AWS Solutions Architect Associate Notes

EC2

What is Amazon EC2

Amazon EC2 lets you run servers in the cloud. Before the cloud you would have a physical server, now you can just rent one from Amazon.

AMI

When you launch an instance you select an Amazon Machine Image (AMI) which packages up an operating system and any additional software you will need for your server. Below is an image of part of the selection menu, see that we can select a Linux or Windows instance type.

If you start with a basic AMI and customize it for your needs, you can take that EC2 instance and make an AMI from it so you don't have to do that work all over again.

Your AMI can only be used by instances in the region it lives in (AMIs are stored in S3). You can easily copy an AMI to another region by right clicking on it.

AMI docs

Instance IP Addressing

There are two types of IP addresses used by AWS, IPv4 and IPv6. We focus on IPv4. There are private IPv4 addresses which can communicate with your machine within the AWS infrastructure. There are public IP addresses which can communicate with your machine over the web.

Whenever you stop your machine AWS disassociates the public IPv4 address. When you start it again, you will receive a new address. You shouldn't rely on your instance having an unchanging public IPv4 address. If you really need a constant IPv4 address you can use an **elastic IP**, which associates a constant public IPv4 address with your instance. **Private IPv4 addresses are always constant.**

Instance IP Addressing Docs

User data scripts

When you launch an instance in Amazon EC2, you have the option of passing a **user data** script to the instance that will run when the machine starts. This way instead of making several AMIs that are similar, you can have a single AMI and use the user data script to customize it.

EC2 Instance Metadata

You don't need to understand the script below, but in the docs they paste this to the console at instance creation to configure an instance as a web server.

```
#!/bin/bash
yum update -y
amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2
yum install -y httpd mariadb-server
systemctl start httpd
systemctl enable httpd
usermod -a -G apache ec2-user
chown -R ec2-user:apache /var/www
chmod 2775 /var/www
find /var/www -type d -exec chmod 2775 {} \;
find /var/www -type f -exec chmod 0664 {} \;
echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php
```

Run commands on your Linux instance at launch, AWS docs

Instance purchasing options

- **On-Demand Instances**
  - Pay for compute capacity by the second with no long-term commitments
  - Use On-Demand Instances for applications with short-term, irregular workloads that cannot be interrupted.
  - On-Demand Instances docs
- **Reserved Instances**
  - Pay up front for an EC2 instance, reserving it for a 1 to 3 year period.
  - Do this for predictable workloads that will run for a long time.
  - The longer you reserve the instance for and the more you pay up front (you can do a partial upfront) the larger the discount.
  - There are different types of reserved instances.
    - **Standard Reserved Instances** work as described above.
    - **Convertible Reserved Instances** allow you to change instance type. You get a smaller discount for this.
    - **Scheduled Reserved Instances** let you reserve within a part of the day. This seems to be discontinued.
  - Reserved Instances docs
- **Spot Instances** are unused EC2 instances that are available for less than the On-Demand price.
  - The spot price is the hourly price set by Amazon EC2 based on the supply and demand for EC2 instances.
  - You set a maximum price per hour you are willing to pay. When the spot price exceeds your maximum price then your EC2 instance is terminated.
  - Well suited for things that don't need to run long, like data analysis or batch processing.
  - Spot Instances docs
- **Dedicated Hosts** are physical servers with EC2 instance capacity fully dedicated for your use. You do not share the physical hardware with anyone else.

    - Some software licenses might need you to know information about your instances per-socket, per-core, or per-VM. Dedicated hosts provides visibility into these things for compliance with licenses.
    - Dedicated Hosts docs
- **Dedicated instances** are Amazon EC2 instances dedicate to a single customer. The instance may share hardware with other non-dedicated instances from the same account.
    - Dedicated Instances docs

Dedicated Hosts and Dedicated Instances can both be used to launch Amazon EC2 instances onto physical servers that are dedicated for your use.

Here are some differences between the two:

Dedicated Hosts docs

More on Spot Instances

Spot Fleet

A Spot Fleet is a collection, or fleet, of Spot Instances, and optionally On-Demand Instances. This fleet of instances tried to meet the capacity specified in the spot fleet request.

A Spot Instance pool is a set of unused EC2 instances with the same instance type (for example, m5.large), operating system, Availability Zone, and network platform.

The allocation strategy for the Spot Instances in your Spot Fleet determines how it fulfills your Spot Fleet request from the possible Spot Instance pools represented by its launch specifications. The following are the allocation strategies that you can specify in your Spot Fleet request:

- `lowestPrice`
    - The Spot Instances come from the pool with the lowest price. This is the default strategy.
- `diversified`
    - The Spot Instances are distributed across all pools.
- `capacityOptimized`
    - The Spot Instances come from the pool with optimal capacity for the number of instances that are launching.

Spot Fleet docs

Request Types

In the diagram below we see that a spot request launches instances. The spot request has a **request type** which determines if launched instances restart or not upon interruption (if the spot price goes above your max price or if you manually interrupt). Instances launched from a one-time spot request will go away, but instances launched from a persistent spot request will be restarted by the spot request. Thus, if you wish to terminate a persistent spot instance you must first terminate the request.

Spot Instance request docs

EC2 Instance Types

Instances are classified as general purpose, compute optimized, memory optimized, and storage optimized.

- **General purpose** instances are not specialized for particular use case. **M** type instances are good all-around. **T** type instances are burstable
- **Compute optimized** instances are for high computational loads requiring more CPU. Consider using for scientific computing. **C** type instances.
- **Memory optimized** have higher RAM for applications that need more memory like in-memory caches. **R** type instances.
- **Storage optimized** instances are made for workloads with lots of sequential read/write access on data sets in local storage. Good for data warehousing applications like MapReduce and Hadoop. **D, H, and I** type instances.

EC2 Instance Types docs

To determine if your instance is over-provisioned you can use the AWS Compute Optimizer.

Placement groups

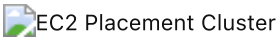
EC2 tries to spread out your instances to minimize correlated failures. You can use placement groups to influence the placement of a group of interdependent instances to meet the needs of your workload. Types of placement groups are -

- **Cluster** packs instances close together inside an Availability Zone. This strategy enables workloads to achieve the low-latency network performance necessary for tightly-coupled node-to-node communication that is typical of HPC applications.
- **Partition** multiple groups of instances where each group belongs to the same rack in a data center, and different groups belong to different racks. This strategy is typically used by large distributed and replicated workloads, such as Hadoop, Cassandra, and Kafka.
- **Spread** strictly places a small group of instances across distinct underlying hardware to reduce correlated failures.

Rules and Limitations

Cluster

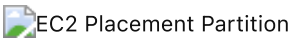
Use this for low network latency and high network throughput. Correlated failures are a risk.


EC2 Placement Cluster

Partition

Use this for distributed data processing. If a rack fails a group of instances may go offline.

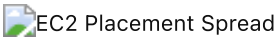
You can only have 7 partitions per AZ, so if there are three AZ in a region we can have 21 partitions. Within each partition you can have as many instances as allowed by your account.


EC2 Placement Partition

Spread

Each instance is on its own rack. Each rack has its own power source and network.

You can only have 7 *instances* per AZ, so if there are six AZ in a region we can have 42 partitions. Within each partition you can have many instances.

 EC2 Placement Spread

Placement Groups docs

Network Interfaces

An elastic network interface is a logical networking component in a VPC that represents a virtual network card. It can include the following attributes:

- A primary private IPv4 address from the IPv4 address range of your VPC
- One or more secondary private IPv4 addresses from the IPv4 address range of your VPC
- One Elastic IP address (IPv4) per private IPv4 address
- One public IPv4 address
- One or more IPv6 addresses
- A MAC address

You can create a network interface, attach it to an instance, detach it from an instance, and attach it to another instance. The attributes of a network interface follow it as it's attached or detached from an instance and reattached to another instance. When you move a network interface from one instance to another, network traffic is redirected to the new instance.

Each instance has a default network interface, called the primary network interface. You cannot detach a primary network interface from an instance.

VPC

VPC Basics

The VPC (Virtual Private Cloud) is how networking is done in the AWS cloud. The VPC is like having your own data center inside AWS. The VPC separates resources from different customers and different projects.

VPCs are specific to a region, and a single VPC will span all the availability zones in a region. You can make subnets of a VPC, subnets are specific to availability zones.

When you create a VPC, you must associate an IPv4 CIDR block for it. The CIDR block must contain between 16 and 65,536 IP addresses (netmasks of `/28` and `/16` respectively). If you see a question asking about CIDR block sizes know that it is between `/28` and `/16`.

Your CIDR block must be in the private IP ranges:

- `10.0.0.0 - 10.255.255.255`, CIDR `10.0.0.0/8`
- `172.16.0.0 - 172.31.255.255`, CIDR `172.16.0.0/12`
- `192.168.0.0 - 192.168.255.255`, CIDR `192.168.0.0/16`

You can add multiple CIDR blocks to your VPC. CIDR blocks must not overlap, so we can't have both `10.0.0.0/28` and `10.0.0.1/28` in a VPC. You can never modify the range of an existing CIDR block

How VPC works, AWS docs

Subnet Sizing

When you divide an IP network into multiple parts, each part is called a subnet. The subnets will have CIDR blocks that are subsets of the CIDR block of the VPC.

The number of available IPv4 addresses in your subnet's CIDR block is not exactly what you think it would be.


IPv4

In a `/24` IPv4 we would expect for there to be 256 addresses, 2^(32-24) = 2^8 = 256. The reason there are only 251 available is that AWS reserves some of the IP addresses for it's own use.

The 5 missing IP addresses are reserved as follows:

- `172.31.80.0` is used as the network address
- `172.31.80.1` is reserved for the VPC router
- `172.31.80.2` is reserved for the DNS
- `172.31.80.2` is reserved by AWS for future use
- `172.31.80.255` is the network broadcast address. AWS does not support broadcast so this is reserved.

If a question asks what IP addresses you can use, the first 4 IP addresses are reserved, as well as the last one. Be able to calculate the CIDR range for a simple example like 10.0.0.0/24.

VPC subnets docs

Route Tables

Route tables specify how network traffic from subnets or the internet should be directed within the VPC.

Every subnet needs to be associated with a route table. This route table will direct traffic to the subnet.

Example

Here is a route table for a VPC with CIDR block `172.31.0.0/16`.

This route table is saying that traffic to the VPC (`172.31.0.0/16`) is local to the VPC and that traffic elsewhere (`0.0.0.0/0`) goes to `igw-d2b99dba` (this is an **internet gateway**, we discuss this later).

Implied Routing

At the beginning of this section we said that every subnet needs to be associated with a route table, but our route table didn't say anything about any subnets. This is explained by the following image:

There is a **main route table** which is created when a new VPC is created. You do not need to explicitly associate a new subnet with a route table, there is an automatic association with the main route table.

You do not need to explicitly define routes for traffic *between subnets*. The VPC knows what ranges your subnets exist on and will take care of this for you.

Main Route Table

Let's go over an example on the main route table from the AWS documentation.

Suppose you have two subnets and two route tables. Initially, both subnets have an implicit association with Route Table A, the main route table. We want to change both subnets to be associated with route table B.

We can create an explicit association between subnet 2 and Route Table B.

We can change the main route table from A to B, which will update the implicit association of subnet 1 from A to B.

We can delete the explicit association between subnet 2 and table B, and it will still have an implicit association with the route table.

A route table can be associated with multiple subnets, but a subnet cannot be associated with multiple route tables.

Route tables docs

Internet Gateway

Instances that have a public IP address in a subnet can connect to the internet we need an internet gateway. Let's look at a route table again to see how this works:

Traffic going to the private IPs of the CIDR block for the VPC stay `local` to the VPC. All other traffic goes to `igw-d2b99dba`, which is an internet gateway that will take this traffic to the internet. If this rule is not in the route table, then traffic will not get routed to the internet gateway even if there is a route table.

The internet gateway horizontally scales, is redundant, and is highly available. AWS manages these things. You do not need to worry about availability or scaleability of your internet gateways.

Internet Gateway docs

NAT devices

The internet gateway connects devices with public IPs to the internet. Some things shouldn't be accessible by the internet so they don't have a public IP. They still might need to access the internet to update or install new software though!

We solve this problem by using a NAT (Network Address Translation) device. The NAT device is in a public subnet and has a public IP that is used to communicate with the internet.

The NAT device allows devices with private IPs to initiate outbound connections to the internet over IPv4 while preventing unwanted inbound connections

NAT devices are either a **NAT instance** or a **NAT gateway**. The NAT instance has you do translation on an EC2 instance, the NAT gateway is managed by Amazon. NAT gateways are now the preferred NAT device (they are better and easier to use), but NAT instance questions can still appear on the exam.

NAT instances

In the diagram below we see database servers that are in a private subnet and are connected to the internet using a NAT instance. The private subnet contains no routes pointing to the internet gateway, and the servers inside do not have public IPs.

Here is how it works:

1. The Database servers send a request to a public IP.
2. This public IP is in the range `0.0.0.0/0` and so it is routed to the NAT instance at `nat-instance-id`.
3. The NAT instance sends this request to the public IP and receives a response back.
4. The NAT instance sends the response back to the database servers on the private subnet at `10.0.1.0/24`.

The NAT instance has an elastic IP (an IP address that doesn't change), this is required.

You must disable **source/destination checks** on your EC2 instance when using it as a NAT instance. These check if the instance is either the source or the destination of network traffic before accepting the traffic. The NAT instance is not the source/destination of the traffic, it is a middleman between the private subnet and the internet.

NAT Gateway

The NAT Gateway does NAT but is managed by AWS so you don't have to worry about managing your own EC2 instance. NAT gateways will automatically scale up to 45 Gbps, but the NAT instance is dependent on the EC2 instance type.

Just like NAT instances, the NAT gateway has an elastic IP and lives in a public subnet.

The NAT gateway is redundant within each availability zone, but if the availability zone experiences problems then you will lose connectivity. You might be tempted to use a single NAT gateway in a single availability zone (NAT Gateways are specific to subnets which are specific to AZ), but then everything in your region loses connectivity if you lose a single AZ. This is why it is best to have separate NAT gateways for each AZ.

A comparison of the NAT gateways and instances from the AWS documentation:

NAT docs

Egress-only internet gateways

When we have a public IPv4 address or an IPv6 address and our EC2 instance is connected to a public gateway, we can send and receive traffic from the internet.

People on the internet can send us traffic even if we didn't ask for the traffic. This can be solved for IPv4 by having an instance in a private subnet that is attached to a NAT gateway.

For IPv6 traffic we can connect to an egress-only internet gateway (instead of a plain internet gateway). This will prevent unwanted traffic from reaching our instance. It will allow us to send requests to the internet and receive responses though.

Egress-only internet gateway docs

Security Groups

A security group is a virtual firewall. A firewall is a network device that decides what incoming and outgoing traffic to allow or to disallow. This firewall controls which CIDR blocks and security groups can communicate with an EC2 instances, as well as what ports this communication can happen on.

Multiple EC2 instances can have the same security group. The security group belongs to a VPC and all instances in that VPC can use the security group. All EC2 instances must have a security group.

Here are some properties of security groups:

- You can specify allow rules but not deny rules, traffic that is not explicitly allowed is denied.
- There are separate rules for inbound and outbound traffic.
- Security groups are stateful. Suppose they sent a request to an IP address. They will be able to receive the response to their request even if they do not have an inbound security rule that would allow the traffic.

Default Security Groups

- When creating a new security group no inbound traffic is allowed by default. This is in contrast to the default security group that is created when a VPC is created, which does allow inbound traffic **only for traffic originating from the same security group**.
- For both custom and default security groups, all outbound traffic is allowed by default.

Security Groups docs

Network ACL (Network Access Control List)

Network ACLs are also a type of firewall. Here are some differences between the network ACL and the security group.

- Network ACLs are applied to subnets, security groups are applied to EC2 instances.
- The network ACL is stateless, responses to inbound traffic are subject to the rules for outbound traffic. This is not true for security groups, where the outbound rules don't apply for responses to received inbound traffic (it is stateful).
- The network ACL can have both allow and deny rules. The security group only has allow rules and everything not explicitly allowed is implicitly denied.

Here are some similarities between the network ACL and the security group.

- They both control acccess within a VPC.
- All subnets must be associated with a network ACL, just like all EC2 instances must have a security group.
- You can associate a network ACL with multiple subnets, just like a security group can be associated with multiple EC2 instances.
- VPCs come with a default network ACL, just like EC2 instances come with a default security group.

The default NACL that comes with the VPC allows all traffic in and out of the VPC. If you make a custom NACL, it will deny all inbound and allow all outbound traffic unless you make changes to these settings.

Since there are both allow and deny rules we need a way of deciding which rule is correct when the rules conflict. To solve this, there are numbers associated with each rule, and the lowest number wins. These numbers are from 1 to 32766. AWS recommends spacing out your rules, so that you can put a rule between your rules (ex. put rule 15 between rule 10 and rule 20) in case you need to.

Network ACL docs

VPC Endpoints

VPC endpoints enable private connections between your VPC and AWS services.

Suppose we want to access an AWS service (like a database) from a private subnet. This service has a public IP that we can use. So we can use a NAT gateway to communicate from our private subnet to the internet which will communicate with the database.

It would be better if we could just talk to our AWS services over a private IP address and not send any traffic into the public internet. **This is the purpose of VPC endpoints.**

VPC endpoints are horizontally scaled, redundant, and highly available.

There are two types of VPC endpoints, the type you choose is determined by the service you are accessing:

- Gateway endpoints are used for S3 and DynamoDB (two AWS services discussed in depth later).
- Interface endpoints are used for other services.

VPC endpoint docs

AWS PrivateLink (VPC endpoint services)

You can privately connect to AWS services using VPC endpoints. You can build your own **VPC endpoint services** on AWS and connect to them with a VPC endpoint just like you would connect to a prebuilt AWS service.

In the diagram below our VPC endpoint service is in VPC B. The dashed line around it is the service provider, `vpce-svc-1234`. The ENI of subnet A can access the network load balancer using an interface endpoint. If the network load balancer has instances spread across multiple availability zones then the solution will be fault-tolerant.

VPC endpoint services docs

VPC Peering

VPC peering is a connection between two VPCs that allows you to route traffic between them privately. It allows you to communicate between different VPCs as if they were in the same VPC. VPCs can be peered within your own account or across different accounts. The VPCs must have non-overlapping CIDR blocks.

VPC peering is not transitive. In the image below, VPC A is connected to both VPC B and VPC C, but **VPC B is not connected to VPC C**.

When peering VPCs you need to update your route tables to route traffic in the appropriate private IP range to the VPC peering connection.

VPC Peering docs

Bastion Hosts

You can set up a bastion host where you ssh into a public subnet. This bastion host in the public subnet will be able to ssh into private subnets.

Bastion hosts blog post

Site to Site VPN Connection

You can configure a connection between a VPC and a local network, like your home network or the network for your office.

The diagram below shows the connection between an AWS VPC and an on-premises network. Let's talk about some of the components:

- **VPN connection**: A secure connection between AWS and your on premises network.
- **Virtual private gateway**: A VPN concentrator (this just means that it sets up a secure connection) on the AWS side.
- **Customer gateway**: A physical device that you set up on-premises to connect with AWS.

Site-to-Site VPN docs

VPN CloudHub

VPN cloudhub allows for multiple site-to-site connections and for communications between your sites. Sites must have non-overlapping IP ranges.

VPN CloudHub docs

Transit Gateway

Cloudhub is specifically for the case when you need to connect multiple on-premise data centers to AWS, **but the transit gateway is more general**.

Transit gateway lets you connect VPN, direct connect, VPCs and more.

Suppose we want to connect many VPC, direct connect, and VPNs together. VPC peering is not transitive and is not general enough. We would need 45 connections to connect 10 things together so that everything can talk to eachother.

With the transit gateway we connect all of the VPCs to the transit gateway and they can all talk to eachother. In the diagram below everyone can talk to eachother because the transit gateway allow transitive connection.

Transit gateways support IP multicast. This means that you can send multiple IP addresses at once to a transit gateway and it will communicate to them all. This is not supported in our other routing methods.

Transit gateway docs

Direct Connect

AWS direct connect allows you to connect to AWS while bypassing your internet service provider. This is done to improve bandwidth and latency for things like working with large data sets or real-time data feeds.

Direct Connect docs

VPC Flow Logs

VPC flow logs help you track information about IP traffic going in and out of the network interfaces (ENIs) of your VPC.

Flow logs can help you -

- Troubleshoot issues with security group rules
- Monitor traffic reaching an instance

You can create flow logs at varying levels of granularity: for a VPC, subnet, or a network interface. If you create a flow log for a subnet or VPC all the ENIs in the VPC/subnet will be monitored. You can write flow logs either to an S3 bucket (a storage service) or to Cloudwatch (a cloud monitoring service).

Flow logs docs

S3

Buckets

You can upload files (pictures, videos, data sets) to an Amazon S3 **bucket**. Generally you will access the buckets and their contents programatically, but you can also use the AWS console to work with the buckets. The buckets must have a name that follows certain conventions:

Buckets Overview docs

- Bucket names must be between 3 and 63 characters long.
- Bucket names can consist only of lowercase letters, numbers, dots (.), and hyphens (-).
- Bucket names must begin and end with a letter or number.
- Bucket names must not be formatted as an IP address (for example, 192.168.5.4).
- Bucket names must be unique within a partition. A partition is a grouping of Regions. AWS currently has three partitions: aws (Standard Regions), aws-cn (China Regions), and aws-us-gov (AWS GovCloud [US] Regions).

Bucket Naming docs

Objects

The files you store in S3 are called *__objects__. Each object has a __key__ which is a unique identifier within the bucket, and associated with this key is the __value__ which is the stored file.

Objects can be up to 5TB in size but you can only upload 5GB at a time so you will need to use a multi-part upload for files larger than 5GB.

Using Objects

Folders can be represented within S3 -

S3 only supports buckets and objects (the files are a lie!) and this filesystem interface is a convenience provided to users of the console.

The object key refers to the entire "path" to the object. So the object key might be "folder/fileInFolder.png".

Object Keys docs

Data Consistency

In December 2020 AWS delivered strong read-after-write consistency for PUTs and DELETEs on objects. This means that if you upload a new object then you can immediately read it from the bucket.

This is how you want things to behave but before this S3 had eventual consistency, which meant you often had to wait a moment to guarantee you had fresh data.

If you have concurrent requests things can get tricky. Like if you are uploading a new version at the same time as you are requesting the object, you might get either the old or the new version.

Bucket configurations are eventually consistent, so if you enable versioning you might want to wait 15 minutes before starting to upload things.

Welcome to S3 docs

Encryption

You can do encryption server-side or client-side, and within server-side you have several options.

- Server-Side
  - SSE-S3: Server-Side Encryption with Amazon S3-Managed Keys
  - SSE-KMS: Server-Side Encryption with Customer Master Keys (CMKs) Stored in AWS Key Management Service
  - SSE-C: Server-Side Encryption with Customer-Provided Keys
- Client-Side

Encryption docs

SSE-S3

S3 encrypts each object with a key, which is also encrypted by a master key, and the master key is regularly rotated. Uses AES-256 encryption.

When using the REST API for S3 set the `x-amz-server-side-encryption` request header to `AES-256` to let AWS know that they need to server-side encrypt the object before uploading it to S3.

SSE docs

SSE-KMS

When using the REST API set the `x-amz-server-side-encryption` request header to `aws:kms`.

AWS KMS is the AWS Key Management Service and it manages keys for encryption. Using this service to encrypt S3 data via SSE-KMS will provide a better audit trail than using SSE-S3.

KMS docs

SSE-C

You must provide the encryption key to AWS when you upload the object. If you want to read the encrypted object, you must provide the same key! If you forget which key goes to which object or lose the key, the object is lost forever.

You must use HTTPS because you are sending sensitive information (the key) over the internet.

Customer keys docs

Client-Side Encryption

You write an encrypted object to S3 and store it there. You can retrieve this encrypted object and decrypt it yourself.

Client-side encryption docs

Managing Access

By default only the person creating an S3 bucket has access to it. To allow other users access we can either make the bucket more permissive or give the other user elevated permissions.

The bucket controls who accesses it via **Bucket ACLs** and **Bucket Policys**. Objects within the bucket can have an **Object ACL** that specifies who can access.

Users are granted privileges through AWS IAM.

S3 Access Control

CORS

Suppose we have a website https://example1.com, and someone else has an api at https://example2.com/api. The owner of example2.com might not want you using their api. By default the browser will prevent this unless the owner of example2.com explicitly allows it using CORS (Cross Origin Resource Sharing).

The implementation is that the browser will send a preflight request asking example2.com/api if example1.com can access it.

If you get an error like `No 'Access-Control-Allow-Origin' header is present on the requested resource.` then you have a CORS error.

Versioning

Versioning helps you prevent accidentally overwriting or deleting a file. In a versioning enabled bucket if the same object key is written multiple times, all of the writes will be recorded with the same object key but having different version IDs.

If you delete an object in a versioned S3 bucket all of the values associated with that key will remain, but a delete marker will be placed to indicate that the object has been deleted. To restore the data you can simply delete the delete marker. If you want to actually delete something from a versioned bucket you will have to specify which version you want to delete.

If you enable versioning on a previously unversioned bucket then all existing objects will be given a version ID of null. Once you enable versioning on a bucket it can no longer go back to the unversioned state, but you can suspend versioning. Suspended versioning buckets retain all existing versions but otherwise behave like an unversioned bucket.

Versioning docs

MFA Delete

You can enable **MFA delete** to add another layer of security.

MFA delete requires additional authentication for either of the following operations:

- Changing the versioning state of your bucket
- Permanently deleting an object version

MFA delete requires two forms of authentication together:

- Your security credentials
- The concatenation of a valid serial number, a space, and the six-digit code displayed on an approved authentication device

This means that if your account is compromised that the attacker will not be able to delete any object versions.

The bucket owner, which is the AWS account that created the bucket (root account), and all authorized IAM users can enable versioning. However, only the bucket owner (root account, not IAM user) can enable MFA delete.

You cannot enable MFA Delete using the AWS Management Console, you must use the CLI.

Storage Classes

- **S3 Standard**
  - Designed for frequently accessed data.
  - Replicated across multiple availability zones for increased availability.
  - No retrieval fees.
- **S3 Standard-IA**
  - Long-lived infrequent accessed data.
  - Per-GB retrieval fees, which is why you use this for infrequently accessed data.
- **S3 One Zone-IA**
  - Long-lived, infrequently accessed, non-critical data.
  - Only stored in one AZ, not resilient to loss of AZ.
  - Less expensie than Standard-IA.
  - Yes retrieval fees.
- **S3 Intelligent-Tiering**
  - Use for long-lived data with changing or unknown access patterns.
  - Automatically puts the object in an optimal storage class based on historical access patterns.
  - Monitoring and automation fees apply per object.

    - No retrieval fees.
- **S3 Glacier**
    - Use for archives where portions of the data might need to be retrieved in minutes. Data stored in the S3 Glacier storage class has a minimum storage duration period of 90 days and can be accessed in as little as 1-5 minutes using expedited retrieval. If you have deleted, overwritten, or transitioned to a different storage class an object before the 90-day minimum, you are charged for 90 days.
    - Per GB retrieval fees apply. You must first restore archived objects before you can access them.
- **S3 Glacier Deep Archive**
    - Use for archiving data that rarely needs to be accessed. Data stored in the S3 Glacier Deep Archive storage class has a minimum storage duration period of 180 days and a default retrieval time of 12 hours. If you have deleted, overwritten, or transitioned to a different storage class an object before the 180-day minimum, you are charged for 180 days.
    - The lowest cost storage option in AWS.
    - Per GB retrieval fees apply. You must first restore archived objects before you can access them.

Lifecycle Transitions

There are two types of lifecycle transitions.

- **Transition actions**: Define when objects transition to a different storage class.
- **Expiration actions**: Define when objects expire. Amazon S3 deletes expired objects on your behalf.

Amazon S3 supports a waterfall model for transitioning between storage classes, as shown in the following diagram. So you can't transition from a lower step to a higher step.

Before you transition objects from the S3 Standard or S3 Standard-IA storage classes to S3 Standard-IA or S3 One Zone-IA, you must store them at least 30 days in the S3 Standard storage class.

Optimizing Performance

Prefixes

Amazon S3 automatically scales to high request rates. For example, your application can achieve at least 3,500 PUT/COPY/POST/DELETE or 5,500 GET/HEAD requests per second per prefix in a bucket. Valid prefixes for a bucket `mybucket` include `mybucket/folder1`, `mybucket/folder2`, `mybucket/folder1/subfolder`. There are no limits to the number of prefixes in a bucket.

You can increase your read or write performance by parallelizing reads. For example, if you create 10 prefixes in an Amazon S3 bucket to parallelize reads, you could scale your read performance to 55,000 read requests per second. Similarly, you can scale write operations by writing to multiple prefixes.

Transfer Acceleration

S3 Transfer Acceleration lets users upload to an edge location in AWS Cloudfront which then sends the data to AWS over an optimized path.

Transfer acceleration will not happen if Amazon does not think it will cause meaningful improvements.

Databases

What is a database?

Let's talk about the basic types of databases before we talk about AWS services. Databases help you store and interact with data.

Let's review some different types of databases.

What is a relational database?

Skip this section unless you need a review of relational databases.

Consider an application like Instagram. There are users. Users can take photos. People can like the photos. How should we represent this? The relational database solution is to have separate tables that represent the users, photos, and likes.

The user table contains our users. If we have two users, one cat and one dog, the user table might look like this:

Suppose we have one photo, a selfie uploaded by the cat. Notice that we use the `id` from the user table to indicate the owner of the photo. This is why it is a **relational** database.

The user with id 2 likes the photo with id 1, the dog liked the cat's selfie.

Here is a rough diagram of the way things work. ER diagrams are outside the scope of discussion so some conventions are not followed.

What is a NoSQL database?

NoSQL databases don't have the rectangular tables and relationships between tables the way relational databases do. Instead of relationships we store data in the way that it is queried. When we view a user's profile we need their photos, so maybe photos should be included with the username.

```
[
  {
    "id": 1,
    "username": "cat",
    "password": "meow",
    "photos": [
      {
        "id": 1,
        "caption": "selfie",
      }
    ]
```

```
  },
  {
    "id": 2,
    "username": "dog",
    "password": "woof",
    "likes": [
      1
    ]
  },
]
```

Since the dog had no photos there is no attribute for photos. Different items have different fields. We say that it is **schemaless**.

NoSQL drawbacks: We have all the data as the relational database previously discussed. A pain point of NoSQL databases is the lack of flexibility in querying. Consider that we can easily access all the likes of the dog, but that our photo doesn't know who liked it. We could solve this by including likes on the photo, but what if a user has 1000 photos that each get 1,000,000 likes? Should we really store all that information in one place? NoSQL requires a different way of thinking. You need to know how you are going to use your data before setting up the tables. With relational databases SQL gives us a lot of flexibility in how we write our queries.

NoSQL advantages: NoSQL databases are built for scalability and performance. With relational databases you typically scale vertically by using a bigger machine to host your database. With NoSQL you scale horizontally, adding servers as you need.

Online transaction processing vs. Online analytical processing

OLTP is used to manage transactional workloads, like inserting a new user when a user signs up to a website. OLAP is used to run complex queries for things like business intelligence.

Relational databases can be row oriented or column oriented.

Row oriented databases store entries of a row in adjacent memory locations. This makes it easy to insert/delete rows for transactional (OLTP) workloads.

Column oriented databases store entries of a column in adjacent memory locations. Analytical workloads (OLAP) often involve operations on a column like summing it or grouping it, so this architecture is optimized for analytical workloads.

Amazon Redshift is column oriented, databases on RDS are row oriented. You can learn more about Redshift architecture here

What is an in-memory database

Computers have memory (RAM) and storage. Storage is either Solid-state drive (SSD) or hard disk drive (HDD).

Memory is everything your computer is currently thinking about. Storage is everything your computer knows.

It is fast to access something from memory, it is slow to access something from storage. This is why more RAM speeds up computers, they access storage less often.

Relational databases store all of their data in storage, what if we had a database that stored everything in memory? We would have faster access times. This is why we use in-memory databases (IMDB).

Amazon Relational Database Service (RDS)

Amazon RDS manages a server with a relational database installation and automates common tasks like performing backups or patching software. AWS doesn't even give you access to the shell for the machine RDS is running on, you won't need it.

RDS supports the following databases:

- MySQL
- MariaDB
- Oracle
- SQL Server
- PostgreSQL

Let's talk features.

Scalability

- Scale storage on the fly with no downtime.
- **Read replicas** - a read-only replica of your database. Serve high-volume read traffic from the read-replica and the primary database for increased overall read capacity.

Availability and durability

- Backups allow you to restore your database from a previous state. There are two types.
  - Automated backups: Allows you to recover to a point in time from the retention period The retention period can be configured to be up to 35 days.
  - Database snapshots: User initiated snapshots that are stored in S3 and must be manually deleted.
- You can deploy to multiple availability zones (AZs). This makes a primary instance in one AZ which replicates data to an instance in another AZ. If the infrastructure in your primary AZ fails, RDS will automatically transfer to the replica with minimal downtime.

Security

At rest and in transit: You are able to encrypt data at rest with keys managed in AWS Key Management Service (KMS). On a database running with encryption, data in the underlying storage is encrypted, as well as the data in automated backups, read replicas, and snapshots. Data in transit is secured using SSL.

Network isolation: Run your database in a VPC. Use network access control lists (NACLs) and security groups to control traffic to the RDS instance.

Manageability

AWS provides Amazon CloudWatch metics for your database instances at no extra charge.

RDS features overview

Aurora

Amazon Aurora is a relational database engine that is compatible with MySQL and PostgreSQL. Aurora isn't competing with RDS, Aurora is an instance type on RDS just like PostgreSQL. There are some special features of Aurora that improve upon the other RDS instance types.

Performance

Aurora offers five times the performance of MySQL and three times the performance of PostgreSQL.

Hardware and Scaling

You can scale storage: You do not provision storage for Aurora. Aurora will automatically scale storage up to 128TB with a minimum storage of 10GB depending on your use. Scaling does not impact performance.

You can scale compute: Compute depends on your instance type. You can change the instance type of an existing DB but this will impact availability during the maintenance period.

Backup and Restore

As with all RDS instances, we can take snapshots to backup the database. For Aurora specifically, automated backups are always enabled.

High Availability and Replication

Aurora is fault tolerant. Your database is replicated 6 ways across 3 availability zones. Aurora is also self healing, like wolverine.

You can make Aurora replicas for read-scaling or high availability. You can also make MySQL replicas if you are using Aurora MySQL. The main benefit of MySQL replicas is that they can be cross-region, and Aurora replicas are confined to the region of the instance. I am not sure the use case for MySQL replicas because for cross region replication I would use Aurora Global Database.

Aurora Global Database allows for you to physically replicate your database across regions. It is recommended for low-latency global reads and disaster recovery.

Security

You can encrypt data at rest and in transit as you would in other RDS instance types.

Aurora Serverless

There is a serverless offering that autoscales to your required capacity. Ideal for unknown or variable workloads.

https://aws.amazon.com/rds/aurora/faqs/

Amazon DynamoDB

Amazon DynamoDB is a serverless NoSQL database. With RDS you have to choose what kind of EC2 instance Amazon will manage your database on. With DynamoDB we don't worry about that. We can scale up and down as much as we like without worrying about the size of an EC2 instance.

We don't have to worry about availability or fault tolerance, they are built right in.

Performance

DynamoDB Accelerator (DAX) is an in-memory cache that improves read performance up to 10x, bringing read times from single digit millisecond to microseconds.

DynamoDB global tables replicate tables across regions to scale capacity and allow local access in selected regions for improved performance.

Serverless

- DynamoDB has two modes, on-demand and provisioned.
  - Use on-demand if you don't know how much peak capacity you need.
  - Provisioned will automatically scale, but up to a certain maximum capacity (unlike on-demand). Provisioned capacity is more cost effective when you are able to predict the max capacity.
- When data is modified in DynamoDB you can capture it in a DynamoDB **stream**. This will capture writes/updates/deletes from a DynamoDB table. You can use the stream with AWS Lambda to perform actions in response to changes made to the table.

Enterprise Ready

- Supports transactions. These are multiple batched operations that either all succeed or all fail. Useful when a single operation needs to make multiple operations.
- Data is encrypted by default with AWS Key Management Service (KMS).
- Point-in-time recovery allows you to continuously back up your data for a specified retention period.
- On-demand backup and restore lets you manually create a full backup of your tables.

DynamoDB features overview

Amazon Redshift

Redshift is a column oriented database used for OLAP.

You load data into Redshift tables, and then you do parallel processing on the loaded data to perform the analytics.

The leader node sends instructions to the compute nodes which perform the instructions in parallel on data from the client applications.

You can query live databases using federated queries. You can query from S3 without loading into Redshift tables using Amazon Redshift Spectrum.

You pay per hour with the payment rate determined by the size of your compute node.

Elasticache

Amazon Elasticache makes it easy to host an in-memory database (IMDB) in the cloud. It supports two types of databases, Redis and Memcached. Redis is more popular and more flexible than Memcached.

Elasticache gives you sub-millisecond performance. Use Elasticache if you need low latency.

You can cache results from a database to improve latency and performance, or you can just use it as a fast key-value store for things like user-authentication tokens.

Elaticache stores simple data, not complex related tables.

To set up Elasticache you tell AWS what size instance you need. They provision the resources and manage the installation for you (installing patches, server maintenance, etc).

Elasticache resources belong to a VPC so you can use security groups and network ACLs to control access to your instance.

Elasticache is integrated with CloudWatch for monitoring.

Other Databases

I'm not so sure about what kind of questions you might see.

- AWS Database Migration Service is for migrating databases.
- Amazon Neptune is a graph database for highly relational data
- Amazon Elasticsearch Service allows you to perform a fuzzy search on JSON data.

AWS Global Infrastructure

Traditional Data Centers

Before cloud computing companies had to manage physical computation resources themselves. This meant a room full of servers, cables, and people that needed to understand how all of these things worked. This room is called a data center.

AWS vs. Traditional Data Centers

With AWS you are paying to use Amazon's machines. They manage the data centers, not you.

One advantage of this is that you can just pay for what you need. In the past companies would have to commit to buying servers. This is a big commitment, and it would be easy to overestimate or underestimate your needs.

You might not have money to buy servers. You might only need them for one month of the year, or one hour of the day. This is why it is better to just pay for what you need with AWS.

Global Infrastructure

AWS has lots of data centers so that it can meet the needs of huge companies like Amazon and Netflix.

A cluster of data centers is called a **region**. Within this cluster there are more clusters called availability zones.

The reason for having physically separate availability zones is that the region is more fault-tolerant in case of a natural disaster.

Availabilty zones within a region are connected with redundant, high-bandwidth, low-latency networking. Data centers within the availability zone are also connected.

Amazon Simple Notification Service (SNS)

What is SNS?

Amazon SNS delivers messages from publishers to subscribers, this pattern is often called pub-sub. You can publish a message to an SNS topic and all of the subscribers to this topic will receive the message.

Subscribers to the topic can be SQS queues, HTTP, email, mobile push notifications, and mobile text messages (SMS). The publishers are AWS services, an example use case is using SNS to send yourself a text message when you set off an Amazon CloudWatch alarm.

Fanout pattern

A common use case is sending messages to many different subscribers. Having an SNS topic means our publisher only has to send data to one place, the SNS topic is used to "fanout" the message.

You can use the fanout pattern to replicate data sent to your production environment with your test environment, this way you can test your application with data received from your production application.

Common SNS Scenarios - docs

FIFO

Just like there are SQS standard and SQS FIFO, there are SNS standard and SNS FIFO. The differences are quite similar.

- Standard is best-effort message ordering, FIFO is always in the same order the message was sent.
- Standard is at-least once message delivery, FIFO is exactly-once message delivery.
- Standard has higher throughput than FIFO.
- Only FIFO SQS queues can subscribe to FIFO SNS topics. Lambda, HTTP, SMS, email, and mobile apps can all subscribe to standard SNS topics but not FIFO SNS topics.

Use FIFO SNS topics with FIFO SQS queues when you need to decouple an application and the order of messages is important.

Message Filtering

By default, an Amazon SNS topic subscriber receives every message published to the topic. To receive a subset of the messages, a subscriber must assign a filter policy to the topic subscription.

Messages sent from an SNS topic can include a `MessageAttributes` field that can be filtered on by subscribers.

```
{
   "Type": "Notification",
   "MessageId": "a1b2c34d-567e-8f90-g1h2-i345j67klmn8",
    ...,
   "MessageAttributes": {
      "customer_sport": {
         "Type": "String",
         "Value": "soccer"
      },
      "store": {
         "Type": "String",
         "Value":"example_corp"
```

```
        },
        "event": {
            "Type": "String",
            "Value": "order_placed"
        },
        "price_usd": {
            "Type": "Number",
            "Value":210.75
        }
    }
}
```

In the subscription filter we can define a JSON policy that determines which messages we accept. Here is a policy that accepts.

```
{
    "store": ["example_corp"],
    "event": [{"anything-but": "order_cancelled"}],
    "customer_interests": "soccer",
    "price_usd": [{"numeric": [">=", 100]}]
}
```

This should be detailed enough for the exam, but you can learn more in the documentation.

Amazon Simple Queue Service (SQS)

What is SQS?

Amazon Simple Queue Service helps you decouple message producers and message consumers.

Suppose you have an online voting application that will have millions of people voting at once. It would be hard to handle millions of messages per second, but with SQS you can have a queue that stores all the votes and then you process them as they come.

A queue is just a fancy word for a line, you take all the votes and you make them sit in a line until you are ready to process them, just like lines work at the grocery store.

What are possible use cases for Amazon SQS - StackOverflow

Standard vs. FIFO

There are standard and there are FIFO queues.

- Standard queues
    - Nearly unlimited number of transactions per second.
    - Messages are delivered at least once, but sometimes more than once.
    - Best effort ordering. Occasionally, messages will be delivered out of the order they were sent in.
    - Useful for very high throughput.
- FIFO queues
    - Up to 3000 messages per second, standard is nearly unlimited.
    - First-in-first-out means the first message in the first message out of the queue, compare this to standard queues which are best effort ordering.

    - No duplicates, messages delivered exactly once.

Configuration

- **Message Visibility Timeout**: The queue holds messages but the consumer must ask for the message, it doesn't "push" messages. Once a consumer asks for a message, the message disappears so that no other consumers ask for the same message while it is being processed.
    - The default visibility is 30 seconds, so once the message is consumed you have 30 seconds to process **and delete** the message before it will be returned to the queue.
    - If you do not delete the message before the timeout is up it will be returned to the queue.
- **Message Retention Period**: You need to delete your message from the queue once you process it otherwise it stays in the queue, but for how long? It stays until the message retention period is over.
    - The default message retention period is 4 days, so after 4 days a message will automatically be deleted from the queue.
- **Delivery Delay**: When you send a message to the queue the consumers can't access it until the delivery delay is over. By default the delay is 0 seconds.

Access Policy

You can manage access with IAM, but just like S3 buckets have their own access policies, SQS queues have their own access policies written in JSON. Here is the default policy saying that only the owner of the queue can send and receive messages from the queue.

```
{
  "Version": "2008-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__owner_statement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "889703873633"
      },
      "Action": [
        "SQS:*"
      ],
      "Resource": "arn:aws:sqs:us-east-2:889703873633:"
    }
  ]
}
```

Server-side encryption

SQS messages are encrypted in-flight by default, but not at rest. You can encrypt them at rest with server-side encryption and amazon will encrypt the messages in the queue with KMS and decrypt them when received by a consumer.

Dead-letter queue

We usually want to delete a message after using it, when a message has been received too many times we think something is wrong. We can configure how many times a message is received before we send it to the dead-letter queue.

The dead letter queue is just another SQS queue. You can look at the messages in the DLQ to see what is failing to process. It is recommended to set the DLQ retention period to be longer than your other queues, because it's expiration is based on when it entered the first queue, not when it entered the DLQ.

Amazon Kinesis

What is Kinesis?

Amazon Kinesis makes it easy to collect, process, and analyze real-time, streaming data so you can get timely insights and react quickly to new information.

Traditionally data will be stored in a database and analyzed days or weeks later. With streaming, you analyze the data in real-time as it is collected. This lets you make decisions faster since you see analytics as the data is collected.

There are four parts of Amazon Kinesis

- Kinesis Video Streams
- Kinesis Data Streams
- Kinesis Data Firehose
- Kinesis Data Analytics

We don't worry about Kinesis Video Streams for this exam.

Kinesis Data Streams

Shards

For Kinesis Data Streams you provision capacity in units called *shards*. The more shards you provision, the more it costs. One shard provides 1MB/sec data input and 2MB/sec data output along with 1000 PUT records per second. So if we had 10 shards that is 10MB/sec input, 20MB/sec output, and 10,000 PUT records per second.

We can increase the data output of a shard by enabling *enhanced fan-out*, which lets you read 2MB/sec **per consumer** from a shard instead of 2MB/sec across all consumers.

Records

A **record** is the unit of data stored in an Amazon Kinesis Data stream. Records have the following properties

- **Data blob**: The data of interest added to a data stream.
- **Partition key**: The partition key is defined by the data producer while adding data to the stream and determines which shard the record will go to.
- **Sequence number**: A sequence number is a unique identifier for each record that Amazon Kinesis adds when data is added to the stream
    - Sequence numbers from the same shard are ordered.
    - Use a partition key to messages from the same message sender a consistent shard, which will make the sequence numbers for this messager ordered.

Producers and Consumers

Data can be loaded into Kinesis Data streams using the API over HTTPS, the Kinesis Producer Library, and the Kinesis Agent.

You can read from data streams using AWS Lambda, Kinesis Data Analytics, Kinesis Data Firehose, or the Kinesis Client Library. The Kinesis Client Library lets you read from a data stream with a custom application, the library is available for Java, Node.js, .NET, Python, and Ruby.

Kinesis Data Firehose

Kinesis Data Firehose is fully managed, unlike Kinesis Data Streams. Kinesis Data Firehose will automatically scale to match the throughput of your data and requires no administration.

Firehose can't send to a custom application. We can only send our data to Amazon S3, Amazon Redshift, Amazon Elasticsearch Service, HTTP, and some third party vendors that have integrations like MongoDB.

Kinesis Data Firehose is near-realtime, data is loaded within 60 seconds of being received. You can specify the **buffer size** and the **buffer interval** to determine how much data needs to be collected or how long to wait before delivering the data. For S3 the buffer size is between 1 and 128 MB and the buffer interval is between 60 and 900 seconds.

You can transform data that passes through Kinesis Data Firehose with a lambda function.

Kinesis Data Analytics

Kinesis Data Analytics lets you apply complicated transformations to a stream of data. These transformations can be applied using SQL or a language that supports Apache Flink.

- Input data must be
  - A Kinesis data stream
  - A Kinesis data firehose delivery stream
- Data can be output to
  - A Kinesis Data Stream
  - Kinesis Data Firehose
  - A lambda hose for post-processing