Contents

[Contents 1](#_Toc64045426)

[**IAM** 5](#_Toc64045427)

[Key concepts 5](#_Toc64045428)

[Using Roles to access AWS more securely 5](#_Toc64045429)

[AWS Directory Service 5](#_Toc64045430)

[IAM Policies 6](#_Toc64045431)

[Policies 6](#_Toc64045432)

[AWS Resource Access Manager (RAM) 7](#_Toc64045433)

[AWS Single Sign-On (SSO) 7](#_Toc64045434)

[**S3 – Simple Storage Service** 8](#_Toc64045435)

[Features: 8](#_Toc64045436)

[Storage classes: 8](#_Toc64045437)

[Charges: 8](#_Toc64045438)

[S3 Security and Encryption 9](#_Toc64045439)

[Versioning – typical versioning 9](#_Toc64045440)

[Locking policies 9](#_Toc64045441)

[S3 Performance 10](#_Toc64045442)

[S3 Select 10](#_Toc64045443)

[AWS Organisations 10](#_Toc64045444)

[Sharing S3 buckets between accounts 10](#_Toc64045445)

[S3 Cross region replication 10](#_Toc64045446)

[Transfer Acceleration 10](#_Toc64045447)

[Data sync 11](#_Toc64045448)

[CloudFront 11](#_Toc64045449)

[Signed URLs and Cookies 11](#_Toc64045450)

[Snowball 12](#_Toc64045451)

[Athena vs Macie 12](#_Toc64045452)

[Storage Gateway 12](#_Toc64045453)

[**EC2 – Elastic Cloud Compute** 13](#_Toc64045454)

[Pricing 13](#_Toc64045455)

[Instance types 13](#_Toc64045456)

[Security Groups 14](#_Toc64045457)

[EBS 14](#_Toc64045458)

[Moving EBS volumes / EC2 instances to another AV 14](#_Toc64045459)

[Moving EBS volumes / EC2 instances to another region 14](#_Toc64045460)

[Snapshots 15](#_Toc64045461)

[AMI types 15](#_Toc64045462)

[ENI vs ENA vs EFA 15](#_Toc64045463)

[Encrypted root volumes and snapshots 16](#_Toc64045464)

[Spot Instances & Spot Fleets 16](#_Toc64045465)

[EC2 Hibernate 16](#_Toc64045466)

[CloudWatch & CloudTrail 17](#_Toc64045467)

[AWS Command Line (CLI) 17](#_Toc64045468)

[Bootstrap Scripts 17](#_Toc64045469)

[Instance metadata 17](#_Toc64045470)

[AWS EFS (Elastic File System) 18](#_Toc64045471)

[Windows FSx for Windows 18](#_Toc64045472)

[Amazon FSx for Lustre 18](#_Toc64045473)

[EC2 Placement Groups 18](#_Toc64045474)

[Clustered 18](#_Toc64045475)

[Spread 18](#_Toc64045476)

[Partitioned 18](#_Toc64045477)

[HPC (High-performance computing) 19](#_Toc64045478)

[AWS WAF (Web Application Firewall) 19](#_Toc64045479)

[Quiz 19](#_Toc64045480)

[**AWS Databases** 20](#_Toc64045481)

[DynamoDB 21](#_Toc64045482)

[Redshift 23](#_Toc64045483)

[Aurora 24](#_Toc64045484)

[Elasticache 25](#_Toc64045485)

[Caching strategies on AWS 25](#_Toc64045486)

[EMR (Elastic Map Reduce) Overview 25](#_Toc64045487)

[Quiz 25](#_Toc64045488)

[**Route 53** 26](#_Toc64045489)

[DNS 101 26](#_Toc64045490)

[Routing Policies 27](#_Toc64045491)

[**VPC** 28](#_Toc64045492)

[Internet Gateway 29](#_Toc64045493)

[Route Tables 29](#_Toc64045494)

[NAT Instances & Gateways 30](#_Toc64045495)

[NACLs vs Security Groups 31](#_Toc64045496)

[VPC Flow Logs 32](#_Toc64045497)

[Bastions (jumpboxes) 32](#_Toc64045498)

[Direct Connect 32](#_Toc64045499)

[Global Accelerator 33](#_Toc64045500)

[VPC Endpoints 34](#_Toc64045501)

[AWS Private Link 34](#_Toc64045502)

[AWS Transit Gateway 34](#_Toc64045503)

[VPN CloudHub 35](#_Toc64045504)

[AWS Network Costs 35](#_Toc64045505)

[**High Availability** 36](#_Toc64045506)

[Elastic Load Balancers 36](#_Toc64045507)

[Sticky Sessions 36](#_Toc64045508)

[Cross-Zone Load Balancing 36](#_Toc64045509)

[Path Patterns 37](#_Toc64045510)

[Auto Scaling 37](#_Toc64045511)

[Bastion HA 37](#_Toc64045512)

[On-Prem Services with AWS 38](#_Toc64045513)

[Elastic Beanstalk 38](#_Toc64045514)

[Setting up an entire architecture example (Wordpress site) 38](#_Toc64045515)

[**Applications** 39](#_Toc64045516)

[SQS (Simple Queue Service) 39](#_Toc64045517)

[SWF (Simple Work Flow Service) 39](#_Toc64045518)

[SNS (Simple Notification Service) 40](#_Toc64045519)

[Elastic Transcoder 40](#_Toc64045520)

[API Gateway 40](#_Toc64045521)

[Kinesis 40](#_Toc64045522)

[Web Identity Federation & Cognito 40](#_Toc64045523)

[Event-Driven Architecture 41](#_Toc64045524)

[**Security** 41](#_Toc64045525)

[KMS (Key Management Service) 41](#_Toc64045526)

[CloudHSM (Hardware Security Modules) 42](#_Toc64045527)

[Parameter Store 42](#_Toc64045528)

[Secrets Manager 43](#_Toc64045529)

[AWS Shield 43](#_Toc64045530)

[Web Application Firewall (WAF) 43](#_Toc64045531)

[**Serverless** 43](#_Toc64045532)

[Lambda 43](#_Toc64045533)

[SAM (Serverless Application Model) 43](#_Toc64045534)

[ECS (Elastic Container Service) 43](#_Toc64045535)

# **IAM**

* New users appear without any rights

## Key concepts

* Users
  + Basically the different accounts
  + Can have two sets of login details – aws secret & key, and username and password
    - Username + password is for console access
    - Secret & key is for programmatic access
* Groups
  + Groups of users. Used to categorise bundles of users (say, developers, accounting, etc…)
* Policies
  + Basically rules. Rights to do this and that.
  + In AWS, anything that is not explicitly allowed is implicitly forbidden.
  + Explicit denies override anything else, regardless of which comes later or is more specific.
* Roles
  + Compilations of policies (think groups for policies instead of users)
  + Can be attached to stuff such as ec2 instances for example

## Using Roles to access AWS more securely

Add a role – go to the EC2 insance -> Actions -> Instance Settings -> Attach/Replace IAM role -> Select the role

Allows us to interact with AWS resources (as per the policy of the role) from the AWS CLI without the need to pass along credentials. Can be assigned via the console or CLI and applies globally.

Why? – Easier to manage than credentials on each EC2 instance, more secure (as they can be fine-grained better than accounts, and also if an EC2 gets hacked, it remains the only point of entry).

## AWS Directory Service

A collection of AWS services related to (integrated with?) Microsoft AD.

Allows you to connect AWS resources to existing on-prem Microsoft AD (for example, lets you access AWS resources/log in the AWS Management Console via existing corporate credentials on the corporate AD). Allows for Single Sign-On (SSO) to domain-joined EC2 instances (wut?).

## IAM Policies

Policies govern the usage of specific resources. We identify resources from the PoV of the IAM account, so they are all addressed by unique ARNs that follow this structure:

* arn:partition:service:region:account-id:
  + The resource is described from global to particular level, narrowing down
  + Partition can be aws (almost always) or smth like aws-cn if dealing with the Chinese AWS infrastructure
  + Service is the service the resource belongs to (s3, ec2, rds, iam…)
  + Region is the region the resource is placed in
  + Account ID is out unique 12 digit acc id
  + And then the resource name ends with:
    - resource
    - resource\_type/resource
    - resource\_type/resource/qualifier
  + Basically, we zoom in more and more during the description. Infrastructural partition -> Sevice -> Region -> Account -> Particular Instance
    - Example – arn:aws:ec2:us-east-1:012345678977:instance/instanceID

Policies are basically JSON files that define permissions. They can be related to an identity (Josh can access this and that and do this and that there) or to resources (this bucket can be accessed by Sharon and Joe, and they can only read its contents).

Policies have no effect until they are attached to a User, Group, or Role. They specify the version, followed by list of statements (who can do what).

* Each statement matches an AWS API request. Format is as follows:
  + Sid (Statement ID, human readable, used for us to know what’s up)
  + Effect – either Allow or Deny
  + Action – list of API calls that are allowed or denied, in the form of “service name:action”
  + Resource – the ARN the abovementioned actions are against (either identity or resource)

**Inline Policies** are policies that are limited to a specific role (see them as variables within a function).

**Permission Boundaries** – Used to prevent privilege escalation. Basically, limit the rights a User or Role can have. While ordinary policies would give rights to IAM entities, permission boundary policies would cap those rights – they do not create rights themselves.

However, if a right is derived from a resource-based policy, it is not capped by the permission boundary (say, if an S3 bucket is given a role that allows its contents to be read by everyone, and the user Allan does not have S3 listed in his permission boundaries, he’d still be able to read the bucket’s contents). – Test that tomorrow! <https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_boundaries.html>

## AWS Resource Access Manager (RAM)

Sharing access to resources across accounts. For example, letting Account B create and make use of EC2 instances in a VPC of Account A, or access a database of Account A.

Stuff from only a few services are shareable – App Mesh, Code Build, EC2, Aurora, Route 53, VPC, and a few others.

## AWS Single Sign-On (SSO)

Helps to centrally manage access to AWS accounts and business apps. For example, can allow

Also, can for example sign into AWS with one’s Office 365 account?

SAML 2.0

# **S3 – Simple Storage Service**

Globally accessible (but locally stored), object-based (think files, like photos, movies, documents, etc) storage service. Organised in buckets (basically folders). No min file size, max 5TB. Due to it being globally acessible, an s3 bucket needs a globally unique name (as it gets accessed via its url).

Super high durability – 99.999999999% durability, 99.9% availability

When uploading a file, if the upload was successful you’ll get a http 200 code

S3 objects consist of:

* Key (the name of the file)
* Value (the content of the file, the sequence of bytes)
* Version ID (in case of multiple versions)
* Metadata
* Subresources (ACL’s, torrents)

### Features:

* Tiered storage
* Lifecycle management (say, when this file is 30 days old, move it to Glacier Deep Archive)
* Versioning
* Encryption
* MFA (Multifactor authentication) delete
* Control access to data via ACL’s and bucket policies

### Storage classes:

* Standard (99.99% availability, 11x9 durability, can sustain the loss of two facilities concurrently)
* Infrequent Access (IA) – (99.9% availability) for data that you access infrequently, but when you do you want it to happen fast. Lower fee at rest, but has a retrieval fee
* One Zone IA – Same as above, but lower availability/durability (99.5% availability)
* Intelligent Tiering – will see how you use the data and change its class based on usage
* Glacier – lower cost, higher retrieval time (configurable, from minutes to hours), retrieval fee
* Glacier Deep Archive – very low cost, very high retrieval time (12h), retrieval fee

### Charges:

* Storage (GB/month)
* Requests
* Per tier
* Data transfer
* Transfer acceleration (making use of Cloud Front)
* Cross region replication

Remember for exam – MFA auth for delete, Read after Write for PUTS of new Objects, Eventual Consistency for overwrites or deletes (can take some time to propagate due to the replication)

### S3 Security and Encryption

Can config the bucket security via bucket policies as well as access control lists (ACLs). Bucket policies are for the whole bucket, while the ACLs are more granular and can reach individual file level

Can also be configured to create access logs that tracks all the requests made to that bucket. Those logs can then be sent to another bucket, even one in a different account. They can be server level (free for the logging, normal storage fees for the space taken by the logs themselves) or at the file level (through CloudWatch, fees apply)

Two types of encryption – in transit (SSL/TLS, think HTTPS) and at rest

* At rest server side – i.e. you upload an object and Amazon encrypts it
  + S3 Managed Keys – SSE-S3 (Server-Side Encryption S3) (keys come from Amazon)
  + AWS Key Management Service, Managed Keys – SSE-KMS (shared)
  + Server-Side Encryption with Customer Provided Keys – SSE-C (keys come from you)
* At rest client side – i.e. you encrypt the object and upload it to Amazon

### Versioning – typical versioning

* Once enabled, can only be suspended (paused). Cannot be disabled, and previous versions persist unless deleted (directly or via lifecycle rules). Also supports MFA delete (remember for exam)

### Locking policies

* S3 Object Lock – WORM (write once, read many) model. Idea is to pin the object and ensure it will not get changed for a while (or forever) after being uploaded. Can be applied at object level or at bucket level.
  + Governance mode – locks it for most users
  + Compliance mode – locks it for EVERYONE, INCLUDING ROOT. Be very careful when setting up the retention period of compliance mode locked objects
  + Legal hold – works like a retention period, but it does not expire automatically. It can, however, be applied and removed at will
* Glacier Vault Lock – basically the same thing, but for vaults inside of Glacier (as opposed to objects in S3). Also does not have the modes?

### S3 Performance

* Prefixes (i.e. folders in the bucket)
  + Supports up to 3500 put / 5000 get requests per second, per prefix (that is, folder in the bucket). So if we want to go above those limits, we should spread our files across more folders in the bucket.
* KMS request rates
  + When using SSE-KMS, we call GenerateDataKey (when uploading) and Decrypt (when downloading), so that does also add limitations
  + Quota is region specific, either 5500, 10 000, 30 000 requests per second
* Multipart uploads
  + Recommended for files above 100 MB
  + Required for files over 5 GB
  + Breaks up the object into chunks and uploads the diff chunks in parallel
  + Download equivalent is Byte-Range Fetches

S3 Select

* Using a SQL query to only download the subset of data that we need from out objects, as opposed to downloading the entire file and the extracting what was necessary
* Glacier Select – same but for Glacier

AWS Organisations – consolidate multiple AWS accounts. Centralised management, consolidated billing (very useful for getting into higher usage tiers for stuff like s3, so getting the volume discounts)

* Root account – enable MFA, use complex password, only use for billing (don’t deploy any resources there)
* Enable/disable services on account level by utilizing policies applied to OUs (Organisation Units) and putting the accounts in different OUs

### Sharing S3 buckets between accounts

* Bucket policies and IAM at the entire bucket level (programmatic only)
* Bucket ACLs and IAM at the object level (programmatic only)
* Cross-account IAM roles (programmatic & console)

### S3 Cross region replication

* Versioning must be enabled
* Replicates files added after versioning was put in place, but not ones that were there since before that
* New files and changes do get replicated, deletes do not

### Transfer Acceleration

* Uploading to an edge location instead of to the S3 bucket directly. Thus making use of the lower ping
* Done through a distinct url
* Can actually be a tad slower if the main region’s AVs are super close to you

### Data sync

* Synching data between on-prem and AWS (can be done hourly, daily, or weekly)
* Need to install an agent on-prem
* Synchs with S3, FSx for Windows, and EFS
* Mostly used to move large amounts of data from on-prem to AWS
* Needs NFS or SMB compatible file systems
* Can also replicate EFS to EFS (so, from one place in the cloud to another)

### CloudFront

* A Content Delivery Network (CND) utilizing edge locations to reduce latency and increase accessibility to data for end users. The data can come from S3, EC2, ELB, or Route 53
* Distribution – the collection of edge locations we are utilizing. We can have a web distribution, which is used for websites, and an RTMP distribution usually used for media streaming (deprecated Dec 31, 2020)
* Edge locations are not read only, they can also be written to (like with transfer acceleration)
* Objects are cached for the TTL (Time To Live), which is configurable
* You can clear cached objects (called invalidating the cache), but you will be charged (useful if you’ve changed smth on the data, but its old version is still cached on the edge location and delivered to the customers)
* Can restrict access to the data using signed urls and cookies (think all the online magazines behind a paywall)

### Signed URLs and Cookies

* Used to provide access to restricted resources (cloudfront & S3)
* Use URLs for individual files (1 file = 1 URL)
* Use Cookies for multiple files (1 cookie = multiple files)
* Those signed URLs/Cookies include a policy with
  + A URL expiration, IP ranges, Trusted signers (which AWS accounts can create signed URLs)
* In the case of CloudFront the user accesses CloudFront, which then accesses the data via OAI (Origin Access Identity). In the case of S3, the user accesses the data directly. The S3 signed URLs have a limited lifetime

Snowball – a petabyte scale data transport solution (basically a cool, huge, portable disk). Used to enable data transfer without involving the internet, in a more old-fashioned mail form. Receive the snowball by courier, put all the data on it, send it to Amazon back with another courier.

* Comes in 50 & 80 TB versions. Very secure (both physical protections and 256 bit encryption)
* Snowball Edge – 100 TB, comes with on-board storage and compute capabilities. Basically a portable mini cloud (can ensure your apps run without access to the cloud)
* Snowmobile – Exabyte-scale data transfer service. A huge data container, housing up to 100 PB each, pulled by a truck

### Athena vs Macie

* Athena
  + Query service enabling us to analyse and query data stored on S3 using standard SQL. Good for analytics
  + Pay per query / TB scanned
* Macie
  + Machine Learning powered service that scans through our S3 data and looks for sensitive (as in, personally identifiable) information. Security service
  + Includes Dashboards, Reports, and Alerting

### Storage Gateway

* Connects on-prem software to cloud-based storage. Can be a virtual device or a physical one
* It’s a VM image to be installed on a host in the data center (supports Microsoft Hyper-V and VMware ESXi) and associated with your AWS account
* Three types:
  + File gateway (NFS & SMB)
    - Store files in S3 buckets, accessed through a NFS (Network File System) mount point
    - Once the files are in S3, they are just like all other S3 files (as in, all the S3 features and policies can be applied to them – versioning, lifecycle, CRR)
  + Volume Gateway (iSCSI – Stored & Cached Volumes) – used to store copies of (virtual) hard disk drives
    - The data on the volumes is stored in S3, while snapshots of the volumes is stored on Amazon EBS. Act like other snapshots – incremental backups, capture only changed blocks
    - Stored volumes – primary data stored locally on the storage gateway, and then replicated on AWS. Done so that entire dataset is available on-prem for low-latency access, and AWS is just backup
    - Cached volumes – unlike with stored volumes, not the entire dataset is kept locally, but just the most frequently accessed elements. Again, everything is replicated on AWS
  + Tape Gateway (VTL – Virtual Tape Library)
    - If using tapes, this can leverage the existing tape infrastructure. Basically lets you create virtual tape cartriges on the cloud on S3

Note – check onto the different urls (virtual hosting, path, static website).

# **EC2 – Elastic Cloud Compute**

EC2 – resizable compute capacity in the cloud.

## Pricing

* On demand
* Reserved
  + 1-3 years, no/partial/full upfront pay
  + Scheduled - only available during a certain time of the day
  + Convertible – can upgrade at certain point (only upwards)
* Spot
  + Making use of unrequired capacity. Much cheaper than a standard instance, but can be terminated at any point if someone outbids us.
  + Note – if terminated by EC2, you get charged only for the time you used the instance. If terminated by yourself, you get charged for the full hour.
* Dedicated hosts
  + All machines on the host are ours. Good for compliance.

Instance types mnemonic – Fight Dr McPxz AU

* F – FPGA (big dick shit, allows for the very chips to be reprogrammed. Think genomics research)
* I – (IOPS) - High Speed Storage (NoSQL, Data Warehousing)
* G – Graphics intensive (video encoding, gaming, 3D app streaming)
* H – High Disk Throughput (distributed file systems)
* T – Lowest cost, general usage
* D – Dense Storage (???)
* R – (RAM) Memory Optimised
* M – General purpose, higher cost (T’s big bro. Meant for heavier, more consistent workloads – no CPU burst credits unlike T’s system)
* C – Compute optimised
* P – Graphics/General Purpose GPU (Machine Learning, Crypto mining)
* X – Memory Optimised
* Z – High compute capacity / high memory footprint
* A – Arm-based workloads (???)
* U – Bare metal (physical server, avoid the overhead of hypervisor and VMs)

Note – can encrypt root device volumes (e.g. /dev/xvda for Linux) from the get-go (for default AMIs).

Termination protection is off by default. On an EBS-backed instance, the default is for the root volume to be deleted upon termination, but for additional volumes to not be deleted.

## Security Groups

Here we set the rules about accessing the instances (basically firewall settings – controlling ports). All inbound traffic is blocked by default. All outbound is allowed (hmm?). Rule changes take effect immediately.

Rules are stateful (hmmm?). If you create an inbound rule (e.g. open port 80 for requests from 0.0.0.0), an equivalent outbound rule is created automatically (i.e. open port 80 to return requests to anyone). Contrast with NACLs (Network Access Control Lists, VPC), which are stateless and we need to create an outbound rule for every inbound rule manually.

Can only allow things, cannot block (e.g. cannot block a specific IP). To blacklist things we need NACLs in the VPC settings.

Can attach multiple security groups, thus opening up everything from them (up to 5 by default, up to 16 by request). Can also attach any number of instances to a given security group. Max 60 rules (inbound or outbound, 120 total) per security group.

## EBS

EBS – Elastic Block Store (basically the VM’s VHD). Persistent Storage (survives stopping an instance, can even remain after instance termination). Automatically replicated within its own AV to provide high durability.

5 types

* General Purpose SSD (**gp2**) – balanced, standard
* Provisioned IOPS SSD (**io1**) – for databases
* Cold HDD (cheapest) (**sc1**) – cheapest, for infrequent access (file servers)
* Throughput Optimised HDD (**st1**) – Data Warehouses
* EBS Magnetic (**standard**) – old gen HDD. Up to 1TiB only (rest are up to 16)

IOPS vs Throughput – The former measures operations handled per second, whereas the later measures total bits handled per second (so, for larger operations)

EBS volumes should be in the same AV as their related EC2 instances (this is the default setup) – this is the only rational option, as otherwise there will be a lot of lag.

### Moving EBS volumes / EC2 instances to another AV

1. Create a snapshot of the root EBS volume
2. Create an image (AMI) from that snapshot (best to use HVM virtualisation type)
3. Create a new EC2 from that image, and select the desired new AV in the launch options (3 – Configure Instance Details – Subnet)

### Moving EBS volumes / EC2 instances to another region

1 and 2 – Same as above

1. Copy that image to a new region (Images -> AMIs -> Actions -> copy AMI -> new destination =region)
2. Use that image to create a new EC2 instance in that region

## Snapshots

Snapshots are point in time copies of EBS volumes (basically, the volume as it was at that time). They are incremental – only the changes since the last snapshot are saved (think .bak vs .trx on MS SQL). It is best practice to stop the instance before taking a snapshot of the root volume (also possible to do it with the instance running though). The first snapshot might take some time to create. We can create AMIs from snapshots.

Volumes exist on EBS, snapshots on S3. Can change EBS volumes on the fly, without stopping the instance (but might need to re-partition the file system of the instance to make use of the newly allocated space). This includes type (e.g. promote to Provisioned IOPS – io1) and size.

AMI types – EBS (external, persistent) vs Instance Store (internal, physical, ephemeral)

EBS – external volumes that attach to an instance. Practically independent of the instance, so they persist after shutdown (as long as we still pay for them).

* For EBS backed AMIs, the root volume is launched from an EBS snapshot

Instance store – physically attached to the instance, so they are ephemeral – they get deleted once the instance is terminated (not rebooted, but terminated. It cannot be stopped). This is so that the instance can be used by someone else while we aren’t using it

* For Instance store backed Amis, the root volume is launched from a template stored on S3

## ENI vs ENA vs EFA

ENI – Elastic Network Interface – **basic**ally a virtual network card. It allows for:

* One primary and some secondary private IPv4 addresses (from the VPC range)
* One elastic IPv4 address per private IPv4 address
* One public IPv4 address
* One or more IPv6 addresses, one or more Security Groups
* A MAC address, source/destination check flag, description

ENA – Enhanced Networking

* Utilizes single root I/O virtualisation (SR-IOV) to increase networking performance
* Helps with reducing latency, higher bandwidth
* No additional charge (but the EC2 instance has to support it)
* Can enable via ENA (Elastic Networking Adapter, **up to 100 Gbps**) or VF (Intel Virtual Function Interface, up to **10 Gbps**, used with older instances)

EFA – Elastic Fabric Adapter

* For **High-Performance Computing (HPC)** and **Machine Learning**. Faster than TCP communication, lower and more consistent latency. Can use OS bypass (Linux only).

## Encrypted root volumes and snapshots

New way of encrypting root volume – select encrypted as an option when creating the ec2

Old way (or when wanting to encrypt after having launched the ec2 already) – go to volumes, actions -> create snapshot, snapshots, actions -> copy, tick ‘Encrypt this snapshot’. Then, select copied snapshot, actions -> create image. Then launch from that image.

Snapshots of encrypted volumes are encrypted automatically – cannot launch an EC2 with an unencrypted root volume from their AMI. Same for volumes restored from encrypted snapshots.

Only unencrypted snapshots can be shared with other AWS accounts or made public.

## Spot Instances & Spot Fleets

Useful for things like big data, containerised workloads, CI/CD, web services.  
Don’t use for mission critical or persistent workloads or for databases.

Spot request – one time vs persistent.

* One time – if spot price goes beyond max price stipulated, instance gets terminated and that’s that.
* Persistent – if spot price goes beyond max price stipulated, instance gets terminated. Then, if it goes back down again, instance gets rebooted for as long as the price is under max price given.

Spot fleet – collection of spot instances (and optionally on-demand instances). So it tries to launch the requested number of spot instances, but it if can’t (say, there isn’t enough available capacity), it launches as many spots as it can and also some on-demands to fill in the remainder.

Spot Block – have a spot instance run for a finite duration (1 to 6 hours).

## EC2 Hibernate

Hibernate saves the current RAM contents to the EBS root volume. That way it boots much faster. Also, maintains the same instance ID. Requires that the root device volume is encrypted, there is enough free space on the root volume to fit the ram (and the ram must be less than 150GB), and it needs to be enabled upon ec2 launch. Cannot hibernate for more than 60 days.

Useful for processes that are long-running or take a long time to initialize.

## CloudWatch & CloudTrail

CloudWatch - monitoring service – Compute (EC2, Autoscaling Groups, Elastic Load Balancers, Route53 Health Checks), Storage & Content Delivery (EBS Volumes, Storage Gateways, CloudFront), etc…

Metrics are usually:

* CPU
* Network
* Disk
* Status Checks (Hypervisor – is it running, EC2 instance)

By default it monitors EC2 stuff every 5 min, can switch to every 1 min with detailed monitoring. Can also create CloudWatch alarms that trigger notifications, as well as dashboards, events, and logs.

Setting up a CloudWatch alarm:

* Recommended - Enable CloudWatch detailed monitoring.
* CloudWatch -> Alarms -> Create Alarm. Select Metrics, details and actions (as in – what to look for, datapoints (e.g. when 3 of the last 4 datapoints fulfill the criteria), what to do when the conditions are met – send a notification for example).

CloudTrail – logging service. Logs Console actions and API calls (what about CLI?). Can see who (users, accounts, IP’s), when (and what?).

## AWS Command Line (CLI)

Need an account’s access key id and secret access key in order to use the AWS CLI.

aws configure -> enter key id, secret key, region, output format

## Bootstrap Scripts

Basically a shell script to run upon instance creation -> #!/bin/bash

## Instance metadata

Information about the instance – e.g. IP address, instance ID, credentials, AMI, network, security groups, etc…

Find it at 169.254.169.254. For example, from within the EC2 –

* curl <http://169.254.169.254/latest/meta-data/local-ipv4>

## AWS EFS (Elastic File System)

File storage service that can be shared between different EC2 instances (Network File System version 4 – NFSv4). Scalable (meaning you pay for what you use, unlike EBS which is a specified size that you pay for, even if you use a tenth of it).

Create an EFS (make sure it is in the same subnet as the EC2 instances we want to use it)

yum install -y amazon-efs-utils on the EC2 instances -> then attach the EFS to the EC2 via sudo mount -t efs -o tls EFS ID:/ (e.g. fs-853687fd:/) relevant directory we want to put on the EFS (e.g. /var/www/html)

That makes sure that the EFS is mounted on the specified directories in both machines. Effectively, the directories act as separate doors leading to the same filesystem.

Windows FSx for Windows

Windows Server that runs **SMB (Server Message Block)** – based file services. Designed for Windows and Windows apps, supports AD, ACLs, groups and security policies, DFS (Distributed File System) namespaces and replication.

### Amazon FSx for Lustre

File System optimized for HPC – think machine learning, media data processing, etc. It can store data directly on S3.

Overall, EFS is for Linux, FSx for Windows is for… well, Windows (SMB protocol), and FSx for Luster is for high-performance computing.

## EC2 Placement Groups

Ways of placing EC2 instances

### Clustered

Placing the EC2 within a single AZ, close together. Done when we require good networking (high throughput, low latency) connection. Only certain instances can be launched in to a Clustered Placement Group, and it’s recommended to have homogenous instances (i.e. same type).

### Spread

The default option and the opposite of clustered – placing the instances on distinct (‘distinct’ means separate rack with separate power /network source) underlying hardware (can be within the same or different AZ’s). Done in order to minimize the impact of certain hardware failing.

Max of 7 instances per AV.

### Partitioned

Similar to spread placement group, but instead of having each individual instance on a distinct piece of hardware, we have each partition (i.e. group) of instances on a distinct piece of hardware.

Can move an already created instance in a placement group, but it must be in the stopped state. Only feasible via the SDK/CLI for now, not yet via the console.

## HPC (High-performance computing)

Important elements (bottlenecks) – Data Transfer, Compute and Networking, Storage, Orchestration and Automation

Data Transfer – Snowball/mobile, AWS DataSync (agent on a VM and push to AWS), Direct Connect (dedicated line form the data center to AWS)

Compute and Networking – EC2 instances (CPU or GPU optimized) and fleets, placement groups. Enhanced networking, Elastic Network Adapters, Elastic Fabric Adapters.

Storage – Instance-attached -> EBS (up to 64 000 IOPS with Provisioned IOPS) or Instance Store (up to millions of IOPS, low latency, ephemeral).

* Network -> S3 (object-based storage, not a file system), EFS (IOPS scales based on size or use Provisioned IOPS), Amazon FSx for Lustre (millions of IOPS, can be backed by S3)

Orchestration and Automation – AWS Batch (allows for batch computing jobs, can be multi-node in parallel (so a single job that spans many instances), can schedule jobs and launch EC2 instances). AWS PaeallelCluster – automates creation of VPCs, subnets, cluster and instance types.

## AWS WAF (Web Application Firewall)

Allows you to monitor http/s requests forwarded to Amazon CloudFront, an Application Load Balancer, or an API Gateway, and control the access to those (?) resources.

Essentially, can have more fine-grained control rules regarding requests to the abovementioned resources.

Three main modes – 1/2 – Allow/Block all requests but those that fit a condition, 3 – count all requests that fit a condition. Conditions can include IP addresses, countries, request length or parameters, potentially malicious SQL code or scripts.

## Quiz

Note – stuff to revise (88%)

An AMI continually uses the EBS snapshot it is created from, meaning that if we want to continue using that AMI, we cannot delete the EBS snapshot it uses.

For an EC2 that is already running, the ‘delete on termination’ (EBS root volume) must be changed via the CLI.

Underlying Hypervisors – Xen (old) and Nitro

Can perform actions on an existing EBS Snapshot

# **AWS Databases**

Relational – all records follow the same format (have the same columns, even if some have N/A in certain fields)

NoSQL – records are more in JSON format of key-value pairs. We can add more fields for one record without touching the rest of them

Relational Databases (RDS) on AWS -> MS SQL Server, Oracle, MySQL, PostreSQL, Aurora, MariaDB.

Two key features – Multi-AZ (when we’re going for high availability) and read replicas (when going for performance).

* Multi-AZ means we have two copies of the DB in two AZ’s (one primary and another synchronously updating backup). If one instance goes down, traffic gets automatically redirected to the second under the same DNS address. So, automatic failover (as in, Amazon automatically changes the relevant IPs and reroutes the traffic to the other instance).
  + Used for DR, not performance. Read replicas are for performance.
  + Available for anything but Aurora, as Aurora is designed to be highly available to begin with.
* Read replicas copy (asynchronously) the content of the main DB, but can only be read (for example traffic can be split between the main DB and the read replica to improve response time). You can have up to 5 Read Replicas, and each has a different DNS address. No automatic failover (so, if one goes under, gotta manually reroute the traffic).
  + Used for performance, not DR.
  + Available for anything but MS SQL
  + Must have automatic backups turned on in order to use read replicas
  + Can have read replicas of read replicas (but watch out for the replication latency)
  + Replicas can be promoted to become their own databases (breaks replication)
  + Replica can be in a different region, and also can work together with Multi-AZ

NoSQL -> DynamoDB (Neptune, DocumentDB (MongoDB compatible) ?)

Data Warehousing – for data analytics (business intelligence) – for very large data sets – OLAP (On-line Analytics Processing) (as opposed to OTLP – online transaction processing) - Redshift

In-memory databases – used to cache frequently accessed data, thereby speeding up the DB performance a lot – Elasticache (Redis and MemcacheD)

Backups – two types

* Automated
  + Recover to any point in time within a retention period (between 1 and 35 days). Same as MS SQL server – full daily snapshot and transaction logs in between. Enabled by default, accurate to the second, during backup I/O is suspended and thus there might be higher latency. Backups are stored on S3, where we get free storage the size of the database itself.
* Database Snapshots
  + Taken manually, stored even if you delete the OG RDS instance (unlike the automated backups).
* Restoring backups leads to a brand new DNS for the restored version.

Encryption – via the Amazon KMS. At rest – available for all RDS types (Oracle, MS SQL, MySQL, Postresql, MariaDB, Aurora), and as soon as it’s turned on, it encrypts the DB as well as snapshots and backups.

## DynamoDB

- Amazon`s (read – fully managed) NoSQL solution. Advantages – consistent, single-digit latency at any scale (basically, speed).

- Supports both dict (key-value pair) and document data models (formats)

Stored on SSD (yup, all about that speed), spread across 3 geographically distinct data centers (for redundancy)

All data fully encrypted at rest (via KMS).

Eventual Consistency Reads (default), but can activate Strongly Consistent Reads. Difference is that with eventual consistency any change will be seen in reads after 1 sec, whereas with strongly, changes can be read in 1 sec or less

Paying per DB size and for the read/write requests (either provisioned (i.e. reserved) or on-demand).

**DynamoDB Accelerator (DAX)**

- fully managed, highly available, in-memory cache (up to 10x performance improvement) – takes requests from milisecons to microseconds

- Completely compatible with existing DynamoDB API calls. Basically, the idea is that instead of having a separate cache with its own logic and separate calls, the cache is inbetween the app and the DB. This way it offloads some of the read traffic that would otherwise reach DynamoDB and makes it less likely to throttle (as it reduces the need for read replicas on the DB side).

Good for read intensive apps, but not for write intensive ones [because it only has eventual consistency on the writes due to its multi-node structure (basically uses multiple memory optimised EC2`s as the cache)]

**Transactions**

Multiple „all or nothing“ (two things must happen simultaneously, or nothing should happen. Example – financial transactions, where one account is credited and another is debited.) operations. Each write (and read??) involving a transaction takes two requests – one to prepare the transaction, and one to commit it. Meaning – transactions cost double the requests.

Up to 25 simultaneous items and up to 4MB of data at any time.

**On demand capacity** (standard option is provisioned - basically compare to bandwith)

Pay-per-request pricing, no minimum capacity request, best for unpredictable traffic

**Backup and restore**

Full backups at any time, backups and restore execute with zero impact on table performance or availability

Consistent within seconds (wtf does that mean?), retained until deleted – no specific period

Backup and restore works in the same region as the source table.

Point-In-Time Recovery – can restore to any time inbetween the last 35 days and the last 5 minutes (works via incremetal backups). Not enabled by default

**Streams**

Time-ordered sequence of item-level changes in a DynamoDB table. Basically a log of changes? Stored for 24 hours, logs inserts, updates and deletes. Can combine with Lambda to create the NoSQL equivalent of stored procedures (basically saved queries)

**Global Tables**

Managed (so no need to rewrie the app to make use of that) Multi-Master Multi-Region Replication – one master per region, cross replication?

Ideal for globaly distributed apps, enhances HA (high availability) and DR (Disaster Recovery)

Replication latency under one second

**Database Migration Service (DMS)**

Automates the migration of the entries from the source DB to a target DB. Source DB remains fully operational during the migration. Supports both homogenous (same DB type, say Oracle to Oracle) and heterogeneous (say, MS SQL Server to Aurora) migrations. The latter necessitates the use of AWS SCT (Schema Conversion Tool).

It can either create the tables and primary keys on the target itself, or the user can do so manually (or via the AWS Schema Conversion Tool – SCT).

It’s practically a task that runs on an AWS machine, which pulls the data from the source DB and pushes it to the target DB.

Source DB (Can be on-prem, EC2, or RDS) → DMS → Target DB (Can be on-prem, EC2, or RDS)

## Redshift

Fully managed data warehousing service (for BI). Can scale up to petabytes.

Reminder – data warehousing is used for Business Intelligence. That is, while traditional databases hold trasaction records (and are used for editing or adding individual rows), data warehouses are used for Analytics purposes, where most queries involve complex logics that involves multiple tables and rows (think back to Ingenico).

Configuration:

* Single node (160GB)
* Multi Node
  + One Leader that manages client connections and receives queries
  + One or more (up to 128) compute nodes that store the data and perform the computations

Available only in one AZ (so no Multi-AZ setup like the RDS. Makes sense, as its unlikely to be business critical?). Can restore snapshots to a different AZ tho, in the event of an outage.

Uses **Advanced Compression** – Compresses by column instead of by row, which makes it more efficient. No need for indexes or materialised views (wat?)

**Massively Parallel Processing (MPP)** – Basically, if using multinode setup, autobalances the load between the nodes.

**Backups** – enabled by default, 1 day retention period (max 35 days). Tries to (?) keep at least 3 copies – Original and replica on the nodes, and a backup on S3. Can also async replicate snapshots to S3 in another region for DR purposes

**Pricing**

* Compute Node Hours, basically like EC2. Pay per the total computation time on the computation nodes (leader node hours not included). So 3h on 8 nodes = 24 hours.
* Backups (storage? check)
* Data Transfer

**Security**

In transit – SSL (so connection to Redshift only over 443?). At rest – AES-256, AWS managed keys (but can use own keys as well, via KMS or HSM (Hardware Security Module)

## Aurora

Amazon proprietary DB – tries to combine speed and availability of popular consumer DBs (a la Oracle and SQL Server) with the ease of use (and affordability) of open source DBs (MySQL & PostgreSQL compatible).

Starts at 10GB and scales in 10GB increments up to 64TB (Storage Autoscaling – does it automatically). Compute resources can scale up to 32vCPU and 244GB of Memory.

2 copies of the data per AZ, min 3 AZs. So 6 copies total. Can handle the loss of two copies without affecting write availability, 3 copies without affecting read availability. Self-healing – data blocks and disks continuously scanned for errors and repaired automatically (Think RAID 5 or 10 with hot spares?)

Up to 15 read replicas (5 for MySQL, 1 for PostgreSQL). Automated failover for Aurora replicas only

**Backups**

Always enabled, do not impact DB performance. Can also take snapshots (again no impact on performance) and even share those snapshots with different AWS accounts.

**Aurora Serverless**

On-demand, autoscaling config for the MySQL and PostgreSQL-compatible editions of Aurora. Automatically starts, scales capacity, and shuts down based on the connected app`s needs. Basically on-demand DB, good as a cost-effective solution for infrequent, intermittent, or unpredictable workloads

## Elasticache

In-memory cache, improves DB & web app performance. Supports two in-memory caching engines – Memcached & Redis.

The latter has more capabilities (Multi-AZ, backup and restore, advanced data types), but Memcached is simpler and offers multi-threaded performance.

## Caching strategies on AWS

Caching is a balancing act between up-to-date, accurate information and latency. Services that make use of caching:

* CloudFront (its very purpose, cache in edge locations closer to the users)
* API Gateway
* Elasticache (Memcached & Redis)
* DynamoDB Accelerator (DAX)

## EMR (Elastic Map Reduce) Overview

Big data analysis platform. Central component is the cluster of EC2’s, each of which is a node. The role of the node in the cluster is its **type**, and EMR puts different software on each node depending on its type.

* Master Node – tracks status of tasks and monitors cluster health. Obligatory.
* Core Node – runs tasks and stores data in the Hadoop Distributed File System (HDFS). Cluster needs at least one of those.
* Task Node – only runs tasks and does not store data. Optional (horizontal scaling?)

Logs are stored on /mnt/var/log/ on the Master Node. To make sure they would persist through a failure/shutdown of the cluster/Master Node – configure a cluster to periodically archive them to S3. Usually this archival to S3 is done at 5 min intervals. This configuration **CAN ONLY** be done upon first creating the cluster. No option to create a cluster and then later add archiving logs to S3.

## Quiz

When you add a rule to an RDS DB security group, you must specify a port number or protocol.

If you are using Amazon RDS Provisioned IOPS storage with a Microsoft SQL Server database engine, what is the maximum size RDS volume you can have by default?

Which of the following data formats does Amazon Athena support?

If you want your application to check RDS for an error, have it look for an \_\_ code in the response from the Amazon RDS API.

Which AWS DB platform is most suitable for OLTP? – RDS vs Dynamo, why?!?

What happens to the I/O operations of a single-AZ RDS instance during a database snapshot or backup?

You are hosting a MySQL database on the root volume of an EC2 instance. The database is using a large number of IOPS, and you need to increase the number of IOPS available to it. What should you do?

Under what circumstances would I choose provisioned IOPS over standard storage when creating an RDS instance?

# **Route 53**

## DNS 101

Route 53 because port 53 is reserved for DNS

Top level domains can be found here - iana.org/domains/root/db/

All domain names are registered in the WhoIS database as to avoid duplication (that’s done by domain registrars like Amazon, GoDaddy, Namecheap).

DNS resolution process:

Overall, regular LDAP process

* Request goes to the root serves [Of IANA (Internet Address Numbers Authority) ?]
* They refer the request to the top-level domain server
* (If applicable) That domain refers the request to the subdomain
* Eventually, within the last (sub)domain, the name is searched for and the IP is found

ELBs do not have pre-defined IP addresses, you resolve to them using a DNS name

Alias record vs CNAME – always choose Alias record over CNAME (basically the same thing, but CNAME cannot resolve to naked domain names (zone apex address), e.g. [http://example.com](http://example.com/)). Alias records are a specific thing for AWS

SOA (Start of Authority) record – contains admin info and TTL (Time To Live - default 48h)

NS (Name Server) record – points to the Authoritative Name Server, which contains the SOA

A (Address) record – used to look up an IP from the associated name

CNAME (Canonical Name – mapping one DNS name to another, so that both refer to the same IP address. For example, m.website.com and mobile.website.com)

MX (mail) records

PTR Records – reverse of A record, used to look up a name from its IP address

Getting a domain – buy straight from AWS, can take up to 3 days to register (usually much shorter). Services -> Route53 -> Register (or Transfer) Domain -> pick a free name, fill in details. Default limit of 50 domain names per account.

## Routing Policies

To adjust the policies: Route53 -> Hosted Zones - > Create an A record -> Routing Policy

* Simple
  + One record with multiple IP addresses (user gets a random one upon request, which remains for the duration of the TTL)
* Weighted
  + Split traffic based on different weights assigned. E.g. – 10% to eu-east-1, 90% to eu-west-1
  + Single record per IP address, add the weight
  + Weight is relative, and not necessary to add up to 100 – we can have weights of 6 and 24, for 20% and 80% chance of the user being sent there respectively
  + Can associate each record with a health check, so if a resource is unavailable it gets removed from the list of addresses. Say if we have 3 IPs with weights 1, 2, and 97, almost the entirety of the traffic will go to 97. If it does not pass its health check, now the traffic will be split 33/66 between the 1 and the 2.
* Latency-based
  + As the name suggests, routes the traffic to the path that results in the least latency for the end user.
* Failover
  + Effectively the same as weighted with a health check. Sends all traffic to the primary address, if that one fails its health check -> sends all traffic to the secondary address.
* Geolocation
  + Chooses where to send the traffic based on the geographic location of the users (think of a website automatically sending you to the German version when you are in Germany for example)
  + Defined by national boundaries
* Geoproximity (traffic flow only) – not important for exam
  + Similar to geolocation, but also takes into account the geographic location of the resources
  + Defined by latitude/longitude
* Multivalue Answer Policy
  + Same as simple routing, but gives the option to associate the different IP addresses with health checks
  + Addresses need to have a separate A record each (so effectively more like weighted with all similar weights then)

Health checks – can create health checks on individual record sets (e.g specific IP address), that monitor is the IP has gone down. If it has, it can be configured to alert us.

# **VPC**

Virtual Private Cloud – a logically isolated section of the Amazon Cloud, where we can set up our own network (we select a range of IP addresses, subnets, route tables and network gateways). Like a data center in the cloud. We are allowed 5 VPCs per region by default.

Private IP addresses are kept through restarts (for EBS backed instances), Public ones are not.

Can create a Hardware Virtual Private Network (VPN) connection between on-prem and AWS and treat the AWS resources as an extension of the on-prem datacenter. A VPN consists of a Customer Gateway and a Virtual Private Gateway.

Jumpbox / Bastion – a machine in a VPC which we can access from the internet, and from which we can then ssh into other VPC machines that are not directly accessible from the internet.

Largest subnet we can create in AWS VPC is /16, smallest -> /28. Each subnet is in one Availability Zone only. We cannot have a subnet that spans multiple AZ’s

VPC – logically separated partition of the cloud (datacenter). It consists of Subnets, Internet Gateways (or Virtual Private Gateways???), Route Tables, Security Groups, and Network Access Control Lists (NACLs)

Subnets – separate networks within the datacenter. Subnets cannot span AZs, always one subnet – one AZ!

Route Tables – determine where does the traffic flow through.

Internet Gateways – allows the VPC to talk to the internet

Security Groups – act like web firewalls (only permissive rules, opening ports for inbound traffic. Outbound traffifc always allowed by default. Cannot span VPCs (so, unique for each VPC).

NACLs (Network Access Control Lists) – more granular cotrol of traffic. Can allow and deny more specific things

Creating a new VPC does automatically create a new route table (with singular entry, so that our different subnets can talk to each other?), NACL, and a Security Group. We need to set up the subnets and the Internet Gateway ourselves.

Internet → Route Table → NACLs → Security Groups → Subnet

Subnets – AWS reserves 5 addresses of each subnet

0 - network address

255 - broadcast address

1 – VPC router

2 – DNS Server

3 – Reserved for future use

Auto-assign public IP – for networks we want to expose to the Internet. EC2 instances we add in those groups will automatically be assigned a public IP when created (when added to the group?).

## Internet Gateway

Is the gateway through which internet-bound traffic of the subnet gets sent, and through which subnet-bound traffic from the Internet arrives. Is the point where NAT (Network Address Translation) happens.

One Internet Gateway per VPC.

Only instances with Public IP addresses can access the internet via the IGW(?)(!).

## Route Tables

Contains a set of rules, called routes, that are used to determine where network traffic from your subnet or gateway is directed

New subnets that have not been associated with any route table get associated with the main one by default. Thus, it would be a good idea to keep the main route table not public.

When adding gateways, we should mention them in the route table (e.g. for NAT gateways, when we are trying to reach the internet → destination would be 0.0.0.0/0, target the gateway).

AZs are randomised. us-east-1a in one account might not be the same AZ as us-east-1a in another (might be us-east-1c there). Done to make sure that the natural human bias towards the first options does not tilt the load spread between the Azs.

Security Groups act as firewalls. However, for them to even trigger, we need the routing to be on point! So, the subnet should be connected to an Internet Gateway through the route table, if we want to even have the technical chance of reaching it.

## NAT Instances & Gateways

Necessary in order for our EC2’s in private subnets to still have the opportunity to download and update software (basically to stay private, but still have access to the internet).

NAT Instances are single EC2’s that cover that functionality, whereas NAT gateways are highly available, span multiple AZs. NAT Instances are becoming obsolete, but are still on the exam.

1. Create the NAT instance in the public subnet.
2. Disable source/destination checks. (EC2 instances perform source/destination checks by default. Meaning that the instance must be either the source or destination of traffic for it to pass through)
3. Create a route in the (default, main) route table of that VPC, destination 0.0.0.0/0 (internet), target – the NAT instance.
4. Should be behind a security group (like web DMZ)

Using a NAT instance creates a single point of failure, and can also be easily overwhelmed if used by multiple servers, meaning its a bottleneck. To deal with the latter, we can increase instance size. To deal with the former, we can design for HA with autoscaling groups, instances in subnets in different AZs, and a script to automate failover (?)

NAT Gateways

Intuitive to make, same as NAT instance. Steps 1, 2, & 4 don’t concern us, just need to add the route.

HA within the AZ. Can direct traffic from multiple AZs to a single NAT gateway, but if its AZ goes down, all instances would lose internet access. Best practice – have one gateway per AZ, and direct resources from that AZ to that instance.

Starts at 5 Gbps, scales to 45 Gbps.

## NACLs vs Security Groups

Each subnet can have only one NACL (unlike the security groups). But one NACL can have multiple subnets assigned to it. Every subnet must have an ACL. If none is assigned, it is associated with the default NACL, just like with the security groups.

The default ACL allows everything. Newly created NACLs start with the default setting of denying everything. To let communication pass through, add allow rules on specific ports (and potentionally, for specific origin addresses).

For internet traffic to run well, open up ephemeral ports (1014-65535) in the outbound rules!!

Rules are being evaluated in order → lower to higher numbers. So, rule 100 will be evaluated before 200, 300, and so on. Thus, if rule 100 allows something that rule 200 denies, that thing will be allowed. And vice versa.

Also, NALCs are evaluated before the Security Groups. So if a port is denied on the NACL, it wont even reach the Security Group.

To make sure the instance can also use yum update, make sure to add the ephemeral ports for the inbound rules as well.

Differences between NACLs and Security Groups

- NACLs get evaluated first

- NACL’s rules are evaluated in numbered order, Security Group rules are aggregated

- NACLs are more granular – can deny a specific IP (?)

- NACLs add the opportunity to deny things, Security Groups only open up ports

- ONLY ONE NACL per subnet, while there can be multiple Security Groups (???)

- NACLs are stateless, responces to allowed outbound traffic are subject to the inbound traffic rules (and vice versa). Security Groups are stateful, meaning that if you send a request from your instance (outbound), the response traffic for that request is allowed to flow in regardless of inbound security group rules.

- NACLs are at the subnet level, while Security Groups act at the instance level

Custom VPCs and ELBs

Types of ELBs:

* Application (HTTPS)
* Network (TCP)
* Classic (Previous gen - both app & network)

ELBs can be internet-facing, or internal. They are placed in a specific subnet (or AZ? Test!), so if internet-facing, the subnet should have an Internet Gateway.

A loadbalancer requires at least two public subnets! (For ALB – in two different AZs)

## VPC Flow Logs

Logs internet traffic coming from and to network interfaces(?) in our VPC. Stored in Cloud Watch.

Can log at 3 levels – VPC, Subnet, Network Interface

Cannot enable log trafficking for peered VPCs, unless both are within the same account.

Can tag flow logs. Cannot change their configuration though, once they are created they are set in stone.

Not all IP traffic is logged. The following are not logged – traffic to Amazon DNS Servers, traffic for Windows license activations, traffic to 169.254.169.254 (for instance metadata), DHCP traffic, and traffic to the VPC router

## Bastions (jumpboxes)

Computer on a network (either outside a firewall or, as in our case, in a DMZ (that is, a public subnet) that has been configured specifically to withstand attacks. Usually only hosts a single application (i.e. a proxy server) and all else is removed or limited in order to limit attack vectors.

While we use NAT gateways or instances to let servers in private networks access the internet, we use Bastions to access those instances FROM the internet (ssh or rdp).

Bastions are within our public subnets, thus located behind the Security Groups and ACLs.

Cannot use a NAT gateway as a Bastion host.

## Direct Connect

Dedicated network connection between on-prem and the AWS cloud. Basically, a literal physical cable connection to the AWS. Works like this:

AWS has its own Direct Connect locations (kind of like edge locations?). There, you can purchase a physical router which will be connected to the local AWS router, which is plugged into the AWS backbone network. All that is needed is the „last mile“ -> a private, physical connection between our datacenter and the AWS Direct Connect location.

Done to avoid network congestion and bandwidth limits, so useful for high throughput workloads, or when we need a reliable and secure connection.

Steps for setting up (a VPN over our VPC?):

* Create a public virtual interface in the Direct Connect console
* VPC Console -> VPN Connections -> create a Customer Gateway
* Create a Virtual Private Gateway
* Attach Virtual Private Gateway to desired VPC
* VPN Connections -> create new VPN connection
* Select the Virtual Private Gateway and the Customer Gateway
* Set up the VPN on the customer gateway

## Global Accelerator

Directs traffic to optimal endpoints in the AWS infrastructure.

By default – 2 static IP addresses that we associate with our accelerator (alternatively, can use our own).

Basically, instead of going through the internet and traversing multiple public networks to reach the IP of the EC2 (or load balancer), our traffic goes to the nearest Edge Location, and then gets intelligently routed by AWS within their infrastructure. Also uses healthchecks, so if a resource is down, it knows and does not route traffic to it, hence improved availability (customers never know something was down). Moreover, due to using static (anycast) IPs, the customers always have the same IPs cached, meaning that failure somewhere does not mean they‘d be sent to a dead resource until their cache expires.

Static IPs

* Anycast IPs, so they always remain the same

Accelerators

* Associated with the abovementioned IPs. Receives a DNS name (smth.awsglobalaccelerator.com) and includes at least 1 listener. Directs the traffic through the optimal route, and to the optimal endpoint.

Network Zone

* Isolated subnet (one per static IP). There for HA, same as AZs.

Listeners

* Processes requests based on the port specified. Each listener has an endpoint group(s) associated with it.

Endpoint Group

* Basically a region

Endpoint

* Loadbalancers, EC2s or EIPs.

Can control the traffic like with weighted routing policy (using traffic dials, at the endpoint group).

## VPC Endpoints

Allows our VPC resources to connect to supported AWS services without leaving the Amazon network. Basically makes an internal connection to those resources over the AWS backbone – without using public IPs, IGWs, NAT Gateways, VPN connections, Direct Connect connections, etc...

* Interface Endpoints
  + ENI (elastic network interface) with a private IP address, that is an entry point for traffic destined to a supported service (whole bunch of those)
  + Powered by AWS PrivateLink
* Gateway Endpoints
  + Similar to NAT gateways, support only S3 and DynamoDB

The route tables are where the magic happens. When we add a VPC endpoint, it effectively acts as a NAT gateway – so we have to specify in which route table do we want it to go. Then it acts as a gateway for the subnets associated with that route table (meaning that only those subnets will be using it).

After adding a VPC Gateway endpoint, we need to specify our region when passing our requests (e.g if our ec2 is in us-east-2, then our request becomes “aws s3 ls –region us-east-2” instead of the former “aws s3 ls”).

## AWS Private Link

Makes it easier to have peering between multiple VPCs

Opening up applications on one VPC for access from other VPCs:

* One way is to simply open them to the internet. Obvious security drawbacks
* Another is to use VPC peering. Good in small numbers, hell to scale (need a new peering connection for each VPC that we want to be able to access our own)
* AWS provided way is with Private Link
  + Provides a connection between the NLB (Network Load Balancer) of the service in the VPC we want to open, and the ENI (Elastic Network Interface) of the VPC we want to provide access to
  + Basically a good and easy way to scale VPC peering

## AWS Transit Gateway

Basically a router acting as a central point of connection, used to simplify network topology. Instead of having a bunch of interrelated connections between different VPCs, on-prem sites, etc, we just access the transit gateway, and go to our target through there.

Allows for transitive peering between thousands of VPCs and on-prem networks.

Hub-and-spoke model. Regional basis, but can be used across regions, as well as across AWS accounts (via RAM – Resource Access Manager). Also works with Direct Connect and VPN connections.

Can use route tables to limit how VPCs talk to each other. Supports IP multicast (no other AWS service supports that).

## VPN CloudHub

If we have multiple sites with their own VPNs, we can integrate and connect those through a single gateway. That way all of the sites can access the AWS resources through that gateway, and can also access each other’s VPNs.

Again hub-and-spoke model.

## AWS Network Costs

Traffic coming in (as in, requests) to the VPC is free.

Then, traffic inbetween VPC components:

* Within the same AZ, via PrivateIP –> Free
* From one AZ to another
  + via PrivateIP –> very cheap (say, 1 cent/GB), as it stays on the AWS backbone
  + via PublicIP –> more expensive (say, 2cents/GB), as it leaves the backbone
* Inter-region – again more expensive (say, 2cents/GB)

Overall – PrivateIPs help save costs, as the traffic remains within AWS. If we put everything in the same AZ and communicate via PrivateIPs, network costs will be fully eliminated.

N.B. – will be exposed to a single point of failure in such a case!

Build a VPC from memory

Private and public subnets, make sure the ones form the public subnet can communicate with the ones in the private, and that the instances in the private one can use yum update

# **High Availability**

## Elastic Load Balancers

Types:

* Application Load Balancers (HTTP & HTTPS)
  + Operate at Layer 7, and are application-aware (meaning, can create advanced request routing and send specified requests to specific web servers – e.g. can see that the site language was changed to French, so it can balance the requests to the French servers)
* Network (TCP)
  + Operate at Layer 4, and are used where extreme performance is the priority. Can handle millions of requests with ultra-low latency.
* Classic (legacy, can be used for either http/s or tcp)
  + A bit cheaper (so, can work if we don’t try to be smart with the routing but just do simple round robin)

In case of application issues returns 504 (Gateway Timeout). This means we should look at the application for the issues.

Instances monitored by an ELB are reported as either InService or OutOfService.

X-Forwarded-For

* Usually, when our EC2 receives a request from our ELB, it receives it with the IP of the ELB instead of the IPv4 of the customer who sent it. We can find the IPv4 address of the original user that sent the request to the ELB in the X-Forwarded-For header.

We never get an IP for an Application/Classic Load Balancer, only ever a DNS name.

## Sticky Sessions

* Useful if storing information locally
* In the case of Classic Load Balancer - binds a user’s session to a specific EC2
* In the case of an ALB – binds a user’s session to a specific Target Group

## Cross-Zone Load Balancing

* Without it, an ELB in one AZ would only balance the load it receives across servers within its own AZ.
* With it, any ELB can balance its traffic across servers in all of the different AZs.

## Path Patterns

* A listener that can forward requests based on the url path. Known as path-based routing. Ideal for microservices (think back to Zetta and the different EC2’s/target groups) for the different site elements.

## Auto Scaling

Groups

* Logical component where we put our EC2’s (e.g. Web Servers, DB Servers, etc)

Configuration Templates

* A template for the instances to raise – AMI, instance type, key pair, Security Groups, block device mapping(?)

Scaling Options

* Think of a HPA in Kubernetes. Select minimum, desired, and maximum capacity (i.e. – number of EC2 instances)
* Can be based on a condition (dynamic scaling, say CPU utilization) or a schedule
* 5 options:
  + Maintain current instance levels
    - I.e. only raise new instances if problems with the old ones). Based on periodic health checks, if an instance fails – it gets terminated and a new one is raised.
  + Scale manually
    - Change min/desired/max capacity, and AWS removes/adds instances accordingly
  + Scale based on a schedule
    - For example, add a bunch of servers every day after work hours if you’re Netflix, and take them down gradually from 2am onward.
  + Scale based on demand
    - Use policies which let you determine the parameters that control the scaling process. Good example – when all the servers are getting over 90% CPU util, add another one. If they are all under 30% - remove one.
  + Use predictive scaling
    - Kinda like schedule, but automatic

## Bastion HA

Two (or more) Bastions in different AZs, with an NLB in front of them. Has to be an NLB instead of an ALB or Classic, because the other ones work on layer 7, whereas an NLB works on layer 4 and can forward our SSH/RDP connections. Also, the NLB has a static IP address.

Alternatively, we can have an autoscaling group for our Bastions with an Elastic IP address and a user data (bootstrap?) script, where if one fails, it gets auto replaced. N.B – downtime until health checks detect that one Bastion is down, as well as while another is provisioned.

## On-Prem Services with AWS

Database Migration Service (DMS)

Server Migration Service (SMS)

AWS Application Discovery Service – maps your datacenter

VM Import/Export

Download Amazon Linux 2 AMI as an ISO

## Elastic Beanstalk

Watch again

Sets up the necessary infrastructure (can slightly configure some vars though) automatically, so we can just deploy code and have it running immediately.

'Availability' - Availability can be described as the % of a time period when the service will be able to respond to your request in some fashion. Uptime

'Durability' - the likelihood that a resource will continue to exist until you decide to remove it

'Reliability' – Performs as expected

'Resiliency' – likelihood of a resource to recover from damage or disruption

## Setting up an entire architecture example (Wordpress site)

Architecture:

* Customer-facing side
  + Route 53 for the DNS
  + CloudFront for the latency improvements
* Application side
  + S3 for storage (separate buckets for code & media)
  + EC2s for Application and DB
* Networking side
  + Autoscaling groups & mirror 2-AZ deployment for HA
  + ELB
  + NAT gateways
  + Separate Security Groups for the DBs and App Servers

First lesson learned – fucking mind the route tables on new VPCs! Gotta add an internet gateway, then add it to the routing tables.

Second lesson – bootstrap scripts cannot execute if the environment has not been set up properly yet. So mind the internet gateway and route tables on a new VPC.

* The default Amazon Linux 2 AMI comes with some standard, older versions of packages in the default repo. If we want easy access to newer versions through yum:
  + Enable required package (if present) through amazon-linux-extras
  + Clear metadata (yum clean metadata) && install

# **Applications**

## SQS (Simple Queue Service)

Provides a way to decouple systems by putting a message queue between the parts that process messages and the ones that create those messages. That way, if a work server is overburdened, a backlog (queue) will form without work being lost. A message can be kept there for between 1min and 14 days, default 4 days. Max message size – 256 KB text (or more if it goes in S3).

Standard queues

* Near unlimited transactions per second, however some might be duplicates and some might be out of order

FIFO (First In First Out) queues

* 300 transactions per second max, but they will be all in order and without duplicates

Long polls – while short polls just query the queue and immediately return a result (even if the queue is empty), long polls can wait for a while before either a message shows up in the queue, or their timer expires. Max waiting time 20sec.

Visibility timeout – when a component picks up a message from the queue, that message gets made invisible for a certain time (max 12 hours). If the message is processed in that time, it gets deleted from the queue. If it does not, it gets made visible again and something else can pick it up. A too low visibility timeout might result in certain messages being processed twice.

## SWF (Simple Work Flow Service)

Similar to SQS, but creates workflows (a collection of which is called a domain) instead of message queues. Main difference is that some tasks in the workflow can involve human interaction. A task would be assigned only once and never duplicated.

Consists of

- Workflow Starters - an app that launches a workflow

- Deciders - control the workflow (if smth finishes of fails, the deciders decide what happens next)

- Workers - they carry out the activity tasks

SWF vs SQS

- SQS max retention is 14 days, SWF workflow executions can last for 1 year.

- SWF is task-oriented API, whereas SQS is message-oriented API.

- SWF keeps track of the tasks and events in an application. With SQS, you gotta implement that yourself.

## SNS (Simple Notification Service)

Enables pushing (unlike SQS where we pull) messages from an application to its subscribers. Can deliver the messages to mobile phone OS’, SMS, email, SQS, and any http endpoint.

Can group subscribers based on topics (e.g. billing vs autoscaling alerts).

## Elastic Transcoder

A cloud tool that enables you to convert media from one format to another. Provides presets for popular devices (i.e. for Android/Iphone). Pay based on the minutes to be transcoded and on the resolution.

## API Gateway

A place for all user API calls to go to, which then divides and redirects them as appropriate. Because they all pass through there, it also enables better monitoring, logging, and control of all the incoming traffic. Scales automatically, but we can also throttle it to prevent attacks.

Can cache endpoint responses (for a certain ttl). Enable CORS on the API gateway.

## Kinesis

Streaming data is one that is generated continuously, often by many sources, sent simultaneously, and usually quite small (KBs). Kinesis is a platform to send streaming data to in order to load and analyze it.

* Streams
  + Place to store the received streaming data. Default 24h, can store it for up to 7 days
  + Data is kept in shards that can be accessed and analyzed by EC2 instances (consumers). Once processed, that data can be stored in other places (e.g. S3)
* Firehose
  + No persistent storage – the data should be analyzed as it comes in
* Analytics
  + Works with both streams and firehose

## Web Identity Federation & Cognito

Web Identity Federation – basically login via google/facebook/amazon.

User authenticates with FB/Google, they give that user a token, the user sends the token to Cognito and Cognito provides them with temp credentials mapped to whatever IAM role they can use. It also syncs user data across multiple devices (push synchronization via SNS – also sends a notification when user data saved in the cloud changes).

* User Pools – user directories to manage sign-in and sign-up for apps. User based.
* Identity Pools – provide temp AWS credentials for access to AWS services such as S3
* So user pools deal with the users, whereas Identity pools deal with granting rights (authorizing access) to an IAM role.

## Event-Driven Architecture

Apps can be decoupled via Pub/Sub (Publish/Subscribe) Messaging. That allows messages to be broadcast to different parts of the architecture asynchronously. Prime example – SNS, where publishers (which can be our app or different AWS services) can publish a message to a topic, and all of the subscribers instantly receive a copy of that message. If messages there fail all retry logic, they will be sent to the Dead-Letter Queue.

* Dead-Letter Queue (DLQ):
  + Effectively a ‘lost and found’. When an SNS message fails to deliver, it gets sent to an SQS queue where it is held for further analysis or reprocessing. In SQS, if a message is received by an EC2 beyond a certain number of times (say, it cannot be processed, so it never gets deleted, as SQS guarantees at least once processing), it gets sent to the dead-letter queue. Finally, Lambda functions that end in an error after the second retry are also sent there.
  + Effectively, the DLQ is a list of issues (bugs, exceptional cases) that are to be looked at at some point.
* Fanout Pattern:
  + When instead of sending new messages directly to different SQS Queues, we send them to an SNS topic that then sends them to the SQS Queues
* S3 Event Notification
  + S3 can also talk to SQS Queues, SNS Topics, or Lambda functions about some events that happen in the bucket. To make sure events are not missed, we need to enable versioning.
  + Events that qualify – Object Created/Removed/Restored(from Glacier)/Lost(from RRS), Replication(fails, takes too long, no longer tracks smth)

Scaling EC2 using SQS

# **Security**

Blacklisting IPs can be done via NACLs (at the subnet level of the EC2s, or of the ALB) or WAF. Also, we can put a host-based (as in, on the EC2 level) firewall as well (win firewall, iptables for linux). The more buffers we put in front of our traffic, the further away must we push our controls. This is because when we put stuff like Load Balancers and CloudFront distributions, our inner parts receive their IPs on all requests, instead of the original ones (except when using an NLB, then the traffic passes directly through).

## KMS (Key Management Service)

A **regional** service (keys from one region cannot be used on another) for managing Customer Master Keys (CMK). Ideal for encrypting S3 objects, db passwords, API keys (basically env values) stored in the Systems Manager Parameter Store. Pay per API call, supports CloudTrail auditing, FIPS 140-2 Level 2 – evidence of tampering.

When we create a CMK in KMS, the default is a symmetric key (AES-256). AWS services that integrate with KMS only support symmetric keys.

* AWS Owned CMK – irrelevant, really.
* AWS Managed CMK
  + Free, used by default when we pick encryption and used directly by the services.
* Customer Managed CMK
  + Customer creates and manages the keys (rotation, permissions)

Can encrypt and decrypt data up to 4KB in size.

* To encrypt bigger data, we use what is known as envelope encryption with a Data Encryption Key (DEK). Basically, we use the CMK to generate the DEK, and then use the plaintext DEK to encrypt the data. Then, we encrypt the DEK and throw away the plaintext version of the DEK and store the encrypted key along with the encrypted data.
* To decrypt, we call the CMK to decrypt the DEK, and then use the decrypted DEK to decrypt our data

## CloudHSM (Hardware Security Modules)

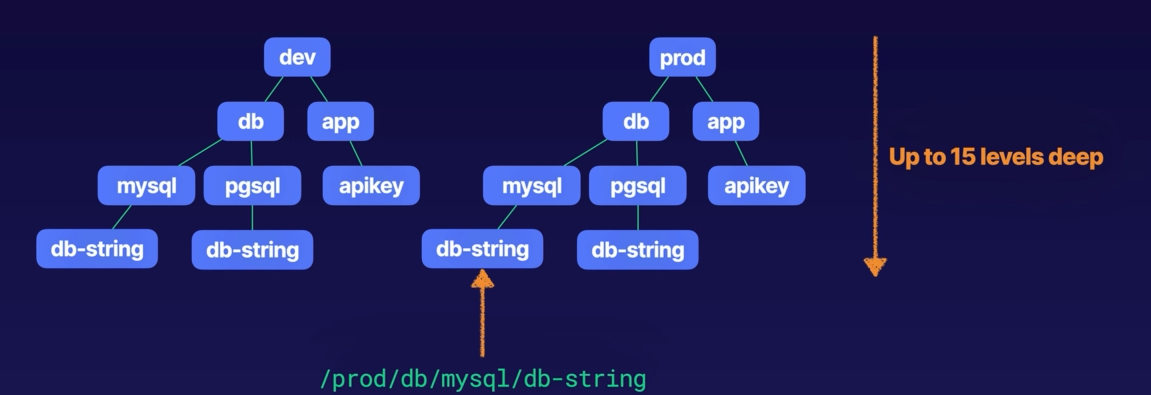
Dedicated hardware security module. Basically, like the shared vs dedicated tenancy for EC2s, but for CMKs. While all (?) CMKs are kept in HSMs, they share those modules with other keys. With this solution, we have our own private module.

Works with industry standard APIs (CNG, JCE, PKCS#11), no AWS proprietary ones. Keys are irretrievable if lost, and there is no HA by default (gotta explicitly provision in different AZs).

## Parameter Store

Component of Systems Manager (SSM). Basically, a serverless place for our env stuff (db connection strings, usernames and passwords, api keys, etc). It can have version control and set TTL for values to ensure rotation.

Parameters are stored in hierarchies. E.g.:

 Each level can have its own special permissions (its like a file system).

## Secrets Manager

Same as Parameter Store, but:

More expensive – charge per secret stored and per 10,000 API calls.

Added functionality – can automatically rotate secrets (out-of-the-box for RDS, can easily make a Lambda function for others). Can generate random secrets. Can be shared across accounts.

## AWS Shield

For D/DoS protection.

* Shield Standard
  + Free and automatically enabled with WAF (for CloudFront or ALB)
  + Protects against common layer 3 & 4 attacks (SYN/UDP floods, Reflection attacks)
* Shield Advanced
  + 3000$/month, per AWS Organisation – business/enterprise support customers receive 24/7 access to the DDoS response team (DRT)
  + DDoS cost protection (if a DDoS attack would have increased your AWS bill)
  + Covers EC2, ELB, CloudFront, Global Accelerator, Route 53

## Web Application Firewall (WAF)

Monitors HTTP/S requests to CloudFront, ALB, or API Gateway.

Can filter based on IP address, Query String Parameters, protects against SQL query injections. If a filter denies access, it returns HTTP 403 Forbidden. WAF can allow all but the specified, deny all but the specified, or count all that trigger specified conditions.

Properties considered – IP address, country, request size, value in request headers, strings in requests that match regex patterns, SQL code, cross-site scripting (XSS).

AWS Firewall Manager – used to centrally configure and manage rules across an AWS Organization. Covers WAF rules, AWS Shield Advanced protections and scans security groups.

# **Serverless**

## Lambda

At the front there will always be an API gateway. Lambda architectures can get very complex and hard to debug, so AWS X-Ray is useful in keeping track of what’s going on.

Pricing - First 1 mil requests are free, afterwards 0.2$ per 1 mil requests. Also, $0.00001667 per GB second (so, $0.25 for a bit more than 4 GB hours).

Lambda triggers:

## SAM (Serverless Application Model)

CloudFormation extension (basically, upgraded CF) optimized for serverless applications. Can define some more things, such as functions (surprise surprise) and test them locally in a Docker container.

## ECS (Elastic Container Service)

Container Orchestration Service (similar to Kubernetes). Creates clusters of EC2 (or Fargate) instances. Works a lot like Kubernetes. Revisit later?

Can be used with a load balancer (of any kind), which can target containers as opposed to servers.

Security

* EC2 Instance Role – all tasks (pods) running on an EC2 instance inherit its role
* Task Role – apply roles on a per-task basis.

Fargate – serverless container engine – it creates containers as opposed to EC2 instances.

EKS – managed Kubernetes service. Manages the control plane (API server, etcd), automatically monitors and replaces unhealthy nodes, handles patching and upgrading. Basically, cuts management overhead at the cost of lacking some features and being behind in version.

ECR – Container Registry, where we can store, manage, and deploy Docker images. Integrates with IAM, pay for Storage and Data transfer.

If services are to interact (esp true for compute services) with other services, they should have the relevant IAM roles that allow them to do so (e.g. to put new objects in S3).

Synchronous triggers – ALB, Cognito, Lex, Alexa, API Gateway, CloudFront, Kinesis Data Firehose

Asynchronous triggers – S3,

File Gateway???