AWS® Certified Developer Official Study Guide

Associate (DVA-C01) Exam

one of your buckets and allow the user to add, update, and delete objects. You can grant them access with a user policy.

Now we will discuss the differences between IAM policies and Amazon S3 bucket policies. Both are used for access control, and they are both written in JSON using the AWS access policy language. However, unlike Amazon S3 bucket policies, IAM policies specify what actions are allowed or denied on what AWS resources (such as, allow `ec2:TerminateInstance` on the Amazon EC2 instance with `instance_id=i8b3620ec`). You attach IAM policies to IAM users, groups, or roles, which are then subject to the permissions that you have defined. Instead of attaching policies to the users, groups, or roles, bucket policies are attached to a specific resource, such as an Amazon S3 bucket.

Managing Access with Access Control Lists

Access with access control lists (ACLs) are resource-based access policies that you can use to manage access to your buckets and objects, including granting basic read/write permissions to other accounts.

There are limits to managing permissions using ACLs. For example, you can grant permissions only to other accounts; you cannot grant permissions to users in your account. You cannot grant conditional permissions, nor can you explicitly deny permissions using ACLs.

ACLs are suitable only for specific scenarios (for example, if a bucket owner allows other accounts to upload objects), and permissions to these objects can be managed only using an object ACL by the account that owns the object.

TIP

You can only grant access to other accounts using ACLs—not users in your own account.

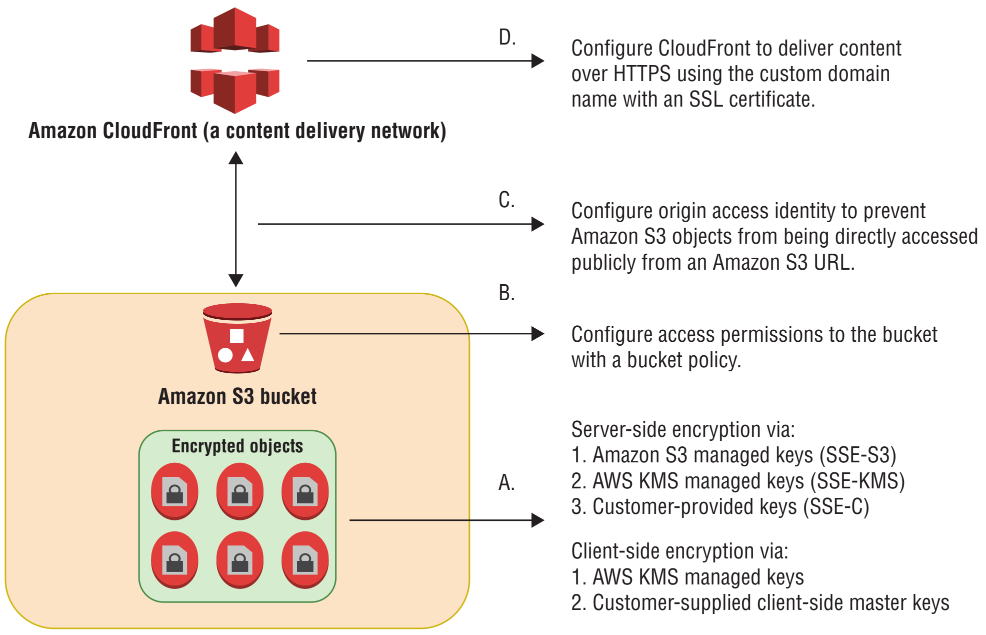
Defense in Depth—Amazon S3 Security

Amazon S3 provides comprehensive security and compliance capabilities that meet the most stringent regulatory requirements, and it gives you flexibility in the way that you manage data for cost optimization, access control, and compliance. With this flexibility, however, comes the responsibility of ensuring that your content is secure.

You can use an approach known as *defense in depth* in Amazon S3 to secure your data. This approach uses multiple layers of security to ensure redundancy if one of the multiple layers of security fails.

Figure 3.14 represents defense in depth visually. It contains several Amazon S3 objects (A) in a single Amazon S3 bucket (B). You can encrypt these objects on the server side or the client side, and you can also configure the bucket policy such that objects are accessible only through Amazon CloudFront, which you can accomplish through an origin access identity (C). You can then configure Amazon CloudFront to deliver content only over HTTPS in addition to using your own domain name (D).

FIGURE 3.14 Defense in depth on Amazon S3


Amazon CloudFront (a content delivery network)
D.
Configure CloudFront to deliver content over HTTPS using the custom domain name with an SSL certificate.
C.
Configure origin access identity to prevent Amazon S3 objects from being directly accessed publicly from an Amazon S3 URL.
B.
Configure access permissions to the bucket with a bucket policy.
Amazon S3 bucket
Encrypted objects
A.
Server-side encryption via:
1. Amazon S3 managed keys (SSE-S3)
2. AWS KMS managed keys (SSE-KMS)
3. Customer-provided keys (SSE-C)
Client-side encryption via:
1. AWS KMS managed keys
2. Customer-supplied client-side master keys

To meet defense in depth requirements on Amazon S3:

- Data must be encrypted at rest and during transit.
- Data must be accessible only by a limited set of public IP addresses.
- Data must not be publicly accessible directly from an Amazon S3 URL.
- A domain name is required to consume the content.

You can apply policies to Amazon S3 buckets so that only users with appropriate permissions are allowed to access the buckets. Anonymous users (with public-read/public-read-write permissions) and authenticated users without the appropriate permissions are prevented from accessing the buckets.

You can also secure access to objects in Amazon S3 buckets. The objects in Amazon S3 buckets can be encrypted at rest and during transit to provide end-to-end security from the source (in this case, Amazon S3) to your users.

Query String Authentication

You can provide authentication information using *query string parameters*. Using query parameters to authenticate requests is useful when expressing a request entirely in a URL. This method is also referred to as *presigning* a URL.

With presigned URLs, you can grant temporary access to your Amazon S3 resources. For example, you can embed a presigned URL on your website, or alternatively use it in a command line client (such as Curl), to download objects.

The following is an example presigned URL:

```
https://s3.amazonaws.com/examplebucket/test.txt
?X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=<your-access-key-id>/20130721/us-east-1/s3/aws4_request
&X-Amz-Date=20130721T201207Z
&X-Amz-Expires=86400
&X-Amz-SignedHeaders=host
&X-Amz-Signature=<signature-value>
```

In the example URL, note the following:

- The line feeds are added for readability.
- The `X-Amz-Credential` value in the URL shows the / character only for readability. In practice, it should be encoded as `%2F`.

Hosting a Static Website

If your website contains static content and optionally client-side scripts, then you can host your *static website* directly in Amazon S3 without the use of web-hosting servers.

To host a static website, you configure an Amazon S3 bucket for website hosting and upload your website content to the bucket. The website is then available at the AWS Region–specific website endpoint of the bucket in one of the following formats:

```
<bucket-name>.s3-website-<AWS-region>.amazonaws.com
<bucket-name>.s3-website.<AWS-region>.amazonaws.com
```

Instead of accessing the website by using an Amazon S3 website endpoint, use your own domain (for instance, `example.com`) to serve your content. The following steps allow you to configure your own domain:

1. Register your domain with the registrar of your choice. You can use Amazon Route 53 to register your domain name or any other third-party domain registrar.
2. Create your bucket in Amazon S3 and upload your static website content.
3. Point your domain to your Amazon S3 bucket using either of the following as your DNS provider:
   - Amazon Route 53
   - Your third-party domain name registrar

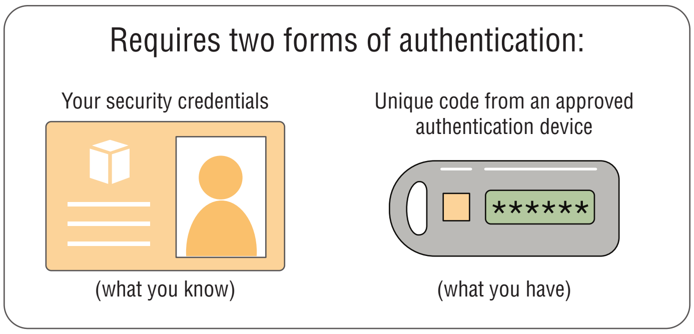
Amazon S3 does not support server-side scripting or dynamic content. We discuss other AWS options for that throughout this study guide.

TIP Static websites can be hosted in Amazon S3.

MFA Delete

MFA is another way to control deletes on your objects in Amazon S3. It does so by adding another layer of protection against unintentional or malicious deletes, requiring an authorized request against Amazon S3 to delete the object. MFA also requires a unique code from a token or an authentication device (virtual or hardware). These devices provide a unique code that will then allow you to delete the object. Figure 3.15 shows what would be required for a user to execute a delete operation on an object when MFA is enabled.

FIGURE 3.15 MFA Delete


Requires two forms of authentication:
Your security credentials
(what you know)
Unique code from an approved authentication device
******
(what you have)

Cross-Region Replication

Cross-region replication (CRR) is a bucket-level configuration that enables automatic, asynchronous copying of objects across buckets in different AWS Regions. We refer to these buckets as the *source* bucket and *destination* bucket. These buckets can be owned by different accounts.

To activate this feature, add a replication configuration to your source bucket to direct Amazon S3 to replicate objects according to the configuration. In the replication configuration, provide information including the following:

- The destination bucket
- The objects that need to be replicated
- Optionally, the destination storage class (otherwise the source storage class will be used)

The replicas that are created in the destination bucket will have these same characteristics as the source objects:

- Key names
- Metadata
- Storage class (unless otherwise specified)
- Object ACL

All data is encrypted in transit across AWS Regions using SSL.

You can replicate objects from a source bucket to only one destination bucket. After Amazon S3 replicates an object, the object cannot be replicated again. For example, even after you change the destination bucket in an existing replication configuration, Amazon S3 will not replicate it again.

TIP After Amazon S3 replicates an object using CRR, the object cannot be replicated again (such as to another destination bucket).

Requirements for CRR include the following:

- Versioning is enabled for both the source and destination buckets.
- Source and destination buckets must be in different AWS Regions.
- Amazon S3 must be granted appropriate permissions to replicate files.

VPC Endpoints

A *virtual private cloud (VPC) endpoint* enables you to connect your VPC privately to Amazon S3 without requiring an internet gateway, *network address translation (NAT)* device, *virtual private network (VPN)* connection, or *AWS Direct Connect* connection. Instances in your VPC do not require public IP addresses to communicate with the resources in the service. Traffic between your VPC and Amazon S3 does not leave the Amazon network.

Amazon S3 uses a gateway type of VPC endpoint. The gateway is a target for a specified route in your route table, used for traffic destined for a supported AWS service. These endpoints are easy to configure, are highly reliable, and provide a secure connection to Amazon S3 that does not require a gateway or NAT instance.

Amazon EC2 instances running in private subnets of a VPC can have controlled access to Amazon S3 buckets, objects, and API functions that are in the same region as the VPC. You can use an Amazon S3 bucket policy to indicate which VPCs and which VPC endpoints have access to your Amazon S3 buckets.

Using the AWS SDKs, AWS CLI, and AWS Explorers

You can use the AWS SDKs when developing applications with Amazon S3. The AWS SDKs simplify your programming tasks by wrapping the underlying REST API. The AWS Mobile SDKs and the AWS Amplify JavaScript library are also available for building connected mobile and web applications using AWS. In addition to AWS SDKs, AWS explorers are available for Visual Studio and Eclipse for *Java Integrated Development Environment* (IDE). In this case, the SDKs and AWS explorers are available bundled together as AWS Toolkits. You can also use the AWS CLI to manage Amazon S3 buckets and objects.

AWS has deprecated SOAP support over HTTP, but it is still available over HTTPS. New Amazon S3 features will not be supported over SOAP. We recommend that you use either the REST API or the AWS SDKs for any new development and migrate any existing SOAP calls when you are able.

Making Requests

Every interaction with Amazon S3 is either authenticated or anonymous. *Authentication* is the process of verifying the identity of the requester trying to access an AWS product (you are who you say you are, and you are allowed to do what you are asking to do). Authenticated requests must include a signature value that authenticates the request sender, generated in part from the requester's AWS access keys. If you are using the AWS SDK, the libraries compute the signature from the keys that you provide. If you make direct REST API calls in your application, however, you must write code to compute the signature and add it to the request.

Stateless and Serverless Applications

Amazon S3 provides developers with secure, durable, and highly scalable object storage that can be used to decouple storage for use in *serverless applications*. Developers can also use Amazon S3 for storing and sharing state in *stateless applications*.

Developers on AWS are regularly moving shared file storage to Amazon S3 for stateless applications. This is a common method for decoupling your compute and storage and increasing the ability to scale your application by decoupling that storage. We will discuss stateless and serverless applications throughout this study guide.

Data Lake

Traditional data storage can no longer provide the agility and flexibility required to handle the volume, velocity, and variety of data used by today's applications. Because of this, many organizations are shifting to a *data lake* architecture.

A *data lake* is an architectural approach that allows you to store massive amounts of data in a central location for consumption by multiple applications. Because data can be stored as is, there is no need to convert it to a predefined schema, and you no longer need to know what questions to ask of your data beforehand.

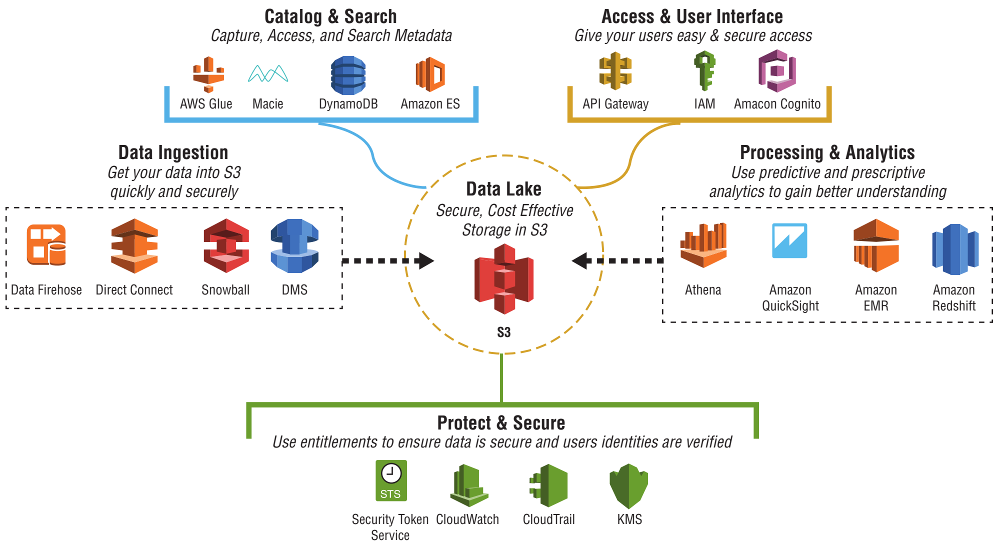
Amazon S3 is a common component of a data lake solution on the cloud, and it can complement your other storage solutions. If you move to a data lake, you are essentially separating compute and storage, meaning that you are going to build and scale your storage and compute separately. You can take storage that you currently have on premises or in your data center and instead use Amazon S3, which then allows you to scale and build your compute in any desired configuration, regardless of your storage.

That design pattern is different from most applications available today, where the storage is tied to the compute. When you separate those two features and instead use a data lake, you achieve an agility that allows you to invent new types of applications while you are managing your storage as an independent entity.

In addition, Amazon S3 lets you grow and scale in a virtually unlimited fashion. You do not have to take specific actions to expand your storage capacity—it grows automatically with your data.

In the data lake diagram shown in Figure 3.16, you will see how to use Amazon S3 as a highly available and durable central storage repository. From there, a virtually unlimited number of services and applications, both on premises and in the cloud, can take advantage of using Amazon S3 as a data lake.

FIGURE 3.16 Data lakes


Catalog & Search
Capture, Access, and Search Metadata
AWS Glue
Macie
DynamoDB
Amazon ES
Access & User Interface
Give your users easy & secure access
API Gateway
IAM
Amacon Cognito
Data Ingestion
Get your data into S3 quickly and securely
Data Firehose
Direct Connect
Snowball
DMS
Data Lake
Secure, Cost Effective Storage in S3
S3
Processing & Analytics
Use predictive and prescriptive analytics to gain better understanding
Athena
Amazon QuickSight
Amazon EMR
Amazon Redshift
Protect & Secure
Use entitlements to ensure data is secure and users identities are verified
STS
Security Token Service
CloudWatch
CloudTrail
KMS

Customers often set up a data lake as part of their migration to the cloud so that they can access their data from new applications on the cloud, migrated applications to the cloud, and applications that have not yet been migrated to the cloud.

Performance

There are a number of actions that Amazon S3 takes by default to help you achieve high levels of performance. Amazon S3 automatically scales to thousands of requests per second per prefix based on your steady state traffic. Amazon S3 will automatically partition your prefixes within hours, adjusting to increases in request rates.

Consideration for Workloads

To optimize the use of Amazon S3 mixed or `GET`-intensive workloads, you must become familiar with best practices for performance optimization.

Mixed request types If your requests are typically a mix of `GET`, `PUT`, `DELETE`, and `GET` Bucket (list objects), choosing appropriate key names for your objects ensures better performance by providing low-latency access to the Amazon S3 index.

`GET`-intensive workloads If the bulk of your workload consists of `GET` requests, you may want to use Amazon CloudFront, a content delivery service (discussed later in this chapter).

Tips for Object Key Naming

The way that you name your keys in Amazon S3 can affect the data access patterns, which may directly impact the performance of your application.

It is a best practice at AWS to design for performance from the start. Even though you may be developing a new application, that application is likely to grow over time. If you anticipate your application growing to more than approximately 1,000 requests per second (including both PUTs and GETs on your object), you will want to consider using a three- or four-character hash in your key names.

If you anticipate your application receiving fewer than 1,000 requests per second and you don't see a lot of traffic in your storage, then you do not need to implement this best practice. Your application will still benefit from Amazon S3's default performance.

NOTE In the past, customers would also add entropy in their key names. Because of recent Amazon S3 performance enhancements, most customers no longer need to worry about introducing entropy in key names.

Example 1: Random Hash

```
examplebucket/232a-2017-26-05-15-00-00/cust1234234/photo1.jpg
examplebucket/7b54-2017-26-05-15-00-00/cust3857422/photo2.jpg
examplebucket/921c-2017-26-05-15-00-00/cust1248473/photo2.jpg
```

TIP A random hash should come before patterns, such as dates and sequential IDs.

Using a *naming hash* can improve the performance of heavy-traffic applications. Object keys are stored in an index in all regions. If you're constantly writing the same key prefix over and over again (for example, a key with the current year), all of your objects will be close to each other within the same partition in the index. When your application experiences an increase in traffic, it will be trying to read from the same section of the index, resulting in decreased performance as Amazon S3 tries to spread out your data to achieve higher levels of throughput.

TIP Always first ensure that your application can accommodate a naming hash.

By putting the hash at the beginning of your key name, you are adding randomness. You could hash the key name and place it at the beginning of your object right after the bucket name. This will ensure that your data will be spread across different partitions and allow you to grow to a higher level of throughput without experiencing a re-indexing slowdown if you go above peak traffic volumes.

Example 2: Naming Hash

```
examplebucket/animations/232a-2017-26-05-15-00/cust1234234/animation1.obj
examplebucket/videos/ba65-2017-26-05-15-00/cust8474937/video2.mpg
examplebucket/photos/8761-2017-26-05-15-00/cust1248473/photo3.jpg
```

In this second example, imagine that you are storing a lot of animations, videos, and photos in Amazon S3. If you know that you are going to have a lot of traffic to those individual prefixes, you can add your hash after the prefix. That allows you to write prefixes into your lifecycle policies or perform *list API* calls against a particular prefix. You are still getting the performance benefit by adding the hash to your key name, but now you can also use the prefix as necessary.

This example allows you to balance the need to list your objects and organize them against the need to spread your data across different partitions for performance.

Amazon S3 Transfer Acceleration

Amazon S3 Transfer Acceleration is a feature that optimizes throughput when transferring larger objects across larger geographic distances. Amazon S3 Transfer Acceleration uses Amazon CloudFront edge locations to assist you in uploading your objects more quickly in cases where you are closer to an edge location than to the region to which you are transferring your files.

Instead of using the public internet to upload objects from Southeast Asia, across the globe to Northern Virginia, take advantage of the global *Amazon content delivery network* (CDN). AWS has edge locations around the world, and you upload your data to the edge location closest to your location. This way, you are traveling across the AWS network backbone to your destination region, instead of across the public internet. This option might give you a significant performance improvement and better network consistency than the public internet.

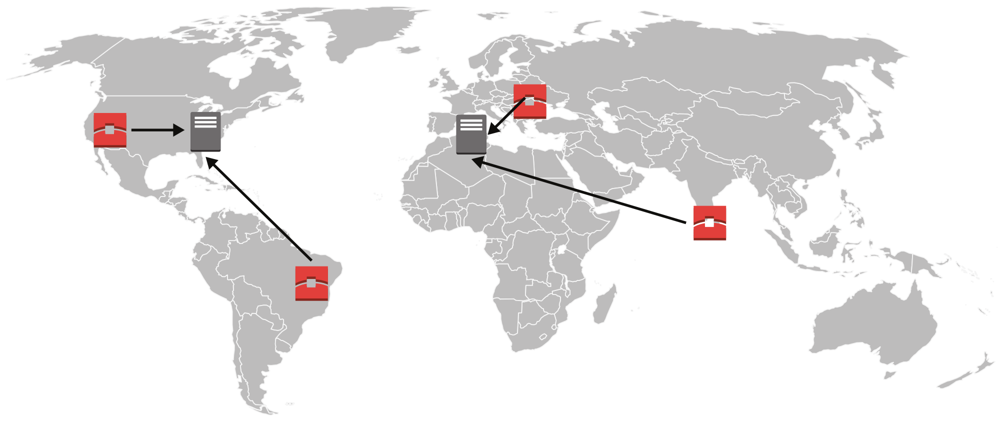
To implement Amazon S3 Transfer Acceleration, you do not need to make any changes to your application. It is enabled by performing the following steps:

1. Enable Transfer Acceleration on a bucket that conforms to DNS naming requirements and does not contain periods (.).
2. Transfer data to and from the acceleration-enabled bucket by using one of the s3-accelerate endpoint domain names.

There is a small fee for using Transfer Acceleration. If your speed using Transfer Acceleration is no faster than it would have been going over the public internet, however, there is no additional charge.

The further you are from a particular region, the more benefit you will derive from transferring your files more quickly by uploading to a closer edge location. Figure 3.17 shows how accessing an edge location can reduce the latency for your users, as opposed to accessing content from a region that is farther away.

FIGURE 3.17 Using an AWS edge location



Multipart Uploads

When uploading a large object to Amazon S3 in a single-threaded manner, it can take a significant amount of time to complete. The multipart upload API enables you to upload large objects in parts to speed up your upload by doing so in parallel.

To use multipart upload, you first break the object into smaller parts, parallelize the upload, and then submit a manifest file telling Amazon S3 that all parts of the object have been uploaded. Amazon S3 will then assemble all of those individual pieces to a single Amazon S3 object.

Multipart upload can be used for objects ranging from 5 MB to 5 TB in size.

Range GETs

Range GETs are similar to multipart uploads, but in reverse. If you are downloading a large object and tracking the offsets, use range `GET`s to download the object as multiple parts instead of a single part. You can then download those parts in parallel and potentially see an improvement in performance.

Amazon CloudFront

Using a CDN like Amazon CloudFront, you may achieve lower latency and higher-throughput performance. You also will not experience as many requests to Amazon S3 because your content will be cached at the edge location. Your users will also experience the performance improvement of having cached storage through Amazon CloudFront versus going back to Amazon S3 for each new `GET` on an object.

TCP Window Scaling

Transmission Control Protocol (TCP) window scaling allows you to improve network throughput performance between your operating system, application layer, and Amazon S3 by supporting window sizes larger than 64 KB. Although it can improve performance, it can be challenging to set up correctly, so refer to the AWS Documentation repository for details.

TCP Selective Acknowledgment

TCP selective acknowledgment is designed to improve recovery time after a large number of packet losses. It is supported by most newer operating systems, but it might have to be enabled. Refer to the Amazon S3 Developer Guide for more information.

Pricing

With Amazon S3, you pay only for what you use. There is no minimum fee, and there is no charge for data transfer into Amazon S3.

You pay for the following:

- The storage that you use
- The API calls that you make (`PUT`, `COPY`, `POST`, `LIST`, `GET`)
- Data transfer out of Amazon S3

Data transfer out pricing is tiered, so the more you use, the lower your cost per gigabyte. Refer to the AWS website for the latest pricing.

TIP Amazon S3 pricing differs from the pricing of Amazon EBS volumes in that if you create an Amazon EBS volume and store nothing on it, you are still paying for the storage space of the volume that you have allocated. With Amazon S3, you pay for the storage space that is being used—not allocated.

Object Lifecycle Management

To manage your objects so that they are stored cost effectively throughout their lifecycle, use a *lifecycle configuration*. A lifecycle configuration is a set of rules that defines actions that Amazon S3 applies to a group of objects.

There are two types of actions:

Transition actions *Transition actions* define when objects transition to another storage class. For example, you might choose to transition objects to the STANDARD_IA storage class 30 days after you created them or archive objects to the GLACIER storage class one year after creating them.

Expiration actions *Expiration actions* define when objects expire. Amazon S3 deletes expired objects on your behalf.

When Should You Use Lifecycle Configuration?

You should use lifecycle configuration rules for objects that have a well-defined lifecycle. The following are some examples:

- If you upload periodic logs to a bucket, your application might need them for a week or a month. After that, you may delete them.
- Some documents are frequently accessed for a limited period of time. After that, they are infrequently accessed. At some point, you might not need real-time access to them, but your organization or regulations might require you to archive them for a specific period. After that, you may delete them.
- You can upload some data to Amazon S3 primarily for archival purposes. For example, archiving digital media, financial, and healthcare records; raw genomics sequence data, long-term database backups; and data that must be retained for regulatory compliance.

With lifecycle configuration rules, you can tell Amazon S3 to transition objects to less expensive storage classes or archive or delete them.

Configuring a Lifecycle

A *lifecycle configuration* (an XML file) comprises a set of rules with predefined actions that you need Amazon S3 to perform on objects during their lifetime. Amazon S3 provides a set of API operations for managing lifecycle configuration on a bucket, and it is stored by Amazon S3 as a *lifecycle subresource* that is attached to your bucket.

You can also configure the lifecycle by using the Amazon S3 console, the AWS SDKs, or the REST API.

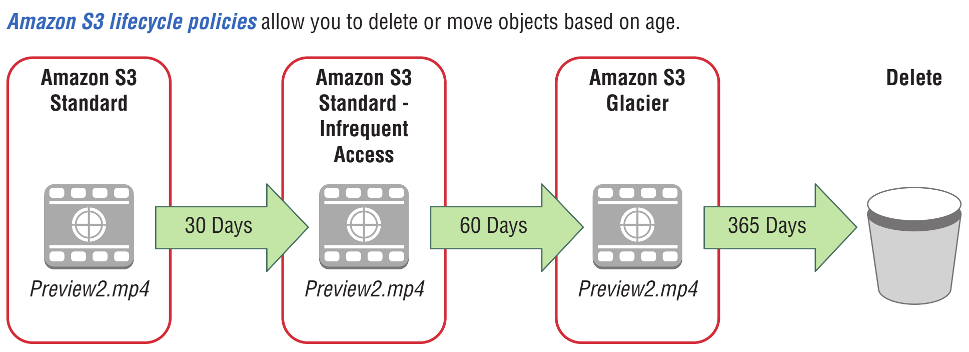
The following lifecycle configuration specifies a rule that applies to objects with key name prefix `logs/`. The rule specifies the following actions:

- Two transition actions
  - Transition objects to the STANDARD_IA storage class 30 days after creation
  - Transition objects to the GLACIER storage class 90 days after creation
- One expiration action that directs Amazon S3 to delete objects a year after creation

```
<LifecycleConfiguration>
  <Rule>
    <ID>example-id</ID>
    <Filter>
       <Prefix>logs/</Prefix>
    </Filter>
    <Status>Enabled</Status>
    <Transition>
      <Days>30</Days>
      <StorageClass>STANDARD_IA</StorageClass>
    </Transition>
    <Transition>
      <Days>90</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
    <Expiration>
      <Days>365</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

Figure 3.18 shows a set of Amazon S3 lifecycle policies in place. These policies move files automatically from one storage class to another as they age out at certain points in time.

FIGURE 3.18 Amazon S3 lifecycle policies


Amazon S3 lifecycle policies allow you to delete or move objects based on age.
Amazon S3 Standard
Preview2.mp4
30 Days
Amazon S3 Standard - Infrequent Access
Preview2.mp4
60 Days
Amazon S3 Glacier
Preview2.mp4
365 Days
Delete

AWS File Storage Services

AWS offers Amazon Elastic File System (Amazon EFS) for file storage to enable you to share access to files that reside on the cloud.

Amazon Elastic File System

Amazon Elastic File System (Amazon EFS) provides scalable file storage and a standard file system interface for use with Amazon EC2. You can create an Amazon EFS file system, configure your instances to mount the file system, and then use an Amazon EFS file system as a common data source for workloads and application running on multiple instances.

Amazon EFS can be mounted to multiple Amazon EC2 instances simultaneously, where it can continue to expand up to petabytes while providing low latency and high throughput.

Consider using Amazon EFS instead of Amazon S3 or Amazon EBS if you have an application (Amazon EC2 or on premises) or a use case that requires a file system and any of the following:

- Multi-attach
- GB/s throughput
- Multi-AZ availability/durability
- Automatic scaling (growing/shrinking of storage)

Customers use Amazon EFS for the following use cases today:

- Web serving
- Database backups
- Container storage
- Home directories
- Content management
- Analytics
- Media and entertainment workflows
- Workflow management
- Shared state management

WARNING

Amazon EFS is not supported on Windows instances.

Creating your Amazon EFS File System

File System

The *Amazon EFS file system* is the primary resource in Amazon EFS, and it is where you store your files and directories. You can create up to 125 file systems per account.

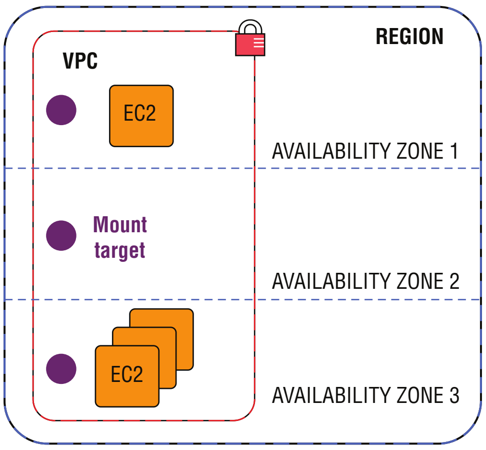
Mount Target

To access your file system from within a VPC, create mount targets in the VPC. A *mount target* is a Network File System (NFS) endpoint within your VPC that includes an IP address and a DNS name, both of which you use in your mount command. A mount target is highly available, and it is illustrated in Figure 3.19.

Accessing an Amazon EFS File System

There are several different ways that you can access an Amazon EFS file system, including using Amazon EC2 and AWS Direct Connect.

FIGURE 3.19 Mount target



Using Amazon Elastic Compute Cloud

To access a file system from an *Amazon Elastic Compute Cloud* (Amazon EC2) instance, you must mount the file system by using the standard Linux `mount` command, as shown in Figure 3.20. The file system will then appear as a local set of directories and files. An NFS v4.1 client is standard on Amazon Linux AMI distributions.

FIGURE 3.20 Mounting the file system

```
mount -t nfs4 -o nfsvers=4.1
     ↳ [file system DNS name]:/
     ↳ /[user's target directory]
```

In your command, specify the file system type (nfs4), the version (4.1), the file system DNS name or IP address, and the user's target directory.

A file system belongs to a region, and your Amazon EFS file system spans all Availability Zones in that region. Once you have mounted your file system, data can be accessed from any Availability Zone in the region within your VPC while maintaining full consistency. Figure 3.21 shows how you communicate with Amazon EC2 instances within a VPC.

Using AWS Direct Connect

You can also mount your on-premises servers to Amazon EFS in your Amazon VPC using AWS Direct Connect. With *AWS Direct Connect*, you can mount your on-premises servers to Amazon EFS using the same mount command used to mount in Amazon EC2. Figure 3.22 shows how to use AWS Direct Connect with Amazon EFS.

FIGURE 3.21 Using Amazon EFS

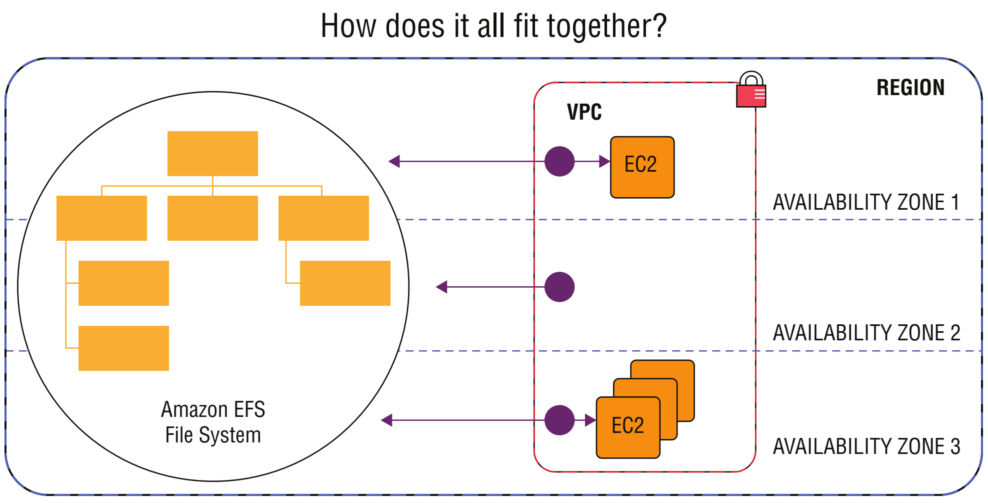
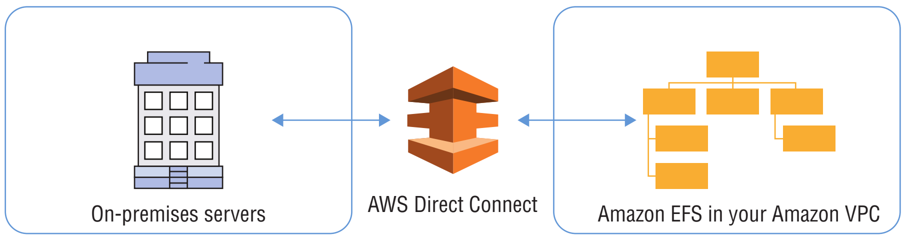


FIGURE 3.22 Using AWS Direct Connect with Amazon EFS



Customers can use Amazon EFS combined with AWS Direct Connect for migration, bursting, or backup and disaster recovery.

Syncing Files Using AWS DataSync

Now that you have a functioning Amazon EFS file system, you can use *AWS DataSync* to synchronize files from an existing file system to Amazon EFS. AWS DataSync can synchronize your file data and also file system metadata such as ownership, time stamps, and access permissions.

To do this, download and deploy a sync agent from the Amazon EFS console as either a *virtual machine* (VM) image or an AMI.

Next, create a sync tack and configure your source and destination file systems. Then start your task to begin syncing the files and monitor the progress of the file sync using Amazon CloudWatch.

Performance

Amazon EFS is designed for a wide spectrum of performance needs, including the following:

- High throughput and parallel I/O
- Low latency and serial I/O

To support those two sets of workloads, Amazon EFS offers two different performance modes, as described here:

General purpose (default) General-purpose mode is the default mode, and it is used for latency-sensitive applications and general-purpose workloads, offering the lowest latencies for file operations. While there is a trade-off of limiting operations to 7,000 per second, general-purpose mode is the best choice for most workloads.

Max I/O If you are running large-scale and data-heavy applications, then choose the max I/O performance option, which provides you with a virtually unlimited ability to scale out throughput and IOPS, but with a trade-off of slightly higher latencies. Use max I/O when you have 10 or more instances accessing your file system concurrently, as shown in Table 3.6.

TABLE 3.6 I/O Performance Options

Mode	What's It For?	Advantages	Trade-Offs	When to Use
General purpose (default)	Latency-sensitive applications and general-purpose workloads	Lowest latencies for file operations	Limit of 7,000 ops/sec	Best choice for most workloads
Max I/O	Large-scale and data-heavy applications	Virtually unlimited ability to scale out throughput/ IOPS	Slightly higher latencies	Consider if 10 (or more) instances are accessing your file system concurrently

If you are not sure which mode is best for your usage pattern, use the `PercentIOLimit` Amazon CloudWatch metric to determine whether you are constrained by general-purpose mode. If you are regularly hitting the 7,000 IOPS limit in general-purpose mode, then you will likely benefit from max I/O performance mode.

As discussed with the CAP theorem earlier in this study guide, there are differences in both performance and trade-off decisions when you're designing systems that use Amazon EFS and Amazon EBS. The distributed architecture of Amazon EFS results in a small increase in latency for each operation, as the data that you are storing gets pushed across multiple servers in multiple Availability Zones. Amazon EBS can provide lower latency than Amazon EFS, but at the cost of some durability. With Amazon EBS, you provision the size of the device, and if you reach its maximum limit, you must increase its size or add more volumes, whereas Amazon EFS scales automatically. Table 3.7 shows the various performance and other characteristics for Amazon EFS as related to Amazon EBS Provisioned IOPS.

TABLE 3.7 Amazon EBS Performance Relative to Amazon EFS

		Amazon EFS	**Amazon EBS Provisioned IOPS**
Performance	Per-operation latency	Low, consistent	Lowest, consistent
	Throughput scale	Multiple GBs per second	Single GB per second
Characteristics	Data availability/ durability	Stored redundantly across multiple Availability Zones	Stored redundantly in a single Availability Zone
	Access	1 to 1000s of EC2 instances, from multiple Availability Zones, concurrently	Single Amazon EC2 instance in a single Availability Zone
	Use cases	Big Data and analytics, media processing workflows, content management, web serving, home directories	Boot volumes, transactional and NoSQL databases, data warehousing, ETL

Security

You can implement security in multiple layers with Amazon EFS by controlling the following:

- Network traffic to and from file systems (mount targets) using the following:
  - VPC security groups
  - Network ACLs
- File and directory access by using POSIX permissions
- Administrative access (API access) to file systems by using IAM. Amazon EFS supports:
  - Action-level permissions
  - Resource-level permissions

TIP Familiarize yourself with the Amazon EFS product, details, and FAQ pages. Some exam questions may be answered by components from those pages.

Storage Comparisons

This section provides valuable charts that can serve as a quick reference if you are tasked with choosing a storage system for a particular project or application.

Use Case Comparison

Table 3.8 will help you understand the main properties and use cases for each of the cloud storage products on AWS.

TABLE 3.8 AWS Cloud Storage Products

If You Need:	Consider Using:
Persistent local storage for Amazon EC2, relational and NoSQL databases, data warehousing, enterprise applications, big data processing, or backup and recovery	Amazon EBS
A file system interface and file system access semantics to make data available to one or more Amazon EC2 instances for content serving, enterprise applications, media processing workflows, big data storage, or backup and recovery	Amazon EFS
A scalable, durable platform to make data accessible from any internet location for user-generated content, active archive, serverless computing, Big Data storage, or backup and recovery	Amazon S3
Highly affordable, long-term storage that can replace tape for archive and regulatory compliance	Amazon S3 Glacier
A hybrid storage cloud augmenting your on-premises environment with AWS cloud storage for bursting, tiering, or migration	AWS Storage Gateway
A portfolio of services to help simplify and accelerate moving data of all types and sizes into and out of the AWS Cloud	AWS Cloud Data Migration Services

Storage Temperature Comparison

Table 3.9 shows a comparison of instance store, Amazon EBS, Amazon S3, and Amazon S3 Glacier.

TIP

Understanding Table 3.9 will help you make decisions about latency, size, durability, and cost during the exam.

TABLE 3.9 Storage Comparison

	Instance Store	Amazon EBS	Amazon S3	Amazon S3 Glacier
Average latency	ms		ms, sec, min (~ size)	hrs
Data volume	4 GB to 48 TB	1 GiB to 1 TiB	No limit	
Item size	Block storage		5 TB max	40 TB max
Request rate	Very high		Low to very high (no limit)	Very low (no limit)
Cost/GB per month	Amazon EC2 instance cost	¢¢	¢	
Durability	Low	High	Very high	Very high
Temperature	Hot <——————————————————————> Cold			

Comparison of Amazon EBS and Instance Store

Before considering Amazon EC2 instance store as a storage option, make sure that your data does *not* meet any of these criteria:

- Must persist through instance stops, terminations, or hardware failures
- Needs to be encrypted at the full volume level
- Needs to be backed up with Amazon EBS snapshots
- Needs to be removed from instances and reattached to another

If your data meets any of the previous four criteria, use an Amazon EBS volume. Otherwise, compare instance store and Amazon EBS for storage.

Because instance store is directly attached to the host computer, it will have lower latency than an Amazon EBS volume attached to the Amazon EC2 instance. Instance store is provided at no additional cost beyond the price of the Amazon EC2 instance you choose (if the instance has instance store[s] available), whereas Amazon EBS volumes incur an additional cost.

Comparison of Amazon S3, Amazon EBS, and Amazon EFS

Table 3.10 is a useful in helping you to compare performance and storage characteristics for Amazon's highest-performing file, object, and block cloud storage offerings. This comparison will also be helpful when choosing the right data store for the applications that you are developing. It is also important for the exam.

TABLE 3.10 Storage Service Comparison (EFS, S3, and EBS)

		File Amazon EFS	**Object Amazon S3**	**Block Amazon EBS**
Performance	Per-operation latency	Low, consistent	Low, for mixed request types, and integration with CloudFront	Low, consistent
	Throughput scale	Multiple GB per second		Single GB per second
Characteristics	Data Availability/ Durability	Stored redundantly across multiple Availability Zones		Stored redundantly in a single Availability Zone
	Access	One to thousands of Amazon EC2 instances or on-premises servers, from multiple Availability Zones, concurrently	One to millions of connections over the web	Single Amazon EC2 instance in a single Availability Zone
	Use Cases	Web serving and content management, enterprise applications, media and entertainment, home directories, database backups, developer tools, container storage, Big Data analytics	Web serving and content management, media and entertainment, backups, Big Data analytics, data lake	Boot volumes, transactional and NoSQL databases, data warehousing, ETL

run. The remaining sections allow for fine-grained configurations to integrate packages, sources, files, and container commands.

NOTE Launch environments from integrated development environment (IDE) tools to avoid poorly formatted configurations and source bundles that could cause unrecoverable failures.

You apply configuration files in the `ebextensions` directory to Elastic Beanstalk stacks. The stacks are the AWS resources that you allocate for your infrastructure and application. If you have any resource, such as Amazon VPC, Amazon EC2, or Amazon S3, that was updated or configured, these files deploy with your changes. You can zip your `ebextension` files, upload, and apply them to multiple application environments. You can view your environment variables in `option_settings` for future evaluation or changes. These are accessible from the AWS Management Console, command line, and API calls.

TIP You can view Elastic Beanstalk stacks in AWS CloudFormation, but always use the Elastic Beanstalk service and `ebextensions` to make modifications. This way, edits and modifications to the application stacks are simplified without introducing unrecoverable failures.

Elastic Beanstalk generates logs that you can view to troubleshoot your environments and resources. The logs display Amazon EC2 operational logs and logs that are specific to servers running for your applications.

Integrating with Other AWS Services

Elastic Beanstalk automatically integrates or manages other AWS services with application code to provision efficient working environments. However, you might find it necessary to add additional services, such as Amazon S3 for content storage or Amazon DynamoDB for data records, to work with an environment. To grant access between any integrated service and Elastic Beanstalk, you must configure permissions in IAM.

Amazon S3

You can use Amazon S3 to store static content you want to integrate with your application and point directly to objects you store in Amazon S3 from your application or from other resources. In addition to setting permissions in IAM policies, take advantage of presigned URLs for controlled Amazon S3 `GET` and `PUT` operations.

Amazon CloudFront

You can integrate your Elastic Beanstalk environment with Amazon CloudFront, which provides content delivery and distribution through the use of edge locations throughout the world. This can decrease the time in which your content is delivered to you, as the content is cached and routed through the closest edge location serving you. After you deploy your application on Elastic Beanstalk, use the Amazon CloudFront content delivery network (CDN) to cache static content from your application. To identify the source of your content in Amazon CloudFront, you can use URL path patterns to cache your content and then retrieve it from the cache. This approach serves your content more rapidly and offloads requests directly sourced from your application.

AWS Config

With *AWS Config*, you can visualize configuration history and how configurations evolve over time. Tracking changes helps you to fulfill compliance obligations and meet auditing requirements. You can integrate AWS Config directly with your application and its versions or your Elastic Beanstalk environment. *You can customize AWS Config to record changes per resource, per region, or globally*. In the AWS Config console, you can select Elastic Beanstalk resource types to record specific applications and environment resources. You can view the recorded information in the AWS Config dashboard under Resource Inventory.

Amazon RDS

Various options are available for creating databases for your environment, such as Amazon Relational Database Service (Amazon RDS) for SQL databases and Amazon DynamoDB for NoSQL databases. Elastic Beanstalk can create a database and store and retrieve data for any of your environments. Each service has its own features to handle scaling, capacity, performance, and availability.

To store, read, or write to your data records, you can set up an Amazon RDS database instance or an Amazon DynamoDB table by using the same configuration files for your other service option settings. You must create connections to the database, which require you to set up password management in Elastic Beanstalk. Your configurations are saved in the `ebextensions` directory. You can also create direct connections, within your application code or application configuration files, to both internal and external databases. When using Amazon RDS, avoid accidentally deleting and re-creating databases without a properly installed backup. To reduce the risk of losing data, take a manual snapshot of the master Amazon RDS database immediately before deleting.

NOTE

If you create periodic tasks with a worker environment, Elastic Beanstalk automatically creates an Amazon DynamoDB table to perform leader election and stores task information.

Amazon ElastiCache

For caching capabilities, you can integrate Amazon ElastiCache service clusters with the Elastic Beanstalk environment. If you use a nonlegacy container, you can set your configuration files to use the supported container and then offload requests to the cache cluster.

Route 53 to configure DNS health checks to route traffic to healthy endpoints or to monitor independently the health of your application and its endpoints. Amazon Route 53 Traffic Flow makes it easy for you to manage traffic globally through a variety of routing types, including latency-based routing, geolocation, geoproximity, and weighted round-robin, all of which can be combined with DNS failover to enable a variety of low-latency, fault-tolerant architectures.

Using Amazon Route 53 Traffic Flow's simple visual editor, you can easily manage how your end users are routed to your application's endpoints—whether in a single AWS Region or distributed around the globe. Amazon Route 53 also offers domain name registration. You can purchase and manage domain names such as `example.com`, and Amazon Route 53 will automatically configure DNS settings for your domains.

Speeding Up Content Delivery with Amazon CloudFront

Latency is an increasingly important aspect when you deliver web applications to the end user, as you always want your end user to have an efficient, low-latency experience on your website. Increased latency can result in both decreased customer satisfaction and decreased sales. One way to decrease latency is to use *Amazon CloudFront* to move your content closer to your end users. Amazon CloudFront has two delivery methods to deliver content. The first is a web distribution, and this is for storing of `.html`, `.css`, and graphic files. Amazon CloudFront also provides the ability to have an RTMP distribution, which speeds up distribution of your streaming media files using Adobe Flash Media Server's RTMP protocol. An RTMP distribution allows an end user to begin playing a media file before the file has finished downloading from a CloudFront edge location.

To use Amazon CloudFront with your Amazon S3 static website, perform these tasks:

1. Choose a delivery method.

   In the example, Amazon S3 is used to store a static web page; thus, you will be using the Web delivery method. However, as mentioned previously, you could also use RTMP for streaming media files.

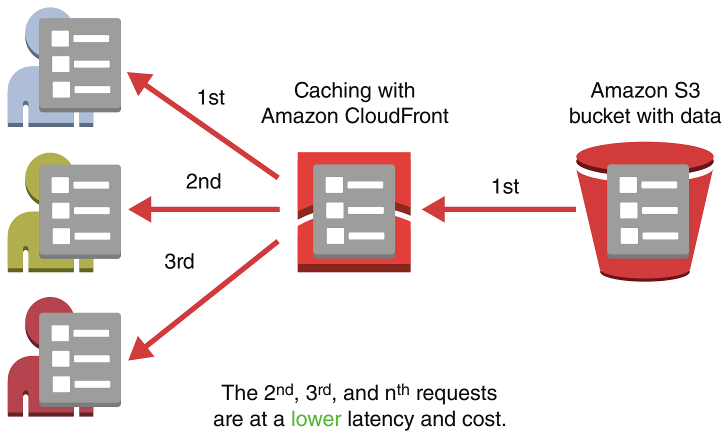
2. Specify the cache behavior. A cache behavior lets you configure a variety of CloudFront functionality for a given URL path pattern for files on your website.

3. Choose the distribution settings and network that you want to use. For example, you can use all edge locations or only U.S., Canada, and Europe locations.

Amazon CloudFront enables you to cache your data to minimize redundant data-retrieval operations. Amazon CloudFront reduces the number of requests to which your origin server must respond directly. This reduces the load on your origin server and reduces latency because more objects are served from Amazon CloudFront edge locations, which are closer to your users.

The Amazon S3 bucket pushes the first request to Amazon CloudFront's cache. The second, third, and n^{th} requests pull from the Amazon CloudFront's cache at a lower latency and cost, as shown in Figure 13.1.

FIGURE 13.1 Amazon CloudFront cache


1st
Caching with Amazon CloudFront
Amazon S3 bucket with data
2nd
1st
3rd
The 2nd, 3rd, and nth requests are at a lower latency and cost.

The more requests that Amazon CloudFront is able to serve from edge caches as a proportion of all requests (that is, the greater the cache hit ratio), the fewer viewer requests that Amazon CloudFront needs to forward to your origin to get the latest version or a unique version of an object. You can view the percentage of viewer requests that are hits, misses, and errors in the Amazon CloudFront console.

A number of factors affect the cache hit ratio. You can adjust your Amazon CloudFront distribution configuration to improve the cache hit ratio.

Use Amazon CloudFront with Amazon S3 to improve your performance, decrease your application's latency and costs, and provide a better user experience. Amazon CloudFront is also a serverless service, and it fits well with serverless stack services, especially when you use it in conjunction with Amazon S3.

Dynamic Data with Amazon API Gateway (Logic or App Tier)

This section details how to use dynamic data with the Amazon API Gateway service in a logic tier or app tier.

Amazon API Gateway is a fully managed, serverless AWS service, with no server that runs inside your environment to define, deploy, monitor, maintain, and secure APIs at any scale. Clients integrate with the APIs that use standard HTTPS requests. Amazon API Gateway can integrate with a service-oriented multitier architecture with Amazon services, such as AWS Lambda and Amazon EC2. It also has specific features and qualities that make it a powerful edge for your logic tier. You can use these features and qualities to enhance and build your dynamic web application.

The Amazon API Gateway integration strategy that provides access to your code includes the following:

Control service Uses REST to provide access to Amazon services, such as AWS Lambda, Amazon Kinesis, Amazon S3, and Amazon DynamoDB. The access methods include the following:

- Consoles
- CLI
- SDKs
- REST API requests and responses

Execution service Uses standard HTTP protocols or language-specific SDKs to deploy API access to backend functionality.

WARNING Do not directly expose resources or the API—always use AWS edge services and the Amazon API Gateway service to safeguard your resources and APIs.

Endpoints

There are three types of *endpoints* for Amazon API Gateway.

Regional endpoints Live inside the AWS Region, such as `us-west-2`.

Edge optimized endpoints Use Amazon CloudFront, a content delivery web service with the AWS global network of edge locations as connection points for clients, and integrate with your API.

Private endpoints Can live only inside of a *virtual private cloud* (VPC).

You use Amazon API Gateway to help drive down the total response time latency of your API. You can improve the performance of specific API requests with Amazon API Gateway to store responses in an optional in-memory cache. This not only provides performance benefits for API requests that repeat, but it also reduces backend executions, which helps to reduce overall costs.

The API endpoint can be a default host name or a custom domain name. The default host name is as follows:

```
{api-id}.execute-api.{region}.amazonaws.com
```

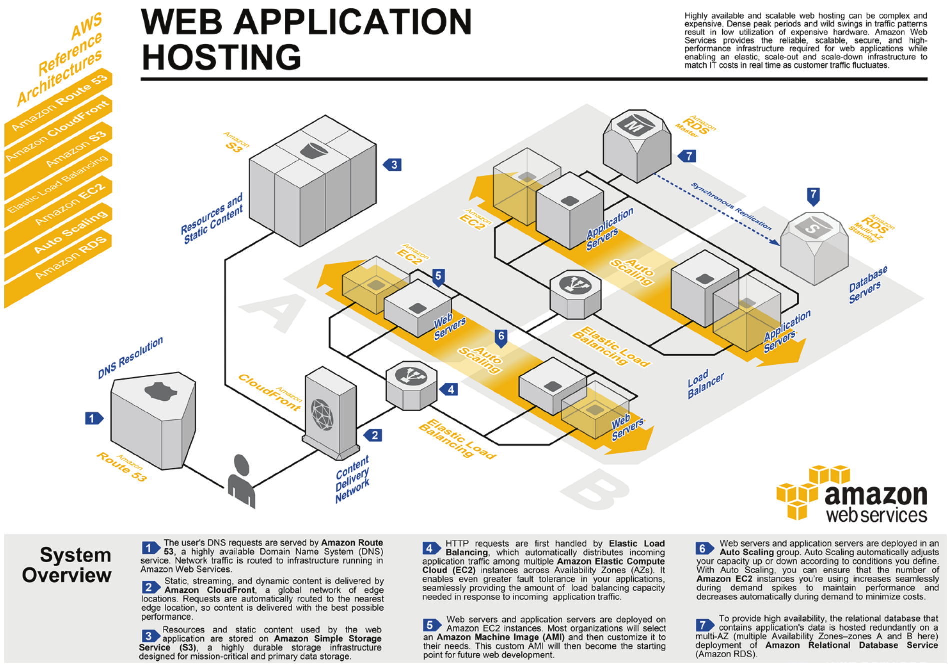
In addition to using the higher-level mobile and JavaScript SDKs, you can also use the lower-level APIs available via the following AWS SDKs to integrate all Amazon Cognito functionality in your applications:

- Java SDK
- .NET SDK
- Node.js SDK
- Python SDK
- PHP SDK
- Ruby SDK

Standard Three-Tier vs. the Serverless Stack

This chapter has introduced serverless services and their benefits. Now that you know about some of the serverless services that are available in AWS, let's compare a traditional three-tier application against a serverless application architecture. Figure 13.5 shows a typical three-tier web application.

FIGURE 13.5 Standard three-tier web infrastructure architecture



Highly available and scalable web hosting can be complex and expensive. Dense peak periods and wild swings in traffic patterns result in low utilization of expensive hardware. Amazon Web Services provides the reliable, scalable, secure, and high-performance infrastructure required for web applications while enabling an elastic, scale-out and scale-down infrastructure to match IT costs in real time as customer traffic fluctuates.

1 The user's DNS requests are served by **Amazon Route 53**, a highly available Domain Name System (DNS) service. Network traffic is routed to infrastructure running in Amazon Web Services.

2 Static, streaming, and dynamic content is delivered by **Amazon CloudFront**, a global network of edge locations. Requests are automatically routed to the nearest edge location, so content is delivered with the best possible performance.

3 Resources and static content used by the web application are stored on **Amazon Simple Storage Service (S3)**, a highly durable storage infrastructure designed for mission-critical and primary data storage.

4 HTTP requests are first handled by **Elastic Load Balancing**, which automatically distributes incoming application traffic among multiple **Amazon Elastic Compute Cloud (EC2)** instances across Availability Zones (AZs). It enables even greater fault tolerance in your applications, seamlessly providing the amount of load balancing capacity needed in response to incoming application traffic.

5 Web servers and application servers are deployed on Amazon EC2 instances. Most organizations will select an **Amazon Machine Image (AMI)** and then customize it to their needs. This custom AMI will then become the starting point for future web development.

6 Web servers and application servers are deployed in an **Auto Scaling** group. Auto Scaling automatically adjusts your capacity up or down according to conditions you define. With Auto Scaling, you can ensure that the number of **Amazon EC2** instances you're using increases seamlessly during demand spikes to maintain performance and decreases automatically during demand to minimize costs.

7 To provide high availability, the relational database that contains application's data is hosted redundantly on a multi-AZ (multiple Availability Zones–zones A and B here) deployment of **Amazon Relational Database Service** (Amazon RDS).

Source: `https://media.amazonwebservices.com/architecturecenter/AWS_ac_ra_web_01.pdf`

This architecture uses the following components and services:

Routing: Amazon Route 53

Content distribution network (CDN): Amazon CloudFront

Static data: Amazon S3

High availability/decoupling: Application load balancers

Web servers: Amazon EC2 with Auto Scaling

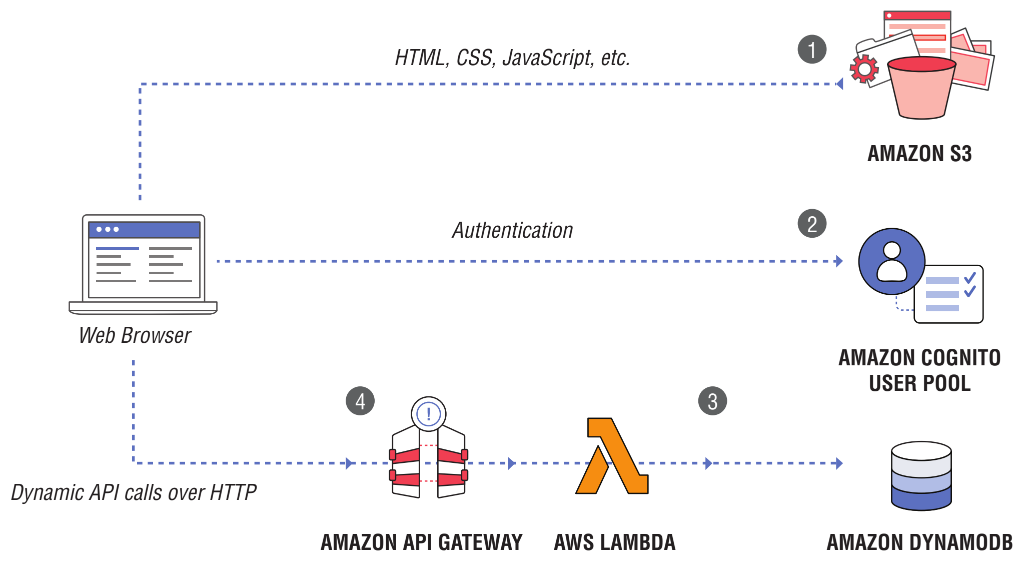
App servers: Amazon EC2 with Auto Scaling

Database: Amazon RDS in a multi-AZ configuration

Amazon Route 53 provides a DNS service that allows you to take domain names such as `examplecompany.com` and translate them to an IP address that points to running servers.

The CDN shown in Figure 13.6 is the Amazon CloudFront service, which improves your site performance with the use of its global content delivery network.

FIGURE 13.6 Serverless web application architecture



Source: `https://aws.amazon.com/getting-started/projects/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/`

Amazon S3 stores your static files such as photos or movie files.

Application load balancers are responsible for distributing load across Availability Zones to your Amazon EC2 servers, which run your web application with a service such as Apache or NGINX.

Application servers are responsible for performing business logic prior to storing the data in your database servers that are run by Amazon RDS.

Amazon RDS is the managed database server, and it can run an Amazon Aurora, Microsoft SQL Server, Oracle SQL Server, MySQL, PostgreSQL, or MariaDB database server.

While this architecture is a robust and highly available service, there are several downsides, including the fact that you have to manage servers. You are responsible for patching those servers, preventing downtime associated with those patches, and proper server scaling.

In a typical serverless web application architecture, you also run a web application, but you have zero servers that run inside your AWS account, as shown in Figure 13.6.

Serverless web application architecture services include the following:

Routing: Amazon Route 53

Web servers/static data: Amazon S3

User authentication: Amazon Cognito user pools

App servers: Amazon API Gateway and AWS Lambda

Database: Amazon DynamoDB

Amazon Route 53 is your DNS, and you can use Amazon CloudFront for your CDN.

You can also use Amazon S3 for your web servers. In this architecture, you use Amazon S3 to host your entire static website. You use JavaScript to make API calls to the Amazon API Gateway service.

For your business or application servers, you use Amazon API Gateway in conjunction with AWS Lambda. This allows you to retrieve and save data dynamically.

You use Amazon DynamoDB as a serverless database service, and you do not provision any Amazon EC2s inside of your Amazon VPC account. Amazon DynamoDB is also a great database service for storing session state for stateful applications. You can use Amazon RDS instead if you need a relational database. However, it would not then be a fully serverless stack. There is a new service released called *Amazon Aurora Serverless*, which is a full RDS MySQL 5.6–compatible service that is completely serverless. This would allow you to run a traditional SQL database, but one that has the benefit of being serverless. Amazon Aurora Serverless is discussed in the next section.

You use Amazon Cognito user pools for user authentication, which provides a secure user directory that can scale to hundreds of millions of users. Amazon Cognito User Pools is a fully managed service with no servers for you to manage. While user authentication was not shown in Figure 13.6, you can use your web server tier to talk to a user directory, such as *Lightweight Directory Access Protocol* (LDAP), for user authentication.

As you can see, while some of the components are the same, you may use them in slightly different ways. By taking advantage of the AWS global network, you can develop fully scalable, highly available web applications—all without having to worry about maintaining or patching servers.

Amazon Aurora Serverless

Amazon Aurora Serverless is an on-demand, auto-scaling configuration for the Aurora MySQL-compatible edition, where the database automatically starts, shuts down, and scales up or down as needed by your application. This allows you to run a traditional SQL database in the cloud without needing to manage any infrastructure or instances.

With Amazon Aurora Serverless, you also get the same high availability as traditional Amazon Aurora, which means that you get six-way replication across three Availability Zones inside of a region in order to prevent against data loss.

Amazon Aurora Serverless is great for infrequently used applications, new applications, variable workloads, unpredictable workloads, development and test databases, and multitenant applications. This is because you can scale automatically when you need to and scale down when application demand is not high. This can help cut costs and save you the heartache of managing your own database infrastructure.

Amazon Aurora Serverless is easy to set up, either through the console or directly with the CLI. To create an Amazon Aurora Serverless cluster with the CLI, you can run the following command:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora
--engine-version 5.6.10a \
--engine-mode serverless --scaling-configuration
MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true \
--master-username user-name --master-user-password password \
--db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
-region us-east-1
```

Amazon Aurora Serverless gives you many of the similar benefits as other serverless technologies, such as AWS Lambda, but from a database perspective. Managing databases is hard work, and with Amazon Aurora Serverless, you can utilize a database that automatically scales and you don't have to manage any of the underlying infrastructure.

AWS Serverless Application Model

The *AWS Serverless Application Model* (AWS SAM) allows you to create and manage resources in your serverless application with *AWS CloudFormation* to define your serverless application infrastructure as a SAM template. A *SAM template* is a JSON or YAML configuration file that describes the AWS Lambda functions, API endpoints, tables, and other resources in your application. With simple commands, you upload this template to AWS CloudFormation, which creates the individual resources and groups them into an *AWS CloudFormation stack* for ease of management. When you update your AWS SAM template, you re-deploy the changes to this stack. AWS CloudFormation updates the individual resources for you.

AWS SAM is an extension of AWS CloudFormation. You can define resources by using the AWS CloudFormation in your AWS SAM template. This is a powerful feature, as you can use AWS SAM to create a template of your serverless infrastructure, which you can then build into a DevOps pipeline. For example, examine the following:

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: 'Example of Multiple-Origin CORS using API Gateway and Lambda'
```

```
Resources:
  ExampleRoot:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: '.'
      Handler: 'routes/root.handler'
      Runtime: 'nodejs8.10'
      Events:
        Get:
          Type: 'Api'
          Properties:
            Path: '/'
            Method: 'get'
  ExampleTest:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: '.'
      Handler: 'routes/test.handler'
      Runtime: 'nodejs8.10'
      Events:
        Delete:
          Type: 'Api'
          Properties:
            Path: '/test'
            Method: 'delete'
        Options:
          Type: 'Api'
          Properties:
            Path: '/test'
            Method: 'options'

Outputs:
    ExampleApi:
      Description: "API Gateway endpoint URL for Prod stage for API Gateway
Multi-Origin CORS Function"
      Value: !Sub "https://${ServerlessRestApi}.execute-api.${AWS::Region}
.amazonaws.com/Prod/"
    ExampleRoot:
      Description: "API Gateway Multi-Origin CORS Lambda Function (Root) ARN"
      Value: !GetAtt ExampleRoot.Arn
    ExampleRootIamRole:
      Description: "Implicit IAM Role created for API Gateway Multi-Origin CORS
Function (Root)"
      Value: !GetAtt ExampleRootRole.Arn
    ExampleTest:
      Description: "API Gateway Multi-Origin CORS Lambda Function (Test) ARN"
      Value: !GetAtt ExampleTest.Arn
    ExampleTestIamRole:
      Description: "Implicit IAM Role created for API Gateway Multi-Origin CORS
Function (Test)"
      Value: !GetAtt ExampleTestRole.Arn
```

In the previous code example, you create two AWS Lambda functions and then associate *three* different Amazon API Gateway endpoints to trigger those functions. To deploy this AWS SAM template, download the template and all of the necessary dependencies from here:

```
https://github.com/awslabs/serverless-application-model/tree/develop/
examples/apps/api-gateway-multiple-origin-cors
```

AWS SAM is similar to AWS CloudFormation, with a few key differences, as shown in the second line:

```
Transform: 'AWS::Serverless-2016-10-31'
```

This important line of code transforms the AWS SAM template into an AWS CloudFormation template. Without it, the AWS SAM template will not work.

Similar to the AWS CloudFormation, you also have a `Resources` property where you define infrastructure to provision. The difference is that you provision serverless services with a new `Type` called `AWS::Serverless::Function`. This provisions an AWS Lambda function to define all properties from an AWS Lambda point of view. AWS Lambda includes `Properties`, such as `MemorySize`, `Timeout`, `Role`, `Runtime`, `Handler`, and others.

While you can create an AWS Lambda function with AWS CloudFormation using `AWS::Lambda::Function`, the benefit of AWS SAM lies in a property called `Event`, where you can tie in a trigger to an AWS Lambda function, all from within the `AWS::Serverless::Function` resource. This `Event` property makes it simple to provision an AWS Lambda function and configure it with an Amazon API Gateway trigger. If you use AWS CloudFormation, you would have to declare an Amazon API Gateway separately with `AWS::ApiGateway::Resource`.

To summarize, AWS SAM allows you to provision serverless resources more rapidly with less code by extending AWS CloudFormation.

AWS SAM CLI

Now that we've addressed AWS SAM, let's take a closer look at the AWS SAM CLI. With AWS SAM, you can define templates, in JSON or YAML, which are designed for provisioning serverless applications through AWS CloudFormation.

AWS SAM CLI is a command line interface tool that creates an environment in which you can develop, test, and analyze your serverless-based application, all locally. This allows you to test your AWS Lambda functions before uploading them to the AWS service. AWS SAM CLI also allows you to develop and test your code quickly, and this gives you the ability to test it locally, which allows you to develop it faster. Previously, you would have had to upload your code each time you wanted to test an AWS Lambda function. Now, with the AWS SAM CLI, you can develop faster and get your application out the door more quickly.

To use AWS SAM CLI, you must meet a few prerequisites. You must install Docker, have Python 2.7 or 3.6 installed, have pip installed, install the AWS CLI, and finally install the AWS SAM CLI. You can read more about how to install AWS SAM CLI at `https://github.com/awslabs/aws-sam-cli`.

With AWS SAM CLI, you must define three key things.

- You must have a valid AWS SAM template, which defines a serverless application.
- You must have the AWS Lambda function defined. This can be in any valid language that Lambda currently supports, such as Node.js, Java 8, Python, and so on.
- You must have an event source. An *event source* is simply an `event.json` file that contains all the data that the Lambda function expects to receive. Valid event sources are as follows:
  - Amazon Alexa
  - Amazon API Gateway
  - AWS Batch
  - AWS CloudFormation
  - Amazon CloudFront
  - AWS CodeCommit
  - AWS CodePipeline
  - Amazon Cognito
  - AWS Config
  - Amazon DynamoDB
  - Amazon Kinesis
  - Amazon Lex
  - Amazon Rekognition
  - Amazon Simple Storage Service (Amazon S3)
  - Amazon Simple Email Service (Amazon SES)
  - Amazon Simple Notification Service (Amazon SNS)
  - Amazon Simple Queue Service (Amazon SQS)
  - AWS Step Functions

To generate this JSON event source, you can simply run this command in the AWS SAM CLI:

```
sam local generate-event <service> <event>
```

AWS SAM CLI is a great tool that allows developers to iterate quickly on their serverless applications. You will learn how to create and test an AWS Lambda function locally in the "Exercises" section of this chapter.

AWS Serverless Application Repository

The *AWS Serverless Application Repository* enables you to deploy code samples, components, and complete applications quickly for common use cases, such as web and mobile backends, event and data processing, logging, monitoring, Internet of Things (IoT), and more. Each application is packaged with an AWS SAM template that defines the AWS resources. Publicly shared applications also include a link to the application's source code. There is no additional charge to use the serverless application repository. You pay only for the AWS resources you use in the applications you deploy.

You can also use the serverless application repository to publish your own applications and share them within your team, across your organization, or with the community at large. This allows you to see what other people and organizations are developing.

Serverless Application Use Cases

Case studies on running serverless applications are located at the following URLs:

The Coca-Cola Company:

`https://aws.amazon.com/blogs/aws/things-go-better-with-step-functions/`

FINRA:

`https://aws.amazon.com/solutions/case-studies/finra-data-validation/`

iRobot:

`https://aws.amazon.com/solutions/case-studies/irobot/`

Localytics:

`https://aws.amazon.com/solutions/case-studies/localytics/`

Summary

This chapter covered the AWS serverless core services, how to store your static files inside of Amazon S3, how to use Amazon CloudFront in conjunction with Amazon S3, how to integrate your application with user authentication flows using Amazon Cognito, and how to deploy and scale your API quickly and automatically with Amazon API Gateway.

Serverless applications have three main benefits: no server management, flexible scaling, and automated high availability. Without server management, you no longer have to provision or maintain servers. With AWS Lambda, you upload your code, run it, and focus on your application updates. With flexible scaling, you no longer have to disable Amazon EC2 instances to scale them vertically, groups do not need to be auto-scaled, and you do not need to create Amazon CloudWatch alarms to add them to load balancers. With AWS Lambda, you adjust the units of consumption (memory and execution time), and AWS adjusts the rest of the instance appropriately. Finally, serverless applications have built-in availability and fault tolerance. When periods of low traffic occur, you do not spend money on Amazon EC2 instances that do not run at their full capacity.

You can use an Amazon S3 web server to create your presentation tier. Within an Amazon S3 bucket, you can store HTML, CSS, and JavaScript files. JavaScript can create HTTP requests. These HTTP requests are sent to a REST endpoint service called Amazon API Gateway, which allows the application to save and retrieve data dynamically by triggering a Lambda function.

After you create your Amazon S3 bucket, you configure it to use static website hosting in the AWS Management Console and enter an endpoint that reflects your AWS Region.

Amazon S3 allows you to configure web traffic logs to capture information, such as the number of visitors who access your website in the Amazon S3 bucket.

One way to decrease latency and improve your performance is to use Amazon CloudFront with Amazon S3 to move your content closer to your end users. Amazon CloudFront is a serverless service.

The Amazon API Gateway is a fully managed service designed to define, deploy, and maintain APIs. Clients integrate with the APIs using standard HTTPS requests. Amazon API Gateway can integrate with a service-oriented multitier architecture. The Amazon API Gateway provides dynamic data in the logic or app tier.

There are three types of endpoints for Amazon API Gateway: regional endpoints, edge-optimized endpoints, and private endpoints.

In the Amazon API Gateway service, you expose addressable resources as a tree of API Resources entities, with the root resource (/) at the top of the hierarchy. The root resource is relative to the API's base URL, which consists of the API endpoint and a stage name.

You use Amazon API Gateways to help drive down the total response-time latency of your API. Amazon API Gateway uses the HTTP protocol to process these HTTP methods and send/receive data to and from the backend. Serverless data is sent to AWS Lambda to process.

You can use Amazon Route 53 to create a more user-friendly domain name instead of using the default host name (Amazon S3 endpoint). To support two subdomains, you create two Amazon S3 buckets that match your domain name and subdomain.

A stage is a named reference to a deployment, which is a snapshot of the API. Use a stage to manage and optimize a particular deployment. You create stages for each of your environments such as DEV, TEST, and PROD, so you can develop and update your API and applications without affecting production. Use Amazon API Gateway to set up authorizers with Amazon Cognito user pools on an AWS Lambda function. This enables you to secure your APIs.

An Amazon Cognito user pool includes a prebuilt user interface (UI) that you can use inside your application to build a user authentication flow quickly. A user pool is a user directory in Amazon Cognito. With a user pool, your users can sign in to your web or mobile app through Amazon Cognito. Users can also sign in through social identity providers such as Facebook or Amazon and through Security Assertion Markup Language (SAML) identity providers.

Amazon Cognito identity pools allow you to create unique identities and assign permissions for your users to help you integrate with authentication providers. With the combination of user pools and identity pools, you can create a serverless user authentication system.

You can choose how users sign in with a username, an email address, and/or a phone number and to select attributes. Attributes are properties that you want to store about your end users. You can also configure password policies. Multi-factor authentication (MFA) prevents anyone from signing in to a system without authenticating through two different sources, such as a password and a mobile device–generated token. You create an Amazon Cognito role to send Short Message Service (SMS) messages to users.

The AWS Serverless Application Model (AWS SAM) allows you to create and manage resources in your serverless application with AWS CloudFormation as a SAM template. A SAM template is a JSON or YAML file that describes the AWS Lambda function, API endpoints, and other resources. You upload the template to AWS CloudFormation to create a stack. When you update your AWS SAM template, you redeploy the changes to this stack, and AWS CloudFormation updates the resources. You can use AWS SAM to create a template of your serverless infrastructure, which you can then build into a DevOps pipeline.

The `Transform: 'AWS::Serverless-2016-10-31'` code converts the AWS SAM template into an AWS CloudFormation template.

The AWS Serverless Application Repository enables you to deploy code samples, components, and complete applications for common use cases. Each application is packaged with an AWS SAM template that defines the AWS resources.

Additionally, you learned the differences between the standard three-tier web applications and the AWS serverless stack. You learned how to build your infrastructure quickly with AWS SAM and AWS SAM CLI for testing and development purposes.

Exam Essentials

Know serverless applications' three main benefits. The benefits are as follows:

- No server management
- Flexible scaling
- Automated high availability

Know what no server management means. Without server management, you no longer have to provision or maintain servers. With AWS Lambda, you upload your code, run it, and focus on your application updates.

Know what flexible scaling means. With flexible scaling, you no longer have to disable Amazon Elastic Compute Cloud (Amazon EC2) instances to scale them vertically, groups do not need to be auto-scaled, and you do not need to create Amazon CloudWatch alarms to add them to load balancers. With AWS Lambda, you adjust the units of consumption (memory and execution time), and AWS adjusts the rest of the instances appropriately.

Know what serverless applications mean. Serverless applications have built-in availability and fault tolerance. You do not need to architect for these capabilities, as the services that run the application provide them by default. Additionally, when periods of low traffic occur on the web application, you do not spend money on Amazon EC2 instances that do not run at their full capacity.

Know what services are serverless. On the exam, it is important to understand which Amazon services are serverless and which ones are not. The following services are serverless:

- Amazon API Gateway
- AWS Lambda
- Amazon SQS
- Amazon SNS
- Amazon Kinesis
- Amazon Cognito
- Amazon Aurora Serverless
- Amazon S3

Know how to host a serverless web application. Hosting a serverless application means that you need Amazon S3 to host your static website, which comprises your HTML, JavaScript, and CSS files. For your database infrastructure, you can use Amazon DynamoDB or Amazon Aurora Serverless. For your business logic tier, you can use AWS Lambda. For DNS services, you can utilize Amazon Route 53. If you need the ability to host an API, you can use Amazon API Gateway. Finally, if you need to decrease latency to portions of your application, you can utilize services like Amazon CloudFront, which allows you to host your content at the edge.

Resources to Review

Serverless Computing and Applications:

`https://aws.amazon.com/serverless/`

Deleting Objects

Amazon S3 provides several options for deleting objects from your bucket. You can delete one or more objects directly from your S3 bucket. If you want to delete a single object, you can use the Amazon S3 `Delete` API. If you want to delete multiple objects, you can use the Amazon S3 Multi-Object Delete API, which enables you to delete up to 1,000 objects with a single request. The Multi-Object Delete operation requires a single query string `Delete` parameter to distinguish it from other bucket `POST` operations.

When deleting objects from a bucket that is not version-enabled, provide only the object key name. However, when deleting objects from a version-enabled bucket, provide the version ID of the object to delete a specific version of the object.

Deleting Objects from a Version-Enabled Bucket

If your bucket is version-enabled, then multiple versions of the same object can exist in the bucket. When working with version-enabled buckets, the `DELETE` API enables the following options:

Specify a nonversioned delete request Specify only the object's key, not the version ID. In this case, Amazon S3 creates a delete marker and returns its version ID in the response. This makes your object disappear from the bucket.

Specify a versioned delete request Specify both the key and a version ID. In this case, the following two outcomes are possible:

- If the version ID maps to a specific object version, then Amazon S3 deletes the specific version of the object.
- If the version ID maps to the delete marker of that object, Amazon S3 deletes the delete marker. This causes the object to reappear in your bucket.

Performance Optimization

This section discusses Amazon S3 best practices for optimizing performance.

Request Rate and Performance Considerations

Amazon S3 scales to support high request rates. If your request rate grows steadily, Amazon S3 automatically partitions your buckets as needed to support higher request rates. However, if you expect a rapid increase in the request rate for a bucket to more than 300 `PUT/LIST/DELETE` requests per second or more than 800 `GET` requests per second, AWS recommends that you open a support case to prepare for the workload and avoid any temporary limits on your request rate.

If your requests are typically a mix of `GET`, `PUT`, `DELETE`, or `GET` Bucket (List Objects), choose the appropriate key names for your objects to ensure better performance by providing low-latency access to the Amazon S3 index. It also ensures scalability regardless of the number of requests that you send per second. If the bulk of your workload consists of `GET` requests, AWS recommends using the Amazon CloudFront content delivery service.

TIP

The Amazon S3 best practice guidelines apply only if you are routinely processing 100 or more requests per second. If your typical workload involves only occasional bursts of 100 requests per second and fewer than 800 requests per second, you do not need to follow these recommendations.

Workloads with Different Request Types

When you are uploading a large number of objects, you might use sequential numbers or date-and-time values as part of the key names. For instance, you might decide to use key names that include a combination of the date and time, as shown in the following example, where the prefix includes a timestamp:

```
demobucket/2018-31-05-16-00-00/order1234234/receipt1.jpg
demobucket/2018-31-05-16-00-00/order3857422/receipt2.jpg
demobucket/2018-31-05-16-00-00/order1248473/receipt2.jpg
demobucket/2018-31-05-16-00-00/order8474937/receipt2.jpg
demobucket/2018-31-05-16-00-00/order1248473/receipt3.jpg
…
demobucket/2018-31-05-16-00-00/order1248473/receipt4.jpg
demobucket/2018-31-05-16-00-00/order1248473/receipt5.jpg
demobucket/2018-31-05-16-00-00/order1248473/receipt6.jpg
demobucket/2018-31-05-16-00-00/order1248473/receipt7.jpg
```

The way these keys are named presents a performance problem. To get a better understanding of this issue, consider the way that Amazon S3 stores key names. Amazon S3 maintains an index of object key names in each AWS Region. These keys are stored in UTF-8 binary ordering across multiple partitions in the index. The key name dictates where (or which partition) the key is stored in. In this case, where you use a sequential prefix such as a timestamp, or an alphabetical sequence, would increase the chances that Amazon S3 targets a specific partition for a large number of your keys, overwhelming the I/O capacity of the partition. However, if you introduce randomness in your key name prefixes, the key names, and therefore the I/O load, distribute across more than one partition.

If you anticipate that your workload will consistently exceed 100 requests per second, avoid sequential key names. If you must use sequential numbers or date-and-time patterns in key names, add a random prefix to the key name. The randomness of the prefix more evenly distributes key names across multiple index partitions.

The guidelines for the key name prefixes also apply to the bucket names. When Amazon S3 stores a key name in the index, it stores the bucket names as part of the key name (`demobucket/object.jpg`).

One way to introduce randomness to key names is to add a hash string as a prefix to the key name. For instance, you can compute an MD5 hash of the character sequence that you plan to assign as the key name. From the hash, select a specific number of characters and add them as the prefix to the key name.

TIP A hashed prefix of three or four characters should be sufficient. AWS strongly recommends using a hexadecimal hash as the prefix.

GET-Intensive Workloads

If your workload consists mostly of sending `GET` requests, then in addition to the preceding guidelines, consider using Amazon CloudFront for performance optimization.

Integrating CloudFront with Amazon S3 enables you to deliver content with low latency and a high data transfer rate. You will also send fewer direct requests to Amazon S3, which helps to lower your costs.

Suppose that you have a few objects that are popular. CloudFront fetches those objects from Amazon S3 and caches them. CloudFront can then serve future requests for the objects from its cache, reducing the number of `GET` requests it sends to Amazon S3.

Storing Large Attribute Values in Amazon S3

Amazon DynamoDB currently limits the size of each item that you store in a table to 400 KB. If your application needs to store more data in an item than the DynamoDB size limit permits, store the large attributes as an object in Amazon S3. You can then store the Amazon S3 object identifier in your item.

You can also use the object metadata support in Amazon S3 to store the primary key value of the corresponding table item as Amazon S3 object metadata. Doing this provides a link back to the parent item in DynamoDB, which helps with maintenance of the Amazon S3 objects.

Example 11: Store Large Attribute Values in Amazon S3

Suppose that you have a table that stores product data, such as item price, description, book authors, and dimensions for other products. If you want to store an *image* of each product that was too large to fit in an *item*, use Amazon S3 images as an item in DynamoDB.

When implementing this strategy, remember the following limitations and restrictions:

- DynamoDB does not support transactions that cross Amazon S3 and DynamoDB. Therefore, your application must deal with any failures, which could include cleaning up orphaned Amazon S3 objects.
- Amazon S3 limits the length of object identifiers. You must organize your data in a way that does not generate excessively long object identifiers or violate other Amazon S3 constraints.

Optimizing Data Transfer

Optimizing data transfer ensures that you minimize data transfer costs. Review your user presence if global or local and how the data gets located in order to reduce the latency issues.

- Use *Amazon CloudFront*, a global content delivery network (CDN), to locate data closer to users. It caches data at edge locations across the world, which reduces the load on your resources. By using CloudFront, you can reduce the administrative effort in delivering content automatically to large numbers of users globally, with minimum latency. Depending on your application types, distribute your entire website, including dynamic, static, streaming, and interactive content through CloudFront instead of scaling out your infrastructure.
- *Amazon S3 transfer acceleration* enables fast transfer of files over long distances between your client and your S3 bucket. Transfer acceleration leverages Amazon CloudFront globally distributed edge locations to route data over an optimized network path. For a workload in an S3 bucket that has intensive `GET` requests, you should use Amazon S3 with CloudFront.
- When uploading large files, use *multipart uploads* with multiple parts uploading at once to help maximize network throughput. Multipart uploads provide the following advantages:
  - *Improved throughput*—You can upload parts in parallel to improve throughput.
  - *Quick recovery from any network issues*—Smaller part size minimizes the impact of restarting a failed upload due to a network error.
  - *Pause and resume object uploads*—You can upload object parts over time. After you initiate a multipart upload, there is no expiry; you must explicitly complete or abort the multipart upload.
  - *Begin an upload before you know the final object size*—You can upload an object as you are creating it.
- Using *Amazon Route 53*, you can reduce latency for your users by serving their requests from the AWS Region for which network latency is lowest. Amazon Route 53 latency-based routing lets you use Domain Name System (DNS) to route user requests to the AWS Region that will give your users the fastest response.

Caching

Caching improves application performance by storing frequently accessed data items in memory so that they can be retrieved without accessing the primary data store. Cached information might include the results of I/O-intensive database queries or the outcome of computationally intensive processing.

AWS Trusted Advisor

AWS Trusted Advisor inspects your AWS environment and makes recommendations that help to improve the speed and responsiveness of your applications.

The following are a few Trusted Advisor checks to improve the performance of your service. The Trusted Advisor checks your service limits, ensuring that you take advantage of provisioned throughput, and monitors for overutilized instances:

- Amazon EC2 instances that are consistently at high utilization can indicate optimized, steady performance, but this check can also indicate that an application does not have enough resources.
- Provisioned IOPS (SSD) volumes that are attached to an Amazon EC2 instance that is not Amazon EBS–optimized. Amazon EBS volumes are designed to deliver the expected performance only when they are attached to an EBS-optimized instance.
- Amazon EC2 security groups with a large number of rules.
- Amazon EC2 instances that have a large number of security group rules.
- Amazon EBS magnetic volumes (standard) that are potentially overutilized and might benefit from a more efficient configuration.
- CloudFront distributions for alternate domain names with incorrectly configured DNS settings.

  Some HTTP request headers, such as Date or User-Agent, significantly reduce the cache hit ratio. This increases the load on your origin and reduces performance because CloudFront must forward more requests to your origin.

Summary

In this chapter, you learned about the following:

- Cost-optimizing practices
- Right sizing your infrastructure
- Optimizing using Reserved Instances, Spot Instances, and AWS Auto Scaling
- Optimizing storage and data transfer
- Optimizing using NoSQL database (Amazon DynamoDB)
- Monitoring your costs and performance
- Tools, such as AWS Trusted Advisor, Amazon CloudWatch, and AWS Budgets

Achieving an optimized system is a continual process. An optimized system uses all the provisioned resources efficiently and achieves your business goal at the lowest price point. Engineers must know the cost of deploying resources and how to architect for cost optimization. Practice eliminating the waste and bring accountability in every step of the build process. Use mandatory cost tags on all of your resources to gain precise insights into usage. Define IAM policies to enforce tag usage, and use tagging tools, such as AWS Config and AWS Tag Editor, to manage tags. Be cost-conscious, reduce the usage by terminating unused instances, and delete old snapshots and unused keys.

Right size your infrastructure by matching instance types and sizes, and set periodic checks to ensure that the initial provision remains optimum as your business changes over time. With Amazon EC2, you can choose the combination of instance types and sizes most appropriate for your applications. Amazon RDS instances are also optimized for memory, performance, and I/O.

Amazon EC2 Reserved Instances provide you with a significant discount (up to 75 percent) compared to On-Demand Instance pricing. Using Convertible Reserved Instances, you can change instance families, OS types, and tenancies while benefitting from Reserved Instance pricing. Reserved Instance Marketplace allows you to sell the unused Reserved Instances or buy them from other AWS customers, usually at lower prices and shorter terms. With size flexibility, discounted rates for Amazon RDS Reserved Instances are automatically applied to the usage of any size within the instance family.

Spot Instances provide an additional option for obtaining compute capacity at a reduced cost and can be used along with On-Demand and Reserved Instances. Spot Fleets enable you to launch and maintain the target capacity and to request resources automatically to replace any that are disrupted or manually terminated. Using the termination notices and persistent requests in your application design help to maintain continuity as the result of interruptions.

AWS Auto Scaling automatically scales if your application experiences variable load and uses one or more scalable resources, such as Amazon ECS, Amazon DynamoDB, Amazon Aurora, Amazon EC2 Spot requests, and Amazon EC2 scaling groups. Predictive Scaling uses machine learning models to forecast daily and weekly patterns. Amazon EC2 Auto Scaling enables you to scale in response to demand and known load schedules. It supports the provisioning of scale instances across purchase options, Availability Zones, and instance families to optimize performance and cost.

Containers provide process isolation and improve the resource utilization. Amazon ECS lets you easily build all types of containerized applications and launch thousands of containers in seconds with no additional complexity. With AWS Fargate technology, you can manage containers without having to provision or manage servers. It enables you to focus on building and running applications, not the underlying infrastructure.

AWS Lambda takes care of receiving events or client invocations and then instantiates and runs the code. That means there's no need to manage servers. Serverless services have built-in automatic scaling, availability, and fault tolerance. These features allow you to focus on product innovation and rapidly construct applications, such as web applications, websites, web-hooked systems, chatbots, and clickstream.

AWS storage services are optimized to meet different storage requirements. Use the Amazon S3 analytics feature to analyze storage access patterns to help you decide when to transition the right data to the right storage class and to yield considerable savings. Monitor Amazon EBS volumes periodically to identify ones that are unattached or appear to be underutilized or overutilized, and adjust provisioning to match actual usage.

Optimizing data transfer ensures that you minimize data transfer costs. Use options such as Amazon CloudFront, Amazon S3 transfer acceleration, and Amazon Route 53 to let data reach Regions faster and reduce latency issues.

NoSQL database systems like Amazon DynamoDB use alternative models for data management, such as key-value pairs or document storage. DynamoDB enables you to offload the administrative burdens of operating and scaling a distributed database so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling. Follow best practices, such as distributing data evenly, effective partition and sort keys usage, efficient data scanning, and using sparse indexes for maximizing performance and minimizing throughput costs, when working with Amazon DynamoDB.

AWS provides several tools to help you identify those cost-saving opportunities and keep your resources right-sized. AWS Trusted Advisor inspects your AWS environment to identify idle and underutilized resources and provides real-time insight into service usage to help you improve system performance and save money. Amazon CloudWatch collects and tracks metrics, monitors log files, sets alarms, and reacts to changes in AWS resources automatically. AWS Cost Explorer checks patterns in AWS spend over time, projects future costs, identifies areas that need further inquiry, and provides Reserved Instance recommendations.

Optimization is an ongoing process. Always stay current with the pace of AWS new releases, and assess your existing design solutions to ensure that they remain cost-effective.

Exam Essentials

Know the importance of tagging. By using tags, you can assign metadata to AWS resources. This tagging makes it easier to manage, search for, and filter resources in billing reports and automation activities and when setting up access controls.

Know about various tagging tools and how to enforce the tag rules. With AWS Tag Editor, you can add tags to multiple resources at once, search for the resources that you want to tag, and then add, remove, or edit tags for the resources in your search results. AWS Config identifies resources that do not comply with tagging policies. You can use IAM policy conditions to force the usage of tags while creating the resources.

Know the fundamental practices in reducing the usage. Follow the best practices of cost optimization in every step of your build process, such as turning off unused resources, spinning up instances only when needed, and spinning them down when not in use. Use tagging to help with the cost allocation. Use Amazon EC2 Spot Instances, Amazon EC2, and Reserved Instances where appropriate, and use alerts, notifications, and cost-management tools to stay on track.

Know the various usage patterns for right sizing. By understanding your business use case and backing up the analysis with performance metrics, you can choose the most appropriate options, such as steady state; variable; predictable, but temporary; and development, test, and production usage.

Know the various instance families for right sizing and the corresponding use cases. Amazon EC2 provides a wide selection of instances to match capacity needs at the lowest cost and comes with different options for CPU, memory, and network resources. The families include General Purpose, Compute Optimized, Memory Optimized, Storage Optimized, and Accelerated Computing.

Know Amazon EC2 Auto Scaling benefits and how this feature can make your solutions more optimized and highly available. AWS Auto Scaling is a fast, easy way to optimize the performance and costs of your applications. It makes smart scaling decisions based on your preferences, automatically maintains performance even when your workloads are periodic, unpredictable, and continuously changing.

Know how to create a single AWS Auto Scaling group to scale instances across different purchase options. You can provision and automatically scale Amazon EC2 capacity across different Amazon EC2 instance types, Availability Zones, and On-Demand, Reserved Instances, and Spot purchase options in a single AWS Auto Scaling group. You can define the desired split between On-Demand and Spot capacity, select which instance types work for your application, and specify preferences for how Amazon EC2 Auto Scaling should distribute the AWS Auto Scaling group capacity within each purchasing model.

Know how block, object, and file storages are different. Block storage is commonly dedicated, low-latency storage for each host, and it is provisioned with each instance. Object storage is developed for the cloud, has vast scalability, is accessed over the web, and is not directly attached to an instance. File storage enables accessing shared files as a file system.

Know key CloudWatch metrics available to measure the Amazon EBS efficiency and how to use them. CloudWatch metrics are statistical data that you can use to view, analyze, and set alarms on the operational behavior of your volumes. Depending on your needs, set alarms and response actions that correspond to each data point. For example, if your I/O latency is higher than you require, check the metric `VolumeQueueLength` to make sure that your application is not trying to drive more IOPS than you have provisioned. Review and learn more about the available metrics that help optimize the block storage.

Know tools and features that help in efficient data transfer. Using Amazon CloudFront, you can locate data closer to users and reduce administrative efforts to minimize data transfer costs. Amazon S3 Transfer Acceleration enables fast data transfer over an optimized network path. Use the multipart upload file option while uploading a large file to improve network throughput.

Know key differences between RDBMS and NoSQL databases to design efficient solutions using Amazon DynamoDB. Schema flexibility and the ability to store related items together make DynamoDB a solution for solving problems associated with changing business needs and scalability issues unlike relational databases.

Know the importance of distributing the data evenly when designing DynamoDB tables. Use provisioned throughput more efficiently by making the partition key more distinct. That way, data spreads throughout the provisioned space. Use the sort key with the