# Learning Plan

1. Udemy course: <https://www.udemy.com/course/amazon-web-services-aws/learn/lecture/5813770#overview>
2. Front-end course: <https://frontendmasters.com/courses/aws-frontend-react/>
3. Front-end course on lambda (frontendmasters)

# Terminology

Excellent fundamentals and core concepts site: <https://aws.amazon.com/getting-started/fundamentals-core-concepts>

* Application Load Balancer (ALB) – a single point of contact for your app that direct traffic to your different servers, depending on the load, their health etc.
* Amazon Resource Name (ARN) – unique identifier for AWS resources. Examples: IAM policies, RDS tags and API endpoints. The usual format for ARN is:  
  arn:{partition}:{service}:{region}:{account-id}:{optional: resource-type}:{resource-id} for example:  
  arn:aws:iam:us-east-1:123456789012:my\_SQS
* VPC – your own private data center in the cloud
* CDK – AWS Cloud Development Kit – infrastructure as code - a SW development framework for defining cloud infrastructure in code and provisioning it through AWS ClodFormation.
* SaaS – Software as a Service
* IaaS – Infrastructure as a Service
* Iaac – Infrastucture as Code
* Scaling – healthy server utilization is between 50-90%.
* EC2 – Amazon Server – where our back-end program runs
* S3 – Amazon storage (for storing files, images etc)

# AWS

In Amazon: create a burner account for training: access.amazon.com

<https://access.amazon.com/aws/burner/get_console_access_url?account_id=043163097029>

1. EC2 -> Security Group -> click default (default VPC security group) -> Edit inbound Rule
2. select ssh, then click custom and change to my IP

# Common Tasks

## Create server-based app

* 1. Create a new EC2 instance.   
     usually, use Amazon Linux (comes with lots of useful service) for OS.
     1. You can configure your EC2 auto-scaling group to auto-scale your resources up/down.
     2. Add EBS (SSD on server) for storing data on the server

# Security

AWS works in the model of **ZERO TRUST** which means that every entity is considered malicious and is protected against. This includes also all the sw that runs on your service, all agents that can access your resources and even the network fabric itself.

* IAM – Uses the principle of least privilege to manage **resources**, **principles** (entities) and **actions** privileges.
* VPC (Virtual Private Cloud) for Network Security –
  + Subnet – the range of IP addresses within your VPC
  + Route tables: a set of rules that determine where traffic is directed
  + Internet gateway – the component that allow communication between the internet and your internal network.
  + Web Application Firewall (WAF) – VPC firewall.
* Security Groups – a resources’ firewall that can control access to your resources based on accessed ip or source.
* Encryption
  + In transit: using HTTPS endpoints and ALB (Application Load Balancer) to enforce HTTPS connections to all your endpoints.
  + At rest: usually supported by default at no additional change and with negligible performance hit.
    - Key Management Service (KMS) – key management system to create and manage Customer Managed Keys (CMK) to encrypt your data. This gives you access to additional AWS services for enforcing keys rotations, auditing keys usage etc.

# Identity Access Management (IAM)

## Terms

* Users – End users (people)
* Groups – a collection of users under one set of permissions. For example: HR, Marketing etc. each of them requiring different types of permissions.
* Roles – you create roles (e.g. S3-ReadOnlyAccess) and then you can assign them to users or other AWS resources (e.g. EC2 ).
  + Roles are controlled by policies
  + They are much more secure than using (secret) access keys.
  + You can change a policy on a role and it will take immediate effect
  + You can attach and detach roles to running EC2 instances without having to stop and restart the instances.
* Policies – A document that defines one (or more) permissions. The policy can then be attached to a user/group/role. Readable json format.

## What is IAM

Allow you to manage users and their level of access to the AWS console.

IAM gives you:

* Centralized control of your AWS account
* Shared access to your AWS account
* Granular Permissions
* Identity Federation (allowing users to login to your site using their Active Directory, Facebook or LinkedIn etc accounts)
* Multifactor Authentication
* Provides temporary access for users/devices and services, as necessary.  
  For example, allowing users of your app to retrieve data from your S2 database.
* Allows you to set up you own password rotations policy
* Integrated with many AWS services
* Supports PCI DSS Compliance for applications associated with the payments industry.
* IAM is universal. It does not apply to specific regions.

## IAM Lab

1. Create a new AWS account: <https://aws.amazon.com/premiumsupport/knowledge-center/create-and-activate-aws-account/>
2. Create user accounts. Use your root account only when you absolutely don’t have another choice. For everything else – use a user account.  
   NOTE: When you create the account, this is the only time that you can see the user’s Secret Access Key which is required to run CLI commands. So, Download the .csv and keep it safe. Otherwise, you won’t be able to access this account anymore!! Or even better, **save it in keypass!!**
   1. User & paswd – required for logging into the Console
   2. Access Key ID & Secret Access Key – required for using the CLI
   3. New users don’t have any permissions. You need to give them permissions.
   4. Notes:
      1. If you lost your access (secret) key, you can generate a new one and then you’ll need to re-run >> aws configure to use the cli again
      2. Never save the (secret) access keys in code!!
      3. Never share the same access keys between multiple developers.
      4. Do NOT USE access (secret) keys to control the access to your services! It is too insecure and risky. Instead, **use roles** to control them (see below)
3. Create Roles: for example: if we want to allow EC2 instances to write to our database (S3 buckets).
   1. After creating the role, you can go to the EC2 instance and choose Action-> Instance setting -> attach role to connect the role to the target.

# Principals

<https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements_principal.html>

When using resource-specific or IAM-roles policies (not IAM-identity based policy), you do this by defining the principals that have access to this resource. The principals can be:

* AWS account and root user
* IAM Users
* Federated users
* AWS services
* Etc.

Permissions language:

* arn:*partition*:*service*:*region*:*account*:*resource*

# Elastic Compute Cloud - EC2

* Secure, resizable compute capacity in the cloud.
* Like a virtual machine in the cloud (hosted in AWS)
* Allow super-easy scaling
* The capacity you want when you need it
* You have complete access to your instances (root)
* Pay only for what you use, when you use it.
* You can select your capacity and then grow/shrink it as you need.
* Pricing models:
  + On demand – you pay by the hour/second depending on the type of instance you run
    - Good for application that are short-term, spiky or unpredictable workloads that cannot be interrupted.
    - For testing the waters – new application in development
  + Reserved Instance (RI) – you reserve the capacity for yourself for 1-3 years. This is regional and allow you to get up to 72% discount (as long as you’re in the same region)
    - Suitable when you know in advance that you need capacity X long term (years)
    - Require you to pay up front
    - Standard RI – a static capacity (you can’t change it) – up to 72% off
    - Convertible RIs – you can change the capacity for equal/greater value – up to 54%
    - Scheduled RIs – when you don’t need the capacity all the time but instead only X days every month so you can pay only for these re-occuring times.
  + Spot – Purchase unused capacity at discounts of up to 90% (fluctuate according to supply and demand). Once the capacity drop below your ask or the price above it, you will use your instance.
    - Suitable when you need to run something occasionally and you can spin it up from scratch easily (no long installations etc needed) and when it’s not time-critical.
  + Dedicated – a physical EC2 server dedicated for your use. This is the most expensive option.
    - This is suitable for when you have licenses that are tied to a physical HW or you have some regulations that forces you to use this option.

## EC2 Instance Types

* The HW of the underlying server
* Capabilities: different compute, memory, storage etc
* Different instance types are optimised for different types of applications
* Provisioned in availability zones

## EC2 Lab

* Compute->EC2->scroll to Launch instance-> Select Amazon Linux 2 AMI
* You can filter the instances types
* The ‘Type’ is the instance names. The number after the letter (e.g. t2.nano) is the generation. In this case, the 2nd generation of the nano isntances.
* Greyed out options mean that they are not supported for our choosed OS
* Main types:
  + Micro instances are a low-cost option for very small/low throughput applications.
  + Compute optimized – high-processing applications that don’t need that much storage
  + FPGA – for processing on HW. For highly paralalised processing on HW
  + GPU – for graphics and apps that needs parallel processing
  + Machine learning – use custom-built ASIC for machine learning
  + Memory optimised – a lot of memory. Database applications, distributed caches etc.
  + Storage optimized
* Configure Instance Details – important notes:
  + You can select Spot instance and set the price limit
  + Austo-assign public IP – the public IP to use in order to access your app
  + Shutdown behaviour – can choose either Stop or Terminate
  + Enable termination protection – very important to production systems so that they can be accidently terminated.
  + Monitoring – CloudWatch monitor your services at 5 minutes intervals bey default for free. If you want a more detailed monitoring in higher resolution (upto 1 min), you need to set it up and pay extra.
  + Advanced Details – User data – this is where you can add bootstrap batch files to run whenever the system boots up. We can use this for installing updates/applications etc.
  + Add storage – by default you get one EBS volume on which the linux will be installed.
    - Volume type – for most applications you should choose the default (General Purpose SSD (gp2)).
    - Encryption – you can encrypt your volume with KMS when you create it.
    - You can add additional Volumes. For example, if you need to have your application on a different volume that the OS.
  + Tags – completely user-defined. It’s a great way to organise your EC2s and other resources. You can add ‘name’ , ‘team’ etc.
  + Security Groups – a virtual firewall
    - SSH – allow to access it with SSH
    - HTTP – for web apps. Port should be 80
    - Source:
      * For HTTP – should be anywhere so that everyone can access my site. For SSH – should be ‘My IP’ or ‘Custom IP’
      * 0.0.0.0/0 – means all IPV4 addresses will have access ; ::/0 – means all IPV6 addresses will have access.
  + Select an existing key pair or create a new key:  
    Create a new key pair and download it. This will create a private-public key pair that will be used for accessing your EC2 instance. Only users that have the private key will be able to access the EC2 instance.
    - After the ke y-pair downloads, change its permissions:  
      >> chmod 400 my-key-pair.pem  
      otherwise, it won’t work
  + Launch your EC2 instance

To ssh into your EC2 instance:

ssh ec2-user@{Public IPV4} -i {private-key file}  
for example:  
ssh ec2-user@ 3.25.146.91 -i MyTrainingKeyPair.pem

* + Notes:
    - Use the IPV4 and not the DNS because the DNS will not always work!!
    - in order to ssh from a windows machine, use’ll need to use Putty and PuttyKeyGen to convert your .pem key to .ppk file.
* Move to su account:  
  >> sudo su
* Install all latest OS updates:  
  >> yum update -y
* Install apache:  
  >> yum install httpd -y
* Start apache:  
  >> systemctl start httpd
* Start apache automatically on boot:  
  >> systemctl enable httpd
* Verify that it’s running:   
  >> systemctl status httpd
* Create a minimal website:

1. Browse to /var/www/html
2. Create a minimal index.html file with nano editor:  
   >> nano index.html
3. Now you should be able to access your website using the ‘IPV4 Public IP’ on your instance’s Description page (On your EC2 Dashboard->Volumes->choose your volume)

## Elastic Block Store - EBS

Storage volumes that you can attach to your EC2 instances (like a hard disk). You will use them as you would any physical HD.

* Mission Critical. EBS are:
  + Designed for production workloads
  + High Availability: redundancy: automatically replicated within a single availability zone to protect against HW failures.
  + Scalable: dynamically increate capability and change the type volume with no downtime or performance impact to your live system.
* Types of EBS:
  + **IOPS** – IO operations per second. Important for quick database transactions, low latency apps.
  + General Purpose SSD (gp2) – a balance of price and performance.
    - good for boot volumes or development and test applications which are not latency sensitive.
    - Max 16,000 IOPS per volume.
  + Provisioned IOPS SSD (io1)
    - The highest performance and the most expensive one
    - If you need more then 16,000 IOPS
    - Up to 64,000 IOPS per volume
    - IO intensive applications, large databases
    - Suitable for OLTP (Online Transaction Processing) and Latency-sensitive applications.
  + Throughput Optimized HDD (st1):
    - Low cost HDD volume
    - Max **throughput** 500 MB/s per volume. Important for large datasets, large IO sizes, complex queries. Effec the ability to deal with large datasets.
    - When you need to store huge amounts of data and access it frequently, throughput-intensive workloads
    - Big Data, data warehouses, ETL, log processing
    - Cannot be a boot volume
  + Cold HDD (sc1)
    - Lowest cost option
      * Max throughput 250 MB/s per volume
      * A good choice for colder data requiring fewer scans per day
      * Where performance is not a factor
      * Cannot be a boot volume.
* To create a new volume, we can create it from the EC2 Dashboard:
  + Notice the availability zone of your EC2 instance since it has to be the same for your volume
  + Snapshot ID – allows you to use existing snapshot (image). If you choose a snapshot, it will also copy the same encryption.
  + To attach, choose the volume and in actions-> Choose attach

## EC2 CLI

<https://docs.aws.amazon.com/cli/latest/reference/s3/index.html>

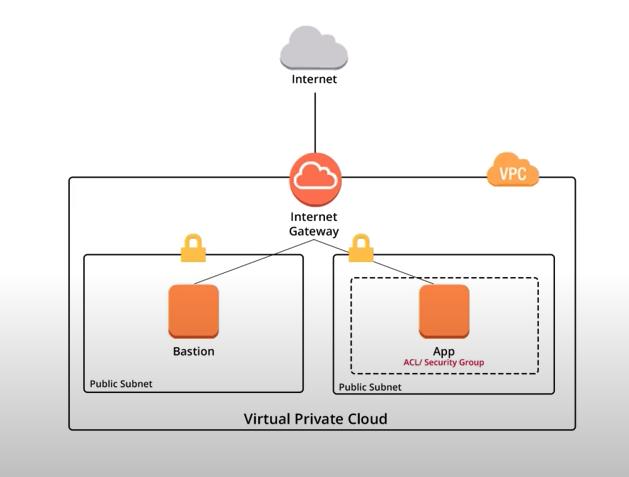
In SSH session:

>> aws configure

And copy the Access Key ID and the Sercret access key from the user’s details.

* Clean your configuration on the EC2:  
  >> cd ~/.aws  
  >> rm config  
  >> rm credentials
* Create an s3 bucket:  
  >> aws s3 mb s3://{my bucket name} [optional: --region {region name}]  
  for example: >> aws s3 mb s3://my-training-bucket1234-rk –region us-west-1
* See your s3 buckets:  
  >> aws s3 ls
* See all the files in your bucket:  
   >> aws s3 ls {bucket name}
* Move files into your s3 bucket:  
  >> aws s3 cp {file name} {bucket name}  
  for example:  
  >> aws s3 cp hello.txt s3://my-training-bucket1234-rk

# Virtual Private Cloud (VPC)



# Elastic Load Balancers

* Allow us to balance the load across different servers.
* Application Load Balancer – works in the OSI Level 7 (application level) – http/https level. Can make very clever decisions according to the application layer data. So for example, they can route the traffic to different servers based on the sub-domains of the calls (e.g. all site/marketing/… URLs will be routed to the marketing servers etc).
* Network Load Balancer – works in OSI Level 4 (transport – TCP/UDP) – optimised to be super-fast (extreme performance). This is what you would usually use in production systems and when latency is critical.
* Classic Load Balancer – they are legacy only and not recommended for new applications. You can load balance with it both layer 4 or layer 7 (x-forwarded and sticky sessions). It’s not as good as the new load balancers.

## Classic Load Balancer

* Classic Load Balancer Error – if your application stops responding, the ELB responds with a 504 error (gateway timeout):
  + This is not the ELB failing but a failing of the application
  + To troubleshoot it -> troubleshoot the application
  + This could be either at the Web Server Layer or at the Database Layer.
  + Identify where the application is failing and scale it up or out where possible
* X-Forwarded-For Header:  
  

Your application will not see the user’s IP. Instead, it will only see the ELB private IP (e.g. 10.0.0.23). In order to see the end user’s IPV4 look for the X-Forwarded-For header.

## Route 53

Route53 is Amazon’s DNS Service

It allows us to map our domain names to:

* EC2 instances
* Load Balancers
* S3 Buckets

## Lab

1. Services->route 53->DNS Management
2. If you don’t have a domain, you can register (buy) a domain there. Once your domain is registered, you can route it.
3. Create a load balancer:
   1. Go to your EC2 dashboard -> Load Balancers -> Create Load Balancer
   2. Choose application load balancer
   3. Listeners – on which ports will the load balancer listen. Can choose http, https etc.
   4. Availability Zones – choose all for maximum availability
   5. Configure Security Groups – you can choose the security group we created before.
   6. Configure Routing – create/choose your target group – what will be the target to which the routing rule will apply (when you configure the route 53 mapping)
      1. Target type – what will be the type of the target: Instance (EC2), IP or Lambda Function
      2. Which protocol and port to use for the target
      3. Health checks – only route if these checks are ok (?)
   7. Register Targets: choose your target (e.g. EC2 instance)
   8. Wait until you see that the:
      1. Load Balancer State is healthy
      2. In your EC2 -> Status Checks, the status is healthy (what we defined in the target group).
4. Go to your Route 53 dashboard -> hosted zones -> mark your domain -> Go to records set. This is where we create our DNS records
   1. A record: map a DNS
   2. Create Record Set and choose:
      1. Name: leave is as naked domain name (also called Apex Record which means that it doesn’t have the www. Prefix)
      2. Type: A-IPV4 address
      3. Alias: Yes. Choose your EC2 instance
   3. Now, when you browse to your new domain name (e.g. iamacloudguru.com), you’ll be automatically routed to your EC2 instance.

# Databases

## Terms

### Data Warehousing

Used for business intelligence.

Tools like Cognos, Jaspersoft, SQL Server Reporting Services, Oracle Hyperion, and SAP NetWeaver.

Used to pull in very large and complex data sets. Usually used by management to do complex queries on data (such as current performance vs targets etc).

### OLTP vs OLAP

* Online Transaction Processing (OLTP) – simple transactions that happens very frequently. For example: get/write an order data into a database.
* Online Analytics Processing (OLAP) – complex queries that happen very infrequently. For example, the net profit for some sort of product. This will need pulling a lot of different data and doing calculations on it.

### ElastiCache

A web service that makes it easy to deploy, operate and scale an in-memory cache in the cloud. The service improves the performance of web applications by allowing you to retrieve information from fast, managed, in-memory caches, instead of relying entirely on slower disk-based databases.

ElastiCache supports 2 open-source in-memory caching engines:

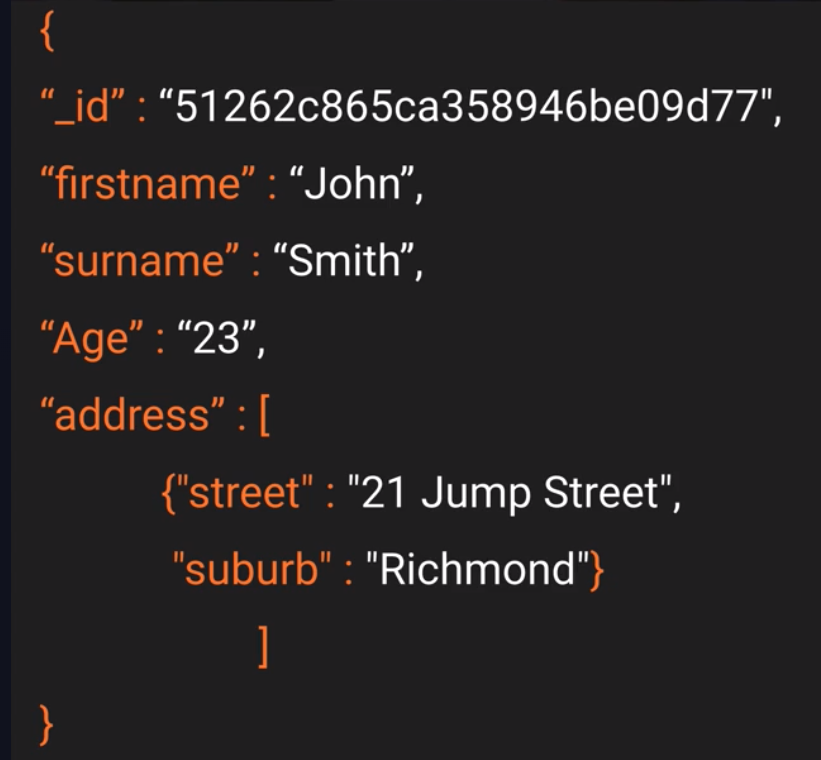
* Memcached
* Redis

## Relational Database Service (RDS) - OLTP

* Think spreadsheets:
  + Database with
  + Tables each table has
  + Rows and
  + Fields (columns)
* RDS Types in Amazon:
  + SQL Server
  + Oracle
  + MySQL Server
  + PostgeSQL
  + Aurora – Amazon flagship SQL database
  + MariaDB

## Non Relational Database

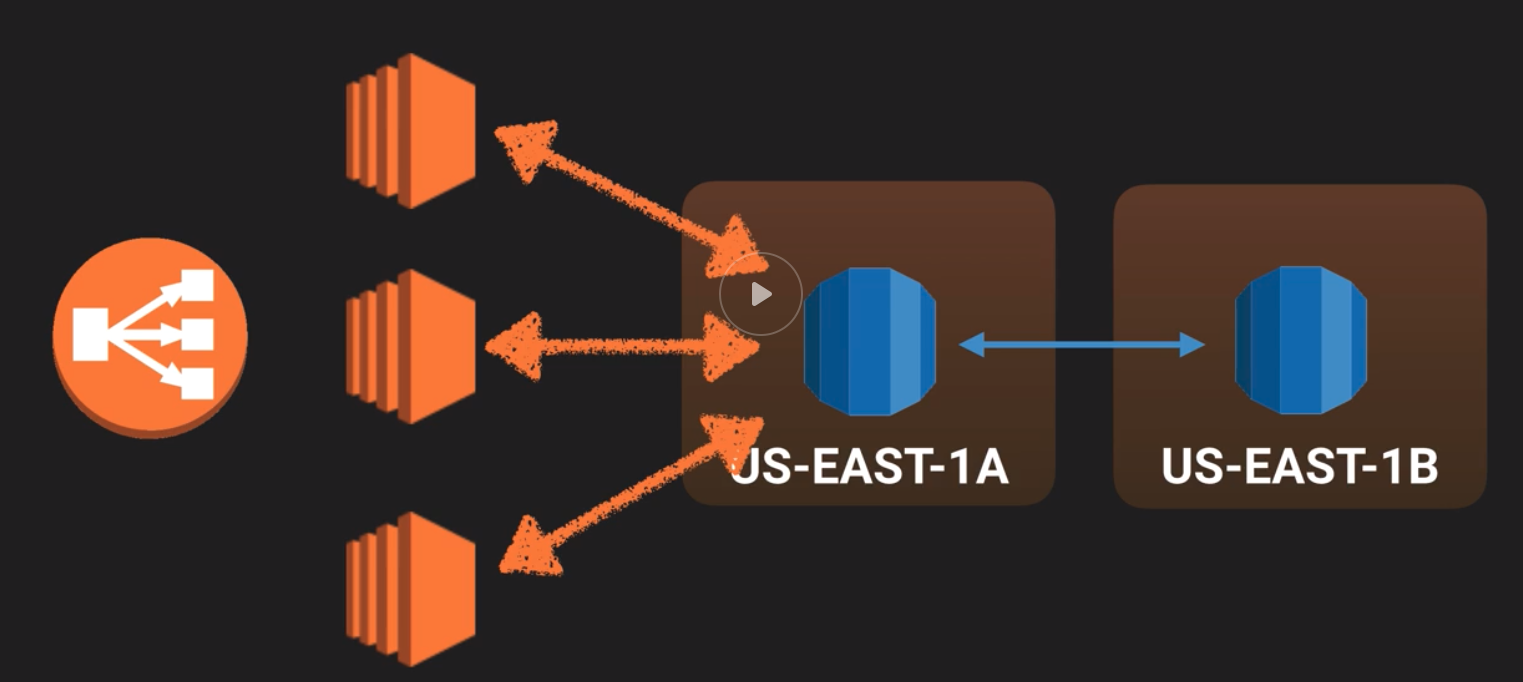
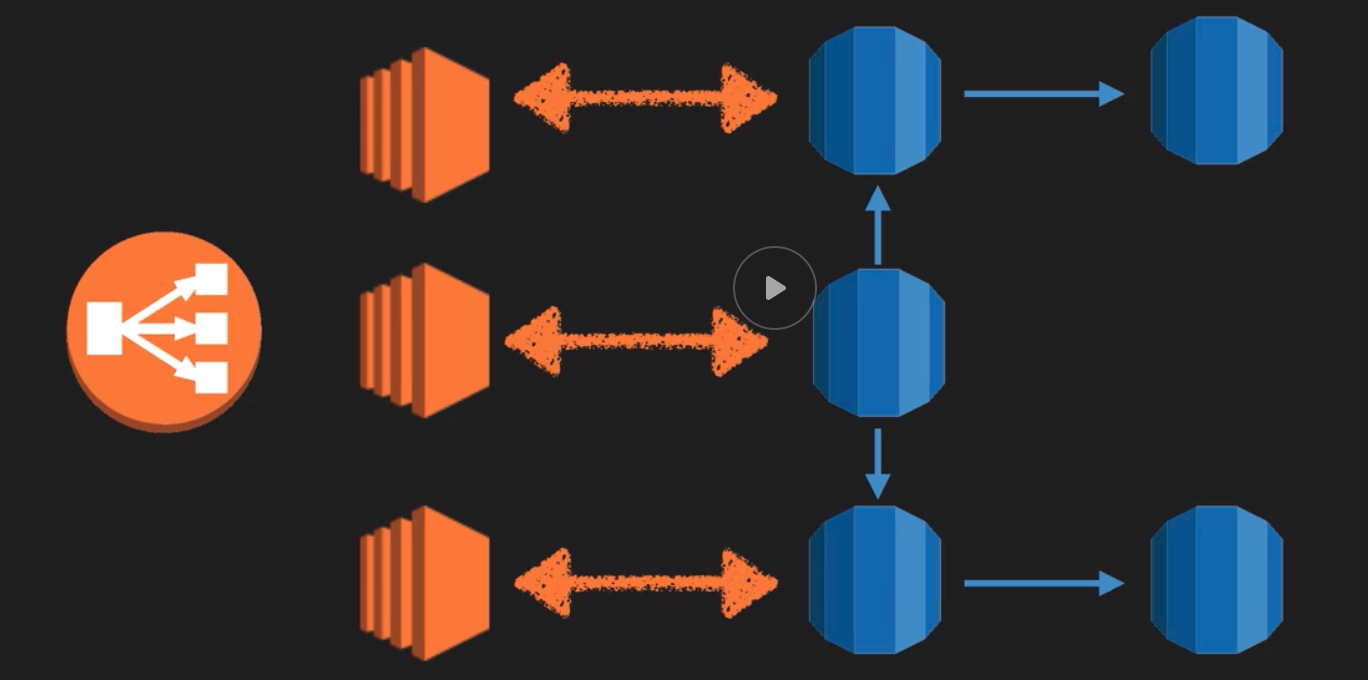
Contains:

* Database with
  + Collections (=tables)
  + Documents (=rows) with
  + Key Value Pairs (=fields)
* Advantages over relational databases:
  + You do not need to define your fields (columns) in advance like in the RDs
* Examples:
  + JSON/NoSQL:  
    
* In Amazon: DynamoDB – No SQL

## Redshift – OLAP

* When your need to improve your database performance - if management is running OLAP transactions on it, consider changing your database to be Redshift to have better performance for OLAP.
* For data warehouses

## Backups, Multi-AZ & Read Replicas

* There's two types of backups for AWS:
  + **Automated Backups** will allow you to recover your database to any point in time, within a retention period. The retention period can be between 1-35 days.  
     Automated backups will basically take a full daily snapshot and will also store transaction logs throughout the day. And then when you recover, AWS will first choose the most recent daily backup and then apply those transaction logs that are relevant to that day. This basically allows you to do a point in time recovery down to a second, within the retention period window.
    - Automated backups are enabled by default. The backup data is stored in an S3 bucket and you get free storage space equal to the size of your database.
    - Backups are taken within a defined window. During the backup time, storage IO may be suspended so you may experience some elevated latency.
  + **Snapshots** – they are a manually created (use command) snapshots of the DB that are saved in a stand-alone file. They can be saved after we delete the RDS itself.
* **Restoring Backups** – for both automatic backups and snapshots, always create a completely new RDS with a new endpoint:  
  ****
* **Encryption:** Encryption at rest is supported on all supported database types. It is done using the AWS Key Management Service (KMS). Once your RDS instance is encrypted, the data stored at rest in the underlying storage is encrypted as are its automated backups, read replicas and snapshots.
  + At the moment encrypting an existing SB instance is not supported. To work around this, you can create a snapshot, make a copy of it and encrypt the copy.
* **Multi-AZ** (Multi Availability Zone) – Synchronous Disaster Recovery (not for performance improvement) - Multi-AZ allows you to have an exact copy of your production database in another availability zone. AWS handles the replication for you, so when your production database is written to, this write will automatically be synchronized to the stand-by database.  
  In the event of planned database maintenance, DB instance failure or an Availability Zone failure, Amazon RDS will automatically failover to the standby DB so that the database operations can resume quickly without administrative intervention.
  + If you want Multi-AZ support, you need to turn it on in the RDS configuration. Aurora have this turned on by default.  
      
    ****
* **Read Replica** – Asynchronous read-only copies of your DB (supports upto 5 read replicas) that can be used for scaling your service. The read replicas can be in the same or in different availability zones and/or regions.
  + The primary RDS replicates to the replicas asynchronously
  + You would want to use read replicas to improve the performance of very read-heavy workloads.
  + Are not supported for MySql
  + You can have read replicas of read replicas (watch out for latency)
  + Each replica will have its own DNS end point
  + You can have read replicas that have Multi-AZ
  + Read replicas can be promoted to RDS. This breaks the replication.
  + To create a read replica: in your RDS dashboard: Actions->Create read replica  
      
    ****

## Elasticache

Elisticache is a web service that makes it easy to deploy, operate and scale an in-memory cache in the cloud to improve the service’s performance.

When your service is under a lot of stress/load and you need to improve its performance. If your database is particularly read-heavy and not prone to frequent changing, Elasticache can help.

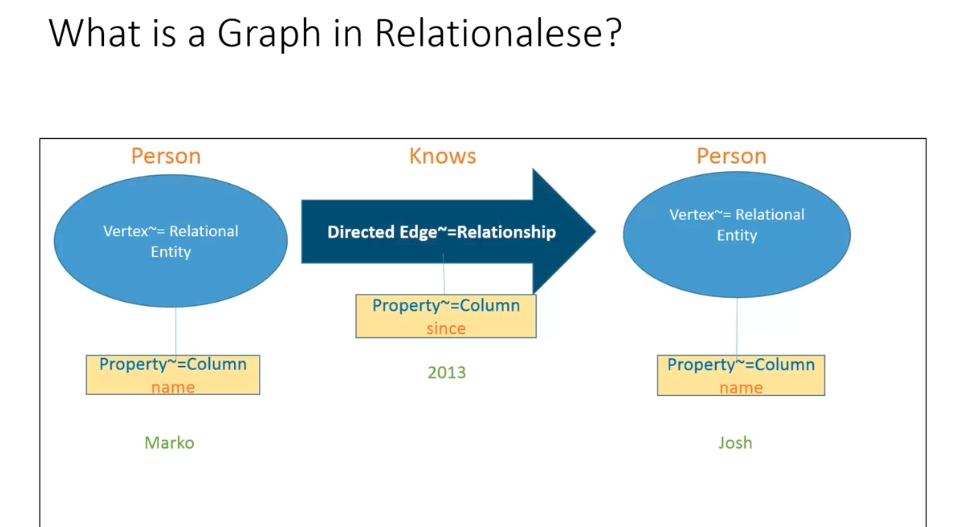
* Automatically save your queries and data in the cache so you don’t need to query the production database or to re-calculate common values every time.
* Types of caches supported by Elasticache:
  + **Memcached:**  a widely used memory object caching system.
    - Doesn’t have persistence
    - Elasticache manages Memcached nodes as a pool that can grow and shrink, similar to an Amazon EC2 Auto Scaling Group.
    - Individual nodes are expendable and Elastcache provides additional capabilities such as automatic node replacement and auto discovery.
    - Use Memcached if:
      * Object caching is your primary goal
      * Simple caching model
      * Planning to run large cache nodes and require multithreaded performance with utilization of multiple cores
      * Ability to scale your cache horizontally as you grow.
  + **Redis:** a popular open-source in-memory key-value store that supports data structures like sorted sets and lists.
    - **Elasticache**  supports master/Slave replication and Multi-AZ which can be used to achiever cross AZ redundancy.
    - Elasticache manages Redis more as a relational database. Redis Elasticache clusters are managed as stateful entities that include failover, similar to how Amazon RDS manages database failover.
    - Use Redis if:
      * Need more advanced data types such as lists, hashes and sets
      * Has sorting and ranking datasets in memory (e.g. leaderboards)
      * Persistent
      * Want to run in multiple AWS availability zones with failover.
  + If you want to have Multi-AZ redundancy -> use Redis.   
    Otherwise -> use Memcached.

## Lab

* 1. Create My SQL database
     1. DB instance identifier - DB name
     2. Set the user name and password
     3. Notice your database port
     4. Create a new security group for the DB
     5. Additional config:
        1. Initial database name – the same one you put in instance identifier
     6. Note: after your RDS is created, you’ll find it’s endpoint in its dashboard under ‘Connectivity & security.
  2. In your EC2 add the bash boot script to get the connect.php script and restart your EC2.
  3. Ssh into your EC2 and update your connect.php script with the right database data. Note: $hostname is the DB endpoint.
  4. In your RDS Security group, add another inbound rule to allow your EC2’s to access it, so eventually it should have the following rules:
     1. Custom TCP – TCP – 3306 – {your IP}
     2. MUSQL/Aurora – TCP – 3306 – {your EC2 Security Group}
  5. Go to your web browser to: “{your EC2 IP address}/connect.php” you should now see a message saying that it’s now connected to MySQL with all the parameters

## Naptune

* Fully managed graph database built on top of Oscar and Grover (the storage for Aurora)
* Supports Apache Tinkertop framework (java framework) and Gremilin (graph traversile language)
* Sparql compatible



The vertexes are independent objects in the database. They are not stored in a ‘Persons’ table.

Relationships are 1st order objects. For example: Knows relationship.

Graph databases are not appropriate for every application and evey database.

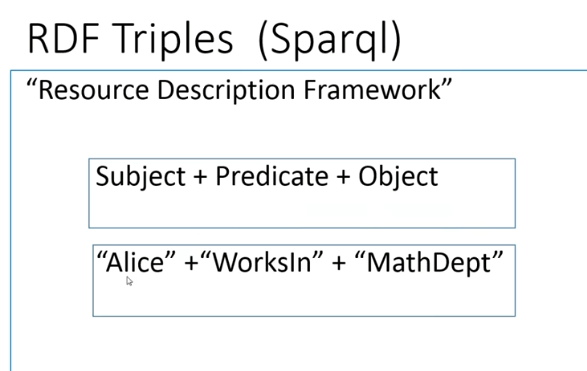
Directed graph is ideal when you have:

* Loosely structured data with
* Many-to-many relationship

Notes:

* Cardinality – how many properties of the same name (key) can be for a single vertex? The options in Gremlin are:
  + Single
  + List
  + Set – this is the default in Neptune which permits multiple values for every key!!!

### RDF (Resource-Description-Framwork):

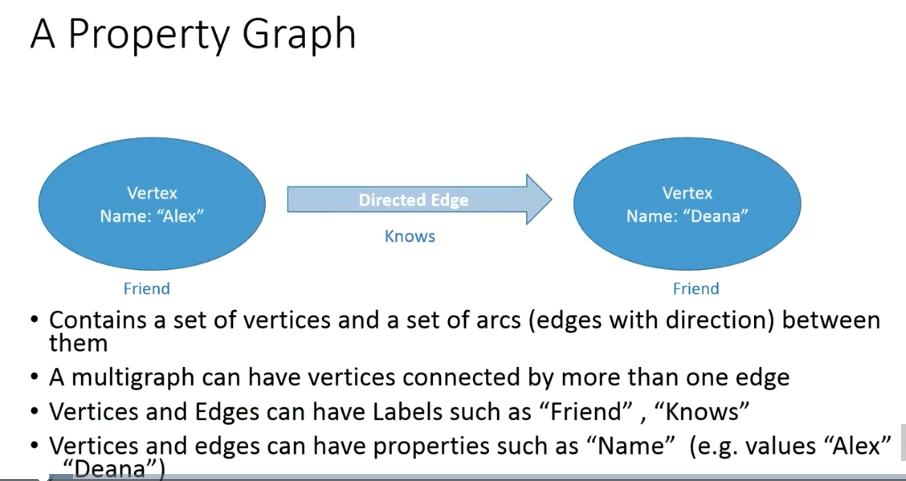


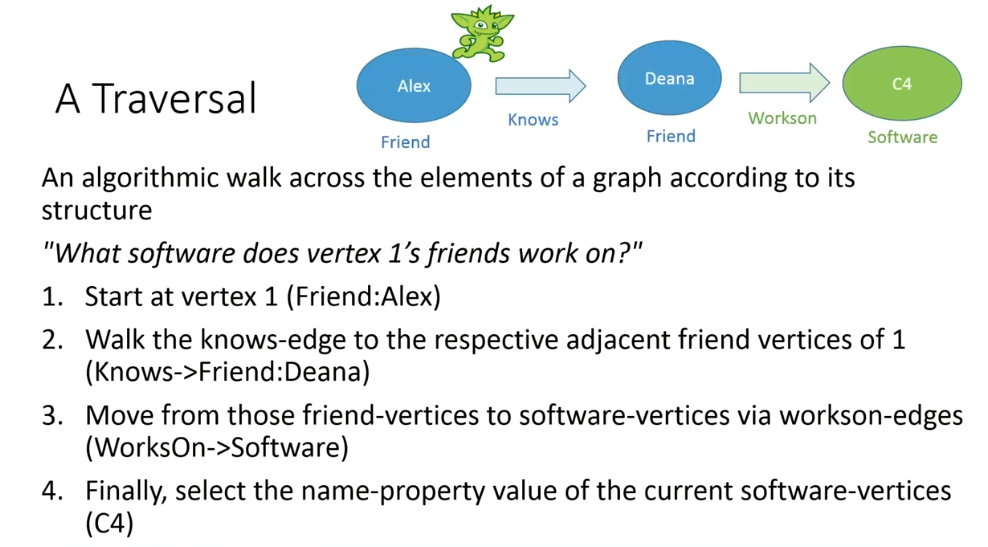
Every triples are the data units. So, there is no ‘Alice’ object. There is just a ‘Alice WorksIn MathDep’ object.

### Gremlin

#### Apache TinkerPop

* An abstraction layer over different graph databases and different graph processors
* Structure API “Property Graph”: a directed, attributed multi-graph that supports labels and key-value properties. Accesses functions such as transactions.
* Process API “TraversalSource”: Traversal<S,E>: functional data flow process transforming objects of type S into object type E.
* See link to Tinkerpop tutorial in the NaptuneBeta wiki.





### Gremlin

* The graph traversal language for Tinkerpop
* Wrapper for Apache Groovy Java

#### “Modern” Graph and Tinkergraph

* In-memory implementation offered by Tinkerpop
* From local gremlin console:
  + gremlin> graph=TinkerFactory.createModern()  
    greamlin> g=graph.traversal()  
    gremlin> g.V().count()  
    🡺 tinkergraph[vertices:6 edges:6]
* “graph” and “g” variables are predefined in Neptune
* Neptune format “modern” graph in Neptune format can be loaded to Neptune from: S3://com-deanah-neptune/gremlin

#### Transactions

* Tinkerpop provides a Transaction interface. Transactions can be implemented as AUTO or MANUAL.
* Neptune implements AUTO transactions (i.e. auto-commit and rollback)
* A REST call is an atomic transaction. You can bundle a collection of operations in a single REST call and they will succeed or fail together.  
  note: this might have changed!!

### Useful Gremln Commands

To connect to the database:

* ssh to the endpoint and start gremlin:

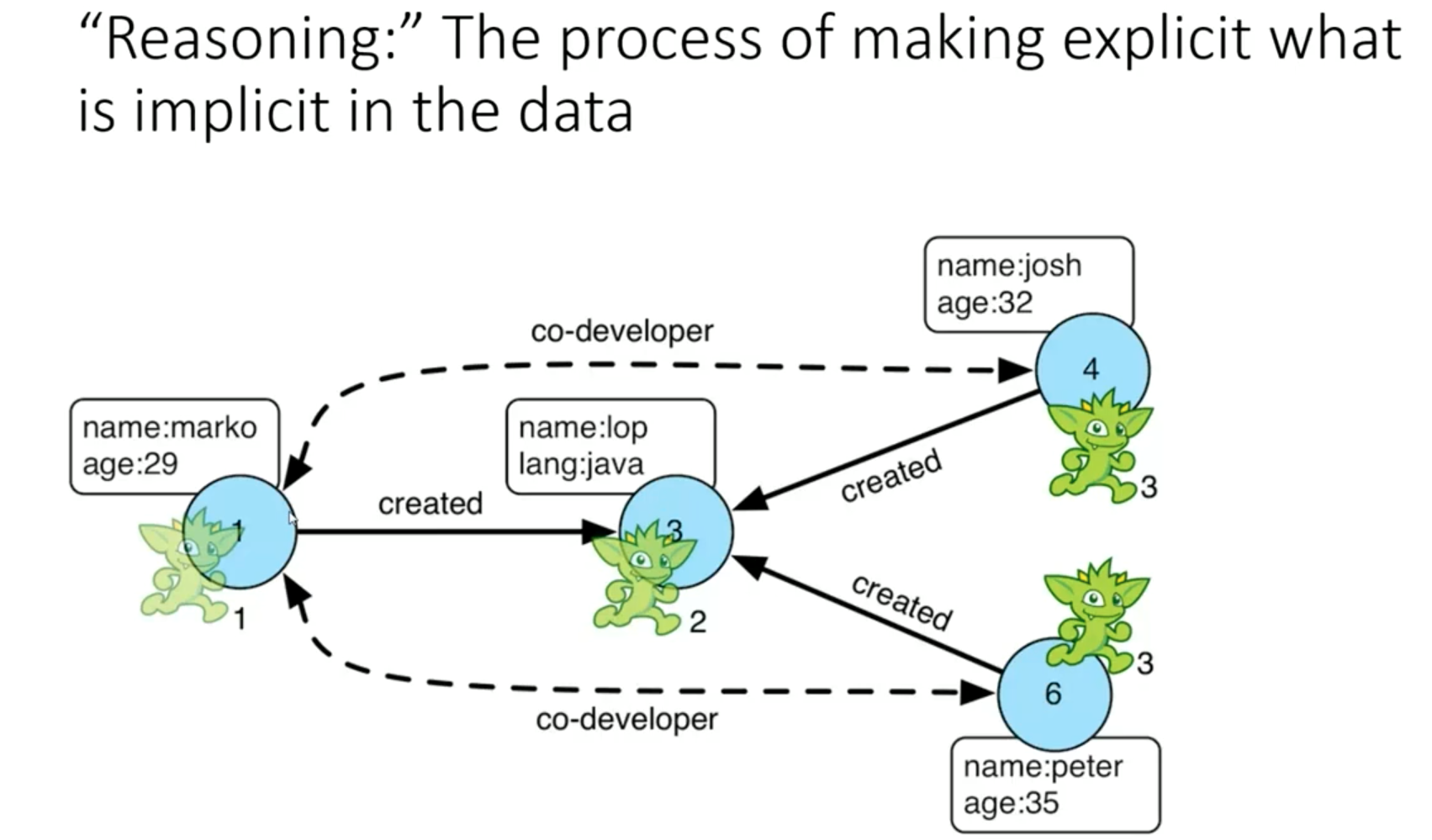
ssh -i "ssh-amparry-key-pair.pem" [ec2-user@35.165.93.172](mailto:ec2-user@35.165.93.172)

cd neptune/gremlin/console/apache-tinkerpop-gremlin-console-3.4.3

bin/gremlin.sh

:remote connect tinkerpop.server conf/neptune-remote.yaml

:remote console

* g.V().count() , g.E().count()
* g.V().drop() – empty your DB
* g.V() , g.E()
* g.V().properties() , g.E().properties()
* g.V('3').properties()
* g.V('3').outE('knows')
* g.V(‘3’).outE(‘knows’).inV().values(‘name’)
* g.V(‘3’).outE(‘knows’).values(‘name’)
* g.V(‘3’).out(‘knows’).has(‘age’, gt(30)).values(‘name’)  
  🡺 josh
* g.V(‘3).out() – all the adjacent vertexes to V[3]
* g.V().has(‘name’, ‘marko’).out().values(‘name’) – the names of all adjacent vertexes of a vertex with the name ‘marko’
* g.V().has(‘name’, ‘marko’).out().out().values(‘name’) – the names of all adjacent vertexes of all the adjacent vertexes of a vertex with the name ‘marko’ (friends of friends)  
  we can do the same with a path step that remembers the history of the traversal:
  + g.V().has(‘name’,’marko’).repeat(out()).times(2).path().by(‘name’)  
    so this will return the pull path from marko to each of his friends friends
* g.V().has(‘name’,’marko’).as(‘marko’).out(‘created’).in(‘created’).where(neq(‘marko’)).values(‘name’) – return the names of all the vertexes that created the same sw as marko but is not called marko.
* 
  + So in the last example: creating a ‘co-developer’ edge will be reasoning:
  + g.V().has(‘name’,’marko’).as(‘marko’).out(‘created’).in(‘created’).where(neq(‘marko’)).addE(‘co-creator’).from(‘marko’).property(‘joinedteam’,’2017’)  
    this will create all the co-developer edges for marko
* g.V().has(‘name’,’stephen’).property(single,’age’,18)  
  Will replace the property ‘age’ with a new one with a value of 18. Note that since Neptune’s default cardinality is set. If we didn’t specify it in the step, it will add another ‘age’ property with a different value instead.
* a step modulator – add a feature to a step. Examples:
  + by() – filter the property
  + as() – gives an alias to the output of a step.

### Lab

* + 1. Google ‘redshift tcp timeout’ and choose ‘Troubleshooting Connection Issues in AmazonRedshift” – copy the command from the ‘Change TCP/IP Timeout Settings’
    2. To install console, go to tinkerpop for instructions on installing the gremlin console.

# Simple Storage Service - S3

<https://aws.amazon.com/s3/faqs/>

* Secure, durable, highly scalable object storage (not for OS/database). For files, images etc.
  + Built for 99.99% availability. Amazon guarantee 99.9% availability
  + Amazon guarantee 99.999999999% durability for S3 information (11x9s) – the amount of data you expect to lose in a single year. We want the durability to be as close to 100% as possible.  
    note: you should still backup your data.
* Easy to use, with a simple web services interface to store and retrieve any amount of data from anywhere on the web.
* Already includes redundancy. Spread over multiple devices and facilities and availability zones
* Allows you to upload files 0-5GB
* There is unlimited storage on S3. You don’t need to allocate or plan in advance.
* Files are stored in Buckets (similar to folder)
* S3 is a universal namespace. That is, bucket names must be unique globally. Simlar to DNS. For example:  
  <https://s3-eu-west-1.amazonaws.com/acloudguru>
* When you upload a file to S3, you will receive a HTTP 200 code if the upload was successful (on the API/CLI).

## Data Consistency Model for S3

* Read after Write consistency for PUTS of new Objects -   
  As soon as you uploaded the file to S3, you can access it.
* Eventual Consistency for overwriting PUTS and deletes -   
  It can take some time for updates to existing files and for deleting files to propagate through the system.

## S3 Structure

S3 is Object based. Objects consist of the following:

* Key: the name of the object (file)
* Value: the constent of the file as a sequence of bytes
* Version ID (important for versioning)
* Metadata: data about the data you’re storing (e.g. user defined tags such as team-name etc)
* Sub-resources – bucket-specific configuration:
  + Bucket policies, access control lists
  + Cross Origin Resource Sharing (CORS) – allowing files from one bucket to access files in another bucket
  + Transfer Acceleration – a service that allow you to accelerate the upload speed when you upload a large amount of files.

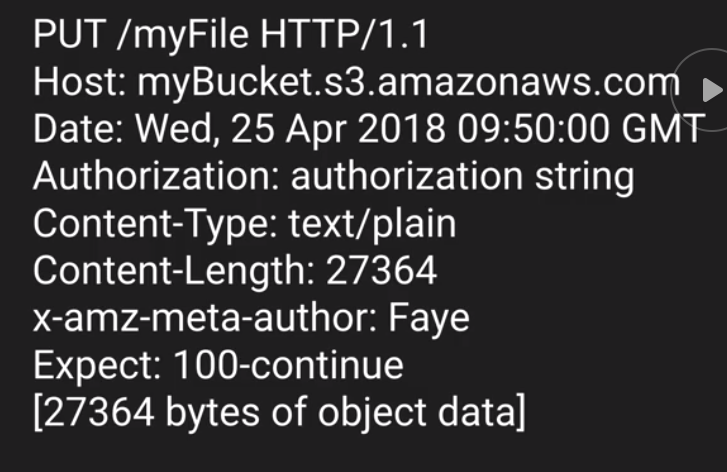
## S3 Services

* **Tiered Storage available**
  + S3 – suitable to most workloads;
    - 99.99 availability
    - 99.999999999% durability
    - Stored redundantly across multiple devices in multiple facilities and is designed to sustain the loss of 2 facilities concurrently
  + S3 – IA (Infrequently Accessed) – for data that is accessed less frequently but requires rapid access when needed.
    - The same as S3 but
    - Lower fee than S3 but you are charged a retrieval fee.
  + S3 – One Zone IA – Same as IA but the data is stored in a single availability zone only.
    - Still 11x9s durability. But only
    - 99.5% availability
    - Cost is 20% less than regular S3 – IA
  + Reduced Redundancy Storage:
    - 99.99% availability
    - 99.99% durability.
    - Used for data that can be recreated if lost (e.g. thumbnails)
    - Is being phased out from AWS.
  + Glacier – for archiving
    - Very cheap
    - Optimized for data that is infrequently accessed
    - Takes 4-5 hours to access
  + S3 Intelligent Tiering – suitable for data that has unpredictable/unknown access pattern:
    - 99.99 availability
    - 99.999999999% durability
    - Automatically moves your data to the most cost-effective tier (frequent or infrequent access) based on how frequently you access it:
      * If you don’t access it for 2 days, it’s moved to the infrequent tier. But
      * When you access it again, it moves back to the frequent tier.
    - Optimized cost
    - No fees for accessing your data but a (very) small monthly fee for monitoring/automation: $0.0025/1000 objects
* Charges:
  + Storage per GB
  + Requests (get/put/copy etc)
  + Storage management pricing:
    - Inventory, analytics and object tags.
  + Data management pricing:
    - Data transferred out of S3
  + Transfer Acceleration
    - Use CloudFront to optimize transfers
* Lifecycle management
* Versioning – support version control
* Encryption
* Secure your data – with Access Control Lists and Bucket Policies.

## S3 Security

* By default, all newly created buckets are PRIVATE
* You can set up access control to your buckets using:
  + Bucket Policy – applied at bucket level. Written in Jason.
  + Access Control List – applied at an object (file) level. Allow read/write/read-write control
* Access logs: S3 buckets can be configured to create access logs, which log all request made to the S3 bucket. These logs can be written to another bucket.

### Encryption

* In transit: SSL/TLS
* At rest:
  + Server Side Encryption:
    - SSE-S3 : S3 Managed Keys – AES 256 bits encryption. each object is encrypted with its own unique key. In addition, all keys are encrypted by a master key.
    - SSE-KMS: AWS Key Management Service, Managed Keys – you get a separate private key to encrypt your master key, you can also use your own private key. In addition, you get audit trail to know which keys were used and by who.
    - SSE-C: Server Side Encryption with Customer Provided Keys – AWS mange the encryption/decryption but you manage your own keys
  + Client Side Encryption: you encrypt the files yourself before uploading them into the bucket.
* Every time a file is uploaded to S3, a PUT request is initiated. For example:   
  
  + Expect: 100-continue means that the header must be acknowledged before the body (file data) of the message is sent to the bucket.
  + If the file is to be encrypted at upload time, the x-amz-server-side-encryption parameter will be included in the request header. This parameter can be AES256 or ams:kms for SSE-S3 and SSE-KMS managed keys respectively.
  + You can enforce the use of Server Side Encryption by using a Bucket Policy which denies any S3 PUT request that doesn’t include the x-amz-server-side-encryption parameter in the request header.

## Performance Optimization

* Already optimized out of the box.
* If we need to support > 3500 PUT/LIST/DELETE or >5500 GET requests per seconds, follow these guidelines for optimizing it further:
  + For GET-intensive workloads: use CloudFront content delivery service to get the best performance. To cache your most frequency

## Cross Origin Resource Sharing (CORS)

Allowing code or resources from one bucket to access resources in another bucket.

## Cloud Front – Amazon’s Content Delivery Network (CDN)

A system of distributed servers (network) that deliver webpages and other web content to a user based on the geographic location of the user, the origin of the webpage and the content delivery server.

* **Edge locations** – the location where the content is cached and can also be written. Separate to an AWS Region/Availability Zone
* **Origin** – The origin of all the files that the CDN will distribute. Origins can be S3 Bucket, an EC2 Instance, an Elastic Load Balancer or Route53.
* **Distribution** – Amazon’s name for the CDN which consist of a collection of Edge Locations.
  + Web Distribution – typically used for websites, HTTP/HTTPS.
  + RTMP (Adobe Real Time Messaging Protocol) – Used for media streaming and Flash multi-media content.
* Amazon CloudFront can be used to deliver your entire website, including dynamic, static, streaming and interactive content using a global network of edge locations.
  + Requests for your content are automatically routed to the nearest edge location so content is delivered with the best performance.
  + If the data doesn’t exist in the location, it is received from the origin and cached for future accesses.
  + The cache is cleared after its time-to-live expires.
    - You can also force a clear of the cache, for a fee.
  + You can use CloudFront also to change/upload files and not only to read/download them.
  + Integrated with all Amazon Web Services.
  + Can also work with any non-AWS origin server, which stores the original, definitive versions of your files.
  + You can have multiple origins per Cloud Front Distribution
  + Used for Amazon S3 Transfer Acceleration:
    - Amazon S3 Transfer Acceleration enables fast, easy and secure transfers of files over long distances between your end users and an S3 bucket.  
      Transfer Acceleration takes advantage of Amazon CloudFront’s globally distributed edge locations. As the data arrives at an edge location, it is routed to Amazon S3 over an optimized network path.

## Lab

* When creating a bucket:
  + In Set Permissions -> Manage system permissions:  
    If you want to enable server-access logging, you will have to choose ‘Grant Amazon S3 Log Delivery group write access to this bucket’
* To enforce encryption through bucket policy:
  + In your bucket -> Permissions-> Bucket Policy -> Policy generator
    - Select Type of Plicy: S3 Bucket Policy
    - Effect: Deny
    - Principle: \* (meaning: apply to all objects)
    - AWS Service: Amazon S3
    - Actions: PutObject
    - Amazon Resource Name : see the ARN in your bucket’s Permissions tab (see above)
    - Add Conditions:
      * Condition: StringNotEquals
      * S3: x-amz-server-side-encryption
      * Value: aws:kms
    - Add Condition -> Add Statement -> generate policy
    - You will now get the JSON of the policy. Copy it and paste it in the console in the BucketPolicy editor.  
      Note: if you get an error message ‘Action does not apply to any resource(s) in statement’ it means that instead of the ARN, you need to add wildcards to it so it will apply to all objects under it. So for example, if your ARN is   
      “arn:aws:s3:::fayes-encrypted-file”, you will need to set:  
      “Resources”: “arn:aws:s3:::fayes-encrypted-file**/\***” and save to remove the error.
  + Note: you can also set the encryption when you create the S3 bucket
* Host Static Website:
  + When creating the bucket, we made it public.
  + Under Properties -> Static website hosting -> Use this bucket to host a website
    - The Endpoint is the endpoint to your site.
* CORS configuration:
  + In the bucket that wants to allow another bucket to access it
  + Under permissions -> CORS configuration copy-paste from the CORS-config-example.txt.
  + In <AllowedOrigin>\*</AllowedOrigin> - replace the ‘\*’ with the end-point site url of the bucket you want to allow access.

### Create a CDN

* + 1. Create the origin. For example: create an S3 bucket.
    2. Under Networking->CloudFront-> Create Distribution -> Web
       1. Origin Domain Name: choose your bucket/EC2/an ip address (for your own server)
       2. Origin Path: if it’s a specific path within the domain name
       3. Restrict Bucket Access: yes means that users will not be able to directly access the bucket and will only access it through CloudFront
       4. Origin Access Identity – you require an identity to access CloudFront.
       5. Grant Read Permissions on Bucket – set to Yes
       6. Viewer Protocol Policy: choose ‘Redirect HTTP to HTTPS’
       7. Default TTL: think what is the change-rate of the data in your cache and then choose ½, 1/3 or ¼ of this time as your Default TTL, depending on how important it is for the data to be up to date.
       8. Restrict Viewer Access (Use Signed URL or Signed Cookies) – a way to restrict access to the data for paying customers only.
       9. AWS WAF (Web Application Firewall) Web ACL – a firewall that protect your content at the application level.
       10. Alternate Domain Names – if you want a user-friendly name for your CDN
       11. SSL Certificate – to be used for https
       12. Create Distribution
    3. In your distribution:
       1. Restrictions: you can limit the countries that can access your content.
       2. Invalidations – manually invalidating (clearing) files from your cache. Note: every time you use this, you will be charged a fee
    4. To use your CDN: use the CDN’s Domain Name as the root URL to access all files on it.

# Serverless

## Lambda

Lambda: allow you to run your code on the cloud without having to worry about any infrastructure, servers etc.

A compute service where you can upload your code and create a Lambda function. AWS Lambda takes care of provisioning and managing the servers that you use to run the code. You don’t have to worry about operating systems, patching, scaling etc. You can use Lambda in the following ways:

* As an event-driver compute service where AWS Lambda runs your code in response to events. There events could be changes to data in an Amazon S3 bucket or an Amazon DynamoDB table that triggers the Lambda function.
* As a compute service to run your code in response to HTTP requests using Amazon API Gateway or API calls made using AWS SDKs.

Lambda supports different languages:

* Node.js
* Java
* Python
* C#
* Go

Additional Notes:

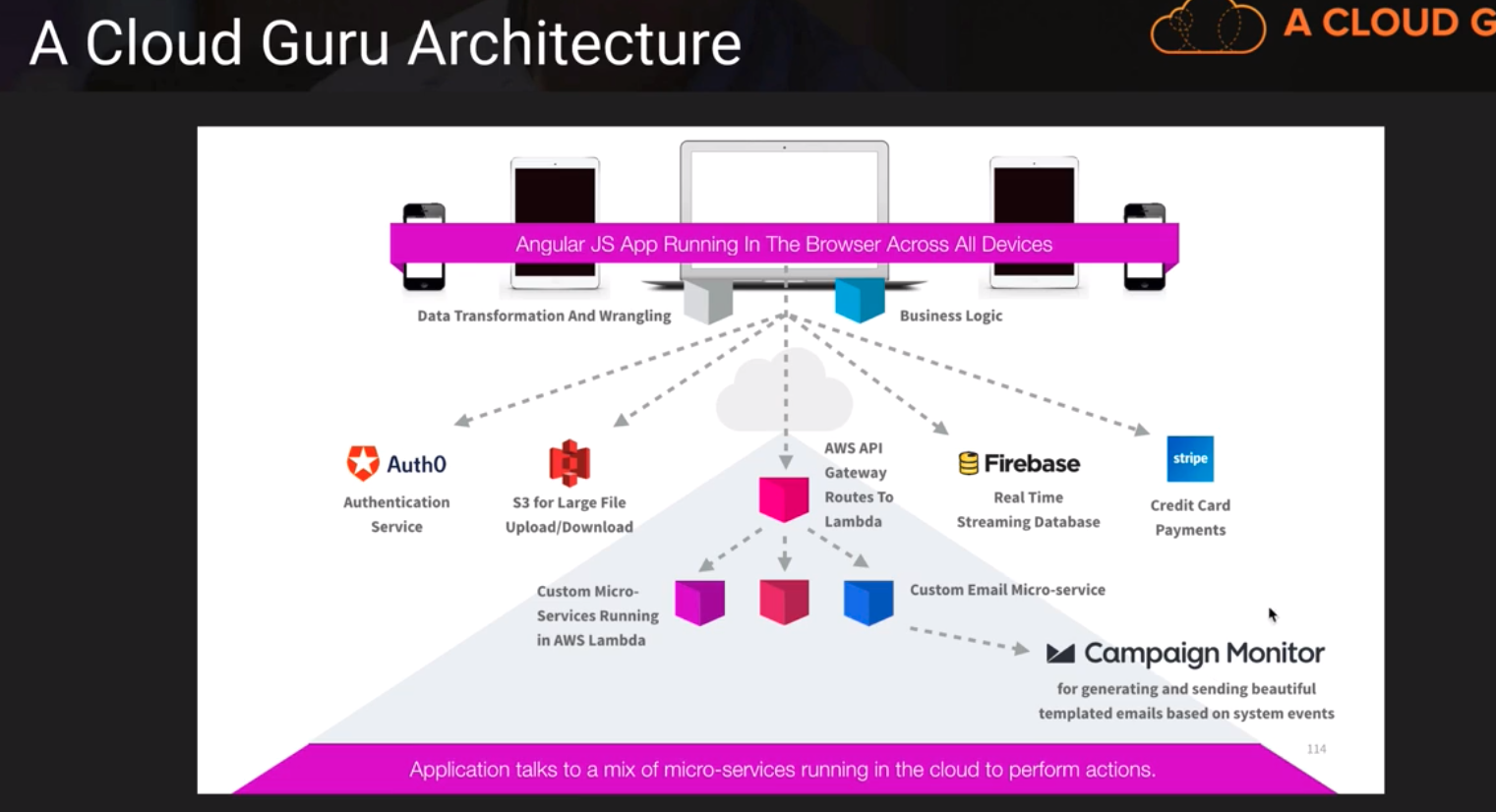
* Lambda functions are independent. 1 event => triggers 1 function (which in turn can trigger additional functions)
* Lambda is serverless.
* Architectures can get extremely complicated. AWS X-ray allows you to debug what is happening.
* Lambda can do things globally. You can use it to back up S3 buckets to other S3 buckets etc.
* Know your Lambda triggers:
* EMF – recommended for logging and metrics – easier to use and much more powerful than the traditional way (using PublishMetric)

### Advantages

* No servers
* Continuous, automatic scaling out (not up)
* Very cheap

### Pricing

* Number of requests
  + First 1 million requests per month are free. $0.20 per 1 million requests after this.
* Duration – the time your functions execute.



## API Gateway

# Messaging

## SQS

Simple Queue Service (SQS) – fully managed message queueing service.

* Guaranteed delivery – whether the other service is available or not (will receive them when becomes available)
* Standard SQS service:
  + Maximum throughput
  + Guaranteed at least once delivery
  + Best effort in order delivery
  + This means that messages can be delivered out of order and with duplications.
* Fifo SQS service:
  + Guarantee that messages arrive exactly once and
  + In order
  + Throughput will be lower than the Standard SQS

## SNS

Simple Notification Service (SNS) – a subscriber-producer service that allows registering to the producer and

* High throughput
* Push-based (producer pushes the messages to its subscribers)
* Many-to-many – many producers can trigger the SNS to notify multiple subscribers.
* Uses retries if the delivery failed (e.g. service is offline). If the retries fail, you can set up a DeadLetterQueue (DLQ) where SNS will drop the messages that it failed to deliver.
* Supported subscribers:
  + SQS
  + AWS Lambdas
  + HTTPS endpoints
  + Direct to user:
    - SMS
    - Mobile push
    - Email

### SNS + SQS Pattern

* Create an SNS topic
* Create SQS queues and subscribe them to the SNS topic.
  + Endpoint: For the sqs protocol, the endpoint is the ARN of an Amazon SQS queue.
  + You can add filtering rules for the SQS queues. If there are no filtering rules, you will receive all messages.

# CDK – Cloud Development Kit

Defining your cloud configuration in code (Infrastructure as a Code - IaaC)

* Allows you to create CloudFormation templates to request and initialise all the AWS infrastructure you need to run your app. Support python, java etc.
* AWS CLI – command line tool to interact with AWS services.
* CloudFormation – configuration as code.

## Yaml

JSON to YAML converter: <https://www.json2yaml.com/>

YAML to JASON: <https://www.convertjson.com/yaml-to-json.htm>

A configuration language that can be used to configure CloudFormation templates.

* --- : start of yaml file
* … : ends yaml file (usually optional)
* Yaml file consist of {key}: {value} pairs
* Nesting is marked with indentation (like python).  
  Note: tabs are not allowed!
* New line ends a field
* # a full line comment in yaml
* Supported values:
  + Null: ~ or null
  + String with ‘ or “
  + Floating-point number e.g. 3.14159
  + Boolean: e.g. true, on, yes vs false, no or off.
  + Integer e.g. 5, 0x6 etc
  + Array:
    - On one line: items: [ 1, 2, 3, 4, 5 ]
    - On multiple lines, each element will start on a new line with a ‘-‘ at the beginning of it. Example:

calling-birds:

  - huey

  - dewey

  - louie

  - fred

* + Dictionary:
    - On one line: foo: { thing1: huey, thing2: louie, thing3: dewey }
    - On multiple lines – by using another level of indentation. In the following example, ‘xmas-fifth-day’ is a dictionary with ‘calling-birds’, etc elements. ‘partridges’ is another dictionary:

xmas-fifth-day:

  calling-birds: four

  french-hens: 3

  golden-rings: 5

  partridges:

    count: 1

    location: "a pear tree"

  turtle-doves: two

Special yaml-parameters for CloudFormation templates:  
<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/AWS_SQS.html>

* !Ref {logical name} – return the value of the parameter or resource. Examples:
  + !Ref logicalName
  + Long format:  
    Ref: logicalName
* !GetAtt {logical name}.{attribute name} -   
  return the attribute defined in the template file(s) for this resource.  
  Examples:
  + !GetAtt logicalNameOfResource.attributeName
  + # long format of the same command:  
    Fn::GetAtt: [ logicalNameOfResource, attributeName ]

ReferenceForOneValue:

Ref: MyLogicalResourceName

ReferenceForOneValueShortCut: !Ref MyLogicalResourceName

FunctionResultWithFunctionParams: !Sub |

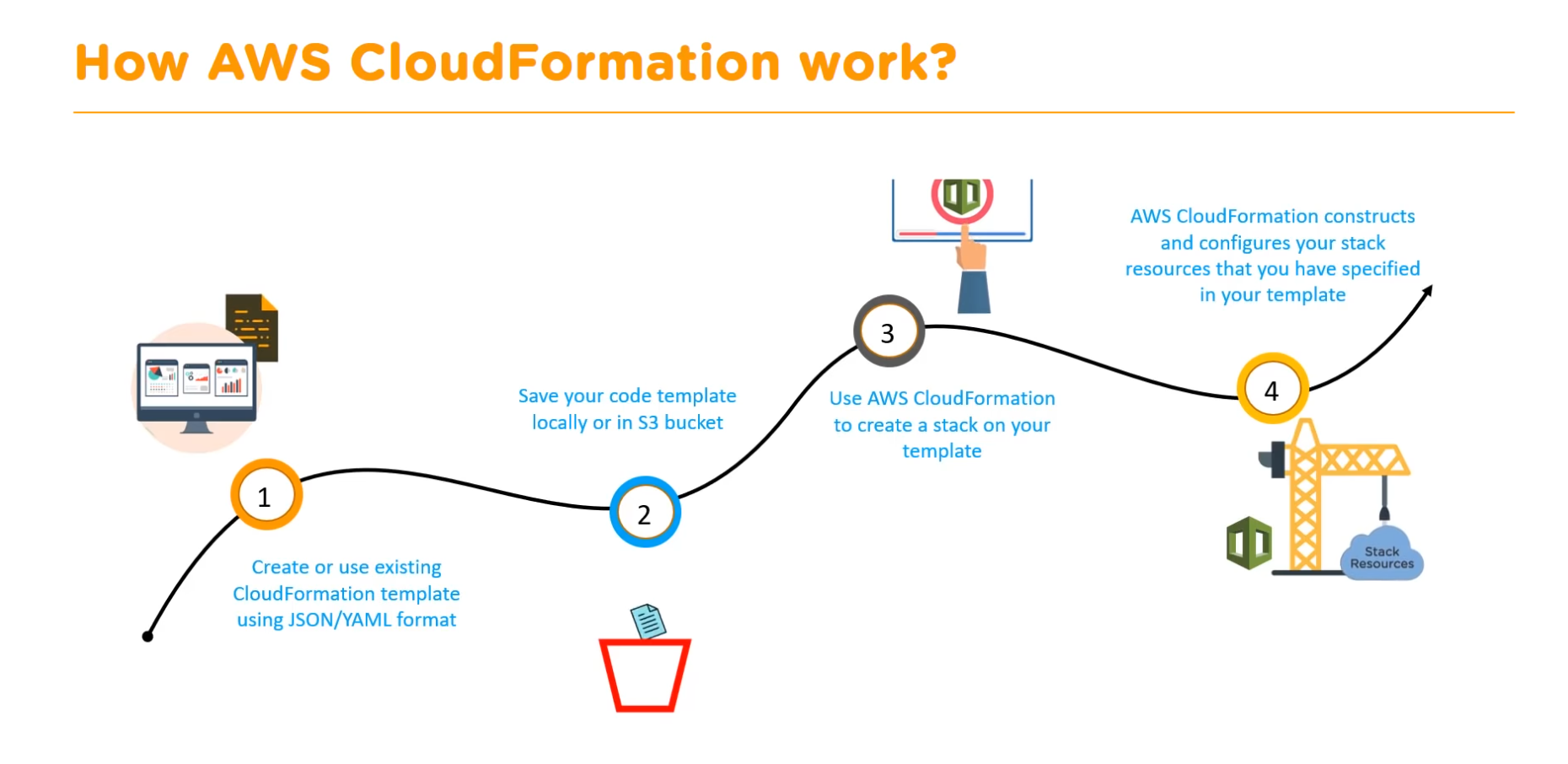
Key=%${MyParameter}

## CloudFormation

A service that provision and configure AWS resources based on a template file we create. CloudFormation takes care of finding and configuring the dependencies.

CloudFormation support

* Template – the text file (JSON or yaml) that define our required resources configuration.
* Stack – the resources that the template provisioned.



### CloudFormation Template

* CloudFormation Designer – allow us to create the cloudFormation template.
* Includes:
  + Format Version – identify our version
  + Description
  + Metadata- details on the resources in the template
  + Parameters – allow us to customize the template.
  + Mapping – match a key to a set of values. To support different parameters per region for example
  + Conditions – conditional creation of additional resources etc
  + Transform –
* Note:
* Recognising Drift: <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-stack-drift.html?icmpid=docs_cfn_console>

# Fargate

Managed Service to run containers. Allows customers to use Amazon Elastic Containers Service (ECS) and Amazon Elastic Kubernetes Service (EKS) to launch applications without having to deal with the maintenance of the infrastructure.

* Deploy and manage applications, not infrastructure
* Secure isolation by design – Each ECS task/EKS pod run in their own runtime environment. They don’t share CPU, memory or network resources to make them more secure.
* Auto scaling and flexible pricing
* Metrics and logs support out of the box.

# Reliability & Resiliency

## Fault Isolation

Limit the blast radius of a failure by using redundant independent components separated across **fault isolation zones.**

* Resource and Request – done automatically by AWS through partitioning every resource and request into cells and ensuring they contain and isolate failures internally.
* Availability Zone (AZs) – completely independent facilities in different geographic zones that allow redundancy of your service. Note: the AZs are location such that there will be minimal latency when propagating information between them.
* Region – more physically distant than AZs. Each region contain 2+ AZs. Used for even higher redundancy but also has higher overhead because of lack of shared infrastructure. Can use Route53 and additional cross-regions services to scale your service to work across regions.

# Performance

Think of your services as cattle, not pets.

## Compute

Services that will process your data (CPUs, virtual machines):

* EC2- VM-based compute – is the most familiar one. The most expensive and requires the most maintenance.
* Container-based – enable a finer division of your workload and can scale quickly but requires more complex configuration and orchestration.
* Serverless-based – abstract away most of the management and scaling complexity but has hard system limitations and require adopting new toolchain and processes.

## Storage

For storing files, block, object or archive

* EBS – file storage for a single EC2
* EFS (Elastic File System) – storing files for multiple clients
* S3 – for storing big blobs of data to be access by many clients
* S3 Glacier – for storing infrequently-accessed archives

## Databases

* Relational Databases – let you have joins and ACID properties (transactions) but have an upper limit on performance and data storage
* Non-relational databases – more flexible and scalable but usually lack joins and ACID properties
* Data warehouse – enable large-scale analytics
* Data indexing and searching solutions – let you index and search data from a wide range of sources.

## Networks

# Testing

## Hydra

Serverless test orchestration platform for NAWS services.

# Monitoring

## Collection

Collecting metrics:

* Infrastructure-level: collected automatically by AWS and can be accessed through CloudWatch
* Application-level: generated by your app. Can be collected by CloudWatch Custom Metrics. Can be stored using Cloud Watch or propagated to S3 buckets.
* Account-level metrics: automatically collected by AWS and can be collected through CloudTail service

## Analytics

Choosing the right one depends on your use case:

* To analyze logs stored in CloudWatch Logs, consider using [CloudWatch Logs Insight](https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/AnalyzingLogData.html), a service that lets you interactively search and analyze your Cloudwatch log data
* To analyze logs stored in S3, consider using [Athena](https://aws.amazon.com/athena/?id=docs_gateway), a serverless query service
* To analyze structure data, consider using [RDS](https://docs.aws.amazon.com/rds/?id=docs_gateway), a managed relational database service
* To analyze large amounts of structured data, consider using [RedShift](https://docs.aws.amazon.com/redshift/?id=docs_gateway), a managed petabyte-scale data warehouse service
* To analyze log-based data, consider using the [Elasticsearch Service](https://docs.aws.amazon.com/elasticsearch-service/?id=docs_gateway), a managed version of Elasticsearch, the popular open-source analytics engine

## Action

You can use CloudWatch to:

* Configure alarms
* Dashboards
* Track performace and business KPI

## CloudWatch

Logs->insights you can find fine-grained data

Logs->Groups

* + searchAll to search all the log streams (each thread create it’s own stream?) at once.

# For the exam

* See the Exam Blueprint pdf
* I don’t need to memorise the EC2 instance types for the exam

# Lab

# Additional Tools

Tools that are commonly used in AWS applications, although not an interal part of the AWS stack

## Elastic Search (ES)

Open source, document-based search platform. Optimized for speed of searching and for searching huge amounts of data. Not optimized for consistency or atomicity.

Terminology:

* Cluster – a group of one or more servers
* Node – one server in a cluster.
* Records – documents
* Documents are stored in indexes which can be sharded, or split into smaller pieces.
* ES can run these shards on different nodes in order to improve performance and scale. You can and should replicate shards onto other nodes in case of network/server issues.
* ES uses Apache Lucene to index documents for fast searching.
* Both Apache Lucene and ES are written in Java.
* You can run ES locally or consume it via AWS or Google Cloud Platform (GCP). You can also use Docker containers with ES on them.
* Development mode: for local use, without clustering. For playing around with ES, see <https://www.docker.elastic.co/> for docker images with ES already installed on them.
* Production mode: requires setting up the cluster, including security
* <https://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/ultrawarm.html>
* <https://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/sizing-domains.html>

### ES Setting

In your AWS account -> Elastic Search Service.

This allows you to configure and see your setting and performance.

See best practices: <https://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/aes-bp.html>

To send ES console commands:

* + - 1. Open Kibana to your ES Kibana endpoint (see in your ES configuration in your AWS account)
      2. In Kibana, click the 'DevTools' icon on the left menu to open the Kibana console where you'll be able to send console commands to ElasticSearch
      3. See supported ES commands here: <https://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/aes-supported-es-operations.html>