# AWS®

# Certified Developer

## Official Study Guide

### Associate (DVA-C01) Exam

This chapter covers how to provision storage using just-in-time purchasing, which helps you avoid overprovisioning and paying for unused storage into which you expect to grow eventually.

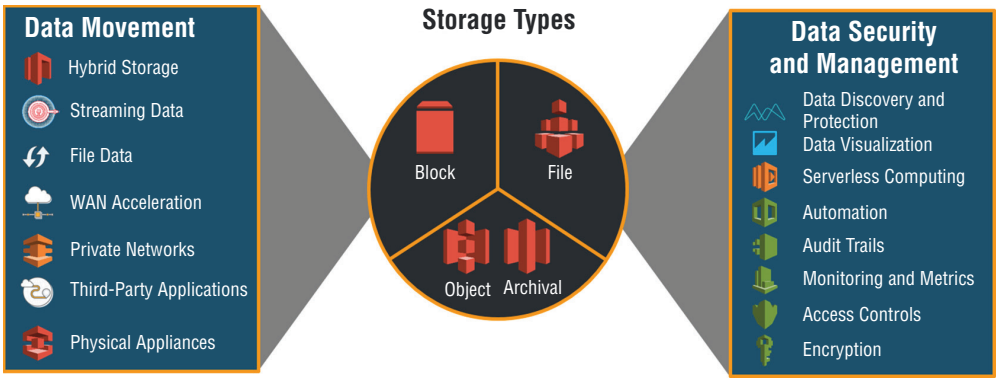
# Storage Fundamentals

Before we explore the various AWS storage services, let's review a few storage fundamentals. As a developer, you are likely already familiar with block storage and the differences between hot and cold storage. Cloud storage introduces some new concepts such as object storage, and we will compare these new concepts with the traditional storage concepts with which you are already familiar. If you have been working on the cloud already, these fundamentals are likely a refresher for you.

The goal of this chapter is to produce a mental model that will allow you, as a developer, to make the right decisions for choosing and implementing the best storage options for your applications. With the right mental model, people can usually make the best decisions for their solutions.

The AWS storage portfolio mental model starts with the core data building blocks, which include block, file, and object storage. For block storage, AWS has *Amazon Elastic Block Store* (Amazon EBS). For file storage, AWS has *Amazon Elastic File System* (Amazon EFS). For object storage, AWS has *Amazon Simple Storage Service* (Amazon S3) and Amazon S3 Glacier. Figure 3.2 illustrates this set of storage building blocks.

**FIGURE 3.2** A complete set of storage building blocks



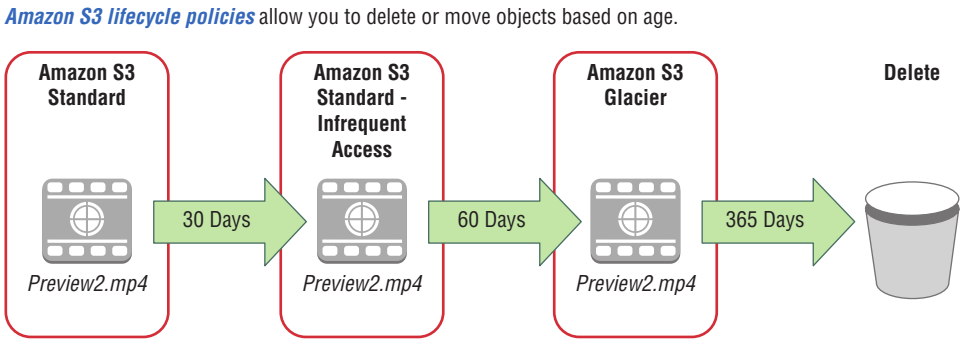

## Data Dimensions

When investigating which storage options to use for your applications, consider the different dimensions of your data first. In other words, find the right tool for your data instead of squeezing your data into a tool that might not be the best fit.

```
    <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




# 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


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.

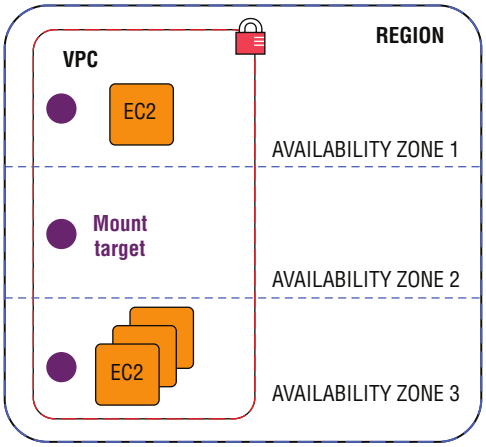
### Mount Target

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

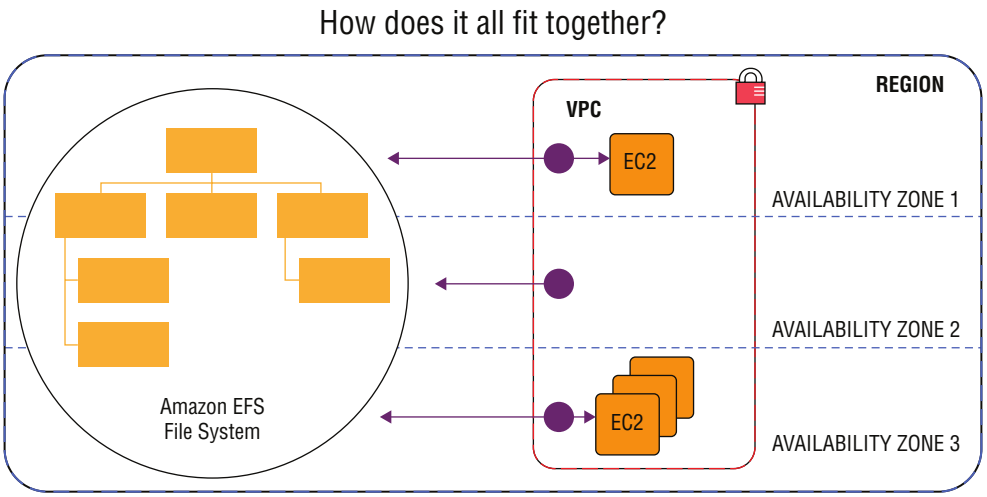
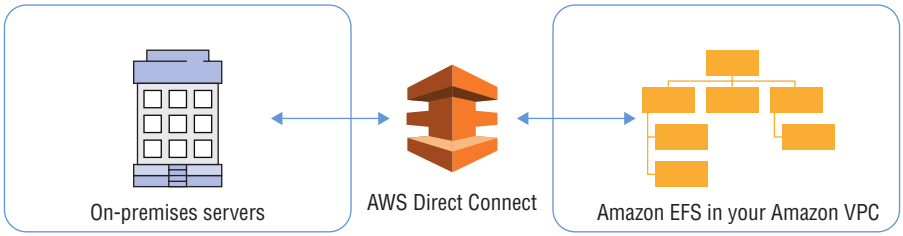



**FIGURE 3.22** Using AWS Direct Connect with Amazon EFS




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

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 |

# Amazon Elastic File System

*Amazon Elastic File System* (Amazon EFS) provides simple, scalable file storage for use with Amazon EC2. With Amazon EFS, storage capacity is elastic, growing and shrinking automatically as you add and remove files so your applications have the storage they need when they need it. The simple web services interface helps you create and configure file systems quickly and easily. The service manages all of the file storage infrastructure, meaning that you can avoid the complexity of deploying, patching, and maintaining complex file system configurations, such as situations where it must facilitate user uploads or interim results of batch processes. By placing those files in a shared storage layer, it helps you to avoid the introduction of stateful components.

Amazon EFS supports the Network File System versions 4.0 and 4.1 (NFSv4) protocol, so the applications and tools that you use today work seamlessly with Amazon EFS. Multiple Amazon EC2 instances can access an Amazon EFS file system at the same time, providing a common data source for workloads and applications running on more than one instance or server.

The service is highly scalable, highly available, and highly durable. Amazon EFS stores data and metadata across multiple Availability Zones in a region, and it can grow to petabyte scale, drive high levels of throughput, and allow massively parallel access from Amazon EC2 instances to your data.

Amazon EFS provides file system access semantics, such as strong data consistency and file locking. Amazon EFS also allows you to control access to your file systems through Portable Operating System Interface (POSIX) permissions.

Amazon EFS supports two forms of encryption for file systems: encryption in transit and encryption at rest. You can enable encryption at rest when creating an Amazon EFS file system. If you do, all of your data and metadata is encrypted. You can enable encryption in transit when you mount the file system.

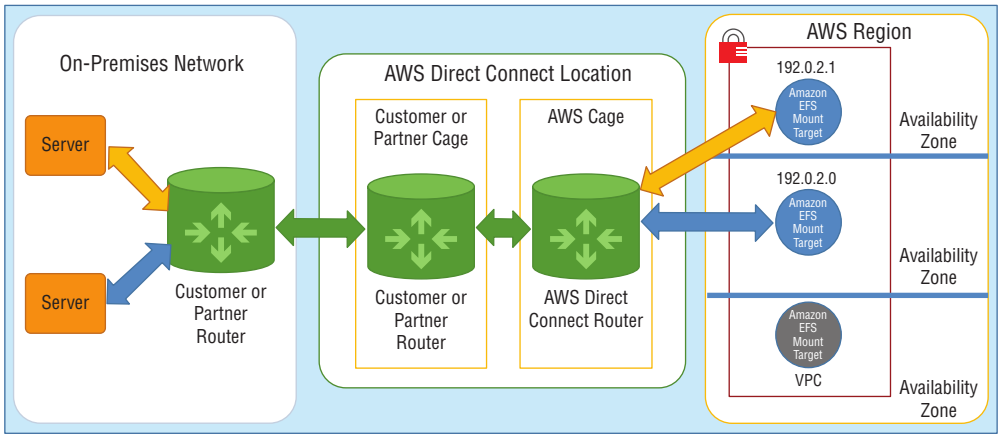
Amazon EFS is designed to provide the throughput, input/output operations per second (IOPS), and low latency needed for a broad range of workloads. With Amazon EFS, throughput and IOPS scale as a file system grows, and file operations are delivered with consistent, low latencies.

## How Amazon EFS Works

Figure 14.19 shows an example of VPC accessing an Amazon EFS file system. In this example, Amazon EC2 instances in the VPC have file systems mounted.

Amazon EFS provides file storage in the AWS Cloud. With Amazon EFS, you can create a file system, mount the file system on an Amazon EC2 instance, and then read and write data to and from your file system. You can mount an Amazon EFS file system in your VPC through the NFSv4 protocol.

**FIGURE 14.19** VPC accessing an Amazon EFS




You can access your Amazon EFS file system concurrently from Amazon EC2 instances in your Amazon VPC so that applications that scale beyond a single connection can access a file system. Amazon EC2 instances running in multiple Availability Zones within the same region can access the file system so that many users can access and share a common data source.

However, there are restrictions. You can mount an Amazon EFS file system on instances in only one VPC at a time. Both the file system and VPC must be in the same AWS Region.

To access your Amazon EFS file system in a VPC, create one or more mount targets in the VPC. A *mount target* provides an IP address for an NFSv4 endpoint at which you can mount an Amazon EFS file system. Mount your file system using its DNS name, which resolves to the IP address of the Amazon EFS mount target in the same Availability Zone as your EC2 instance. You can create one mount target in each Availability Zone in a region. If there are multiple subnets in an Availability Zone in your Amazon VPC, create a mount target in one of the subnets, and all EC2 instances in that Availability Zone share that mount target.

Mount targets themselves are designed to be highly available. When designing your application for both high availability and the ability to failover to other Availability Zones, keep in mind that the IP addresses and DNS for your mount targets in each Availability Zone are static. After mounting the file system via the mount target, use it like any other POSIX-compliant file system.

You can mount your Amazon EFS file systems on your on-premises data center servers when connected to your Amazon VPC with AWS Direct Connect (DX). You can mount your Amazon EFS file systems on on-premises servers to migrate datasets to Amazon EFS, enable cloud-bursting scenarios, or back up your on-premises data to Amazon EFS.

You can mount Amazon EFS file systems on Amazon EC2 instances or on-premises through a DX connection.

## How Amazon EFS Works with AWS Direct Connect

Using an Amazon EFS file system mounted on an on-premises server, you can migrate on-premises data into the AWS Cloud hosted in an Amazon EFS file system. You can also take advantage of bursting. This means that you can move data from your on-premises servers into Amazon EFS, analyze it on a fleet of Amazon EC2 instances in your Amazon VPC, and then store the results permanently in your file system or move the results back to your on-premises server.

Consider the following when using Amazon EFS with DX:

- Your on-premises server must have a Linux-based operating system. AWS recommends Linux kernel version 4.0 or later.
- For the sake of simplicity, AWS recommends mounting an Amazon EFS file system on an on-premises server using a mount target IP address instead of a DNS name.
- AWS Virtual Private Network (AWS VPN) is not supported for accessing an Amazon EFS file system from an on-premises server.

You can use any one of the mount targets in your Amazon VPC as long as the subnet of the mount target is reachable by using the DX connection between your on-premises server and Amazon VPC. To access Amazon EFS from an on-premises server, you must add a rule to your mount target security group to allow inbound traffic to the NFS port (2049) from your on-premises server.

In Amazon EFS, a file system is the primary resource. Each file system has properties such as ID, creation token, creation time, file system size in bytes, number of mount targets created for the file system, and the file system state.

Amazon EFS also supports other resources to configure the primary resource. These include mount targets and tags:

### Mount Target

- To access your file system, create mount targets in your Amazon VPC. Each mount target has the following properties:
  - Mount target ID
  - Subnet ID where it is created
  - File system ID for which it is created
  - IP address at which the file system may be mounted
  - Mount target state

  You can use the IP address or the DNS name in your mount command.

### Tags

- To help organize your file systems, assign your own metadata to each of the file systems that you create. Each tag is a key-value pair.

Each file system has a DNS name of the following form:

```
file-system-ID.efs.aws-region.amazonaws.com
```

You can configure this DNS name in your `mount` command to mount the Amazon EFS file system.

Suppose that you create an `efs-mount-point` subdirectory in your home directory on your EC2 instance or on-premises server. Use the `mount` command to mount the file system. For example, on an Amazon Linux AMI, you can use following `mount` command:

```
$ sudo mount -t nfs -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,
retrans=2,noresvport file-system-DNS-name:/ ~/efs-mount-point
```

You can think of mount targets and tags as *subresources* that do not exist without being associated with a file system.

Amazon EFS provides API operations for you to create and manage these resources. In addition to the `create` and `delete` operations for each resource, Amazon EFS also supports a `describe` operation that enables you to retrieve resource information. The following options are available for creating and managing these resources:

- Use the Amazon EFS console.
- Use the Amazon EFS command line interface (CLI).

You can also manage these resources programmatically as follows:

**Use the AWS SDKs** The AWS SDKs simplify your programming tasks by wrapping the underlying Amazon EFS API. The SDK clients also authenticate your requests by using access keys that you provide.

**Call the Amazon EFS API directly from your application** If you cannot use the SDKs, you can make the Amazon EFS API calls directly from your application. However, if you use this option, you must write the necessary code to authenticate your requests.

## Authentication and Access Control

You must have valid credentials to make Amazon EFS API requests, such as creating a file system. In addition, you must also have permissions to create or access resources. By default, when you use the account root user credentials, you can create and access resources owned by that account. However, AWS does not recommend using account root user credentials. In addition to creating or accessing resources, you must grant permissions to any AWS IAM users and roles you create in your account.

## Data Consistency in Amazon EFS

Amazon EFS provides the open-after-close consistency semantics that applications expect from NFS. In Amazon EFS, write operations are durably stored across Availability Zones when an application performs a synchronous write operation (for example, using the `open` Linux command with the `O_DIRECT` flag, or the `fsync` Linux command) and when an application closes a file.

Amazon EFS provides stronger consistency than open-after-close semantics, depending on the access pattern. Applications that perform synchronous data access and perform non-appending writes have read-after-write consistency for data access.

## Creating an IAM User

Services in AWS, such as Amazon EFS, require that you provide credentials when you access them so that the service can determine whether you have permissions to access its resources. AWS recommends that you do not use the AWS account credentials of your account to make requests. Instead, create an IAM user, and grant that user full access. AWS refers to these users as administrators. You can use the administrator credentials, instead of AWS account credentials, to interact with AWS and perform tasks, such as creating a bucket, creating users, and granting them permissions.

For all operations, such as creating a file system and creating tags, a user must have IAM permissions for the corresponding API action and resource. You can perform any Amazon EFS operations using the AWS account credentials of your account. However, using AWS account credentials is not considered a best practice. If you create IAM users in your account, you can give them permissions for Amazon EFS actions with user policies. Additionally, you can use roles to grant cross-account permissions.

## Creating Resources for Amazon EFS

Amazon EFS provides elastic, shared file storage that is POSIX-compliant. The file system that you create supports concurrent read and write access from multiple Amazon EC2 instances, and it is accessible from all of the Availability Zones in the AWS Region where it is created. You can mount an Amazon EFS file system on EC2 instances in your Amazon VPC through the Network File System versions 4.0 and 4.1 protocol (NFSv4).

As an example, suppose that you have one or more EC2 instances launched in your Amazon VPC. You want to create and use a file system on these instances. To use Amazon EFS file systems in the VPC, follow these steps:

1. **Create an Amazon EFS file system**. When creating a file system, AWS recommends that you consider using the Name tag, because its value appears in the console, making the file easier to identify. You can also add other optional tags to the file system.
2. **Create mount targets for the file system**. To access the file system in your Amazon VPC and mount the file system to your Amazon EC2 instance, you must create mount targets in the VPC subnets.
3. **Create security groups**. Both an Amazon EC2 instance and mount target must have associated security groups. These security groups act as a virtual firewall that controls the traffic between them. You can use the security group that you associated with the mount target to control inbound traffic to your file system by adding an inbound rule to the mount target security group that allows access from a specific EC2 instance. Then you can mount the file system only on that EC2 instance.

## Creating a File System

You can use the Amazon EFS console, or the AWS CLI, to create a file system. You can also use the AWS SDKs to create file systems programmatically.

## Using File Systems

Amazon EFS presents a standard file system interface that supports semantics for full file system access. Using NFSv4.1, you can mount your Amazon EFS file system on any Amazon EC2 Linux-based instance. Once mounted, you can work with the files and directories as you would with a local file system. You can also use AWS DataSync to copy files from any file system to Amazon EFS.

After you create the file system and mount it on your EC2 instance, you should be aware of several rules to use it effectively. For example, when you first create the file system, there is only *one root directory* at `/`. By default, only the *root user* (`UID 0`) has read-write-execute permissions. For other users to modify the file system, the root user must explicitly grant them access.

### NFS-Level Users, Groups, and Permissions

Amazon EFS file system objects have a Unix-style mode associated with them. This value defines the permissions for performing actions on that object, and users familiar with Unix-style systems can understand how Amazon EFS manages these permissions.

Further, on Unix-style systems, users and groups are mapped to numeric identifiers, which Amazon EFS uses to represent file ownership. A single owner and a single group own file system objects, such as files or directories on Amazon EFS. Amazon EFS uses these numeric IDs to check permissions when a user attempts to access a file system object.

### User and Group ID Permissions on Files and Directories within a File System

Files and directories in an Amazon EFS file system support standard Unix-style `read/write/execute` permissions based on the user ID and group ID asserted by the mounting NFSv4.1 client. When a user tries to access files and directories, Amazon EFS checks their user ID and group IDs to determine whether the user has permission to access the objects. Amazon EFS also uses these IDs as the owner and group owner for new files and directories that the user creates. Amazon EFS does not inspect user or group names—it uses only the numeric identifiers.

When you create a user on an EC2 instance, you can assign any numeric UID and GID to the user. The numeric user IDs are set in the `/etc/passwd` file on Linux systems. The numeric group IDs are in the `/etc/group` file. These files define the mappings between names and IDs. Outside of the EC2 instance, Amazon EFS does not perform any authentication of these IDs, including the root ID of `0`.

If a user accesses an Amazon EFS file system from two different EC2 instances, depending on whether the UID for the user is the same or different on those instances, you may observe different behavior. If the user IDs are the same on both EC2 instances, Amazon EFS considers them the same user, regardless of the EC2 instance they use. The user experience when accessing the file system is the same from both EC2 instances. If the user IDs are not the same on both EC2 instances, Amazon EFS considers them to be different users, and the user experience will not be the same when accessing the Amazon EFS file system from

the two different EC2 instances. If two different users on different EC2 instances share an ID, Amazon EFS considers them the same user.

## Deleting an Amazon EFS File System

File system deletion is a permanent action that destroys the file system and any data in it. Any data that you delete from a file system is gone, and you cannot restore the data.


Always unmount a file system before you delete it.

## Managing Access to Encrypted File Systems

Using Amazon EFS, you can create encrypted file systems. Amazon EFS supports two forms of encryption for file systems: encryption in transit and encryption at rest. Any key management that you must perform is related only to encryption at rest. Amazon EFS automatically manages the keys for encryption in transit. If you create a file system that uses encryption at rest, data and metadata are encrypted at rest.

Amazon EFS uses AWS KMS for key management. When you create a file system using encryption at rest, specify a customer master key (CMK). The CMK can be `aws/elasticfilesystem` (the AWS managed CMK for Amazon EFS), or it can be a CMK that you manage. File data, the contents of your files, is encrypted at rest using the CMK that you specified when you created your file system.

The AWS managed CMK for your file system is used as the master key for the metadata in your file system; for instance, file names, directory names, and directory contents. You are responsible for the CMK used to encrypt your file data (the contents of your files) at rest. Moreover, you are responsible for the management of who has access to your CMKs and the contents of your encrypted file systems. IAM policies and AWS KMS control these permissions. IAM policies control a user's access to Amazon EFS API actions. AWS KMS key policies control a user's access to the CMK that you specified when the file system was created.

As a key administrator, you can both import external keys and modify keys by enabling, disabling, or deleting them. The state of the CMK that you specified (when you created the file system with encryption at rest) affects access to its contents. To provide users access to the contents of an encrypted at rest file system, the CMK must be in the enabled state.

## Amazon EFS Performance

Amazon EFS file systems are spread across an unconstrained number of storage servers, allowing file systems to expand elastically to petabyte scale. The distribution also allows them to support massively parallel access from Amazon EC2 instances to your data. Because of this distributed design, Amazon EFS avoids the bottlenecks and limitations inherent to conventional file servers.

This distributed data storage design means that multithreaded applications and applications that concurrently access data from multiple Amazon EC2 instances can drive substantial levels of aggregate throughput and IOPS. Analytics and big data workloads, media processing workflows, content management, and web serving are examples of these applications.

Additionally, Amazon EFS data is distributed across multiple Availability Zones, providing a high level of availability and durability.

## Performance Modes

To support a wide variety of cloud storage workloads, Amazon EFS offers two performance modes: General Purpose and Max I/O. At the time that you create your file system, you select a file system's performance mode. There are no additional charges associated with the two performance modes. Your Amazon EFS file system is billed and metered the same, irrespective of the performance mode chosen. You cannot change an Amazon EFS file system's performance mode after you have created the file system.

**General Purpose performance mode** AWS recommends the General Purpose performance mode for the majority of your Amazon EFS file systems. General Purpose is ideal for latency-sensitive use cases, such as web serving environments, content management systems, home directories, and general file serving. If you do not choose a performance mode when you create your file system, Amazon EFS selects the General Purpose mode for you by default.

**Max I/O performance mode** File systems in the Max I/O mode can scale to higher levels of aggregate throughput and operations per second with a trade-off of slightly higher latencies for file operations. Highly parallelized applications and workloads, such as big data analysis, media processing, and genomics analysis can benefit from this mode.

## Throughput Scaling in Amazon EFS

Throughput on Amazon EFS scales as a file system grows. Because file-based workloads are typically spiky, driving high levels of throughput for short periods of time and low levels of throughput the rest of the time, Amazon EFS is designed to burst to high throughput levels for periods of time.

All file systems, regardless of size, can burst to 100 MB/s of throughput, and those larger than 1 TB can burst to 100 MB/s per TB of data stored in the file system. For example, a 10-TB file system can burst to 1,000 MB/s of throughput (10 TB × 100 MB/s/TB). The portion of time a file system can burst is determined by its size, and the bursting model is designed so that typical file system workloads will be able to burst virtually any time they need to.

Amazon EFS uses a credit system to determine when file systems can burst. Each file system earns credits over time at a baseline rate that is determined by the size of the file system, and it uses credits whenever it reads or writes data. The baseline rate is 50 MB/s per TB of storage (equivalently, 50 KB/s per GB of storage).

Accumulated burst credits give the file system permission to drive throughput above its baseline rate. A file system can drive throughput continuously at its baseline rate. Whenever the file system is inactive or when it is driving throughput below its baseline rate, the file system accumulates burst credits.

# Summary

In this chapter, stateless applications are defined as those that do not require knowledge of previous individual interactions and do not store session information locally. Stateless application design is beneficial because it reduces the risk of loss of session information or critical data. It also improves user experience by reducing the chances that context-specific data is lost if a resource containing session information becomes unavailable. To accomplish this, AWS customers can use Amazon DynamoDB, Amazon ElastiCache, Amazon Simple Storage Service (Amazon S3), and Amazon Elastic File System (Amazon EFS).

DynamoDB is a fast and flexible NoSQL database service that is used by applications that require consistent, single-digit millisecond latency at any scale. In stateless application design, you can use DynamoDB to store and rapidly retrieve session information. This separates session information from application resources responsible for processing user interactions. For example, a web application can use DynamoDB to store user shopping carts. If an application server becomes unavailable, the users accessing the application do not experience a loss of service.

To further improve speed of access, DynamoDB supports global secondary indexes and local secondary indexes. A secondary index contains a subset of attributes from a table and uses an alternate key to support custom queries. A local secondary index has the same partition key as a table but uses a different sort key. A global secondary index has different partition and sort keys.

DynamoDB uses read and write capacity units to determine cost. A single read capacity unit represents one strongly consistent read per second (or two eventually consistent reads per second) for items up to 4 KB in size. A single write capacity unit represents one write per second for items up to 1 KB in size. Items larger than these values consume additional read or write capacity.

ElastiCache enables you to quickly deploy, manage, and scale distributed in-memory cache environments. With ElastiCache, you can store application state information in a shared location by using an in-memory key-value store. Caches can be created using either Memcached or Redis caching engines. Read and write operations to a back-end database can be time-consuming. Thus, ElastiCache is especially effective as a caching layer for heavy-use applications that require rapid access to backend data. You can also use ElastiCache to store HTTP sessions, further improving the performance of your applications.

ElastiCache offers various scalability configurations that improve access times and availability. For example, read-heavy applications can use additional cache cluster nodes to respond to queries. Should there be an increase in demand, additional cluster nodes can be scaled out quickly.

There are some differences between the available caching engines. AWS recommends that you use Memcached for simple data models that may require scaling and partitioning/sharding. Redis is recommended for more complex data types, persistent key stores, read-replication, and publish/subscribe operations.

In certain situations, storing state information can involve larger file operations (such as file uploads and batch processes). Amazon S3 can support millions of operations per second on trillions of objects through a simple web service. Through simple integration, developers can take advantage of the massive scale of object storage.

Amazon S3 stores objects in buckets, which are addressable using unique URLs (such as `http://johnstiles.s3.amazonaws.com/`). Buckets enable you to group similar objects and configure access control policies for internal and external users. Buckets also serve as the unit of aggregation for usage reporting. There is no limit to the number of objects that can be stored in a bucket, and there is no performance difference between using one or multiple buckets for your web application. The decision to use one or more buckets is often a consideration of access control.

Amazon S3 buckets support versioning and lifecycle configurations to maintain the integrity of objects and reduce cost. Versioning ensures that any time an object is modified and uploaded to a bucket, it is saved as a new version. Authorized users can access previous versions of objects at any time. In versioned buckets, a `delete` operation places a marker on the object (without deleting prior versions). Conversely, you must use a separate operation to fully remove an object from a versioned bucket. Use lifecycle configurations to reduce cost by automatically moving infrequently accessed objects to lower-cost storage tiers.

Amazon EFS provides simple, scalable file storage for use with multiple concurrent Amazon EC2 instances or on-premises systems. In stateless design, having a shared block storage system removes the risk of loss of data in situations where one or more instances become unavailable.

# Exam Essentials

**Understand block storage vs. object storage.** The difference between block storage and object storage is the fundamental unit of storage. With block storage, each file saved to the drive is composed of blocks of a specific size. With object storage, each file is saved as a single object regardless of size.

**Understand when to use Amazon Simple Storage Service and when to use Amazon Elastic Block Storage or Amazon Elastic File System.** This is an architectural decision based on the type of data that you are storing and the rate at which you intend to update that data. Amazon Simple Storage Service (Amazon S3) can hold any type of data, but Amazon S3 would not be a good choice for a database or any rapidly changing data types.

**Understand Amazon S3 versioning.** Once Amazon S3 versioning is enabled, you cannot disable the feature—you can only suspend it. Also, when versioning is activated, items that are deleted are assigned a delete marker and are not accessible. The deleted objects are still in Amazon S3, and you will continue to incur charges for storing them.

**Know how to control access to Amazon S3 objects.** IAM policies specify which actions are allowed or denied on specific AWS resources. Amazon S3 bucket policies are attached only to Amazon S3 buckets. Amazon S3 bucket policies specify which actions are allowed or denied for principals on the bucket to which the bucket policy is attached.

**Know how to create or select a proper primary key for an Amazon DynamoDB table.** DynamoDB stores data as groups of attributes, known as *items*. Items are similar to rows or records in other database systems. DynamoDB stores and retrieves each item based on the primary key value, which must be unique. Items are distributed across 10 GB storage units, called *partitions* (physical storage internal to DynamoDB). Each table has one or more partitions. DynamoDB uses the partition key value as an input to an internal hash function. The output from the hash function determines the partition in which the item is stored. The hash value of its partition key determines the location of each item. All items with the same partition key are stored together. Composite partition keys are ordered by the sort key value. If the collection size grows bigger than
10 GB, DynamoDB splits partitions by sort key.

**Understand how to configure the read capacity units and write capacity units properly for your tables.** When you create a table or index in DynamoDB, you must specify your capacity requirements for read and write activity. By defining your throughput capacity in advance, DynamoDB can reserve the necessary resources to meet the read and write activity your application requires while ensuring consistent, low-latency performance.

One read capacity unit (RCU) represents one strongly consistent read per second, or two eventually consistent reads per second, for an item up to 4 KB in size. If you need to read an item that is larger than 4 KB, DynamoDB must consume additional RCUs. The total number of RCUs required depends on the item size and whether you want an eventually consistent or strongly consistent read.

One write capacity unit (WCU) represents one write per second for an item up to 1 KB in size. If you need to write an item that is larger than 1 KB, DynamoDB must consume additional WCUs. The total number of WCUs required depends on the item size.

**Understand the use cases for DynamoDB streams.** A DynamoDB stream is an ordered flow of information about changes to items in an DynamoDB table. When you enable a stream on a table, DynamoDB captures information about every modification to data items in the table. Whenever an application creates, updates, or deletes items in the table, DynamoDB Streams writes a stream record with the primary key attributes of the items that were modified. A stream record contains information about a data modification to a single item in a DynamoDB table. You can configure the stream so that the stream records capture additional information, such as the before and after images of modified items.

**Know what secondary indexes are and when to use a local secondary index versus a global secondary index and the differences between the two.** A global secondary index is an index with a partition key and a sort key that can be different from those on the base table.

A global secondary index is considered *global* because queries on the index can span all of the data in the base table, across all partitions. A local secondary index is an index that has the same partition key as the base table, but a different sort key. A local secondary index is *local* in the sense that every partition of a local secondary index is scoped to a base table partition that has the same partition key value.

**Know the operations that can be performed using the DynamoDB API.** Know the more common DynamoDB API operations: `CreateTable`, `UpdateTable`, `Query`, `Scan`, `PutItem`, `GetItem`, `UpdateItem`, `DeleteItem`, `BatchGetItem`, and `BatchWriteItem`. Understand the purpose of each operation and be familiar with some of the parameters and limitations for the batch operations.

**Be familiar with handling errors when using DynamoDB.** Understand the differences between 400 error codes and 500 error codes and how to handle both classes of errors. Also, understand which techniques to use to mitigate the different errors. In addition, you should understand what causes a `ProvisionedThrouphputExceededException` error and what you can do to resolve the issue.

**Understand how to configure your Amazon S3 bucket to serve as a static website.** To host a static website, you configure an Amazon S3 bucket for website hosting and then upload your website content to the bucket. This bucket must have public read access. It is intentional that everyone has read access to this bucket. The website is then available at the AWS Region specific website endpoint of the bucket.

**Be familiar with the Amazon S3 API operations.** Be familiar with the API operations, such as `PUT`, `GET`, `SELECT`, and `DELETE`. Understand how having versioning enabled affects the behavior of the `DELETE` operation. You should also be familiar with the situations that require a multipart upload and how to use the associated API.

**Understand the differences among the different Amazon S3 storage classes.** The storage classes are Standard, Infrequent Access (IA), Glacier, and Reduced Redundancy. Understand the differences and why you might choose one storage class over the other and knowing the consequences of those choices.

**Know how to use Amazon ElastiCache.** Improve the performance of your application by deploying ElastiCache clusters as a part of your application and offloading read requests for frequently accessed data. Use the lazy loading caching strategy in your solution to first check the cache for your query results before checking the database.

**Understand when to choose one specific cache engine over another.** ElastiCache provides two open source caching engines. You are responsible for choosing the engine that meets your requirements. Use Redis when you must persist and restore your data, you need multiple replicas of your data, or you are seeking advanced features and functionality, such as sort and rank or leaderboards. Redis supports these features natively. Alternatively, you can use Memcached when you need a simpler, in-memory object store that can be easily partitioned and horizontally scaled.