Exam Guide

* MC
* Multi-Response
* 720 / 1000
* <https://aws.amazon.com/certification/certified-developer-associate/>
* Domains (5)
  + Deployment
    - 22%
    - CICD
    - Beanstalk
    - Serverless
  + Security
    - 26%
    - Deep-dive each service
  + Development and AWS Services
    - 30%
    - Serverless, API, SDK, & CLI
  + Refactoring
    - 10%
    - Understand all the AWS services for the best migration to the cloud
  + Monitoring and Troubleshooting
    - 12%
    - CloudWatch, CloudTrail, and X-Ray

Navigating the AWS Spaghetti bowl

The AWS developer journey

Fundamentals - Three Tier approach

Developer Tools

* The AWS developer journey - Deploying Properly and Continuously with monitoring and infrastructure as code

Application Decoupling and Integration

Serverless Paradigm

Always Secure (Deep dive)

Overview of other services

AWS Fundamentals:

Regions geographical locations with AZs that are physical data centers

* All AZs are separated so that they are isolated from disasters
* AWS consoles are region scoped, except for IAM and S3)

Identity and access management

* All AWS is found here
  + Users
  + Groups
  + Roles
* The root account should never be used (and shared)
* Users must be created with proper permissions
* IAM is at the center of AWS functionality
* Policies are written in JSON

IAM introductions

* Users
* Groups (Multiple Users)
* Roles (Internal usage within AWS resources)
* Policies define what the above can do or cannot do
* It is best practice to give users the minimal amount of permissions they need to perform their job ( least privilege principles)

IAM Federation (Big Enterprises)

* Big enterprises usually integrate their own repository of users with IAM
* So that one can log in to AWS using company credentials
* Identity Federation uses the SAML standard (Active Directory)

IAM 101 Brain DUMP

* One IAM User per Physical Person
* One IAM Role per Application
* IAM credentials should never be shared
* NEVER, ever, ever, ever write IAM credentials in code….EVER
* Never use the ROOT account except for initial setup
* Never use ROOT IAM Credentials’

EC2

* Functionality
  + Renting virtual machines (EC2)
  + Storing data on virtual devices (EBS)
  + Distributing load across machines (ELB)
  + Scaling the services using an auto-scaling group (ASG)

SSH (Allows you to control a remote machine all using the command line.)

* sudo ssh -i KevValue.pem ec2-user@IPv4PublicId
* Chmod 0400 KeyValue.pem

Security Groups:

* Access to Ports
* Authorized IP ranges - IPv4 and IPv6
* Control of inbound network (from other to instance)
* Control of outbound network (from the instance to other)
* Proper Security Properties with SSH port 22
  + Add IP xx.xx.xx.xx
* GOOD TO KNOW
  + Can be attached to multiple instances
  + Locked down to a region/VPC combination
  + Does live “outside” the EC2 - if traffic is blocked the EC2 instance won’t see it
  + *It’s good to maintain one separate security group for SSH*
  + If your application is not accessible (Time Out) then it’s a security group issue
  + If your application gives a “connection refused” error, then it’s as application error or it’s not launched
  + All inbound traffic is **blocked** by default
  + All outbound traffic is authorized by default

Private vs Public IP (IPv4)

* IPv4 is the most common format used online
* IPv6 is newer and solves problems for the internet of things (IoT)
* IPv4 3.7 billion different addresses in the public space

Public IP:

* Public IP means that the machine can be identified on the internet
* Must be unique across the whole web ( not two machines can have the same public ip).
* Can be geo-located easily

Private IP:

* Private Ip means the machine can only be identified on a private network only
* The ip must be unique across the private network
* But two different private networks (two companies) can have the same IPs.
* Machines connect to the internet using an internet gateway or a proxy
* Only a specified range of IPs can be used as private IP

Elastic IP:

* When you stop and then start an EC2 instance, it can change its public IP.
* If you need to have a fixed public Ip for your instance you need an Elastic IP
* An elastic IP is a public IPvt IP you own as long as you don’t delete it
* You can attach it to one instance at a time
* With an Elastic IP address, you can mask the failure of an instance or software by rapidly remapping the address to another instance in your account.
* You can only ave 5 elastic IPs in you account ( you can ask AWS to increase that)
* However, overall you should avoid being tied to elastic IPS. (INSTEAD)
  + They often reflect poor architectural decisions
  + Instead, use a public Ip and register a DNS name to it
  + Or, as we’ll see later , use a LOad Balance and don’t use a public IP

**HANDS-ON:**

* By default, your instance comes with:
  + A private Ip for the internal AWS network
  + A public IP for internet access.
* When we are doing SSH into our instances:
  + We can’t use a private IP because we are not in the same network
  + We can only use the public IP
* If your machine is stopped and then started the public IP can change.
* An account will be charged for the number of elastic IPs an account has

**Create an apache webserver:**

* Update
  + Sudo su
  + Yum update -y
* Install
  + Yum install -y httpd.86\_64
* Start
  + Systemctl start httpd.service
    - A common error here is that systemctl will not be found and this is because you are running on amazon linux instead of amazon linux 2
  + Systemctl enable httpd.service

EC2 User Data

* It is possible to bootstrap our instance using an EC2 User Data script
* **Bootstrapping** means launching commands when a machine starts
* That script is **only run once** at the instance first start
* EC2 user data is used to automate boot tasks such as:
  + Installing updates
  + Installing software
  + Downloading common files from the internet
  + Or anything you can think of
* The EC2 User Data Scripts runs with the root user

EC2 Instance Launch Types

* On-Demand Instances: short workload, predictable pricing
  + Pay for what you use (billing per second after the first minute)
  + Has the highest cost but no upfront payment
  + No long term commitment
  + **Recommended for short-term and un-interrupted workloads, where you can’t predict how the application will behave.**
* Reserved Instances: long workloads( >= 1 year)
  + Up to 75% discount compared to on-demand
  + Pay upfront for what you use with a long term commitment
  + Reservation period can be 1 or 3 years
  + Reserve a specific instance type
  + **Recommended for steady-state usage applications (think database)**
* Convertible Reserved Instances: long workloads with flexible instances
  + Can change the EC2 instance type
  + Up to 54% discount
* Scheduled Reserved Instances: launch with time window you reserve
  + Launch within the time window that you reserve
  + When you require a fraction of day/week/month
* Spot Instances: short workloads, for cheap, can lose instances
  + Can get a discount of up to 90% compared to on-demand
  + You bid a price and get the instance as long as its under the price
  + Price varies based on offer and demand
  + Spot instance is reclaimed with a 2-minute notification warning when the spot price goes above your bid
  + **Used for batch jobs, big data analysis, or workloads that are resilient to failures.**
  + **Not Great for critical jobs or databases**
* Dedicated Instances: No other customers will share your hardware
  + Instances running o hardware that ‘s dedicated to you
  + May share hardware with other instances in the same account
  + No control over instance placement (can move hardware after stop//start
* Dedicated Hosts: book an entire physical server, control instance placement
  + Physical dedicated ec2 server for your use
  + Full control of ec2 instance placement
  + Visibility into the underlying sockets/ physical cores of the hardware
  + Allocated for you account for a 3 year period reservation
  + More expensive
  + **Useful for software that complicated licensing model (BYOL - Bring Your Own License)**

**DIFFERENCES BETWEEN DEDICATED HOSTS AND INSTANCES**

| **CHARACTERISTIC** | **INSTANCE** | **HOST** |
| --- | --- | --- |
| **Enables the use of dedicated physical servers** | **x** | **x** |
| **Per instance billing (subject to a 2$ per region fee** | **x** |  |
| **Per host billing** |  | **x** |
| **Visibility of sockets, cores, host ID** |  | **x** |
| **The affinity between a host and instance** |  | **x** |
| **Targeted instance placement** |  | **x** |
| **Automatic instance placement** | **x** | **x** |
| **Add Capacity using an allocate request** |  | **x** |

EC2 Pricing

* EC2 Instances prices (per hour) varies based on these parameters:
  + The region you’re in
  + Instance Type being used
  + On-Demand vs Spot vs Reserved vs Dedicated
  + Linux vs Windows vs Private OS
* You are billed by the second, with a minimum of 60 seconds
* You also pay for other factors such as storage, data transfer, fixed IP public addresses, load balancing
* You do not pay for the instances if the instance has been stopped
* **PRICING PAGE**
  + [**https://aws.amazon.com/ec2/pricing/on-demand/**](https://aws.amazon.com/ec2/pricing/on-demand/)

**Building custom AMI’s**

* Advantages
  + Pre-installed packages needed
  + Faster boot time
  + The machine comes configured with monitoring/ enterprise software
  + Security concerns - control over the machines in the network
  + Control of maintenance and upstates of AMI’s overtime
  + Active Directory Integration out of the box
  + Installing your app ahead of time (for faster deploys when auto-scaling)
  + Using someone else’s AMI that is optimized for running an app DB etc..
  + **AMI is built for a specific AWS Region**
* **Instances have 5 distinct characteristics advertised on the website:**
  + The RAM
  + The CPU
  + The I/O
  + The Network
  + The GPU
  + [Https://aws.amazon.com/ec2/instance-types/](https://aws.amazon.com/ec2/instance-types/)
  + <https://ec2instances.info/>
  + M instance types are balanced
  + T2/T3 are burstable instances
    - AWS has the concept of fo burstable instances (t2 time machines)
    - Burst means that overall, the instance has OK CPU performance
    - When the machine needs to process something unexpected ( A spike in load for example\_, it can burst,m and CPU can be VERY good
    - If the machine bursts it utilizes “burst credits”
    - If all the credits are gone the CPU becomes BAD
    - If the machine stops bursting, credits are accumulated over time
    - **Burstable Instances AD**
      * Burstable instances can be amazing to handle unexpected traffic and getting the insurance that it will be handled correctly
      * If your instance consistently runs low on credit, you need to move to a different kind of non-burstable instance all the ones described before.
    - CPU credits
    - T2 Unlimited
      * It is possible to have an unlimited burst credit balance
      * You pay extra money if you go over you credit balance, but you don’t lose in performance
      * Overall it is new offering so be careful of costs as they can go high if you’re not monitoring the health of your instances
      * Read more https://aws.amazon.com/blogs/aws/new-2 -unlimited-going=beyond-the-burst-with-high-performance/
* EC2 Checklist
  + SSH into EC2
  + Use Securely Groups Properly
  + Difference between private, public, elastic IPS
  + How to use User Data to customize your instance at boot time (automated build scripts)
  + How to build custom AMI’s to enhance you OS
  + EC2 instances are billed by the second and can be easily created and thrown away

Load Balancer, Auto Scaling groups, and Elastic Block Store

* Load balancers are servers that forward internet traffic to multiple servers (ec2 instances) downstream.
* Spread load across multiple downstream instances
* Expose a single pint of access (DNS) to your applications
* Seamlessly handle failures of downstream instances
* Do regular health checks to your instances
* Provide SSL termination (HTTPS\_ for your websites
* Enforce stickiness with cookies
* High availability across zones
* Separate public traffic from private traffic

Why use an EC2 Load Balancer

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

Types of load balancer on AWS

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

Health Checks

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

Application Load Balancer (v2) Layer 7 HTTP/HTTPS traffic

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

Good to Know (ALB) Application Load Balancer

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

Network LOad Balancer (v2) Layer 4 TCP Traffic

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

Good to Know

* Classic Load Balancers are Deprecated
  + Application Load Balancers for HTTP/ HTTPS & Websockets
  + Network Load Balancer for TCP
* CLB and ALB support SSL certificates and provide SSL termination
* All Load Balancers have health check capability
* ALB can route on based on hostname/path
* ALB is a great fit with ECS (DOCKER)

Any load balancer has a static hostname. DO not resolve and use underlying IP

* LBs can scale but not instantaneously - contact ASWS for a “warm-up”
* NLB directly see the client IP
* 4xx errors are client induced errors
* 5xx errors are application induced errors
  + Load Balancer Errors 503 means at capacity or no registered target
* If the LB can’t connect to your application, check your security groups!

Load Balancer

1. Select type of load balancer
2. Configure load balancer
   1. Name
   2. Scheme (external or internal)
   3. IP address type (IPv4 or IPv6)
   4. The listener is a process that checks for connection requests using the protocols and port in these configurations.

Auto Scaling Groups (ASG)

* In real-life, the load on your websites and applications can change
* In the cloud, you can create and get rid of servers very quickly
* The goal of an Auto Scaling Group is to:
  + Scale-out (add more ec2 instances) to match an increased load
  + Scale in (remove ec2 instances) to match a decreased load
  + Ensure we have a minimum and the maximum number of machines running.
  + Automatically register new instances to a load balancer
  + Also the desired capacity and the actual size.
* ASGs have the following attributes
  + A launch configuration
    - AMI + Instance Type
    - EC2 User Data
    - EBS Volumes
    - Security Groups
    - SSH Key Pair
  + Min Size/ Max Size/ Initial Capacity
  + Network + Subnets Information
  + Load Balancer information
  + Scaling Policies
* Auto Scaling Alarms
  + It is possible to scale an ASG based on CloudWatch alarms
  + An alarm monitors a metric (such as average CPU)
  + Metrics are computed for the overall ASG instances
  + Based on the alarm
    - We can create scale-out or in policies depending on if the demands need to increase or decrease running instances.
* It is now possible to define “better” auto-scaling rules that are directly managed by EC2
  + Target Average CPU Usage
  + Number of requests on the ELB per instance\
  + Average network in or out
* Theses rules are easier to set up and can make more sense
* Auto Scaling Custom Metric
  + We can auto-scale based on a custom metric (ex: number of connected users)
  + Send custom metrics from the application on EC2 to CloudWatch.(PutMetric API)
  + Create a CloudWatch alarm to react to low/high values
  + Use the CloudWatch alarm as the scaling policy for ASG

ASG Brain Dump

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

What’s an EBS Volume?

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

EBS Volume:

* It’s a network drive
  + It uses the network to communicate with the instance which means there might be a bit of latency
  + It can be detached from an EC2 instance and attached to another one quickly
* It’s locked to an Availability Zone (AZs)
  + An EBS volume in us-east 1 can not be attached to an instance in Us-east-2
  + To move a volume across you first need to snapshot it
* Have a provisioned capacity ( size in GBs and IOPS (inputs/outputs per second))
  + You get billed for all the provisioned capacity

EBS Volume Types

* EBS Volumes come in 4 types
  + GP2 (SSD): General purpose SSD volume that balances price and performance for a wide variety of workloads
  + IO1 (SSD): Highest-Performance SSD volume for mission-critical low-latency or high-throughput workloads
  + ST1 (HDD): Low-cost HDD volume designed for frequently accessed, throughput intensive workloads.
  + SC1 (HDD): Lowest cost HDD volumes designed for less frequently accessed workloads
* EBS Volumes are characterized in Size/Throughput, IOPS
* When in doubt always consult the AWS documentation - it’s good!

EBS Volume Resizing

* You can resize the EBS volumes
* You can only increase the EBS volumes:
  + Size(Any Volume Type)
  + IOPS(only in IO1)
* After resizing an EBS volume, you need to repartition your drive

EBS Snapshots

* EBS Volumes can be backed up using “snapshots”
* Snapshots only take the actual space of the blocks on the volume
* If you snapshot a 100GB drive that only has 5GB of data, then you EBS snapshot will only be 5GB
* Snapshots are used for:
  + Backups: ensuring you can save your data in case of catastrophe
  + Volume Migration:
    - Resizing a volume down
    - Changing the volume type
    - Encrypt a volume

EBS Encryption

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

EBS vs Instance Store

* Some instance does not come with Root EBS volumes
* Instead, they come with the “Instance Store”.
* Instance store is physically attached to the machine
  + Pros
    - Better I/O performance
  + Cons:
    - On termination, the instance store is lost
    - You can’t resize the instance store
    - Backups must be operated by the user
  + Overall, EBS-backed instances should fit most application workloads
  + Only specific cases and a team of good DevOps engineers should use instance stores within the cloud.

EBS Brain Dump:

* EBS can be attached to only one instance at a time
* EBS is locked at the AZ level
* Migrating an EBS volume across AZ means first backing it up (snapshot), then recreating it in the other AZ
* EBS backups use IO and you should not run then while your application is handling a lot of traffic
* Root EBS Volumes of instances get terminated by default if the EC2 instance gets terminated (you can disable this though)

AWS Route 53 Overview

* Route53 is a managed DNS
* DNS is a collection of rules and records which helps clients understand how to reach a server through URLs.
* In AWS, the most common records are:
  + A: URL to IPv4
  + AAAA: URL to IPv6
  + CNAME: URL to URL
  + Alias: URL to AWS resource

ROUTE 53 - The way it works:

The way that route 53 works are that the browser will make a request to the domain name website. Route 53 then sends back to the browser the IP address associated with the domain name. The browser then makes a request to the specific IP address that it received back.

Route 53 Overview:

* Route 53 can use:
  + Public domain names you own or buy
  + Private domain names that can be resolved by your instances in you VPCs
* Route 53 advanced features
  + Load balancing (through DNS -also called client load balancing)
  + Health check ( although limited)
  + Routing policy: simple, failover, geolocation, geo proximity, latency, weighted.
* Prefer Alias over CNAME for AWS resources (for performance reasons)

AWS Things to remember for DEV exam

* Record types
  + A
  + AAAA
  + CNAME
  + Alias (use over CNAME)

RDS (Relational Database Service):

* A managed DB service for DB use SQL as a query language
* It allows you to create databases in the cloud that are managed by AWS, Supports:
  + Postgres
  + Oracle
  + MySQL
  + MariaDB
  + Microsoft SQL
  + Aurora (AWS proprietary database)
* Advantage over using RDS versus deploying a DB on EC2:
  + Managed Service
  + OS patching level
  + Continuous back-ups and restore to specific timestamp (point in time restore)
  + Monitoring dashboards
  + Read replicas for improved read performance
  + Multi az setup for Disaster Recovery (RD)
  + Maintenance windows for upgrades
  + Scaling Capability (vertical and horizontal)
  + HOWEVER, you CANNOT SSH into your instances
* RDS Read Replicas for reading scalability
  + Up to 5 Read Replicas
  + Within AZ, Cross AZ or Cross-Region
  + Replication is ASYNC, so reads are eventually consistent
  + Replicas can be promoted to their own DB
  + The application must update the connection string to leverage read replicas
* RDS Multi AZ (Disaster Recovery)
  + SYNC replication
  + One DNS name -automatic app failover to standby
  + Increase availability
  + Failover in case of loss of AZ, loss of network, instance or storage failure
  + No manual intervention in apps
  + Not used for scaling
* RDS Backups
  + Backups are automatically enabled by RDS
  + Automated Backups:
    - Daily full snapshots of the database
    - Capture transaction logs in real time
    - Ability to restore to any point in time
    - 7 day retention (can be increased to 35 days
  + DB Snapshots:
    - Manually triggered by the user
    - Retention of backups for as long as you want
* RDS ENcryption
  + Encryption at rest capability with AWS KMS -AES-256 encryption
  + SSL certificates to encrypt data to RDS in flight
  + To enforce SSL:
    - PostreSQL: rds.force\_ssl=1 in the AWS RDS console (parameter groups)
    - MySQL: Within the DB:
      * GRANT USAGE ON \*.\* TO ‘mysqluser’@%’ REQUIRE SSL;
  + To Connect using SSL:
    - Provide the SSL trust certificate (can be downloaded from AWS)
    - Provide the SSL options when connecting to database
* RDS security
  + RDS databases are usually deployed within a private subnet not in a public one
  + RDs Security works by leveraging security groups (the same concept for EC2 instances) - it controls who can communicate with RDS
  + IAM policies help control who can manage AWS RDS
  + Traditional USername and PAssword can be used to login to the database
  + IAM users can now be used too (for MySQL / Aurora - NEW!)
* RDS vs Aurora
  + Aurora is a proprietary technology from AWS (not open sourced)
  + Postgres and MySQL are both supported as Aurora DB ( that means your drivers will work as if Aurora was a Postgres or MySQL database)
  + Aurora is “AWS cloud optimized” and claims 5x performance improvements over MySQL on RDS< over 3x the performance of Postgres on RDS
  + Aurora storage automatically grows in increments of 10GB, up to 64TB.
  + Aurora can have 15 replicas while MySQL has 5, and the replication process is faster (sub 10 ms replica lag)
  + Failover in Aurora is instantaneous. It’s HA native
  + Aurora costs more than RDS (20% more) - but it is more efficient

AWS ElastiCache Overview

* The same way RdS is to get managed relational databases
* ElastiCache is to get managed Redis or Memcached
* Caches are in-memory databases with really high performance, low latency
* Helps reduce load off of databases for read intensive workloads
* Helps make you applications stateless
* Write scaling using sharding
* Read scaling using read replicas
* Multi AZ with Failover capability
* AWS takes care of OS maintenance / patchin optimizations , setup configuration, monitoring, failure recovery and backups
* Basically an RDS for caches

ElastiCache

Solution Architecture - DB Cache

* Applications queries ElastiCache, if not available, get from RDS and store in ElastiCache
* Basically it stores frequently accessed stored data so that it does not have to go all the way to RDS
* Helps relieve load in RDS (usually read load)
* Cache must have an invalidation strategy to make sure only the most current data is used in there.

Elasticache Solution Architecture User Session Store

* User logs into any of the application, then the application writes the session data inot Elasticache
* The user hits another instance of our application then it will retrieve the session from the elasticache
* The Instance retrieves the data and the user is already logged in
* Relieve load and share state

Redis Overview

* Redis is an in-memory key value store
* Super low latency (sub ms)
* Cache survive reboots by default (its called persistence)
* Great to host user session
  + User sessions
  + Leaderboard for gaming because it has sorting ability
  + Distributed states
  + Relieve pressure on databases (such as RDS)
  + Pub / Sub capability for messaging
* Multi AZ with Automatic Failover for disaster recovery if you don’t want to lose your cache data
* Support for Read Replicas

Memcached Overview

* Memcached is an in-memory object store
* Cache doesn't survive reboots
* Use cases:
  + Quick retrieval of objects from memory
  + cache often accessed objects
* Overall, Redis has largely grown in popularity and hasbetter feature sets than memcached.
* Recommendation: Use Redis for caching needs over memcached

Other:

* Can be encrypted in transit and at rest

ElasticCache patterns

* ElastiCache is helpful for read-heavy application workloads
  + Social networks, gaming, media sharigin, Q&A portals (read heavy)
* & compute-intensive workloads (recommendation engines)
* There are two patterns / cache strategies for ElastiCache
  + Lazy Loading
  + Write Through
* Strategies may be different based on the kind of application you have

Lazy Loading -- Load only when necessary

* Cache hit great -- return data
* Cache miss
* Read from DB to application
* Then application writes to cache
* Pros:
  + Only requested data is cached ( the cache sn’t fileld up with unused data)
  + Node failures are not fatal (just increased latency to warm the cache)
* Cons:
  + Cache miss penalty taht results in 3 round trips, noticeable delay for that request
  + Stale data: data can be updated in teh database and outdated in the cache
  + Way to get around use a Time To Live rule (TTL)

Write Through --

* Add or Update cache when database is updated
* Only looks at caches on read actions
* If it is a write action it will write straight to the DB
* Then immediately write to the cache
* Very few cache misses
* Pros
  + Data in cache is never stale
  + Write penalty vs read penalty (each write requires 2 calls)
* Cons
  + Missing Data until it is added / updated in the DB. Mitigation is to implement Lazy Loading strategy as well
  + Cache churn - a lot of data will never be read
  + Cache is almost as big as the DB

AWS VPC

* Within a Region, you’re able to create VPCs (Virtual Private Cloud)
* Each VPC contains submets (networks)
* Each subnet must be mapped to an AZ
* It’s common to have a public subnet (public IP)
* It’s also common to have a private subnet (private IP)
* It’s common to have many subnets per AZ

Public Subnet

* Load Balancers
* Static Websites
* Files
* Public Authentication Layers

Private Subnet

* Web Application Servers
* Databases

-- Public and private subnets can communicate if they’re are in the same VPC

AWS VPC Brain Dump

* VPC & Regions aren’t much asked at the developer associate exam
* All new accounts come with a default VPC
* It’s possible to use a VPN to connect to a VPC (and access all the private IP straight from your laptop)
* VPC Flow Logs allow you to monitor the traffic within , in and out of youVPC (useful for security performance audit)
* VPC are per Account per Region
* Subnets are per VPC per AZ
* Some AWS resources can be deployed in VPC while others can’t
* You can peer VPC ( within or across accounts) to make it look like they’re part of the same network ( good for larger organizations)

The Three Tier Architecture for Web Applications Flow

* User access the alias of the website from route 53
* That then hits the public subnets elastic load balancer
* Within the Private subnet the data is then sent to the auto scaling group
* Which then spreads the requests to the different ec2 instances within an autoscaling group across multiple AZ sections
* These then hit the elasticache or rds depending on the elasticache pattern
* The DB with RDS with a cross AZ Replication for data protections

Amazon S3

* Amazon S3 is one of the main building blocks in AWS
* It’s advertised as “infinitely scaling” storage
* It’s widely popular and deserves its own section
* Many websites use AWS S3 as a backbone
* Many AWS services uses AWS S3 as an integration as well

Buckets (regional however all the names are unique in global)

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

S3 Object Overview

* Objects(files) have a key, that key is the full path
* There’s no concept of directories within buckets
  + (the ui will trick you into thinking there is)
* Just keys with very long names that contain slashes
* Object values are the content of the body:
  + Max size is 5TB
  + If uploading more than 5Gb must use multi-part upload
* Metadata (list of text key/value pairs - system or user metadata)
* Tags (Unicode key / value pair - up to 10) - useful for security / lifecycle
* Version ID (if versioning is enabled)

S3 -Versioning

* You can version your files in AWS s3
* It is enabled at the bucket level
* Same key overwrites will increment the “version”: 1, 2, 3
* It is best practice to version your buckets
  + Protect against unintended deletes ability to restore a version
  + Easy roll back to previous version
* Any file that is not versioned prior to enabling versioning will have version “null”

S3 Encryption for Objects (Exam loves this section)

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

SSE-S3

* SSE-S3: encryption using keys handled & managed by AWS S3
* Object is encrypte server side
* AES-256 encryption type
* Must set header: “x-a,z-server-side-encryption”: “AES256”
* Process
  + HTTP/S request with header
  + S3 receive object and given a s3 managed data key
  + Then will be put into the s3 bucket encrypted

SSE-KMS

* SSE-KMS: encryption using keys handled & managed by AWS KMS
* KMS Advantages: More finite control for the user and an audit trail
* Object is encrypted server side
* Must set header: “x-amz-server-side-encryption”:”aws:kms”
* Process
  + HTTP/S + Header request
  + Object is stored with a key from KMS a customer master key CMK
  + Put into the bucket encrypted

SSE-C

* SSE-C: server side encryption using dta keys fully managed by the customer outside of AWS
* Amazon s3 does not store the encryption key you provide
* HTTPS must be used
* Encryption key must provided in HTTP headers, for every HTTP request made
* Process
  + Client sends object and the client side data key over HTTPS with data key in header
  + The object is then encrypted and stored in the bucket and aws removes the client provided key.
  + This means the key is not stored with the object when using SSE-C

Client Side Encryption

* Client library such as the Amazon s3 encryption client
* Clients must encrypt data themselves before sending to S3
* Clients must decrypt data themselves when retrieving from s3
* Customers fully manages the keys and encryption cycle
* Process
  + Object is encrypted on the client and then sent via HTTP/S and stored in the AWS S3 bucket

Encryption in transit (SSL)

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

S3 Security

* User based
  + IAM policies - which API calls should be allowed for a specific user from IAM console
* Resource Based
  + Bucket Policies - bucket wide rules from the S3 console -allows cross account
  + Object Access Control List ( ACL) - finer grain
  + Bucket Access Control List (ACL) - less common

S3 Bucket Policies

* JSON based policies
  + Resources: buckets and objects
  + Actions: Set of API to allow or DENy
  + Effect: Allow / Deny
  + Principal: The account or user to apply the policy
* Use s3 bucket for policy to:
  + Grant public access to the bucket
  + Force objects to be encrypted at upload
  + Grant access to another account (Cross Account)

S3 Security - Other

* Networking:
  + Supports VPC Endpoints ( for instances in VPC without www internet)
* Logging and Audit:
  + S3 access logs can be stored in other s3 bucket
  + API calls can be logged in AWS Cloud Trail
* User Security:
  + MFA (multi factor authentication) can be required in versioned buckets to delete objects
  + Signed URLs: URLs that are valid only for a limited time (ex: premium video service for logged in users)

S3 Websites:

* S3 can host static websites and have them accessible on the www
* The website URL will be:
  + <bucket-name>.s3-website-<AWS-region>.amazonaws.com
  + OR (The difference is that it is either a - or . in between the .s3-website and <AWS-region>
  + <bucket-name>.s3-website.<AWS-region>.amazonaws.com
* If you get a 403 (Forbidden) error, make sure the bucket policy allows for public reads!
* In permssions and properties you need to allow for public facing s3 buckets

S3 others:

* Remember that bucket policies need time to refresh if you have made changes to the bucket policy, even up to 5 minutes

S3 CORS

* If you request data from another S3 bucket, you need to enable CORS
* Cross Origin Resource Sharing allows you to limit the number of websites that can request your files in S3 (and limit your costs)
* It’s a popular exam question
* Access-Control-Allow-Origin: <domain>

AWS S3 - Consistency Model

* Read after write consistency for PUTS of new objects
  + As soon as an object is written, we can retrieve it (ex PUT 200 -> GET 200)
  + This is true, except if we did a GET before to see if the object existed ex: (GET 101 -> PUT 200 -> GET 404) -- eventually consistent
    - The response from the previous get request is cached and will be returned first because the PUT has not reached eventual consistency
* Eventual Consistency for DELETES and PUTS of existing objects
  + If we read an object after updating, we might get the older version ex:(PUT 200 -> PUT 200 -> GET 200 (might be older version)
  + If we delete an object we might still be able to retrieve it for a short time ex: (DELETE 200 -> GET 200)

AWS S3 Performance -Key Names

**Historic fact and current exam**

* When you had > 100 TPS (transactions per second), S3 performance could degrade
* Behind the scene, each object goes to an S3 partition and for the best performance, we want the highest partition distributions
* In the exam, and historically, it was recommended to have random characters in front of youkey name to optimise performance:
  + <my\_bucket>/5r4d\_my\_folder/my\_file1.txt
  + <my\_bucket>/a01e\_my\_folder/my\_file2.txt
* It was recommended never to use dates to prefix keys as well because the partitions will be very similar to each other which would degrade performance

AWS S3 Performance - Key Names Current performance (not yet exam)

LINK: <https://aws.amazon.com/about-aws/what-new/2018/07/amazon-s3-announces-increased-request-rate-performance/>

* As of July 17th 2018, we can scale up to 3500 RPS for PUT and 5500 RPS for GET for each prefix
* “This S3 request rat4e performance increase removes any previous guidance to randomize object prefixes to achieve faster performances”
* It’s a “good to know”, until the exam gets updated

AWS S3 Performance

* Faster upload of large objects (>= 100MB), use multipart upload:
  + Parallelism PUTs for greater throughput
  + Maximize your network bandwidth and efficiency
  + Decrease time to retry in case a part fails
  + MUST use multi-part if object size is greater than 5GB
* Use CloudFront to cache S3 objects around the world (improves reads)
* S3 Transfers Acceleration (uses edge locations) -- just need to change the endpoint you write to , not the code
* If using SSE-KMS encryption yo may be limited to your AWS limits for KMS usage ( ~100s – 1000s downloads / uploads per second)

S3 & Glacier Select ( only available via code)

* Note: “Glacier” is another S3 storage tier for long term archival
* If yo retrieve data in S3 and Glacier you may want only a subset of it
* If you retrieve all the data, the network cost may be high!
* With S3 Select / Glacier Select, you can use SWL SELECT queries to let S3 or Glacier know exactly which attributes / filters you want
  + Select \* from s3object s where s.\”Country (NAME)\” like ‘%United State%’
* Save cost up to 80% and increase performance by up to 400%!
* The “SELECT” happens “within” S3 or Glacier
* Works with files in CSV, JSON or Parquet format
* Files can be compressed with GZIP or BSZIP2
* No subqueries or JOins are supported

Developing on AWS

Section Introductions

* So fare, we’ve interacts with service manually and they exposed standard information for clients:
  + EC2 exposes a standard linux machine we can use any way we want
  + RDS exposes a standard database we can connect to using a URL
  + ElastiCache exposes a cache URL we can connect to using URL
  + ASG / ELB are automated and we don’t have to program against them
  + Route53 was setup manual
* Developing against AWS has two components
  + How to perform interactions with AWS without the online console
  + How to interact with AWS proprietary services (S3, Dynamo, etc)
* Developing and performing aws tasks against aws can be done in several ways
  + Using the AWS CLI on our local computer
  + Using the AWS CLI on our EC2 machines
  + Using the AWS SDK on our local computer
  + USing the AWS SDK on our EC2 machines
  + Using the AWS instance Metadata service for EC2
* How to do all these things right and most secure way adhering to best practices

Installing AWS CLI Install on MAC : <https://docs.aws.amazon.com/cli/latest/userguide/install-cliv1.html>

AWS CLI Configuration

* Do not share you AWS Access Key and Secret Key with anyone!
* Get security credentials from IAM
* Terminal → aws configure this will prompt you for keys for access
* Ls ~/.aws → this is where the files with the info are kept

AWS CLI Practice for S3

AWS Elastic Beanstalk: (One of the more difficult Exam sections)

Developer Problems on AWS

* Managing Infrastructure
* Deploying Code
* Configuring all the databases, load balances, auto scaling groups, etc.
* Scaling concerns
* Most web apps have the same architecture (ALB + ASG)
* ALl the developers want is for their code to run!
* Possibly , consistently across different applications and environments

AWS Elastic BeanStalk Overview

* Elastic Bean Stalk is a developer centric view of deploying an application on aws
* It uses all the components we’ve seen before
* But it’s all in one view that’s easy to make sense of
* We still have full control over the configurations
* Elastic BeanStalk is free and you only get charge for the underlying services

Elastic Beanstalk

* Managed service
  + Instance configuration / OS is handled by beanstalk
  + Deployment strategy is configurable but performed by ElasticBeanStalk
* Just the application code is the responsibility of the developer
* Three architecture models
  + Single Instance deployment: good for dev
  + LB + ASG: Great for production or pre-production web applications
  + ASG only: great for non-web apps in production ( workers, etc.)
* Three components
  + Application
  + Application version: each deployment gets assigned a version
  + Environment name ( dev, test, prod….) free naming
* You deploy application versions to environments and can promote application version to the next environment
* Rollback feature to previous application versions
* Full control over life cycle of environments
* Support for many platforms
  + GO
  + Java SE
  + Java with Tomcat
  + .Net on windows server with IIS
  + Node.js
  + PHP
  + Python
  + Ruby
  + Packer Builder
  + Single Container Docker
  + Multi Container Docker
  + Preconfigured Docker
* If not not supported you can write your custom platform (advanced and is not expect of this certification level)

Elastic Beanstalk Deployment Modes

* Single Instance Great for Dev (All in one AZ)
* High Availability with Load Balancer Great for Prod

Beanstalk Deployment Options for Updates

* All at once ( deploy all in one go) - fastest, but instances aren’t available to serve traffic for a bit (downtime)
* rolling : update a few instances at a time (bucket), and then move onto the net bucket once the first bucket is healthy.
* Rolling with additional batches: like rolling, but spins up new instances to move the batch (so that the old application is still available)
* Immutable: spins up new instances in a new ASG, deploys versions to these instances,m and then swaps all the instances when everything is healthy.

Elastic Beanstalk Deployment All at once:

* Process
  + 4 instances at v1
  + Stop all the instances (deployment)
  + Upload the changes and deploy 4 v2s
* Fastest deployment
* Application has downtime
* Great for quick iterations in development environment
* No additional cost

Elastic Beanstalk Deployment Rolling

* Process
  + 4 instances on v1
  + 2 instances are stopped
  + 2 instances of v2 are deployed
  + Then rinse and repeat until all instances are v2
* Application is running below capacity
* Can Set the bucket size
* Application is running both versions simultaneously
* No additional cost

Elastic Beanstalk Deployment Rolling with additional batches

* Application is running at capacity can set the buckets size ( Some times running with larger than capacity)
* Can set the bucket size
* Applications are running bot versions simultaneously
* Small additional cost
* Additional batch is removed at the end of the deployment
* Longer deployment
* Good for Prod

Elastic Beanstalk Deployment Immutable

* Zero downtime
* New Code is deployed to new instances on a temporary ASG
* High Cost, double capacity
* Longest Deployment
* Quick rollback in case of failures (just terminate new ASG)
* Great for Prod
* Process:
  + Sprint up a separate ASG with all of the v2 instances
  + Then combine them into the current Asg
  + Then terminate the old ASG of v1 instances

Elastic BeanStalk Deployment Blue/Green Deployment

* Not a “direct feature” of Elastic Beanstalk
* Zero downtime and release facility
* Create a new “stage” environment and deploy v2 instances there
* The new environment (gree) can be validated independently and roll back if issues
* Route 53 can be setup using weighted policies to redirect a little bit of traffic to the stage environment for testing
* Using Beanstalk, “swap URLs” when done with the environmental test
* Manual not provided
* Process
  + Two environment with the different version instances
  + Then using route53 we redirects a percentage of the traffic to the new environment
  + Then when happy you switch everything to the new environment
* <https://docs.aws.amsazon.com/elasticbeanstalk/latest/dg/using-features.deploy-existing-version.html>

Elastic Beanstalk Extensions

* A zip file containing our code must be deployed to Elastic Beanstalk
* All the parameters set in the UI can be configured with code using files
* Requirements:
  + In the .ebextensions/ directory in the root of source code
  + YAML / JSON format
  + .config extensions (example: logging.config)
  + Able to modify some default settings using: option\_settings
  + Ability to add resources such as RDS, ElastiCache, DynamoDB, etc…
* Resources managed by .ebextensions get deleted if the environment goes away

Elastic Beanstalk CLI

* We can install an additional CLI called the EB cli which makes working with Beanstalk from the CLI easier
* It’s helpful for your deployment pipelines for elastic beanstalk

Elastic Beanstalk : Under the hood, Elastic Beanstalk relies on cloudformation

ElasticBeanStalk Deployment mechanism

* Describe dependencies ( requiremetns.txt for python, package.json for Node.js)
* Package code as zip
* Zip file is uploaded to each EC2 machine
* Each EC2 machine resolves dependencies (SLOW)
* Optimization in case of long deployments: package dependencies with source code to improve deployment performance speed.

Additional Elastic Beanstlak Exam Tips

* Beanstalk with HTTPS
  + Idea: Load the SSL certificate onto the Load Balancer
  + Can be done from the Console ( EB console, load balancer configuration)
  + Can be done from the code .ebextensions/securelistener-alb.config
  + SSL Certificate can be provisioned using ACM (AWS certificate manager) or CLI
  + Must configure a security group rule to allow incoming port 443 (HTTPS port)
* Beanstalk redirect HTTP to HTTPS
  + Configure your instances to redirect HTTP to HTTPS:
  + <https://github.com/awsdocs/elastic-beanstalk-samples/tree/master/configuration-files/aws-provided/security-configuration/http-redirect>
  + OR configure the Application Load Balancer (ALB only ) with a rule
  + Make sure health checks are not redirected (so they keep giving 200 OK)
* Beanstalk Lifecycle Policy
  + Elastic Beanstalk can store at most 1000 application versions
  + If you don’t remove old versions, you won’t be able to deploy anymore
  + To phase out old application versions, use a lifecycle policy
    - Based on time (old versions are removed)
    - Based on space ( when you have too many versions)
  + Versions that are currently used won’t be deleted
  + Option not to delete the source bundle in S3 to prevent data loss
* Web Server vs Worker Environment
  + If your application performs tasks that are long to complete offload these tasks to a dedicated worker environment
  + Decoupling your application into two tiers is common
  + Example: processing a video, generating a zip file, etc
  + You can define periodic tasks in a file cron.yaml
  + Common Pattern
    - Users hit web tier in beanstalk which is ec2 and elb
    - The ec2 inserts work into a SQS work queue
    - The process work is then transferred to the worker tier in beanstalk which is the sqs and ec2
* RDS with Elastic Beanstalk
  + RDS can be provisioned with Beanstalk, which is great for dev / test
    - This is not great for prod as the database lifecycle is tied to the beanstalk environment lifecycle
    - The best for prod is to separately create an RDS database and provide our WEB application with the connection string
  + Steps to migrate from RDS coupled in EB to standalone RDS
    - Take an RDS DB snapshot
    - Enable deletion protection in RDS
    - Create a new environment without an RDS point to existing old RDS
    - Perform blue/green deployment and swap the new and old environments
    - Terminate the old environment (RDS won’t get deleted thanks to the protection)
    - Delete CloudFormation stack ( will be in DELETE\_FAILED state)

CICD AWS

* The more manual steps there are in a setup process increases the chances of errors occurring.
* What we should be doing is pushing our code that is in a repository straight to aws and have it deployed onto the network
  + Automatically
  + The Right Way
  + Making sure it is tested before it is deployed
  + With possibility to go into different states (dev, test, pre-prod, prod)
* Therefore to be a proper AWS Developer…. We need to learn AWS CICD

The Developer exam has an entire section on CICD and is incredibly important to know

CICD steps

* Dev pushes to repository
* A testing / build server checks the code as soon as it is pushed
* The developer gets feedback about the tests and checks that have passed/failed
  + Find bugs early, fix bugs early
  + Deliver faster as the code is tested
  + Deploy Often
  + Happier developers, as they’re are unblocked from DevOps work
* Deployment Server that will change the servers and applications from version x to version y

Continuous Delivery:

* Ensure that the software can be released reliably whenever needed
* Ensures deployments happen often and quick
* Shift away from “one release every 3 months” to “5 releases a day”
* That usually means automated deployment
  + CodeDeploy

Technology Stack for CICD

* Code → Build → Test → Deploy → Provision
* AWS CodeCommit → AWS CodeBuild → AWS Elastic Beanstalk/AWS CodeDeploy (for lower level deployment of aws resources)
* The orchestrator of these tests is **AWS CodePipeline**

CodeCommit:

* Version Control: is the ability to understand the various changes that happened to the code over time ( and possibly roll back).
* All these are enabled by using a version control system such as Git
* A Git repository can live on one’s machine, but is usually lives on a central online repository:
* Benefits are:
  + Collaborate with other developers
  + Make sure the code is backed up somewhere
  + Makes sure its fully viewable and auditable

CodeCommit

* Git repositories can be expensive:
* The industry includes
  + Github: Free public and paid private repositories
  + BitBucket
  + Etc.
* And AWS CodeCommit
  + Private git repositories
  + No size limit on repositories (scale seamlessly)
  + Fully managed, highly available
  + Code only in AWS Cloud account => increased security and compliance
  + Secure
  + Integrated with Jenkins/ codebuild/ other cicd tools

CodeCommit Security

* Interactions are done using Git
* Authentication in Git:
  + SSH Keys: AWS USers can configure SSH keys in their IAM console
  + HTTPS: Done through the AWS CLI Authentication helper or generating HTTPS credentials
  + MFA (multi factor authentication) can be enabled for extra safely
* Authorization in Git:
  + IAM policies manage user / roles rights to repositories
* Encryption:
  + Responsible are automatically encrypted at rest using KMS
  + Encrypted in transit (can only use HTTPS or SSH -- both secure)
* Cross Account Access:
  + DO NOT SHARE YOUR SSH KEYS
  + DO NOT SHARE YOUR AWS CREDENTIALS
  + Use IAM Role in you AWS Account and use AWS STS(Security Token Service) (with AssumeRole API)

CodeCommit vs GitHub

* **Similarities**
  + Both are git repositories
  + Both support code review process (git pull requests)
  + GitHub and CodeCommit can be integrated with AWS CodeBuild
  + Both support HTTPS and SSH method of authentication
* **Differences**
  + Security:
    - GitHub: GitHub Users
    - CodeCommit AWS IAM users & Roles
  + Hosted
    - GitHub: hosted by github
    - GitHub Enterprise: Self hosted on you servers
    - CodeCommit: Managed & and hosted by AWS
  + UI:
    - GitHub is fully featured
    - CodeCommit UI is minimal

CodeCommit Notifications

* You can trigger notifications in CodeCommit using **AWS SNS (simple notification service)**  or **AWS Lambda** or **AWS CloudWatch Event Rules**
* Use cases for notifications SNS/AWS Lambda notifications
  + Deletion of branches
  + Trigger for pushes that happen in master branch
  + Notify external build system
  + Trigger aws Lambda function to perform codebase anaylsis ( maybe credentials got commited in the code)
* Use cases for CloudWatch Event Rules:
  + Trigger for pull requests updates ( created / updated/ deleted/ commented)
  + Commit comment events
  + CloudWatch Event Rules goes into a SNS topic.

CodePipeline

* Continuous Delivery
* Visual Workflow
* Source: GItHub/CodeCommit/ Amazon S3
* Build: CodeBuild/Jenkins/ etc…
* LoadTesting: 3rd party tools
* Deploy: AWS CodeDeploy / Beanstalk/ CloudFormation/ ECS….
* Made of Stages:
  + Each stage can have sequential action and / or parallel actions
  + Stages examples: Build/ Test/ Deploy/ LoadTest/ etc…
  + Manual Approval can be defined at any stages.

AWS CodePipeline Artifacts

* Each pipeline stage can create “artifacts”
* Artifacts are passed stored in Amazon S3 and passed onto the next stage
* Trigger → CodeCommit (source) → output artifacts → s3 → input artifacts → CodeBuild(build) → output artifacts → s3 → input artifacts → Deploy (CodeDeploy) → Deploy

CodePipeline Troubleshoot

* CodePipeline state changes happen in AWS CloudWatch Events, which can in return create SNS notifications
  + You can create events for failed pipelines
  + You can create events for cancelled stages
* If CodePipeline fails a stage, your pipeline stops and you can get information in the console
* AWS CloudTrail can be used to audit AWS API calls
* If Pipeline can’t perform an action, make sure the “IAM Service Role” attached does have enough permissions (IAM Policy)

CodeBuild:

* Fully managed build service
* ALternative to other build tools such as jenkins
* Continuous scaling (no servers to manage or provisions - no build queue)
* Pay for usage: the time it takes to complete the builds
* Leverages Docker under the hood for reproducible builds
* Possibility to extend the capabilities leveraging our own base Docker images
* Secure: Integration with KMS for encryption of build artifacts, IAM for build permissions, and VPC for network security, CloutTrail for API call logging.

CodeBuild Overview:

* Source Code from a code repository
* Build instructions can be defined in code
* Output logs to S3 & AWS CloudWatch logs
* Metrics to monitor CodeBuild statistics
* Use CloudWatch Alarms to detect failed builds and trigger notification
* CloudWatch Events / AWS Lambda as glue
* SNS notifications
* Ability to reproduce CodeBuild locally to troubleshoot in case of errors
* Builds can be defined within codepipeline or codebuild itself

CodeBuild Support Environments

* Java
* Ruby
* Python
* Go
* Node.js
* Android
* .NET core
* PHP
* Docker: extend any environment you like.

How CodeBuild Works

* Source Code: CodeCommit -- Buildspec.yml
* Build Docker Image -- Docker file
* Then both are placed into a code build container (docker) then it runs the instructions from the buildspec.yml
* Optional AWS S3 cache bucket
* AWS s3 bucket Artifacts
* Save Logs: AWS CloudWatch AWS S3

Buildspec.yml

* Must be at the root of your code
* Define environment variables
* Plaintext variables
* Secure secrets use SSM parameter store
* Phases
  + Install: install dependencies you may need for you build
  + Pre build: final commands to execute before build
  + Build: actual build commands
  + Post Build: Finishing touches (zip output for example
* Artifcats: what to upload to s3 (encrypted KMS)
* Cache: files to cache ( usually dependencies ) to S3 for future build speed up

CodeBuild Local Build

* In case of need of deep troubleshooting beyond logs…
* You can run CodeBuild locally on your desktop (after installing)
* For this, leverage the CodeBuild Agent
* https://docs. aws.amazon.com/codebuild/latest/userguide/use-codebuild-agent.html

AWS CodeDeploy

* We want to deploy our application automatically to many EC2 instances
* These instances are not managed by Elastic Beanstalk
* There are several ways to handle deployments using open source tools( Anisble, Terrafor, Chef, Puppet, etc…)
* We can use the managed service AWS CodeDeploy

AWS CodeDeploy - Steps to make it work

* Each EC2 machine ( or on premise machine) must be running the CodeDeploy Agent
* The Agent is continuously polling AWS CodeDeploy for work to do
* CodeDeploy sends appspec.yml
* Application is pulled from GitHub or S3
* EC2 will run the deployment instructions
* CodeDeploy Agent will report of success/failure of deployment on the instance
* Process:
  + Source code with appspec.yml file
  + Push to a git repository
  + Trigger a deployment with codeDeploy
  + The EC2 instances are polling to see if a deployment was triggered
  + If they do see it they will download the code and appspec.yml file from the repository

AWS CodeDeploy – other

* EC2 instances are grouped by deployment group (dev/test/prod)
* Lots of flexibility to define any kind of deployments
* CodeDeploy can be chained into CodePipeline and use artifacts from there
* CodeDeploy can re-use existing setup tools, works with any application, auto scaling integration
* Note: blue/green only workswith ec2 instances (not on premise)
* Supports for AWS Lambda deployment
* CodeDeploy does not provision resources

AWS CodeDeploy Primary Components

* Application: unique name
* Compute platform: EC2/ On-premise or Lambda
* Deployment configuration: Deployment rules for success / failures
  + EC2/On-premise: you can specify the minimum number of healthy instances for the deployment
  + AWS Lambda: specify how traffic is routed to your updated Lambda function versions
* Deployment Group: group of tagged instances (allows to deploy gradually)
* Deployment type: In-place deployment or blue/green deployment:
* IAM instance profile: need to give #C2 the permissions to pull from S3 / GitHub
* Application Revions: application code + appspec.yml file
* Service Role: role for codedeploy to perform what it needs
* Target revision:Target deployment application version

AWS CodeDeploy AppSpec

* File section: how to source and copy from S3/GitHub to filesystem
* Hooks: set of instructions to do to deploy the new version (hooks can have timeouts). The order is:
  + ApplicationStop : To stop the applications
  + DownloadBundle : How to download my new application
  + BeforeInstall: Prep before Installing
  + Install -- no control --
  + AfterInstall: Things to do after install
  + ApplicationStart : Things to do to start application
  + ValidateService: really important : Steps to check if the deployment was successful

AWS CodeDeploy Deployment Config

* Configs:
  + One a time: one instance at a time, one instance => deployment stops
  + Half ag a time: 50%
  + All at once: quick but no healthy host, downtime, good for dev
  + Custom: min healthy host = 75%
* Failures:
  + Instances stay in “failed state”
  + New deployments will first be deployed to “failed state” instances
  + To roll back: rdeploy old deployment or enable automated rollback for failures
* Deployment Targets:
  + Set of EC2 instances with tags
  + Directly to an ASG
  + Mix of ASG/ Tags so you can build deployment segments
  + Customization in scripts with DEPLOYMENT\_GROUP\_NAME environment variables

CodeStar -- A Wrapper to provide a dashboard for all of the deployment cycle

* CodeStar is an integrated solution that regroups: GitHub, CodeCommit, CodeBuild, CodeDeploy, CloudFormation, codePipeline, CloudWatch
* Helps quickly create”CICD-ready” projects for EC2, Lambda, Beanstalk
* Supported Languages: c#, go, html5, java, node, php, python, ruby
* Issue tracking integration with jira and github issues
* Ability to integrate with cloud9 to obtain a web IDE ( not all regions)
* One dashboard to view all your components
* Free Service pay only for the underlying usage of other services
* Limited Customization
* Orchestrated: By code pipeline
  + Source
  + Build
  + Test
  + Deploy
  + Monitoring

AWS CloudFormation:

* CloudFormation is a declarative way of outlining your AWS Infrastructure, for any resources (most of them are supported).
* Cloud Template you can say:
  + I want a security group
  + I want to EC2 instances using this security group
  + I want two Elastic IPs for these EC2 machines
  + I want an s3 bucket
  + I want a load balancer (ELB) in front of these machines
* Then Cloud Formation creates those for you, in the **right order,** with the exact configuration that you specify

Benefits of AWS CloudFormation:

* Infrastructure as code
  + No resources are manually created, which is excellent for control
  + The code can be version controlled for example using git
  + Changes to the infrastructure are reviewed through code
* Costs
  + Each resource within the stack is tagged with an identifier so you can easily see how much each stack costs you
  + You can estimate the costs of your resources using the CloudFormation template.
  + Savings Strategy:
    - In Dev, you could automation deletion of templates at 5 PM and recreated at 8 AM, Safety.

Benefits of AWS CloudFormation

* Productivity
  + Ability to destroy and re-create an infrastructure on the cloud on the fly
  + Automated generation of Diagram for your templates!
  + Declarative programming ( no need to figure out ordering and orchestration)
* Separation of concern: create many stacks for many apps and many layers. Ex:
  + VPC stacks
  + Network Stacks
  + App Stacks
* Don’t re-invent the wheel
  + Leverage existing templates on the web
  + Leverage the documentation

How Cloud Formation Works

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

Deploy CloudFormation:

* Manual way:
  + Editing templates in the CloudFormation Designer
  + Using the console to input parameters, etc
* Automated Way:
  + Editing templates in a YAML file
  + Using the aws CLI (command line interface) to deploy the templates
  + Recommended way when you fully want to automate your flow.
* CloudFormation Building Blocks
  + Templates components (one course section for each):
    - Resources: your AWS resources declared in the template (MANDATORY)
    - Parameters: the dynamic inputs for yourtemplate
    - Mappings: the static variable fo your template
    - Outputs: References to what has been created
    - Conditionals: List of conditions to perform resource creation
    - Metadata
  + Template Helppers:
    - References
    - Function

YAML Crash Course

* YAML and JSON are the languages you can use for CloudFormation
* JSON is horrible for Cloud Formation
* YAML is great in so many ways
* Key Value Pairs
* Nested Objects
* Supports Array (indicated as a hyphen)
* Multiline strings
* Can include comments

Resources

* Resources are the core of your CloudFormation template (MANDATORY)
* They represent the different AWS Components that will be created and configured
* Resources are declared and can reference each other
* AWS figures out creation, updates, and deletes the resources for us
* There are over 224 types of resources
* Resources types are identifiers are of the form:
  + AWS: :aws-product-name: :data-type-name
* All the resources can be found here:
* <http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resources-type-ref.html>

FAQ:

* Can I create a dynamic amount of resources
  + No, you can’t. Everything in the CloudFormation template has to be declared. You can’t perform code generation there
* Is every AWS service supported
  + Almost. Only a select few niches are not there yet
  + You can work around that using AWS Lambda Custom Resources

Parameters:

* Parameters are a way to provide inputs to your AWS CloudFormation template
* They’re important to know about if.
  + You want to reuse your templates across the company
  + Some inputs can not be determined ahead of time
* When should I use a parameter?
  + Is the CloudFormation resource configuration likely to change in the future?
  + Then turn it into a parameter because we can change the content without needing to reupload the CloudFormation template
* Settings
  + Type
    - String
    - NUmber
    - Comma Delimited List
    - List <type>
    - AWS Parameter (to help catch invalid values -- match against existing values in the AWS account)
  + Description
  + Constraints
  + Constraint Description
  + Min/MaxLength
  + Min/MaxValue
  + Defaults
  + AllowedValues (array)
  + AllowedPatterns (Regex)
  + NoEcho (boolean) for private values
* How to use a paramerter?
* **A Reference Parameter**
  + The Fn::Ref function can be leveraged to reference parameters
  + Parameters can be used anywhere in a template
  + The shorthand for this in YAML is !Ref
  + The function can be used to reference other element within the template
* Psudedo Parameters
  + AWS offers us pseudo parameters in any CloudFormation template.
  + These can be used at any time and are enabled by default

Mappings

* Mappings are fixed variables within CloudFormation Template
* They’re very handy to differentiate between different environments (dev vs. prod), regions (AWS regions), AMI types, etc.

Mapping vs Parameters

* Mapping are great when you know in advance all the values that can be taken and that they can be deduced from variables such as:
  + Region
  + AZs
  + AWS Account
  + Environment (dev vs prod)
  + Etc…
* They allow for safer control over the template
* Parameters are great when if the resource is undecided or may be changed by the user.

FN::FindINMap: -- Accessing Mapping Values

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

Outputs:

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

Template:

Outputs:

NameOfResourceGroup:

Description:

Value:

Export:

Name:

Importing: Cross Stack Reference

* FN::ImportValue (Short hand !ImportValue)

What are conditions

* Conditions are used to control the creation of resources or outputs based on a condition.
* Conditions can be whatever you want them to be, but common ones are:
  + Enviornement (dev/test/prod)
  + AWS Region
  + Any parameter value
* Each condition can reference another condition, parameter value or mapping
* Conditions:
  + CreateProdResources: !Equals [ !Ref EnvType, prod]
* How to:
  + The logical function
    - Fn::And
    - Fn::Equals
    - Fn::If
    - Fn::Not
    - Fn::Or
* Resources:
  + MountPoint:
    - Type: “AWS:: EC2::VolumeAttachment”
    - Condition: CreateProdResources (True or False)

CloudFormation Muse Know Intrinsic Functions

* Fn::Ref
  + The function can be leveraged to reference
    - Parameter => returns the value of the parameter
    - Resources => returns the physical ID of the underlying resource
  + The shorthand for this in YAML is !Ref
* Fn::GetAtt (When you need a specific resource attribute value)
  + Attributes are attached to any resources you create
  + To know the attributes of you resources the best place to look is at the documentation
  + AZ of an EC2 machine
* Fn::FindInMap
  + We use Fn::FIndInMap to return a named value from a specific key
  + !FindInMap [ MapName, TopLevelKey, SecondLevelKey ]
* Fn::ImportValue
  + Import values that are exported in other templates
* Fn::Join
  + Join values with a delimiter
  + !Join [ delimiter, [comma-delimited list of values ] ]
  + This creates a:b:c
  + !join [ “:”, [a,b,c] ]]
* Fn::Sub
  + Is used to substitute variables from a text. It’s a very handy function that will allow you to fully customize you templates
  + For example you can combine Fn::Sub with references or AWS pseudo variables
  + String must contain ${VariableName} and will subsititue them
  + !Sub
    - -String
    - - { varName: Var1Value, VarName2: Var2Value
* All of the Condition Functions (Logic Operators)

CloudFormation Rollbacks

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

AWS Monitoring, Troubleshooting & Auditing

--CloudWatch, X-Ray, and CloudTrail--

Why Monitoring is Important

* We know how to deploy applications
  + Safely
  + Automatically
  + Using infrastructure as code
  + Leveraging the best AWS components
* Our applications are deployed, and our users don’t care how we did it…
* Our Users only care that the application is working!
  + Application latency: will it increase over time
  + Application outages: cusotmer experience should not be degraded
  + Users contacting the IT department or complaining is not a good outcome
  + Troubleshooting and remediation
* Internal monitoring:
  + Can we prevent issues before they happen?
  + Performance and cost
  + Trends (scaling patterns)
  + Learning and Improvement

Monitoring in AWS

* AWS CloudWatch:
  + Metrics: collect and track key metrics
  + Logs: Collect, montor, analyze and store log files
  + Events: Send notifications when certain events happen in your AWS
  + Alarms: ract in real-time to metrics/evnets
* AWS X-Ray:
  + Troubleshooting application performance and errors
  + Distributed tracing of microservice
* AWS Cloudtrail:
  + Internal monitoring of API calls being made
  + Audit changes to AWS Resources by your users

AWS CloudWatch Metrics

* CloudWatch provides metrics for every services in AWS
* Metric is a variable to monitor (CPUUtilization, Networking etc…)
* Metrics belong to namespaces
* Dimensions is an attribute of a metric (instance id, environments, etc…)
* UP to 10 dimensions per metric
* Metrics have timestamps
* Can create cloudWatch dashboards of metrics

AWS CloudWatch EC2 Detailed Monitoring

* EC2 instances metrics have metrics “every 5 minutes”
* With detailed monitoring (for a cost), you get data “every 1 minute”
* Use detailed monitoring if you want to more promptly scale you ASG!
* The AWS Free Tier allows us to have 10 detailed monitoring metrics
* Note: EC2 Memory usage is by default not pushed ( must be pushed from inside the instance as a custom metric)

AWS CloudWatch Custom Metrics

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

AWS CloudWatch Metrics

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

AWS CloudWatch Logs

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

AWS CloudWatch Events

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

AWS X-Ray

* Debugging in production, the good old way:
  + Test locally
  + Add log statement everywhere
  + Re-deploy in production
* Log formats differ across applications using CloudWatch and analytics is hard.
* Debugging: monolith “easy”, distributed service “hard”
* No common views of your entire architecture!
* This is where xray shines

AWS X-Ray: Visual analysis of our applications

* It provide visual tracing of errors

AWS X-Ray advantages

* Troubleshooting performance (bottlenecks)
* Understand dependencies in a microservice architecture
* Pinpoint service issues
* Review request behavior
* Find errors and exceptions
* Are we meeting time SLA?
* Where I am throttled?
* Identify users that are impacted

AWS X-RAY Compatibility:

* AWS Lambda
* Elastic Beanstalk
* ECS
* ELB
* API Gateway
* EC2 Instances or any application server ( even on premise)

AWS X-Ray Leverages Tracing

* Tracing is an end to end way to following a “request”
* Each component dealing with the request adds its own “trace”
* Tracing is made of segments ( + sub segments)
* Annotations can be added to traces to provide extra-information
* Ability to trace:
  + Every request
  + Sample request ( as a % for example or a rate per minute
* X-Ray security:
  + IAM for authorization
  + KMS for encryption at rest

AWS X-Ray

How to enable it?

1. Your code ( java, python, go, node, .net) must import the aws x-ray sdk
   1. Very little code modification is needed
   2. The application SDK will then capture:
      1. Calls to AWS service
      2. HTTP/HTTPS requests
      3. Database Calls ( MySQL, PostgreSQL, DynamoDB)
      4. Queue calls (SQS)
2. Install the X-Ray daemon or enable X-Ray AWS Integration
   1. X-Ray dameon works as a low level UDP packet interceptor
   2. AWS Lambda / other AWS services already run the X-ray daemon for you
   3. Each application must have the IAM rights to write data to X-Ray

X-Ray magic

* X-Ray service collects data from all the different services
* Service map is computed from all the segments and traces
* X-Ray is graphical, so even non technical people can help troubleshoot

AWS X-Ray Troubleshooting

* If X-Ray is not working on EC2
  + Ensure the EC2 IAM Role has the proper permissions
  + Ensure the EC2 instances is running the X-Ray Daemon
* To enable on AWS Lambda:
  + Ensure it has an IAM execution role with the proper policy (AWSX-Ray WriteOnlyAccess)
  + Ensure that X-Ray is imported in the code

AWS X-Ray Additional Exam Tips ( ½)

* The X-Ray daemon /agent has a config to send traces cross account:
  + Make sure the IAM permissions are correct - the agent will assume the role
  + This allows to have a central account for all you application tracing
* Segments: each application / service will send them
* Trace: segments collected together to form an end-to-end trace
* Sampling: decrease the amount of request sent to X-Ray, reduce cost
* Annotations: Key Value pairs used to index traces and use with filters
* Metadata: Key Value pairs, **not** indexed not used for searching

AWS X-Ray Additional Exam Tips (2/2)

* Code must be instrumented to use the AWS X-Ray SDK ( interceptors, handlers, http clients)
* IAM role must be correct to send traces to X-Ray
* X-Ray on EC2 / On-Premise:
  + Linux Systems must run the X-Ray Daemon
  + IAM instance role if EC2, other AWS credentials on on premise instance
* X-Ray on Lambda:
  + Make sure x-ray intetgration is ticked on lambda ( Lmabda runs the daemon)
  + IAM role is lambda role
* X-Ray on beanstalk:
  + Set configuration on EB console
  + Or use a beanstalk extension (.ebextensions/xray-daemon.config)
* X-Ray on ECS / EKS / Fargate (Docker):
  + Create a Docker image that runs the Daemon / or use the official X-Ray Docker image
  + Ensure port mappings & mappings & network settings are correct and IAM task roles are defined

AWS Cloudtrail

* Provides governance, compliance and audit for your aws account
* Cloudtrail is enabled by default
* Get an history of events / api calls made within you aws account by:
  + Console
  + SDK
  + CLI
  + AWS Services
* Can put logs from CloudTrail into CloudWatch Logs
* If a resource is deleted in AWS, look into CloudTrail first!

AWS Cloudtrail vs AWS Cloudwatch vs AWS X-Ray

* CloudTrail
  + Audit api calls made by users /services/ aws console
  + Useful to detec unauthorized calls or root cause of changes
* CloudWatch
  + Cloudwatch metrics over time for monitoring
  + Cloudwatch logs for storing application log
  + Cloudwatch alarms to send notifications in case of unexpected metrics
* X-Ray
  + Automated trace analysis and central service map visualization
  + Latency , Errors , Fault Analysis
  + Request tracking across distributed systems

AWS integration and Messaging ( SQS, SNS and Kinesis )

* When we start deploying multiple applications, they will inevitably need to communicate with each other
* There are two patterns of application communication
  + Synchronous communications (application to application)
  + Asynchronous / Event Based (application to queue to application)
* Synchronous between applications can be problematic if there are sudden spikes of traffic
* What if you need to suddenly encode 1000 videos but usually it’s 10?
* In that case, it’s better to decouple your applications
  + Using SQS: queue model
  + Using SNS: pub/sub model
  + Using Kinesis: real-time streaming model
* These service can scale independently from our application

AWS SQS

What’s a queue?

* Take messages from producers (no limit to producers)
* Then consumers will take messages from the queue (no limit to consumers)

AWS SQS - Standard Queue

* Oldest offering (over 10 years old)
* Fully managed
* Scales from 1 message per second to 10k per second
* Default retention of messages: 4 days, maximum of 14 days
* No limit to how many messages can be in the queue
* Low latency ( < 10 ms on publish and receive)
* Horizontal scaling in terms of number of consumers
* Can have duplicate messages (at least once delivery , occasionally)
* Can have out of order messages ( best effort ordering)
* Limitation of 256KB per message sent

AWS - Delay Queue:

* Delay messages ( consumers don’t see it immediately) up to 15 minutes
* Default is 0 seconds ( messages are available right away)
* Can set a default at queue level
* Can override the default using the DelaySeconds Parameter

SQS Producing messages

* Define the boda up 256kb string
* Add message attributes (metadata -optional )
* Provide Delay Delivery (optional)
* Get Back
  + Message identifier
  + MD5 hash of the body

SQS -- consuming messages

* Consumers …
* Poll SQS for messages (receive up to 10 messages at a time)
* Process the message within the visibility timeout
* Delete the message using the message ID & receipt handle

SQS -- visibility timeout

* When a consumer polls a message from a queue, the message is “invisible” to the other consumers for a defined period … the visibility timeout:
  + Set between 0 seconds to 12 hours (default 30 seconds)
  + If too high ( 15 minutes) and consumers fails to process the message, you must wait a long time before processing the message again
  + If too low (30 seconds) and consumers needs time to process the message ( 2 minutes), another consumer will receive the message and the message will be processed more than once
* ChangeMessageVisibility API to change the visibility while processing a message
* Deletemessage API to tell SQS the message was successfully processed

AWS SQS – Dead Letter Queue

* If a consumer fails to process a message within the Visibility Timeout… the message goes back to the queue
* We can set a set a threshold of how many times a message can go back to the queue – it’s called a “redrive policy”
* After the threshold is exceeded, the message goes into a dead letter queue (DLQ)
* We have to create a DLQ first and then designate it dead letter queue
* Make sure to process the messages in the DLQ before they expire!

AWS SQS - Long Polling

* When a consumer requests message from the queue it can optionally “wait” for messages to arrive if there are none in the queue
* This is called Long Polling
* LongPolling decreases the number of API calls made to SQS while increasing the efficiency and latency of your application
* The wait time can be between 1 - 20 sec (20 sec preferable)
* Long Polling is preferable to short polling
* Long Polling can be enabled at the queue leave or at the API level using WaitTimeSeconds.

AWS SQS – FIFO Queue

* Newer offering (First in - First out) - not avaliable in all regions
* Name of the queue must end in .fifo
* Lower throughput (up to 3,000 per second with batching, 300/s without )
* Messages are processed in order by the consumer
* Messages are sent exactly once
* No per message delay ( only per queue delay )

SQS FIFO – Features

* Deduplication: ( not send the same message twice)
  + Provide a MessageDeduplicationId with your message
  + De-duplication interval is 5 minutes
  + Content based duplication : the MessageDeduplicationId is generated as the SHA-256 of the message body (not the attributes)
* Sequencing:
  + To ensure strict ordering between messages,m specify a MessageGroupId
  + Messages with different Group ID may be received out of order
  + E.g. to order messages for a user, you could use the user id as a group id
  + Messages with the same Group Id are delivered to one consumer at a time

SQS Extended Client

* Message size limit is 256KB, how to send large messages
* Using the SQS Extended clinet ( java library )
* Process:
  + Producer stores a item in a s3 bucket
  + Sends a small metadata message to sqs
  + The consumer then reads the small metadata message and then retrieves the item from amazon s3

AWS SQS Security

* Encryption in flight using the HTTPS endpoint
* Can enable SSE (Server Side Encryption) using KMS
  + Can set the CMK (custom master key ) we want to use
  + Can set the data key reuse period (between 1 min to 24 hours)
    - Lower and kms api will be used often
    - Higher and KMS API will be called less
  + SSE only encrypts the body, not the meta data ( message ID, timestamp and attributes)
* IAM policy must allow usage of SQS
* SQS queue access policy
  + Finer grained control over IP
  + Control over the time that requests come in
* No VPC Endpoint, must have internet access to access SQS

SQS Must know API

* CreateQueue and DeleteQueue
* PurgeQueue: delete all the messages in queue
* SendMessage, ReceiveMessage, DeleteMessage
* ChangeMessageVisibility: Change the timeout
* Batch APIs for SendMessage, DeleteMessage, ChangeMessageVisibility helps decrease your costs

AWS SNS

* What if you want to send one message to many receivers?
* Pub / Sub → SNS Topic
* Publisher sends a message to a SNS Topic and multiple subscribers can then be notified of the messages

AWS SNS

* The “event producer” only sends message to one SNS topic
* As many “event receivers” (subscriptions) as we want to listen to the SNS topic notifications
* Each subscriber to the topic will get all the messages ( note: new feature to filter messages)
* Up to 10,000,000 subscriptions per topic
* 100,000 topics limit
* Subscribers can be:
  + SQS
  + HTTP/HTTPS ( with delivery retires – how many times)
  + Lambda
  + Emails
  + Emails JSON
  + SMS messages
  + Mobile Notifications

SNS integrates with a lot of Amazon products

* Some services can send data directly to SNS for notifications
* CloudWatch(for alarms)
* Auto Scaling Group notfications
* Amazon S3 (on bucket events)
* CloudFormation ( upon state changes => failed to build, etc)
* Etc….

AWS SNS – How to publish

* Topic Publish ( within your aws server using the sdk)
  + Create a topic
  + Create a subscription ( or many)
  + Publish to the topic
* Direct Publish (for mobile apps SDK)
  + Create a platform application
  + Create a platform endpoint
  + Publish to the platform endpoint
  + Works with Google GCM, Apple APNS, Amazon ADM…

SNS + SQS: Fan out

* Push once in SNS receive in many SQS
* Process:
  + Publisher sends a message to an SNS topic
  + Multiple SQS are subscribed to the SNS topic
  + They then take that message and que it up for consumers to process them
* Fully decoupled
* No data lossAbility to add receivers of data laterSQS allows for delayed processing
* SQS allows for retrieds of work
* May have many workers on one queue and one worker on the other queue

AWS Kinesis Overview:

* Kinesis is a managed alternative to apache kafka
* Great for application logs, metrics, IOT, clicstreams
* Great for real time big data
* Great for streaming processing frameworks
* Data is automatically replicated to 3AZ
* Kinesis streams: low latency streaming ingest at scale
* Kinesis analytics: performining real time analytics on streams using sql
* Kinesis Firehose; load streams into S3, Redshift Elastic search

Kinesis

* Data is converted into amazon kinesis streams
* Streams are sent to amazon kinesis analytics
* Kinesis then sends this data via firehose to wherever you want to store data

Kinesis Streams Overview

* Streams are divided in order Shards / Partitions
  + To increase throughput increase shards
* Data retention is 1 day by default, can go up to 7days
* Ability to reprocess / replay data
* Multiple applications can consume the same stream
* Real time processing with scale of throughput
* Once data is inserted in kinesis it can’t be deleted ( immutability)

Kinesis streaming shards

* One stream is made of many different shards
* 1 mb/s or 1000 messages/s at write per shard
* 2 mb/s at read per shard
* Billing is per shard provisioned, can have as many shards as you want
* Batching is available or per message calls.
* The number of shards can evolve over time (reshard (add shards) / merge (reduces shards)
* Records are ordered per shard

AWS Kinesis API – Put Records

* Put Record API + Partition key that gets hashed
* The same key goes to the same partition ( helps with ordering for a specific key)
* Messages sent get a “sequence number”
* Choose a partition key that is highly distributed ( helps prevent “hot partition”)
  + User\_id if many users
  + Not country id if 90% of the users are in one country therefore all the data will be sent to one shard and that will cause a hot partition
* Use Batching with PutRecords to reduce costs and increased throughput
* ProvisionedThroughput Exceeded if we go over the limits
* Can useCLI AWS SDK or producer libraries from various frameworks

AWS Kinesis API – Exceptions

* ProvisionedThroughputexceeded Exceptions
  + Happens when sending more data ( exceeding MB/s or TPS for any shard)
  + Make sure you don’t have a hot shard ( such as your partition key is bad and too much data goes to that partition)
* Solution:
  + Retries with backoff
  + Increase shards (scaling)
  + Ensure your partition key is a good one

AWS Kinesis API – Consumers

* Can use a normal consumer ( CLI, SDK, etc…)
* Can use Kinesis Client Library ( in Java, Node, Python, Ruby, .Net)
  + KCL uses DynamoDB to checkpoint offsets
  + KCL uses DynamoDB to track other workers and share the work amongst shards

AWS Lambda

* Templates have to be uploaded in S3 and then referenced in CloudFormation

Serverless:

* A new paradigm in which the developers don’t have to manage servers anymore…
* They just deploy code
* They just deploy functions
* Initially… Serverless == FaaS( functions as a service)
* Serverless was pioneered by AWS Lambda but now also includes anything that’s managed: “databases, messaging, storage, etc.”
* **Serverless does not mean there are no servers…**  it means you just don’t manage / provision / see them

**Serverless services in AWS:**

* AWS Lambda & Step Functions
* DynamoDB
* AWS Cognito
* AWS API Gateway
* Amazon S3
* AWS SNS & SQS
* AWS Kinesis
* Aurora Serverless

**Why AWS Lambda:**

* **EC2**
  + Virtual Servers in the Cloud
  + Limited by RAM and CPU
  + Continuously Running
  + Scaling means intervention to add/remove servers
* **Lambda**
  + **Virtual Functions -** No servers to manage!
  + Limited by time - **Short executions**
  + Run **on-demand**
  + **Scaling is automated**
* **Benefits:**
  + **Easy Pricing:**
    - Pay per request and compute time
    - Free Tier of 1,000,000 AWS Lambda requests and 400,000 GBs of computing time
    - Integrate with the whole AWS stack
    - Integrate with many programming languages
    - Easy monitoring through AWS CloudWatch
    - Easy to get more resources per functions ( up to 3GBs of RAM! each)
    - Increasing RAM will also improve the CPU and Network as well.
  + **AWS Lambda Language Support:**
    - Node.js (javascript)
    - Python
    - Java (Java 8 Compatible)
    - C#
    - Golang
    - C# / Powershell
  + **AWS Lambda Integrations (Main):**
    - API Gateway
    - Kinesis
    - DynamoDB
    - AWS S3 - Simple Storage Service
    - AWS IoT
    - CloudWatch Events
    - CloudWatch Logs
    - AWS SNS
    - AWS Cognito
    - Amazon SQS
  + **AWS Lambda Pricing**
    - [**https://aws.amazon.com/lambda/pricing/**](https://aws.amazon.com/lambda/pricing/)
    - Pay per calls:
      * First 1 million requests are free
      * .20$ per 1 million requests thereafter
    - Pay per duration: (in increments of 100ms):
      * 400,000 GB-seconds of compute time per month if FREE
      * == 400,000 seconds if functions is 1Gb of RAM
      * == 3,200,000 seconds if function is 128 MB RAM
      * After that 1$ for 600,000GB-seconds
    - It is usually very cheap to run AWS Lambda’s so its VERY popular
  + **Creating a Lambda function:**
    - **Pick a region**
    - **AWS Lambda**
    - **Create Function (Three options)**
      * **Author from scratch**
      * **Use a blueprint**
      * **Browse serverless app repository**
    - **Basic Information**
      * **Function Name**
      * **Execution Role (IAM Role) - Existing or New etc.**
      * **Place Base Code**
      * **Create Function**
  + **Testing Functions**
    - Click on test from the specific lambda’s dashboard
    - Create an existing test or new one
    - Then name the event
    - Then add the JSON event that is being passed into the lambda
  + By giving an IAM role to the lambda function you can enable CloudWatch Logs
* AWS Lambda Configuration
  + Timeout: default 3 seconds, with a new limit of up to 15 min (900s)
  + Environment Variables
  + Allocated Memory (128M to 3G)
  + Ability to deploy within a VPC - assign security groups
  + IAM execution role must be attached to the Lambda function
* AWS Lambda Concurrency and Throttling
  + Concurrency: up to 1000 executions ( can be increased via a ticket to AWS)
  + Can set a “reserved concurrency” at the function
  + Each invocation over the concurrency limit will trigger a “Throttle”:
  + Throttle behavior:
    - If synchronous invocation => return Throttle Error -429
    - If asynchronous invocation => retry automatically and then go to DLQ
* AWS Lambda Retries and DLQ (Dead Letter Queue)
  + If a lambda function asynchronous invocation fails, it will be retried twice
  + After all the retries, unprocessed events go to the Dead Letter Queue
  + DLQ can be SNS topic or SQS queue
  + The original event payload is sent to the DLQ
  + This is an easy way to debug what’s wrong with your functions in production without changing the code.
  + Make sure the IAM execution role is correct for your lambda function
  + Debugging and error handling has been replaced with **AWS X-Ray**. The configuration of DLQ is now located in the configuration section under the **Asynchronous Invocation**
* AWS Lambda Logging, Monitoring and Tracing
  + CloudWatch
    - AWS Lambda execution logs are stored in AWS CloudWatch Logs
    - AWS Lambda metrics are displayed in AWS CloudWatch metrics
    - **Make sure you AWS Lambda functions as an execution role with an IAM policy that authorizes writes to CloudWatch**
  + X-Ray
    - It’s possible to trace Lambda with X-Ray
    - Enable in Lambda configuration (runs the X-Ray daemon for you)
    - Use AWS SDK in code
    - **Ensure Lambda Function ahs correct IAM Execution Role.**
* **AWS Lambda Limits to Know**
  + **Execution**
    - Memory allocation: 128 - 3008 Mb (64 mb increments)
    - Maximum execution time is 15 min
    - Disk capacity in the “function container” (in/tmp): 512 mb
    - Concurrency Limits: 1000 (Can be increased through a ticket)
  + **Deploy**
    - Lambda function deployment size (compressed .zip): 50Mb
    - Size of uncompressed deployment ( code + dependencies): 250Mb
    - Can use the /temp directory to load other files at startup
    - Size of environment variables: 4 KB

**AWS Lambda Versions**

* When you work on a Lambda function, we work on $LATEST
* When we’re ready to publish a lambda function, we create a version
* Versions are immutable
* Versions have increasing version numbers
* Versions get their own ARN (Amazon Resource Name)
* Versions = code + configuration (nothing can be changed since they are immutable)
* Each version of the lambda function can be accessed.

**AWS Lambda Aliases:**

* Aliases are “pointers” to Lambda function versions
* We can define a “dev”,” test”, “prod” aliases and have them point at different lambda versions
* Aliases are mutable
* Aliases enable Blue / Green deployment by assigning weights to lambda functions
  + So you can say what percentage of users are going to different versions and test new lambdas to test possible configurations.

**Setting up Lambda Versions and Aliases**

1. Inside of the lambda dashboard, you can click on the Qualifiers tab to begin
2. Qualifiers
   1. Versions
      1. Create a new version by clicking on the actions tab in the lambda dashboard and selecting create a new version
         1. This will take a snapshot of the lambda and save the code and configurations within AWS and assign it an ARN
   2. Aliases
      1. Create a new alias by clicking on the actions tab in the lambda dashboard and selecting create a new Alias
      2. Create a name and description for the new alias that you will expose to users
      3. Then select the version that the specific alias is pointing towards or will use
      4. Then if you want select the weight of traffic between the different lambda versions in the same alias.

**Lambda Function Dependencies**

* If your Lambda function depends on external libraries:
* **You need to install the packages alongside your code and zip together.**
  + For Node.js, use NPM and include the node\_modules directory
  + For Python, use pip --target options
  + For Java, include the relevant .jar files
* Steps.sh file will run the commands: (This is very customizable and should not be used verbatim understand the importance)
  + //Install all dependencies from package.json
  + Npm install
  + //Set proper permissions for project files
  + Chmod a+r \*
  + // you need to have the zip command available
  + Zip -r function.zip .
* Upload the zip straight to Lambda if less than 50Mb, else you have to upload it to S3 and make a reference to it
* Native libraries work: they need to be compiled on the Amazon Linux but they can be imported normally.

Lambda and CloudFormation

* You must store the Lambda zip in s3
* You must refer to the s3 zip location in the CloudFormation code

This is an introduction to CloudFormation

* It can take over to 3 hours to properly learn and master CloudFormation

DynamoDB

Traditional Architecture

* Traditional applications leverage RDBMS databases
* These databases have the SQL query language
* Strong requirements about how the data should be modeled
* Ability to do join, aggregations, computations
* Vertical Scaling ( means usually getting a more powerful CPU/RAM/IO)

NoSQL databases

* NOSQL databases are non-relational databases and are distributed
* NoSQL databases include MongoDB, DynamoDB, etc
* NoSQL databases do not support join
* All the data that is needed for a query is present in one row
* NoSQL databases don’t perform aggregations such as SUM
* NoSQL databases scale horizontally
* There’s no “right or wrong” for NoSQL vs SQL, they just require to model the data differently and think about user queries differently

DynamoDB

* Fully Managed, Highly available with replication across 3 AZ
* NoSQL database - not a relational database
* Scales to massive workloads, distributed database
* Millions of requests per seconds, trillions of row, 100s of TB of storage
* Fast and consistent in performance (low latency on retrieval)
* Integrated with IAM for security, authorization and administration
* Enables event driven programming with DynamoDb Streams
* Low cost and auto scaling capabilities

DynamoDB Basics

* DynamoDB is made of tables
* Each table has a primary key ( must be decided at creation time)
* Each table can have infinite number of items ( = rows)
* Each item has attributes ( can be added over time can be null)
* Maximum size of a item is 400KB
* Data types supported are:
  + Scalar Types: String, Number, Binary, Boolean, Null
  + Document Types: List, Map
  + Set Types: String Set, Number Set, Binary Set

DynamoBD – Primary Keys

* Option 1: Partition key only (HASH)
  + Partition key must be unique for each item
  + Partition key must be “divers” so that the data is distributed
  + Example: user\_id for a users table
* Option 2: Partition + Sort Key (Unique)
  + The combination must be unique
  + Data is grouped by partition key
  + Sort == range key
  + Example: users-games table
    - User\_id for partition key
    - Game\_id for the sort key

DynamoDB – Provisioned Throughput

* Table must have provisioned read and write capacity units
* Read Capacity Units (RCU): throughput for reads
* Write Capacity Units (WCU): throughput for writes
* Option to setup auto-scaling of throughput to meet demand
* Throughput can be exceeded temporarily using “burst credit”
* If burst credit are empty you’ll get a “ProvisionedThroughputException”
* It's then advises to do an exponential back-off retry

DynamoDB – Write Capacity Units

* One write capacity unit represents one write per second for an item up to 1KB in size.
* If the items are larger than 1KB, more WCU are consumed
* Example 1: we write 10 objects pers seconds of 2KB each.
  + We need 10 \* 2 = 20WCU
* Example 2: we write 6 objects per second of 4.5KB each
  + We need 6 \* 5 = 30WCU (4.5 gets rounded to the upper KB)
* Example 3: we write 120 objects per minute of 2KB each
  + We need 120/60\*2 = 4WCU

Strongly Consistent Read vs Eventually Consistent Read:

* Eventually Consistent Read: If we read just after a write, it’s possible we’ll get unexpected response because of replication
* Strongly Consistent Read: If we read just after a write, we will get the correct data
* By Default: DynamoDB uses Eventually Consistent Reads, but GetItem, Query & Scan provide a “Consistent Read” parameter you can set to True

DynamoDB – Read Capacity Units

Consistent read divide reads by 2

Strong consistent reads multiply by 4

KB per seconds % 4 round up

* One read capacity unit represents one strongly consistent read per second, or two eventually consistent reads per seconds, for an item up to 4KB in size.
* If the item are larger than 4KB, more RCU are consumed
* Example 1: 10 strongly consistent reads per seconds of 4KB each
  + We need 10 \* 4 / 4 KB = 10 RCU
* Example 2: 16 eventually consistent reads per seconds of 12 KB each
  + We need (16/2) \* (12/4) = 24 RCU
* Example 3: 10 strongly consistent reads per seconds of 6 KB each
  + We need 10 \* 8 KB / 4 = 20 RCU (we have to round up 6KB to 8KB)

DynamoDB – Partitions Internal

* Data is divided in partitions
* Partition keys go through a hashing algorithm to know to which partition they go to
* To compute the number of partitions:
  + By capacity (TOTAL RCU / 3000) + (TOTAL WCU / 1000)
  + By size: Total Size / 10 GB
  + Total partitions = CEILING(MAX(Capacity, Size))
* WCU and RCU are spread evenly between partitions

DynamoDB -Throttling

* If we exceed our RCU or WCU, we get ProvisionedThroughputExceededExceptions
* Reasons:
  + Hot keys: one partitionkey is being read too many times
  + Hot partitions:
  + Very large items: remember RCU and WCU depends on size of items
* Solutions:
  + Exponential back-off when exception is encountered (already in SDK)
  + Distribute partition keys as much as possible
  + If RCU issue we can use DynamoDB Accelerator (DAX)

DynamoDb \_ Writing Data

* PutItem - write data to dynamoDB (create data or full replace
  + Consumes WCU
* UpdateItem - Update data in DynamoDB (partial update of attributes
  + Possibility to use atomic counters and increase them
* Conditonal Writes:
  + Accepts a write/ update only if conditions are respected, otherwise reject
  + Helps with concurrent access to items
  + No performance impact
* Delete Item
  + Delete an individual row or item
  + Ability to peform a conditional delete
* DeleteTable
  + Delete a whole table and all its items
  + Much quicker deletion than delete item
* BatchWriteItem
  + Up to 25 putitem and / or deleteitem in one call
  + Up to 16 MB of data written
  + Up to 400KB of data per item
* Batching allows you to save in latency by reducing the number of API calls done against DynamoDB
* Operations are done in parallel for better efficiency
* It’s possible for part of a batch to fail, in which case we have the try the failed items ( using exponential back off strategy)
* GetItem:
  + Read based on primary key
  + Primary key = hash or hash range
  + Eventually consistent read by default
  + Option to use strongly consistent reads ( more RCU - might take longer)
  + Projection expression can be specified to include only certain attributes
* BatchGetItem:
  + Up to 100 items
  + Up to 16 MB of data
  + Items are retrieved in parallel to minimize latency\
* Query return items based on:
  + partition key value ( must be = operator)
  + SortKey value ( operators ) -- optional
  + FilterExpression to further filter (client side filtering)
* Returns:
  + Up to 1mb of data
  + Or number of items specified in limit
* Able to do pagination on the results
* Can query table a local secondary index or a global secondary index, or a global secondary index
* Scan the entire table and then filter out data (inefficient)
* Returns up to 1 MB of data - use pagination to keep on reading
* Consumes a lot of RCU
* Limit impact using limit or reduce the size of the result and pause
* For faster performance use parallel scans:
  + Multiple instances scan multiple partitioners at the same time
  + Increases the throughput and RCU consumed
  + Limit the impact of parallel scans just like you would for scans
* Can use a ProjectionExpression + FilterExpression ( no change to RCU )

DynamoDB -LSI (Local Secondary Index)

* Alternate range key for your table, local to the hash key
* Up to five local secondary indexs per table
* The sort key consists of exactly one scalar attribute.
* The attribute that you choose must be a scalar String, Number, or Binary
* LSI must be defined at table creation time

DynamoDB – GSI (Global Secondary Index)

* To speed up queries on non-key attributes, use a Global Secondary Index
* GSI = Partitionkey + Optional sort key
* The index is a new “table” and we can project attributes on it
  + The partition key and sort key of the original table are always porjected (KEYS\_ONLY)
  + Can specify extra attributes to project (INCLUDE)
  + Can use all attributes from main table (ALL)
* Must define RCU / WCU for the index
* Possiblity to add / modify GSI ( not LSI )

DynamoDB Indexes and Throttling

* GSI:
  + If the writes are throttled on the GSI, then the main table will be throttled!
  + Even if the WCU on the main tables are fine
  + Choose your GSI partition key carefully!
  + Assign your WCU capacity carefully
* LSI:
  + Uses the WCU and RCu of the main table
  + No special throttling considerations

DynamoDB Concurrency

* DynamoDB has a feature called “conditional update / delete
* That means that you can ensure an item hasn’t changed before altering it
* That makes DynamoDB an optimistic locking/ concurrency database

DynamoDB –DAX

* DAX = DynamoDB Accelerator
* Seamless cache for DynamoDB, no application re-write
* Writes go through DAX to DynamoDB
* Micro second latency for cached reads and queries
* Solves the Hot Key problem ( too many reads)
* 5 min TTL for cache by default
* Up to 10 nodes in a cluster
* Multi AZ ( 3 nodes min recommended for production)
* Secure (encryption at rest with KMS, VPC, IAM, CloudTrail … )

DynamoDB Streams

* Changes in DynamoDB Streams ( create, update delete) can end up in a DynamoDB stream
* This stream can be read by AWS Lambda, and we can then do:
  + React to changes in real time ( welcome email to new users)
  + Analytics
  + Create derivative tables / views
  + Insert into Elastic Search
* Could implement cross region replication using Streams
* Stream has 24 hours of data retention

DyanmoDB – TTL ( Time to Live )

* TTL = automatically delete an item after an expiry date / time
* TTL is provided at no extra cost, deletions do not use WCU / RCU
* TTL is a background task operated by the DynamoDB service itself
* Helps reduce storage and manage the table size over time
* Helps adhere to regulatory norms
* TTL is enable per row ( you define a TTL column , and add a date there
* DynamoDB Typically deletes expired items within 48 hours of expiration
* Deleted items due to TTl are also deleted in GSI / LSI
* DynamoDB streams i can help recover expire items

API Gateway:

Client → Amazon API Gateway → AWS Lambda → DynamoDB

* AWS Lambda \_ API Gateway: No infrastructure to manage
* Handle API versioning (v1, v2)
* Handle different environments
* Handle Security
* Create API keys, handle request throttling
* Swagger /Open API import to quickly define APIs
* Transforms and validate requests and responses
* Generate SDK and API specifications Cache API responses

API Gateway Integrations:

* Outside of the VPC:
  + AWS Lambda
  + Endpoints on EC2
  + Load Balancers
  + Any AWS service
  + External and Publically accessible HTTP endpoints
* Inside of VPC
  + AWS Lambda in your VPC
  + EC2 endpoints in you VPC

API gateway Deployment Stages

* Making changes in the API Gateway does not mean they’re effective
* You need to make a “deployment” for them to be ineffect
* It’s a common source of confusion
* Changes are deployed to “Stage” ( as many as you want)
* Use the naming you like for stages ( dev, test, prod)
* Each stage has its own configuration parameters
* Stages can be rolled a back as a history of deployments is kept

API Gateway – Stage Variables

* Stage variables are like environment variables for API Gateway
* Use them to change often changing configuration values
* They can be used in:
  + Lambda function ARN
  + HTTP Endpoint
  + Parameter mapping templates
* Use Cases:
  + Configure HTTP endpoints your stages talk to ( dev, test, prod..)
  + Pass configuration parameters to AWS Lambda through mapping templates
* Stage variables are passed to the “context” object in AWS Lambda

API Gateway Stage Varibales & Lambda Aliases

* We create a stage variable to indicate the corresponding Lambda alias
* Our API gateway will automatically invoke the right Lambda function

API Gateway – Canary Deployment

* Possibility to enable canary deployment for any stage (usually prod)
* Choose the % of traffic the canary channel receives
* Metrics & Logs are sparate ( for better monitoring)
* Possibility to override stage variables for canary
* This is blue / green deployment with AWS Lambda & API Gateway

Mapping Templates

* Mapping templates can be used to modify / response
* Rename parameters
* Modify body content
* Add headers
* Map JSON to XML for sending to backend or back to client
* Uses velocity Template Language (VTL) : for loop, if etc…
* Filter output results ( remove unnecessary data )

AWS API Gateway Swagger / Open API spec

* Common way of defining REST APIs, using API, using API definition as code
* IMport existing Swagger / OpenAPI 3.0 spec to API Gateway
  + Method
  + Method Request
  + Integration Request
  + Method Response
  + + AWS extensions for API gateway and setup every single option
* Can export current API as Swagger / OpenAPI spec
* Swagger can be written in YAML or JSON
* Using Swagger we can generate SDK for our applications

Caching API responses

* Caching reduces the number of calls made to the backend
* Default time TTL (time to live) is 300 seconds (min: 0s, max 3600s)
* Caches are defined per stage
* Cache are defined per stage
* Cache capacity between 0.5GB to 237GB
* Possible to override cache settings for specific methods
* Able to flush the entire cache (invalidate it) immediately
* Clients can invalidate the cache with header: **Cache-Control: max-age=0** (with proper IAM authorization)

API Gateway – Logging, Monitoring, Tracing

* CloudWatch Logs:
  + Enable CloudWatch logging at the Stage level (with Log Level )
  + Can override settings on a per API basis (ex: ERROR, DEBUG, INFO)
  + Log contains information about request / response body
* CloudWatch Metrics:
  + Metrics are by stage
  + Possibility to enable detailed metrics
* X-Ray:
  + Enable tracing to get extra information about requests in API Gateway
  + X-Ray API Gateway + AWS Lambda give you the full picture

AWS API Gateway - CORS

* CORS must be enabled when you received
* The Options pre-flight requert must contain the following headers:
  + Access-Control-Allow-Methods
  + Access-Control-Allow-Headers
  + Access-Control-Allow-Origin
  + CORS can be enabled through the console

API Gateway – Usage plans & API Keys

* What if you want to limit your customers usage of your API?
* **Usage Plans:** 
  + Throttling: set overall capacity and burst capacity
  + Quotas: number of requests made per day / week / month
  + Associate with desired API Stages
* **API Keys:**
  + Generate one per customer
  + Associate with usage plans
* Ability to track usage for API Keys

API Gateway – Security

**IAM Permissions**

* Create an IAM policy authorization and attach to User/ROle
* API gateway verifies IAM permissions passed by the calling application
* Good to provide access within your own infrastructure
* Leverages “Sig v4” capability where IAM credential are in headers

API Gateway –Security

**Lambda Authorizer (formerly Custom Authorizers)**

* Uses AWS Lambda to validate the token in header being passed
* Option to cache results of authenticatin
* Helps to use OAUTH/ SAML / 3rd party type of authentication
* Lambda must return an IAM policy for the user

API Gateway – Security

* Cognito fully manages user lifecycle
* API gateway verifies identity automatically from AWS Cognito
* No custom implementation required
* Cognito only helps with authentication, not authorization

API Gateway – Security – Summary

* IAM
  + Great for users / roles already within your AWS account
  + Handle authentication + authorization
  + Leverages Sig v4
* Custom Authorizer:
  + Grerat for 3rd party tokens
  + Very flexible in terms of what IAM policy is returned
  + Handle Authentication and Authorization
  + Pay per Lambda Invocation
* Cognito User Pool:
  + You manage your own user pool ( can be backed by Facebook, Google, etc)
  + No need to write any custom code
  + Must implement authorization in the backend

AWS Serverless Application Model (SAM)

* SAM = serverless APplication Model
* Framework for developing and deploying serverless applications
* All teh configuration is YAML code
* Generate complex CloudFormation from simple SAM YAML file
* Supports anything from CloudFormation: Outputs, Mappings, Parameters, Resources
* Only two commands to deploy to AWS
* SAM can use CodeDeploy to deploy Lambda functions
* SAM can help you to run Lambda, API Gateway, DynamoDB locally

AWS SAM – Recipe

* Transform Header indicates it’s SAM template:
  + Transform: “AWS::Serverless-2016-10-31”
* Write Code
  + AWS::Serverless::Function
  + AWS::Serverless::Api
  + AWS::Serverless::SimpleTable
* Package & Deploy:
  + Aws cloudformation package / sam package
  + Aws cloudformation deploy / sam deploy
* New section at the exam - some tricky questions
* Docker Introduction
* ECS
  + Cluster
  + Services
  + Tasks
  + Tasks Definition
* ECR
* Fargate
* Exam Tips

What is Docker?

* Docker is a software development platform to deploy apps
* APps are packaged in containers that can be run on any OS
* Apps run the same, regardless of where they’re run
  + Any machine
  + No compatibility
  + Issues
  + Predictable behavior
  + Less work
  + Easier to maintain and deploy
  + Works with any language, any OS< any technology

Where Docker images are stored?

* Docker images are stored in Docker Repositories
* Public: Docker Hub
  + Find base images for many technologies or OS:
  + Ubuntu
  + MySQL
  + NodeJS, Java etc
* Private: Amazon ECR ( Elastic Container Registry)

Docer versus virtual Machines

* Docker is “sort of “ a virtualization technology, but not exactly
* Resources are shared with the host => many containers on on server
* Architecture top down VM
  + Apps
  + Guest OS (VM)
  + Hypervisor
  + Host OS
  + Infrastructure
* Docker Architecture
  + Container x
  + Docker Daemon
  + Host OS (EC 2 Instance)
  + Infrastructure

Getting Started with Docker

* Download Docker
* Docker file
* Build them
* Creates docker image
* Run docker image to make container
* Push them to store them in docker hub or amazon ecr
* Pull images from docker hub or amazon ecr

AWS CloudFront

* Content Delivery Network
* Improve read performance, content is cached at the edge
* 216 Points of presence globally ( edge locations)
* DDos Protection, integration with Shield AWS Web Application Firewall
* Can expose external HTTPS and can talk to internal HTTPS backends

CloudFront – Origins

* S3 bucket
  + For distributing files and caching them at the edge
  + Enhanced security with CloudFront Origin ASccess Idenitity (OSAI)
  + CloudFront can be used as an ingress ( to upload files to s3))
* Custom Origin (HTTP)
  + Application Load Balancer
  + Ec2 instance
  + S3 website ( must first be enabled the bucket as as tactic s3 website)
  + Any HTTP backend you want

CloudFront at a high level

CloudFront – S3 as an Origin

* The origin of the delivered content is an S3 bucket with an Origin Access Identity and a S3 bucket policy. The edge locations then get cache information from the S3 bucket

CloudFront  – ALB or EC2 as an origin

* The Ec2 instances must be public and allow public IP of edge locations
* Allow Security Group of Load Balancers which allows for a public application load balancers that delivers information to EC2 instances that are private

CloudFront Geo Restriction

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

CloudFrong vs S3 Cross Region Replication

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

CloudFront Caching

* Cache based on
  + Headers
  + Session Cookies
  + Query String Parameters
* The cache lives at each CloudFront Edge Location
* You want to maximize the cache hit rate to minimaize requests on the origin
* Control the TTL can be set by the origin using the cache control h4eader and expire header

CloudFront Security

* CloudFront and HTTPS
  + Viewer Protocol Policy
    - Redirect HTTPS to HTTPS
    - Or use HTTPS only
  + Origin Protocol Policy (HTTP or S#):
    - HTTPS only
    - Or MatchViewer HTTP to HTTP and ditto for HTTPS
  + S3 buckets do not support HTTPS

CloudFront Signed URL /Signed Cookies

* You want to distribute paid shgared content to premium users over the world
* We can use CloudFront Signed URL / Cookie. We attached a policy with:
  + Includes URL expiration
  + Icludes IP ranges to access the data from Trusted signers
* How long should the uRL be valid for?
  + Shared content ( movie, music): make it short ( a few minutes)
  + Private content ( private to the user): you can make it last for years
* Signed URL = access to individual files ( one signed URL per file)
* Signed Cookies = access to multiple files ( one signed cookie for many files)

cloudFrong Signed URL vs S3 PRe-Signed URL

* CloudFroont Signed URL:
  + Allow access to a path , no matter the origin
  + Account wide key-pari only the root can manage it
  + Can filter by IP, path, date, expiration
  + Can leverage caching features
* S3 Pre-Signed URL:
  + Issue a request as the person who pre-signed the URL
  + Uses the IAM key of the signing IAM principal
  + Limited Lifetime

Docker Containers Management

* To manage containers, we need a container management platform
* Three choices
  + ECS: Amazon’s own platform
  + Fargate: Amazon’s own Serverless platform
  + EKS: Amazon’s managed Kubernetes (open source docker management system)

ECS Clusters Overview

* ECS Clusters are logical grouping of EC2 instances
* EC2 instances run the ECS agent (Docker container)
* The ECS agents registers the instance to the ECS cluster
* The EC2 instances run a special AMI, made specifically for ECS

ECS Task Definitions

* Tasks definitions are metadata in JSON form to tell ECS how to run a Docker Container
* It contains crucial information around:
  + IMage Name
  + Port Binding for Container and HOst
  + Memory and CPU required
  + Environment variables
  + Networking Information

ECS Service

* ECS Service help define how many tasks should run and how they should be run
* They ensure that the number oftasks desired is running across our fleet of EC2 instances
* They can be linked to ELB / NLB / ALB if needed

ECR

* ECR is a private Docker image repository
* Access is controlled through IAM
* You need to run some commands to push pull:
  + $(aws ecr get-login --no-include-email --region eu-west-I)
  + Docker push 1234567890.dkr.ecr.eu-west-I.amazonaws.com/demo:latest
  + Docker pull 1234567890.dkr.ecr.eu-west-I.amazonaws.com/demo:latest

Fargate

* When launching an ECS Cluster, we have to create our EC2 instances
* If we needed to scale, we need to add EC2 instances
* So we mangae infrastructrues
* With Fargate, it’s all serverless!
* We don’t provision EC2 instances
* We just create task definitions, and SAWS will run our containers for us
* To scale, just increase the task number. Simple! No more EC2

ECS+ X-Ray integration options

ECS Cluster

Xray Container as a Daemon

Elastic Beanstalk + ECS

* You can run Elastic Beanstalk in Single & Multi Docker Container mode
* Multi Docker helps run multiple containers per EC2 instance in EB
* This will create for you:
  + EcS sCluster
  + Ec2 instances, configured to use the ECS Cluster
  + Load Balancer( in high availability mode)
  + Task Definition And execution
  + Requires a config file Dockerrun.aws.json at the root of source code
  + Your Docker images must be pre-built and stored in ECR for example

ECS Summary + Exam Tips

* ECS is used to run Docker containers and has 3 flavors:
* ECS “Classic”: provisioned EC2 instances to run containers onto
* Fargate: ECS Serverless, no more EC2 to provision
* EKS: Managed Kubernetes by AWS

ECS Classic

* EC2 instances must be created
* We must configure the file /etc/ecs/ecs.config with the cluster name
* The Ec2 instane must run an ECS agent
* EC2 instances can ru multiple containers on the same type:
  + You must not specify a host port (only container prot)
  + You should use an Application load balancer with the dynamic port ampping
  + The ec2 instance security group must allow traffic from the ALB on all ports
* ECS tasks can have IAM ROles to execute actions against AWS
* Secrurity Groups operate at the instance level not task level

ECR is used to store DOcker Images

ECR is tightly integrated with IAM

If you can’t pull or push a image check the IAM policy

AWS Fargate

* Fargate is Serverless ( no Ec2 to manage)
* AWS provisions containers for us and assigns them ENI
* Fargate containers are provisioned by the container spec (CPU / RAM)
* Fargate tasks can have IAM roles to execute actions Against AWS

ECS iNtegrations

* ECS does integrate with X-Reay
  + To make it work, X-ray must be running as a 2nd container within the task definition
  + ECS does integrate with CloudWatch Logs:
    - You need to setup logging at the task definition level
    - Each container will have a different log stream

AWS KMS (Key Management Service

* Anytime you hear encryption for an aws service its most likely KMS  
  Easy way to control access to your data, AWS manages kyes for us
* Fully integrated with IAM for authorization
* Seamlessly integreated int:
  + Amazon EBS: encrypt volumes
  + Amazon S3: Server side encryption of objects
  + Amazon RedShift: encryption of data
  + Amazon RDS: encryption of data
  + Amazon SSM: Parameter store

Encryption in flight (SSL)

* Data is encrypted before sending and decrypted after receiving
* SSL certificates help weith encryption (HTTPS)
* Encrtyption in flight ensrures no man in the middle attack (MITM) can happen

Server side encryption at rest

* Data i encrypted after being received by the server
* Data is decrypted before bein sent
* It is stored in an encrypted forme thanks to a key ( usua.lly a data key)
* TThe encryption / decryption keys must be managed somewhere and the server must have access to it

Client Side encryption

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

AWS KMS 101

* Anytime you need to share sensitive informatin use KMS
  + Database passwords credentials to external service
  + Private key of SSL certificates
  + Private Key of SSL certificates
* The value in KMS i sthat the CMK used to encrypt data can never be retireved by the user , and the CMK can be rotated for etra security
* Never ever store your secrets in plaintext especially in your code!
* Encrypted secrets can be stored in the code / enviornment variables
* KMS can only help in encryption up to 4KB of data per call
* If Data > kB, use enveloipe encryption
* To give access to KMS to someone:
  + Make sure teh key policy allows the user
  + Make sure the IAM policy allows teh API calls

AWS KMS (key management service

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

Envelope Encryption

* KMS Encrypt API call has a limit of 4KB
* If you want to encrypt > 4KB, we need to use Envelope eEncryption
* The main API that will help us is the GenerateDataKey API
* FOR THE EXAM: anything over 4KB of data that needs to be encrypted must use the envelope encryption == GenerateDataKey API

Deep dive into enveklope encryption generateDataKey API

* GenerateDataKey API
* KMS CM
* Check IAM
* Send plaintext data key
* Big file
* Placed in an envelope
* Inside the envelope
* Encrypt data key with CMK
* Is the encrypted bif gile with the the encrypted data key

Decrypt envelope data

* Envelope Decrypt API
* CMK
* Check IAM permissions
* Decrypt data key using CMK

Both processes occur on the Client Side

Encryption SDK

* The AWS Encryption SDK implemented Envelope Encryption for us
* THe encryption SDK also exists as a CLI tool we can install
* Implementations for Java, Python, C, JavaScript
* Feature - Dta Key caching:
  + Re-use data keys instead of creating new ones for each encryption
  + Helps with reducing the number of call s to KMS with a security trade-off
  + Use LocalCryptoMaterialsCache (max age, maxbytes, max numbers of messages)

KMS Symmetric –API summary

* Encrypt: encrypt p to 4 KB of data through KMS
* GenerateDataKey: generates a unique symmetric data key (DEK)
  + Returns a plaintext copy of the data key
  + AND a copy that is encrypted under the CMK that you specify
* GenerateDataKeyWithoutPlaintext
  + Generate a DEK to use at some pint ( not immediately)
  + DEK that is encrypted under the CMK that you specify ( must use decrypt later)
* Decrypt: decrypt up to 4KB of data ( including Data Encryption Keys)
* GenerateRandom: Returns a random bite string

KMS Request Quotas

* When you exceed a request quota you get a ThrottlingException
* To respond, use exponential backoff ( backoff and retry)
* For cryptographic operations they share quota
* This includes requests made by AWS on your behalf
* For GenerateDataKey, consider using DEK caching from the Encryption SDK
* You can request a Request Quotas increase through API or AWS support

KMS Request Quotas

Symmetric CMK quota

* 5,500
* 10,000
* 30,000

Asymmetric

* 500
* 300

S3 Encryption of Objects

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

SSE-KMS

* SSE-KMS: encryption using keys handled & managed by KMS
* KMS Advantages: user control + audit trail
* Object is encrypted server side
* Must set header: “x-amz-server-side-encryption”: “aws:kms”
* Process
* Object will be sent over HTTP/S + Header
* The object is then encrypted with a Customer Master Key from KMS
* Then is stored within the S3 bucket

SSE-KMS Deep Dive

* SSE-KMS leverages the GenerateDataKey & Decrypt KMS API calls under the hood
* These KMS API calls will show up in CloudTrail, helpful for logging
* Toe perform SSE-KMS, you need:
  + A KMS key policy that authorizes the user/ role
  + An IAM policy that authorizes access to KMS
  + Otherwise you will get an access denied error
* S3 calls to KMS for SSE#-KMS count against your KMS limits
  + If throttling, try exponential backoff
  + If throttling, you can request an increase in KMS limits
  + Ther service that is throttling is KMS, not Amazon S3

Exponential backoff

* Exponential backoff is an algorithm that uses feedback to multiplicatively decrease the rate of some process in order to gradually find an acceptable rate.

S3 Bucket Policies – Force SSL

* To force SSL, create an S3 bucket policy with a DENY on the condition
* aws: SecureTransport = false
* Note: Using an allow on aws:SecureTransport = false
* Note: Using an allow on aws:SecureTransport = true would allow anonymous GTetObject if using SSL

S3 bucket Policy – Force Encryption of SSE-KMS

* 1. Deny incorrect encryption header: make sure it includes aws:kms (=== SSE-KMS)
* 2. Deny no encryption header to ensure objects are not uploaded un-encrypted
* Note: could swap 2) for S3 default encryption of SSE-KMS

AWS Encryption SDK

* What if you want to encrypt over 4 KB using KMS
* For this, we need to use Envelope Encryption.
* Envelope Encryption is a bit cumbersome to implement
* The AWS Encryption SDK helps us use Envelope Encryption
* Note: it is different from the S3 Encryption SDK
* The Encryption SDK also exists as a CLI tool we can install
* For the exam: anything over 4KB of data that needs to be encrypted must use the Encryption SDK == Envelope Encryption == GenerateDataKey API

AWS Parameter Store

* Secure storage for configuration and secrets
* Optional Seamless Encryption using KMS
* Serverless, scalable, durable, easy SDK, free
* Version tracking of configurations / secrets
* Configuration management using path & UAN
* Notification with CloudWatch Events
* Integration with CloudFormation

AWS Parameter Store Hierarchy

* /my-department/
  + my-app/
    - dev/
      * Dburl
      * Db password
    - prod/
      * Db-url
      * Db-password
    - other-app/
  + /other-department/

Secrets Manager

* Newer service meant for storing secrets
* Capability to force rotation of secrets every x days
* Automate generation of secrets on rotation (uses Lambda)
* Integration with Amazon RDS
* Secrets are encrypted using KMS
* Mostly meant for RDS encryption

SSM Parameter Store vs Secrets Manager

Secrets Manager ($$$)

* Automatic rotation of secrets with AWS Lambda
* Integration with RDS, Redshift, DocumentDB
* KMS encrtyption is mandatory
* Can integration with CloudFormation

SSM Parameter Store ( $)

* Simple API
* No secret rotation
* KMS encryption is optional
* Can integrate with CloudFormation
* Can pull a secrets manager secret using the SSM Parameter store API

CloudWatch Logs - Encrtyption

* You can encrypt CloudWatch logs with KMS keys
* Encryption is enabled at the log group level, by associating a CMK with a log group level, bny associating a CMK with a log gorup either when you create the log group or after it exists.
* You cannot associate a CMK with aa log group using the CloudWatch Console
* You must use the CloudWatch Logs API:
  + Associate-kms-key: if the log group already exists
  + Create-log-group: if the log group doesn’t exist yet

CodeBuild Security

* TO acces resources in your VPC, make sure you specify a VPSC configuration for your COdeBuild
* Secrets in CodeBuild:
* Don’t store them as plaintext in environment variables
* Instead
  + Environment variables can reference the parameter store parameters
  + Environment variables can reference secret manager secrets

AWS STS – Security Token Service

* Allows to grant limited and temporary access to AWS resources ( up to 1 hours).
* AssumeRole: Assumes roles within your account or cross account
* AssumeRoleWithSAML: return credentials for users logged with SAML
* AssumeRoleWithWebIdentity
  + Return creds for users logged with an IdP ( Facebook, google, OIDC comp)
  + AWS recommends against using this, and using Cognito Identity pools instead
* GetSessionToken: for MFA, forom a user orr AWS account root user
* GetGederationToken: obtain temporary creds for a federated user
* GetCallerIdentity: return details about the IAM user or role used in the API call
* DecodeAuthorizationMessage: decode error messages when an AWS API is denied

Using STS to Assume a Role

* Define an IAM Role within your account or cross account
* Define which principles can access this IAM Role
* USe AWES STS (security token service) to retrieve credentials and impersonate the IAM Role you have access to (AssumeRoleAPI)
* Temporary credentials can be valid between 15 minutes to 1 hour
* Process
  + User uses AssumeRole API to AWS STS IAM will check for permissions
  + Then a temporary security credential is give to the user to allow to use another role for a limited amount of time

STS with MFA

* Use GetSessionToken from STS
* Appropriate IAM policy using IAM conditions
* aws:MultiFactoAuthPresent:true
* Reminder, GetSessionToken returns the following:
  + Access ID
  + Secret Key
  + Session Token
  + Expiration date

Advanced IAM - Authorization Model Evaluation of Policies, simplified

1. If there’s an explicit DENY, end decision and DENY
2. If there’s an ALLOW, end decision with ALLOW
3. Else DENY

IAM Policies & S3 Bucket Policies

* IAM Policies are attached to users, roles, groups
* S3 Bucket Policies are attached to buckets
* When evaluating if an IAM Principal can perform an operation X on a bucket, the union of its assigned IAM policies and S3 Bucket Policies will be evaluated

Example 1

* IAM Role attached to EC2 instance, authorizes Read and Write to “my bucket”
* No S3 Bucket Policy attached
* Then the EC2 instance can read and write to “ my bucket”

Example 2

* IAM Role attached to EC2 instance, authorizes RW to “my bucket”
* S3 Bucket Policy attached, explicitly deny to the IAM role
* Then the EC2 instance cannot read and write to “my bucket”

Example 3

* IAM Role attached to EC2 instance, no S3 bucket permissions
* S3 Bucket Policy attached, explicit RW allow to the IAM Role
* EC2 instance can read and write to “my bucket” because of the union policy between the IAM role and bucket policy

Example 4

* IAM Role attached to EC2 instance, explicit deny s3 bucket permissions
* S3 Bucket Policy attached, explicit RW allow to the IAM Role
* EC2 instances cannot read and write to “my bucket” the reason why is that explicit Deny is evaluated before explicit allow.

Dynamic Plicies with IAM

* How do you assign each user a /home/<user> folder in an S3 bucket
* Option 1:
  + Create an IAM policy allowing georges to have access to /home/georges
  + Create an IAM policy allowing sarah to have access to / home/sarah
  + Create an IAM policy allowing matt to have access to /home/matt
  + … one policy per user!
  + So this doesn’t scale
* Option2:
  + Create one dynamic policy with IAM
  + Leverage the special policy variable ${aws:username}

Dynamic policy example

{

“Sid”: “AllowAllS3ActionsInUserFolder”,

“Action” : [“s3:\*”],

“Effect” : “Allow”,

“Resource” : [“arn:aws:s3:::my-company/home/${aws:username}/\*”]

}

Inline vs Managed Policies

* AWS Managed Policy
  + Maintained by AWS
  + Good for power users and administrators
  + Updated in case of new services / new APIs
* Customer Managed Policy
  + Best Practice, re-usable, can be applied to many principals
  + vVersion Controlled + rollback, central change management
* Inline
  + Strict one to one relationship between policy and principal
  + Policy is deleted if you delete the IAM principal
  + Can only be 2KB large

Granting a User Permissions to Pass a Role to an AWS Service

* To configure many AWS service, you must pass an IAM role to the service ( this happens only once during the setup)
* The service will later assume the role and perform actions
* Example of passing a role:
  + Toan EC2 instance
  + To a Lambda function
  + To an ECS task
  + To codepipeline to allow it to invoke other services
* For this you need the IAM permission iam:PassRole
* It often comes with iam:GetRole to view the role being passed

IAM PassRold example

{

“Version” : “2012-10-17”,

“Statement” : [

{

“Effect”: “Allow”,

“Action”: [

“ec2:\*”

],

“Resource”: “\*”

},

{

“Effect”: “Allow”,

“Action”:”iam:PassRole”,

“Resource”: “ arn here”

}

]

}

Can a role be passed to any service?

* NO: Roles can only be passed to what their trust allows
* A trust policy for the role that allows the service to assume the role

What is Microsoft Active Directory (AD)?

* Found on any Windows Server with AD Domain Services
* Database of objects: User Accounts, COmputers, Printers, File Shares, Security groups
* Centralized security management, create account, assign permissions
* Objects are organized in trees
* A group of trees is a forest

AWS Directory Services

* AWS Managed Microsoft AD (Hybrid)
  + Create your own AD in AWS, manage users locally, supports MFA
  + Establish “trust” connection switch your on-premise AD
  + Users sit in both on premise and in aws
* AD Connector (on-premise)
  + Directory Gateway (proxy) to redirect to on-premise AD
  + Users are managed on the on-premise AD
  + All users are stored on the on-premise AD
* Simple AD (AWS)
  + AD-compatible managed directory on AWS
  + Cannot be joined with on-premise AD

The idea is that this will allow users to be filed in AWS for EC2 instances that uses window VMs

LAMDA ADDITIONAL

Lambda  – Synchronous Invocations

* Synchronous: CLI, SDK, API Gateway, Application Load Balancer
  + Results is returned right away
  + Error handling must happen client side ( retries, exponential backoff, etc)

Lambda – Synchronous Invocations - Services

* User Invoked:
  + Elastic Load balancing (Application Load Balancer
  + Amazon API Gateway
  + Amazon CloudFront ( Lambda@Edge)
  + Amazon S3 batch
* Service Invoked
  + Amazon Cognito
  + AWS Step Functions
* Other Services:
  + Amazon Lex
  + Amazon Alexa
  + Amazon Kinesis Data Firehose

Lambda Integration with ALB

* To expose a Lambda functions as an HTTP(S) endpoint
* You can use the application Load balancer ( or an API gateway)
* The lambda function must be registered in a target group

ALB to Lambda: HTTP to JSON

ALB Multi-Header Values

* ALB can support multi header values (ALB setting)
* When you enable multi-value headers, HTTP headers and query string parameters that are sent with multiple values are shown as arrays within the AWS Lambda event and response objects.
* HTTP
  + <http://example.com/path?name=foo&name=bar>
* JSON
  + “queryStringParameters”: {“name”: [“foo”,”bar”]}

Lamdba@Edge

* You have deployed a CDN using CloudFront
* What if you wanted to run a global AWS Lambda alongside?
* Or how to implement request filtering before reaching your applications
* For this, you can use Lambda@Edge:
  + Deploy lambda functions alongside your CloudFront CDN
  + Build more responsive applications
  + You don’t manage servers, Lambda are deployed globally
  + Customize the CDN content
  + Pay only for what you use

Lambda@Edge

* You can use Lambda to change CloudFront requests and responses
  + After CloudFront receives a request from a viewer (viewer request) 1
  + Before CloudFront forwards a request to the origin ( origin request) 2
  + After CloudFront receives the responses from the origin ( origin response) 3
  + Before CloudFront forwards the response to the viewer ( viewer response) 4
* You can also generate responses to viewers without ever sending the requests to the origin.

Example

* User visits an S3 bucket that hosts a website
* That website then makes a dynamic request to an API endpoint
* CloudFront cached results takes care of some things
* Lambda@Edge function ( Runs code in each CloudFront Edge, globally)
* The Lambda Queries data from a Amazon DynamoDB

Lambda@Edge: Use cases

* Website Security and privacy
* Dynamic Web Applications at the Edge
* Search Engine Optimizations (SEO)
* Intelligently Route Across Origins and Data Centers
* Bot mitigation at the edge
* Real-time Image transformation
* A/B Testing
* User Authorization and Authentication
* User Prioritization
* User Tracking and Analytics

Lambda – Asynchronous Invocations

* S3, SNS, CloudWatch Events…
* The events are placed in an Even Queue
* Lambda attempts to retry on errors
  + 3 tries total
  + 1 minute wait after 1st, then 2 minutes wait
* Make sure the processing is idempotent ( in case of retries)
* If the function is retired, you will see duplicate logs entries in CloudWatch Logs
* Can define a DLQ (dead letter queue) – SNS or SQS – for failed processing ( need correct IAM permissions)
* Asynchronous invocations allow you to speed up the processing if you don’t need to wait for the results (ex: you need 1000 files processed)

Lambda - Asynchronous Invocations - Services

* Amazon Simple Storage Service (S3)
* Amazon Simple Notification Service (SNS)
* Amazon CloudWatch Events / EventBridge
* AWS code commit ( CodeCommit Trigger: new branch , new tag, new push)
* AWS CodePipeline ( invoke a Lambda function during the pipeline, Lambda must callback)

------ other -------

* Amazon CloudWatch Logs (log processing)
* Amazon Simple Email Service
* AWS Cloudformation
* AWS Config
* AWS IoT
* AWS IoT Events

CloudWatch Events / EventBridge

API Gateway – Canary Deployment

* Possibility to enable canary deployments for any stage ( usually prod)
* Choose the % of traffic the canary channel receives
* Metrics & Logs are separate ( for better monitoring)
* Possibility to override stage variables for canary
* This i9s blue / green deployment with AWS Lambda & API Gateway

API Gateway - Integration Types

* Integration Type MOCK
  + API Gateway returns a response without sending the request to the backed
* Integrtion type HTTP / AWS ( Lambda & AWS Services)
  + You must configure both the integration request and integration response
  + Setup data mapping using mapping templates for the request and response
* Integration type AWS\_PROXY (Lambda Proxy):
  + Incoming request from the client is the input to Lambda
  + The function is responsible for the logic of request / response
  + No mapping template, headers, query string parameters… are passed as arguments
* Integration Type HTTP\_PROXY
  + No mapping template
  + The HTTP request is passed to the backedn
  + The HTTP response form the backedn is forwarded bny API Gateway

Mapping Templates (AWS & HTTP Integraiton)

Cannot be used with Proxy version integrations within API Gateway

* Mapping templates canb be used to modify request/ responses
* Rename/ Modify query string parameters
* Modify body content
* Add headers
* Uses Velocity Template Language (VTL): for loops and conditionals
* Filter output results ( remove unnecessary data)

Mapping Example: JSON to XML with SOAP

* SOAP API are XML based, whereas REST API are JSON based
* In this case, API Gateway should:
  + Extract data from the request: either path , payload, or header
  + Build SoAP messages based on request data ( mapping template)
  + Call SOAP service and receive XML response
  + Transform XML response to desired format ( like JSON) and respond to the user

AWS API Gateway Swagger / Open API spec

* Common way of definigng RES APIS, using API definition as code
* Import existing Swagger / OpenAPI 3.0 spec to API Gateway
  + Method Method request
  + Integration Request
  + Method Response
  + +AWS extensions for API gateway and setup every single option
* Can export current ASPI as Swagger / Open API spec
* Swagger can be written in YAML or JSON
* Using Swagger we can generate SDK for our applications

Caching API responses

* Caching reduces the number of calls made to the backed
* Degault TTL is 300 seconds min 0 s and max q hour
* Caches are defined per stage
* Possible to override cache settings per method
* Cache encryption option
* Cache capacity between 0.5 to 237 GB
* Cache is expensive, makes sense in production but not in dev

API Gateway cache Invalidation

* Able to flush the entire cache (invalidate it) immediately
* Clients can invalidate the cache with header: cache-control: max-age=0 (with proper IAM authorization)
* If you don’t impose an invalidateCache policy ( or choose the Require authorization check box in the console), any client can invalidate the API cache

API Gateway – Usage Plans & API keys

* If you want to make an API available as an offering ($) to your customers
* Usage Plan:
  + Who can access one or more deployed API stages and methods
  + How much and how fast they can access them
  + Uses API keys to identify API clients and meter access
  + Configure throttling limits and quota limits that are enforced on individual client
* API Keys:
  + Alphanumeric string values to distribute to your customers
  + Can use with usage plans to control access
  + Throttling limits are applied to the API keys
  + Quotas limits is the overall number of max requests

API – Correct Order for API keys

* To configure a usage Plan
  + Create one or more APIs, configure the methods to require an API key, and deploy the APis to stages
  + Generate or importa API keys to distribute to applications developers ( your customers ) who will be using your api
  + Create the usage plan with the desired throttle and quota limits
  + Associate API stages and API keys with the usage plan.
* Callers of the API must supply an assigned api key in the x-api-key header in requests to the API

API Gateway – Logging & Tracing

* CloudWatch Logs:
  + Enable CloudWatch logging at the Stage Level ( with Log LEvel)
  + Can override settings on a per APPI basis
  + Log contains information about request / response body
* X-Ray:
  + Enable tracing to get extra information about request s in API Gateway
  + X-Ray API Gateway + AWS Lambda gives you the full picture

API Gateway – CloudWatch Metrics

* Metrics are by stage, Possibility to enable detailed metrics
* CacheHitCount & CacheMissCount: efficiency of the cache
* Count: The total number API requests in a given period
* IntegrationLatency: The time between when API Gateway relays a request to the backend and when it receives a response from the backend.
* Latency: The tie between when API Gateway receives a request from a client and when it returns a response to the client. The latency includes the integration latency and either API Gateway overhead.
* Two type error 4XXError (client-side) & 5XXError (server-side)

API Gateway Throttling

* Account Limit
  + API Gateway throttles requests t 10000 rps across all API
  + Soft limit that can be increased upon request
* In case of throttling => 429 too many requests ( retriable error)
* Can set stage limit & Method limits to improve performance
* Or you can define Usage Plans to throttle per customer
* Just like Lambda Concurrency, one API that is overloaded, if not limited, can cause the other API to be throttled

API Gateway -Errors

* 4XX means Client errors
  + 400: bad Request
  + 403 Access Denied, WAF filtered
  + 429: quota exceeded, Throttle
* 5X means Server errors
  + 502; Bad gateway exception, usually for an incompatible output returned from a Lambda proxy integration backend and occasionally fro out of order invocations due to heavy loads
  + 503: service Unavailable exception
  + 504: Integration Failure – ex eEndpoint Request Timed-out Exception
  + API Gateway requests time out after 29 second max

AWS API Gateway -CORS

* CORS must be enabled when you receive API calls from another domain
* The Options preflight request must contain the following headers:
  + Access-Control-Allow-Methods
  + Access-Control-Allow-headers
  + Access-Control-Allow-Origin
* CORS can be enabled through the console

API Gateway – Security

IAM Permissions

* Create an IAM policy authorization and attached to USer / Role
* Authentication = IAM Authorization = IAM Policy
* Good to provide access within AWS (EC2, Lambda, Iam users)
* Leverages “sig v4” capability where IAM credentials are ion headers

API Gateway – Resource Policies

* Resource Policies ( similar to lambda Resource Policy
* ALlow for cross account access ( combined with IAM Security
* Allow for a specific source IP address Allow fro VPC endpoints

API Gateway – Security

Cognito USer Pools

* COgnito fully managed user lifecycle token expires automatically
* API gateway verifies identity automatically from AWS Cognito
* No custom implementation required
* Authentication = cognito user pools authorization = api gateway methods

API GateWay Security

Lambda Authorizer (formerly custom authorizers)

Token-based authorizer ( bearer token 0 - ex JWT (JSON web token ) or Oauth

* A request parameter-based lambda authorizer ( headers, query string, stage var)
* Lambda must return an IAM policy for the user, result policy is cached
* Authentication = external Authorization = Lambda function

API Gateway – Security – Summary

* IAM:
  + Great for users/roles already within your aws account, + resource policy for cross account
  + Handle authentication + authorization
  + Leverages Sig v4
* Custom Authorizer
  + Great for 3rd party tokens
  + Very flexible in terms of what IAM policy is returned
  + Handle Authentication verification + Authorization in the LAmbda function
  + Pay per Lambda invoaction, results are cached
* Cognito User Pool:
  + You manage yourown user pool (can be backed bby facebook, google login, etc)
  + No need to write any custom code
  + Must implement authorization in the backend

API Gateway – HTTP API vs REST API

* HTTP APIs
  + Low-latency, cost effectiveAWS Lambda proxy, Http proxy apis and private integeraition ( no data mapping)
  + Suupport OIDC and Oaauth 2.0 authorization and built in suuppport for CORS
  + No usage plans andAPI keys
* ReST APIS
  + All features ( except Native Open ID connect / OAuth 2.0)

API Gateway – WebSocket API – overview

* What’s Websocket
  + Two-way interactive communication between a user’s browser and a server
  + Server can push information to the client
  + This enables stateful application use cases
* WebSocket API are often used in real time applications such as chat applications, collaboration platforms, multiplayer games, and financial trading platforms
* Works with AWS services ( Lambda, DynamoDB ) or HTTP endpoints

AWS Cognito

* We want to give our users an identity so that they can interact with our application
* Cognito USer Pools:
  + Sign in functionality for app users
  + Integrate with API gateway & application load balancer
* Cognito Identity pools ( federated identity):
  + Provide AWS credentials to users so they can access AWS resources directly
  + Integrate with cognito user pools as an identity provider
* Cognito Sync:
  + Synchronize data from device to cognito
  + Is deprecated and replaced by AppSync

Cognito vs IAM

The keywords for cognito is

* “Hundreds of users
* “Mobile Users”
* “Authenitcate with SAML”

Cognito is used for users outside of the AWS network whereas IAM is used for people and services already located within the AWS network

Cognito User Pools (CUP) – user Features

* Create a serverless database of user for your web & mobile apps
* Simple login: username ( or email / password combination
* Password reset
* Email & phone number verification
* Multi-factor authentication (MFA)
* Federated Identities: users from Facebook, Google, SAML
* Feature: block users if their credentials are compromised elsewhere
* Login sends back aJSON webtoken (JWT)

Cognito User Pools ( CUP ) – Diagram

SAML = Security application markup language

* CUP integrates with API Gateway and Application Load Balancer
* With Application Load Balancer + Listeners & Rules it uses cognito user pools authenticate

Cognito User Pools - Lambda triggers

* CUP can invoke a lambda function synchroinously on these triggers:

Authentication Events

* PRe Authentication Lambda Trigger
  + Custom validation to accept or deny the sign in request
* Post Authentication Lambda Trigger
  + Event logging for custom analytics
* Pre Token generation Lambda Trigger
  + Augment or suppress token claims

Sign-up

* Pre Sign-up lambda trigger
  + Custom validation to accept or deny the sign-up request
* Post confirmation Lambda trigger
  + Custom welcome messages or even logging for custom analytics
* Migrate User Lambda Trigger
  + Migrate a user from an existing user directory to user pools

Messages

* Custom Message Lambda Trigger
  + Advanced customization and localization of messages

Token Creation

* Pre Token generation Lambda Trigger
  + Add or remove attributes in ID tokens

Cognito USer Pools – Hosted Authentication UI

* Cognito has a hosted authentication UI that you can add to your app to handle sign up and sign in workflows
* Using the hosted UI, you have a foundation for integration with Social logins, OIDC aor SAML
* Can customize with a custom logo and custom css

Cognito Identity Pools (Federated Identities)

* Get identities for “users” so they obtain temporary AWS credentials
* Your identity pool (e.g identity source) can include:
  + Public Providers ( login with amazon, facebook, google, apple)
  + Users in an Amazon Cognito User Pool
  + OpenID connect providers & SAML identity providers
  + Developer Authenticated Identities ( custom login server)
  + Cognito identit pools allow for unauthenticated (guest) access
* Users can then access AWS services directly or through API Gateway
  + The IAM policies applied to the credentials are defined in cogniot
  + They can be customized based on the User\_id for fine grained control

Cognito Identity Pools – with CUP

Cognit Identity Pools – IAM Roles

* Default IAM roles for authenticated and guest users
* Define rules to choose the role fore ach user based ont heuser’s ID you can partition your users access using policy variables
* IAM credentials are obtained by cognito identity pools through STS  
  THe roles must have a “trust” policy of cognito Identity pools

What is the difference between

Cognito USer Pools vs Identity Pools

* Cognto USer Pools:
  + Database of users for your web and mobile application
  + Allows to federate logins through public social, oidc, saml
  + Can customize the hosted UI for authentication
  + Has triggers with AWS lambda to use during the authentication flow
* Cogniot Identity pools
  + Obtain AWS credentials for your users
  + USers can login though PUblic Social, OIDC, SAML & Cogniot USer Pools
  + Users can be unauthenticated (guests)
  + Users are mapped to IAM roles & policies, can leverage policy variables
* CUP + CIP = manage user/ password + access AWS services

Cognito Sync

* Deprecated - use AWS APpSync now
* Store preferences, configuration, state of app
* Cross device synchronization ( any platform)
* Offline capability ( synchronization when aback online)
* Stroe data in datasets ( up to 1MB) up to 20 datasets to synchronize
* Push Sync: silently notify across all devices when identity data changes
* Cognito Stream: stream data from Cognito into Kinesis
* Gognito Events; execute Lambda functions in response to events

AWS Step Functions

* Build Serverless visual workflow to orchestrate your lambda functions
* Represent flow as a JSON state machine
* Features: sequence, parallel, conditions, timeouts, error handling…
* Can also integrate with EC2, ECS on premise servers, API Gateway
* Max execution time of 1 year
* Possiblity to implement human approval feature
* Use cases:
  + Order fulfillment
  + Data processing
  + Web applications
  + Any workflow
* Can see a visual workflow in teh step functions

Step Functions – Error Handling

* Any state can encounter runtime errors fro various reasons:
  + State machine definition issues ( for example, no matching rule in a cChgoice sate)
  + Task failures (for example, an exception in a lambda function
  + Transient Issues ( for example network partition events)
* By default, when as state reports an error, AWS step functions causes the execution to fail entirely
* Retrying failures - Retry: intervalSeconds, MaxAttempts, BackoffRate
* Moving on - Catch: errorEquals, Next
* Best practice is to include data in the error messages

Step Functions – Standard vs Express

Standard Workflows

* Maximum duration
  + 1 year
* Supported execution start rate
  + Over 2k per second
* Supported state transition rate
  + Over 4k per second per account
* Pricing
  + Priced per state transition. A state transition is counted each time a step in your execution is completed ( more expensive)
* Execution history
  + Executions can be listed and described with Step Functions APIs, and visually debugged through the console. They can also be inspected in Cloudwatch logs by enabling logging on your state machine
* Execution semantics
  + Exactly once workflow execution

Express Workflows

* Maximum duration
  + 5 minutes
* Supporte execution start rate
  + Over 100,000 per second
* Supported State transition rate
  + Nearly unlimited
* Pricing
  + Priced by the number of executions you run, their duration, and memory consumption ( cheaper)
* Execution history
  + Executions can be inspected in cloudwatch logs by enabling logging on your state machine
* Execution semantics
  + At least once workflow execution

AWS APpsync - Overview

* AppSync is a managed service that uses GraphQL
* GraphWL makes it easy for applications to get exactly the data they need
* This includes combining data from one or more sources
  + NoSQL data stores, Relational databases, HTTP APIS..
  + Integrates with DynamDB , Aurora, Elasticsearch & others
  + Custom sources with AWS Lambda
* Retrieve data in realtime with websocket or MQTT on Websocket
* FOr mobile apps: local data access & data synchronization
* It all starts with uploading one GraphWL schema

AppSync – Security

* THere are four way s you can authorize applications to interact with AWS aAPP Sync Graph WL API:
* API\_KEy
* AWS\_IAM: IAM users/ roles/ cross-accounts access
* OPENID\_CONNECT: OPenID Connect provide ? JSON WebToken
* AMAZON\_COGNITO\_USER\_POOLS
* For custom domains & HTTPS, use CloudFront in front of APPSync

BEGINNING Moving Forward AGAIN

Load Balancer Stickiness

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

Cross-ZOne Load Balancing

* With Cross Zone Load Balancing: each load balancer instance distributed evenly across all registered instances in all AZ
* Best way to spread out traffic across a network
* Otherwise, each load balancer node distributes requests evenly across the registered instances in its AZ only.

Cross-ZOne Load Balancing

* Classic Load balancer
  + Disabled by default
  + No charges for inter AZ data if enabled
* Application Load Balancer
  + Always on ( can’t be disabled)
  + No charges for inter AZ data
* Network Load Balancers
  + Disabled by default
  + You pay charges for inter AZ data if enabled

SSL? TLS - basics

* An SSL Certificate allows traffic between your clients and your load balancer to be encrypted in transit ( in-flight encryption
* SSL refers to secure sockets Layer , used to encrypt connections
* TLS refers to transport layer security which is a newer version
* Nowadays, tsls certificates are mainly used, but people still refer as SSL
* Public SSL certificates are issued by certificate authorities (CA)
* Comodo,Symantac, godaddy, globalsign, digicert, letsencrypt, etc
* SSL certificates have an expiration date ( you set) and must be renewed

Load balancer - SSL Certificates

* The load balancer uses an x.509 certificate ( SSL/TLS Server certificates)
* You can manage certification using ACM ( AWS certificate MAnager)
* You can create, upload your own certificates alternatively
* HTTPS listener
  + YOu must specify a default certificate
  + You can add an optional list of certs to support multiple domains
  + Clients can use SNI (Server name Indications) to specify the hostname they reach
  + Ability to specify a security policy to support older veisons of SSL/ TLS ( legacy clients)

SSL - server name indication

* SNI solves the problem of loading multiple SSL certificates on one web server ( to serve multiple websites)
* It’s a “newer” protocol, and requires the client to indicate the host name of the target server in the initial SSL handshake
* The server will then find the correct certificate or return the default one
* NOTE  
  Only words for application load balancer & network load balancers ( newer generation) , CloudFront
* Does not work for classic load balancer

Elastic Load Balancers – SSL certificates

* Classic Load Balancers (v1)
  + Supports only one SSL certificate
  + Must use multiple CLB for multiple hostname with multiple SSL certifcates
* Applciation Load Balancer (v2)
  + Supports multiple listeners with multiple SSL certifcates
  + Uses server name indication (SNI ) to make it work
* Network Load Balancer (v2)
  + Supports multiple listeners with multiple SSL certificates
  + Uses server name indication (SNI) to make it work

ELB – Connection Draining

* Feature naming:
  + CLB: Connection draining
  + Target Group: Deregistration Delay (for ALB & NLB)
* What is connection draining and deregistration delay
  + The time to complete in flight requests while the instance is de-registering or unhealthy
  + Stops sending new request s to the instance which is de-registering
* Basically if a instance becomes unhealthy it brings waiting for existing connections to complete their work, and no further traffic will be sent to it
* Between 1 to 3600 seconds default is 300 seconds
* Can be disabled ( set value to 0)
* Set to a low value if your requests are short

Auto Scaling Groups – Scaling Policies

* Target Tracking Scaling
  + Most simple and easy to set-up
  + Example: I want the average ASG CPU to stay at around 40%
* Simple/ Step Scaling
  + When a cloudwatch alarm is triggered (example CPU > 70%), then add 2 units
  + When a cloudwatch alarm is triggered (example CPU < 30%), then remove 1 unit
* Scheduled Actions
  + Anticipate a scaling based on known usage patterns
  + Example: increase the min capacity to 10 at 5pm on Fridays

Auto Scaling Groups Scaling Cooldowns

* The cooldown period helps to ensure that your auto scaling group doesn't launch or terminate additional instances before the previous scaling activity takes effect.
* In addition to default cooldown for auto scaling group we can create cooldowns that apply to a specific simple scaling policy
* A scaling-specific cooldown period overrides the default cooldown period.
* One common use for scaling-specific cooldowns is with a scale-in policy --- a policy that terminates instances based on specific criteria or emetric. Because this policy terminated instances, Amazon EC2 auto scaling needs less time to determine whether to terminate additional instances
* If the default cooldown period of 300 seconds is too long you can reduce costs by applying a scaling specific cooldown period of 180 seconds to the scale in policy
* If your application is scaling up and down multiple times each hour, modify the auto scaling groups cool down timers and the cloudwatch alarm period that triggers the scale in

What’s an EBS Volume?

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

EBS Volume

* It’s a network drive (ie not a physical drive)
  + It uses the network to communicate the instance, which means there might be a bit of latency
  + It can be detached from an EC2 instances and attached to another one quickly
* It’s locked ito an AZ
  + An EBS Volume in us-east-1a cannot be attached to us-east-1b
  + To move a volume across you first need to snapshop it
* Have a provision capacity ( size in GBs, and IOPS)
  + You get billed for all the provisioned capacity
  + You can increase the capacity of the drive over time

EBS Volume Types

* EBS volumes come in 4 types
  + GP2 (SSD)
    - General purpose SSD volume that balances price and performance for a wide variety of workloads
  + IO 1(SSD):
    - Higest-peformance SSD volume for mission critical low latency or high thorughoput workloads
  + ST 1 (HDD):
    - Low cost HDD volume designed for frequently accessed, throughput-intensive workloads
  + SC1 HDD)
    - Lowest cost HDD volume designed for less freequently accessed workloads
* EBS volumes ar characterized in Size | Thorughput | IOPS (I/O ops per sec)
* When in doubt always consult he aws documentation
* Only GP2 and IO1 can be used as boot volumes

EBS Volume Types USe cases GP2

* Recommended for most workloads
* System boot volumes
* Virtual desktops
* Low-latency interactive apps
* Development and test environments
* 1 GiiB - 16TiB
* Small gp2 volumes can burst IIOPS to 3000
* Max OPS is 16k
* 3 IOPS per GB means at 5,334 Gb the IOPS will not increase anymore

EBS Volume Types USe cases IO 1

* Critical business applications that require sustained IOPS performance, or more than 16k IOPS per volume ((gp2 limit)
* Large database workloads, such as:
* MongoDB, cassandra, microsoft sql server, mysql, psostgrsql oracle
* 4 gib - 16 Tib
* IOPS is provisioned (PIOPS) - min 100 max 64k (nitro instances) else max 32 k (other instances)
* The max ratio of provisioned IOPS to requested volume size (inGiB) is 50:1

EBS Volume Types Use cases

ST1

* Streaming workloads requiring consistent, fast throughput at a low price
* Big data, Data warehouses, loog processing
* Apache kafka
* Cannot be aboot volume
* 500gib - 16 tib
* Max IOPS is 500
* Max throughput of 500 mib/s - can burst

EBS Volume Types Use Cases SC1

* througput -oriented storage for large volumes of data that is infrequently accessed
* Scenarios where the lowest storage cost is important
* Cannot be a boot volume
* 500 gib - 16Tib
* Max IOPS is 250
* Max throughput of 250 miB/s can burst

EBS – Volume Types Summary

* gp2 : general purpose voumes (cheap)
  + 3 IOPS / Gb minimum 100 IOPS burst to 3k IOPS, max 16k IOPS
  + 1gb -16TB + 1TB = +3000 IOPS
* Io1: Provisioned IOPS (expensive)
  + Min 100 IOPS, Max 64k IOPS (nirto) or 32k (other)
  + 4 gb - 16 tb. Size of voumes and IOPS are independent
* St1: throughput optimized HDD
  + 500gb - 16TB, 400 mb/sec throughput
* Sc1: cold HDD infrequently accessed data
  + 250 Gb - 16 Tb
  + 250 mb/s throughput

Difference between EBS vs Instance store

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

Local EC2 Instance Store

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

EFS – Elastic File System

* Managed NFS ( network file system) that can be mounted on many EC2
* EFS works with EC2 instances in multi-AZ
* Highly available , scalable, expensive, pay per use
* Attached a security group to it that will manage the connections

Use Cases

* Content management
* Web serving
* Data sharing
* Wordpress

Uses

* NFSv4.1 protocol
* Uses security grup to control access to EFS
* Compatible only with linux based AMI
* Encryption at rest using KMS
* POSIX file system (~Linux) that has a standard file API
* File System scales automatically, pay-per-use no capacity planning

EFS Performancing & Storage Classes

* EFS Scale
  + 1000s of concurrent nFS clients, 10Gb /s throughput
  + Grow to petabyte-scale network file system, automatically
* Performance mode( set at EFS creation time)
  + General purpose (default); latency sensitive use cases ( web server, cms, etc..)
  + Max I/O – higher latency throughput highly parallel, ( big data , media processing)
* Storage Tiers ( lifecycle management feature – move file after N days)
  + Standard: for frequently accessed field
  + Infrequently Access (EFS-IA): cost to retrieve files, lower price to store

EBS vs EFS – Elastic Block Storage

* EBS volumes
  + Can be attached to only one instances at a time
  + Are locked a the AZ level
  + Gp2: IO increases if the disk size increases
  + Io 1: can increase IO independently
* To migrate an EBS volume across AZ
  + Take a snapshot
  + Restore the snapshot to another AZ
  + EBS backups use IO and you shouldn’t run them while your application is handling a lot of traffic
* Root EBS volumes of instances get terminated by default if the Ec2 instance gets terminated ( you can disable that)

EBS vs EFS – Elastic File System

* Mounting 100s of instances across AZ
* EFS Share website files ( wordpress)
* Only for linux instances (POSIX)
* EFS has a higher price point than EBS
* Can leverage EFS-IA for cost savings

AWSRoute 53 Overview

* Roue53 is a Managed DNS (Domain Name System)
* DNS is a collection of rules and records which helps clients understand how to reach a server through its domain name
* In AWS, the most common records are:
  + A: hostname to IPv4
  + AAAA: hostname to IPv6
  + CNAME: Hostname to hostname
  + Alia: hostname to AWS Resource

AWS ROute 53 Overview

* Route 53 can use:
  + Public domain names you own ( or buyg)
  + Private domain names that can be resolved by your instance s in your VPCs
* Route53 had advanced features such as:
  + Load balancing (through DNS –also called client laid balancing
  + Health Checks (although limited)
  + Routing policy : simple, failover, geolocation, latency, weighted, multi value
* You pay .50$ per month per hosted zone

DNS Records TTL (time to live)

* High TTL: (e.g. 24 hrs)
  + Less traffic on DNS
  + Possibly outdated records
* Low TTL eg 60s
  + More traffic on DNS
  + Records are outdated for less time
  + Easy to change records
* TTL is mandatory for each DNS record

CNAME vs Alias

* AWS Rsources expose an AWS hostname: and you want to route a domain name to it
* CNAME
  + Points a hostname to any other hostname
  + Only for non root domain
* Alias
  + Points a hostname to an aws resource
  + Works for root domain and non root domain
  + Free of charge native health check

Simple Routing Policy

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

Weighted Routing Policy

* Control the % of the requests that go to specific endpoint
* Helpful to test 1% of traffic on new app version for example
* Helpful to split traffic between two regions
* Can be associated with health checks

Latency Routing Policy

* Redirect to the server that has the least latency close to us
* Super helpful when latency of a user is priority
* Latency is evaluated in terms of user to designated AWS region
* Germany may be directed to the US if that’s the latency

Health Checks

* Have X health checks failed => unhealthy (default 3)
* After x health checks passed => health (default 3)
* Must be in a row
* Default Health Check interval : 30 s ( can set to 10 s - higher cost)
* About 15 checkers will check the endpoint health
* => one request evey 2 seconds on average
* Can have HTTP, TCP and HTTPS health checks ( no SSL verification)
* Possibility of integrating the health check with cloudwatch
* Health checks can be linked to Route53 DNS queries

Failover Routing Policy

* Primary and Secondary instances
* When a health check fails on the primary the failover will send connections to the secondary

Geolocation Routing Policy

* Different from Latency based
* This is routing based on user location
* Here we specific traffic from the UK should go to this specific IP
* Should create a “default” policy (in case there’s no match on location)

MultiValue Routing Policy

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

VPC & Subnets Primer

* VPC: private network to deploy your resources (regional resource)
* Subnets allow you to partition your network inside your VPC (AZ resource)
* A Public Subnet is a subnet that is accessible from the internet
* A private subnet is a subset that is n0ot accessible from the internet
* To define access to the internet and between subsets, we use route tables

Internet Gateway & NAT Gateways

* Internet Gateways helps our VPOC instances connect with the internet
* Public Subnets have a route to the internet gateway
* NAT Gateways (AWS managed) & NAT instances (self-managed) allow your instances in your Private Subnets to access the internet while remaining private

Network ACL & Security Groups

* NACL ( network ACL )
  + A firewall which controls traffic from and to subnet
  + Can have allow and deny rules are attached at the subnet level
  + Rules only include IP addresses
* Security Groups
  + A firewall that controls traffic to and from an EnI / an EC2 instance can have only allow rules
  + Rules include IP addresses and other security groups

VPC Flow Logs

* Capture information about IP traffic goinginto your interfaces:
  + VPC Flow Logs
  + Subnet Flow Logs
  + Elastic Network Interface Flow Logs
* Helps to monitor & trouble shoot connectivity issues
* Caputes network information from aws managed interfaces too: Elastic Load Balancers, Elastic

S3:ObjectCreated, Sw3:ObjectRemoved

S3ObjectRestore, S3:Replication

AWS Lambda Limits to know

Execution

* Mmeory allocation 128 - 3008 in 64 increments
* Max execution time: 900 seconds
* Environment variabl;es
* Dis k capacity in the function
* Concurrency executions 1000

Dpeoployment

* 50 mb max
* Uncompressed should be 250
* /tem to load other files at start p
* Size of env varas is 4KB