benefits of using a cloud environment is the ability to use the cloud's Application Program Interfaces (APIs) to automate your deployment process. It is recommended that you take the time to create an automated deployment process early on during the migration process and not wait until the end. Creating an automated and repeatable deployment process will help reduce errors and facilitate an efficient and scalable update process.

Bootstrap Your Instances

When you launch an AWS resource like an Amazon EC2 instance, you start with a default configuration. You can then execute automated bootstrapping actions as described in Chapter 3, "Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Elastic Block Store (Amazon EBS)." Let your instances ask a question at boot: "Who am I and what is my role?" Every instance should have a role to play in the environment (such as database server, application server, or slave server in the case of a web application). Roles may be applied during launch and can instruct the AMI on the steps to take after it has booted. On boot, an instance should grab the necessary resources (for example, code, scripts, or configuration) based on the role

and “attach” itself to a cluster to serve its function.

Benefits of bootstrapping your instances include:

- Recreate environments (for example, development, staging, production) with few clicks and minimal effort.
- Maintain more control over your abstract, cloud-based resources.
- Reduce human-induced deployment errors.
- Create a self-healing and self-discoverable environment that is more resilient to hardware failure.

Designing intelligent elastic cloud architectures, where infrastructure runs only when you need it, is an art. As a Solutions Architect, elasticity should be one of the fundamental design requirements when defining your architectures. Here are some questions to keep in mind when designing cloud architectures:

- What components or layers in my application architecture can become elastic?
- What will it take to make that component elastic?
- What will be the impact of implementing elasticity to my overall system architecture?

Leverage Different Storage Options

AWS offers a broad range of storage choices for backup, archiving, and disaster recovery, as well as block, file, and object storage to suit a plethora of use cases. For example, services like Amazon Elastic Block Storage (Amazon EBS), Amazon S3, Amazon RDS, and Amazon CloudFront provide a wide range of choices to meet different storage needs. It is important from a cost, performance, and functional aspect to leverage different storage options available in AWS for different types of datasets.

One Size Does Not Fit All

Your workload and use case should dictate what storage option to leverage in AWS. No one storage option is suitable for all situations. Table 14.1 provides a list of some storage scenarios and which AWS storage option you should consider to meet the identified need. This table is not meant to be an all-encompassing capture of scenarios, but an example guide.

TABLE 14.1 Storage Scenarios and AWS Storage Options

Sample Scenario	Storage Option
Your web application needs large-scale storage capacity and performance.	
-or-	Amazon S3
You need cloud storage with high data durability to support backup and active archives for disaster recovery.	
You require cloud storage for data archiving and long-term backup.	Amazon Glacier
You require a content delivery network to deliver entire websites, including dynamic, static, streaming, and interactive content using a global network of edge locations.	Amazon CloudFront
You require a fast and flexible NoSQL database with a flexible data model and reliable performance.	Amazon DynamoDB
You need reliable block storage to run mission-critical applications such as Oracle, SAP, Microsoft Exchange, and Microsoft SharePoint.	Amazon EBS
You need a highly available, scalable, and secure MySQL database without the time-consuming administrative tasks.	Amazon RDS
You need a fast, powerful, fully-managed, petabyte-scale data warehouse to support business analytics of your e-commerce application.	Amazon Redshift
You need a Redis cluster to store session information for your web application.	Amazon ElastiCache
You need a common file system for your application that is shared between more than one Amazon EC2 instance.	Amazon Elastic File System (Amazon EFS)

Let's return to our sample web application architecture and show how different storage options can be leveraged to optimize cost and architecture. We can start by moving any static assets from our web instances to Amazon S3, and then serve those objects via Amazon

CloudFront. These static assets would include all of the images, videos, CSS, JavaScript, and any other heavy static content that is currently delivered via the web instances. By serving these files via an Amazon S3 origin with global caching and distribution via Amazon CloudFront, the load will be reduced on the web instances and allow the web tier footprint to be reduced. Figure 14.4 shows the updated architecture for our sample web application.

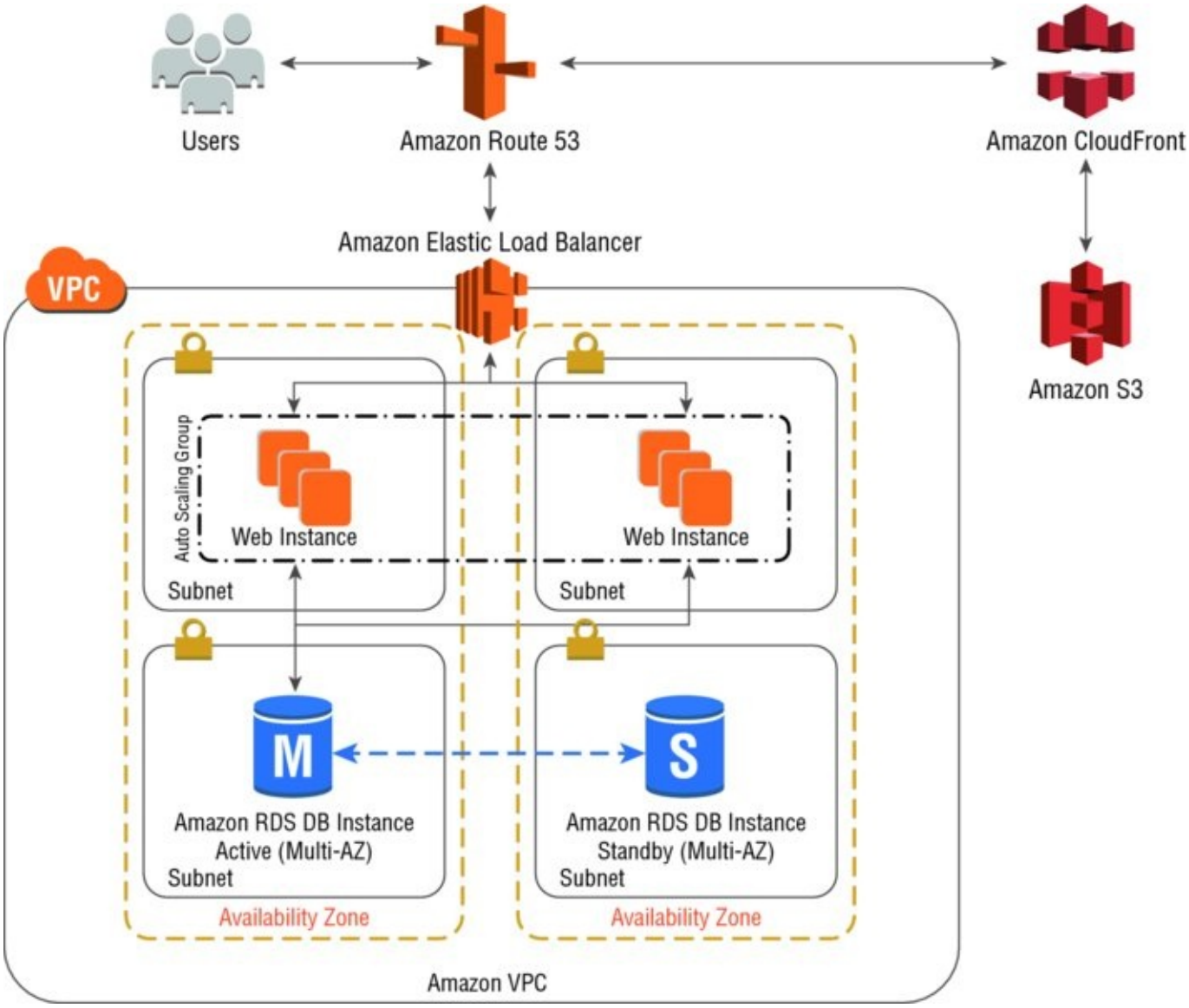

FIGURE 14.4 Updated web application architecture with Amazon S3 and Amazon CloudFront

To further optimize our storage options, the session information for our sample web application can be moved to Amazon DynamoDB or even to Amazon ElastiCache. For our scenario, we will use Amazon DynamoDB to store the session information because the AWS Software Development Kits (SDK) provide connectors for many popular web development frameworks that make storing session information in Amazon DynamoDB easy. By removing session state from our web tier, the web instances do not lose session information when horizontal scaling from Auto Scaling happens. Additionally, we will leverage Amazon ElastiCache to store common database query results, thereby taking the load off of our database tier. Figure 14.5 shows the addition of Amazon ElastiCache and Amazon DynamoDB to our web application architecture.

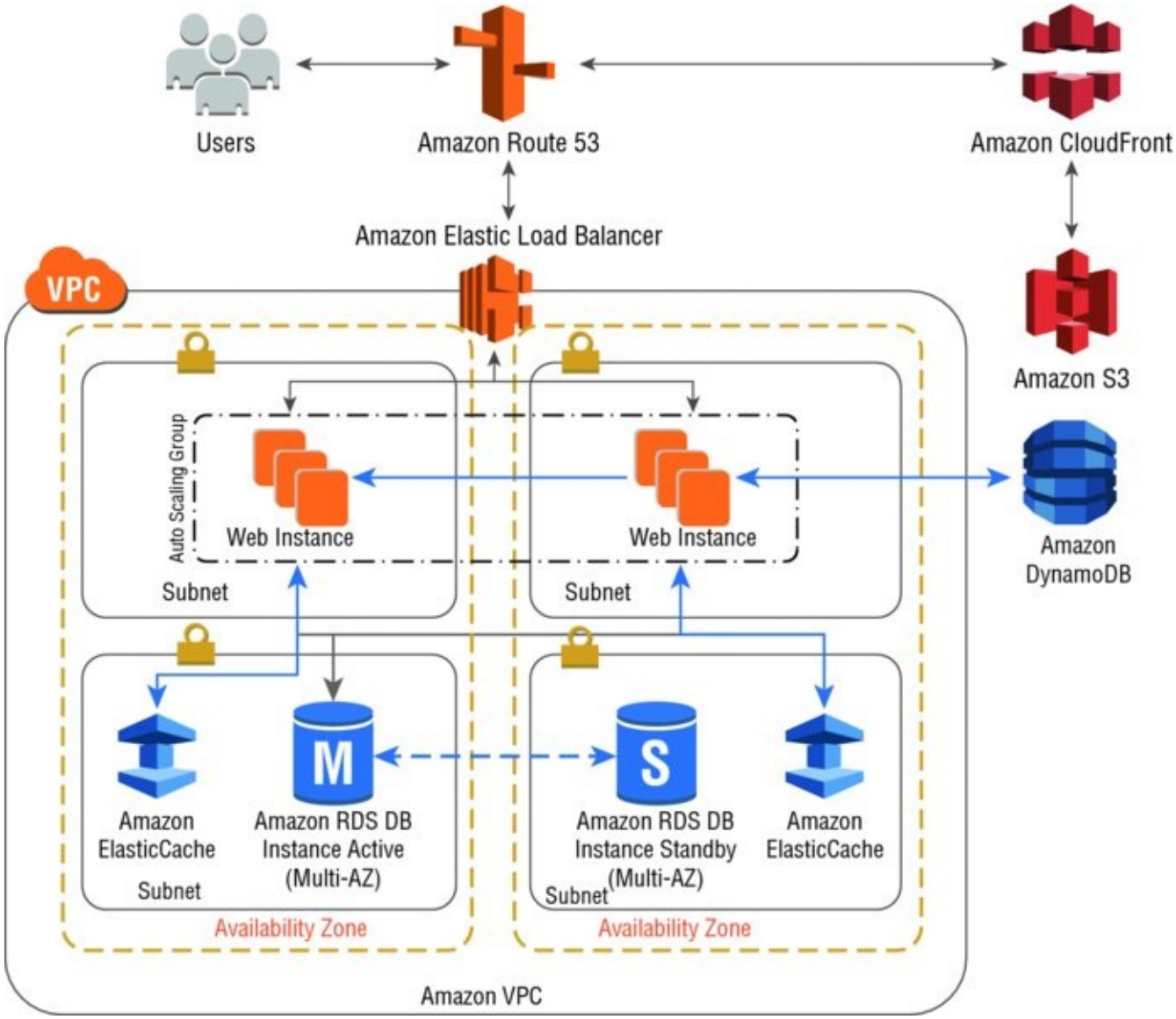

FIGURE 14.5 Updated web application architecture with Amazon ElastiCache and Amazon DynamoDB

As a Solutions Architect, you will ultimately come to a point where you need to decide and define what your storage requirements are for the data that you need to store on AWS. There are a variety of options to choose from depending on your needs, each with different attributes ranging from database storage, block storage, highly available object-based storage, and even cold archival storage. Ultimately, your workload requirements will dictate which storage option makes sense for your use case.

Build Security in Every Layer

With traditional IT, infrastructure security auditing would often be a periodic and manual process. The AWS Cloud instead provides governance capabilities that enable continuous monitoring of configuration changes to your IT resources. Because AWS assets are programmable resources, your security policy can be formalized and embedded with the design of your infrastructure. With the ability to spin up temporary environments, security testing can now become part of your continuous delivery pipeline. Solutions Architects can leverage a plethora of native AWS security and *encryption* features that can help achieve higher levels of data protection and compliance at every layer of cloud architectures.

> **Best Practice**
>
> Inventory your data, prioritize it by value, and apply the appropriate level of encryption for the data in transit and at rest.

Most of the security tools and techniques with which you might already be familiar in a traditional IT infrastructure can be used in the cloud. At the same time, AWS allows you to improve your security in a variety of ways. AWS is a platform that allows you to formalize the design of security controls in the platform itself. It simplifies system use for administrators and those running IT and makes your environment much easier to audit in a continuous manner.

Use AWS Features for Defense in Depth

AWS provides a wealth of features that help Solutions Architects build *defense in depth*. Starting at the network level, you can build an Amazon Virtual Private Cloud (Amazon VPC) topology that isolates parts of the infrastructure through the use of subnets, security groups, and routing controls. Services like AWS Web Application Firewall (AWS WAF) can help protect your web applications from SQL injection and other vulnerabilities in your application code. For access control, you can use AWS Identity and Access Management (IAM) to define a granular set of policies and assign them to users, groups, and AWS resources. Finally, the AWS platform offers a breadth of options for protecting data with encryption, whether the data is in transit or at rest.

>  Understanding the security features offered by AWS is important for the exam, and it is covered in detail in Chapter 12, “Security on AWS.”

Offload Security Responsibility to AWS

AWS operates under a shared responsibility model, where AWS is responsible for the security of the underlying cloud infrastructure, and you are responsible for securing the workloads you deploy on AWS. This way, you can reduce the scope of your responsibility and focus on your core competencies through the use of AWS managed services. For example, when you

use managed services such as Amazon RDS, Amazon ElastiCache, Amazon CloudSearch, and others, security patches become the responsibility of AWS. This not only reduces operational overhead for your team, but it could also reduce your exposure to vulnerabilities.

Reduce Privileged Access

Another common source of security risk is the use of service accounts. In a traditional environment, service accounts would often be assigned long-term credentials stored in a configuration file. On AWS, you can instead use IAM roles to grant permissions to applications running on Amazon EC2 instances through the use of temporary security tokens. Those credentials are automatically distributed and rotated. For mobile applications, the use of Amazon Cognito allows client devices to get controlled access to AWS resources via temporary tokens. For AWS Management Console users, you can similarly provide federated access through temporary tokens instead of creating IAM users in your AWS account. In that way, an employee who leaves your organization and is removed from your organization's identity directory will also lose access to your AWS account.

Best Practice

Follow the standard security practice of granting least privilege—that is, granting only the permissions required to perform a task—to IAM users, groups, roles, and policies.

Security as Code

Traditional security frameworks, regulations, and organizational policies define security requirements related to things such as firewall rules, network access controls, internal/external subnets, and operating system hardening. You can implement these in an AWS environment as well, but you now have the opportunity to capture them all in a script that defines a "Golden Environment." This means that you can create an AWS CloudFormation script that captures and reliably deploys your security policies. Security best practices can now be reused among multiple projects and become part of your continuous integration pipeline. You can perform security testing as part of your release cycle and automatically discover application gaps and drift from your security policies.

Additionally, for greater control and security, AWS CloudFormation templates can be imported as "products" into AWS Service Catalog. This enables centralized management of resources to support consistent governance, security, and compliance requirements while enabling users to deploy quickly only the approved IT services they need. You apply IAM permissions to control who can view and modify your products, and you define constraints to restrict the ways that specific AWS resources can be deployed for a product.

Real-Time Auditing

Testing and auditing your environment is key to moving fast while staying safe. Traditional approaches that involve periodic (and often manual or sample-based) checks are not sufficient, especially in agile environments where change is constant. On AWS, you can implement continuous monitoring and automation of controls to minimize exposure to security risks. Services like AWS Config Rules, Amazon Inspector, and AWS Trusted Advisor

continually monitor for compliance or vulnerabilities giving you a clear overview of which IT resources are or are not in compliance. With AWS Config Rules, you will also know if some component was out of compliance even for a brief period of time, making both point-in-time and period-in-time audits very effective. You can implement extensive logging for your applications using Amazon CloudWatch Logs and for the actual AWS API calls by enabling AWS CloudTrail. AWS CloudTrail is a web service that records API calls to supported AWS Cloud services in your AWS account and creates a log file. AWS CloudTrail logs are stored in an immutable manner to an Amazon S3 bucket of your choice. These logs can then be automatically processed either to notify or even take action on your behalf, protecting your organization from non-compliance. You can use AWS Lambda, Amazon Elastic MapReduce (Amazon EMR), Amazon Elasticsearch Service, or third-party tools from the AWS Marketplace to scan logs to detect things like unused permissions, overuse of privileged accounts, usage of keys, anomalous logins, policy violations, and system abuse.

While AWS provides an excellent service management layer around infrastructure or platform services, organizations are still responsible for protecting the confidentiality, integrity, and availability of their data in the cloud. AWS provides a range of security services and architectural concepts that organizations can use to manage security of their assets and data in the cloud.

Think Parallel

The cloud makes *parallelization* effortless. Whether it is requesting data from the cloud, storing data to the cloud, or processing data in the cloud, as a Solutions Architect you need to internalize the concept of parallelization when designing architectures in the cloud. It is advisable not only to implement parallelization wherever possible, but also to automate it because the cloud allows you to create a repeatable process very easily.

When it comes to accessing (retrieving and storing) data, the cloud is designed to handle massively parallel operations. In order to achieve maximum performance and throughput, you should leverage request parallelization. Multi-threading your requests by using multiple concurrent threads will store or fetch the data faster than requesting it sequentially. Hence, a general best practice for developing cloud applications is to design the processes for leveraging multi-threading.

When it comes to processing or executing requests in the cloud, it becomes even more important to leverage parallelization. A general best practice, in the case of a web application, is to distribute the incoming requests across multiple asynchronous web servers using a load balancer. In the case of a batch processing application, you can leverage a master node with multiple slave worker nodes that processes tasks in parallel (as in distributed processing frameworks like Hadoop).

The beauty of the cloud shines when you combine elasticity and parallelization. Your cloud application can bring up a cluster of compute instances that are provisioned within minutes with just a few API calls, perform a job by executing tasks in parallel, store the results, and then terminate all of the instances.

Loose Coupling Sets You Free

As application complexity increases, a desirable characteristic of an IT system is that it can be broken in to smaller, *loosely coupled* components. This means that IT systems should be designed in a way that reduces interdependencies, so that a change or a failure in one component does not cascade to other components.

Best Practice

Design system architectures with independent components that are "black boxes." The more loosely system components are coupled, the larger they scale.

A way to reduce interdependencies in a system is to allow the various components to interact with each other only through specific, technology-agnostic interfaces (such as RESTful APIs). In this way, the technical implementation details are hidden so that teams can modify the underlying implementation without affecting other components. As long as those interfaces maintain backward compatibility, the different components that an overall system is comprised of remain decoupled.

Amazon API Gateway provides a way to expose well-defined interfaces. Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. It handles all of the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management.

Asynchronous integration is a common pattern for implementing loose coupling between services. This model is suitable for any interaction that does not need an immediate response and where an acknowledgement that a request has been registered will suffice. It involves one component that generates events and another that consumes them. The two components do not integrate through direct point-to-point interaction, but usually through an intermediate durable storage layer, such as an Amazon Simple Queue Service (Amazon SQS) queue or a streaming data platform like Amazon Kinesis. Figure 14.6 shows the logical flow for tight and loosely coupled architectures.

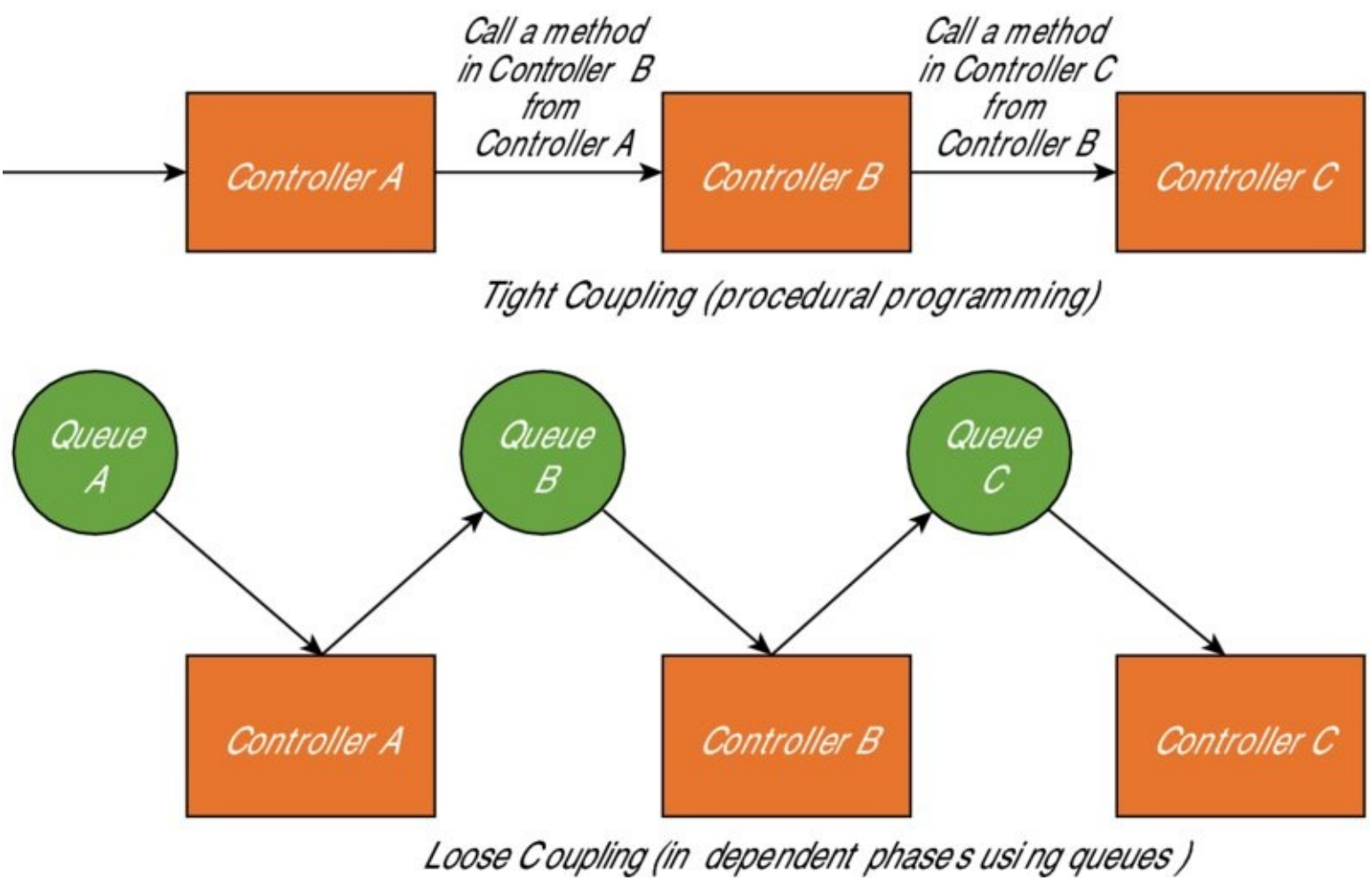

FIGURE 14.6 Tight and loose coupling

Leveraging asynchronous integration decouples the two components and introduces additional resiliency. For example, if a process that is reading messages from the queue fails, messages can still be added to the queue to be processed when the system recovers. It also allows you to protect a less scalable back-end service from front-end spikes and find the right tradeoff between cost and processing lag. For example, you can decide that you don't need to scale your database to accommodate for an occasional peak of write queries if you eventually process those queries asynchronously with some delay. Finally, by moving slow operations off of interactive request paths, you can also improve the end-user experience.

Design Scenario

Sample Loosely Coupled Architecture

A company provides transcoding services for amateur producers to format their short films to a variety of video formats. The service provides end users with an easy-to-use website to submit videos for transcoding. The videos are stored in Amazon S3, and a message ("the request message") is placed in an Amazon SQS queue ("the incoming queue") with a pointer to the video and to the target video format in the message. The transcoding engine, running on a set of Amazon EC2 instances, reads the request message from the incoming queue, retrieves the video from Amazon S3 using the pointer, and transcodes the video into the target format. The converted video is put back into Amazon S3 and another message ("the response message") is placed in another Amazon SQS queue ("the outgoing queue") with a pointer to the converted video. At the same time, metadata about the video (such as format, date created, and length) can be indexed into Amazon DynamoDB for easy querying. During this whole workflow, a dedicated Amazon EC2 instance can constantly monitor the incoming queue and, based on the number of messages in the incoming queue, can dynamically adjust the number of transcoding Amazon EC2 instances to meet customers' response time requirements.

Applications that are deployed as a set of smaller services will depend on the ability of those services to interact with each other. Because each of those services could be running across multiple compute resources, there needs to be a way for each service to be addressed. For example, in a traditional infrastructure, if your front-end web service needed to connect with your back-end web service, you could hardcode the IP address of the compute resource where this service was running. Although this approach can still work on cloud computing, if those services are meant to be loosely coupled, they should be able to be consumed without prior knowledge of their network topology details. Apart from hiding complexity, this also allows infrastructure details to change at any time. In order to achieve this agility, you will need some way of implementing service discovery. Service discovery manages how processes and services in an environment can find and talk to one another. It involves a directory of services, registering services in that directory, and then being able to look up and connect to services in that directory.

Loose coupling is a crucial element if you want to take advantage of the elasticity of cloud computing, where new resources can be launched or terminated at any point in time. By architecting system components without tight dependencies on each other, applications are positioned to take full advantage of the cloud's scale.

Don’t Fear Constraints

When organizations decide to move applications to the cloud and try to map their existing system specifications to those available in the cloud, they notice that the cloud might not have the exact specification of the resource that they have on premises. For example, observations may include “Cloud does not provide X amount of RAM in a server” or “My database needs to have more IOPS than what I can get in a single instance.”

You should understand that the cloud provides abstract resources that become powerful when you combine them with the on-demand provisioning model. You should not be afraid and constrained when using cloud resources because even if you might not get an exact replica of your on-premises hardware in the cloud environment, you have the ability to get more of those resources in the cloud to compensate.

When you push up against a constraint, think about what it’s telling you about a possible underlying architectural issue. For example, if AWS does not have an Amazon RDS instance type with enough RAM, consider whether you have inadvertently trapped yourself in a scale-up paradigm. Consider changing the underlying technology and using a scalable distributed cache like Amazon ElastiCache or sharding your data across multiple servers. If it is a read-heavy application, you can distribute the read load across a fleet of synchronized slaves.

Organizations are challenged with developing, managing, and operating applications at scale with a wide variety of underlying technology components. With traditional IT infrastructure, companies would have to build and operate all of those components. While these components may not map directly into a cloud environment, AWS offers a broad set of complementary services that help organizations overcome these constraints and to support agility and lower IT costs.

On AWS, there is a set of managed services that provides building blocks for developers to leverage for powering their applications. These managed services include databases, machine learning, analytics, queuing, search, email, notifications, and more. For example, with Amazon SQS, you can offload the administrative burden of operating and scaling a highly available messaging cluster while paying a low price for only what you use. The same applies to Amazon S3, where you can store as much data as required and access it when needed without having to think about capacity, hard disk configurations, replication, and other hardware-based considerations.

There are many other examples of managed services on AWS, such as Amazon CloudFront for content delivery, Elastic Load Balancing for load balancing, Amazon DynamoDB for NoSQL databases, Amazon CloudSearch for search workloads, Amazon Elastic Transcoder for video encoding, Amazon Simple Email Service (Amazon SES) for sending and receiving emails, and more.

Architectures that do not leverage the breadth of AWS Cloud services (for example, they use only Amazon EC2) might be self-constraining the ability to make the most of cloud computing. This oversight often leads to missing key opportunities to increase developer productivity and operational efficiency. When organizations combine on-demand provisioning, managed services, and the inherent flexibility of the cloud, they realize that apparent constraints can actually be broken down in ways that will actually improve the

scalability and overall performance of their systems.

Summary

Typically, production systems come with defined or implicit requirements in terms of uptime. A system is highly available when it can withstand the failure of an individual or multiple components. If you design architectures around the assumption that any component will eventually fail, systems won't fail when an individual component does.

Traditional infrastructure generally necessitates predicting the amount of computing resources your application will use over a period of several years. If you underestimate, your applications will not have the horsepower to handle unexpected traffic, potentially resulting in customer dissatisfaction. If you overestimate, you're wasting money with superfluous resources. The on-demand and elastic nature of the cloud enables the infrastructure to be closely aligned with the actual demand, thereby increasing overall utilization and reducing cost. While cloud computing provides virtually unlimited on-demand capacity, system architectures need to be able to take advantage of those resources seamlessly. There are generally two ways to scale an IT architecture: vertically and horizontally.

The AWS Cloud provides governance capabilities that enable continuous monitoring of configuration changes to your IT resources. Because AWS assets are programmable resources, your security policy can be formalized and embedded with the design of your infrastructure. With the ability to spin up temporary environments, security testing can now become part of your continuous delivery pipeline. Solutions Architects can leverage a plethora of native AWS security and encryption features that can help achieve higher levels of data protection and compliance at every layer of cloud architectures.

Because AWS makes parallelization effortless, Solutions Architects need to internalize the concept of parallelization when designing architectures in the cloud. It is advisable not only to implement parallelization wherever possible, but also to automate it because the cloud allows you to create a repeatable process very easily.

As application complexity increases, a desirable characteristic of an IT system is that it can be broken into smaller, loosely coupled components. Solutions Architects should design systems in a way that reduces interdependencies, so that a change or a failure in one component does not cascade to other components.

When organizations try to map their existing system specifications to those available in the cloud, they notice that the cloud might not have the exact specification of the resource that they have on-premises. Organizations should not be afraid and feel constrained when using cloud resources. Even if you might not get an exact replica of your hardware in the cloud environment, you have the ability to get more of those resources in the cloud to compensate.

By focusing on concepts and best practices—like designing for failure, decoupling the application components, understanding and implementing elasticity, combining it with parallelization, and integrating security in every aspect of the application architecture—Solutions Architects can understand the design considerations necessary for building highly scalable cloud applications.

As each use case is unique, Solutions Architects need to remain diligent in evaluating how best practices and patterns can be applied to each implementation. The topic of cloud computing architectures is broad and continuously evolving.

Exam Essentials

Understand highly available architectures. A system is highly available when it can withstand the failure of an individual or multiple components. If you design architectures around the assumption that any component will eventually fail, systems won't fail when an individual component does.

Understand redundancy. Redundancy can be implemented in either standby or active mode. When a resource fails in standby redundancy, functionality is recovered on a secondary resource using a process called failover. The failover will typically require some time before it is completed, and during that period the resource remains unavailable. In active redundancy, requests are distributed to multiple redundant compute resources, and when one of them fails, the rest can simply absorb a larger share of the workload. Compared to standby redundancy, active redundancy can achieve better utilization and affect a smaller population when there is a failure.

Understand elasticity. Elastic architectures can support growth in users, traffic, or data size with no drop in performance. It is important to build elastic systems on top of a scalable architecture. These architectures should scale in a linear manner, where adding extra resources results in at least a proportional increase in ability to serve additional system load. The growth in resources should introduce economies of scale, and cost should follow the same dimension that generates business value out of that system. There are generally two ways to scale an IT architecture: vertically and horizontally.

Understand vertical scaling. Scaling vertically takes place through an increase in the specifications of an individual resource (for example, upgrading a server with a larger hard drive or a faster CPU). This way of scaling can eventually hit a limit, and it is not always a cost efficient or highly available approach.

Understand horizontal scaling. Scaling horizontally takes place through an increase in the number of resources. This is a great way to build Internet-scale applications that leverage the elasticity of cloud computing. It is important to understand the impact of stateless and stateful architectures before implementing horizontal scaling.

Understand stateless applications. A stateless application needs no knowledge of the previous interactions and stores no session information. A stateless application can scale horizontally because any request can be serviced by any of the available system compute resources.

Understand loose coupling. As application complexity increases, a desirable characteristic of an IT system is that it can be broken into smaller, loosely coupled components. This means that IT systems should be designed as "black boxes" to reduce interdependencies so that a change or a failure in one component does not cascade to other components. The more loosely system components are coupled, the larger they scale.

Understand the different storage options in AWS. AWS offers a broad range of storage choices for backup, archiving, and disaster recovery, as well as block, file, and object storage to suit a plethora of use cases. It is important from a cost, performance, and functional aspect to leverage different storage options available in AWS for different types of datasets.

Exercises

In this section, you will implement a resilient application leveraging some of the best practices outlined in this chapter. You will build the architecture depicted in Figure 14.7 in the following series of exercises.

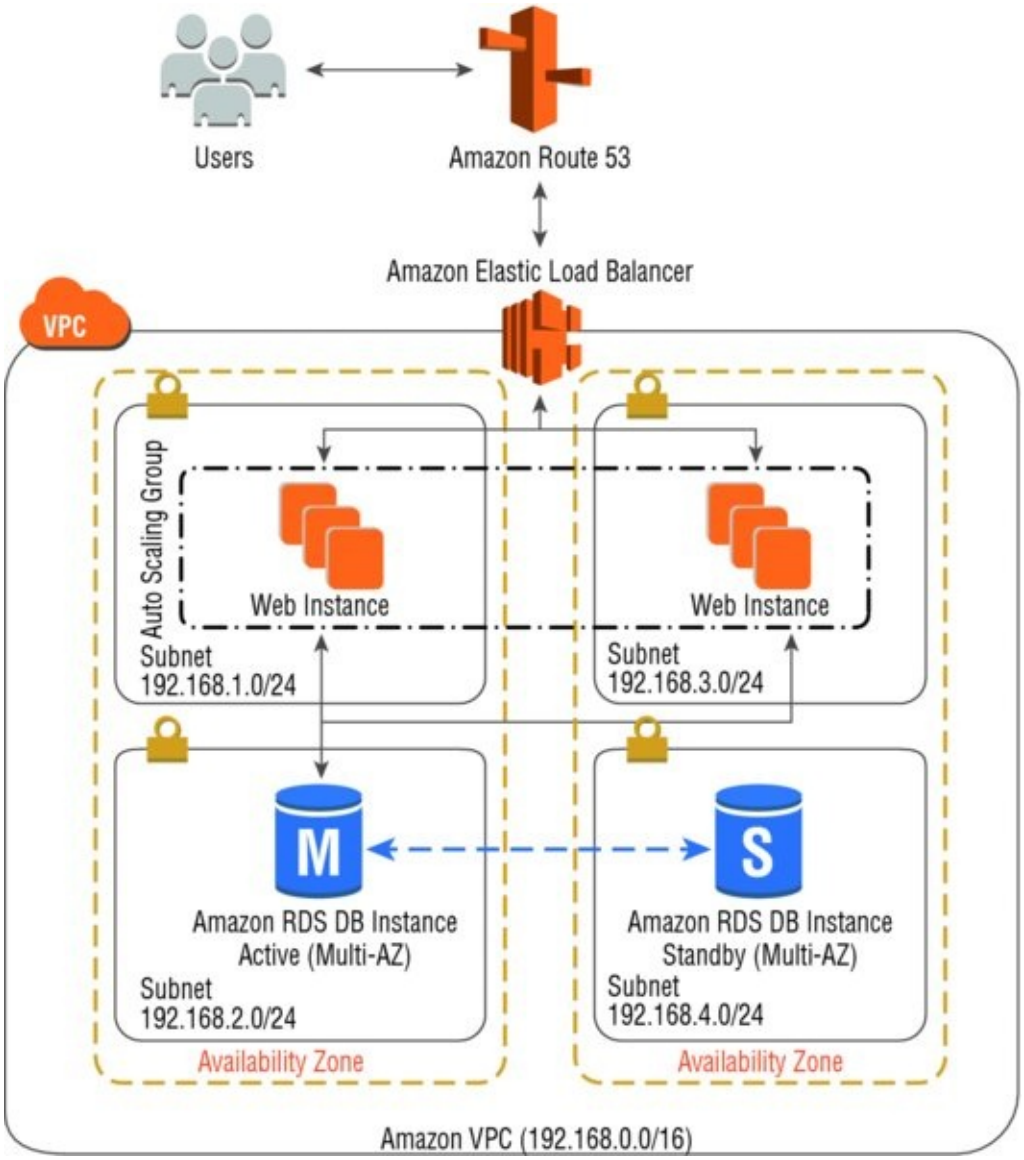

FIGURE 14.7 Sample web application for chapter exercises

For assistance in completing the following exercises, reference the following user guides:

- **Amazon VPC**—http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/GetStarted.html
- **Amazon EC2 (Linux)**—http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html
- **Amazon RDS (MySQL)**—http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.Crea

EXERCISE 14.1

Create a Custom Amazon VPC

1. Log in to the AWS Management Console.
2. Navigate to the Amazon VPC console.
3. Create an Amazon VPC with a Classless Inter-Domain Routing (CIDR) block equal to 192.168.0.0/16, a name tag of **`Ch 14–VPC`**, and default tenancy.

EXERCISE 14.2

Create an Internet Gateway for Your Custom Amazon VPC

1. Log in to the AWS Management Console.
2. Navigate to the Amazon VPC console.
3. Create an Internet gateway with a name tag of **`Ch 14-IGW.`**
4. Attach the **`Ch 14-IGW`** Internet gateway to the Amazon VPC from Exercise 14.1.

EXERCISE 14.3

Update the Main Route Table for Your Custom Amazon VPC

1. Log in to the AWS Management Console.
2. Navigate to Amazon VPC console.
3. Locate the main route table for the Amazon VPC from Exercise 14.1.
4. Update the route table name tag to a value of **`Ch 14–Main Route Table.`**
5. Update the route table routes by adding a destination of 0.0.0.0/0 with a target of the Internet gateway from Exercise 14.2.

EXERCISE 14.4

Create Public Subnets for Your Custom Amazon VPC

1. Log in to the AWS Management Console.
2. Navigate to the Amazon VPC console.
3. Create a subnet with a CIDR block equal to 192.168.1.0/24 and a name tag of **`Ch 14–Public Subnet 1`**. Create the subnet in the Amazon VPC from Exercise 14.1, and specify an Availability Zone for the subnet (for example, `US-East-1a`).
4. Create a subnet with a CIDR block equal to 192.168.3.0/24 and a name tag of **`Ch 14–Public Subnet 2`**. Create the subnet in the Amazon VPC from Exercise 14.1, and specify an Availability Zone for the subnet that is different from the one previously specified (for example, `US-East-1b`).

EXERCISE 14.5

Create a NAT Gateway for Your Custom Amazon VPC

1. Log in to the AWS Management Console.
2. Navigate to the Amazon VPC console.
3. Create a Network Address Translation (NAT) gateway in the Amazon VPC from Exercise 14.1 within the **`Ch 14–Public Subnet 1`** subnet from Exercise 14.4.

EXERCISE 14.6

Create a Private Route Table for Your Custom Amazon VPC

1. Log in to the AWS Management Console.
2. Navigate to the Amazon VPC console.
3. Create a route table for the Amazon VPC from Exercise 14.1 with a name tag of **`Ch 14 –Private Route Table`**.
4. Update the route table routes by adding a destination of 0.0.0.0/0 with a target of the NAT gateway from Exercise 14.5.

EXERCISE 14.7

Create Private Subnets for Your Custom Amazon VPC

1. Log in to the AWS Management Console.
2. Navigate to the Amazon VPC console.
3. Create a subnet with a CIDR block equal to 192.168.2.0/24 and a name tag of **`Ch 14–Private Subnet 1`**. Create the subnet in the Amazon VPC from Exercise 14.1, and specify the same Availability Zone for the subnet that was used in Exercise 14.4 for the **`Ch 14–Public Subnet 1`** (for example, `US-East-1a`).
4. Update the route table for the created subnet to the **`Ch 14–Private Route Table`** from Exercise 14.6.
5. Create a subnet with a CIDR block equal to 192.168.4.0/24 and a name tag of **`Ch 14–Private Subnet 2`**. Create the subnet in the Amazon VPC from Exercise 14.1, and specify the same Availability Zone for the subnet that was used in Exercise 14.4 for the **`Ch 14–Public Subnet 2`** (for example, `US-East-1b`).
6. Update the route table for the created subnet to the **`Ch 14–Private Route Table`** from Exercise 14.6.

EXERCISE 14.8

Create Security Groups for Each Application Tier

1. Log in to the AWS Management Console.
2. Navigate to the Amazon VPC console.
3. Create an Amazon VPC security group for the ELB with a name tag and group tab of **`Ch14-ELB-SG`** and a description of **`Load balancer security group for Ch 14 exercises`**. Create the security group in the Amazon VPC from Exercise 14.1 with an inbound rule of Type HTTP, a protocol of TCP, a port range of 80, and a source of 0.0.0.0/0.
4. Create an Amazon VPC security group for the web servers with a name tag and group tab of **`Ch14-WebServer-SG`** and a description of **`Web server security group for Ch 14 exercises`**. Create the security group in the Amazon VPC from Exercise 14.1 with an inbound rule of Type HTTP, a protocol of TCP, a port range of 80, and a source of the **`Ch14-ELB-SG`** security group. You may want to add another inbound rule of Type SSH, a protocol of TCP, a port range of 22, and a source of your IP address to provide secure access to manage the servers.
5. Create an Amazon VPC security group for the Amazon RDS MySQL database with a name tag and group tab of **`Ch14-DB-SG`** and a description of **`Database security group for Ch 14 exercises`**. Create the security group in the Amazon VPC from Exercise 14.1 with an inbound rule of Type MYSQL/Aurora, a protocol of TCP, a port range of 3306, and a source of the **`Ch14-WebServer-SG`** security group.

EXERCISE 14.9

Create a MySQL Multi-AZ Amazon RDS Instance

1. Log in to the AWS Management Console.
2. Navigate to the Amazon RDS console.
3. Create a DB subnet group with a name of **`Ch14-SubnetGroup`** and a description of **`Subnet group for Ch 14 exercises`**. Create the DB subnet group in the Amazon VPC from Exercise 14.1 with the private subnets from Exercise 14.7.
4. Launch a MySQL Amazon RDS instance with the following characteristics:

   DB Instance Class: db.t2.small

   Multi-AZ Deployment: yes

   Allocated Storage: no less than 5GB

   DB Instance Identifier: ch14db

   Master User Name: your choice

   Master Password: your choice

   VPC: the Amazon VPC from Exercise 14.1

   DB Security Group: Ch14-SubnetGroup

   Publicly Accessible: No

   VPC Security Group: Ch14-DB-SG

   Database Name: appdb

   Database Port: 3306

EXERCISE 14.10

Create an Elastic Load Balancer (ELB)

1. Log in to the AWS Management Console.
2. Navigate to the Amazon EC2 console.
3. Create an ELB with a load balancer name of **`Ch14-WebServer-ELB`**. Create the ELB in the Amazon VPC from Exercise 14.1 with a listener configuration of the following:

   Load Balancer Protocol: HTTP

   Load Balancer Port: 80

   Instance Protocol: HTTP

   Instance Port: 80
4. Add the public subnets created in Exercise 14.4.
5. Assign the existing security group of **`Ch14-ELB-SG`** created in Exercise 14.8.
6. Configure the health check with a ping protocol of HTTP, a ping port of 80, and a ping path of `/index.html`.
7. Add a tag with a key of **`Name`** and value of **`Ch14-WebServer-ELB`**.
8. Update the ELB port configuration to enable load-balancer generated cookie stickiness with an expiration period of 30 seconds.

EXERCISE 14.11

Create a Web Server Auto Scaling Group

1. Log in to the AWS Management Console.
2. Navigate to the Amazon EC2 console.
3. Create a launch configuration for the web server Auto Scaling group with the following characteristics:

   AMI: latest Amazon Linux AMI

   Instance Type: t2.small

   Name: Ch14-WebServer-LC

   User data:

   ```
   #!/bin/bash
   yum update –y
   yum install -y php
   yum install -y php-mysql
   yum install -y mysql
   yum install -y httpd
   echo "<html><body><h1>powered by AWS</h1></body></html>" >
   /var/www/html/index.html
   service httpd start
   ```

   Security Group: Ch14-WebServer-SG

   Key Pair: existing or new key pair for your account

4. Create an Auto Scaling group for the web servers from the launch configuration **`Ch14-WebServer-LC`** with a group name of **`Ch14-WebServer-AG`**. Create the Auto Scaling group in the Amazon VPC from Exercise 14.1 with the public subnets created in Exercise 14.4 and a group size of 2.
5. Associate the load balancer **`Ch14-WebServer-ELB`** created in Exercise 14.10 to the Auto Scaling group.
6. Add a name tag with a key of **`Name`** and value of **`Ch14-WebServer-AG`** to the Auto Scaling group.

You will need your own domain name to complete this section, and you should be aware that Amazon Route 53 is not eligible for AWS Free Tier. Hosting a zone on Amazon Route 53 will cost approximately $0.50 per month per hosted zone, and additional charges will be levied depending on what routing policy you choose. For more information on Amazon Route 53 pricing, refer to http://aws.amazon.com/route53/pricing/.

EXERCISE 14.12

Create a Route 53 Hosted Zone

1. Log in to the AWS Management Console.
2. Navigate to the Amazon Route 53 console and create a hosted zone.
3. Enter your domain name and create your new zone file.
4. In the new zone file, you will see the Start of Authority (SOA) record and name servers. You will need to log in to your domain registrar's website and update the name servers with your AWS name servers.

If the registrar has a method to change the Time To Live (TTL) settings for their name servers, it is recommended that you reset the settings to 900 seconds. This limits the time during which client requests will try to resolve domain names using obsolete name servers. You will need to wait for the duration of the previous TTL for resolvers and clients to stop caching the DNS records with their previous values.

5. After you update your name servers with your domain registrars, Amazon Route 53 will be configured to serve DNS requests for your domain.

EXERCISE 14.13

Create an Alias A Record

1. Log in to the AWS Management Console.
2. Navigate to the Amazon Route 53 console.
3. Select your Route 53 hosted zone created in Exercise 14.12. Create a record set with a name of `www` and a type of **`A–IPv4 Address`**.
4. Create an alias with an alias target of the ELB **`Ch14-WebServer-ELB`** created in Exercise 14.10 and leave your routing policy as simple.

EXERCISE 14.14

Test Your Configuration

1. Log in to the AWS Management Console.
2. Navigate to the Amazon EC2 console.
3. Verify that the ELB created in Exercise 14.11 has **`2 of 2 instances in service`**.
4. In a web browser, navigate to the web farm (`www.example.com`) using the Hosted Zone A record created in Exercise 14.13. You should see the **`powered by AWS`** on the web page.

Review Questions

1. When designing a loosely coupled system, which AWS services provide an intermediate durable storage layer between components? (Choose 2 answers)
   A. Amazon CloudFront
   B. Amazon Kinesis
   C. Amazon Route 53
   D. AWS CloudFormation
   E. Amazon Simple Queue Service (Amazon SQS)
2. Which of the following options will help increase the availability of a web server farm? (Choose 2 answers)
   A. Use Amazon CloudFront to deliver content to the end users with low latency and high data transfer speeds.
   B. Launch the web server instances across multiple Availability Zones.
   C. Leverage Auto Scaling to recover from failed instances.
   D. Deploy the instances in an Amazon Virtual Private Cloud (Amazon VPC).
   E. Add more CPU and RAM to each instance.
3. Which of the following AWS Cloud services are designed according to the Multi-AZ principle? (Choose 2 answers)
   A. Amazon DynamoDB
   B. Amazon ElastiCache
   C. Elastic Load Balancing
   D. Amazon Virtual Private Cloud (Amazon VPC)
   E. Amazon Simple Storage Service (Amazon S3)
4. Your e-commerce site was designed to be stateless and currently runs on a fleet of Amazon Elastic Compute Cloud (Amazon EC2) instances. In an effort to control cost and increase availability, you have a requirement to scale the fleet based on CPU and network utilization to match the demand curve for your site. What services do you need to meet this requirement? (Choose 2 answers)
   A. Amazon CloudWatch
   B. Amazon DynamoDB
   C. Elastic Load Balancing
   D. Auto Scaling
   E. Amazon Simple Storage Service (Amazon S3)
5. Your compliance department has mandated a new requirement that all data on Amazon

Elastic Block Storage (Amazon EBS) volumes must be encrypted. Which of the following steps would you follow for your existing Amazon EBS volumes to comply with the new requirement? (Choose 3 answers)

A. Move the existing Amazon EBS volume into an Amazon Virtual Private Cloud (Amazon VPC).

B. Create a new Amazon EBS volume with encryption enabled.

C. Modify the existing Amazon EBS volume properties to enable encryption.

D. Attach an Amazon EBS volume with encryption enabled to the instance that hosts the data, then migrate the data to the encryption-enabled Amazon EBS volume.

E. Copy the data from the unencrypted Amazon EBS volume to the Amazon EBS volume with encryption enabled.

6. When building a Distributed Denial of Service (DDoS)-resilient architecture, how does Amazon Virtual Private Cloud (Amazon VPC) help minimize the attack surface area? (Choose 3 answers)

A. Reduces the number of necessary Internet entry points

B. Combines end user traffic with management traffic

C. Obfuscates necessary Internet entry points to the level that untrusted end users cannot access them

D. Adds non-critical Internet entry points to the architecture

E. Scales the network to absorb DDoS attacks

7. Your e-commerce application provides daily and *ad hoc* reporting to various business units on customer purchases. This is resulting in an extremely high level of read traffic to your MySQL Amazon Relational Database Service (Amazon RDS) instance. What can you do to scale up read traffic without impacting your database's performance?

A. Increase the allocated storage for the Amazon RDS instance.

B. Modify the Amazon RDS instance to be a Multi-AZ deployment.

C. Create a read replica for an Amazon RDS instance.

D. Change the Amazon RDS instance DB engine version.

8. Your website is hosted on a fleet of web servers that are load balanced across multiple Availability Zones using an Elastic Load Balancer (ELB). What type of record set in Amazon Route 53 can be used to point `myawesomeapp.com` to your website?

A. Type A Alias resource record set

B. MX record set

C. TXT record set

D. CNAME record set

9. You need a secure way to distribute your AWS credentials to an application running on Amazon Elastic Compute Cloud (Amazon EC2) instances in order to access

supplementary AWS Cloud services. What approach provides your application access to use short-term credentials for signing requests while protecting those credentials from other users?

A. Add your credentials to the `UserData` parameter of each Amazon EC2 instance.

B. Use a configuration file to store your access and secret keys on the Amazon EC2 instances.

C. Specify your access and secret keys directly in your application.

D. Provision the Amazon EC2 instances with an instance profile that has the appropriate privileges.

10. You are running a suite of microservices on AWS Lambda that provide the business logic and access to data stored in Amazon DynamoDB for your task management system. You need to create well-defined RESTful Application Program Interfaces (APIs) for these microservices that will scale with traffic to support a new mobile application. What AWS Cloud service can you use to create the necessary RESTful APIs?

A. Amazon Kinesis

B. Amazon API Gateway

C. Amazon Cognito

D. Amazon Elastic Compute Cloud (Amazon EC2) Container Registry

11. Your WordPress website is hosted on a fleet of Amazon Elastic Compute Cloud (Amazon EC2) instances that leverage Auto Scaling to provide high availability. To ensure that the content of the WordPress site is sustained through scale up and scale down events, you need a common file system that is shared between more than one Amazon EC2 instance. Which AWS Cloud service can meet this requirement?

A. Amazon CloudFront

B. Amazon ElastiCache

C. Amazon Elastic File System (Amazon EFS)

D. Amazon Elastic Beanstalk

12. You are changing your application to move session state information off the individual Amazon Elastic Compute Cloud (Amazon EC2) instances to take advantage of the elasticity and cost benefits provided by Auto Scaling. Which of the following AWS Cloud services is best suited as an alternative for storing session state information?

A. Amazon DynamoDB

B. Amazon Redshift

C. Amazon Storage Gateway

D. Amazon Kinesis

13. A media sharing application is producing a very high volume of data in a very short period of time. Your back-end services are unable to manage the large volume of transactions. What option provides a way to manage the flow of transactions to your

back-end services?

A. Store the inbound transactions in an Amazon Relational Database Service (Amazon RDS) instance so that your back-end services can retrieve them as time permits.

B. Use an Amazon Simple Queue Service (Amazon SQS) queue to buffer the inbound transactions.

C. Use an Amazon Simple Notification Service (Amazon SNS) topic to buffer the inbound transactions.

D. Store the inbound transactions in an Amazon Elastic MapReduce (Amazon EMR) cluster so that your back-end services can retrieve them as time permits.

14. Which of the following are best practices for managing AWS Identity and Access Management (IAM) user access keys? (Choose 3 answers)

A. Embed access keys directly into application code.

B. Use different access keys for different applications.

C. Rotate access keys periodically.

D. Keep unused access keys for an indefinite period of time.

E. Configure Multi-Factor Authentication (MFA) for your most sensitive operations.

15. You need to implement a service to scan Application Program Interface (API) calls and related events' history to your AWS account. This service will detect things like unused permissions, overuse of privileged accounts, and anomalous logins. Which of the following AWS Cloud services can be leveraged to implement this service? (Choose 3 answers)

A. AWS CloudTrail

B. Amazon Simple Storage Service (Amazon S3)

C. Amazon Route 53

D. Auto Scaling

E. AWS Lambda

16. Government regulations require that your company maintain all correspondence for a period of seven years for compliance reasons. What is the best storage mechanism to keep this data secure in a cost-effective manner?

A. Amazon S3

B. Amazon Glacier

C. Amazon EBS

D. Amazon EFS

17. Your company provides media content via the Internet to customers through a paid subscription model. You leverage Amazon CloudFront to distribute content to your customers with low latency. What approach can you use to serve this private content securely to your paid subscribers?

A. Provide signed Amazon CloudFront URLs to authenticated users to access the paid content.

B. Use HTTPS requests to ensure that your objects are encrypted when Amazon CloudFront serves them to viewers.

C. Configure Amazon CloudFront to compress the media files automatically for paid subscribers.

D. Use the Amazon CloudFront geo restriction feature to restrict access to all of the paid subscription media at the country level.

18. Your company provides transcoding services for amateur producers to format their short films to a variety of video formats. Which service provides the best option for storing the videos?

A. Amazon Glacier

B. Amazon Simple Storage Service (Amazon S3)

C. Amazon Relational Database Service (Amazon RDS)

D. AWS Storage Gateway

19. A week before Cyber Monday last year, your corporate data center experienced a failed air conditioning unit that caused flooding into the server racks. The resulting outage cost your company significant revenue. Your CIO mandated a move to the cloud, but he is still concerned about catastrophic failures in a data center. What can you do to alleviate his concerns?

A. Distribute the architecture across multiple Availability Zones.

B. Use an Amazon Virtual Private Cloud (Amazon VPC) with subnets.

C. Launch the compute for the processing services in a placement group.

D. Purchase Reserved Instances for the processing services instances.

20. Your Amazon Virtual Private Cloud (Amazon VPC) includes multiple private subnets. The instances in these private subnets must access third-party payment Application Program Interfaces (APIs) over the Internet. Which option will provide highly available Internet access to the instances in the private subnets?

A. Create an AWS Storage Gateway in each Availability Zone and configure your routing to ensure that resources use the AWS Storage Gateway in the same Availability Zone.

B. Create a customer gateway in each Availability Zone and configure your routing to ensure that resources use the customer gateway in the same Availability Zone.

C. Create a Network Address Translation (NAT) gateway in each Availability Zone and configure your routing to ensure that resources use the NAT gateway in the same Availability Zone.

D. Create a NAT gateway in one Availability Zone and configure your routing to ensure that resources use that NAT gateway in all the Availability Zones.

Appendix A
Answers to Review Questions