**EC2**

* Operations, necessary to do as a SysOps Administrator
* Troubleshooting
* Instance Types
* And Launch Modes
* Ami
* CloudWatch + EC2

EC2 Changing an Instance Type

* This only works with instances that have a EBS volume
* Stop the instance
* Instance Settings => Change Instance Type
* Start Instance

**EC2 Placement Groups**

* Sometimes you want control over the EC2 Instance placement strategy
* That strategy can be defined using placement groups
* When you create a placement group you specify one of the following strategies for the group
  + **Cluster**:
    - Clusters instances into a low-latency group in a single Availability Zone
    - All EC2 instances are on the same rack and in the same AZ
    - All of these instances are on the same hardware.
    - Low latency 10Gbps network speeds
    - Pros
      * Great network speeds (10Gbps bandwidth between instances )
    - Cons
      * If the rack fails, or if the hardware fails all the instances fails at the same time
    - Use Case:
      * Big Data job that needs to be completed quickly
      * Application that needs extremely low latency and high network throughput
  + **Spread**:
    - Spreads instances across underlying hardware ( a **Max** of 7 instances per group per AZ) – critical applications
    - The exact opposite of a cluster where it tries to eliminate points of failure
    - All the instances are on different hardware racks
    - Then they are spread out among the different AZs in a region
    - Pros
      * Can span across multiple AZs
      * Reduced risk of simultaneous failure
      * EC2 instances on different physical hardware
    - Cons
      * Limited to 7 instances per AZ per placement group
    - Use Case:
      * Application that needs to maximize high availability
      * Critical Applications where each instance must be isolated from failure from each other.
  + **Partition**:
    - Spreads instances across many different partitions ( which rely on different sets of racks) within an AZ. Scales to 100s of EC2 instances per group ( Hadoop, Cassandra, Kafka )
    - Different partitions are different sets of racks
    - Multiple EC2 instances on a single partition
    - If a rack goes down it will take the EC2 instances on that partition with it.
    - Up to 7 partitions per AZ
    - Up to 100s of EC2 instances per partition
    - The instances in a partition do no share racks with other instances in the other partitions
    - A partition failure can affect many EC2 but won’t affect other partitions
    - EC2 instances get access to the partition information as metadata
    - Use Cases
      * HDFS
      * HBase
      * Cassandra
      * Kafka

Shutdown behaviour

* Shutdown behavior:
  + How should the instance behave when a shutdown has been initiated via the OS
    - Stopped: default
    - Terminated
* This is not applicable from the AWS console or AWS API
* CLI Attribute
  + InstanceInitiatedShutdownBehavior

Termination Protection

* Enable termination protection:
  + To protect against accidental termination in AWS console or CLI
* Exam Tip
  + We have an instance where shutdown behavior = terminate and enable terminate protection is ticked
  + We shutdown the instance from the OS, what will happen
    - The instance will still be terminated

**EC2 Launch Troubleshooting**

* # InstanceLimitExceeded **(Instance Limit Exceeded)** error:
  + If you get this error, it means that you have reached the max limit of instances per region
    - Now limits in AWS are counted in vCPU ( instead of instances)
    - The default limit is 32 (it used to be 20)
    - This is not yet reflected in the exam
  + Initiated by:
    - Launching a new instance or restarting a stopped instance
    - There is a limit based on each region and varies by region as well
  + Solution
    - Request an increase of instances from AWS
    - Or launch the instance in a different region
* # InsuffcientInstanceCapacity **(Insufficient instance capacity)**
  + This error reflects that AWS does not have that much On-Demand capacity in the particular AZ to which the instance is launched
  + **Cause**
    - Occurs when you try to launch an instance or restart a stopped instance, AWS does not currently have enough available On Demand capacity to service your request
  + **Solution**
    - Wait a few minutes to resubmit the request
    - Resubmit the request with fewer instances
    - If you;re launching an instance, submit a new request using a different instance type ( which you can resize at a later stage)
    - If you are launching instances into a cluster placement group, you can get insufficient capacity error.
    - Try creating an On-Demand capacity Reservation, which enables you to reserve Amazon EC2 capacity for any duration.
    - Try purchasing Reserved Instances, which are a long term capacity reservation
* **Instance terminates immediately**
  + Instances go from the pending state to the terminated state immediately after restarting it
  + **Cause**
    - You’ve reached your EBS volume limit.
    - An EBS snapshot is corrupt
    - The root EBS volume is encrypted and you do not have permission to access the KMS key for decryption
    - The instance store-backed AMI that you used to launch the instance is missing a required part ( an image.part.xx file)
  + **Solution**
    - You can use the Amazon EC2 console or AWS Command Line Interface to get the termination reason.
      * **To get the termination reason using the Amazon EC2 console**
        + In the EC2 navigation pane, choose instances, and select the particular instance
        + Then in the description tab, find the reason next to the State transition reason
      * **To get the termination reason using the AWS Command Line Interface**
        + Use the describe-instances command and specify the instance ID.

Aws ec2 describe-instances --instance-id *instance\_id*

* + - * + Review the JSON response returned by the command and not ehte values in the

StateReason (response element)

It will have a “Message”

It will have a “Code”

* + - * **To address the issue**
        + Client.VolumeLimitExceeded: Volume limit exceeded

Reached the EBS volume limit

* + - * + Client.InternalError: Client error on launch

Indicates that the root volume is encrypted and that the instance does not have permissions to access the KMS key that will be used for the decryption

To get permissions to access the required KMS key add the appropriate KMS permissions to your IAM user.

**EC2 SSH trouble**

* Make sure the private key ( pem file ) on your linux machine has 400 permissions, else you will get “Unprotected Private Key File error”
* Make sure the username for the OS is given correctly when logging via SSH, else you will get “host key not found” error
* **Possible reasons for ‘connection timeout’ to EC2 instance via SSH:**
  + SG is not configured correctly.
  + CPU load of the instance is high
* ERRORS
  + “Connection Timed Out” or “Connection refused”
  + “Permission Denied” or “Authentication Failed”
  + “Server Refused Our Key”

**EC2 Instance launch Types**

* On Demand Instances: short workload, predictable pricing
* Reserved: (Minimum 1 year)
  + Reserved Instances: Long workloads
  + Convertible Reserved Instances: Long Workloads with flexible instances
  + Scheduled Reserved Instances: example – every Thursday between 3 and 6pm
* Spot Instances: short workloads, for cheap, can lose instances ( less reliable)
* Dedicated Instances: no other customers will share your hardware
* Dedicated hosts: book an entire physical server, control instance placement

EC2 On Demand

* Pay for what you use ( billing per second, after the first minute)
* Has the highest cost but no upfront payment
* No long term commitment
* Recommended for short-term and un-interrupted workloads, where you can’t predict how the application will behave

EC2 Reserved Instances

* Up to 75% discount compared to On-Demand
* Pay upfront for what you use with long term commitment
* Reservation period can be 1 - 3 years
* Reserve a specific instance type
* Recommended for steady state usage applications ( think database )
* VARIATIONS
* Convertible Reserved Instance:
  + Can change the EC2 instance type
  + Up to 54% discount
* Scheduled Reserved Instances
  + Launch within time window you reserve
  + When you require a fraction of day / week / month

EC2 Spot Instances

* Can get a discount of up to 90% compared to On-demand
* Instances that you can “lose” at any point of time if your max price is less than the current spot price
* The MOST cost-efficient instances in AWS
* **Useful for workloads that are resilient to failure**
  + Batch jobs
  + Data analysis
  + Image Processing
  + …
* **Not great for critical jobs or databases**
* **Great combo: Reserved Instances for baseline + On-Demand & Sport for peaks**

EC2 Dedicated Hosts

* Physical dedicated Ec2 server for your use
* Full control of EC2 instance placement
* Visibility into the underlying sockets / physical cores of the hardware
* Allocated for your account for a 3 year period reservation
* More expensive
* Useful for software that have complicated licensing model ( BYOL – Bring Your Own License)
* Or for companies that have strong regulatory or compliance needs

EC2 Dedicated Instances

* Instances running on hardware that’s dedicated to you
* May share hardware with other instances in same account
* No control over instance placement ( can move hardware after Stop / Start)

Pricing - On-demand is the most expensive

EC2 Spot Instances Requests

* Can get a discount of up to 90% compared to On-demand
* Define max spot price and get the instance while current spot price < max
* If the current spot price > your max price you can choose to stop or terminate your instance with a 2 minutes grace period.
* Other strategy: Spot Block
  + “Block” spot instance during a specified time frame ( 1 to 6 hours) without interruptions
  + In rare situations, the instance may be reclaimed
* **Used for batch jobs, data analysis, or workloads that are resilient to failures.**
* **Not great for critical jobs or databases**

Spot Fleets

* Spot Fleets = set of Spot Instances + (optional) On-Demand Instances
* The Spot Fleet will try to meet the target capacity with price constraints
  + Define possible launch pools: instance type, OS, AZ
  + Can have multiple launch pools, so that the fleet can shoes
  + Spot Fleets stops launching instances when teaching capacity or max cost
* Strategies to allocate Spot Instances:
  + lowestPrice: from the pool with the lowest price ( cost optimization, short workload)
  + diversified : distributed across all pools ( great for availability, long workloads)
  + Capacity Optimized: pool with the optimal capacity for the number of instances
* Spot Fleets allow us to automatically request Spot Instances with the lowest price

**EC2 Instance types – Main ones (RCMIGT2T3)**

* R: applications that needs a lot of RAM –in-memory caches
* C: applications that needs good CPU – compute / databases
* M: applications that are balanced ( think ‘ medium”) – general / web app
* I: applications that need good local I/O ( instance storage ) – databases
* G: applications that need a GPU – video rendering / machine learning
* T2 / T3: burstable instances ( up to capacity )
* T2 / T3: unlimited burstable capacity

Burstable Instances (T2/T3)

* Burstable instances can be amazing to handle unexpected traffic and getting the insurance that it will be handled correctly
* If your instance consistently runs low on credit, you need to move to a different kind of non-burstable instance

T2/T3 Unlimited

* Nov 2017: It is possible to have an “unlimited burst credit balance”
* You pay extra money if you go over your credit balance, but you don’t lose in performance
* Overall, it is a new offering, so be careful, costs could go high if you’re not monitoring the health of your instances
* A burstable performance instance configured as unlimited can sustain high CPU performance for any period of time whenever required. The hourly instance price automatically covers all CPU usage spikes if the average CPU utilization of the instance is at or below the baseline over a rolling 24 hour period or the instance lifetime, whichever is shorter.
* For the vast majority of general-purpose workloads, instances configured as unlimited provide ample performance without any additional charges. If the instance runs at a higher CPU utilization for a prolonged period, it can do so for a flat additional rate per vCPU-hour.
* NOTE
  + T3 instances launch as unlimited by default. If you launch T3 spot instances as unlimited and plan to use them immediately and for a short duration with no idle time for occurring CPU credits, you will incur charges for surplus credits.
  + If the average CPU usage over a 24-hour period exceeds the baseline, you will also incur charges for surplus credits. We recommend that you launch your T3 spot instances in standard mode to avoid paying higher costs.

**EC2 AMIs**

What’s an AMI?

* As we saw, AWS comes with base images such as:
  + Ubuntu
  + Fedora
  + RedHat
  + Windows
  + Etc…
* These images can be customised at runtime using EC2 User data
* But what if we could create our own image, ready to go?
* That’s an AMI – an image to use to create our instances
* AMIs can be built for LInux or Windows machines

Why would you use a custom AMI?

* Using a custom built AMI can provide the following advantages:
  + Pre-installed packages needed
  + Faster boot time ( no need for ec2 user data at boot time)
  + Machines come configured with monitoring / enterprise software
  + Security concerns – control over the machines in the network
  + Control of maintenance and updates of AMIs over time
  + Active Directory Integration out of the box
  + Installing your app ahead of time ( for faster deploys when auto-scaling)
  + Using someone else’s AMI that is optimised for running an app, DB, etc..
* AMI are built for a specific AWS region

Using Public AMIs

* You can leverage AMIs from other people
* You can also pay for other people’s AMI by the hour
  + These people have optimised the software
  + The machine is easy to run and configure
  + You basically rent “ expertise “ from the AMI creator
* AMI can be found and published on the Amazon Marketplace
* WARNING:
  + Do not use an AMI you don’t trust!
  + Some AMIs might come with malware or may not be secure for your enterprise

AMI Storage

* Your AMI take space and they live in Amazon S3
* Amazon S3 is a durable, cheap and resilient storage where most of your backups will live ( but you won’t see them in the S3 console)
* By default , your AMIs are private, and locked for your account / region
* You can also make your AMIs public and share them with other AWS accounts or sell them on the AMI Marketplace

AMI Pricing

* AMIs live in Amazon S3, so you get charged for the actual space in takes in Amazon S3
* Overall it is quite inexpensive to store private AMIs.
* Make sure to remove the AMIs you don’t use

Cross Account AMI Copy ( FAQ + Exam Tip )

* You can share an AMI with another AWS account.
* Sharing an AMI does not affect the ownership of the AMI.
* If you copy an AMI that has been shared with your account, you are the wonder of the target AMI in your account
* TO copy an AMI that was shared with you from another account, the owner of the source AMI must grant you read permissions for the storage that backes the AMI, either the associated EBS snapshot (for an Amazon EBS-backed AMI) or an associated S3 bucket (for an instance store backed AMI).
* **Limits**
* You can’t copy an encrypted AMI that was shared with you from another account. Instead if the underlying snapshot and encryption key were shared with you, you can copy the snapshot while encrypting it with a key of your own. You own the copied snapshot, and can register it as a new AMI.
* You can’t copy an AMI with an associated **billingProduct** code that was shared with you from another account. This includes Windows AMIs and AMIs from the AWS Marketplace. To copy a shared AMI with a **billingProduct** code, launch an EC2 instance in your account using the shared AMI and then create an AMI from the instance.
* <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/CopyingAMIs.html>

**Elastic IPs**

* When you stop and then start an EC2 instance, it changes its public IP.
* If you need to have a fixed public IP, you need an Elastic IP
* An Elastic IP is a public IPv4 IP you own as long as you don’t delete it
* YOu can attach it to one instance at a time
* You can remap it across instances
* You don’t pay for the Elastic IP if it’s attached to a server
* You pay for the Elastic IP if it’s not attached to a server
* *With an Elastic IP address, you can mask the failure of an instance or software by rapidly remapping the address to another instance in you account*
* You can only have 5 Elastic IP in your account ( you can ask AWS to increase that ).
* **Overall, try to avoid using Elastic IP**
  + Always think if other alternatives are available to you
  + You could use a random public IP and register a DNS name to it
  + Or use a Load Balancer with a static host name.

**EC2 CloudWatch Metrics**

* **AWS Provided metrics (AWS pushes them):**
  + Basic Monitoring (default): metrics are collected at a **5** minute interval
  + Detailed Monitoring (paid): metrics are collected at a **1** minute interval
  + Default Metrics Includes:
    - ***CPU***
    - ***Network***
    - ***Disk and Status***
* **Custom metric (yours to push):**
  + Basic Resolution: 1 minute resolution
  + High Resolution: all the way down to 1 second resolution
  + Include RAM, and application level metrics
  + Make sure the IAM permissions on the EC2 instance role are correct!

**EC2 Included Metrics**

* CPU: CPU Utilization + Credit Usage / Balance
* Network: Network In/Out
* Status Check:
  + Instance status = check the EC2 VM
  + System status = check the underlying hardware
* Disk:
  + Read / Write for Operation in Bytes ( only for instance store )
* **RAM is NOT included in the AWS EC2 metrics as a DEFAULT**

**EC2 Custom Metrics**

* Sample custom metrics for EC2:
  + RAM usage
  + Swap usage
  + Any custom metric for your application ( requests per second, etc…)
* Two ways to do this:
  + Use CloudWatch Scripts
  + Use the CloudWatch agent
* Make sure to give the EC2 IAM permissions to write to CloudWatch

**EC2 CloudWatch Logs**

* By default, no logs from your EC2 machine will go to CloudWatch
* You need to run a CloudWatch agent on EC2 to push the logs files you want
* Make sure IAM permissions are correct
* The CloudWatch log agent can also be set up on premise too
* Edit the ***/etc/awslogs/awslogs.conf*** file to configure the logs to track. For more information about editing this file, see the [CloudWatch Logs Agent Reference](https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/AgentReference.html)
* By default the ***/etc/awslogs/awscli.conf*** points to the US-EAST-1 region. To push logs to a different region you have to edit the ***awscli.conf*** file and specify that region

**EC2 Management at Scale (Systems Manager & OpsWorks)**

**SSM & OpsWorks Section**

* How do we manage fleet of EC2 instances and on-premise servers?
* How to apply patches at scale?
* How to run automations?
* Store Parameters?
* Use Chef and Puppet?

**AWS Systems Manager Overview**

* Helps you manage your **EC2**  and **On-Premise** systems at scale
* Get operational insights about the state of your infrastructure
* Easily detect problems
* **Patching automation for enhanced compliance**
* Works for both Windows and Linux OS
* Integrated with CloudWatch metrics / dashboards
* Integrated with
* AWS Config
* Free Service

**AWS Systems Manager Features**

* Resource Groups
* Insights:
  + Insights Dashboard
  + Inventory: discover and audit the software installed
  + Compliance
* **Parameter Store**
* Action
  + Automation ( shut down EC2, create AMIs)
  + **Run Command**
  + Session Manager
  + **Patch Manager**
  + Maintenance Windows
  + State Manager: define and maintaining configuration of OS and applications

**How Systems Manager Works**

* We need to install the SSM agent onto the systems we control
* Installed by default on Amazon Linux AMI & some Ubuntu AMI
* If an instance can’t be controlled with SSM, it’s probably an issue with the SSM agent!
* Make sure the EC2 instances have a proper IAM role to allow SSM actions

**AWS Tags**

* You can add TextKey value Pairs called Tags to many AWS resources
* Commonly used in EC2
* Free naming, common tags are: Name, Environment, Team…
* They’re used for
  + Resource grouping
  + Automation
  + Cost Allocation
* Better to have too many tags then too few!

**AWS Systems Manager Resource Groups**

* Create, view or manage logical group of resources thanks to **tags**
* Allows creation of logical groups of resources such as
  + Applications
  + Different layers of an application stack
  + Production versus development environments
* Regional Service
* Works with EC2, S3, DynamoDB, Lambda, etc…

**SSM Documents**

* Documents can be in JSON or YAML
* You define parameters
* You define actions
* Many documents already exist in AWS
* Can be applied to
  + State Manager
  + Patch Manager
  + Automation
  + Run Command
  + Parameter Store

**AWS Systems Manager Run Command**

* Execute a document ( = script ) or just run a command
* Run command across multiple instance ( using resource groups --utilizing tags )
* Rate Control / Error Control
* Integrated with IAM & CloudTrail
* No need for SSH
* Results in the console

**AWS Systems Manager to PATCH**

* Inventory => List software on an instance
* Inventory + Run Command => Patch Software
* Patch manager + Maintenance Window => Patch OS
* Patch manager => Gives you compliance
* State manager => Ensure instances are in a consistent state ( compliance )

**AWS Systems Manager Session Manager**

* Session Manager allows you to start a secure shell on your VM
* **Does not use SSH access and bastion hosts**
* Only EC2 for now, but On Premise coming soon
* Log actions done through secure shells to S3 and CloudWatch Logs
* IAM permissions: access SSM + write to S3 + write to CloudWatch
* CloudTrail can intercept StartSession events
* AWS SEcure Shell versus SSH
  + No need to open the port 22 at all anymore (security)
  + No need for bastion hosts
  + All commands are logged to S3 / CloudWatch (auditing)
  + Access to Secure Shell is done through User IAM, not SSH keys

**What if I lost my SSH key for EC2?**

* (Traditional method) if the instance is EBS baked:
  + Stop the instance, detach the root volume
  + Attach the root volume to another instance as a data volume
  + Modify the ~/.ssh/authorized\_keys file with your new key
  + Move the volume back to the stopped instance
  + Start the instance and you can SSH into it again
* (New method) if the instance is EBS:
  + Run the AWSSupport-ResetAccess automation document in SSM
* Instance Store backed EC2:
  + YOu can’t stop the instance ( otherwise data is lost) – AWS recommends termination
  + Stephane Maarek tip: Use Session Manager access and edit the ~/.ssh/authorized\_keys file directly

**AWS Parameter Store**

* Secure storage for configuration and secrets
* Optional Seamless Encryption using KMS
* Serverless, scalable, durable, easy SDK free
* Version tracking of configurations / secrets
* Configuration management using path & IAM
* Notifications with CloudWatch Events
* Integration with CloudFormation
* Two versions
  + Plaintext configuration
    - Will check IAM first before sending back the information
  + Encrypted configuration
    - Will check IAM first before decryption using AWS KMS
* AWS CLI
  + Ssm get-parameters
  + Ss, get-parameters-by-path
    - --recursive will pull all the different things in the path

**AWS Opswork**

* Chef & Puppet help you perform server configuration automatically, or repetitive actions
* They work great with EC2 & On Premise VM
* AWS Opsworks = Managed Chef & Puppet
* It’s an alternative to AWS SSM
* **In the exam: Chef & Puppet needed => AWS Opsworks**

**Quick word on Chef / Puppet**

* They help with managing configuration as code
* Helps in having consistent deployments
* Works with Linux / Windows
* Can automate: user accounts, cron, ntp, packages, services..
* They leverage “Recipes” or “Manifests”
* Chef / Puppet have similarities with SSM / Beanstalk / Cloudformation but they’re open-source tools that work cross-cloud

**EC2 High Availability and Scalability - Load Balancer and Auto Scaling Groups**

* Load Balancers:
  + Troubleshooting
  + Advanced options and logging
  + CloudWatch integrations
* Auto Scaling
  + Troubleshooting
  + Advanced options and logging
  + CloudWatch integrations

Scalability & High Availability

* Scalability means that an application / system can handle greater loads by adapting
* There are two kinds of scalability:
  + Vertical Scalability
  + Horizontal Scalability ( = elasticity )
* **Scalability is linked but different to High Availability**

Vertical Scalability

* Vertically Scalability means increasing the size of the instance
* For example, your applications runs on a t2.micro
* Scaling that application vertically means running it on a t2.large
* Vertical Scalability is very common for non distributed systems, such as a database
* RDS, ElastiCache are services that can scale vertically
* There’s usually a limit to how much you can vertically scale ( hardware limit )

Horizontal Scalability

* Horizontal Scalability means increasing the number of instances / systems for your application
* Horizontal scaling implies distributed systems.
* This is very common for web applications / modern applications
* It’s easy to horizontally scale thanks the cloud offerings such as Amazon EC2

High Availability

* High Availability usually goes hand in hand with horizontal scaling
* High availability means running your application / system in at least 2 data center ( == Availability Zones )
* The goal of high availability is to survive a data center loss
* The high availability can be passive ( for RDS Multi AZ for example)
* The high availability can be active ( for horizontal scaling )

High AVailability & Scalability For EC2

* Vertical Scaling: Increase instance size ( = scale up / down )
  + From: t2.nano - 0.5G of RAM, 1 vCPU
  + To: u-12tb.metal – 12.3 TB of RAM, 448 vCPUs
* Horizontal Scaling: increasing number of instances ( = scale out / in )
  + Auto Scaling Group
  + Load Balancer
* High Availability: Run instances for the same application across multi AZ
  + Auto Scaling Group multi AZ
  + Load Balancer multi AZ

**What is load balancing?**

* Load balancers are servers that forward internet traffic to multiple servers ( EC2 Instances ) downstream
* Expose a single point of access (DNS) to your application
* Seamlessly handle failures of downstream instances
* Do regular health checks to your instances
* Provide SSL termination (HTTPS) for your websites
* Enforce stickiness with cookies
* High availability across zones
* Separate public traffic from private traffic

**Why use an EC2 Load Balancers?**

* An ELB ( EC2 Load Balancer ) is a managed load balancer
  + AWS guarantees that it will be working
  + AWS takes care of upgrades, maintenance, high availability
  + AWS provides only a few configuration knobs
* It costs less to setup your own load balancer but it will be a lot more effort on your end
* It is integrated with many AWS offerings / services

**Load balancer on AWS**

* AWS has 3 kinds of Load Balancers
* Classic Load Balancer (v1 - old generation) -2009
* Application Load Balancer (v2 - new generation) -2016
* Network Load Balancer (v2 - new generation) -2017
* Overall, it is recommended to use the newer / v2 generation load balancers as they provide more features
* You can setup internal ( private) or external ( public ) ELBs

**Health Checks**

* Health Checks are crucial for Load Balancers
* They enable the load balancer to know if instances it forward traffic to are available to reply to requests
* The health check is done on a port and a route (/health is common)
* If the response is not 200 (OK), then the instance is unhealthy

**Application Load Balancer (v2)**

* Application load balancers ( Layer 7 ) allow to do:
  + Load balancing to multiple HTTP applications across machines (target groups)
  + Load balancing to multiple applications on the same machine (ex: containers)
  + Load balancing based on route in URL
  + Load balancing based on hostname in URL
* Basically they’re awesome for micro services & container-based application (example: Docker & Amazon ECS)
* Has a port mapping feature to redirect to a dynamic port
* In comparison, we would need to create one Classic Load Balancer per application before. That was very expensive and inefficient

**Application Load Balancer v2 Good to Know**

* Stickiness can be enabled at the target group level
  + Same request goes to the same instance
  + Stickiness is directly generated by the ALB ( not the application)
  + ALB supports HTTP/HTTPS & Websockets protocols
  + The application servers don't see the IP of the client directly
    - The true IP of the client is inserted in the header X-Forwarded-For
    - We can also get Port (X-Forwarded-Port) and proto (X-Forwarded-Proto)

**Network Load Balancer (v2)**

* Network load balancers (Layer 4) allow to do:
  + Forward TCP traffic to your instances
  + Handle millions of requests per seconds
  + Support for static IP or elastic IP
  + Less latency ~ 100ms (vs 400 ms for ALB )
* Network Load Balancers are mostly used for extreme performance and should not be the default load balancer you choose
* Overall, the creation process is the same as Application Load Balancers

**Load balancer Good to Know**

* Classic Load Balancers are Deprecated
  + Application Load Balancers for HTTP / HTTPs & Websocket
  + Network Load Balancer for TCP
* CLB and ALB support SSL certificates and provide SSL termination
* All Load Balancers have health check capability
* ALB can route based on hostname / path
* ALB is a great fit with Elastic Container Service (Docker)
* Any Load Balancer (CLB, ALB, NLB) has a static host name. Do not resolve and use underlying IP
* LBs can scale but not instantaneously – contact AWS for a “warm-up”
* NLB directly see the client IP
* 4xx errors are client induced errors
* 5xx errors are applications induced errors
  + Load Balancer Errors 503 means at capacity or no registered target
* If the LB can’t connect to your application, check your security groups!

**Load Balancer Stickiness**

* It is possible to implement stickiness so that the same client is always redirected to the same instance behind a load balancer
* This works for Classic Load Balancers & Application Load Balancers
* The “cookie” used for stickiness has an expiration date you control
* Use case: make sure the user doesn’t lose his session data
* Enabling stickiness may bring imbalance to the load over the backed EC2 instances
* A value between 1 sec and 7 days

**Load Balancer for SysOps**

* Application Load Balancer:
  + Layer 7 (HTTPS, HTTPS, Websocket)
  + URL based routing (hostname or path)
  + Does not support static IP, but has a fixed DNS
  + Provide SSL termination
* Network Load Balancer:
  + Layer 4 ( TCP )
  + No pre-warming needed
  + 1 static IP per subnet
  + No SSL termination (SSL must be enabled by the application itself)
* Exam tip: Question - How to give an Application Load Balancer a fixed IP address:
  + Chain and Network Load Balancer and a Application Load Balancer to “give: the ALB a fixed IP address

**Load Balancer Pre-warming**

* ELB scale gradually to traffic
* ELB may fail in case of sudden spike of traffic (10x traffic)
* If you expect high traffic ( Christmas Season for example), open a support ticket with AWS to pre-warm your Elastic Load Balancer
  + Duration of traffic
  + Expected requests per second
  + Size of request (in KB)

**Load Balancer Error Codes**

* Successful request : Code 200.
* Unsuccessful at client side: 4xx code.
  + Error 400: Bad Request
  + Error 401: Unauthorized
  + Error 403: Forbidden
  + Error 460: Client closed connection
  + Error 463: X-Forwarded For header with > 30 IP ( similar to malformed requests )
* Unsuccessful at server side: 5xx code.
  + An error 500 / Internal server error would mean some error on the ELB itself
  + Error 502: Bad Gateway
  + Error 503 / Service Unavailable
  + Error 504 / Gateway timeout: probably an issue within the server.
  + Error 561: Unauthorized

**Supporting SSL for Old Browsers**

* Common question is: how do we support Legacy Browsers that has an old TLS (such as TLS 1.0 )
* Answer: change the policy to allow for weaker cipher (eg. DES-CBC-SHA for TLS 1.0)
* Note: only a very small % of the internet uses TLS 1.0
* Elastic Load Balancing provides the following security policies for Application Load Balancers
  + ELBSecurityPolicy-2016-08
  + ELBSecurityPolicy-FS-2018-06
  + ELBSecurityPolicy-TLS-1-2-2017-01
  + ELBSecurityPolicy-TLS-1-2-Ext-2018-06
  + ELBSecurityPolicy-TLS-1-1-2017-01
  + ELBSecurityPolicy-2015-05
  + ELBSecurityPolicy-TLS-1-0-2015-04 <= allows for the weaker cipher
* For more information:
  + <https://docs.aws.amazon.com/elasticloadbalancing/latest/application/create-https-listener.html#describe-ssl-policies>

**Load Balancer Common Troubleshooting**

* Check Security Groups
* Check Health Checks
* Stick Sessions may bring ‘imbalance on the load balancing side
* For Multi-AZ, make sure cross zone balancing is enabled
* Use internal load balancer for private applications that don’t need a public access
* Enable Deletion Protection to prevent against accidental deletes

**Load Balancers Monitoring**

* All Load Balancer metrics are directly pushed to CloudWatch metrics
* BackedConnectionErrors
* HealthyHostCount / UnHealthyHostCount
* HTTPCode\_Backend\_2xx: Successful request.
* HTTPCode\_Backend\_3xx, redirected request
* **HTTPCode\_ELB\_4xx: Client error codes**
* **HTTPCode\_ELB\_5xx: Server error codes generated by the load balancer.**
* Latency
* RequestCount
* **SurgeQueueLength:** The total number of requests (HTTP listener) or connections (TCP listener) that are pending routing to a healthy instance. Help to scale out ASG. Max value is 1024.
* **SpilloverCount:** The total number of requests that were rejected because the surge queue is full.

**Load Balancers Access Logs**

* Access logs from Load Balancers can be stored in S3 and contain:
  + Time
  + Client IP address
  + Latencies
  + Request paths
  + Server response
  + Trace Id
* Only pay for the S3 storage
* Helpful for compliance reason
* Helpful for keeping access data even after ELB or EC2 instances are terminated
* Access Logs are already encrypted

**Application Load Balancer Request Tracing**

* Request tracing – Each HTTP request has an added custom header **“X-Amzn-Trace-Id”**
* Example
  + X-Amzn-Trace-Id:Root=1-67891233-abcdef012345678912345678
* This is useful in logs / distributed tracing platform to track a single request
* Application Load Balancer is not (yet) integrated with X-Ray

**Load Balancer Troubleshooting using metrics**

* **HTTP 400: BAD\_REQUEST =>** The client sent a malformed request that does not meet HTTP specifications
  + Description
    - Indicates that the client sent a bad request
  + Cause 1
    - The client sent a malformed request that does not meet HTTP specifications. For example a request can’t have spaces in the URL
  + Cause 2
    - The client used the HTTP CONNECT method, which is not supported by Elastic Load Balancing
  + Solution:
    - Connect directly to your instance and capture the details of the client request> review the headers and the URL for malformed requests. Verify that the request meets HTTP specifications. Verify that HTTP CONNECT is not used
* **HTTP 503: Service Unavailable =>** Ensure that you have healthy instances in every Availability zone that your load balancer is configured to respon in. Look for HealthyHostCount in CloudWatch
  + Description:
    - Indicates that either the load balancer or the registered instances are causing the error
  + Cause 1
    - Insufficient capacity in the load balancer to hand the request
  + Solution 1
    - This should be a transient issue and should not last more than a few minutes. If it persists contact the AWS support center
  + Cause 2
    - There are no registered instances
  + Solution 2
    - Register at least one instance in every AZ that your load balancer is configured to respon in. Verify this by looking at the *HealthyHostCount* metrics in CloudWatch. If you can’t ensure that an instance is registered in each AZ, the recommendation is enabling cross zone load balancing.
  + Cause 3
    - There are no healthy instances
  + Solution 3
    - Ensure that you have healthy instances in every AZ that your load balancer is configured to respond in. Verify this by looking at the *HealthyHostCount* metric
  + Cause 4
    - The surge queue is full
  + Solution 4
    - Ensure that your instances have sufficient capacity to handle the request rate. Verify this by looking at the *SpilloverCount* metric
* **HTTP504:Gateway Timeout =>**  Check if keep-alive settings on you EC2 instance are enabled and make sure that the keep alive timeout is greater than the idle timeout settings of load balancer
  + Description:
    - Indicates that the load balancer closed a connection because a request did not complete within the idle timeout period
  + Cause 1
    - The application takes longer to respond than the configured idle amount
  + Solution 1
    - Monitor the HTTPCode\_ELB\_5xx and Latency metrics. If there is an increase in these metrics, it could be due to the application not responding within the idle timeout period. For details about the request that are timing out, enable access logs on the load balancer and review the 504 response codes in the logs that are generated by Elastic Load Balancing. If necessary, you can increase your capacity or increase the configured idle timeout so that lengthy operations (such as uploading a large file) can complete.
  + Cause 2
    - Registered instances closing the connection to Elastic Load Balancing
  + Solution 2
    - Enable keep-alive setting on EC2 instances and make sure that the keep-alive timeout is greater than the idle timeout setting of your load balancer
* Set alarms & look at the documentation for troubleshooting:
  + <https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/ts-elb-error-message.html>
* OTHERS
* **HTTP 405: METHOD\_NOT\_ALLOWED**
  + Description
    - Indicates that the method length is not valid
  + Cause
    - The length of the method in the request header exceeds 127 characters
  + Solution
    - Check the length of the method
* **HTTP 408: Request Timeout**
  + Description:
    - Indicates that the client cancelled the request or failed to send a full request
  + Cause 1
    - A network interruption or a bad request construction, such as partially formed headers; specified content size doesn’t match the actual content size transmitted; and s on
  + Solution 1:
    - Inspect the code that is making the request and try sending it directly to your registered instances ( or a development / test environment ) where you have more control over inspection of the actual request.
  + Cause 2:
    - Connection to the client is closed ( load balancer could not send a response)
  + Solution 2:
    - Verify that the client is not closing the connection before a response is sent by using a packet sniffer on the machine making the request
* **HTTP 502: Bad Gateway**
  + Description
    - Indicates that the load balancer was unable to parse the response sent from a registered instance
  + Cause
    - Malformed response from the instance or potentially an issue with the load balancer
  + Solution
    - Verify that the response being sent from the instance conforms to HTTP specifications.

**What’s an Auto Scaling Group?**

* In real-life, the load on your websites and application can change
* In the cloud, you can create and get rid of servers very quickly
* The goal of an Auto Scaling Group ( ASG ) is to:
  + **Scale out** (add EC2 instances) to match an increased load
  + **Scale In** (remove EC2 instances) to match a decreased load
  + Ensure we have a minimum and a maximum number of machines running
  + Automatically Register new instances to a load balancer

**ASG have the following attributes**

* A launch configuration
  + AMI + Instance Type
  + EC2 User Data
  + EBS Volumes
  + Security Groups
  + SSH Key Pair
* Min Size / Max Size / Initial Capacity
* Network + Subnets Information
* Load Balancer Information
* Scaling Policies

**Auto Scaling Alarms**

* It is possible to scale an ASG based on CloudWatch alarms
* An Alarm monitors a metric (such as Average CPU)
* Metrics are computed for the overall ASG instances
* Based on the alarm:
  + We can create scale-out policies (increase the number of instances)
  + We can create scale-in policies (decrease the number of instances)

**Auto Scaling New Rules**

* It is now possible to define “better” autoscaling rules that are directly managed by EC2
  + Target Average CPU Usage
  + Number of requests on the ELB per instance
  + Average Network IN
  + Average Network OUT
* These rules are easier to set up and can make more sense

**Auto Scaling Custom Metrics**

* We can auto scale based on a custom metric ( ex: number of connected users)
* 1. Send custom metric from application on EC2 to CloudWatch ( PutMetricAPI)
* 2. Create CloudWatch alarm to react to low / high values
* 3. USe the CloudWatch alarm as the scaling policy for the ASG

**Auto Scaling Brain Dump**

* Scaling policies can be on CPU, Network… and can even be on custom metrics or based on a schedule (if you know your visitors patterns)
* ASGs use Launch configurations and you update an ASG by providing a new launch configuration
* IAM roles attached to an ASg will assigned to EC2 instances
* ASG is free. You pay for the underlying resources being launched
* Having instances under an ASG means that if they get terminated for whatever reason the ASG will restart them.
* ASG can terminate instances marked as unhealthy by an LB ( and hence replace them )

**Scaling Processes in ASG - This is useful for troubleshooting**

* Launch: add a new EC2 to the group, increasing the capacity
* Terminate: Removes an EC2 instance from the group, decreasing its capacity
* HealthCheck: Checks the health of the instances
* ReplaceUnhealthy: Terminate unhealthy instances and recreate them
* AZRebalance: Balancer the number of EC2 instances across AZ
* AlarmNotification: Accept notification from CloudWatch
* Scheduled Actions: Performs scheduled actions that you create
* AddToLoadBalancer: Adds instances to the load balancer or target group
* We can suspend these processes

**Note on AZRebalance**

* AZRebalance = launch new instance then terminate old instance
* If you suspend the Launch process
  + AZRebalance won’t launch instances
  + AZRebalance won’t terminate instances
* If you suspend the Terminate process
  + The ASG can grow up to 10% of this size (it’s allowed during rebalances)
  + The ASG could remain at the increased capacity as it can’t terminate instances

**ASG for SysOps**

* To makee sure you have high availability, means you have least 2 instances running across 2 AZ in your ASG ( must configure multi AZ ASG)
* Health checks available:
  + EC2 status checks
  + ELB Health Checks
* ASG will launch a new instance after terminating an unhealthy one
* ASG will not reboot unhealthy hosts for you
* Good to know CLI:
  + set -instance-health
  + terminate -instance-in-auto-scaling-group

**Troubleshooting ASG issues**

* <number of instances> instance(s) are already running. Launching EC2 instance failed.
  + The Auto Scaling group has reached the limit set by the Desired Capacity parameter. Update your Auto Scaling group by providing a new value for the desired capacity.
* Launching EC2 instances is failing:
  + The security group does not exist. SG might have been deleted
  + The key pair does not exist. The key pair might have been deleted
* If the ASG fails to launch an instance for over 24 hours, it will automatically suspend the processes (administration suspension)

**CloudWatch Metrics for ASG**

* The following metrics are available for ASG (opt-in):
  + GroupMinSize
  + GroupMaxSize
  + GroupDesiredCapacity
  + GroupServicInstance
  + GroupPendingInstance
  + GroupStandbyInstance
  + GroupTerminatingInstances
  + GroupTotalInstances
* You should enable metric collection to see these metrics
* Metrics are collected every 1 minute
* You can also monitor the underlying EC2
* Basic monitoring : 5 minutes granularity
* Detailed monitoring: 1 minute granularity

**Elastic Beanstalk Deployment Automation**

Developer problems on AWS

* Managing infrastructure
* Deploying Code
* Configuring all the databases, load balancers, etc
* Scaling concerns
* Most webapps have the same architecture (ALB + ASG)
* All the developers want is for their code to run
* Possibly, consistently across different applications and environments

**AWS Elastic Beanstalk Overview**

* Elastic Beanstalk is a developer centric view of deploying an application on AWS
* It uses all the components we’ve seen before: EC2, ASG, ELB, RDS, etc…
* But it’s all in one view that’s easy to make sense of!
* We still have full control over the configuration
* Beanstalk is free but you pay for the underlying instances and services

**Elastic Beanstalk**

* Managed Service
* Instance configuration/ OS handled by Beanstalk
* Deployment strategy is configurable but performed by Elastic Beanstalk
* Just the application code is the responsibility of the developer
* Three architecture models:
  + Single instance deployment: good for dev
  + LB + ASG: great for production or pre-production web applications
* Elastic Beanstalk has three components
  + Application
  + Application version: each deployment get assigned a version
  + Environment name (dev, test, prod…): free naming
* You deploy application versions to environments and can promote application versions
* Full control over life cycle of environments
  + Create Application => Create Environments => Upload Version ( + alias ) => Release to environments
* Supported Languages and Platforms
  + GO
  + Java SE
  + Java w/ Tomcat
  + .NET on windows server with IIS
  + Node.js
  + PHP
  + Python
  + Ruby
  + Packer Builder
  + Single Container Docker
  + Multicontainer Docker
  + Preconfigured Docker
* If not supported, you can write your custom platform (advanced)

**Elastic Beanstalk Deployment Modes**

* Single Instance -- Great for Development
  + DNS name => Elastic IP
* High Availability with Load Balancer -- Great for prod
  + DNS name => ELB DNS name
* Elastic Load Balancing
  + EC2 instances located in multiple AZ zones

**Beanstalk Deployment Options for Updates**

* **All at once (deploy all in one go )**
  + Fastest, but instances aren’t available to serve traffic for a bit (downtime)
  + Fastest Deployment
  + Application has downtime
  + Great for quick iterations in development environment
  + No additional cost
* **Rolling:**
  + Update a few instances at a time ( bucket ), and then move onto the next bucket once the first bucket is healthy
  + Application is running below capacity
  + Can set the bucket size
  + Application is running bot versions simultaneously
  + No additional cost
  + Long Deployment
* **Rolling with additional batches:** 
  + Like rolling but spins up new instances to move the batch ( so that the old application is still available)
  + Application is running at capacity
  + Can set the bucket size
  + Application is running both versions simultaneously
  + Small additional cost
  + Additional batch is removed at the end of the deployment
  + Longer deployment
  + Good for prod
* **Immutable:** 
  + Spins up new instances in a new ASG, deploys version to these instances, and then swaps all the instances when everything is healthy
  + Zero downtime
  + New Code is deployed to new instances on a temporary ASG
  + High cost, double capacity
  + Longest deployment
  + Quick rollback in case of failures (just terminate new ASG)
  + Great for prod

**NOT A FEATURE OF ELASTIC BEANSTALK**

**Blue / Green Deployment**

* Not a “direct feature” of Elastic Beanstalk
* Zero downtime and release facility
* Create a new “stage” environment and deploy v2 there
* The new environment (green) can be validated independently and roll back if issues occur
* Route 53 can be setup using weighted policies to redirect a little bit of traffic to the stage environment
* Using Beanstalk, “swap URLs” when done with the environment test

**Elastic Beanstalk Deployment Summary from AWS DOC**

* <https://docs.aws.amazon.com/elasticbeanstalk/lates/dg/using-features.deploy-existing-version.html>

**Beanstalk for SysOps**

* Beanstalk can put applications logs into CloudWatch Logs
* You manage the application, AWS will manage the underlying infrastructure
* Know the difference deployment modes for your application
* Custom domain: Route 53 Alias or CNAME on top of Beanstalk URL
* You are not responsible for patching the runtimes ( Node.js, PHP, etc )
* Questions are very basic compared to the Developer Exam

**How Beanstalk deploys applications**

Ex: Rolling

* EC2 has a base AMI (can configure)
* EC2 gets the new code of the app
* EC2 resolves the app dependencies ( can take a while )
* Apps get swapped on the EC2 instance

Resolving dependencies can take a long time!

We can use Golden AMI to fix the problem

**Golden AMI**

* If your application has a lot of application or OS dependencies, and you want to deploy as quickly as possible, you should create a **Golden AMI:**
* Golden AMI = standardized company-specific AMI with:
  + Package OS dependencies
  + Package App dependencies
  + Package company-wide software
* By using a Golden AMI to deploy to Beanstalk ( in combination of blue/green new ASG deployment), our application won’t need to resolve dependencies or a long time to configure!

**Troubleshooting Beanstalk**

* If the health of your environment changes to red, try the following:
  + Review environment events
  + Pull logs to view recent log files entries
  + Roll back to a previous, working version of the application
* When accessing external resources, make sure the security groups are correctly configured
* In case of command timeouts, you can increase the deployment timeout

**AWS CloudFormation - Managing your infrastructure as code**

**Infrastructure as Code**

* Currently all of the things we have done up to this point is a lot of manual work
* All this manual work will be very tough to reproduces:
  + In another region
  + In another AWS account
  + Within the same region if everything was deleted
* Wouldn’t it be great, if all our infrastructure was … code?
* That code would be deployed and created / updated / deleted our infrastructure

**What is cloudformation**

* Cloudformation is a declarative way of outlining your AWS Infrastructure, for any resources ( most of them are supported ).
* For example, within a CloudFormation template, you say:
  + I want a security group
  + I want two EC2 machines using this security group
  + I want two Elastic IPs for these EC2 machines
  + I want an S3 bucket
  + I want a load balancer (ELB) in front of these machines
* Then CloudFormation creates those for you, in the **right order**, with the **exact configuration** that you specify

**Benefits of AWS CloudFormation**

* Infrastructure as code
  + No resources are manually created, which is excellent fro control
  + The code can be version controlled for example using git
  + Changes to the infrastructure are reviewed through code
* Cost
  + Each resource within the stack is staged with an identified so you can easily see how much each stack will cost you
  + YOU can estimate the costs of your resources using the CloudFormation template
  + Saving Strategy: In Dev, you could automation deletion of templates at 5 PM and recreate them at 8 AM, safely
* Productivity
  + Ability to destroy and re-create an infrastructure on the cloud on the fly
  + Automated generation of Diagram for your templates!
  + Declarative programming ( no need to figure out ordering and orchestration )
* Separation of concern: create many stacks for many apps, and many layers
  + VPC stacks
  + Network stacks
  + App stacks
* Don’t reinvent the wheel
  + Leverage existing templates on the web
  + Leverage the documentation which is extensive

**How CloudFormation Works**

* Templates have to be uploaded in S3 and then referenced in CloudFormation
* To update a template, we can’t edi previous ones. We have to re upload a new version of the template to AWS
* Stacks are identified by a name
* Deleting a stack deletes every single artifact that was created by CloudFormation

**Deploying CloudFormation Templates**

* Manual way:
  + Editing templates in the CloudFormation Designer
  + Using the console to input parameters, etc
* Automated way:
  + Editing templates in a YAML file
  + Using the AWS CLI ( Command Line Interface) to deploy the templates
  + Recommended way when you fully want to automate your flow

**CloudFormation Building Blocks**

**Templates components:**

* Resources: your AWS resources declared in the template ( MANDATORY)
* Parameters: The dynamic inputs for your template
* Mappings: The static variables for your template
* Outputs: References to what has been created
* Conditionals: List of conditions to perform resource creation
* Metadata

**Template Helpers:**

* References
* Functions

**YAML Crash Course**

* YAML and JSON are the languages you can use for CloudFormation
* JSON is horrible for cfn
* YAML is great for cfn
* Key-Value pairs
* Nested Objects
* Support Arrays
* Multi-line strings
* Can include comments

**What are parameters?**

* Parameters are a way to provide inputs to your AWS CloudFormation template
* They’re important to know about if:
  + You want to reuse your templates across the company
  + Some inputs can not be determined ahead of time
* Parameters are extremely powerful, controlled and can prevent errors from happening in your templates thanks to types

**When should you use a parameter**

* Ask yourself this:
  + Is this cloudformation resource configuration likely to change in the future
  + If so make it a parameter
* You won’t have to re-upload a template to change its content

**Parameter Settings**

* Parameters can be controlled by all these settings

Type

* String
* Number CommaDelimitedList
* List<Type>
* AWS Parameter ( to help catch invalid values - match against existing values in teh AWS Account

Description  
Constraints  
Constraints Description (string)

Min/Max Length

Min/Max Value

Defaults

AllowedValues (array)

AllowedPattern(regexp)

NoEcho(boolean)

**How to Reference a Parameter**

* The Fn::Ref function can be leveraged to reference parameters
* Parameters can be used anywhere in a template
* The shorthand for this in YAML is !Ref
* The function can also reference other elements within the template

**Concept: Pseudo Parameters**

* AWS offers us pseudo parameters in any CloudFormation template
* These can be used at any time and are enabled by default
  + AWS::AccountId
    - 1234567890
  + AWS::NotificationARNs
    - [arn:aws:sns:us-east-1:12345678901234:MyTopic]
  + AWS::NoValue
    - Does not return a value
  + AWS::Region
    - us-east-2
  + AWS::StackId
    - arn:aws:cloudformation:us-east-1:123456789012:stack/MyStack/1x2fa620-982a-11e3-aff7-59e2416294e0
  + AWS::StackName
    - MyStack

**What are resources?**

* Resources are the core of you CloudFormation template and are MANDATORY/REQUIRED
* They represent the different AWS Components that will be created and configured
* Resources are declared and can reference each other
* AWS figures out creation, updates and deletes of resources for us
* There are over 224 types of resources
* Resource types identifiers are of the form
  + AWS::aws-product-name::data-type-name

**How do I find resources documentation**

* All resources can be found here
  + <http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resources-type-ref.html>

**Can I create a dynamic amount of resources?**

* No you can’t. Everything in the CloudFormation template has to be declared. You can’t perform code generation there

**Is every AWS Service supported?**

* Almost. Only a select few niches are not yet supported
* You can work around that using AWS Lambda Custom Resources

**What are mappings?**

* Mappings are fixed variables within your CloudFormation Template
* They’re very handy to differentiate between different environments (dev vs prod), regions (AWS regions), AMI types, etc
* All the values are hardcoded within the template
* Example:
  + Mappings:
    - Mapping01:
      * Key01:
        + Name: Value 1
      * Key02:
        + Name: Value 2
      * Key03:
        + Name: Value 3

**When would you use mappings vs parameters**

* Mappings are great when you know in advance all the values that can be taken and that they can be deduced from variables such as
  + Region
  + Availability Zone
  + AWS Account
  + Environment (dev vs prod)
  + Etc…
* They allow safer control over the template
* Use parameters when the values are really user specific

**Fn::FindInMap – Accessing Mapping Values**

* We use **Fn::FindInMap** to return a named value from a specific key
* **!FindInMap [ MapName, TopLevelKey, SecondLevelKey ]**

**What are outputs?**

* The Outputs section declares optional outputs values that we can import into other stacks ( if you export them first)!
* You can also view the outputs in the AWS console or in using the AWS CLI
* They’re very useful for example if you define a network CloudFormation, and output the variables such as VPC ID and your Subnet IDs
* It’s the best way to perform some collaboration cross stack, as you let expert handle their own part of the stack
* You can’t delete a CloudFormation Stack if its outputs are being referenced by another CloudFormation Stack

**Outputs Example**

* Creating a SSH Security Group as part of one template
* We create an output that reference that security group
* Example
  + Outputs:
    - StackSSHSecurityGroup:
      * Description: The SSH Security Group for Company
      * Value: !Ref MyCompanyWideSSHSecurityGroup
      * Export:
        + Name: SSHSecurityGroup

**Cross Stack Reference**

* We then create a second template that leverages that security group
* For this, we use the **Fn::ImportantValue** function
  + YAML
    - !ImportValue
* You can't delete the underlying stack until all the references are deleted too.

**What are conditions used for?**

* Conditions are used to control the creation of resources or outputs based on a condition
* Conditions can be whatever you want them to be, but common ones are:
  + Environment (dev / test / prod)
  + AWS Region
  + Any parameter value
* Each condition can reference another condition, parameter value or mapping

**How to define a condition?**

* The logical ID is for you to choose. It’s how you name condition
* The intrinsic function (logical) can be any of the following:
  + Fn::And
  + Fn::Equals
  + Fn::If
  + Fn::Not
  + Fn::Or

**Using a condition**

* Conditions can be applied to resources / output / etc…

**CloudFormation Must Know Intrinsic Functions**

* Ref
  + The Fn::Ref function can be leveraged to reference:
    - Parameters => returns the value of the parameter
    - Resources => returns the physical ID of the underlying resource (ex: EC2 ID)
  + The shorthand for this in YAML is !Ref
* Fn::GetAtt
  + Attributes are attached to any resources you create
  + To know the attributes of your resources, the best place to look is at the documentation
  + For example: the AZ of an EC2 machine ( EC2Instance.AvailabilityZone)
* Fn::FindInMap
  + We can **Fn::FindInMap** to return a named value from a specific key
  + **!FindInMap [ MapName, TopLevelKey, SecondLevelKey ]**
* Fn::ImportValue
  + Import values that are exported in other templates
  + For this, use the **Fn::ImportValue** function
* Fn::Join
  + Join values with a delimiter
    - !Join [ delimiter, [ *comma-delimited list of values* ] ]
  + This creates “a:b:c”
    - !Join [ “:” , [ a, b, c] ]
* Fn::Sub
  + **Fn::Sub,** or **!Sub** as a shorthand, is used to substitute variables from a text. It’s a very handy function that will allow you to fully customize your templates
  + For example you can combine **Fn::Sub** with Reference or AWS Pseudo variables!
  + **String** must contain **${VariableName}** and will substitute them
* Condition Functions (Listed above)

**User Data in EC2 for CloudFormation**

* We can have user data at EC2 instance launch through the console
* We can also include it in CloudFormation
* **The important thing to pass is the entire script through the function Fn::Base64**
* Good to know: user data script log is in **/var/log/cloud-init-output.log**
* Example : The vertical pipe indicates a long strings with new lines, and is needed
  + UserData:
    - Fn::Base54: |
      * Yum update -y
      * Yum install -y httpd
      * Systemctl start httpd
      * Systemctl enable http
      * Echo “Hello World from user data” > /var/www/html/index.html

**Cfn-init**

* AWS::CloudFormation::Init must be in the metadata of a resource
* With the cfn-init script, it helps make complex EC2 configurations readable
* The EC2 instance will query the CloudFormation service to get init data
* Logs got to /var/log/cfn-init.log (how to troubleshoot the startup of the instances user data)
* Example -- UserData
  + UserData:
    - Fn::Base54: |
      * Yum update -y aws-cfn-bootstrap
      * /opt/aws/bin/cfn-init 0s ${AWS::StackId} -r MyInstance --region ${AWS::Region} || error\_exit ‘Failed to run cfn-init’
* Example -- Metadata
  + Metadata:
    - Comment: Install a simple apache HTTP page
    - AWS::CloudFormation::Init:
      * Config:
        + Packages:

Yum:

Httpd: []

* + - * + Files:

“/var/www/html/index.html

Content: |

<h1> hello world from ec2 instance</h1><p>this was created using cfn-init</p>

Mode: ‘000644’

Commands:

Hello:

Command: “echo ‘hello world’”

Services:

Sysvinit:

Httpd:

Enabled: ‘true’

ensureRunning: ‘true’

**Cfn-signal & wait conditions**

* We still don’t know how to tell CloudFormation that the EC2 instance got properly configured after a cfn-init
* For this, we can use the **cfn-signal** script!
  + We run cfn-signal right after cfn-init
  + Tell CloudFormation service to keep on going or fail
* We need to define **WaitCondition:**
  + Block the template until it receives a signal from cfn-signal
  + We attach a **CreationPolicy** ( also works on EC2, ASG)
* Run this script after the cfn-init script
  + /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackId} **--resource** **SampleWaitCondition** --region ${AWS::Region}
* Inside of the resources block
  + **SampleWaitCondition:**
    - CreationPolicy:
      * ResourceSignal:
        + Timeout: PT2M
        + Count: 1
      * Type: AWS::CloudFormation:WaitCondition

**Wait Condition Didn’t Receive the Required Number of Signals from an Amazon EC2 Instance**

* Verify that the AMI you’re using has the AWS CloudFormation helper scripts installed.
  + If the AMI doesn’t include the helper scripts, you can also download them to your instance
* Verify that the cfn-init & cfn-signal command was successfully run on the instance
  + You can view logs, such as /var/log/cloud-init.log or /var/log/cfn-init.log, to help you debug the instance launch
  + You can retrieve the logs by logging into to your instance, but you must disable rollback on failure or else AWS CloudFormation deletes the instance after your stack fails to create
* Verify that the instance has a connection to the Internet.
  + If the instance is in a VPC the instance should be able to connect to the internet through a NAT deviceb If it's in a private subnet
  + or through an Internet gateway if it’s in a public subnet.

**CloudFormation Rollbacks**

* Stack Creation Fails:
  + Default: everything rolls back ( gets deleted ). We can look at the log
  + Option to disable rollback and troubleshoot what happened
* Stack Update Fails:
  + The stack automatically rolls back to the previous known working state
  + Ability to see in the log what happened and error messages

**Nested Stacks**

* Nested stacks are stacks as part of other stacks
* They allow you to isolate repeated patterns / common components in separate stacks and call them from other stacks
* Example
  + Load Balancer configuration that is reused
  + Security Group that is reused
* Nested stacks are considered best practice
* To update a nested stack, always update the parent (root stack)

**ChangeSets**

* When you update a stack, you need to know what changes before it happens for greater confidence
* ChangeSets won’t say if the update will be successful

**Retaining Data on Deletes**

* You can put a DeletionPolicy on any resource to control what happens when the CloudFormation Template is deleted
* **DeletionPolicy=Retain:**
  + Specify on resources to preserve / backup in case of CloudFormation deltets
  + To keep a resource, specify **Retain**  (works for any resource / nested stack)
* **DeletionPolicy=Snapshot:**
  + EBS Volume, Elasticache, Cluster, Elasticache, Replication Group
  + RDS DBInstance, RDS DBCluster, Redshift Cluster
* **DeletePolicy=Delete (default behaviour):**
  + Note: for **AWS::RDS::DBCluster** resources, the default policy is snapshot
  + Note: to delete an S3 bucket you need to first empty the bucket of its content
* **Termination Protection on Stacks**
  + To prevent accidental deletes of CloudFormation templates, use Termination Protection

**EBS & EFS Section**

* EBS and EFS in depth
* Performance
* Troubleshooting
* Operations
* Monitoring

**What’s an EBS Volume**

* An EC2 machine loses its root volume (main drive) when it is manually terminated
* Unexpected terminations might happen from time to time ( AWS would email you)
* Sometimes, you need a way to store your instance data somewhere
* An EBS (Elastic Block Store) Volume is a network drive you can attach to your instances while they run
* It allows your instances to persist data

**EBS Volume**

* It’s a network drive (i.e. not a physical drive)
  + It uses the network to communicate the instance, which means there might be a bit of latency
  + It can be detached from an EC2 instance and attached to another one quickly
* It’s locked to an Availability Zone (AZ)
  + An EBS Volume in us-east-1a cannot be attached to us-east-1b
  + To move a volume across, you first need to snapshot it
* Have a provisioned capacity (size in GBs, and IOPS)
  + You get billed for all the provisioned capacity
  + You can increase the capacity of the drive overtime
* EBS Volumes come in 4 types
  + GP2(SSD): General purpose SSD volume that balances price and performance for a wide variety of workloads
  + IO 1 (SSD): Highest-performance SSD volume for mission critical low latency or high throughput workloads
  + ST 1 (HHD): Low cost HDD volumes designed for frequently accessed through put intensive workloads
  + SC1 (HHD) Lowest cost HDD volume designed for less frequently accessed workloads
* EBS Volumes are characterized in Size | Throughput | IOPS (I/O Operations per second)
* When in doubt, always consult the AWS documentation - it’s good!
* **Only GP2 and IO 1 can be used as boot volumes**

**EBS Volume Types Use cases GP2 ( from AWS doc ) - General Purpose v2**

* Recommended for most workloads for **General Purpose** Use
* System boot volumes
* Virtual desktops
* Low-latency interactive apps
* Development and test environments
* 1Gb - 16 Tb
* **Small gp2** volumes can **burst IOPS to 3000**
* **Max** IOPS is **16,000**
* **3 IOPS per GB**, means at **5,334GB we are at the max IOPS**

**EBS Volume Types Use Cases IO 1 (From AWS Docs) - Input/Output**

* **Critical business applications** that **require sustained IOPS performance**, or **more than 16,000 IOPS** per volume (gp2 limit)
* **Large database workloads,** such as;
* MongoDB, Cassandra, Microsoft SQL Server, MySQL, PostgreSQL, Oracle
* 4 Gb - 16 Tb
* IOPS is provisioned (PIOPS) – **Min 100** - **Max 64,000 (Nitro instances)** else Max **32,000 (other instances)**
  + **Nitro MAX 64k**
  + **Other MAX 32k**
* The maximum ratio of provisioned IOPS to requested volume size ( in Gb) is 50:1
  + **50 IOPS : 1 Gb**

**EBS Volume Types Use Cases ST 1 (from AWS doc) - STreaming v1**

* **Streaming workloads** requiring consistent, **fast throughput at a low price**
* Big data, Data warehouses, Log processing
* Apache Kafka
* **Cannot be a boot volume**
* 500Gb -16Tb
* **Max IOPS is 500**
* **Max** throughput of **500Mb/s** can burst

**EBS Volume Types Use Cases SC1 (from AWS doc) Storage Container v1**

* Throughput-oriented **storage container** for **large volumes of data** that is **infrequently accessed**
* **Scenarios** where the **lowest storage cost is important**
* Cannot be a boot volume
* 500Gb - 16Tb
* **Max IOPS is 350**
* **Max throughput of 250Mb/s can burst**

**GP2 volumes I/O burst**

* If your gp2 small instance **volume is less than 1000Gb** (1Tb) (means IOPS less than 3000) it **can burst to 3000 IOPS performance**
  + This means what burst is that if less then the max of a small gp2 it can burst up to the max of a small Gp2 for short periods of time (3000 IOPS)
* This is a similar concept to t2 instances with their CPU
* You accumulate “burst credit over time”, which allows your volume to have good performance when needed
* The bigger the volume the faster you fill up your **“burst credit balance”**
* **What happens if I empty my I/O credit balance?**
  + The maximum I/O you get becomes the baseline you paid for
  + If you see the balance being 0 all the time, increase the gp2 or switch to IO 1
  + Use CloudWatch to monitoring the I/O credit balance
* *Note: burst concept also applies to st1 or sc1 (for increase in throughput)*

**Computing MB/s based on IOPS**

* Gp2:
  + **Throughput in Mb/s = (volume size in Gb) x (IOPS per Gb) x (I/O size in Kb)**
  + ( 1Gb - 334Gb ) x (3 IOPS/ GB) x (256Kb/I/O)
  + 100Gb x ( 3 IOPS/s \* 1Gb ) x 256 Kb/IOPS = 76800 Kb/s
  + 76000 Kb/s / 1024 Kb/Mb = 75 Mb/s
  + Limit to a max of 250Mb (means volume >= 334 Gb won’t increase throughput)
* Io1:
  + **Throughput in Mb/s = (Provisioned IOPS) x (I/O size in Kb)**
  + The throughput limit of io 1 volumes is 256 kb/s for each IOPS provisioned
  + Limit to a max of 500Mb/s (at 32,000 IOPS) and 1000Mb/s (64,000 IOPS)
* DATA CONVERSION
  + bit : 1
  + 1 Byte : 8 bits
  + 1Kb : 1024 bytes
  + 1 Mb : 1024 Kb
  + 1 Gb : 1024 Mb
  + 1 Tb : 1024 Gb
  + 1 Pb : 1024 Tb

**EBS Volume Resizing**

* Feb 2017: You can now resize the EBS volumes
* You can only **Increase the size**  of EBS volumes
  + Size (any volume types)
  + IOPS ( but only in IO 1 EBS volumes)
* After resizing an EBS volume, you need to repartition your drive
* After increasing the size, it’s possible for the volume to be in a long time in the “optimisation: phase. The volume is still usable
  + This means that after a increase the EBS volume will be listed as “optimisation”
  + However the EBS volume is still usable in this phase

**EBS Snapshots**

* Incremental - only backup changed blocks
* EBS backups use IO and you shouldn’t run them while your application is handling a lot of traffic
* Snapshots will be stored in S3 (but you won’t directly see them)
* Not necessary to detach volume to do snapshot, but recommended
* Max 100,000 snapshots
* Can copy snapshots across AZ or Region
* Can make Image (AMI) from Snapshot
* **EBS volumes restored by snapshot need to be pre-warmed ( using fio or add command to read the entire volume )**
* **Snapshots can be automated using Amazon Data Lifecycle Manager**

**EBS Migration**

* EBS Volumes are only locked to a specific AZ
* To migrate it to a different AZ (or region):
  + Snapshot the volume
  + (optional) Copy the volume to a different region
  + Create a volume from the snapshot in the AZ of your choice

**EBS Encryption**

* When you create an encrypted EBS volume, you get the following
  + Data at rest is encrypted inside the volume
  + All the data in flight moving between the instance andd the volume is encrypted
  + All snapshots are encrypted
  + All volumes created from the snapshot
* Encryption and decryption are handled transparently ( you have nothing to do)
* Encryption has minimal impact on latency
* EBS Encryption leverages keys from KMS (AES -256)
* Copying an unencrypted snapshot allows encryption
* Snapshots of encrypted volumes are still encrypted

Encryption: encrypt an encrypted EBS Volume

**EBS vs Instance Store**

* Some instance do not come with Root EBS volumes
* Instead, they come with “**Instance Store**” (**= ephemeral storage**)
* **Instance store** is **physically attached to the machine** ( EBS is a network drive)
* Pros:
  + Better I/O performance
  + Good for buffer / cache / scratch data / temporary content
  + Data survives reboots
* Cons:
  + On stop or termination , the instance store is lost
  + You can’t resize the instance store
  + Backups must be operated by the user

**Local EC2 instance store**

* **Physical disk attached to the physical server where your EC2 is**
* Very High IOPS (because of the physical nature )
* Disks up to 7.5 Tb (can change over time), stripped to reach 30 Tb (can change over time)
* Block Storage (just like EBS )
* Cannot be increased in size
* Risk of data loss if hardware fails

**EBS for SysOps**

* If you plan to use the root volume of an instance after its terminated:
  + Set the Delete on Termination flag to “No”
  + You can see this option when creating the EC2 instance
* **If you use EBS for high performance, use EBS-optimized instance types**
* If an EBS volume is unused, you still pay for it
* **For cost saving** over a **long period of time**, it can be **cheaper to snapshot** a volume and **restore it later if unused**

**EBS Troubleshooting**

* **High wait time or slow response for SSD => increase iOPS**
* EC2 won’t start with EBS volume as root: make sure volume names are properly mapped (/dev/xvdb instead of /dev/xvda for example)
* After increasing a volume size, you still need to reparation to use the incremental storage ( xfs\_growfs for example)

**EBS RAID Options**

* EBS is already redundant storage (replicated within an AZ)
* But what if you want to increase IOPS to say 100,000 iOPS
* What if you want to mirror your EBS volumes
* You would mount volumes in parallel in RAID settings
* RAID is possible as long as your OS supports it
* Some RAID options are:
  + RAID 0
  + RAID 1
  + RAID 5 (Not recommended for EBS - see documentation)
  + RAID 6 (Not recommended for EBS - see documentation)

**RAID 0 (Increase Performance)**

* EC2 Instance that is backed by 2 or more volumes
* Writes are distributed between the volumes
* Combining 2 or more volumes and getting the total disk space adn I/O
* But one disk fails, all the data is failed ( meaning that if one fails all fails)
* Basically you are trading security for performance increase
* Use cases would be
  + An application that needs a lot of IOPS and doesn’t need fault tolerance
  + A database that has replication already built-in
* Using this, we can have a very big disk with a lot of IOPS
* For Example
  + Two 500Gb Amazon EBS io 1 volumes with 4000 provisioned iOPS each will create a
  + 1000 Gb RAID 0 array with an available bandwidth of 8,000 IOPS and 1,000 Mb/s of throughput

**RAID 1 ( Increased Fault Tolerance)**

* RAID 1 = Mirroring a volume to another
* If one disk fails, our logical volume is still working
* We have to send the data to two EBS volume a the same time (2x network)
* Use Case:
  + Application that need increase volume fault tolerance
  + Application where you need to service disks
* For example:
  + Two 500 Gb Amazon EBS io 1 volumes with 4k provisioned IOPS each will create a…
  + 500 gb RAID 1 array with an available bandwidth of 4,000 iOPS and 500mb/s of throughput

**CloudWatch and EBS**

* Important EBS Volume CloudWatch metrics:
  + **VolumeIdleTime:** number of seconds when no read / write is submitted
  + **VolumeQueueLength:** Number of operations waiting to be executed. High number means probably an IOPS or application issue
  + **BurstBalance**: If it becomes - we need a volume with more IOPS
* **Gp2 volume** types: **5 minutes interval**
* **Io 1 volume** types: **1 minute interval**
* **EBS volumes** have **status check**
  + **Okay** - The volume is performing good.
  + **Warning** - performance below expected.
  + **Impaired** - stalled, performance severely degraded

**EFS – Elastic File System**

* Managed NFS ( network file system ) that can be mounted on many EC2
* EFS works with EC2 instances in multi-AZ
* Highly available, scalable, expensive (3x gp2), pay per use
* USe cases: content management, web serving, data sharing, Wordpress
* Uses NFSv4.1 protocol
* Ususe security group to control access to EFS
* **Compatible with Linux based AMI ( not Windows)**
* Encrypted at rest using KMS
* POSIX file system (~Linux) that has a standard file API
* File system scales automatically, pay-per-use no capacity planning

**EFS - Performance & Storage Classes**

* **EFS Scale**
  + 1000s of concurrent NFS clients, 10 Gb+ /s throughput
  + Grow to petabyte-scale network file system automatically
* **Performance mode ( set at EFS creation time)**
  + General purpose (default): latency-sensitive use cases ( web server, CMS etc…)
  + Max I/O \_ higher latency, throughput, highly parallel ( big data, media processing)
* **Storage Tiers (life cycle management feature \_ move file after N days)**
  + Standard: for frequently accessed files
  + Infrequent access (EFS-IA): cost to retrieve files, lower price to store

**Amazon S3 Overview - Buckets**

* Amazon S3 allows people to store objects (files) in “buckets” (directories)
* Buckets must have a globally unique name
* Buckets are defined at the region level
* Naming convention
  + No uppercase
  + No underscores
  + 3 - 63 characters long
  + Not an IP
  + Must start with lowercase letter or number

**Amazon S3 Overview - Objects**

* Objects ( files ) have a Key
* The **key** is the FULL path: (Represented with blue bold italics)
  + s3://my-bucket/***my\_file.txt***
  + s3://my-bucket/***my\_folder1/another\_folder/my\_file.txt***
* The **key** is composed of ***prefix*** + ***object name***
  + s3://my-bucket/***my\_folder1/another\_folder/my\_file.txt***
* There’s no concept of “directories” within buckets (although the UI will trick you to think otherwise)
* Just keys with very long names that contain slashes (“/”)

**Amazon S3 Overview - Objects (continued)**

* Object values are the content of the body:
  + Max Object Size is 5TB (5000Gb)
  + If Uploading more than 5Gb, must use “multi-part upload”
* Metadata ( list of text key / value pairs - system or user metadata)
* Tags ( Unicode key / value pair - up to 10) - useful for security / lifecycle
* Version ID (if versioning is enabled)

**Amazon S3 - Versioning**

* You can version your files in Amazon S3
* It is enabled at the **bucket level**
* Same key overwrite will increment the version: 123
* It is best practice to version your buckets
  + Protect against unintended deletes (basically the ability to restore a version)
  + Easy roll back to previous version
* Notes:
  + Any file that is not versioned prior to enabling versioning will have version “null”
  + Suspending versioning does not delete the previous versions

**S3 Encryption for Objects**

* There are 4 methods of encrypting objects in S3
  + SSE-S3: encrypts S3 objects using keys handled & managed by AWS
  + SSE-KMS: leverage AWS Key Management Service to manage encryption keys
  + SSE-C: when you want to manage your own encryption keys
  + Client Side Encryption
* It’s important to understand which ones are adapted to which situation for the exam

**SSE-S3**

* SSE-S3: Encryption using keys handled & managed by Amazon S3
* Object is encrypted server side
* AES-256 encryption type
* Must set header: “**x-amz-server-side encryption**”: “**AES256”**

**SSE-KMS**

* SSE-KMS: encryption using keys handled & managed by KMS
* KMS Advantages: user control + audit trail
* Object is encrypted server side
* Must set header: “**x-amz-server-side-encryption**”: “**aws:kms**”

**SSE-C**

* SSE-C: server-side encryption using data keys fully managed by the customer outside of AWS
* Amazon S3 does not store the encryption key you provide
* **HTTPS must be used**
* Encryption key must provided in HTTP headers, for every HTTP request made

**Client Side Encryption**

* Client library such as the Amazon S3 Encryption Client
* Clients must encrypt data themselves before sending to S3
* Clients must decrypt data themselves when retrieving from S3
* Customer fully manages the keys and encryption cycle

**Encryption in transit (SSL/TLS)**

* Amazon S3 exposes:
  + HTTP endpoint: non encrypted
  + HTTPS endpoint: encryption in flight
* You’re free to use the endpoint you want, but HTTPS is recommended
* Most clients would use the HTTPS endpoint by default
* HTTPS is mandatory for SSE-C
* Encryption in flight is also called SSL/TLS

**S3 Security**

* User Based
  + IAM policies - which API calls should be allowed for a specific user from IAM console
* Resource Based
  + Bucket Policies - bucket wide rules from the S3 console - allows cross account
  + Object Access Control List (ACL) - finder grain
  + Bucket Access Control List (ACL) - less common
* **Note:** an IAM principal can access an S3 Object if
  + The user IAM permissions allow it OR the resource policy ALLOWS it
  + AND there’s no explicit DENY

**S3 Bucket Policies**

* JSON based policies
  + Resources: buckets and objects
  + Actions: Set of API to Allow or Deny
  + Effect: Allow / Deny
  + Principal: The account or user to apply to the policy to
* Use S3 bucket for policy to:
  + Grant public access to the bucket
  + Force objects to be encrypted at upload
  + Grant access to another account ( Cross Account )

**Bucket settings for Block Public Access**

* Block public access to buckets and objects granted through
  + New access control lists ( ACLs)
  + Any access control lists (ACLs)
  + New public bucket or access point policies
* Block public and cross account access to buckets and objects through any public bucket or access point policies
* **These settings were created to prevent company data leaks**
* If you know your bucket should never be public, leave these on
* Can be set at the account level

**S3 Security - Other**

* Networking:
  + Supports VPC Endpoints (for instances in VPC without www internet
* Logging and Audit:
  + S3 Access Logs can be stored in other S3 bucket
  + API calls can be logged in AWS CloudTrail
* User Security:
  + MFA Delete: MFA (multi factor authentication) can be required in versioned buckets to delete objects
  + Pre Signed URLS: URLs that are valid only for a limited time ( ex: premium vide service for logged in users)

**S3 Websites**

* S3 can host static websites and have them accessible on the WWW
* The website URL will be:
  + <bucket-name>.s3-website-<AWS-region>.amazonaws.com
  + Or
  + <bucket-name>.s3-website.<AWS-region>.amazonaws.com
* If you get a 403 (Forbidden) error, make sure the bucket policy allows public reads!

**CORS - Explained**

* An **origin** is a scheme (protocol), host (domain) and port
  + E.g.: <https://www.example.com> (implied port is 443 for HTTPS, 80 for HTTP)
* CORS means Cross-Origin Resource Sharing
* **Web Browser** based mechanism to allow requests to other origins while visiting hte main origin
* Same origin: <https://example.com/app1> & <http://example.com/app2>
* Different origins: <http://www.example.com> & <http://other.example.com>
* The requests won’t be fulfilled unless the other origin allows for the requests, using **CORS Headers (ex: Access-Control-Allow-Origin)**

**S3 CORS**

* If a client does a cross-origin request on our S3 bucket, we need to enable the correct CORS headers
* It's A popular exam question
* You can allow for a specific origin or for \* (all origins)

**Amazon S3 - Consistency Model**

* **Read after write consistency for PUTS of new objects**
  + As soon as a new object is written we can retrieve it
  + Ex: (PUT 200 => GET 200)
* **This is true except if we ded a GET before to see if the object existed**
  + (GET 404 => PUT 200 => GET 404) - eventually consistent
* **Eventual Consistency for DELETES and PUTS of existing objects**
  + If we read an object after updating, we might get the older version
  + Ex: (PUT 200 => PUT 200 => GET 200 (might be older version)
  + If we delete an object, we might still be able to retrieve it for a short time
  + (DELETE 200 => GET 200)
* **NOTE: THERE’S NO WAY TO REQUEST “STRONG CONSISTENCY”**

**S3 SysOps**

* Versioning
* MFA-Delete
* Default Encryption
* Logs
* Cross Region Replications
* Pre-Signed URLS
* CloudFront
* Inventory
* Storage Tiers
* Lifecycle Rules
* Analytics
* Glacier
* Snowball
* Storage Gateway
* Athena

**S3 Versioning for SysOps**

* S3 Versioning creates a new version each time you change a file
* That includes when you encrypt a file
* It’s a nice way to get protected against hackers wanting to encrypt our data
* Deleting a file in S3 bucket just adds a delete marker on the versioning
* To delete a bucket, you need to remove all the file version within it

**S3 MFA-Delete (no way to do this through the console at this current time)**

* MFA (multi factor authentication) forces user to generate a code on a device ( usually mobile phone or hardware) before doing important operations on S3
* To use MFA-Delete, enable versioning on the S3 bucket
* You will need a MFA to
  + Permanently delete an object version
  + Suspend versioning on the bucket
* You won’t need MFA for
  + Enabling versioning
  + Listing deleted versions
* **Only the bucket owner (root account) can enable/disable MFA-Delete**
* MFA-Delete currently can only be enabled using the CLI

**S3 Default Encryption vs Bucket Policies**

* The old way to enable default encryption was to use a bucket policy and refuse any HTTP command without the proper headers:
* The new way is to use the “default encryption” option in S3
* Note: Bucket Policies are evaluated before “default encryption”

**S3 Access Logs**

* For audit purpose, you may want to log all access to S3 buckets
* Any requests made to S3, from any account authorized or denied, will be logged into another S3 bucket
* That data can be analyzed using data analysis tools
* Or amazon Athena
* The log format is at
* <https://docs.aws.amazon.com/AmazonS3/latest/dev/LogFormat.html>

**S3 Access Logs: Warning**

* Do not set your logging bucket to be the monitored bucket
* It will create a logging loop and your bucket will grow in size exponentially
* Seperate the logging buckets and the monitored bucket

**S3 Replication**

* **Cross Region Replication**
* **Same Region Replication**
* **Must enable versioning** in source and destination
* Buckets can be indifferent accounts
* Copying is asynchronous
* Must give proper IAM permissions to S3
* CRR - Use Cases: compliance, lower latency access, replication across accounts
* SRR - Use Cases: log aggregation, live replication, between production and test accounts

**S3 Replication – Notes**

* After activating, only new object are replicated ( not retroactive )
* For DELETE operations:
  + If you delete without a version ID, it adds a delete marker, not replicated
  + If you delete with a version ID, it deletes in the source, not replicated
* There is no “chaining” of replication
  + If bucket 1 has replication into bucket 2, which has replication into bucket 3
  + Then objects created in bucket 1 are not replicated to bucket 3

**S3 Bucket Policies Review**

* [**https://docs.aws.amazon.com/AmazonS3/latest/dev/example-bucket-policies.html**](https://docs.aws.amazon.com/AmazonS3/latest/dev/example-bucket-policies.html)

**Granting Permissions to Multiple Accounts with Added Conditions**

* In the Statement portion the “Principal” section of the bucket policy is where you can add permissions for multiple AWS accounts
* “Principal” : { “AWS”: [“arn:aws:iam::111122223333:root]},
* The added condition of allowing only read actions is done with
* “Condition”:{“StringEquals”:{“s3:x-amz-acl”:[“public-read”]}}

{

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

"Statement":[

{

"Sid":"AddCannedAcl",

"Effect":"Allow",

"Principal": {"AWS": ["arn:aws:iam::111122223333:root","arn:aws:iam::444455556666:root"]},

"Action":["s3:PutObject","s3:PutObjectAcl"],

"Resource":["arn:aws:s3:::*examplebucket*/\*"],

"Condition":{"StringEquals":{"s3:x-amz-acl":["public-read"]}}

}

]

}

**Granting Read-Only Permissions to an Anonymous User**

* Grants get permissions to any public anonymous users.
* Useful for configuring a bucket to act as a website.
* The principal “\*” designates that anyone can access it.

{

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

"Statement":[

{

"Sid":"PublicRead",

"Effect":"Allow",

"Principal": "\*",

"Action":["s3:GetObject"],

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

}

]

}

**Restricting Access to Specific IP Addresses**

* This one only allows users from a specific IP address range to access the information in the bucket
* By saying principal “\*” this will allow any resources
* Within the condition “NotIpAddress” : {“aws:SourceIp” : “54.240.143.0/24”}
  + This will allow all of the IPv4 ips at the specific ip address
  + NOTE
    - The ***NotIpAddress*** condition specifies that all IP address within a specified range are valid
    - The ***IpAddress*** condition specifies that all IP addresses except the specified IP address or range

{

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

"Id": "S3PolicyId1",

"Statement": [

{

"Sid": "IPAllow",

"Effect": "Deny",

"Principal": "\*",

"Action": "s3:\*",

"Resource": "arn:aws:s3:::*examplebucket*/\*",

"Condition": {

"NotIpAddress": {"aws:SourceIp": "54.240.143.0/24"}

}

}

]

}

**Restricting Access to a Specific HTTP Referer**

* This policy allows any user get information from the bucket if the referrer domain matches
* This is done by the “StringLike”:“aws:Referer: [“[http://www.example.com/\*](http://www.example.com/*)”, “[http://example.com/\*](http://example.com/*)”]}

{

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

"Id":"http referer policy example",

"Statement":[

{

"Sid":"Allow get requests originating from www.example.com and example.com.",

"Effect":"Allow",

"Principal":"\*",

"Action":"s3:GetObject",

"Resource":"arn:aws:s3:::*examplebucket*/\*",

"Condition":{

"StringLike":{"aws:Referer":["http://www.example.com/\*","http://example.com/\*"]}

}

}

]

}

**Granting Permission to an Amazon CloudFront OAI**

* By using this policy you can grant a CloudFront Origin Access Identity permission to get objects in your Amazon s3 buckets.
* You can use a CloudFront OAI to allow users to access objects in your bucket through cloudfront but not directly through Amazon S3.
* By using the OAI’s ID as the policy’s Principal
  + "AWS": "arn:aws:iam::cloudfront:user/CloudFront Origin Access Identity EH1HDMB1FH2TC"

{

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

"Id": "PolicyForCloudFrontPrivateContent",

"Statement": [

{

"Effect": "Allow",

"Principal": {

"AWS": "arn:aws:iam::cloudfront:user/CloudFront Origin Access Identity *EH1HDMB1FH2TC*"

},

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::*aws-example-bucket*/\*"

}

]

}

**S3 pre-signed URLs**

* Can generate pre-signed URLs using SDK or CLI
  + For downloads (easy, can use the CLI)
  + For uploads (harder, must use the SDK)
* Valid for a default of 3600 seconds, can change timeout with --expires-in[TIME\_BY\_SECONDS] argument
* Users given a pre-signed URL inherit the permissions of the person who generated the URL for GET / PUT
* Examples:
  + Allow only logged-in users to download a premium video on your S3 bucket
  + Allow an ever changing list of users to download files by generating URLS dynamically
  + Allow temporarily a user to upload a file to a precise location in our bucket

**AWS CloudFront**

* Content Delivery Network (CDN)
* Improves read performance, content is cached at the edge
* 216 points of presences (currently)
* DDoS protection, integration with Shield, AWS Web Application Firewall
* Can expose external HTTPS and can talk to internal HTTPS backends

**CloudFront - Origins**

* **S3 Bucket**
  + For distributing files and caching them at the edge
  + Enhanced security with CloudFront **Origin Access Identity ( OAI )**
  + CloudFront can be used as an ingress ( to upload files to S3 )
* **Custom Origin (HTTP)**
  + Application Load Balancer
  + EC2 instance
  + S3 website ( must first enable the bucket as a static S3 website )
  + Any HTTP backend you want

**CloudFront Geo Restriction**

* You can restrict who can access your distribution
  + **Whitelist**
    - Allow your users to access your content only if they’re in one of the countries on a list of approved countries
  + **Blacklist**
    - Prevent your users from accessing your content if they’re in one of the countries on a blacklist of banned countries
* The “country” is determined using a 3rd party Geo-IP database
* Use case: Copyright Laws to control access to content

**CloudFront vs S3 Cross Region Replication**

* CloudFront
  + Global Edge Network
  + Files are cached for a TTL (maybe a day)
  + **Great for static content that must be available everywhere**
* **S3 Cross Region Replication:**
  + Must be setup for each region you want replication to happen
  + Files are updated in near real-time
  + Read Only
  + **Great for dynamic content that needs to be available at low-latency in few regions**

**CloudFront Access Logs**

* CloudFront access logs: logs every request made to CloudFront into a logging S3 bucket
* Utilize Athena to search the information

**CloudFront Reports**

* It’s possible to generate reports on;
  + Cache statistics
  + Popular Objects
  + Top Referrers
  + Usage
  + Viewers
* These reports are based on the data from the Access Logs.

**CloudFront troubleshooting**

* CloudFront caches HTTP 4xx and 5xx status codes returned by S3 (or the origin server)
* 4xx error code indicates that user doesn’t have access to the underlying bucket (403)
* OR the object user is requesting is not found (404)
* 5xx error codes indicates Gateway issues

**S3 Inventory**

* Amazon S3 inventory helps manage your storage
* Audit and report on the replication and encryption status of your objects
* USe Cases:
  + Business
  + Compliance
  + Regulatory needs
* You can query all the data using Amazon Athena, Redshift, Presto, Hive, Spark…
* You can setup multiple inventories
* Data goes from a source bucket to a target bucket (need to setup policy )

**S3 Storage Classes**

* Amazon S3 Standard - General Purpose
* Amazon S3 Standard-Infrequent Access (IA)
* Amazon S3 One Zone-Infrequent Access
* Amazon S3 Intelligent Tiering
* Amazon Glacier
* Amazon Glacier Deep Archive
* Amazon S3 Reduced Redundancy Storage (deprecated -- omitted)

**S3 Standard – General Purpose**

* High durability (99.999999999%) of objects across multiple AZ
* If you store 10,000,000 objects with Amazon S3, you can on average expect to incur a loss of a single object once every 10,000 years
* 99.99% Availability over a given year
* Sustain 2 concurrent facility failures
* Use Cases: Big Data analytics, mobile & gaming applications, content distribution

**S3 Standard – Infrequent Access ( IA )**

* Suitable for data that is less frequently accessed, but requires rapid access when needed
* High durability (99.999999999%) of objects across multiple AZs
* 99.9% Availability
* Low cost compared to Amazon S3 Standard
* Sustain 2 concurrent facility failures
* Use cases: As a data store for disaster recovery, backups…

**S3 One Zone - Infrequent Access (IA)**

* Same as IA but data is stored in a single AZ
* High durability (99.999999999%) of objects in a single AZ; data lost when AZ is destroyed
* 99.5% Availability
* Low latency and high throughput performance
* Supports SSL for data at transit and encryptions at rest
* Low cost compared to IA (by 20%)
* Use Cases: Storing secondary backup copies of on-premise data, or storing data you can recreate

**S3 Intelligent Tiering**

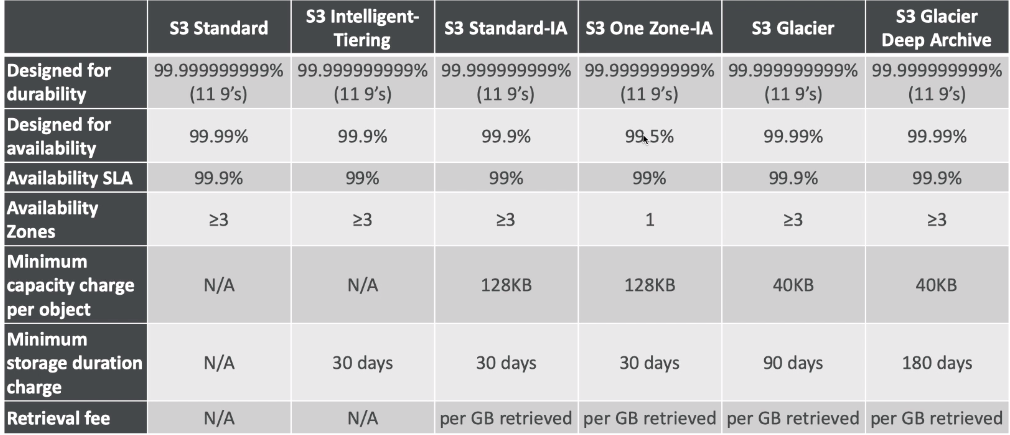
* Same low latency and high throughput performance of S3 Standard
* Small monthly monitoring and auto-tiering fee
* Automatically moves objects between two access tiers based on changing access patterns
* Designed for durability of 99.999999999% of objects across multiple Availability Zones
* Resilient against events that impact an entire AZ
* Designed for 99.9% availability over a given year

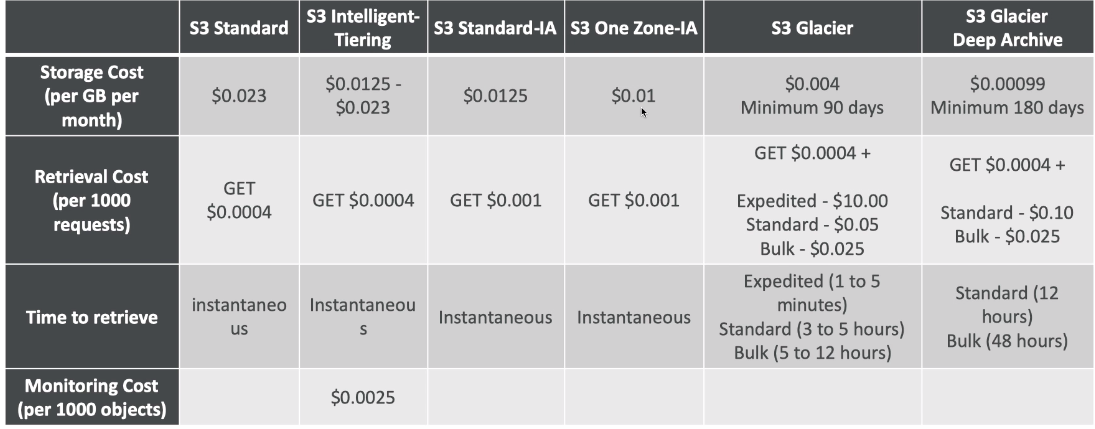
**Amazon Glacier**

* Low cost object storage meant for archiving/backup
* Data is retained for the longer term (decades)
* Alternative to on-premise magnetic tap storage
* Average annual durability is 11 0s %
* Cost per storage per month is ($0004 / GB) + retrieval cost
* Each item in glacier is called **Archive** up to 40 TB
* Archives are stored in **Vaults**

**Amazon Glacier & Glacier Deep Archive**

* Amazon Glacier – retrieval options;
  + Expedited ( 1 to 5 minutes )
  + Standard (3 - 5 hours)
  + Bulk (5 -12 hours)
  + Minimum storage duration of 90 days
* Amazon Glacier Deep Archive – for long term storage – cheaper;
  + Standard (12 hours )
  + Bulk ( 48 hours)
  + Minimum storage duration of 180 days





**S3 – moving between storage classes**

* You can transition objects between storage classes
* For infrequent accessed objects move to Standard\_IA
* Need in real, Glacier or Deep\_Archive
* Moving objects can be automated via a lifecycle configuration

**S3 Lifecycle Rules**

* **Transition actions:**
  + It defines when objects are transitioned to another storage class
    - Move objects to Standard IA class, 60 days after creation
    - Move to Glacier for archiving after 6 months
* **Expiration actions:** Configure objects to expire ( delete ) after some time
  + Access log files can be set to delete after a 365 days
  + **Can be used to delete old versions of files ( if versioning is enabled)**
  + Can be used to delta incomplete multipart uploads
* Rules can be created for a certain prefix
* Rules can be created for certain object tags as well

**S3 Lifecycle Rules – Scenario 1**

* Your application on EC2 creates image thumbnails after profile photos are uploaded to Amazon S3. These thumbnails can be easily recreated, and only need to be kept for 45 days. The source images should be able to immediately be retrieved for these 45 dyas, and afterwards, the user can wait up to 6 hours. How would you design this?
  + Trial:
    - Have them stored in a standard s3 for 45 days before the objects are transferred to S3 Glacier bulk (50% correct)
  + Solution:
    - S3 source images can be on Standard, with a lifecycle configuration to transition them to glacier after 45 days
    - S3 thumbnails can be on onezone\_ia, with a lifecycle configuration to expire them (delete them) after 45 days

**S3 Lifecycle Rules – Scenario 2**

* A rule in your company states that you should be able to recover your deleted s3 objects immediately for 16 days, although this may happen rarely. After this time, and for up to 365 days, deleted objects should be recoverable within 48 hours
  + Trial:
    - Upon S3 object deletion store in a standard\_IA bucket for 16 days, then afterward store them in amazon glacier deep archive bulk
  + Solution:
    - You need to enable S3 versioning in order to have object versions so that deleted objects are in fact hidden by a delete marker an can be recovered
    - You can transition thes noncurrent versions of the object to S3\_IA
    - You can transition afterwards these “noncurrent versions” to DEEP\_ARCHIVE

**S3 – Baseline performance**

* Amazon S3 automatically scales to high request rates, latency 100-200ms
* Your application can achieve at least **3,500 PUT/COPY/POST/DELETE and 5,500 GET/HEAD requests per second per prefix in bucket.**
* There are no limits to the number of prefixes in a bucket
* Example(object path -> prefix) :
  + Bucket/folder1/sub1/file => /folder1/sub1/
  + Bucket/folder2/sub2/file => /folder2/sub2/
  + bucket/1/file
  + bucket/2/file
* If you spread reads across all four prefixes evenly, you can achieve 22,000 requests per second for GET and HEAD

**S3 - KMS Limitation**

* If you use SSE-KMS, you may be impacted by the KMS limits
* When you upload, it calls the GenerateDataKey KMS API
* WHen you download, it calls the Decrypt KMS API
* Count towards the KMS quota per seconds (5500, 10000, 30000 req/s based on region)
* As of today, you cannot request a quota increase for KMS

**S3 Performance ( Optimization)**

* Multi-Par upload:
  + Recommended for files > 100MB
  + Must be used for files > 5Gb
  + Can Help parallelize uploads ( speed up transfers
* S3 Transfer Acceleration (upload only)
  + Increase transfer speed by transferring file to an AWS edge location which will forward the data to the S3 bucket in the target region
  + Compatible with multipart upload

**S3 Performance – S3 Byte-Range Fetches**

* Parallelize GETs by requesting specific byte ranges
* Better resilience in case of failures
* Can be used to speed up downloads
* The idea is you make multiple gets for each byte range then put them together
* Can be used to retrieve only partial data ( for example the head of a file)

**S3 Select & Glacier Select**

* Retrieve less data using SQL by performing server side filtering
* Can filter by rows and columns ( simple SQL statements)
* Less network transfer, less CPU cost client-side
* Which can increase speeds up to 400% and make it 80% cheaper

Example:

* The client makes a get of a csv with S3 Select
* Then server side filtering occurs and sends the filtered data set back to the client

**S3 Event Notifications**

* S3:ObjectCreated,
* S3:ObjectRemoved,
* S3:ObjectRestore
* S3:ObjectReplication
* Object name filtering is possible ie (\*.jpg)
* Use case: generate thumbnails of images upon being uploaded to S3
* Notifications can be used to trigger:
  + SNS
  + SQS
  + Lambda Function
* **Can create as many “S3 events” as desired**
* S3 event notifications typically deliver events in seconds but can sometimes take a minute or longer
* If two writes are made to a single non-versioned object at the same time, it is possible that only a single event notification will be sent
* If you want to ensure that an event notification is sent for every successful write, you can enable versioning on your bucket

**S3 Analytics – Storage Class Analysis**

* You can set up analytics to help determine when to transition objects from standard to standard\_IA
* Does not work for OneZone\_IA or Glacier
* Report is updated on a daily basis
* Takes about 24 to 48 hours to first start
* THis helps you put together effective lifecycle rules

**Glacier**

* Low cost storage meant for archiving / backup
* Data is retained for the longer term (decades)
* Alternative to on-premise magnetic tape storage
* Same durability as S3
* Cost per storage per month ( $0.004 / GB) + retrieval cost
* Each item in Glacier is called **Archive** ( up to 40GB)
* Archives are stored in “**Vaults**” -- Instead of buckets
* Exam Tip: Archival from S3 xxx days => use glacier

**Glacier Operations**

* 3 Glacier operations
  + **Upload**
    - Single operation or by parts ( multi-part upload) for larger archives
  + **Download**
    - First initiate a retrieval job for the particular archive, Glacier then prepares it for download. User then has a limited time to download the data from the staging server
  + **Delete**
    - Use Glacier REST API or AWS SDKs by specifying an archive’s ID
* Restore links have an expiry day
* 3 Retrieval options
  + **Expedited**
    - 1 to 5 min retrieval time
    - .03$ per GB and .01$ per request
  + **Standard**
    - 3-5 hours
    - .01$ per GB and .05$ per 1000 requests
  + **Bulk**
    - 5 - 12 hours
    - .0025$ per GB and .025$ per 1000 requests

**Glacier - Vault Policies & Vault**

* Vault is a collection of archives
* Each Vault has:
  + ONE vault access policy
  + ONE vault lock policy
* Vault Policies are written in JSON
* Vault Access Policy is similar to a bucket policy ( restrict user / account permissions )
* Vault Lock Policy is a policy you lock, for regulatory and compliance requirements
* You can only upload things to Glacier Vault via an SDK you cannot do it through the console
  + The policy is immutable, **it can never be changed ( that’s why it’s called LOCK)**
  + Example
    - Forbid deleting an archive if less than 1 year old
  + Example 2
    - Implement WORM policy
    - WORM === write once but read many
  + You will 24 hours to confirm a Vault Lock via the ID

**AWS Snowball**

* Physical data transport solution tht helps moving TBs or PBs of data in or out of AWS
* Alternative to moving data over the network ( and paying network fees )
* Secure, tamper resistant, uses KMS 256 bit encryption
* Tracking using SNS and text messages. E-ink shipping label
* Pay per transfer job
* Use cases: large data cloud migrations, DC decommission, disaster recovery
* If it takes more than a week to transfer over the network, use the Snowball service

**Snowball Process**

1. Request snowball devices from the AWS console for delivery
2. Install the snowball client on your servers
3. Connect the snowball to your servers and copy files using the client
4. SHip back the device when you’re done ( goes to the right AWS facility)
5. Data will be loaded into an S3 bucket
6. Snowball is completely wiped
7. Tracking is done using SNS, text messages and the AWS console

**Snowball Edge**

* Snowball Edges add computational capability to the device
* 100 TB capacity with either
  + Storage optimized – 24 vCPU
  + Compute optimized – 53 vCPU & optional GPU
* Supports a custom EC2 AMI so you can perform processing while on the go
* Supports custom Lambda functions
* Very useful to pre-process the data while moving
* Use case: data migration, image collation, IoT capture, machine learning

**AWS Snowmobile**

* A literal truck
* Transfer exabytes of data (1 EB = 1,000PB = 1,000,000TBs)
* Each Snowmobile has 100PB of capacity (you can get multiple snowmobiles at the same time)

**Solution Architecture: Snowball into Glacier**

* **Snowball cannot import Glacier directly**
* You have to use Amazon S3 first, and an S3 lifecycle policy

**Hybrid Cloud for Storage**

* AWS is pushing for “hybrid cloud”
  + Part of your infrastructure is on the cloud
  + Part of your infrastructure is on-premise
* This can be due to
  + Long cloud migrations
  + Security requirements
  + Compliance Requirements
  + IT strategy
* S3 is a proprietary storage technology ( unline EFS / NFS ), so how do you expose the S3 data on-premise
* AWS Storage Gateway

**AWS Storage Cloud Native Options**

* Block ( Amazon EBS and EC2 Instance Storage )
* File ( Amazon EFS )
* Object ( S3 and Glacier

**AWS storage Gateway**

* Bridge between on-premise data and cloud data in S3
* Use Cases: disaster recovery, backup & restore, tiered storage
* 3 types of storage gateway:
  + File gateway
  + Volume Gateway
  + Tape Gateway
* Behind the scenes it will put the information into EBS, S3 or Glacier
* Exam Tip: YOu need to know the differences between all 3!

**File Gateway**

* Configured S3 buckets are accessible using the NFS and SMB protocol
* Supports S3 standard, S3 IA, S3 One ZOne IA
* Bucket access using IAM roles for each File Gateway
* Most recently used data is cached in the file gateway
* Can be mounted on many servers

**Volume Gateway**

* Block storage using iSCSI protocol backed by S3
* Backed by EBS snapshots which can help resottre on-premise volumes
* **Cached volumes:** low latency access to most recent data
* **Stored Volumes:** entire data set is on premise, scheduled backups to S3

**Tape Gateway**

* Some companies have backup processes using physical tapes
* With Tape Gateway, companies use the same processes but in the cloud
* Virtual Tape Library (VTL) backed by Amazon S3 and Glacier
* Back up data using existing tape-based processes ( and iSCSI interface)
* Works with leading backup software vendors

**AWS Storage Gateway Summary**

* Exam tip: Read the question well, it will hint at which gateway to use
* On premise data to the cloud => Storage Gateway
* File access / NFS => File Gateway ( backed by S3)
* Volumes / Block Storage/ iSCSI => Volume Gateway (backed by S3 with snapshots)
* VTL Tape Solution / Backup with iSCSI => Tape gateway (backed by S3 and Glacier)

**AWS Athena**

* **Serverless** service to perform analytics **directly against S3 files**
* Uses SQL languages to query the files
* Has a JDBC / ODBC driver
* Charged per query and amount of data scanned
* Supports CSV, JSON, ORC, Avcro , and Parquet ( built on Presto)
* Use cases: Business intelligence / analytics / reporting, analyze & query VPC Flow Logs, ELB Logs, CloudTrail trails, etc….
* Exam Tip: Analyze data directly on S3 => use Athena

**Databases Section**

* RDS in Depth
  + Basic refreshers
  + Multi AZ vs Read Replicas
  + Parameter Groups
  + Backups & Snapshots
  + Security
  + API
  + **CloudWatch and Performance Insights**
* Aurora in depth
* ElastiCache Refresher

**AWS RDS Overview**

* RDS stands Relational Database Service
* It’s managed DB service for DB use SQL as a query language
* It allows you to create databases in the cloud that are managed by AWS
  + Postgres
  + Oracle
  + MySQL
  + MariaDB
  + Oracle
  + Microsoft SQL Server
  + Aurora (AWS Proprietary database )

**Advantage over using RDS versus deploying a DB on EC2**

* Managed service
* OS patching level
* Continuous backups and restore to specific timestamp (Point in Time Restore)
* Monitoring dashboards
* Read replicas for improved read performance
* Multi AZ setup for DR ( Disaster Recovery)
* Maintenance windows for upgrades
* Scaling capability ( vertical and horizontal)
* BUT you can’t SSH into your instances

**RDS Read Replicas for read Scalability**

* Up to 5 read replicas
* Within AZ, CrossAz, ore even cross region
* Replication is ASYNC so reads are eventually consistent
* Replicas can be promoted to their own DB
* Applications must update the connection string to leverage read replicas
* Only the master can take writes

**RDS Multi AZ ( Disaster Recovery)**

* SYNC replication
* One DNS name – automatic app failover to standby
* This increases **availability**
* Fialover in case of loss of AZ loss of network instance or storage failure
* No manual intervention in apps
* Only use for disaster recovery
* Not used for scaling

**RDS Backups**

* Backups are automatically enabled in RDS
* Automated Backups:
  + Daily full snapshot of the database
  + Capture transaction logs in real time
  + => ability to restore to any point in time
  + 7 days retention ( however can be increased to 35 days)
* DB Snapshots:
  + Manually triggered by the user
  + Retention of backup for as long as you want

**RDS**

* Encryption at rest capability with AWS KMS - AES-256 encryption
* SSL certificates to encrypt data to RDS in flight
* To enforce SSL:
  + PostgreSQL: rds.force\_ssl=1 in the AWS RDS Console (Parameter Groups)
  + MySQL: Within the DB:
    - GRAN USAGE ON \*.\* TO ‘mysqluser’@’%’ REQUIRE SSL;
  + To connect using SSL
    - TO provide the SSL Trust certificate ( can be download from AWS)
    - Provide SSL options when connecting to database

**RDS Security**

* RDS databases are usually deployed within a private subnet, not in a public one
* RDS Security works by leveraging security groups ( the same concept as for EC2 instances ) – it controls who can **communicate** with RDS
* IAM policies help control who can **manage** AWS RDS
* Traditional Username and Password can be used to **login** to the database
* IAM users can now be used too ( for MySQL / Aurora – NEW)

**RDS vs Aurora**

* Aurora is a proprietary technology from AWS (not open sourced)
* Postgres and MySQL are booth supported as Aurora DB ( that means your drivers will work as if Aurora was Postgres or MySQL database)
* Aurora is “AWS cloud optimized” and claims 5x performance improvement over MySQL on RDS, over 3x the performance of Postgres on RDS
* Aurora storage automatically grows in increments of 10Gb all the way up to 64Tb
* Aurora can have 15 replicas while MySQL has 5, and the replication process is faster ( sub 10 ms replica lag)
* Failover in Aurora is instantaneous. It’s HA native
* Aurora cost more than RDS ( 20% more) – but it's more efficient

**RDS Multi AZ in depth**

* **Multi AZ is not used to support the reads**
* The **failover happens only in the following conditions:**
  + The primary DB instance fails
  + An Availability Zone outage
  + The DB instance server type is changed
  + The operating system of the DB instance is undergoing software patching.
  + A manual failover of the DV instance was initiated using REboot with failover
* **No failover DB operations:** long-running queries, deadlock, or database corruption errors.
* Endpoint is the same after failover ( no URL change in application needed )
  + A Multi AZ only has ONE DNS ENDPOINT
* Lower maintenance impact it happens on the standby, which is then promoted to master
* Backups are created from the standby
* Multi AZ is only within a single region, not cross regio. Region outages impacts availability
* THIS DOES NOT IMPROVE READ PERFORMANCE

**RDS Read Replicas in Depth**

* Read Replicas help scaling read traffic
* A Read Replica can be promoted as as standalone database ( manually done)
* Read Replicas can be within AZ, Cross AZ or Cross Region
* Each Read Replica has its own DNS endpoint
* You can have Read Replicas of Read Replicas
* Read Replicas can be Multi-AZ
* Read Replicas help with Disaster Recovery by using a cross region read replica ( that can then be promoted to a standalone database if needed)
* RDS Read Replicas are not supported for ORACLE
* **Read Replicas can be used to run BI / Analytics Reports for example**

**DB Parameter Groups**

* You can configure the DB engine using Parameter Groups
* Dynamic parameters are applied immediately
* Static parameters are applied after instance reboot
* You can modify parameter group associated with a DB ( must reboot)
* See documentation for list of parameters for a DB technology
* **Must-know parameter:**
  + PostgreSQL / SQL Server: rds.force\_ssl=1=> SSL connections
  + Remider: for SSL on MySQL / MariaDB you must run:
  + GRANT SELECT ON mydatabasename.\* TO ‘myuser’@’%’ IDENTIFIED BY ‘.....’ REQUIRE SSL;

**RDS Backup Vs Snapshot**

* **Backups**
  + Backups are “ continuous: and allow point in time recovery
  + Backups happen during maintenance windows
  + When You delete a DB instance you can retain automated backups
  + Backups have a retention period you seen between 0 - 35 days
* **Snapshots**
  + Snapshots takes IO operations and can stop the database from seconds to minutes
  + Snapshots taken on Multi AZ DB don’t impact the master - just the standby
  + Snapshots are incremental after the first snapshot (which is full)
  + You can copy & share DB Snapshots
  + Manual Snapshots don’t expire
  + You can take a “final snapshot” when you delete you DB

**RDS Security for SysOps**

* **Encryption at rest:**
  + Is done only when you first create the DB instance
  + Or: unencrypted DB => snapshot => copy snapshot as encrypted => create DB from snapshot
* **Your responsibility:**
  + Check the ports / IP / security group inbound rules in DB’s SG
  + In database user creation and permissions
  + Creating a database with or without public access
  + Ensure parameter groups or DB is configured to only allow SSL connections
* **AWS responsibility :** 
  + No SSH access
  + No manual DB patching
  + No manual OS patching
  + No way to audit the underlying instance

**RDS API for SysOps**

* **DescribeDBInstances** API –
  + Helps to get a list of all the DB instances you have deployed including read replicas
  + Helps to get the DB version
* **CreateDBSnapshot** API – Make a snapshot of a DB
* **DescribeEvents** API – Helps to return information about events related to your DB Instance
* **RebootDBInstance** API - Helps to initiate a ‘forced’ failover by rebooting DB instance

**RDS with CloudWatch**

* CloudWatch metrics associated with RDS ( gathered from the hypervisor)
  + DatabaseConnections
  + SwapUsage
  + ReadIOPS / Write IOPS
  + ReadLatency / WriteLatency
  + ReadThroughPut / WriteThroughPut
  + DiskQueueDepth
  + FreeStorageSpace
* Enhanced Monitoring ( gathered from an agent on the Db instance )
  + Useful when you need to see how different processes or threads use the CPU
  + Access to over 50 new CPU, memory, file system and dis I/O metrics

**RDS Performance Insights**

* Visualize your database performance and analyze any issues that affect it
* With the performance insights dashboard, you can visualize the database load and filter the load
  + By waits => find the resource that is the bottleneck ( CPU, IO, lock, etc)
  + By SQL statements -> find the SQL statement that is the problem
  + By Hosts => find the server that is using our DB the most
  + By users => find the user that is using our DB the most
* **DBLoad = the number of active sessions for the DB engine**
* **You can view the SQL queries that are putting load on your database**

**Amazon Aurora**

* Aurora is a proprietary technology from AWS ( not open sourced )
* Postgres and MySQL are both supported as Aurora DB ( that means your drivers will work as if Aurora was a Postgres or MySQL database)
* Aurora is “AWS Cloud optimized” and claims 5x performance improvement over mySWL on RDS, over 3x the performance of Postgres on RDS
* Aurora storage automatically grows in increments of 10Gb, up to 64TB
* Aurora can have 15 replicas while MySQL has 5, and the replication process is faster ( sub 10 ms replica lag)
* Failover in Aurora is instantaneous. It’s HA native
* Aurora costs more than RDS ( 20% more) - but is more efficient

**Aurora High Availability and Read Scaling**

* 6 copies of your data across 3 AZ:
  + 4 copies out of 6 needed for writes
  + 3 copies out of 6 needed for reads
  + Self healing with peer-to-peer replication
  + Storage is striped across 100s of volumes
* One Aurora Instance takes writes (master)
* Automated failover for master in less than 30 seconds
* Master + up to 15 Aurora Read Replicas serve reads
* Support for Cross Region Replication

**Aurora DB Cluster**

* Shared storage volume, with auto expanding by 10G to 64TB
* Writer endpoint that points to the master
* Auto Scaling
* Reader Endpoint => helps with connection load balancing which will be connected to all the read replicas

**Features of Aurora**

* Automatic fail-over
* Backup and recovery
* Isolation And security
* Industry compliance
* Push button scaling
* Automated patching with zero downtime
* Advanced monitoring
* Routine maintenance
* Backtrack: Restore data at any point of time without using backups

**Aurora Security**

* Encryption at rest using KMS
* Automated backups, snapshots and replicas are also encrypted
* Encryption in flight using SSL ( same process as MySQL or Postgres)
* Authentication using IAM
* You are responsible for protecting the instance with security groups
* You can’t SSH

**Aurora Serverless**

* No need to choose an instance size
* Only supports MySQL 5,6 ( as of Jan 2019) & Postgres (beta)
* Helpful when you can’t predict the workload
* Db cluster starts, shutdown and scales automatically based on CPU / connections
* Can migrate from Aurora Cluster to Aurora Serverles and vice versa
* Aurora Serverless usage is measured in ACU ( aurora capacity units)
* Billed in 5 minutes increments of ACU
* Note: Some features of Aurora aren’t supported in Serverless mode, be sure to check the documentation if you plan on using it

**AWS ElastiCache Overview**

* The same way RDS is to get managed Relational Databases…
* Elasticache is to get managed redis or memcached
* Caches are in-memory databases with really high performance , low latency
* Helps reduce load off of databases for read intensive workloads
* Helps make your application stateless
* Write scaling using sharding
* Read scaling using read replicas
* MultiAz with Failover Capability
* AWS takes care of OS maintenance / patching optimizations, setups, configuration, monitoring, failure recovery and backups

**ElastiCache Solution Architectures -DB Cache**

* Applications queries ElastiCache, if not available, get from RDS and store in ElastiCache
* Helps relieve load in RDS
* Cache must have an invalidation strategy to make sure only the most current data is used in there.

**ElastiCache Solution Architecture – User Session Store**

* User logs into any of the application
* The application writes the session data into ElastiCache
* The user hits another instance of our application
* The instance retrieves the data and the user is already logged in

**Elasticache – Redis vs Memcached**

* REDIS
  + **MultiAZ** with Auto-Failover
  + **Read Replicas** to scale reads and have **high availability**
  + Data Durability using AOF
  + **Backup and Restore features**
* MEMCACHED
  + Multi-node for partitioning of data (sharding)
  + **Non persistent**
  + **No backup and restore**
  + Multi-threaded architecture

**AWS Monitoring Audit and Performance**

**AWS CloudWatch Metrics**

* Cloudwatch provides metrics for every services in AWS
* **Metric** is a variable to monitor **( CPUUtilization, NetworkIn …)**
* Metrics belong to **namespaces**
* **Dimension** is an attribute of a metric (instance id, environment, etc…)
* Up to 10 dimensions pere metric
* Can create CloudWatch dashboards of metrics

**AWS CloudWatch EC2 Detailed Monitoring**

* EC2 instance metrics have metrics “**every 5 minutes**”
* With detailed monitoring ( for a cost ), you get data “**every 1 minute**”
* Use detailed monitoring if you want to prompt scale you Automatic Scaling Groups
* Note:
  + EC2 Memory usage is by default not pushed ( you must manually push from inside the instance as a custom metric)

**AWS CloudWatch Custom metrics**

* Possibility to define and send your own custom metrics to CloudWatch
* Ability to use dimensions ( attributes) to segment metrics
  + Instance.id
  + Environment.name
* Metric Resolution:
  + Standard: **1 minute**
  + High Resolution: up to **1 second** ( StorageResolution API parameter) – **higher cost**
* Use API call **PutMetricData**
* Use exponential back off in case of throttle errors

**CloudWatch Dashboards**

* Great way to set up dashboards for quick access to keys metrics
* **Dashboards are global**
* **Dashboards can include graphs from different regions**
* You can change the time zone & time range of the dashboards
* You can set up automatic refresh (10s, 1m, 2m, 5m, 15m )
* Pricing
  + 3 dashboards (upto 50 metrics) for free
  + $3/dashboard/month afterwards

**AWS CloudWatch Logs**

* Applications can send logs to CLoudWatch using the SDK
* CloudWatch can collect logs from:
  + Elastic Beanstalk: collection of logs from application
  + ECS: collection from containers
  + AWS Lambda: collection from function logs
  + VPC Flow Logs: VPC specific logs
  + API Gateway
  + CloudTrail based on filter
  + CloudWatch log agents: for example on EC2 machines
  + Route53: Log DNS queries
* CloudWatch Logs can go to:
  + Batch exporter to S3 for archival Stream to ElasticSearch cluster for further analytics
* Logs Storage architecture:
  + Log groups: arbitrary name, usually representing an application
  + Log Stream: instances within application / log files/ containers
* Can define log expiration policies ( never expire, 30 days, etc…)
* Using the AWS CLI we can tail CloudWatch logs
* To send logs to CLoudWatch, make sure IAM permissions are correct!
* Security: encryption of logs using KMS at the Group level

**CloudWatch Logs Metric Filter & Insights**

* CloudWatch Logs can use filter expressions
  + For example, find a specific IP inside of a log
  + Metric filters can be used to trigger alarms
* CloudWatch Logs Insights ( new – Nov 2018) can be used to query logs and add queries to CloudWatch Dashboards

**AWS CloudWatch Alarms**

* Alarms are used to trigger notifications for any metric
* Alarms can go to Auto Scaling, EC2 Actions, SNS notifications
* Various Options ( sampling, % of use, max , min etc…)
* Alarm States:
  + OK
  + INSUFFICIENT\_DATA
  + ALARM
* Period:
  + Length of time in seconds to evaluate the metric
  + High resolution custom metrics: can choose 10 or 30 sec

**CloudWatch Alarm Targets**

* Stop, Terminate, Reboot, or Recover an EC2 instance
* Trigger Auto Scaling Action
* Send notification to SNS ( from which you can do pretty much anything)

**CloudWatch Alarm: Good to Know**

* Alarms can be created based on CloudWatch Logs Metrics Filters
* CloudWatch doesn’t test or validate the actions that is assigned
* To test alarms and notifications set the alarm state to Alarm using the CLI
  + **aws cloudwatch set-alarm-state --alarm-name “name of alarm” --state-value ALARM --state-reason “testing purpose”**

AWS CloudWatch Events

* Source + Rule => Target
* Schedule: Cron jobs
* Event Pattern: Event rules to react to a service doing something
  + Ex. CodePipeline state changes!
* Triggers to Lambda functions, SQS/SNS/Kinesis Messages
* CloudWatch Event creates a small JSON document to give information about the change.

**AWS CloudTrail**

* PRovides governance, compliance and audit for your AWS Account
* CloudTrail is enabled by default
* Get an history of events / API calls made within your AWS Account by:
  + Console
  + SDK
  + CLI
  + AWS Services
* Can put logs from CloudTrail into CloudWatch logs
* If a resource is deleted in AWS, look into CloudTrail first!
* CloudTrail shows the past 90 days of activity
* The default UI only shows “Create”, “Modify” or “Delete” events
* CloudTrail
  + Get a detailed list of all the events you choose
  + Ability to store these events in S3 for further analysis
  + Can be region specific or global specific
* CloudTrail Logs have SSE-S3 encryption when placed into S3
* Control access to S3 using IAM, Bucket policy, etc…

**AWS Config**

* Helps with auditing and compliance of youAWS resources
* Helps record configurations and changes over time
* Helps record compliance over time
* Possibility of string AWS config data into S3 ( can be queried by Athena)
* Questions that can be solved by AWS Config:
  + Is there unrestricted SSH access to my security groups?
  + Do my buckets have any public access?
  + How has my ALB configuration changed over time?
* You can receive alerts ( SNS notifications ) for any changes
* AWS Config is a per-region service
* Can be aggregated across regions and accounts

**Config Rules**

* Can use AWS managed config rules ( over 75)
* Can make custom config rules ( must be defined in AWS Lambda )
  + Evaluate if each EBS disk is of type gp2
  + Evaluate if each EC2 instance is t2.micro
* Rules can be evaluated triggered:
  + For each config change
  + And / or : at regular time intervals
* Pricing:
  + No free tier
  + ~$2 USD per active rule per region per month ( less after 10 rules)

**AWS Config Resource**

* View compliance of a resource over time
* View configuration of a resource over time
* View CloudTrail API calls if enabled

**CloudWatch vs CloudTrail vs Config**

* **CloudWatch**
  + Performance monitoring ( metrics, CPU, network, etc….) & dashboard
  + Events & Alerting
  + Log Aggregation & Analysis
* **CloudTrail**
  + Record API calls made within your Account by everyone
  + Can define trails for specific resources
  + Global Service
* **Config**
  + Record configuration changes
  + Evaluate resources against compliance rules
  + Get timeline of changes and compliance

**For an Elastic Load Balancer**

* **CloudWatch:**
  + Monitoring incoming connections metric
  + Visualize error codes as a percentage over time
  + Make a dashboard to get an idea of your load balancer’s performance
* **Config**
  + Track security group rules for the LoadBalancer
  + Track configuration changes for the Load balancer
  + Ensure an SSL certificate is always assigned to the Load Balancer ( compliance)
* **CloudTrial:**
  + Track who made any changes to the Load Balancer with API calls

**AWS Status - Service Health Dashboard**

* SHows all regions, all services health
* Shows historical information for each day
* Has an RSS feed you can subscribe to
* <https://status.aws.amazon.com/>

**AWS Personal Health Dashboard**

* Global Service
* Show how AWS outages directly impact you
* Shows impact on you resources
* List issues and actions you can take to remedy it

**AWS Organizations**

* Global Service
* Allows to manage multiple AWS accounts
* The main account is the master account you can’t change it
* Other accounts are member accounts
* Member accounts can only be part of one organization
* Consolidated Billing across all accounts - single payment method
* Pricing benefits from aggregated usage ( volume discount for EC2, S3, etc….)
* API is available to automate AWS account creation

**Multi Account Settings**

* Create accounts per department, per cost center, per dev/ test / prod, based on regulatory restrictions ( using SCP) , for better resource isolation ( ex: VPC), to have separate per account service limits isolated account for logging
* Multi Account vs One Account Multi VPC
* Use tagging standards for billing purposes
* Enable CloudTrail on all accounts, send logs to a centralized S3 account
* Send CloudWatch Logs to central logging account
* Establish Cross Account Roles for Admin Purposes

**Organizational Units (OU) :**

* Based on how a company set this up
* <https://aws.amazon.com/answers/account-management/aws-multi-account-billing-strategy/>

**Service Control Policies ( SCP) => Look like IAM policies**

* Whitelist or blacklist IAM actions
* Applied at the OU or Account level
* Does not apply to the master account
* SCP is applied to all the Users and Roles of the account including Root
* The SCP does not affect service-linked roles
  + Service-linked roles enable other AWS services to integrate with AWS organizations and can’t be restricted by SCPs.
* SCP must have an explicit Allow ( does not allow anything by default)
  + Use Cases
    - Restrict access to certain services (for example: can’t use EMR)
    - Enforce PCI compliance by explicitly disabling services
* <https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_examples-scps.html>

**AWS Organizations –Moving Accounts**

To migrate accounts from one organization to another

1. Remove the member account from the old organization
2. Send an invite from the new organization
3. Accept the invite of the new organization from the member account

**If you want the master account of the old organization to also join to the new organization, do the following:**

1. Remove the member accounts from the organizations using the procedure above
2. Delete the old organization
3. Repeat the process above to invite the old master account to the new organization

**AWS Service Catalog**

* Users that are new to AWS have too many options, and may create stacks that are not compliant / inline with the rest of the organizations
* Some users just want a quick self-service portal to launch a set of authorized products pre-defined by admins
* Includes: virtual machines, databases, storage options, etc…
* Enter AWS Service Catalog!
* Create and manage catalogs of IT services that are approved on AWS
* The “products” are CloudFormation templates
* Ex: Virtual machine images, Servers, Software, Databases, Regions, Ip Address ranges
* CloudFormation helps ensure consistency, and standardization by admins
* They are assigned to Portfolios (teams)
* Teams are presented a self-service portal where they can launch the products
* All the deployed products are centrally managed deployed services
* Helps with governance, compliance, and consistency
* Can give user access to launching products without requiring deep AWS knowledge
* Integrations with”self-service portals” such as ServiceNow

**AWS Billing Alarms**

* **Billing data metric is stored in CloudWatch us-east-1**
* Billing data are for overall **worldwide** AWS costs
* It’s for actual cost, not for project costs

**AWS Cost Explorer**

* A graphical tool to view and analyze your costs and usage
* Review charges and cost associated with your AWS account or org
* Forecast spending for the next 3 months
* Get recommendations for which EC2 Reserved Instances to purchase
* Access to default reports
* API to build custom cost management applications

**AWS Budgets**

* Create budget and send alarms when costs exceeds the budget
* 3 types of budgets: USage, Cost, Reservation
* For Reserved Instances (RI)
  + Track utilization
  + Supports EC2, ElastiCache, RDS , Redshift
* Up to 5 SNS notifications per budget
* Can filter by :
  + Service
  + Linked Account
  + Tag
  + Purchase Option
  + Instance Type
  + Region
  + Availability Zone
  + API Operation,
  + Etc
* Same Options As AWS Cost Explorer
* 2 budgets are free, then $0.02/day/budget

**AWS Cost Allocation Tags**

* With tags we can track resources that relate to each other
* With cost allocation tags we can enable detailed costing reports
* **Just like tags, but they show up as columns in reports**
* AWS Generated Cost Allocation Tags
  + Automatically applied to the resources you create
  + Starts with prefix aws (eg. aws: createdBy)
  + They’re not applied to resources created before the activation
* User tags
  + Defined by the user
  + Starts with the prefix user:
* Cost Allocation Tags just appear in the Billing Console
* Takes up to 24 hours for the tags to show up in the report

**AWS Shared Responsibility Model**

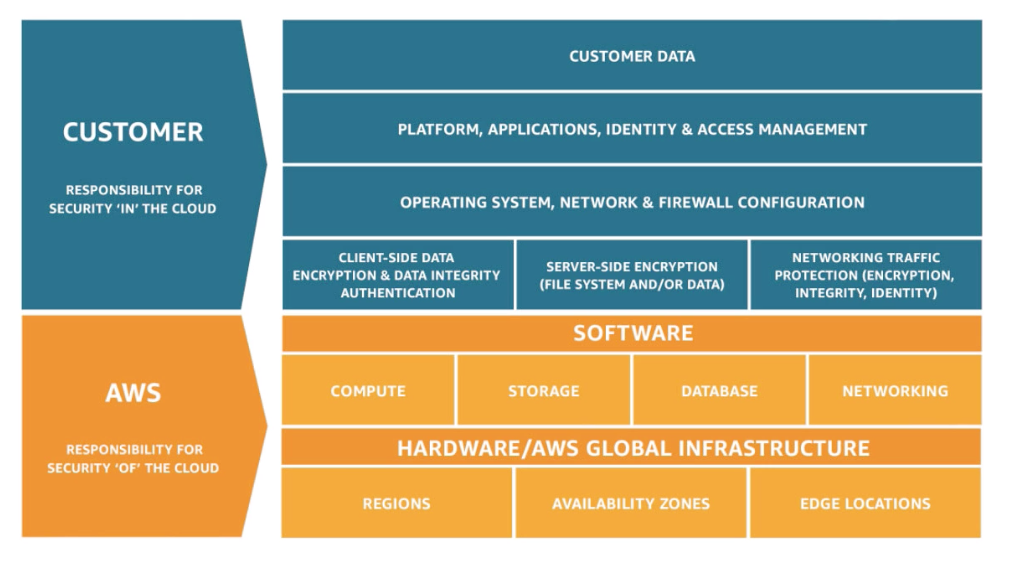
* AWS responsibility - security **of** the Cloud
  + Protecting infrastructure ( hardware, software, facilities, and networking) that runs all of the AWS services
  + Managed services like S3, DynamoDB, RDS, etc…
* Customer responsibility - Security **in** the cloud
  + For EC2 instance, the customer is responsible for management of the guest OS ( including security patches and updates), firewall & network configuration, IAM etc

**Example, for RDS**

* AWS responsibility
  + Manage the underlying EC2 instance, disable SSH access
  + Automated DB patching
  + Automated OS patching
  + Audit the underlying instance and disks & guarantee it functions
* Your responsibility
  + Check the ports / IP / security group inbound rules in DB’s SG
  + In-database user creation and permissions
  + Creating a database with or without public access
  + Ensure parameter groups or DB is configured to only allow SSL connections
  + Database encryption setting

Example, for S3

* AWS responsibility
  + Guarantee you get unlimited storage
  + Guarantee you get encryption
  + Ensure separation of the data between different customers
  + Ensure AWS employees can’t access your data
* Your responsibility
  + Bucket configuration
  + Bucket policy / public settings
  + IAM users and roles
  + Enabling encryption



**What’s a DDOS (Distributed Denial-of-service) Attack?**

* An attacker will attempt to flood the bandwidth of your server

**DDOS Protection on AWS**

* AWS Shield Standard:
  + Protects against DDOS attack for your website and applications, for all customers at no additional costs
* AWS Shield Advanced
  + 24/7 premium DDoS protection
* AWS WAF:
  + Filter specific requests based on rules
* CloudFront and Route 53:
  + Availability protection using global edge network
  + Combined with AWS Shield, provides attack mitigation at the edge locations of CloudFront
* Be ready to scale – leverage AWS Auto Scaling
* Separate static resources (S3 / CloudFront) from dynamic ones (EC2 / ALB)
* Whitepaper
  + <https://d1.awsstatic.com/whitepapers/Security/DDos_White_Paper.pdf>

**AWS Shield**

* AWS Shield Standard:
  + Free service that is activated for every AWS customer
  + Provides protection from attacks such as SYN/UDP Floods, Reflection attacks and other layer 3/ layer 4 attacks
* AWS Shield Advanced
  + Optional DDos Mitigation Service ($3,000 per month per organization)
  + Protect against more sophisticated attack on CloudFront, Route 53, Classic, Application & Network Load Balancer (select regions), Elastic IP/ EC2
  + 24/7 access to AWS DDoS response team (DRP)
  + Protect against higher fees during usage spikes due to DDoS

**AWS WAF – Web Application Firewall**

* Protects your web applications from common web exploits
* Define customizable web security rules:
  + Control which traffic to allow or block to your web applications
  + Rules can include: IP addresses, HTTP headers, HTTP body, or URI strings
  + Protects from common attack - SQL injection and Cross-Site Scripting (XSS)
  + Protect against bots, bad user agents, etc…
  + Size constraints
  + Geo match
* Deploy on CloudFront, Application Load Balancer or API Gateway
* Leverage existing marketplace of rules

**AWS Inspector**

* Only for EC2 instances
* Analyze against known vulnerabilities
* Analyze against unintended network accessibility
* **AWS Inspector Agent must be installed on OS in EC2 instances**
* Define template (rules package, duration, attributes, SNS topics)
* No own custom rules possible – only use AWS managed rules
* After the assessment, you get a report with a list of vulnerabilities

**What does the AWS inspector evaluate?**

* **Remember: only for EC2 instances**
* For Network assessments
  + Network Reachability
* For Host assessments
  + Common Vulnerabilities and exposures
  + Center for Internet Security (CIS) benchmarks
  + Security Best Practices
  + Runtime Behavior Analysis

**Logging in AWS for security and compliance**

* To help compliance requirements, AWS provides many service-specific security and audit logs
* Service logs include:
  + CloudTrail trails - trace all API calls
  + Config Rules - for config & compliance over time
  + CloudWatch Logs - for full data retention
  + VPC Flow Logs: IIp traffic within your VPC
  + ELB Access Logs - metadata of requests made to your load balancers
  + CloudFront Logs - web distribution access logs
  + WAF Logs - full logging of all requests analyzed by the service
* **Logs can be analyzed using AWS Athena if they’re stored in S3**
* **You should encrypt logs in S3, control access using IAM & Bucket Policies, MFA**
* **MOve Logs to Glacier for cost savings**
* Whitepaper
  + <https://d0.awsstatic.com/whitepapers/compliance/AWS_Security_at_Scale_Logging_in_AWS_Whitepaper.pdf>

**GuardDuty**

* Intelligent Threat discovery to protect AWS Account
* Uses Machine Learning, anomaly detection, 3rd party data
* One click to enable (30 day free trial), no need to install software
* Input data includes:
  + CloudTrail Logs: unusual API calls, unauthorized deployments
  + VPC Flow Logs: unusual internal traffic, unusual IP address
  + DNS Logs: compromised EC2 instances sending encoded data within DNS queries
* Notifies you in case of findings
* Integration with AWSLambda

**Trusted Advisor**

* No need to install anything – high level AWS account assessment
* Analyze your AWS accounts and provide recommendation:
  + Cost Optimization
  + Performance
  + Security
  + Fault Tolerance
  + **Service Limits**
* Core Checks and recommendations – all customers
* **Can enable weekly email notification from the console**
* Full Trusted Advisor – Available for Business & Enterprise support plans
  + Ability to set CloudWatch alarms when reaching limits

Why encryption?

**Encryption in flight (SSL)**

* Data is encrypted before sending and decrypted after receiving
* SSL certificates help with encryption (HTTPS)
* Encryption in flight ensures no man in the middle attack can happen

**Server side encryption at rest**

* Data is encrypted after being received by the server
* Data is decrypted before being sent
* It is stored in an encrypted form thanks to a key ( usually a data key)
* The encryption/ decryption keys must be managed somewhere and the server must have access to it

**Client Side Encryption**

* Data is encrypted by the client and never decrypted by the server
* Data will be decrypted by a receiving client
* The server should not be able to decrypt the data
* Could leverage Envelope Encryption

**AWS KMS (Key Management Service)**

* Anytime you hear “encryption” for an AWS service, it’s most likely KMS
* Easy way to control access to your data, AWS manages keys for us
* Fully integrated with IAM for authorization
* Seamlessly integrated into:
  + Amazon EBS: encrypt volumes
  + AMazon S3: Server side encryption of objects
  + Amazon Redshifts : encryption of data
  + Amazon RDS: encryption of data
  + Amazon SSM: Parameter store
  + Etc..

**AWS KMS 101**

* Anytime you need to share sensitive information… use KMS
  + Database Passwords
  + Credentials to external service
  + PRivate Key of SSL certificates
* The value in KMS is that the CMK used to encrypt data can never be retrieved by the user, and the CMK can be rotated for extra security
* **Never ever store your secrets in plaintext especially in your code**
* Encrypted secrets can be stored in the code / environment variables
* **KMS can only help in encryption up to 4KB of data per call**
* If > 4KB, use envelope encryption
* To give access to KMS to someone:
  + Make sure the Key Policy allows the user
  + Make sure the IAM policy allows the API calls

**AWS KMS (Key Management Service)**

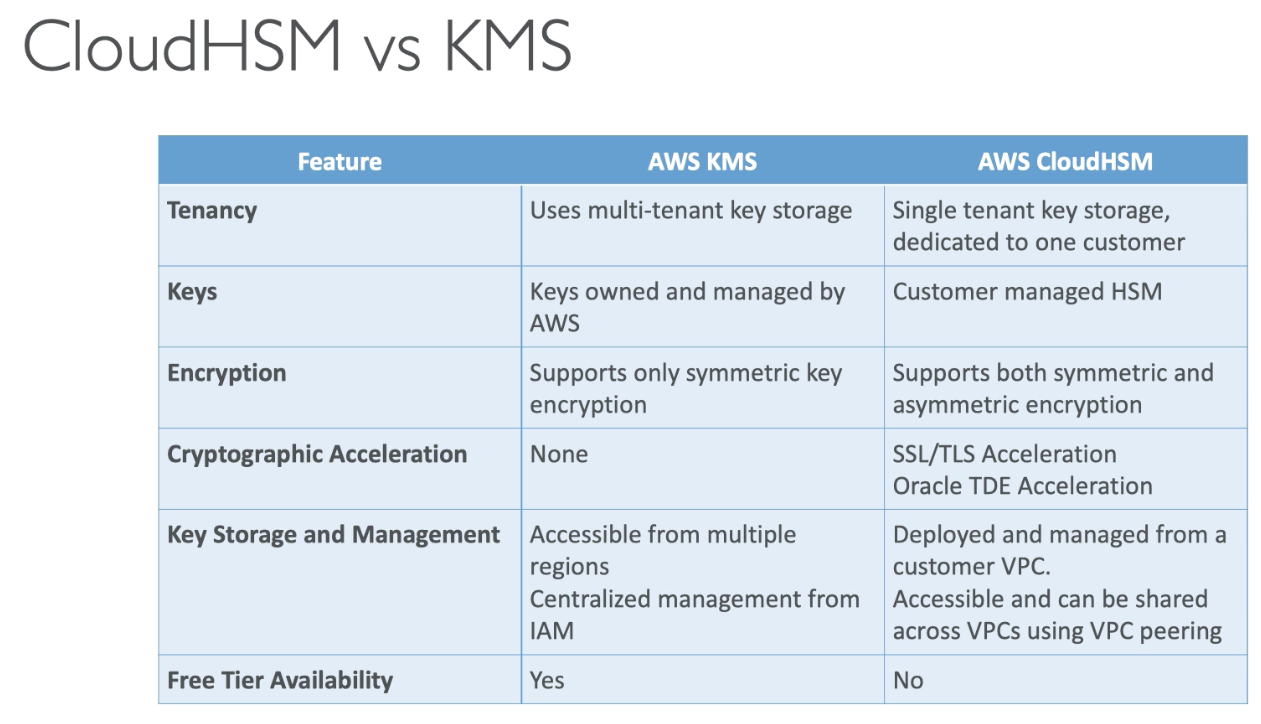
* Able to fully manage the keys & policies:
  + Create
  + Rotation policies
  + Disable
  + Enable
* Able to audit key usage ( using CloudTrail)
* Three types of Customer Master Keys (CMK):
  + AWS Managed Service Default CMK: free
  + User Keys created in KMS: $1 / month
  + USer Keys imported ( must be 256-bit symmetric key): $1 / month
* + pay for API call to KMS ($0.03 / 10000 calls)

Encryption in AWS Services

* Requires migration ( through Snapshot / Backup):
  + EBS Volumes
  + RDS databases
  + ElastiCache
  + EFS network file system
* In-place encryption:
  + S3

Cloud HSM

* KMS => AWS manages the software for encryption
* CloudHSM => AWS provisions encryption hardware
* Dedicated Hardware (HSM = Hardware Security Module)
* You manage your own encryption keys entirely ( not AWS)
* The CloudHSM hardware device is tamper resistant
* FIPS 140-2 Level 3 compliance
* CloudHSM clusters are spread across multi AZ (HA)
* Supports both symmetric and asymmetric encryption ( SSL/ TLS keys)
  + KMS only does symmetric encryption
* No free tier available
* Must use the CloudHSM client software



**IAM + MFA (Multi Factor Authentication)**

* MFA adds an added level of security while accessing your AWS account
* AWS MFA accepts both virtual and hardware MFA devices.
* Virtual MFA device: Google Authenitactor / Authy
* MFA for root user can be configured from IAM dashboards
* MFA can also be configured from CLI
* You can setup MFA for Individual Users
* Credentials Report:
  + A CSV report file on all the IAM users and credentials
  + This shows who all have enabled MFA

**IAM PassRole Option**

* Example: When you create an EC2 instance,m you assign a role to it
* In order to assign a role to an EC2 instance, you need **IAM: PassRole**
* IAM:PassRole can be used for any service where we assign roles ( not only EC2)

**AWS STS**

* **Allows to grant limited and temporary access to AWS resources**
* Token is valid for up to one hour ( must be refreshed)
* **CrossAccount Access**
  + Allows users from one AWS account access resources in another
* **Federation (Active Directory)**
  + Provides a non-AWS user with temporary AWS access by linking users Active Directory credentials
  + Uses SAML (security assertion markup language)
  + Allows Single Sign On (SSO) which enables users to log in to AWS console without assigning IAM credentials
* **Federation with third party providers / Cognito**
  + Used mainly in web and mobile applications
  + Makes use of Facebook/Google/Amazon etc to federate them

**Cross Account Access**

* Define an IAM Role for another account to access
* Define which accounts can access this IAM Role
* Use AWS STS (Security Token Service) to retrieve credentials and impersonate the IAM Role you have access to (**AssumeRole API**)
* Temporary credentials can be valid between 15 minutes to 1 hour

**Why Identity Federation?**

* Federation lets users outside of AWS to assume temporary role for accessingAWS resources
* These users assume identity provided access role
* **Federation assumes a form of 3rd party authentication**
  + LDAP
  + Microsoft Active Directory (~= SAML)
  + Single Sign ON
  + Open ID
  + Cognito
* **Using federation, you don’t need to create IAM users (user management is outside of AWS)**

**SAML Federation**

**For Enterprises**

* To integrate ACtive Directory /ADFS withAWS ( or any SAML 2.0)
* Provides access to AWS Console or CLI ( through temporary creds)
* No need to create an IAM user for each of your employees

**Custom Identity Broker Application For Enterprises**

* Use only if identity provide is not compatible with SAML 2.0
* **The identity broker must determine the appropriate IAM policy**
* The user has to create a identity broker
* <https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_common-scenarios_federated-users.html>

**AWS Cognito -   
Federated Identity Pools For Public Applications**

* Goal:
  + Provide direct access to AWS Resources from the client side
* How:
  + Login to federated identity provider - or remain anonymous
  + Get temporary AWS credentials back from the Federated Identity Pool
  + These credentials come with a pre-defined IAM policy stating their permissions
* Example:
  + Provide (temporary) access to write to S3 bucket using Facebook Login
* Note
  + Web Identity Federation is an alternative to using Cognito but AWS recommends against it

**JSON Policy Lab**

* **IAM:**
  + [**https://docs.aws.amazon.com/IAM/latest/UserGuide/access\_policies\_examples.html**](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_examples.html)
* **S3**
  + [**https://docs.aws.amazon.com/AmazonS3/latest/dev/example-bucket-policies.html**](https://docs.aws.amazon.com/AmazonS3/latest/dev/example-bucket-policies.html)
  + [**https://docs.aws.amazon.com/AmazonS3/latest/dev/example-policies-s3.html**](https://docs.aws.amazon.com/AmazonS3/latest/dev/example-policies-s3.html)

**AWS Artifact ( not really a service)**

* **Portal that provide customers with on-demand access to AWS compliance documentation and AWS agreements**
* Artifact Reports
  + Allows you to download AWS security and compliance documents, like AWS ISO certifications, Payment Card Industry (PCI), and System and ORganization Control (SOC) reports
* Artifact Agreements
  + - Allows you to review, accept, and track the status of AWS agreements such as the bBusiness ASSociate Addendum (BAA)
* Can be used to support internal audit or compliance

**Section Summary: Security & Compliance**

* **AWS Shield**
  + Automatic DDoS Protection + 24/7 support for advanced
* **AWS WAF**
  + Firewall to filter incoming requests based on rules
* **AWS Inspector**
  + For EC2 only, install agent and find vulnerabilities
* **AWS GuardDuty**
  + Find malicious behavior with VPC, DNS & CloudTrail Logs
* **AWS Trusted Advisor**
  + Analyze AWS account and get recommendations
* **AWS KMS**
  + Encryption keys managed by AWS
* **AWS CloudHSM**
  + Hardware encryption, we manage keys, supports asymmetrical
* **AWS STS**
  + Generate SEcurity Token
* **Identity Federation:** 
  + SAML 2.0 or Custom for Enterprise, Cognito for Apps
* **AWS Artifact**
  + Get access to compliance report such as PCI, ISO,etc…
* **AWS Config**
  + Track config changes and compliance against rules
* **AWS CloudTrail**
  + Track API calls made by users within account

**Route 53 Section**

* TTL
* CNAME vs Alias
* Health Checks
* Routing Policies
  + Simple
  + Weighted
  + Latency
  + Failover
  + Geolocation
  + Multi Value
* 3rd party domains integration

**AWS Route 53 Overview**

* Route53 is a Managed DNS ( Domain Name System)
* DNS is a collection of rules and records which helps clients understand how to reach a server through its domain name.
* IN AWS, the most common records are:
  + A: hostname to IPv4
  + AAAA: hostname to IPv6
  + CNAME: hostname to hostname
  + Alias: hostname to AWS resource
* Route53 can use:
  + Public domain names you own (or buy)
  + Application1.mypublicdomain.com
  + Private domain names that can be resolved by your instance in your VPCs
  + Application1.company.internal
* Route53 has advanced features such as:
  + Load balancing (through DNS – also called client load balancing)
  + Health checks (although limited …)
  + Routing Policy: simple, failover, geolocation, latency, weighted, multi value
* You pay .5$ per month per hosted zone

**DNS Records TTL (Time to Live)**

* The time for the IP address to reside in the client’s cache
* High TTL:
  + Less traffic on DNS
  + Possiblky outdated records
* Low TTL:
  + More traffic on DNS
  + Records are outdated for less time
  + Easy to change records
* TTL is mandatory for each DNS record

**CNAME vs Alias**

* AWS Resources ( LOad Balancer, CloudFront…) expose an AWS hostname and you want it to point to a route domain
* CNAME
  + Points a hostname to any other hostname
  + ONLY for non root domain
  + Only works with sub-domains
* Alias
  + Points a hostname to an AWS Resource
  + Works for ROOT DOMAIN and NON ROOT DOMAIN
  + Free of charge
  + Native health checks

**Simple Routing Policy**

* Use when you need to redirect to a single resource
* You can’t attach health checks to simple routing policy
* If multiple values are returned, a random one is chosen by the client
* Client side load balancing

**Weighted Routing Policy**

* Control the % of the requests that go to specific endpoints
* Helpful to test 1% of traffic on a new app version
* Helpful to split traffic between two regions
* Can be associated with Health Checks

**Latency Routing Policy**

* Redirect to the server that has the least latency close to us
* Super helpful when latency of users is a priority
* **Latency is evaluated in terms of user to designated AWS Region**
* EG. Germany may be direct to US if that is the lowest latency

**Route 53 Health Checks:**

* Have X health checks failed -> unhealthy ( default 3 tests)
* After X health checks passed => healthy ( default 3 tests)
* Default Health Check Interval: 30s ( can set to 10s – higher costs)
* **About 15 health checkers will check the endpoint health**
* -> one request every 2 seconds on average
* Can have HTTP, TCP and HTTPS health checks ( no SSL verification)
* Possibility of integrating the health checks with CloudWatch
* **Health checks can be linked to Route53 DNS queries!**

**Failover Routing Policy**

* Primary instance
* Secondary instance
* Health Check will be pointed at the primary
* If the health check fails it will change the routing to the secondary

**Geolocation Routing Policy**

* Different from latency based
* This is routing based on user location
* Here we specify: traffic form the UK should go to a specific IP  
  Should create a “default” policy ( incase there’s no match on location)

**Multi Value Routing Policy (basically SRP w/ health checks)**

* Use when routing traffic to multiple resources
* Want to associate a route 53 health checks with records
* Up to 8 healthy records are returned for each multi value query
* **Multi value is not a substitute for having an ELB**

**Route53 as a Registrar**

* A domain name registrar is an organization that manages the reservation of internet domain names
* Famous names
  + goDaddy
  + Google Domains
  + Etc…
* And also Route 53
* Domain Registrar does not necessarily mean a domain name service

**3rd Party Registrar with AWS Route 53**

* If you buy your domain on 3rd party website, you can still use Route53
* 1) create a hosted zone in route 53
* 2) Update NS Records on 3rd party website to use ROute53 name servers
* Domain Registrar!= DNS (but each domain registrar usually comes with some DNS features)

**Understanding CIDR - IPv4 (Classless Inter-Domain Routing)**

* CIDR are used for Security Groups rules, or AWS networking in general
* They help to define an IP address range
  + We’ve seen WW.XX.YY.ZZ/32 === one IP
  + We’ve seen 0.0.0.0/0 === all IPs
  + But we can define for ex: 192.168.0.0/26: 192.168.0.0 – 192.168.0.63 (64 IP)
* A CIDR has two components:
  + The base IP (xx.xx.xx.xx)
  + The Subnet Mask (/26)
  + The base IP represents an IP contained in the range
  + The subnet masks defines how many bits cn change in the IP
  + The subnet mask can take two forms. Example:
    - 255.255.255.0 -- less common
    - /24 -- more common
* The subnet masks basically allows part of the underlying IP to get additional next values from the base IP
  + /32 allows for 1 IP = 2^0
  + /31 allows for 2 IP = 2 ^ 1
  + /30 allows for 4 IP = 2 ^ 2
  + /29 allows for 8 IP = 2 ^ 3
  + /28 allows for 16 IP = 2 ^ 4
  + /27 allows for 32 IP = 2 ^ 5
  + /26 allows for 64 IP = 2 ^ 6
  + /25 allows for 128 IP = 2 ^ 7
  + /24 allows for 256 IP = 2 ^ 8
  + /16 allows for 65,536 IP = 2 ^ 16
* QUICK MEMO
  + /32 no Ip number can change
  + /24 last Ip number can change
  + /16 last IP two numbers can change
  + /8 last IP three numbers can change
  + /0 all IP numbers can change

**Understanding CIDRs Little Exercise:**

* 192.168.0.0/24
  + 192.168.0.0 - 192.168.0.255 (256 IP)
* 192.168.0.0/16
  + 192.168.0.0 - 192.168.255.255
* 134.56.78.123/32
  + 134.56.78.123
* 0.0.0.0/0
  + All IPs

**Private vs Public IP (IPv4)**

**Allowed ranges**

* Private IP can only allow certain values
  + 10.0.0.0/8 <= in big networks
  + 172.16.0.0/12 <= default AWS one
  + 19 2.168.0.0/16 <= example: Home networks
* All the rest of the IP on the internet are public

**Default VPC Walkthrough**

* All new accounts have a default VPC
* New instances are launched into default VPC if no subnet is specified
* Default VPC have internet connectivity and all instances have public IP
* We also get a public and private DNS name

**VPC in AWS – IPv4**

* VPC = Virtual Private Cloud
* You can have multiple VPCs in a region ( max 5 per region -> soft limit)
* Max CIDR per VPC is 5. For each CIDR
  + Min size is /28 = 16 IP addresses
  + Max size is /16 = 65536 IP addresses
* Because VPC is private, only the Private IP ranges are allowed:
  + 10.0.0.0/8 <= in big networks
  + 172.16.0.0/12 <= default AWS one
  + 19 2.168.0.0/16 <= example: Home networks
* Your VPC CIDR should not overlap with your other networks (ex: corporate)

**Subnets - IPv4**

* AWS reserves 5IPs address ( first 4 and last 1 IP address ) in each Subnet
* These 5 IPs are not available for use and cannot be assigned to an instance
* Ex, if CIDR block 10.0.0.0/24, reserved IP are:
  + 10.0.0.0: Network address
  + 10.0.0.1: Reserved by AWS for the VPC router
  + 10.0.0.2: Reserved by AWS for mapping to Amazon-provided DNS
  + 10.0.0.3: Reserved by AWS for future use
  + 10.0.0.255: Network broadcast address. AWS does not support broadcast in a VPC therefore the address is reserved
* Exam Tip:
  + If you need 29 IP addresses for EC2 instances you can’t choose a subnet of size /27 (32IP)
  + You need at least 64 IP, subnet size needs to /25

**Internet Gateways**

* Internet gateways helps our VPC instances connect with the internet
* It scales horizontally and is HA and redundant
* Must be created separately from VPC
* One VPC can only be attached to one IGW and vice versa
* Internet Gateway is also a NAT for the instances that have a public IPv4
* Internet Gateways on their own do not allow internet access
* Route tables must also be edited

**NAT Instances – Network Address Translation (outdated but still at the exam)**

* Allows instances in the private subnets to connect to the internet
* Must be launched in a public subnet
* Must disable EC2 flag: Source / Destination Check
* Must have Elastic IP attached to it
* Route table must be configured to route traffic from private subnets to NAT instance

NAT Instances – comments

* Amazon Linux AMI pre-configured are available
* Not highly available /resilient setup out of the box
* => Would need to create ASG in multi AZ + resilient user-data script
* Internet traffic bandwidth depends on EC2 instance performance
* Must manage security groups & rules
  + Inbound:
    - Allow HTTP / HTTPS Traffic coming form PRivate Subnets
    - Allow SSH from your home network (access is provide through internet gateway)
  + Outbound:
    - Allow HTTP /HTTPS traffic to the internet

NAT Gateway

* AWS managed NAT, higher bandwidth, better availability, no admin
* Pay by the hour for usage and bandwidth
* NAT is created in a specific AZ uses an EIP
* Cannot be used by an instance in that subnet ( only from other subnets)
* Requires an IGW ( private SUbnet -> NAT -> IGW)
* 5 gbps of bandwidth with automatic scaling up to 45 gbps
* No security group to manage / required

NAT Gateway with High Availability

* NAT gateway is resilient within a single AZ
* Must create multiple NAT Gateway in multiple AZ for fault-tolerance
* There is no cross AZ failover needed because if an AZ goes down it doesn’t need a NAT

NAT Instance vs Gateway

* <https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-comparison.html>

DNS Resolution in VPC

* enableDnsSupport: ( = DNS Resolution setting )
  + Default True
  + Helps decide if DNS resolution is supported for the VPC
  + If true, queries the AWS DNS server at 169.254.169.253
* enabledDnsHostname: ( = DNS Hostname setting)
  + False by default for newly created VPC, True by default for Default VPC
  + Won’t do anything unless enabledDnsSupport=true
  + If tTrue, Assign public hostname to EC2 instance if it has a public
* If you use custom DNS domain names in a private zone in Route 53, you must set both these attributes to true

Network ACL (Access Control Lists)s & Security Group Incoming Requests

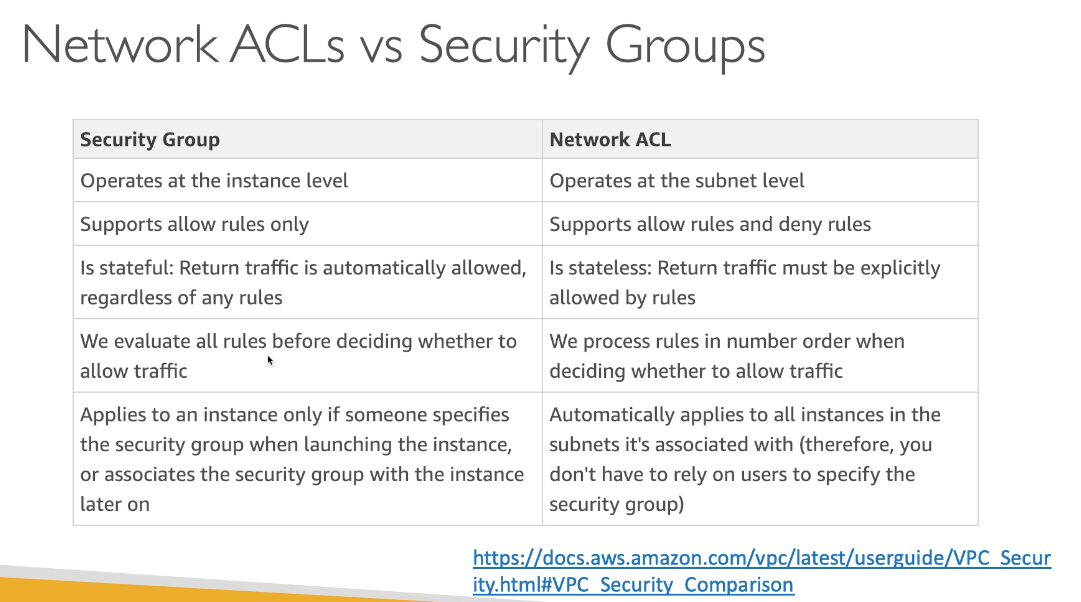
* ORDER
* NACL Inbound Rules
* Security Group Inbound Rules
* Security Group Outbound Rules (always allowed, because they are stateful) - if a inbound passes an outbound will always pass)
* NACL Outbound Rules (stateless) => Means that it will be validated

Network ACLs & Security Group Outgoing Requests

* ORDER
* Security Group Outbound Rules
* NACL Outbound Rules
* NACL Inbound Rules (stateless)
* Inbound Allowed (stateful)

Network ACLs

* NACL are like firewall which control traffic from and to subnet
* Default NACL allows everything outbound and everything inbound
* One NACL per subnet, new subnets are assigned the default NACL
* Define NACL rules:
  + Rules have a number ( 1 - 32766) and higher precedence with a lower number
  + Last rule is an asterisk and denies a request in case of no rule match
  + AWS recommends adding rules by increments of 100
* Newly created NACL will deny everything
* NACL are a great way of blocking a specific IP at the subnet level



Ephemeral ports

* They vary based on the OS in use
* Best practice allow 1024 - 65535, then disable any malicious ports using an earlier rule set in the NACL

VPC Peering

* Connect two VPC , privately using AWS network
* Make them behave as if they were in the same network
* Must not have overlapping CIDR
* VPC peering connection is not transitive ( which means each vpc peer connection needs to be established for each vpc that needs to communicate with one another)
* You can do VPC peering with another AWS account
* You must update route tables in each VPC subnets to ensure instances can communicate

VPC Peering – Good to know

* VPC peering can work inter region, cross account
* You can reference a security group of a peered VPC ( works cross accounts)

VPC Endpoints

* Endpoints allow you to connect to AWS service using a private network instead of the public www network
* They scale horizontally and are redundant they remove the need of IGW NAT tetc to access AWS Services
* INterface: provisions an ENI ( private IP address) as an entry point ( must attach security group)  – most aws services
* Gateway: provisions a target and must be used in a route table – S3 and DynamoDB
* In case of issues
  + Check DNS setting Resolution in your VPC
  + Check Route Tables

VPC Flow Logs + Athena

* Capture information about IP traffic going into your interfaces:
  + VPC Flow Logs
  + Subnet Logs
  + Elastic Network Interface Flow Logs
* Helps to monitor and troubleshoot connectivity issues
* Flow Logs data can go to S3 / CloudWatch Logs
* Captures network information from AWS managed interfaces too:
  + ELB
  + RDS
  + ElastiCache
  + Redshift
  + WorkSpaces

Flow Log Syntax

* <version> <account-id> <interface-id> <srcaddr> <dstaddr> <srcport> <dstport> <protocol> <packets> <bytes> <start> <end> <action> <log-status>
* Srcaddr, dstaddr help identify problematic IP addresses
* Srcport, dstport help identify problematic Ports
* Action: success or failure of the request due to Security Group / NACL
* Can be used for analytics on usage patterns or malicious behavior
* Flow logs example: <https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html#flow-log-records>
* **Query VPC flow logs using Athena on S3 or CloudWatch Log Insights**

**Flow Logs: looking at “action” field how to troubleshoot SG vs NACL issue**

* For incoming requests
  + Inbound REJECT: NACL or SG
  + Inbound ACCEPT, outbound REJECT: NACL
* For outgoing requests
  + Outbound REJECT: NACL or SG
  + Outbound ACCEPT, inbound REJECT: NACL

Bastion Hosts

* We can use a Bastion Host to SSH into our private instances
* The bastion is in the public subnet which is then connected to all other private subnets
* Bastion Host Security group must be the tightest
* Exam Tip: Make sure the bastion host only has port 22 traffic from the IP you need, not from the security groups of your other instances
* Basically the public instance

Site to Site VPN

* Customer Gateway on the corporate DC
* VPN Gateway in the AWS VPC
* Then the site to site VPN connection to connect the VPN gateway and the customer gateway

Virtual Private Gateway:

* VPN concentrator on the AWS side of the VPN
* VGW is created and attached to the VPC from which you want to create the Site-to-Site VPN connection
* Possibility to customise the ASN

Customer Gateway:

* Software application or physical device on customer side of the VPN connection
* <https://docs.aws.amazon.com/vpc/lsatest/adminguide/Introduction.html#DevicesTested>
* IP address
  + Use static internet routable IP address for your customer gateway device
  + If behind a CGW behind NAT ( with NAT-T), use the public IP address of the NAT

Direct Connect

* Provides a dedicated private connection from a remote network to your VPC
* Dedicated connectin must be setup between your DC and AWS Direct Connect locations
* YOu need to setup a virtual private gateway on your vpc
* Access public resources ( s3) and private (ec2) on same connection
* Use Cases
  + Increase Bandwidth throughput -working with large data sets - lower cost
  + More consistent network experience - applications using real time data feeds
  + Hybrid environments ( on prem + on cloud)
* Supports IPv4 and IPv6

Direct Connect Gateway

* If you want to setup a Direct Connect to one or more VPC in many different regions ( same account ) you must use a Direct Connect Gateway
* Does not replace VPC peering connections

Direct Connect – Connection Types

* Dedicated Connections: 1gbps and 10gbps capacity
  + Physical ethernet port dedicated to a customer
  + Request made to AWS first, then completed by AWS Direct Connect Partners
* Hosted Connections: 50Mbps, 500mbps, to 10gbps
  + Connection requests are made via AWS Direct Connect Partners
  + Capacity can be added or removed on demand
  + 1, 2, 5, 10 gbps available at select AWS Direct Connect Partners
* Lead Times are often longer than 1 month to establish a new connection

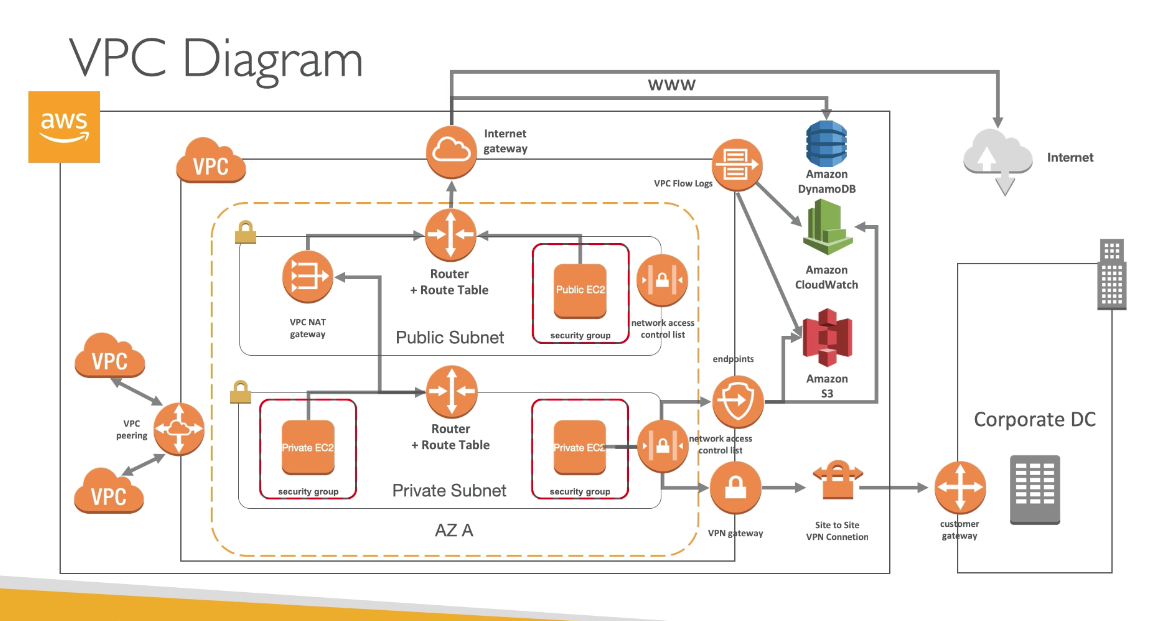
Direct Connect – Encryption

* Data in transit is not encrypted but is private
* AWS Direct Connect + VPN
* Provides an IPsec-encrypted private connection
* Good for an extra level of security but slightly complex to put into place

Egress Only Internet Gateway

* Egress only internet gateway is only for IPv6 only
* Similar functions as a NAT, but a NAT is for IPv4
* Good to know: IPv6 are all public addresses
* Therefore all our instances with IPv6 are publicly accessible
* Egress Only Internet Gateway gives our IPv6 instances access to the internet, but they won’t be directly reachable by the internet
* After creating an Egress Only Internet Gateway, edit the route tables

VPC Section Summary



* CIDR: IP Range
* VPC Virtual Private Cloud => we define a list of IPv4 & IPv6 CIDR
* Subnets: Tied to an AZ, we define a CIDR
* Internet Gateway: at the VPC level, provide IPv4 & IPv6 Internet Access
* Route Tables: Must be edited to add routes from subnets to the IGW, VPC Peering Connections, VPC Endpoints, etc…
* NAT Instances: Give internet access to instances in private subnets. Old, must be setup in a public subnet, disable source / destination check flag
* NAT Gateway: managed by AWS, provides scalable internet access to private instances, IPv4 only
* Private DNS + Route 53: enable DNS Resolution + DNS hostnames (VPC)
* NACL: Stateless subnet rules for inbound and outbound traffic, and don’t forget the ephemeral ports
* Security Groups: stateful and operate at the EC2 instance level
* VPC Peering: Connect two VPC with non overlapping CIDR, non transitive
* VPC Endpoints; Provide private access to AWS Services (S3, Dyanmodb, CloudFormation, SSM) within VPC
* VPC Flow Logs: can be setup at the VPC / Subnet/ ENI Level, for ACCEPT and REJECT traffic, helps identifying attacks, analyze using Athena or CloudWatch Log Insights
* Bastion Host: Public instance to SSH into, that has SSH connectivity to instances in the private subnets
* Site to Site VPN: Setup a customer gateway on DC, a virtual private gateway on VPC, and site to site vpn over public internet
* Direct Connect: setup a virtual private gateway on VPC, and establish a direct private connection to an AWS direct connection location
* Direct Connect Gateway: setup a direct connect to many VPC in different regions
* Internet Gateway Egress: like a NAT Gateway, but for IPv6