**AWS Certified Solutions Architect Associate - SAA-C03**

**Exam Blue Print:**

* 130 Minutes in Length
* 65 Questions (this can change)
* Multiple Choice
* 70% to pass. Results are between 100 and 1000 with 720 of passing score.
* Certification valid for 3 years.
* Scenario based questions.

**Exam Tips:**

* Don’t spend too much time in hard questions first. Flag them and review later.
* Watch for keyword indicators on certain scenarios.
* Remember the Well-Architected Framework pillars.

**Exam Guide:**

* **Scenario Based:** Large majority of the questions based on real-world scenarios.
  + Pay attention to key words and requirements.
* **Multiple choice questions:** One correct response and three incorrect (distractors)
* **Multiple response:** Two correct responses out of five response options.

**Exam Domains:**

* **Design Resilient Architectures** – 26%
  + Design a multi-tier architecture solution.
    - How to make them scalable
    - Loosely coupled (building flexible, standalone, scalable services.)
  + Multitier (Multiple layers, like front end, backend, etc)
  + Design high available and/or fault tolerant architectures.
    - Know difference between both.
  + Design decoupling mechanisms using AWS services.
    - Message Queues
    - Notification topics.
  + Choose appropriate resilient storage.
    - What are the requirements for the storage
    - How often is accessed
    - etc
* **Design High-Performing Architectures 24%**
  + Identify elastic and scalable compute solutions for a workload.
    - Autoscaling groups
    - Load Balancers for handling or dispersing load
  + Select best high-performing and scalable **storage** solutions for a workload.
  + Select best high-performing and scalable **network** solutions for a workload.
    - Do you need VPN
    - Or Direct connect?
    - Etc
  + Choose best high-performing database solutions for a workload.
    - You need relational database? Like Dynamodb
    - Or Amazon Redshift?
    - etc
* **Design Secure Architectures 30%**
  + Design secure access to AWS resources
    - IAM
    - AWS single sign-on
    - Etc
  + Design secure application tiers
  + Select appropriate data security options
    - Encryption at rest
    - Encryption in transit
    - etc
* **Design Cost-Optimized Architectures 20%**
  + Identify cost-effective storage solutions
    - Different tiers in S3 storage
  + Identity most cost-effective compute and database services
    - Aurora serverless
    - Spot instances for EC2
    - Etc
  + Designing cost-optimized network architectures

**AWS Fundamentals**

**The building Blocks of AWS: Availability Zones and Regions**

**Region:** Is a physical location, like London, New York, Sidney, etc

**Availability Zone (AZ):**

* Is a datacenter withing a region. Each region must have at least 3 or more AZ.
* An AZ could be several datacenters if they are close together, then will be grouped in an AZ.
* Each AZ must be separated between them by many kilometres, although all are within 100 km of each other.

**Edge Locations:**

* Endpoint used for caching content.
* There are many more Edge Locations than Regions.
* Typically consists of CloudFront, Amazon’s CDN

**Who Owns What in the Cloud**

**The Shared Responsibility Model:**

**Customer:**

Responsible for security **IN** the Cloud:

A close-up of a sign

Description automatically generated

**AWS:**

Responsible for security **OF** the Cloud:

A purple rectangular object with white text

Description automatically generated

**Exam Tips:**

To know who is responsible (me or Amazon) ask:

Can I do it myself in the AWS management console?

**If yes**, you are likely responsible:

Security Groups, IAM users, patching EC2 OS, pathing db running in EC2.

**If not:**

Management of datacenters, security cameras, cabling, patching RDS OS, etc.

**Shared responsibility:**

Encryption is shared.

You could want to encrypt a volume, you are responsible of hitting the button and AWS is responsible of doing it.

**Compute, Storage, Databases, and Networking**

To make the exam easier we split things up into four key or core services:

* Compute
* Storage
* Databases
* Networking

**Compute:**

* EC2 🡪 Virtual Machines
* Lambda 🡪 Serverless, no virtual machines
* Elastic Beanstalk 🡪 Provisioning engine, to automate the deployment of my apps in AWS.

**Storage:**

* S3 🡪 To store files, etc.
* EBS, Elastic Block Store 🡪 Virtual hard disk attached to our virtual machines.
* EFS, Elastic File Service. 🡪 A way of store our files centrally.
* FSx 🡪 Fileserver storage for windows servers.
* Storage Gateway (connects on prem environment with cloud storage.)

**Databases:** Think about it as a spreadsheet (exel). It’s a reliable way to store and retrieve information.

* RDS, Relational Database Services
* DynamoDB, AWS non-relational database.
* Redshift, it’s a warehousing technology.

**Networking:**

* VPC’s, Virtual Datacenters in the cloud. Where are resources are going to live.
* Direct Connect, to connect our headquarters or on prem dc to AWS
* Route 53, DNS solution, from registering to pointing them out where we want.
* API Gateway, serverless way of replacing our webservers
* AWS Global Accelerator, a way to accelerating your audiences towards your applications within AWS.

**Exam Guide**

Is quite complex, have ~20 pages.

Tell you information about the exam and a lot of things related.

Have a read when you finish the course.

**Well-Architected Framework: Read this white paper before the exam:** <https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html>

**Six Pillars of the Well-Architected Framework**

1. **Operational Excellence:**
   1. Running and monitoring systems to deliver business value, and continually improving processes and procedures.
2. **Performance Efficiency:**
   1. Using IT and computing resources efficiently
3. **Security:**
   1. Protecting information and systems
4. **Cost Optimization:**
   1. Avoiding unnecessary costs
5. **Reliability:**
   1. Ensuring a workload performs its intended function correctly and consistently when it’s expected to
6. **Sustainability:**
   1. Minimizing the environmental impacts of running cloud workloads.

**Exam Tips:**

**Region**: Physical location in the world. Consists in 3 or more AZ

**Availability Zone, AZ:**  Is one discrete datacenter, each with redundant power and networking.

**Edge Location:** Enpoints for AWS for caching content. Typically CloudFront (CDN)

**Who is responsible:**

Can you touch it by yourself: You

Is managed by AWS: AWS

**Compute:** EC2, Lambda, Elastic Beanstalk.

**Storage:** S3, EBS, EFS, FSx, Storage Gateway

**Databases:** RDS, DynamoDB, Redshift.

**Networking:** VPC, Direct Connect, Route 53, API Gateway, AWS Global Accelerator

Read the whitepaper of well architected framework.

**Identity and Access Management 101**

101 Means that is an introductory topic for a lecture. This is used in UK and the States.

**Ai** = I

**Ei** = A

**Em** = M

Ai Ei Em

**Root Account:**

* Is the email account you used to sign up for AWS.
* It has full administrative access to AWS.
* Is very important to SECURE this account.

**Exam Tips:**

* Enable Multi-factor authentication on the root account
* Create an admin group for your administrators and assign the appropriate permissions to this group.
* Create user accounts for your administrators.
* Add your users to the admin group.

**How do we control Permissions using IAM:** We assign permissions using policy documents, which are made up of JSON.

You will need to know how to read and understand JSON policy documents in the exam. [HERE](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html#access_policies-json)

**IAM Policy Documents:** You can assign them to:

* **Groups** 🡪 you assign the user to a group and then assign the policy to the group.
* **Users** (is not typical to assign policies to users as makes difficult to manage)
* **Roles**

IAM does work at a Global level, any user, role, etc is created globally in all the regions of AWS.

**Apply or Create Policies:**

There are more than 1000 policy templates, or you can create your own one (which won’t have the orange box if you create it)

Example: AdministratorAccess policy

It grants access to all the services in AWS, the JSON is very simple:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "\*",

"Resource": "\*"

}

]

}

Allow everything to all the resources.

**Exam Tips:**

* Assign permissions using IAM policy documents consisting of JSON
* In the exam you will have complicated S3 (could be other service) policy documents and will need to know that they do.
* Pick random policies and look how the JSON works.

**Permanent IAM Credentials:**

**User** = Human being. Should be a physical person.

**Groups**: We should group our users into Groups by Job Function (Admins, Marketing, BE Developers, etc)

**Roles:** For Internal usage within AWS. Allows a part of aws to access another part of aws. Like allowing an EC2 to access a S3 bucket.

**Best Practices:**

Inherit the permissions (Policies) from Groups instead of single users.

You can assign to users directly but is hard to manage as people comes and goes to the organization, or you have contractors, etc.

Is much easier to apply an IAM policy document to a group and the users in that group will inherit those permissions.

Never share user accounts.

**Principle of Least Privilege:**

Assign a user only the minimum number of privileges they need to do their job.

Example:

Sysadmin group will need full access to the AWS platform but not Finances or HR department.

If there is no Allow explicitly this means it’s a Deny.

**Tips when creating a user:**

\* Add tags, tags are key value pairs

Use these ones:

Department: Front End Developer

Employee\_ID: 123456

Now each time this user creates a resource, the resource will be tagged with this information and will be much easy to track who did it.

\* Policy type Job function

AWS already have a few job functions policies set up, like Network Admin, Data Scientist, Support User, etc.

For instance: PowerUserAccess 🡪 Can use everything but can NOT create users and groups.

**Once created the user:**

It will give you the option so send the sign-in instructions via email.

You will see the url the user has to use to access the console (aws gui)

You will see the user and password there too.

And will give you the option to download the password. You won’t be able to retrieve this password again so have this in mind.

**Account Settings:** IAM – Access Management – Account Settings

You manage there the password policies, example:

* How much characters the password has to have.
* Capital Letters and/or lowercase?
* Do you want to use special characters?
* Password expiration
* etc

**Created user without group:** Could be a question for the exam.

If you create a user without assigning him a group the only thing he will be able to do is to change its own password.

IAM – Users – test\_user – Permissions: Permissions policies 🡪 it will have only “

IAMUserChabgePassword” policy applied.

**Programmatic Access:** To use the command line.

IAM – Users – my\_user – Access key 1: Create access key – Command Line Interface (CLI) – Next – add tag if you want – Create

It will create an user and password:

Access key = Username

Secret access key = Password

This access key / secret access key is SEPARATED from the access to the AWS Gui (console).

**Identity providers:** Allow us to connect IAM to things like Active Directory.

IAM –Access Management -- Identity providers

* SAML: To connect AD
* OpenID Connect: To use it with services such as Google, Salesforce, etc.

Possible Exam question: You need to use the same SSO (Single sign-on) that you are using to login in into your windows environment in order to access AWS Management Console, what would you do?

I will add an Identity provider with the provider type SAML.

**Explicit Deny:** When you edit a policy by yourself.

An explicit deny will always override any allow in any other policy.

So if there is an explicit deny, IT WILL ALLWAYS DENY IT, doesn’t matter if there is another policy Allowing the same resource.

Let’s say you have two policies applied to your user:

ec2-admin

all-admin

In both you can see/run/stop ec2 instances, but if you edit manually the policy applied to this user:

IAM – Users – user1 – Permissions – select ec2-admin for example – Edit – (it will redirect you to IAM – User Groups – ec2-admin) and you change the Effect *allow* for *deny*

{

"Version": "2012-10-17",

"Statement": [

{

"Action": [

"ec2:Describe\*",

"ec2:StartInstances",

"ec2:StopInstances"

],

"Resource": "\*",

**"Effect": "Allow" 🡪 before the change.**

},

**"Effect": "Deny" 🡪 after the change.**

Now we have two policies, ec2-admin (deny) and all-admin (allow), both applying to actions over ec2 instances.

If one says allow and the other says deny for the same resource it will DENY the use of that resource.

**Exam Tips:**

* **IAM is universal:** It does not apply to regions, is created across the world globally.
* **The root Account:** The account is created when you first set up your AWS account and which has complete admin access. Secure it as soon as possible and DO NOT use it to log in day to day.
* **New Users:**  No permissions when first created, we must add the permissions manually or add the user to a group. The user will have only the “IAMUserChabgePassword” policy.
* Access key ID and Secret access keys are not the same as usernames and passwords, are used for programmatic access to the AWS console. You only will see the access key and the password once, when you create it, so make sure you saved it.
* **Always set up password rotations:** You can create your own password rotation policies.
* **Identity Federation / IAM Federation:** Identity Providers: To Single Sign-On (SSO) into AWS Management Console using the user/password of another provider, like Active Directory, Salesforce, Google, etc. It uses these Protocols:
  + **SAML:** Security Assertion Markup Language, MS Active Directory
  + **OpenID Connect:** Salesforce, Google, etc
* **Explicit Deny:** If you have two policies applying to the same, full\_access and sys\_admin for instance, if you deny a policy in one of them, it will deny the same in the other.

**S3**

**S3 Is object-based storage:** Manages data as objects rather than in file systems or data blocks.

* Can NOT be used to run an OS or Database
* Upload any file type you can think of to S3
* Examples: Photos, videos, code, documents, and text files.

**S3 Basics:**

* Unlimited storage.
* Objects up to 5 TB in size.
* Buckets: Are the folders of S3, is where the files are stored.

**Working with S3:**

* **Universal Namespace:** The name of the buckets has to be globally unique.
  + **Naming**:
    - https://Bucket-name.s3.Region.amazonaws.com/key-name
    - <https://polla_99.s3.us-east-1.amazonaws.com/my_file.txt>
* **Successful code:** If you upload a file successfully you will receive a code 200.

**Key-Value Store:**

* **Key**: Name of the object: my\_file.txt
* **Value**: The data itself, which is made of a sequence of bytes.
* **Version ID**:To store multiple versions of the same object (file).
* **Metadata**: Is data about data. Examples of metadata in s3:
  + The content type.
  + The last time the file was modified.
  + Etc

**Highly Available and Highly Durable:**

* **Built for Availability:** Built for 99.95% - 99.99% service availability, depending on the s3 tier.
* **Designed for Durability:** Designed for 99.999999999% (11 nines) durability for data stored in s3. You should NOT lose any object stored in S3.

**S3 Standard:** The default version of S3 when you store your objects.

* **High Availability and Durability:**
  + Data stored redundantly across multiple devices in multiple facilities (>= AZs):
    - 99.99% availability
    - 99.999999999% (11 nines) durability
* **Designed for Frequent Access:** Like reading and writing every second, or hour or day for insntance.
* **Suitable for Most Workloads:**
  + The default storage class
  + Use cases include websites, content distribution, mobile and gaming applications, and big data analytics.

**Tiered Storage:**

S3 offers a range of storage classes designed for different use cases.

**Lifecycle Management:**

Define rules to automatically transition objects to a cheaper storage tier or delete objects that are no longer required after a set period of time.

**Versioning:**

All versions of an object can be retrieved, including deleted objects.

**Securing your Data:**

* **Server-Side Encryption:** You can set default encryptions o a bucket to encrypt all the new objects when they are stored in the bucket.
* **ACL:** Define which AWS accounts or goups are granted access and the type of access. You can attach S3 ACLs to individual objects withing a bucket.
* **Bucket Policies:** This specify what actions are allowed or denied, example: Allow user Alice to PUT but not DELETE objects in the bucket. Are JSON policies attached to buckets.

**Data Consistency Model:** Strong Read-After-Write Consistency.

As soon as you written something to S3 it will be immediately available.

* **After a successful** write of a new object (PUT) or an overwrite of an existing one, any subsequent read request immediately receives the latest version of the object.

* **Strong Consistency:** For Lists operations, so after a write you can immediately perform a listing of the objects in a bucket with all changes reflected.

**Exam Tips:**

* Object Based storage that allows to upload files
* Files from 0 bytes to 5 TB.
* Not suitable for OS or DB Storage. Only flat or static files.
* Unlimited Storage.
* Files stored in buckets.
* S3 is a universal namespace (you cannot have two buckets with the same name)
* Successful CLI or API uploads will generate an HTTP 200 status code.
* S3 Object tips:
  + Key: The object name, like my\_file.txt
  + Value: The data itself, which is made up of a sequence of bytes.
  + Version ID: Allows to store multiple versions of the same object.
  + Metadata: Data about the data you are storing, like last modified, etc.

**Securing your bucket with S3 Block Public Access:**

**ACL:** Object ACLs work on an individual object level.

You can make an individual file public or readable, you can adjust the permissions so certain AWS users could access and read that file or download it or write to it, or delete it, and then deny the right of other users to do that.

You can do that on an individual object or file level.

**Bucket Policy:** Work on an entire bucket level.

**Exam Tips:**

* **Buckets are private by default:** When you create an S3 bucket is private by default, including all objects within it. You must allow public access on both the bucket and its objects in order to make the bucket public.
* **Objects ACLs:** You can make individual objects public using object ACLs.
* **Bucket Policies:** You can make entire buckets public using bucket policies.
* **HTTP status code:**  When you upload an object to S3 and it’s successful, you will receive an HTTP 200 code**.**

**Hosting a Static Website using S3:**

**Static websites** don’t change, is basically html.

**Dynamic websites,** such as those that require Databases connections and can not be hosted in S3.

**Example:**

You can use a static S3 website to host a movie trailer and the websites him self’s on S3.

That way you can handle the load, you don’t have to worry about load balancers, capacity planning, etc, S3 will do all for us.

S3 scales automatically, so many enterprises will put static websites on S3 when they think there is going to be a large number of requests, like for a movie preview.

**Exam Tips:**

* **Bucket Policies:** Make entire buckets public using bucket policies.
* **Static Content:** Use S3 to host static content only (not dynamic).
* **Automatic Scaling:** S3 scales automatically with demand.

**Versioning Object in S3:**

**What is versioning:** You can enable versioning in S3 so you can have multiple versions of an object within S3.

**Advantages of versioning:**

* **All versions** of an object are stored in S3. This includes all writes and even if you delete an object.

* **Backup:** Can be a great backup tool.
* **Cannot Be Disabled:** Once enabled, cannot be disabled, only suspended.
* **Lifecycle Rules:** Can be integrated with lifecycle rules.
* **Supports MFA:** Can support MFA authentication.

**Important:** Old versions are not publicly available.

If you have a new version of an html code, for instance, and the code is public, and you go to the old version of the code and hit the link to see the public website, you WON’T be able.

Only the last version is publicly available.

**Delete an object:**

The object will be removed from the GUI but all the versions will be there.

You only have to enable “Show versions” in the bucket and they will appear.

A Type: “Delete marker” will appear showing that you deleted that object, but all the versions will be still available and restorable.

**Restore a deleted object:**

You have to delete the “Delete marker”.

Once this is done, the deleted object will appear again in the GUI, with all the versions.

**Exam Tips:**

* **All versions** of an object are stored in S3. This includes all writes and even if you delete an object.

* **Backup:** Can be a great backup tool.
* **Cannot Be Disabled:** Once enabled, versioning cannot be disabled, only suspended.
* **Lifecycle Rules:** Can be integrated with lifecycle rules.
* **Supports MFA**, typical exam question: How do you protect objects of deletion:
  + Enable Versioning
  + Enable MFA

**S3 Storage Classes**

**S3 Standard:**

* High Availability and Durability
  + Data is stored redundantly across multiple devices in multiple facilities >= 3AZs
    - 99.99% Availability
    - 99.999999999% (11 nines)
  + Designed for frequent accesses data
  + Suitable for most workloads
    - The default storage class
    - Use cases include websites, content distribution, mobile and gaming applications and bit data analytics.

**S3 Standard-IA:** Infrequent Access. Designed for infrequently accessed data.

* **Rapid Access**: Used for data that is accessed less but requires rapid access when needed.

* **You pay to Access the data:** There is a low GB storage price and a low GB retrieval fee.
* **Use Cases:** Great for long term-storage, backups and as data store for disaster recovery files.
* 99.9% Availability
* 99.999999999% (11 nines)

**S3 One Zone-Infrequent Access:** Data is stored redundantly within a single AZ.

For Non-Critical data. To save money.

* Costs 20% less than regular S3 Standard-IA
* Great for long-lived, infrequently accessed, non-critical data.
* 99.5% Availability
* 99.999999999% (11 nines)
* You can afford to lose the data, but you need to minimize costs

**S3 Intelligent-Tiering:** When you don’t know whether you data will be accessed frequently or infrequently

* Uses AI and machine learning to figure out how frequently you access your data and then will store on the storage class that’s going to save you most money based on your access patterns.
* Automatically moves your data to the most cost effective tier based on how frequently you access each object.
* 99.9% availability.
* 99.999999999% (11 nines) durability.
* Good for Optimizing Costs: Monthly fee of $0.0025 per 1000 objects.

**Glacier Options:**

* You pay each time you access your data.
* Use only for archiving data.
* Glacier is cheap storage.
* Optimized for data that is very infrequently accessed.
* 99.99% availability.
* 99.999999999% (11 nines) durability.
* Retrieval fee apply for all archive classes.
* **Glacier Instant Retrieval:** Provides long term data archiving with instant retrieval time for your data, examples like:
  + Recovery of production website or database.
* **Glacier Flexible Retrieval:** No require immediate access but needs the flexibility to retrieve large sets of data at no cost, such:
  + Backup or disaster recovery.
  + Can be minutes or up to 12 hours.
  + You only use this if you can wait up to 12 hours to access the data.
* **Glacier Deep Archive:**
  + Cheapest storage.
  + Designed to retain data sets for 7-10 years or longer to meet regulatory compliance.
  + The standard retrieval time is 12 hours.
  + Bulk retrieval time is 48 hours.

A screenshot of a computer screen

Description automatically generated

**Exam Tips:**

A screen shot of a blue and white screen

Description automatically generated

**Lifecycle Management with S3:**

**What is:** Automates moving your objects between the different storage tiers, thereby maximizing cost effectiveness.

It’s a way of automating moving your objects to save you money.

**Use it with versioning:** Move different versions of objects to different storage tiers.

**Exam Tips:**

* Automates moving objects between different storage tiers.
* Can be used in conjunction with versioning.
* Can be applied to current versions and previous versions.

**S3 Object Lock and Glacier Vault Lock:**

Will be in the exam

**S3 Object Lock:** You can use it to store objects using a write once, read many (WORM) model. It can help prevent objects from being deleted or modified for a fixed amount of time or indefinitely.

You can use it to meet regulatory requirements that require WORM storage or add an extra layer of protection against object changes and deletion.

**Modes:**

**Governance Mode:** Only some users can overwrite or delete.

Users cannot overwrite or delete an object version of alter its lock settings unless they have special permissions.

With governance mode, you protect objects against being deleted by most users, but you can still grant some users permissions to alter the retention settings or delete the object if necessary.

**Compliance Mode:** Nobody can overwrite or delete.

A protected object version cannot be overwritten or deleted by any user, including the root user in your aws account.

When an object is locked in compliance mode, its retention mode can’t be changed and its retention period can’t be shortened.

Compliance mode ensures an object cannot be overwritten or deleted for the duration of the retention period.

**Retention Periods:**

A retention periods protects an object version for a fixed amount of time.

When you place a retention period on an object version, S3 stores a timestamp in the object version’s metadata to indicate when the retention period expires.

After the retention period expires, the object version can be overwritten or deleted unless you also placed a legal hold on the object version.

**Legal Hold:** Can be placed and removed by the user with permissions.

S3 Object Lock also enables you to place a legal hold on an object version. Like a retention period, a legal hold prevents an object version from being overwritten or deleted. However, a legal hold doesn’t have an associated retention period and remains in effect until removed.

Legal holds can be freely placed and removed by any user who has the:

S3:PutObjectLegalHold permission.

**Glacier Vault Lock:**

Allows you to easy deploy and enforce compliance controls for individual S3 Glacier vaults with a vault lock policy.

You can specify controls, such as WORN, in a vault lock policy and lock the policy from future edits.

Once locked, the policy can no longer be changed.

**Exam Tips:**

* Use S3 Object Lock to store objects using write once, read many (WORM) model.
* Object Lock can be on individual objects or applied across the bucket as a whole.
* Object Lock comes in two modes:
  + Governance mode: Some users can modify or delete.
  + Compliance mode: No one can modify or delete, not even the root account.
* S3 Glacier Vault Lock: you can deploy and enforce compliance controls for individual S3 Glacier vaults with a vault policy.
  + You can specify controls, such as WORM, in vault lock policy and lock the policy from future edits. Once locked, the policy can no longer be changed.
* If you see WORM without S3 before it means that is talking about Glacier.

**Encrypting S3 Objects:**

Will be asked in the exam

**Types of Encryptions:**

**Encryption in Transit:** To avoid someone to sniff what you send over internet or other networks you encrypt the communication.

* SSH
* VPN
* SSL/TLS
* HTTPS

**At Rest:** You encrypt your files in your hard drive, database, or cloud storage. Then if someone stole the hard drive its content will be encrypted. This is encryption at rest.

**Encryption in Transit:** You send it encrypted. We do it using:

* SSL/TLS
* HTTPS (443)

**Encryption at Rest: Server-Side Encryption** Is when the object is in your server or in S3 Anything you put in S3 will be encrypted by default.

* **SSE-S3:** S3-managed keys, using AES 256-bit encryption.
  + AWS manages all the encryption and decryption for me.
  + The most common type of encryption and the easiest to use.
* **SSE-KMS:** AWS Key Management Service-managed keys.
  + You work with AWS with your own key management using the key management service (KMS).
  + This gives you more options, like key rotation, audit log, granular access.
* **SSE-C:** Customer-provided keys.

**Encryption at Rest: Client-Side Encryption** You encrypt the files yourself before you upload them to S3.

* You encrypted your objects on your desktop, for example, and then you uploaded them to S3.
* You control the encryption and decryption and all the objects inside S3 are encrypted.

**Encryption by default:**

All objects you put in S3 buckets will be encrypted SSE-S3 by default.

**Enforcing Server-Side Encryption for S3 Uploads:**

You can create bucket policies that deny any S3 PUT request that doesn’t include encrypted objects, it doesn’t include the encryption parameter in the request header.

You can enforce encryption using bucket policies.

**Exam Tips:**

* **Encrypt in Transit:**
  + SSL/TLS
  + HTTPS
* **Encrypt at Rest:** Server Side Encryption (encrypt the hard drive)
  + SSE-S3 (AES 256-bit)
  + SSE-KMS
  + SSE-C (customer)
* **Client-Side Encryption**
  + You encrypt the files yourself before you upload to S3
* **Enforcing Encryption with a Bucket Policy**
  + A bucket policy can deny all PUT requests that don’t include the x-amz-server-side-encryption parameter in the request header.

**Optimizing S3 Performance:**

**S3 Prefixes:** Is the folders before your object and after the bucket name.

my\_documents/**studies/maths**/exam.docx

You have the bucket called: my\_documents

Inside this bucket you have the folder: studies

And inside you have the folder: maths

And inside you have the object (file) exam.docx

The prefix is: studies/maths

Another example:

my\_bucket\_name/folder1/subfolder1/myfile.jpg

folder1/subfolder1/ 🡪 this is the S3 prefix.

**S3 Performance:** S3 has extremely low latency, 100-200 milliseconds.

It has high numbers of requests per second per prefix:

3,500 PUT/COPY/POST/DELETE

5000 GET/HEAD

**Get better performance:** The more S3 Prefixes you have the higher performance you are going to get.

Spread your reads across different prefixes:

* 2 x S3 Prefixes = 11,000 requests per second
  + Folder1 and Folder2
* 4 x S3 Prefixes = 22.000 requests per second
  + Folder1, Folder2, Folder3 and Folder4

**S3 Limitations when using KMS:** KMS comes with built-in limits and therefore will count on my KMS quota.

* If you are using SSE-KMS to encrypt S3 you will have KMS limits.
* When you upload a file, you will call GenerateDataKey in the KMS API.
* When you download a file, you will call Decrypt in the KMS API.

KMS Requests Rates:

* Uploading/Downloading will count in the KMS quota.
* Currently, you cannot request a KMS quota increase.
* Region-specific, however, its either 5,000 – 10,000 or 30,000 requests per second.

Possible exam question:

* If you get an encryption question you may be interested in using the default S3 encryption (SSE-S3)
* If you are asked to troubleshoot why encryption doesn’t work, it may be a KMS reached quota matter.

**Uploads: S3 Multipart Uploads:**

* Recommended for files over 100 MB
* Required for files over 5 GB
* Parallelize uploads (increases efficiency) You split into parts a big file and you upload all the parts at the same time.

Possible exam scenario:

How you can get a better performance in S3 uploads, consider multi part uploads.

**Downloads: S3 Byte-Range Fetches:**

* Parallelize downloads by specifying byte ranges.
* If there is a failure in the download, it’s only for a specific byte range.

Can be use to:

* Speed up downloads
* To download partial amounts of the file (e.g., header information)

**Exam Tips:**

* **Prefix**: my\_documents/**studies/maths**/exam.docx
* You can also achieve a high number of requests per second:
  + 3,500 PUT/COPY/POST/DELETE
  + 5000 GET/HEAD
* You can get better performance by spreading your reads across different prefixes.
* **KMS:** Have in mind KMS limitation quotas.
  + Uploading/downloading will count towards KMS quota
  + Region specific, however, its either 5,000 – 10,000 or 30,000 requests per second.
  + Cannot request KMS quota increase.
* **Multipart Uploads:**
  + Using it increase performance when uploading to S3.
  + Should be used for any file over 100 MB and must be used for over 5 GB.
* **S3** **Byte-Range Fetches** (Downloads)
  + Increases performance downloading files from S3.
  + Split your file in several parts and download them at the same time.

**Backing Up Data with S3 Replication:**

Used to be called Crossed Region Replication but now you can do it in the same region as well, so it was renamed to S3 Replication.

**What is it:**  Way to replicate objects from one bucket to another.

* Versioning must be enabled on both the source and destination buckets.
* Objects in an existing bucket are not replicated automatically.
  + Once replication is turned on, all subsequent updated objects will be replicated automatically.
* Deleted markers are not replicated by default.

Possible exam scenario:

You have S3 buckets in Japan and you have earthquake problems there, what can you do to do a backup of your S3 objects?

Use S3 Replication. You can replicate them into Australia region or something similar.

**Exam Tips:**

* You can replicate objects from one bucket to another.
* Objects in an existing bucket are not replicated automatically, but you have the option to do it.
* Delete markers are not replicated by default.
* You need to enable versioning in source and destination buckets.

**EC2**

**Elastic Compute Cloud**: Secure, resizable compute capacity in the cloud.

* Is a virtual machine hosted in AWS instead of your DC.
* Designed to make web-scale cloud computing easier for developers
* The capacity you want when you need it.
* You are in complete control of your instances.
* Multi-tenant virtualization. You share hardware (CPU, RAM, etc) with other AWS customers. Avoid this using Dedicated Hosts instances type.

**Game changer:** Was introduced in 2006 and changed the industry overnight.

* Pay only for what you use.
* Select the capacity you need right now. Grow and shrink when you need.

**Before EC2:**

* Companies had to buy hardware they would need in the future making it much more expensive.
* They had to wait weeks even months to have it (in Rackspace was within10 to 20 working days.)

**Now with EC2:**

* It is provisioned immediately using an API call.
* You only need to provide credit card details and you are in.
* In 5 – 10 minutes you will have your web/db servers up and running.

**Pricing Options:**

* **On Demand:** Pay by the hour or the second, depending on the type of instance you run.
* **Reserved:** Reserved capacity for 1 or 3 years. **Up to 72%** discount on the hourly charge.
* **Spot:** Purchase unused capacity at a discount of **up to 90%.** Prices fluctuate with supply and demand.
* **Dedicated:** A physical EC2 server dedicated for your use. The most expensive option.

**Where do you use the pricing options:**

**On Demand:**

* **Flexibility:** Low cost and flexibility of EC2 without any upfront payment or long-term commitment.
  + Like when you build a website from scratch and want to see how it looks like.
* **Short-Term:** Applications with short-term, spiky, or unpredictable workloads that cannot be interrupted.
  + Like when you launch an app for the first time and you don’t know if will have a lot of traffic or not.
* **Testing the Water:** Applications being developed or tested on EC2 for the first time.
  + Test and dev purposes

**Reserved:** The opposite of On Demand.

* **Predictable Usage:** Applications with steady state or predictable usage.
* **Specific Capacity Requirements:** Applications that require reserved capacity.
  + You know your db or web servers are going to need 4 vCPUs and 16 G for instance.
* **Pay up Front:** You can make upfront payments to reduce the total computing costs even further.
  + The more you pay upfront the more discount you can get.
* **Standard RIs:** Up to 72% off the on-demand price if you pay 3 years upfront and you pay all upfront.
* **Convertible RIs:** Up to 54% off the on-demand price. Has the option to change to a different RI type equal or greater value.
* **Scheduled RIs:** Launch within the time window you define. Match your capacity reservation to a predictable recurring schedule that only requires a fraction of a day, week, or month.
  + Example: You are running an intranet inside your organization with 2000 seats. Every day at 9 am the people log in in the intranet for the first time. You will need extra capacity to handle 2000 people login in, but after that you can scale down as the rest of the day you won’t need extra capacity anymore.
* **Operate at Regional level:** If you are in us-east-1 you can only do the reservation in that Region, you cannot move it to another Region, like London or any other.
* **Super Flexible:** Can be used with other services like Lambda and Fargate.

**Spot Instances:** The price is moving like a stock market. You set the price where you want to run your instances. When it hits that price, your instances will run. Will terminate when the price changes.

* **Use cases:** when you have applications that have flexible start and end times.
* **Cost Sensitive:** Applications that are only feasible at very low computing prices.
* **Urgent Capacity:** Users with an urgent need for large amounts of additional computing capacity.
* **Examples:** 
  + Image rendering
  + Genomic sequencing.
  + Algorithmic trading engines.

**Dedicated Hosts:**

* **Compliance:** Regulatory requirements that may not support multi-tenant virtualization.
  + Banks
  + Government regulations

* **On-Demand:** Can be purchased on demand (hourly). Will be expensive.
* **Licensing:** Great for licensing that does not support multi-tenancy or cloud deployments.
  + Legacy Microsoft licensing
  + Oracle licensing.
* **Reserved:** Can be purchased as a reservation for up to 70% off the on-demand price.

**AWS Pricing Calculator:** Allows you to figure out what your infrastructure would look like on AWS.

Here the link: <https://calculator.aws/#/>

Exam question: You need to estimate the cost of a move to AWS what should you use?

AWS Pricing Calculator.

**Exam Tips:**

* EC2 is like a VM, hosted in AWS instead of your own data center.
  + Select the capacity you need right now.
  + Grow and shrink when you need.
  + Pay for what you use.
  + Wait minutes, not months.
* **Pricing Models:**
  + **On-Demand:** Pay by the hour or the second, depending on the type of instance you run. Great flexibility.
  + **Reserved:** Reserved capacity for 1 or 3 years. Up to 72% discount on the hourly charge. Great if you have known, fixed requirements.
  + **Spot:** Purchase unused capacity at a discount of up to 90%. Prices fluctuate with supply and demand. Great for applications with flexible start and end times.
  + **Dedicated:** A physical EC2 server dedicated for your use. Great if you have server-bound licenses to reuse or compliance requirements.

**Launching an EC2 Instance:**

**Security Group**: Is like a virtual firewall. You add rules to allow specific traffic in or out.

**AWS Command Line:**

**AWS Management Console / AWS Console:** Is the GUI.

Is the website in which you have all the AWS services and resources. You can create and remove them with the mouse.

**AWS Command Line:** Are the AWS commands (aws s3 ls). You run them via CLI.

To have programmatically access (CLI) to AWS resources.

* Installed by default in Amazon Linux AMI.
  + NOT installed in the other Linux OS. You can download it and use it on your PC, Mac, etc.
* **You need to configure it:**  Once aws command line is installed you need to create an access key and a secret access key to access the AWS resource.
  + **To configure the credentials:** Go via cli to the machine in which is installed:
    - Type: *aws configure*
    - Will ask for access key and secret access key: Generate them.
      * Go to AWS Console: IAM – User groups – Create group – Call it: Admin-S3 – Attach permissions: S3 – AmazonS3FullAccess – Create Group
      * Create User: IAM – User – Create user – name it, DON’T grant it access to AWS Console. – Add to Admin-S3 group – Create user.
      * Create Security Credentials: IAM – Select the user – Security credentials – Create access key – select Command Line Interface – Create Access key. **Copy and/or download the keys** because you cannot see the secret key again.
      * Go to CLI: aws configure – paste access key and secret access key, leave default region and default output, and that’s it. You now can type AWS commands from your CLI.
* **What can you do with it:**
  + To interact programmatically with AWS resources.
  + You can create resources using a script run in your local machine.

**Exam Tips:**

* **Least privilege:** Always give your users the minimum amount of access required to do their job. Like in the example above, we gave S3 admin access only.
* **Create IAM Groups:** And assign your users to those groups.
  + Group permissions are assigned using IAM policy documents.
  + Your users will inherit the permissions of the group they are assigned.
* **Secret Access Key:**
  + You generate it to access via cli to aws.
  + This is not your password to access AWS Console (GUI)
  + Once generated will be displayed only once, so copy/download it. If not you will need to generate another key pair.
  + Don’t share key pairs. Each developer/sysadmin should have their own access key ID and secret access key.
* **AWS CLI:** Supports Windows, Linux and MacOS.
  + You can install it on your local device, laptop, pc, etc.
  + You can use it directly from your EC2 instance (install it if is not there)

**AWS SDK:**

**AWS Software Development Kit:** Language-specific APIs (set of libraries). You will have an AWS SDK for a variety of programming languages.

* Enables you to access AWS services programmatically but you don’t use it with your terminal like AWS CLI. You embedded within your application.

* Support many different programming languages.

**Using Roles:**

**What is an IAM Role:** Is an identity you can create in IAM that has specific permissions.

You grant temporary permissions to an AWS service to access another service.

* Similar to an IAM User, it has permission policies (json) that determine what can and cannot do.
  + Instead of being uniquely associated with one person, a Role is intended to be assumable by anyone (service or person) who needs it.
* Much safer than Credentials (id access key / secret access key) as you don’t store any credentials in the service (EC2, Lambda, etc).
* Temporary Access: It lasts one hour.

Example: Create IAM Role: S3\_role to connect EC2 to S3.

When you create the EC2 instance, you attach it to the S3\_role and once deployed, the instance will be able to connect to S3 automatically via AWS Command Line without having to do the aws configure and paste id access key and private one.

**What Roles can also do:**

* Can be assumed by people, AWS services (EC2, Lambda, etc), or other system level accounts.
  + Cross-Account Access: One AWS account can interact with resources in other AWS accounts.

**Add Role to EC2 instance:** You need to create the role but is quite easy.

On your EC2 instance – Actions – Security – Modify IAM role – select the role you want.

Now you can access to the EC2 instance and type aws commands, it will work straight away, without having to add any credentials.

**Exam Tips:**

* A Role is the preferred option for security reasons rather than using Access/Secret Keys.
* Avoid hardcoding your credentials (access/secret keys): You have programmatic access without using access key id and secret access keys.
* Policies: The policies control the role’s permissions.
* Policies Updates: You can update a role policy and will take effect immediately.
* Attaching and Detaching: Attach and detach roles to running EC2 instances without having to stop or terminate those instances.

**Security Groups and Bootstrap Scripts:**

**What are Security Groups:** Are virtual firewalls for your EC2 instances. By default everything is blocked.

* To let everything in: 0.0.0.0/0
* You will need to open the ports for ssh/rdp/http or whatever you need.
* For web servers open all to http/https but not for ssh/rdp or other critical ports to avoid bruteforcing them from the attackers.

**Bootstrap Script, User Data:** It’s a script that runs with your instance first run.

* Will run at root level.
* It adds more time to boot the instance but allows you to automate the installation of applications.

#! /bin/bash 🡪 #! Shebang. /bin/bash 🡪 interpreter, bash in this example.

yum -y install httpd. 🡪 install Apache

yum service httpd start. 🡪 start Apache

**Exam Tips:**

* Changes to Security Group take effect immediately.
* You can have any number of EC2 within a security group.
* You can have multiple security groups attached to EC2 instances.
* All inbound traffic is blocked by default.
* All outbound traffic is allowed.

* **Bootstrap Script:** Script that runs when the instance first runs. It passes user data to the EC2 instance and can be used to install applications like webservers, to do updates, etc.
* **User Data:** bootstrap script is called “User Data”

**EC2 Metadata and User Data:**

**What is EC2 Metadata:** Is data about my EC2 instance.

This can include:

* Private and Public IP
* Hostname
* Security group
* Mac address.
* Etc

**How to retrieve Metadata:**

* Before was only necessary to do: curl PUBLIC\_IP /latest/metadata.
* Now is much more complicated, won’t be asked in the exam.

**Exam Tips:**

* User data is simply bootstrap script.
* Metadata is data about my EC2 instance.
* You can use bootstrap scripts (user data) to access metadata.

**Networking with EC2:**

You can attach 3 different types of virtual networking cards to your EC2 instances:

* **ENI:** Elastic Network Interface, for basic day-to-day networking.
* **EN:** Enhanced Networking, uses single root I/O virtualization (SR-IOV) to provide high performance.
* **EFA:** Elastic Fabric Adapter, Accelerates High Performance Computing (HPC) and machine learning applications.

**ENI, Elastic Network Interface:**

* IPv4
* Public IPv4 Addresses
* Many IPv6 Addresses
* Mac Addresses
* 1or More Security Groups

* **Use Cases:**
  + Create management network.
  + Use network and security appliances in your VPC.
  + Create dual-home instances with workloads/roles on distinct subnets. (Private network addresses separated)
  + Create a low-budget, high-availability solution.

**EN, Enhanced Networking:** For High-Performance Networking.

Between 10 Gbps – 100 Gbps

* Single Root I/O Virtualization (SR-IOV): Provides higher I/O performance and lower CPU utilization.

* Performance: Provides higher bandwidth, higher packet per second (PPS) performance, and consistently lower inter-instance latencies.
* Depending on your instance type, EN can be enabled using:
  + **ENA**, Elastic Network Adapter: Supports speeds of up to **100 Gbps** for supported instance types.

* + **VF**, Intel 82599 Virtual Function Interface: For old instances, up to **10 Gbps.**
  + **Exam question:** Always use ENA over VF.

**EFA, Elastic Fabric Adapter:** For HPC. It’s a fiver network card.

* A network device you can attach to your EC2 to accelerate High Performance Computing and machine learning applications.

* Provides lower and more consistent latency and higher throughput than the TCP transport traditionally used in cloud-based HPC systems.
* **OS-Bypass:** Much faster and much lower latency. Enables HPC and machine learning applications to bypass the OS kernel and communicate directly with the EFA device. Only Linux, no Windows.

**Exam Tips:**

* **ENI**: For basic networking. Low cost. Possible scenarios:
  + Separated management network from production network.
  + Separated logging network.

* **EN:** For when you need speeds between 10 Gbps – 100 Gbps.
  + Reliable.
  + High throughput.

* **EFA:** HPC and machine learning.
  + OS-Bypass: To bypass the OS for much faster and lower latency. Only Linus.

**Optimizing EC2 Placement Groups:**

**Placement Groups:** They are a way of logically grouping your EC2 instances depending on what it is you want to do.

**Cluster Placement Groups:** Grouping instances within a single AZ.

* Recommended for applications that need low network latency, high network throughput, or both.
* Only certain types of instances can be launched into a cluster placement group.

**Spread Placement Groups:** Group of instances that are each placed on distinct underlying hardware.

* Recommended for applications that have a small number of critical instances that should be kept separate from each other.
* They are spread out instead of being together.
* Used for individual instances.
* **Example**: I don’t want my primary DB to stay in the same place than my backup DB.

**Partition Placement Groups:** Each partition has its own set of racks. Each rack has its own network and power source.

* No two partitions within a placement group share the same racks.
* This allows to isolate the impact of hardware failure within your application.
* EC2 divides each group into logical segments called *Partitions*.
* **Scenario Question:** When you have multiple instances and you want them to have their own dedicated power sources

**Exam Tips:**

* **Cluster Placement Groups:** Grouping in single AZ. Low network latency, high network throughput. For HPC.
  + Can’t span multiple AZ.
* **Spread Placement Groups:** Individual critical EC2 instances. Example: Main DB and slave DB.
* **Partition Placement Groups:**  Multiple EC2. Each logical segment is called *Partitions.*
  + Dedicated rack for each *Partition* to have their own network and power source. Each *Partition* have to be in different racks with different network and power source to avoid failures.
  + HDFS, HBase, and Cassandra.

* Only certain types of instances can be launched in a placement group (compute optimized, GPU, memory optimized, storage optimized).

* You cannot merge placement groups.
* You can move an existing instance into a placement group. Before you move the instance, the instance must be in the stopped state.
* You can move or remove an instance using the AWS CLI or AWS SDK, but you cannot doit via Console yet.

**Solving Licensing Issues with Dedicated Hosts:**

**Reminder:**

**Dedicated Hosts:**

* **Compliance:** Regulatory requirements that may not support multi-tenant virtualization.
  + Banks
  + Government regulations

* **On-Demand:** Can be purchased on demand (hourly). Will be expensive.
* **Licensing:** Great for licensing that does not support multi-tenancy or cloud deployments.
  + Legacy Microsoft licensing
  + Oracle licensing.
* **Reserved:** Can be purchased as a reservation for up to 70% off the on-demand price.

**Exam Tips:**

* **Scenario Question:** Any question about special licensing requirements is for Dedicated Hosts.
* Dedicated hosts allows you to use your existing:
  + Per-socket, per-core, or per-VM software licenses.
  + Including Windows Server, MS SQL Server, and Suse Linux Enterprise Server.

**Timing Workloads with Spot instances and Spot fleets:**

**What are EC2 Spot instances:** Are AWS’s unused EC2 capacity. It is up to 90% discount compared to on-demand prices.

**Spot Prices:** They work with Spot prices, like the stocks market. The price varies depending on the demand of EC2 resources from other customers.

The hourly Spot price varies depending on capacity and Region.

If Spot price goes above your maximum, you have 2 minutes to choose between:

* To stop the instance. It will resume your workload once the Stop price is below your maximum.
* Terminate your instance.

**2 minutes grace period:** Once the Spot price hits your maximum price, AWS will give you 2 minutes to close nicely all the applications and systems in your Spot instances.

So, when you create the Spot instance you add in the User Data (bootstrap script) the instructions to close nicely all your apps and save all you need.

AWS will send an Interruption Notice Metadata that will let you know that your Spot instance is going to be shut down in 2 minutes, then your system can act to close all programs or so.

**When to use Spot instances:** Workloads that you can turn down, like stateless, fault-tolerant, or flexible applications.

Are good for applications such as:

* Big data
* Containerized workloads
* CI/CD
* HPC **Uses to come in the exam,** for what is good Spot instances? For HPC
* and other test and development workloads.

NOT good for:

* Persistent workloads:
  + Web servers
  + Database servers
* Critical jobs

**How to use them:** You must first decide on your maximum Spot price.

The instance will be provided so long as the Spot price is BELOW your maximum Spot price.

Example (not with real prices):

* You are prepared to pay up to 1£ /hour.
* If the Spot price is 0.50£ /hour the EC2 Spot instance/s will be provisioned.
* If the Spot price changes to 1.50£ /hour your Spot instance/s will be stopped or terminated, depending on what you choose.

**When to terminate them:** When we create Spot instances, we create a **Spot Request**.

In the **Spot Request** we specify:

* Maximum price
* Desired number of instances
* Launch specification: AMI, etc.
* **Request type**: One-time | Persistent.
  + If persistent is chosen: Valid from, valid until.

**Request type: One-time:** Once created the Spot request:

1. The instances will be provisioned and launched.
2. As soon as the Spot price goes above our maximum price, the Spot instances will be stopped or terminated (depending on what is configured)

**Persistent Request:** Essentially is looking if is open, active, or disabled.

For the exam you have to know: How do you go in and terminate Spot instances under persistent Spot request?

You go in and you cancel the spot request and then you go in and you terminate your instances.

**Spot Fleets:** Is a collection of Spot instances and (optionally) On-Demand Instances.

Will try and match the target capacity with your price restraints.

It’s a way of meting your capacity within your budget.

* Spot Fleet attempts to launch the number of Spot Instances and On-Demand instances to meet the target.
* The request of Spot instances is fulfilled if there is available capacity and the maximum price you specified in the request exceeds the current Spot price.
* The Spot fleet also attempts to maintain its target capacity fleet if your Spot instances are interrupted.

**Spot Fleets Launch Pools:**

1. Set up launch pools. Define things like EC2 instance type, OS, and AZ.
2. You can have multiple pools, and the fleet will choose the best way of implement depending on the strategy you define.
3. Spot fleet will stop launching instances once you reach your price threshold or capacity desired.

**Spot Fleets Strategies:**

* **capatityOptimized:** The Spot Instances come from the pool with optimal capacity for the number of instances launching.

* **diversified:** The Spot instances are distributed across all pools.
* **lowesPrice:** The Spot instances comes from the pool with lowest price. This is default strategy.
* **InstancePoolsToUseCount:** The Spot instances are distributed across the number of Spot instance pools you specify. This parameter is valid only when used in combination with lowestPrice.

**Exam Tips:**

* Spot Instances save up to 90% of On-Demand prices.
* You can block Spot Instances from terminating by using Spot block.
* Useful for any type of computing where you don’t need persistent storage.
* A Spot Fleet is a collection of Spot Instances and (optionally) On-Demand instances.

**Deploying vCenter in AWS with VMware Cloud on AWS**

**Why Use VMWare on AWS:** VMWare is older than AWS.

* Is used by organizations around the world for Private Cloud deployments.
* Some organizations opt for a hybrid cloud strategy and would like to leverage (aprovechar) AWS services.

**Use Cases:**

* **Hybrid Cloud:** Connect your on-premises cloud to the AWS public cloud and manage a hybrid workload. You don’t know how much EC2 instances are you gonna use, or stuff like this.

* **Cloud Migration:** VERY COMMMON. Migrate your existing cloud environment to AWS using VMware’s built-in tools. From legacy private cloud in my DC to AWS using VMWare built-in tools.
* **Disaster Recovery:** VERY COMMMON. VMWare is famous for its disaster recovery technology. Using hybrid cloud, you can have an inexpensive disaster recovery environment on AWS.
* **Leverage (aprovecha) AWS:** Use over 200 AWS services to update your applications or to create new ones. Things that are not native in VMware.

**How is it deployed:**

* It runs on dedicated hardware hosted in AWS using a single AWS account.
* Each host has two sockets with 18 cores per socket, 512 GB Ram, and 15.2 TB Raw SSD storage.
* Each host is capable of running multiple VMware instances (up to the hundreds)
* Clusters can start with two hosts up to a maximum of 16 hosts per cluster.
  + Amazon ask for a minimum of 2 hosts for availability.
  + These clusters are going to be your vCenter clusters.

**Exam Tips:**

* You can deploy vCenter on AWS Cloud using VMware.
* Perfect solution for extending your private VMware Cloud into the AWS public cloud.

**Extending AWS Beyond the Cloud with AWS Outposts:**

**What is Outposts:** It brings the AWS data center directly to you, on-premises.

Outposts allows you to have the large variety of AWS services in your own data center.

You can have Outposts in sizes such as 1U, 2U servers all the way up to 42U racks and multiple-rack deployments.

**Benefits:**

* Create a hybrid cloud where you can leverage AWS services inside your own data center.
* Fully managed infrastructure. AWS can manage the ifr for you, you don’t need a dedicated team to look after your Outposts infrastructure.
* It brings consistency to the AWS Management Console, APIs, and SDKs into your data center, allowing uniform consistency in your hybrid environment.

**Family Members:**

* **Outposts Rack:**
  + **Hardware**: Available starting with a single 42u rack and scale up to 96 racks.
  + **Services**: Provides AWS compute, storage, database, and other services locally.
  + **Results:** Gives the same AWS infrastructure, services, and APIs in your own data center.
* **Outpost Servers:**
  + **Hardware:** Individual servers in 1U or 2U form factor.

* + **Use Cases:** Useful for small space requirements, such as retail stores, branch offices, healthcare provider locations, or factory floors.
  + **Results:** Provides local compute and networking services.

**Common exam scenario:** Which one of the family members you should be using.

**Process:**

1. **Order:** Log in to the AWS management console, and order your Outposts configuration.

1. **Install**: AWS staff will come on-site to install and deploy the hardware, including power, networking, and connectivity.
2. **Launch**: Using the AWS management console, you can launch instances on your Outpost on-site.
3. **Build:** Start building your on-site AWS environment.

**Exam Tips:**

* Scenario about extending AWS to your datacenter?
  + Think about AWS Outposts.

* AWS Outposts rack for large deployments.
* AWS Outposts servers for smaller deployments.

**Cool CLI commands for bootstrap script:**

#!/bin/bash

sudo apt-get update -y

sudo apt-get install apache2 unzip -y

sudo systemctl start apache2

sudo systemctl enable apache2

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

echo '<html><h1>Bootstrap Demo</h1><h3>Availability Zone: ' > /var/www/html/index.html

curl http://169.254.169.254/latest/meta-data/placement/availability-zone >> /var/www/html/index.html

echo '</h3> <h3>Instance Id: ' >> /var/www/html/index.html

curl http://169.254.169.254/latest/meta-data/instance-id >> /var/www/html/index.html

echo '</h3> <h3>Public IP: ' >> /var/www/html/index.html

curl http://169.254.169.254/latest/meta-data/public-ipv4 >> /var/www/html/index.html

echo '</h3> <h3>Local IP: ' >> /var/www/html/index.html

curl http://169.254.169.254/latest/meta-data/local-ipv4 >> /var/www/html/index.html

echo '</h3></html> ' >> /var/www/html/index.html

sudo apt-get install mysql-server -y

sudo systemctl start mysql

sudo systemctl enable mysql

curl http://169.254.169.254/latest/user-data --> Get the content of the bootstrap script.

**Exercise: Using EC2 Roles and Instance Profiles in AWS**

The exercise shows how to create a Policy, a Role and to attach this role to an EC2 instance.

It shows how to create them via CLI.

**Via CLI** is more complicated and has more parts:

1. It asks to get into a bastion host and run an aws configure
   1. No access/secret keys required, just hit enter.
   2. Add us-ease-1
   3. Add json as type
2. **Create IAM Trust Policy for an EC2 Role**
   1. vi trust\_policy\_ec2.json. 🡪 and paste the code below:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Principal": {"Service": "ec2.amazonaws.com"},

"Action": "sts:AssumeRole"

}

]

}

1. **Create the DEV\_ROLE IAM Role**
   1. aws iam create-role --role-name DEV\_ROLE --assume-role-policy-document file://trust\_policy\_ec2.json
2. **We want to grant this role read access to one of the S3 Buckets** (DEV\_Bucket)**:**

Create an IAM Policy Defining Read-Only Access Permissions to an S3 Bucket

* 1. vi dev\_s3\_read\_access.json
  2. This is the bucket name we are going to use:

cfst-3035-ada8da97f7dff243df4b5f6f2006-s3bucketdev-8rntes2fuokx

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "AllowUserToSeeBucketListInTheConsole",

"Action": ["s3:ListAllMyBuckets", "s3:GetBucketLocation"],

"Effect": "Allow",

"Resource": ["arn:aws:s3:::\*"]

},

{

"Effect": "Allow",

"Action": [

"s3:Get\*",

"s3:List\*"

],

"Resource": [

"arn:aws:s3:::cfst-3035-ada8da97f7dff243df4b5f6f2006-s3bucketdev-8rntes2fuokx/\*",🡪 dev\_bucket

"arn:aws:s3:::cfst-3035-ada8da97f7dff243df4b5f6f2006-s3bucketdev-8rntes2fuokx"

]

}

]

}

/\* Means apply to the bucket and all before it (subfolders)

No /\* means apply only to the bucket

1. **Create IAM Policy referencing the file we just created in step 4:**
   1. aws iam create-policy --policy-name DevS3ReadAccess --policy-document <file://dev_s3_read_access.json>
2. **Attach (associate) the Policy to the Role:**
   1. Get the policy arn created in the step 5.
   2. aws iam attach-role-policy --role-name DEV\_ROLE --policy-arn "arn:aws:iam::015518611331:policy/DevS3ReadAccess"
   3. List the policy attached to the role:
      1. aws iam list-attached-role-policies --role-name DEV\_ROLE
      2. You will see the policy name DevS3ReadAccess and the arn mentioned in the step b
3. **Create instance profile:** This is not correct, but is taking too much time and efforts.
   1. aws iam create-instance-profile --instance-profile-name DEV\_PROFILE
   2. Verify configuration: aws iam get-instance-profile --instance-profile-name DEV\_PROFILE
4. **Add Role to the instance profile:**
   1. Copy the instance ID from the Web Server
   2. aws ec2 associate-iam-instance-profile --instance-id i-0e428b7538cf285c4 --iam-instance-profile Name="DEV\_PROFILE"
   3. Verify: aws ec2 describe-instances --instance-ids i-0e428b7538cf285c4
   4. Check that the arn of the instance is attached to the instance profile:
      1. "Arn": "arn:aws:iam::015518611331:instance-profile/DEV\_PROFILE"
5. **Check if works.** Ssh the webserver and aws s3 ls, didn’t worked in my case, you should be seeing the buckets and shit.

**Summary:**

1. Create the Trust Policy (trust\_policy\_ec2.json)
2. Create the Role (DEV\_ROLE) mentioning the Trust Policy
3. Create the IAM Policy, dev\_s3\_read\_access.json (which allows List and Get to the dev\_bucket)
4. Attach the IAM Policy (dev\_s3\_read\_access.json) to the IAM Role (DEV\_ROLE)
5. Create Instance Profile
6. Add (associate) role to instance profile,
7. Attached the instance profile the EC2 instance.
8. Get into the webserver and type aws s3 ls, it should work, not in my case.

**Create a production role, a permissions policy and an instance profile via Console (GUI):**

* When you create and attach a role to an EC2 instance via GUI the creation and use of the Instance Profile is handled behind the scenes.
* From our side looks like we are only dealing with IAM roles, however aws automatically runs commands behind the scenes to replicate the the steps we did in the previous CLI exercise.

**Role:** What Can I do?

**Instance Profile:** Entity or container that’s used for connecting an IAM Role to an EC2 instance. Think of an instance profile as *who am I?*

IP provide temporary credentials which are rotate automatically.

* **Via AWS Console** (GUI): Generated automatically when you create the Role.
* **Via AWS Command Line**: You have to manually create it.

Trust Policy: We create it via CLI and we have to call it when creating a role via CLI too:

aws iam **create-role** --role-name DEV\_ROLE --assume-role-policy-document file://**trust\_policy\_ec2.json**

Permissions Policy = IAM Policy

**Elastic Block Storage (EBS) and Elastic File System (EFS)**