

AWS Developer Associate Notes



Contents

Introduction.....	22
1. 19,000 foot overview.....	22
Compute	25
Storage	25
Databases	26
Migration	27
Network & content delivery.....	27
Developer Tools	27
Management tools.....	28
Media services	29
Machine learning	30
Analytics	30
Security, Identity & Compliance	31
Application Integration	32
Customer Engagement.....	33
Business Productivity	33
Desktop & App Streaming.....	33
Mobile Services.....	34
IoT.....	34
Game development	35
2. Important Service for.....	35
1. AWS SA Associate exam.....	35
2. Developer Associate exam.....	36
3. Sysops Administrative Associate.....	36
3. Identity access management.....	37
Terms.....	38
Lab Setup	38

Dashboard.....	38
Create Roles	39
IAM Federation	40
Points to remember.....	40
Create Policy	40
4. Setting up billing alert.....	41
Setting up Budget setup.....	41
5. S3 – Simple Storage Servers.....	42
Object based.....	42
S3 basics	43
Moving between storage classes	47
S3 – Charges.....	47
Creating S3 Bucket	48
Uploading Files on S3.....	48
Versioning.....	50
LAB	50
Enable MFA Delete	51
S3 Access Logs.....	53
LAB	54
S3 Replication (Cross-Region replication and Same-Region Replication).....	54
LAB	55
S3 Pre-Signed URLs	55
LAB	56
Lifecycle Management, IA, S3 & Glacier lab	56
Encryption.....	57
LAB - I	60
LAB - II	60
Security.....	61
LAB	63

Storage Gateway.....	64
Snowball	66
Transfer Acceleration.....	69
Creating static website using S3	70
LAB	70
CORS	71
Points to remember	72
S3 Performance	72
KMS Limitation.....	74
S3 Select & Glacier Select.....	75
S3 Event notifications	76
LAB	76
AWS Athena.....	77
LAB	77
S3 Object lock & Glacier Vault Lock.....	78
6. Elastic Compute Cloud (EC2)	78
Launching an EC2 instance.....	79
How to use Putty	81
EC2 pricing	83
AMI	83
Security Groups.....	84
Points to remember	85
Referencing other security groups.....	86
Private vs Public IP	86
Elastic IP.....	87
LAB	87
Apache web server	88
EC2 User Data	89
EC2 Options	89

On Demand	89
Reserved.....	90
Spot	90
Dedicated hosts	91
Dedicated Instances.....	92
Which host is right for us?	92
Pricing.....	93
LAB.....	93
Elastic Network Interface	95
LAB	96
EC2 instances types.....	96
T2 Unlimited	98
EBS	99
LAB	100
EBS volume type use cases	101
EBS encryption	102
EBS vs Instance store	103
Reserved Instances	103
Upgrading EC2 Volume Types	104
Elastic File System (EFS)	106
Performance & Storage Class.....	107
LAB	108
RAID volumes & Snapshots	110
RAID.....	110
Snapshots.....	112
Create an AMI.....	112
AMI Types	113
Instance Store Root Device	113
7. Load Balancers	114
Scalability & High Availability	114
Vertical Scalability.....	114

Horizontal scalability.....	115
High Availability	115
Introduction to Load Balancer.....	115
Usage.....	116
Health Checks.....	116
Load Balancer Security Groups	117
Types	118
Application Load Balancer	118
Classic Load Balancer	122
Network Load Balancers	124
Load Balancer Stickiness	126
LAB	126
Cross Zone Load Balancing.....	127
SSL/TLS.....	128
Server Name Indication (SNI).....	129
ELB - SSL certificates.....	130
LAB	130
Connection Draining	131
Command Line Interface (AWS CLI).....	132
AWS CLI Configurations	132
AWS CLI Dry runs.....	133
AWS CLI STS Decode errors.....	133
IAM Roles.....	134
Policy Simulator	135
Bash Scripting	135
Instance Metadata	136
AWS Limits	137
AWS SDK	137
Security Credentials	138
Default credentials provider chain.....	138

Signing AWS API request.....	139
Exponential Backoff	139
Auto scaling Groups	140
Auto Scaling Alarms	141
LAB	142
Scaling Policies	144
Scaling Cooldowns	145
LAB	145
Placement Groups.....	146
Cloud History	147
IAAS.....	147
PAAS.....	147
Containers.....	147
Lambda.....	147
7. DNS	149
SOA records	150
NS records.....	150
A records.....	151
TTL	151
LAB	152
C Names.....	153
Alias records	153
LAB	154
8. Route 53	155
LAB.....	156
Simple	157
LAB	157
Weighted Routing	158
LAB	158
Latency	159

LAB	161
Health Checks	161
LAB	162
Failover Routing Policy.....	162
Lab.....	163
Geo Location Routing.....	165
LAB	165
Multi Value	166
LAB	167
9. Elastic Beanstalk	167
Overview.....	167
LAB - I.....	169
LAB – II	170
Deployment options for updates	170
LAB	177
Elastic Beanstalk CLI.....	177
Beanstalk lifecycle policy	178
LAB	178
Elastic Beanstalk extension	179
Web Server vs Worker Environment.....	180
RDS with Elastic Beanstalk	181
Docker	182
Single Docker.....	182
Multi-Docker	183
Points to remember.....	183
9. CloudFront CDN	185
Origins.....	186
Distributions	188
CloudFront vs. CORS	188

LAB.....	189
Caching	190
LAB	192
Security.....	192
CloudFront Signed URL/Cookies.....	193
CloudFront Signed URL vs. S3 Pre-Signed URL.....	194
9. Databases	195
Relational Database	195
Non-Relational Database	196
Data Warehousing	196
OLTP vs OLAP	197
RDS Backups, Multi AZ & Read replicas.....	197
RDS	197
RDS Backup	198
Multi-AZ	199
Read Replicas	200
LAB	203
RDS security	205
Encryption	205
IAM Authentication.....	207
Points to Remember	207
DynamoDB	208
Pricing.....	208
Setting up DynamoDB table.....	209
RedShift	210
RedShift Configuration.....	210
RedShift – 10 times faster.....	210
Massive parallel processing (MPP)	211
RedShift Pricing	211
RedShift Security.....	212
RedShift availability.....	212

ElastiCache.....	212
Types	214
User Session store.....	216
LAB	216
Points to remember	217
Caching implement consideration	217
Lazy Loading/Cache-Aside/Lazy Population.....	218
Write Though	219
Cache Evictions and Time-to-Live	220
Aurora.....	221
Aurora Scaling	221
Aurora Replicas	222
DB Cluster.....	223
Features	224
Security	224
Serverless	225
Global Aurora	225
LAB	226
10. VPC	227
Applications	230
Default VPC vs Custom VPC.....	230
VPC peering	231
LAB.....	232
Network Access Control Lists vs. Security Groups	236
NACL.....	237
Security Groups.....	237
Custom VPC's & ELB.....	239
VPC Flow logs.....	239
LAB	240
NAT vs Bastion	241
VPC end points.....	242

Site to Site VPN	243
Direct Connect	243
VPC Clean-up	244
Example: A 3-tier architecture	245
LAMP Stack on EC2	246
11. SQS – Simple Queue Service.....	247
12. CICD	247
Continuous Integration	247
Continuous Delivery.....	248
Technology stack	249
Code Commit	249
Code Commit Security	249
CodeCommit notifications	250
CodeCommit vs GitHub.....	250
Similarities.....	250
Differences	251
LAB – CodeCommit	251
CodePipeline	253
CodePipeline Artifacts.....	253
CodePipeline Troubleshooting.....	254
LAB	254
CodeBuild.....	256
Working.....	256
BuildSpec.....	257
Local Build	257
LAB	258
CodeBuild in VPC.....	259
CodeDeploy	259
Requirements.....	260
Working.....	260

Points to remember	261
Primary components.....	261
CodeDeployAppSpec.....	262
Deployment configuration.....	262
LAB	263
CodeDeploy for EC2 and ASG	266
EC2	266
ASG.....	266
Rollback	267
CodeStar	267
LAB	268
13. CloudFormation	268
Introduction.....	269
Infrastructure as Code	269
Cost	270
Working.....	270
Deploying templates.....	270
Building Blocks	270
LAB.....	271
YAML.....	272
Resources	273
Points to remember	274
Parameters	274
How to reference a parameter	275
Pseudo Parameters.....	275
Mappings	275
Fn::FindInMap – Accessing mapping values	276
Mapping vs Parameters	276
Outputs.....	277
Conditions.....	278

Intrinsic Function	279
Rollbacks.....	281
LAB	281
Changeset, Nested Stack, and stack set	281
Changeset.....	281
Nested Stack	281
Cross stack vs. Nested stack.....	282
StackSet.....	283
14. Monitoring.....	283
CloudWatch	284
Dashboard.....	284
Alarm.....	284
Events.....	285
Logs	286
Metrics	288
EventBridge.....	290
EventBridge vs. CloudWatch.....	290
X-Ray.....	290
Advantages.....	291
Compatibility.....	292
Working.....	292
Enabling X-ray	292
Output.....	293
Troubleshooting.....	294
LAB	294
API	296
Exam-tips.....	297
CloudTrail.....	301
LAB	302
CloudTrail vs CloudWatch vs X-Ray	302
15. AWS Integration & Messaging.....	302

SQS	303
Standard Queue	303
Delay Queue.....	307
Dead Letter Queue.....	309
Long Polling.....	309
LAB-I Console Hands On.....	310
LAB-II Dead Letter Queue	311
SQS using CLI.....	312
FIFO Queue	313
Features	314
LAB	314
SQS Advanced	315
Extended Client.....	315
Security	316
API's.....	316
SNS.....	316
SNS Integration	318
How to publish	318
SNS + SQS: Fan Out	318
LAB	319
Kinesis.....	319
Working.....	319
Kinesis Stream Overview.....	320
Shards.....	321
Kinesis API – Put records.....	322
Kinesis API – Consumers	323
LAB	324
KCL.....	325
Kinesis Security	326
Kinesis Data Analytics	327
Fireose.....	327
Difference between SQS, SNS & Kinesis	327

16. AWS Lambda.....	328
Overview.....	329
Why Lambda?	329
Benefits	329
Example.....	330
Pricing.....	331
LAB	331
Configuration parameters.....	333
Synchronous Invocation.....	334
Integration with ALB	336
Concurrency & Throttling.....	338
Concurrency Issue.....	338
Concurrency and Async Invocations	339
Cold Start & Provisioned Concurrency.....	339
Lambda Retries & DLQ	340
LAB	341
CloudWatch Events/EventBridge	342
S3 event notification	343
Logging Monitoring & X-Ray Tracing	344
LAB	344
Limits to know.....	345
Event Source Mapping	346
Streams	346
SQS & SQS FIFO.....	348
Lambda Event Mapper Scaling.....	349
Destination.....	349
Lambda Resource Execution Role.....	350
Resource Based Policies.....	351
Environment Variables.....	351
Lambda in VPC	352
Performance	354

Qualifiers	355
Lambda Versions.....	355
Lambda Aliases.....	356
LAB	357
Lambda & CodeDeploy	358
External Dependencies	359
LAB	360
Lambda & CloudFormation	361
Lambda layers.....	362
tmp space	363
Best Practices.....	363
LAB	364
Lambda@Edge	365
Global Application.....	366
Use Cases	367
17. DynamoDB	367
Traditional architecture	367
NoSQL Databases.....	368
Features.....	368
Primary Keys	369
LAB.....	371
Provisioned Throughput.....	372
Write Capacity Units	372
Strongly Consistent Reads vs. Eventually Consistent Reads	372
Read Capacity Units	373
LAB	373
Partition Internal.....	374
Throttling	374
API's	374
Writing Data.....	374

Deleting Data	375
Batching Writes.....	375
Reading Data.....	376
Querying Data	376
Scanning Data	377
LAB	377
Local Secondary Index.....	378
Global Secondary Index	378
Indexes & Throttling	380
LAB	380
Concurrency.....	382
DAX	383
LAB	384
Streams.....	385
LAB	386
TTL	388
LAB	388
CLI	389
LAB	389
Transactions.....	390
Session State.....	391
Partition strategies.....	391
Write Types.....	392
DynamoDB patterns with S3	394
Large Objects Pattern	394
Indexing objects with S3	394
Operations	395
Security & Other features	396
18. Serverless API gateway & Cognito.....	396
API Gateway.....	397

Integration	397
LAB - I	398
Deployment Stages	399
Stage Variables.....	399
Stage Variables & Lambda Aliases	399
Canary Development	400
LAB – II.....	401
Mapping Templates	402
Swagger / Open API Spec.....	404
Caching.....	405
Monitoring	406
Cross-Origin Resource Sharing (CORS).....	407
Usage plans & API Keys.....	408
Security	409
Cognito	412
Cognito User Pools (CUP).....	412
Federated Identity Pools.....	413
AWS Cognito Sync	414
Cognito user pools vs. identity pools	414
19. Serverless Application Model (SAM)	415
Overview.....	415
Recipe.....	415
SAM Deployment	416
SAM CLI.....	417
Installation	417
LAB	418
Policy Templates	419
20. Docker, ECS, ECR, Fargate	420
Docker	420
Storage.....	421
Docker vs. Virtual Machine.....	422
Docker Container Management.....	423

ECS (Elastic Container Service)	423
Introduction	423
ECS Task definition	425
ECS Service	427
ECS with Load Balancer	429
ECR (Elastic Container Registry)	432
Introduction	432
LAB	432
Fargate	434
LAB	434
ECS IAM Role Deep Dive	436
LAB	438
ECS Tasks	439
Placement	439
Process	440
Strategies	440
Constraints	442
ECS - Service Auto Scaling	443
ECS - Cluster Capacity Provider	444
Summary	446
21. Other Serverless: Step Functions & AppSync	448
AWS Step functions	448
Error handling	449
Step Functions - Standard vs. Express	450
LAB	450
AppSync	453
Overview	453
Example	454
Diagram	455
Security	455
LAB	456

22. Advanced Identity	457
AWS STS - Security Token Service	457
Using STS to Assume Role	458
Cross account access with STS	459
STS with MFA	459
Authorization Model.....	460
IAM Policies & S3 Bucket policies	461
Dynamic Policies with IAM.....	462
Inline vs. Managed Policy.....	463
Granting User Permission	464
Example.....	465
Microsoft Active directory (AD).....	466
AWS Directory Services.....	467
LAB	468
23. AWS Security & Encryption	469
Encryption.....	469
Encryption in Flight (SSL).....	469
Server Side Encryption at REST	469
Client Side Encryption	470
KMS	470
Customer Master Key	472
Uses.....	473
Key Policies.....	473
Request Quotas.....	473
SSE - KMS.....	474
S3 Bucket policies.....	476
API - Encrypt & Decrypt	478
Encryption SDK.....	480
SSM Parameter Store.....	480
Hierarchy.....	481
Standard vs. Advanced Parameter Tier.....	483

Parameter Policies (for Advanced Parameters).....	483
Secret Manager.....	484
SSM Parameter Store vs. Secret Manager.....	484
CloudWatch logs - Encryption	485
CodeBuild Security	485
24. Other Services.....	486
Simple Email Service	486
Summary of Databases	486
ACM - AWS Certificate Manager	487

Introduction

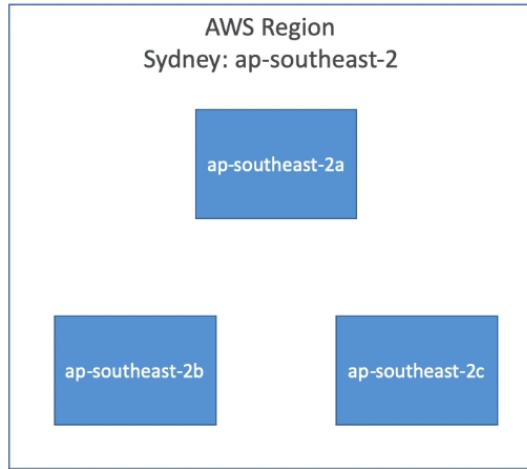
- AWS is a Cloud Provider.
- It provides us with the servers and services that we can use in demand and scale them easily.

1. 19,000 foot overview



Region – It is a geographical area consisting of 2 or more availability zones. It is a cluster of data centers. Most services are region scoped which means if we use a service in Region A, & if we use the same service in Region B, we'll not have our data being replicated or synchronised.

If we navigate to the URL: <https://aws.amazon.com/about-aws/global-infrastructure/> and then navigate to AWS Regional Services, under regional Table we can see the list of Service in the different regions by choosing any of the region.

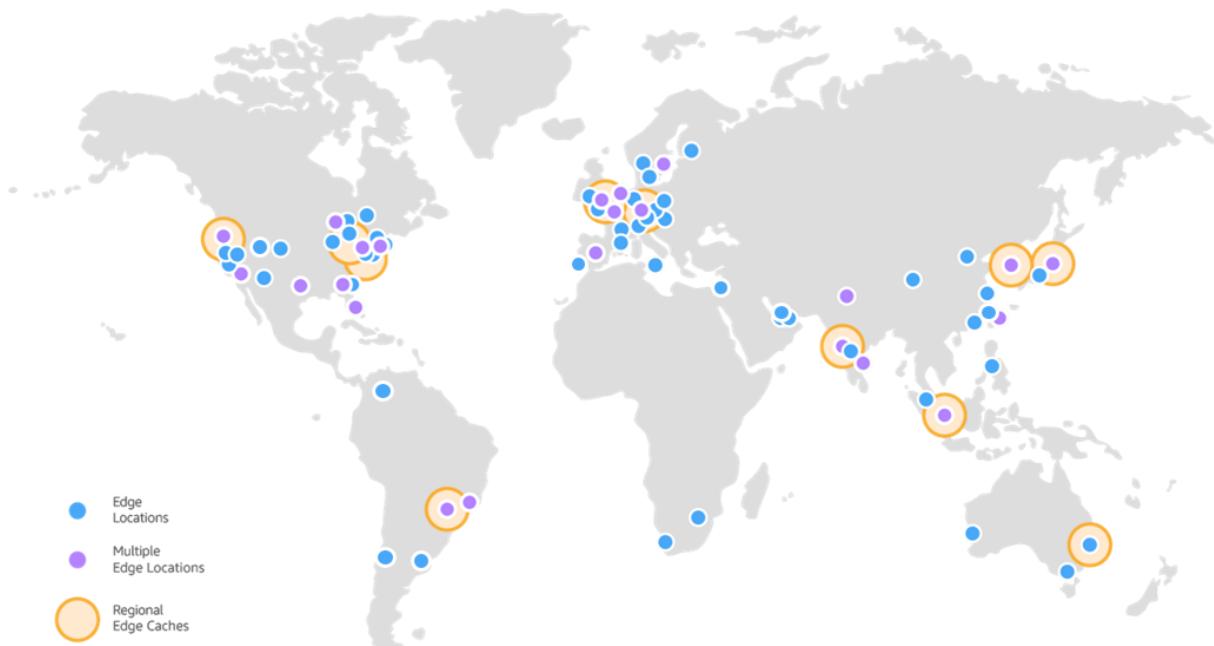


Availability zone – Each region can have multiple availability zones (AZ). It is one or more physical & discrete (geographically isolated) data center, each with redundant power, networking and connectivity, housed in separate facilities (so that they can be separated from disasters). One subnet equals one availability zones. Each consoles/services are region scoped (except IAM and S3). The zones are connected with each other with high bandwidth, ultra-low latency networking. So, we've a connectivity between all these availability zones.

Origin – This is the origin of all the files that the CDN will distribute. This can be either an S3 bucket, an EC2 instance, an Elastic load balancer or Route 53.

Global Infrastructure

Edge locations - They are endpoints for AWS which are used for caching content. Typically, consist of CloudFront (Amazon CloudFront is a fast content delivery network service, that securely delivers data, videos and applications and APIs to customer globally with low latency), CDN (content delivery network – It is a system of distributed servers k/a network that deliver pages and other web content to user based on geographic locations of user, the origin of the webpage, and the content delivery server).



The AWS infrastructure consists of many services like, compute, storage, databases, migration, network and content delivery, developer tools, developer tools, management tools, media services, ML, IoT, etc.

There is a relation between AWS endpoints and AWS edge location. AWS endpoints are used to connect to AWS web console service to configure AWS services such as EC2, S3 etc. by a company or engineers whereas AWS Edge location is where this service is served to the clients. It is a good practice to deploy our servers in minimum 2 availability zones

Compute

EC2 - Elastic compute cloud. These are the virtual machines inside the AWS platform and physical dedicated machines as well.

We have **EC2container services** where we run and manage Docker container.

Elastic Beanstalk – This service is used by the developers who focuses on how to code. The service automatically handles the details of capacity provisioning, load balancing, scaling and application health monitoring.

Lambda is a serverless compute service that runs your code in response to events and automatically manage the underlying compute resources for you. It automatically scales on the amount of work you feed into it.

Lightsail instance is a virtual private server. It is designed to make project easier. It automatically configures networking, access and security environments.

Batch processing – when minimum user interactions is required.

Storage

S3 – Simple storage services. These are usually referred as buckets where we upload our image.

EFS – Elastic File System. It provides a durable, high throughput file for content management system and web serving application.

Glacier – It is used for archiving the data, like if we don't need our data in our daily practices and is cheap.

Snowball – It is a petabyte data transfer solution that uses devices designed to be secure to transfer large amounts of data into & out of the AWS cloud. It is simple, fast and more secure and can be little as 1/5th of the cost of transferring data via internet.

Storage gateway – These are virtual machines that we install in datacenter/head-office & replicate info back to S3.

Databases

RDS – It stands for relational database service like MySQL, MS SQL server, Post GRE SQL

DynamoDB – It is a non-relational DB.

Elasticache – It is a fully managed in-memory data store and cache service by AWS. The service improves the memory of managed Web applications by retrieving info from managed in-memory caches instead of relying on slower disk based DB.

Amazon RedShift – It is an internet hosting and data warehouse product, forms larger part of cloud computing platform. It is built on the top of technology from the massive parallel processing data warehouse company to handle large scale datasets and database migration.

Migration

Amazon Migration Hub – It is a tracking service used to track when we migrate to AWS and integrates them with other services within migration framework.

Application discovery service – This is automated set of tools which detects what application we have, what their dependencies are.

Database Migration service – It is used to migrate DB from on premise to AWS.

Server Migration service – It is used to migrate virtual & physical servers up into AWS cloud.

Network & content delivery

VPC – It is a virtual private cloud (basically a virtual data center). We can configure things like firewall, availability zones, etc.

Cloudfront – It is amazon CDN. Suppose some of your media files are located in London, and you're in North America, then CloudFront stores and provide these files more locally.

Route53 – It is amazon DNS.

API gateway – It is a way of creating API for other services to talk to.

Direct connect – It is a way of running dedicated line from your cooperate head office or from your data center directly into AWS or to your VPC.

Developer Tools

CloudStar – Cloud based service used for creating, managing and working with software development. One can quickly, develop, build & deploy on AWS.

CodeCommit – It act as a place to store your code.

CodeBuild – Compiles the code, run test cases against it and produce software packages that are ready to deploy.

CodePipeline – A continuous integration and release automation service for application you want to release in the cloud. It can pull source code for your pipeline directly from AWS CodeCommit, GitHub, Amazon ECR or Amazon S3.

X-Ray – It help developers analyze and debug production, distributed application, such as those built using microservices structure.

Management tools

CloudWatch – It is a monitoring service.

CloudFormation – It is a way of scripting infrastructure using code.

CloudTrail – It is used to log changes to your AWS environment. It stores record for 1 week.

Config – Monitors the configuration of entire AWS environment.

OpsWorks – It is a similar tool like ElasticBeans. It is used to automate the configuration of the environment.

Service Catalogue – Manages the catalogue of IT services that are used on AWS. It can be OS, software, DB, etc

System manager – It is used to manage AWS resources. We can group our resources in sharepoint application.

Trusted advisor – Gives advice across multiple disciplines like security, how much we are using our AWS services (save money).

Managed Services– Automates common activities like change activities like change request, monitoring, patch management, security and backup services, and provides full life-cycle services to provision, run and support your infrastructure.

Media services

ElasticTranscoder – It is a media transcoding in the cloud. It is designed to be highly scalable, easy to use & cost effective way for developers and business to convert media files from their source format into version that'll playback on devices like smartphone, tablets and PCs

MediaConvert – File based video transcoding service which broadcast features which allows you to create media content for broadcast and multi-screen delivery at scale.

MediaLive – It is a media live broadcasting processing service creating high quality video streams to deliver to multiscreen like TV, mobile, machines, set-up boxes.

MediaPackage - This prepares and protects your video for delivery over the internet.

MediaStore – Used to store the media in an optimized way. Gives good performance, consistency, and low latency used to deliver live and on-demand video content.

MediaTailor – Used to target advertising into video streams without sacrificing broadcast quality level of servers.

Machine learning

SageMaker – It makes easy for developers to use deep learning when it comes to coding for their environment.

Comprehend – Sentiment analysis around data, saying good/bad things around your product, etc.

DeepLens – Artificially aware camera, hence camera configures itself what is it looking at. Deep lens is a physical form of hardware that can be bought.

Lex – It powers the Amazon Alexa Service. It is artificially intelligent way of chatting to your customers.

Polly – Intakes text and converts them to speech (with different varieties in language, region, accent). Then stream out to Amazon echo and Alexa reading out those notes for us.

Rekognition – Tells what's object inside the file with their percentage of accuracy.

Amazon Translate – Translate one language to another.

Amazon Transcribe – It is automatic speech to text recognition technology that makes it easy for developers to add speech to text capability to their applications.

Analytics

Athena – It allows to run SQL queries to run in S3 buckets (A fancy name for folders). We've .csv or .xlsx file in S3 bucket and you've to find out the name of all your employees, so we design a SQL queries to look in those objects in your buckets and return results. It is completely serverless.

EMR (Elastic Map Reduce) – It is a tool for big data processing and analysis. It offers expandable low configuration service as an easier alternative to running in-house cluster computing.

Kinesis – It is processing big data in real time. It is possible of processing 100 of terabyte per hour from high volumes of streaming data from sources such as operational logs, financial transactions & social media feeds.

Kinesis video stream – Securely stream video from connected devices to AWS for analytics, ML, and other processing.

QuickSight – It is a business intelligence tool.

Security, Identity & Compliance

IAM (Identity Access Management) – enables to manage access to AWS Services & resources securely. Using IAM, we can create and manage AWS users & groups and use permissions to allow & deny their access to AWS resources.

Cognito– Used to do device authentication. After using mobile apps like LinkedIn, FB, Gmail, etc for authentication, we can use Cognito service to request temporary access to AWS resources.

GuardDuty – Guards from malicious activity.

Inspector – It is installed on VM or on your EC2 instances and can be scheduled monthly or on time basis.

Macie – Scans the S3 buckets and look for things that contain Personal Identifier information (PII) like name, number, passwords, and credit card details.

Certificate manager – It is a service that lets you easy provision, manage and deploy public and private Secure Socket layer/Transport layer Security Certificates.

CloudHSM – Cloud based hardware security module that enables easily generate and use your encryption keys. Offers isolated security module to give an extra level of protection for data with strict corporate, contractual, and regulatory compliance requirements.

WAF (Web application interface) – helps protect our web application from common web exploits that could affect application, compromise security, or consume excessive resources.

Shield – We get it by default for application like CloudFont, Load-balancer, and Route53. It is a managed DDoS (Distributed Denial of Service) protection service that safeguards application running on AWS. Also, there's an advance shield so that we get 24x7 dedicated advance team to prevent DDoS attack.

Artifact – It is a portal that provides enterprise with access to security & compliance reports that apply to the AWS.

Application Integration

Amazon MQ – It is same as RabbitMQ. It used to send messages

SNS – It is a notification service.

SQS – it is a message queuing service. E.g.: We've a Meme page, and someone uploads a video, then if we've EC2 instance that are polling that queue, and once that is done, it will remove it from the queue. If that EC2 instance dies and it does not process that message successfully, then the message appears back in the queue and another EC2 instance goes and process it.

SWF – it is a cloud workflow service that help developers coordinate, track and audit multi-step, multi-machine application jobs.

Customer Engagement

Connect – It is a cloud based contact center solution. Makes up and easy to manage customer contact center and provide reliable customer engagement at any scale.

SES - it is a cloud based email sending service designed to help digital marketers and application developers send marketing, notification, and transaction emails, It is reliable, cost-effective service, of business of all sizes, that use email to keep in contact with their customers.

Business Productivity

Alexa for Business – It is a service that enables organization and employees to use Alexa to get more work done. We can use it as intelligent assistant to be more productive in the meeting rooms, desks, and Alexa devices at homes.

Chime – It is a communication service that lets you meet, chat & place business calls, inside & outside organization.

WorkDocs – It is an online collaboration tool that allows a business to store, share & update files from different devices.

WorkMail – It is a secure managed business email and calendar service with support for existing desktop and mobile email client application. We can set up interoperability with Microsoft Exchange Server and programmatically manage users, group & resources using Amazon WorkMail SDK.

Desktop & App Streaming

Workspaces – it is a managed, secured Desktop-as-a-Service solution that allow its customers to provide cloud based desktops to their end users. Using this,

the end users can access the documents, applications and resources using devices of their own choice such as iPad, Kindle Fire, laptop or Android tablets.

AppStream 2.0 is a fully managed application streaming service. Using this, we can securely manage desktop application and deliver them to any computer. We can easily scale to any number of users across the globe without acquiring, provisioning, and operating hardware or infrastructure. The experience is fluid & responsive including GPU-intensive 3D design and engineering.

Mobile Services

MobileHub – it is a collection of AWS tools designed to help developers to build, test, configure, and release cloud based application for mobile devices.

Pinpoint – It is a way of using targeted push notification to drive mobile engagement.

AWS AppSync – It updates the data in web and mobile application in real time and update data for offline users as they reconnect.

DeviceFarm – It is a way of testing application on real live devices, it could be android, iPhone devices, etc.

IoT

It is a platform that enables us to connect devices to AWS services & other devices, secure data & interaction, processes & act upon device data, and enable application to interact with devices even when they are offline.

IoT device management – Helps you onboard, organize, monitor & remotely manage IoT devices at scale. It integrates with AWS IoT Core to easily connect devices to the cloud & other devices so that you remotely manage your fleet.

Amazon FreeRTOS – It is an open source operating system that makes small, low power edge devices to program, deploy, secure, connect & manage

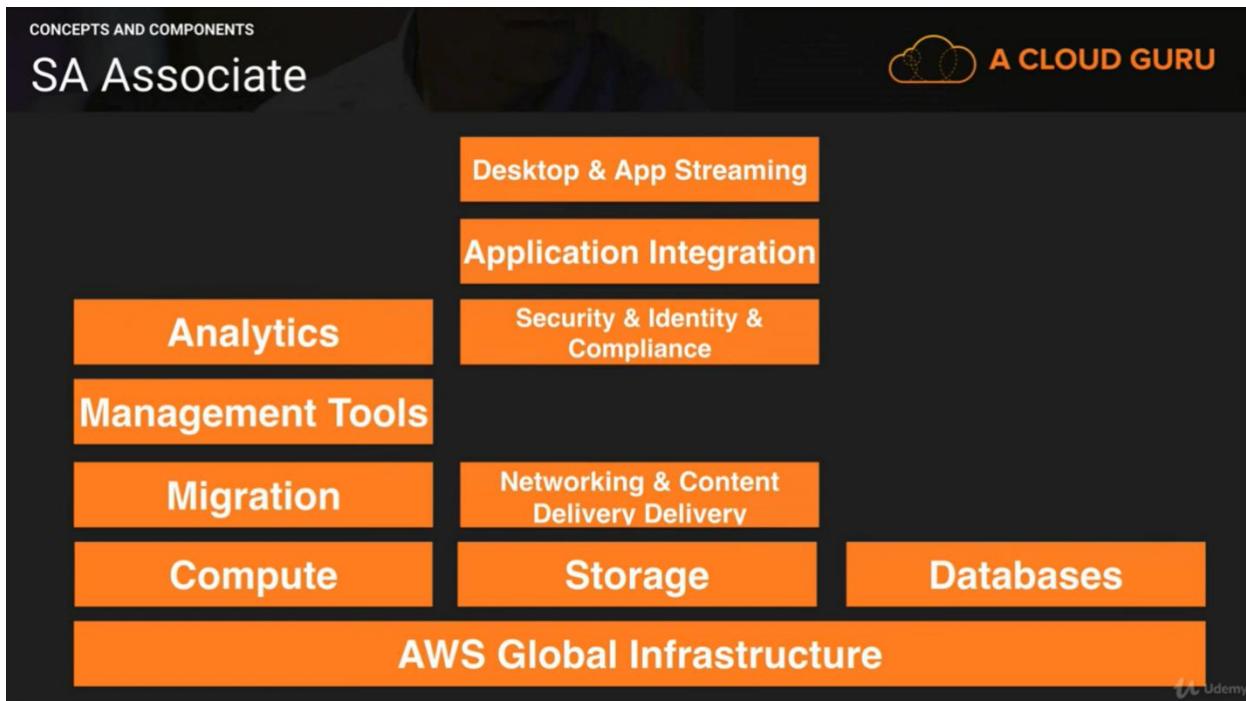
Greengrass – It seamlessly AWS to edge devices so they can act locally on the data they generate, while still using the cloud for management, analytics & cloud for storage. With AWS IoT Greengrass, connected devices can run AWS Lambda functions, execute prediction based on Machine learning models, keep device data in sync, and communicate with other devices securely – even when it is not connected to Internet.

Game development

GameLift – It is a fully managed service for deploying, operating & scaling your session-based multiplayers game servers in the cloud.

2. Important Service for

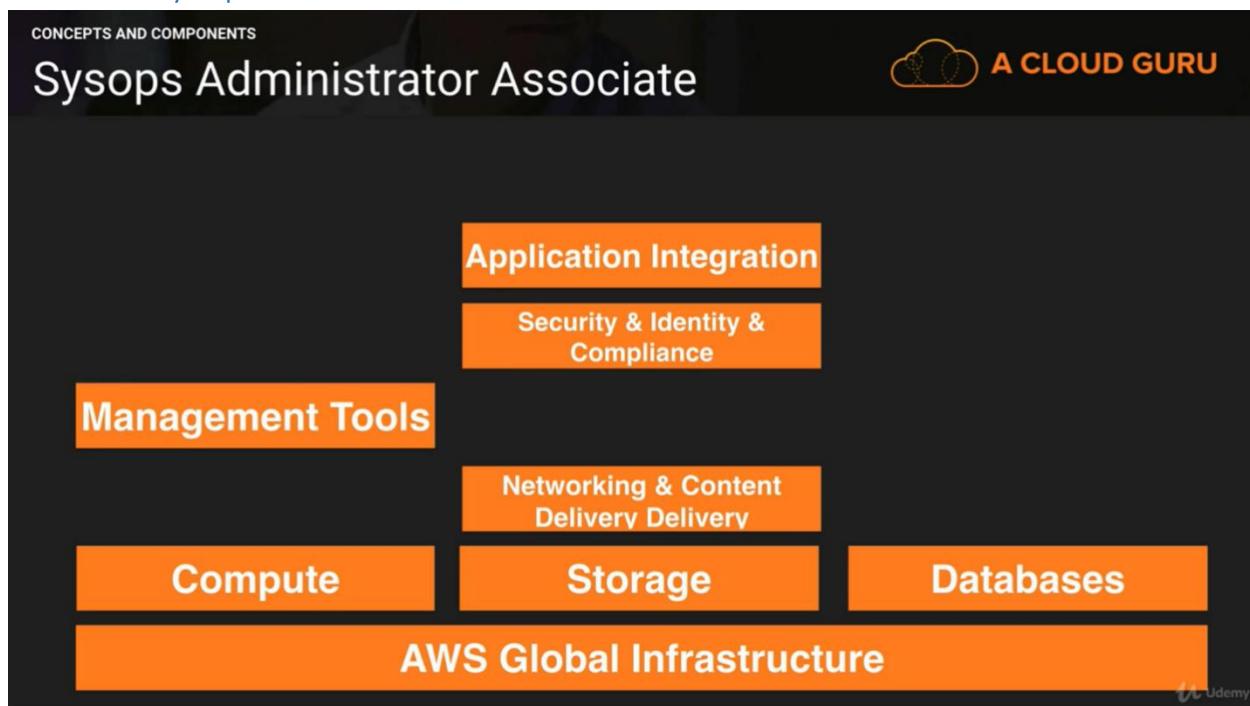
1. AWS SA Associate exam



2. Developer Associate exam



3. Sysops Administrative Associate



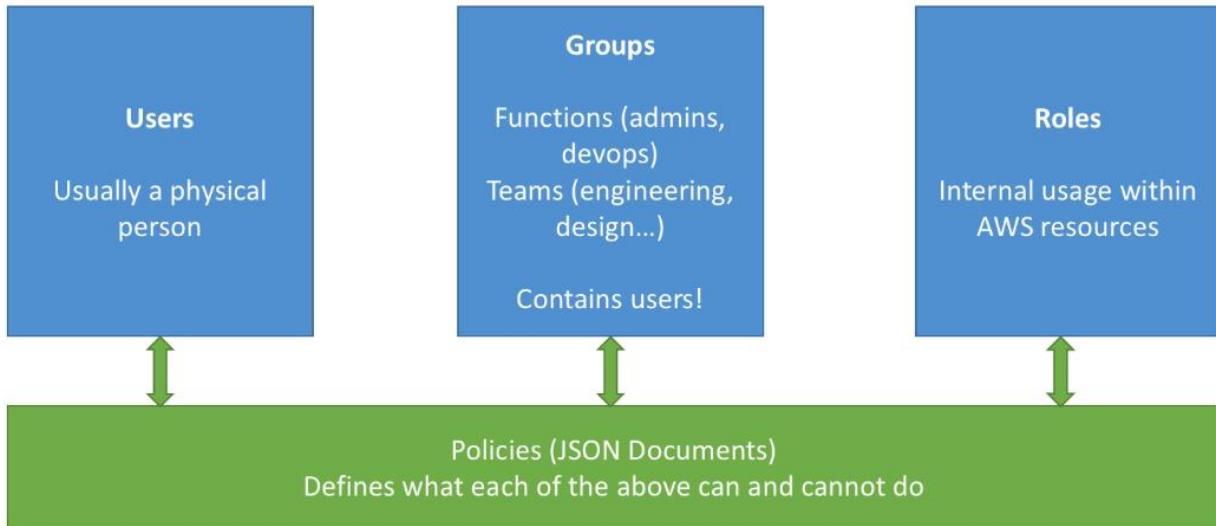
3. Identity access management

It's all about setting up the users and granting rights to those users to the AWS console. It comes under security, Identity & Compliance. Root account is the account which we create our AWS account.

IAM provides:

1. Centralized control of AWS account.
2. Shared access to your AWS account.
3. Granular permissions. (granting different permission to different people for different resources)
4. Identity federation. (Including active directory, Facebook, LinkedIn, etc.)
5. Multifactor Authentication. (2-factor authentication while logging in)
6. Provide temporary access to users/devices and services where necessary.
(If we want to connect our mobile application, web application, so these application can temporarily access our AWS account and store things in dynamo DB or S3.)
7. Allows to set up & manage password rotation policy.
8. Integrates with many different AWS services.
9. Supports PCI DSS Compliance. (Payment Card Industry Data Security Standards) – it is a set of security standards designed to ensure all the companies that store, process, access or transmit debit card information maintain a secure environment.)
10. IAM is global and does not apply to any particular region at this time.
11. New users have no permissions when they are created.
12. New users are assigned Access key ID & Secret access key when newly created and they can be disabled or regenerated.
13. IAM has predefined "managed policies", so we don't need to re-write the whole policies, we can reuse that Amazon puts together for us.

Terms



1. **Users:** End users. Usually a physical person.
2. **Groups:** Collection of users under one set of permission. Like we've a functioning team in an organization like admins, devops, design team, engineering team, etc.)
3. **Roles:** We can create roles and assign them to assign resources. (We've EC2 instance, now we give it a role to access S3 i.e., EC2 instance writing files directly to S3, so we don't need to create password to give access to S3.
4. **Policies:** Permissions that we apply to users, groups & roles. IAM has predefined “managed policies”. It sits on the top of Users, Groups & Roles. It is written in JSON.

Note: It is always a good practice to give users the minimal amount of permissions they need to perform their job (least privilege principles).

Lab Setup

1. Setup the region that's closest to you.
2. When we select IAM, we can see that region is global, since IAM doesn't have a region.
3. Can change the IAM user's sign in link.

Dashboard

1. **Delete your root access key:** Root account: It is the email id that is used to login to the AWS account. It gives a root level access. We should never use

them. In case someone finds out our login id & password, still they will not be able to login without the physical device.

2. **Activate MFA on your account:** We can add multi-factor authentication (**MFA**) on our account
3. Download **Google authenticator** from Google Play Store, scan the QR code and Finish.
4. **Create individual IAM users:** We can add multiple users and give the access. There's 2 type of access:
 - a. Programmatic access: where application the AWS using command line tool installed on your desktop and if we wanna store the files in S3.
 - b. Management console: using the UI
5. **Use groups to assign permissions:** Define their **groups**, like system admin, IAMUserChangePassword, etc. and finish.
6. We can even give access of a permission to a specific user by selecting a user, attaching existing policies directly, and attaching the policy name and finishing it up.
7. We can also give permission by pushing the user to a specific group like Admin, Developers, etc, and adding permission to that group since giving permissions to each user is not a manageable way to assign permission. Once we apply permissions to the group, the users inside it will inherit the permissions of the group. hence we can delete the individual permissions attached to the users.
8. We can make users active/inactive by navigating onto users and its security credentials page. A new access key ID can also be generated for the same user from the same page.
9. **Apply IAM password policy:** We can set password policy like including one uppercase letter, password expires in 30 days, etc.
10. We can sign in as an IAM user using the link available in Dashboard of IAM.

Note: To allow same IAM user in other account, follow the tutorial:
https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial_cross-account-with-roles.html

Create Roles

1. Go to IAM

2. Select Roles → Create Role
3. Select the type of AWS service (e.g.: EC2)
4. Choose the type of policy you wanna assign to the role.
5. Give a role name & description to it.
6. Click on Create Role button, now you can see your newly created role.

Note: Whenever we create a role, it gets created globally.

IAM Federation

- Big enterprise usually integrate their own repository of users such as Active Directory, they can integrate with IAM.
- This way, users can login into their AWS using their company credentials.
- It uses the SAML standard.

Points to remember

1. There should be single IAM users per physical person.
2. There should be one Role per application.
3. IAM credentials shouldn't be shared.
4. Never use root account except for initial setup.
5. It is best to give the users the minimal amount of permission, they need to perform their job.
6. Never write IAM credentials in Code.
7. Never Commit your IAM credentials.
8. Never use ROOT IAM Credentials
9. We can customize the IAM user sign-in link from the IAM user homepage.
Click on customize and we can create a Alias and hit on Create.

Create Policy

1. Go to Ec2 under Services
2. Navigate to Policy
3. Hit on “Create Policy”.
4. Choose a Service, e.g.: S3
5. Select the actions, also we can expand the each action tab.
6. Choose the resources, under specific resources, we can give our ARN.

7. Hit “Review Policy” & create.

Note: Policies can also be generated from website “AWS Policy Generator”. Policy can be tested using AWS policy simulator (search on the web).

4. Setting up billing alert

1. Go to My billing dashboard.
2. Click on “Billing preferences”.
3. Select “**Receive Free Tier Usage Alerts**” and enter your email id.
4. Select “**Receive Billing Alerts**” and then select manage billing alerts.
5. Click on billing → Create alarm → Select a metrics → Set the currency & threshold value → Create an SNS topic → Confirm the Subscription on the entered email account → Set the alarm name → Create the alarm.

Note: Insufficient data available shows since the data limit has not exceeded. Though you’re in a particular region, billing alarm represent your charges for all regions.

Setting up Budget setup

1. Login from your root account.
2. Go to My billing dashboard.
3. Navigate to Budget section.
4. Click on Create a Budget and then select Cost Budget.
5. Click on Set your budget.
6. Give your budget a name.
7. Select the Period to monthly & choose it to be a recurring one.
8. Select Budget amount to be fixed for now & Budgeted amount say to be \$0.01
9. Click on Configure Alerts.
10. We can choose send email Alerts based on the Actual cost.
11. We can set threshold like when we reach 80% of the Budgeted amount.
12. Set up the email which will receive these alerts.
13. Click on confirm Budget.
14. Review and hit on Create.

5. S3 – Simple Storage Servers

It provide IT teams & developers with **highly scalable** (up to petabytes & Exabyte) object storage. Used to store & retrieve any amount of data from anywhere. It is object based storage (other one is block based).

It consists of

1. Simple key where **key** is the full path of the object which are stored in alphabetical orders. E.g.:
 - a. S3://my-bucket /myfile.txt
 - b. S3://my-bucket/myFolder1/anotherFolder/myfile.txt
2. The key is composed of **prefix** + **object name**
 - a. S3://my-bucket/ myFolder1/anotherFolder/myfile.txt
3. There's no concept of directories.
4. Value is simply the data which is made up of sequence of bytes.
5. Version ID which is important for versioning
6. Metadata is the data about the data you're storing respectively.
7. We can have up to 10 tags
8. Subresources:
 - a. Access control list: It defines who can access these objects. We can implement on particular object, bucket or a file.
 - b. Torrent: It supports the Bit Torrent protocol.

The naming convention of bucket is:

1. No uppercase
2. No underscore
3. 3-63 characters long
4. Not an IP
5. Must start with lowercase letter or number.

Object based

Videos, Photos, PDF, documents, etc. data is stored in the form of an object which consists of data file, rich metadata fields & unique key. Object can be retrieved using the metadata or knowing the key. Good option for storing large set of unstructured data.

1. They are **durable** i.e., if particular disk fails we don't have to worry about losing the data.

2. They are highly **available** i.e., they are accessible through REST API's & if single server goes down, still we can access, read, write files.
3. They are **secured** i.e., they allow to store & transfer files in encrypted format.

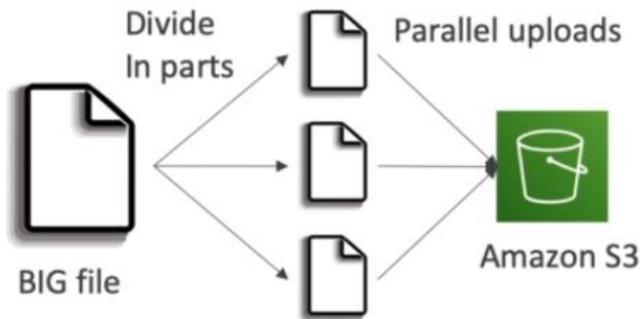
It has use cases in:

1. Disaster recovery/Backup
2. AI & Analytics
3. Cloud native (Since it's cost effective)
4. Data archive.

S3 basics

1. Files can be from 0 to 5 TB.
2. Unlimited storage.
3. Files are stored in Buckets & there can be multiple buckets in S3 environments.
4. S3 is global i.e., S3 bucket does not need a region selection but when we create bucket, it is created in a particular region
5. S3 bucket name must be unique globally.
6. When we create a bucket, we basically create a DNS address. It consist of [https://s3-\(region name\)-amazonaws.com/\(bucketname\)](https://s3-(region name)-amazonaws.com/(bucketname))
7. When we upload a file to S3, we receive a HTTP 200 success code if upload is successful.
8. When we write (PUTS) & access new data, there is good consistency.
9. When we update (overwrite PUTS) & DELETES (it takes some time to propagate. Also, in these operation we don't get partial or corrupted data. We get older or new version of data. It is k/a eventually consistent.
10. It gives us 99.99% availability for S3 platform & 99.9% availability.
11. It guarantees 99.99999999% (11x9's) durability for S3 information.
12. Not meant for storing OS or databases.
13. Tiered storage option available. Tiered storage is a storage networking method where the data is stored on various types of media based on performance, availability & recovery requirements.
14. Lifecycle management.
15. Encryption can be done on S3.
16. Data can also be secured using Access Control Lists & Bucket policies.

17. It is designed to sustain the loss of 2 facilities concurrently.
18. S3 – **IA** (Infrequently accessed) is for the data that is accessed less frequently but requires rapid access when required. It has lower fee than S3 but you're charged a retrieval fee.
19. It can also sustain 2 concurrent facility failures.
20. We can upload files much faster to S3 by



- a. Enabling multipart upload (recommended for files greater than 100 MB and highly recommended for files > 5GB).
- b. Parallelizing PUTs for greater throughput.
- c. Maximize network bandwidth.
- d. Decrease time to retry in case a part fails.
21. **S3 One Zone - IA** - It is same as IA, but here the data is stored in single AZ.
 - a. It has high durability (99.99999999%) of objects in a single AZ but the data is lost when AZ is destroyed.
 - b. 99.5% availability.
 - c. Low latency & high throughput performance.
 - d. Supports SSL for data at transit and encryption at rest.
 - e. Low cost compared to IA (by 20%).
 - f. Use cases - storing backup copies of on-premise data, or storing data that we can recreate.
22. **S3 Intelligent Tiering** - Same low latency and high throughput performance of S3 Standard.
 - a. Small monthly monitoring and auto-tiering fee.
 - b. Automatically moves objects between 2 access tiers based on changing patterns. So it moves objects between S3 general purpose, S3 IA, and hence it chooses for us if our object is less frequently accessed or not.

- c. We are going to pay a fee to do that little monitoring.
 - d. Designed for durability of 99.999999999% of objects across multiple AZ.
 - e. Resilient against events that impacts an entire AZ.
 - f. Desired for 99.9% availability over a given year.
- 23. Reduced redundancy storage** – Designed to provide 99.99% durability & 99.99% of availability of objects over a given year. For e.g., we can store files in one bucket & thumbnails in another. So even if the file goes missing, we can generate the file using thumbnail.

	Standard	Standard - Infrequent Access	Reduced Redundancy Storage
Durability	99.99999999%	99.99999999%	99.99%
Availability	99.99%	99.9%	99.99%
Concurrent facility fault tolerance	2	2	1
SSL support	Yes	Yes	Yes
First byte latency	Milliseconds	Milliseconds	Milliseconds
Lifecycle Management Policies	Yes	Yes	Yes

- 24. Glacier** – It is very cheap but used for data archival only & it takes 3-5 hours to restore from glaciers.

- a. It stores data for as little as 0.004\$ per gigabyte per month + retrieval cost.
- b. Low cost object storage meant for archiving / backup.
- c. Data is retained for the longer term (10s of year).
- d. Alternative to on-premise magnetic tape storage.
- e. Average annual durability is 99.99999999%
- f. Each item in Glacier is called "Archive" (up to 40TB).
- g. Archives are stored in "Vaults".
- h. There are 3 retrieval options:
 - i. Expedited (1 to 5 minutes).
 - ii. Standard (3 to 5 hours).
 - iii. Bulk (5 to 12 hours).

iv. Minimum storage duration of 90 days.

25. Glacier Deep Archive - for long term storage - cheaper.

a. There are 3 retrieval options:

i. Standard (12 hours).

ii. Bulk (48 hours).

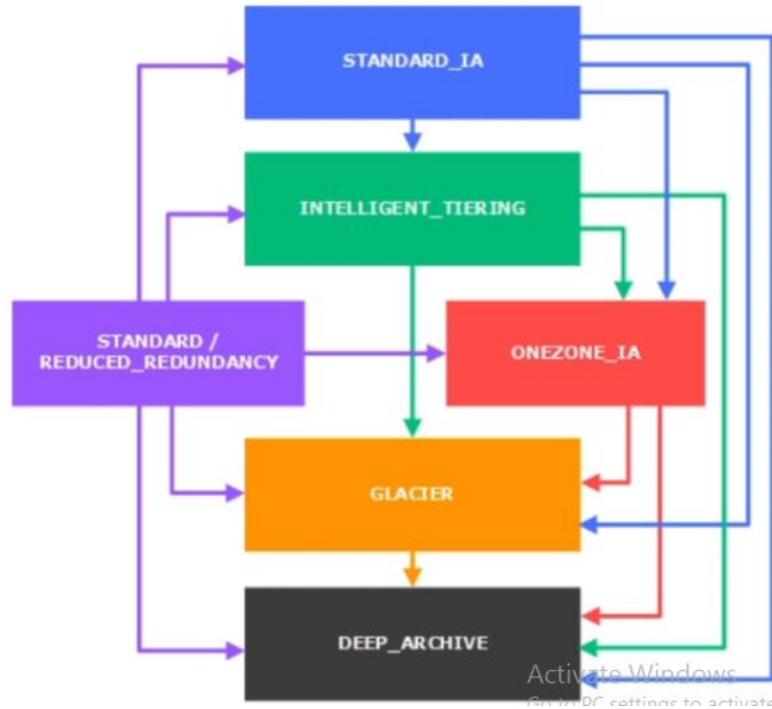
iii. Minimum storage duration of 180 days.

	S3 Standard	S3 Intelligent-Tiering	S3 Standard-IA	S3 One Zone-IA	S3 Glacier	S3 Glacier Deep Archive
Designed for durability	99.999999999% (11 9's)					
Designed for availability	99.99%	99.9%	99.9%	99.5%	99.99%	99.99%
Availability SLA	99.9%	99%	99%	99%	99.9%	99.9%
Availability Zones	≥3	≥3	≥3	1	≥3	≥3
Minimum capacity charge per object	N/A	N/A	128KB	128KB	40KB	40KB
Minimum storage duration charge	N/A	30 days	30 days	30 days	90 days	180 days
Retrieval fee	N/A	N/A	per GB retrieved	per GB retrieved	per GB retrieved	per GB retrieved

	S3 Standard	S3 Intelligent-Tiering	S3 Standard-IA	S3 One Zone-IA	S3 Glacier	S3 Glacier Deep Archive
Storage Cost (per GB per month)	\$0.023	\$0.0125 - \$0.023	\$0.0125	\$0.01	\$0.004 Minimum 90 days	\$0.00099 Minimum 180 days
Retrieval Cost (per 1000 requests)	GET \$0.0004	GET \$0.0004	GET \$0.001	GET \$0.001	GET \$0.0004 + Expedited - \$10.00 Standard - \$0.05 Bulk - \$0.025	GET \$0.0004 + Standard - \$0.10 Bulk - \$0.025
Time to retrieve	instantaneous	Instantaneous	Instantaneous	Instantaneous	Expedited (1 to 5 minutes) Standard (3 to 5 hours) Bulk (5 to 12 hours)	Standard (12 hours) Bulk (48 hours)
Monitoring Cost (per 1000 objects)		\$0.0025				

Activate Windows

Moving between storage classes

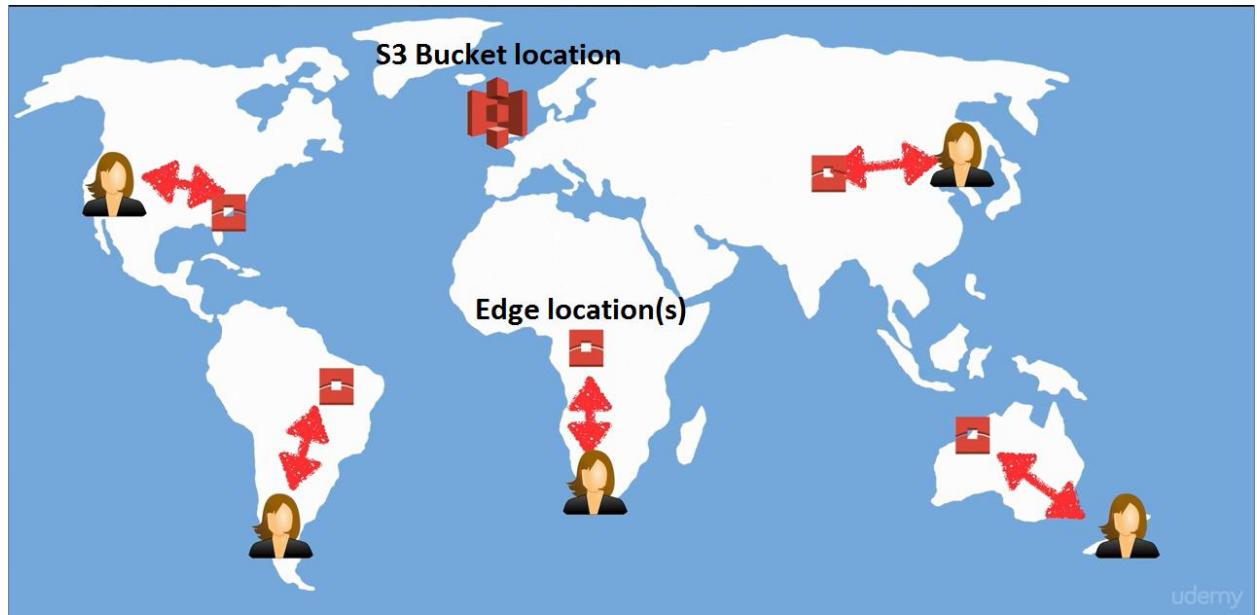


- We can transition objects between storage classes.
- Moving objects can be automated using a lifecycle configuration.

S3 – Charges

S3 charge us for:

1. Storage
2. Retrieval
3. Storage management pricing (Amazon S3 inventory, analytics, and object tagging. Tags can be associated with it that this data is associated with this & let you track your cost on per tags basis.)
4. Data transfer pricing - Data moving in S3 like from one region to another costs.
5. Transfer acceleration – Enables easy, secure & fast transfer of files over a long network between users & S3 buckets. It takes advantage of Amazon CloudFront's globally distributed edge location. As the data arrives on edge location, data is routed to S3 over an optimized network path.



Note: Take S3 FAQ's before giving exams!

Creating S3 Bucket

1. Go to Services and Select S3.
2. Click on Create bucket.
3. Type-in the bucket name (unique one as it contains DNS) & select your region.
4. We can choose tags, versioning (keep multiple version of an object in the same bucket) & logging (it sets up log records that provide details about the access request).
5. We can manage users & even add another AWS to it.
6. After giving it access & permissions, once review all the data and click on create bucket!
7. Now, we can upload files on this bucket, after clicking on the bucket name.

Uploading Files on S3

1. Click on the bucket name.
2. Select Upload and choose the file.
3. We can manage users, set permissions, define the storage class or directly upload with defaults.

4. When upload is finished, click on the checkbox, it shows the property of the file. We also get a DNS link to the object.
5. When we click on the link, it browse and says that access denied! It says so, coz by default the buckets are private. But if we browse by doing right click on it & select open, then we can view the object. This long URL is called pre-signed URL.
6. By default no public access is granted to the buckets, to make it public firstly.

Then, select the file name & click on “actions” & “make public”. After browsing through file’s DNS, we can view the files on browser. We can give list, read, write to a file and even select the set of rules for granting permission for buckets.

7. We can edit other properties at file level like “Storage class (Standard, IA, Reduced redundancy, Intelligent tiering, Glacier deep archive, One zone-IA), Encryption (AES-256, AWS-KMS or none), Metadata, Tags, Object lock” by clicking on the file.

Note: Though we create Tag for the bucket, but the file inside the bucket don’t inherit the tags. Also, Object lock block object deletion during customer-defined retention period so that we can enforce retention policies as an added layer of data protection on for regulatory compliance.

For more details visit: <https://aws.amazon.com/about-aws/whats-new/2018/11/s3-object-lock/>

8. The minimum size of the file can be 0 bytes.
9. The properties (versioning, server access logging, static website hosting, object level logging, default encryption) can also be set at bucket level.
10. Some other settings like Object lock, Tags, Transfer acceleration, Events and Requester pays can also be done at bucket levels.
11. We can generate policy for S3 buckets by clicking on “Bucket policy” & then click on “Policy generator” or write policies in JSON format. Select the type of policy, principal is used to select the type of users (* for all users) or particular IAM user. Select the actions (the permissions for the policy & the ARN name and then generate a policy.
12. Encryption can be done on:
 - a. Client side
 - b. Server side. It consist of 3 types:

- i. With Amazon S3 managed keys (SSE-S3).
- ii. With KMS (SSE-KMS)
- iii. With customer provided keys (SSE-C)

Versioning

- Once we enable versioning, we can't disable it, only we can suspend and versioning is enabled at a bucket level.
- It enables us to keep multiples copies of files in S3 bucket. To enable versioning, click on the bucket name & go to properties. Select versioning and enable!
- It is a good practice to enable versioning as
 - It protects against unintended delete and
 - It helps in rolling back to previous version.
- Any file that is not versioned prior to enabling versioning will have version "null".
- If we suspend versioning, it does not delete previous version, it just make sure that the future files do not have version assigned to it.
- If we delete the file, we get a "Delete marker" on the file. Delete marker is a file with zero size and this Delete makes the file hide.
- If we delete the file with the "Delete marker", the latest version is one of the previous version and if we navigate back to the bucket, we've our file back.
- Deleting the file from versioning, deletes the file permanently.

LAB

1. Create a file, add some content to it and upload.
2. Edit the content of the file, save & upload (with same name).
3. After clicking on the checkbox, on the right hand side we can see the versions.
4. Click on the versions & you will be able to see the different versions of the file.
5. We can also delete the version which is not needed.

6. If we want to see the original copy, click on “Show/Hide” tag. Hide button contains the history of the file. Deleting the file from here, permanently deletes the file.
7. **MFA delete:** Using this, we can add another layer of security by configuring a bucket to enable MFA (Multi-factor authentication) which require additional authentication for changing the versioning state of your bucket & permanently deleting an object version. We can enable MFS delete only when versioning is enabled. We'll need MFA to:
 - a. Permanently delete an object version.
 - b. Suspend versioning on the bucket.

We don't need MFA for:

- a. Enabling versioning
- b. Listing deleted version.

Only the bucket owner can enable/disable MFA delete. MFA delete can only be enabled using the CLI. MFA delete requires 2 forms of authentication together:

- i. Your security credentials.
- ii. The concatenation of a valid serial number, a space, & 6-digit code displayed on approved authentication device.

Enable MFA Delete

Once we enable MFA delete, we cannot delete any object from console even if we delete the delete marker (Versioning). Also, versioning cannot be disabled once we've enabled MFA delete. To disable the above settings, we need to disable MFA (by running the same command which was used for enabling, but this time we'll set MFADelete=Disabled).

1. Run `pip3 install awscli`
2. Upgrade awscli to the latest version `pip3 install --user --upgrade awscli`
3. Check the path where AWSCLI is installed
(Typical path include: C:\Program Files\Python37\Scripts\)
4. Add this path to the environment variables.
5. Restart command prompt.
6. Configure aws: `aws configure`
7. Type in the access key ID & Secret key ID of the root account.

The screenshot shows the AWS Identity and Access Management (IAM) dashboard. On the left, a sidebar lists options like Dashboard, Groups, Users, Roles, Policies, Identity providers, Account settings, and Credential report. A search bar for 'Search IAM' is also present. The main area is titled 'Welcome to Identity and Access Management'. It displays 'IAM users sign-in link: https://technoronicks.signin.aws.amazon.com/console' and 'Customize' options. Below this, 'IAM Resources' are listed with 'Users: 2', 'Groups: 2', 'Roles: 3', and 'Identity Providers: 0'. A 'Customer Managed Policies' section shows 1 policy. The 'Security Status' section has a progress bar indicating '4 out of 5 complete.' It includes a warning about deleting root access keys and a link to 'Manage Security Credentials'. Another section shows a checked checkbox for 'Activate MFA on your root account'.

The screenshot shows the 'Your Security Credentials' page under the IAM service. The left sidebar is identical to the previous screen. The main content area is titled 'Your Security Credentials' and provides instructions for managing credentials. It lists sections for Password, Multi-factor authentication (MFA), and Access keys (access key ID and secret access key). A note states: 'Use access keys to make programmatic calls to AWS from the AWS CLI, Tools for PowerShell, the AWS SDKs, or direct AWS API calls. You can have a maximum of five active or inactive at a time.' A 'Create New Access Key' button is located at the bottom of the access key section. A table below shows four existing access keys:

Created	Deleted	Access Key ID	Last Used	Last Used Region	Last Used Service	Status
Nov 30th 2019		AKIAJ1OOQWAZP5PUGXAQ	2019-11-30 22:06 UTC+0530	ap-south-1	s3	Active
Nov 30th 2019	Nov 30th 2019	AKIAJDILR6TP57F1QKEA	N/A	N/A	N/A	Deleted
Nov 30th 2019	Nov 30th 2019	AKIAJ0ZON6MACFD5UIIA	N/A	N/A	N/A	Deleted
Nov 30th 2019	Nov 30th 2019	AKIAJP6FMLIKHURGFD7A	N/A	N/A	N/A	Deleted

8. Type in the region name of the root account, it's better to download the .csv file (all details of the users are mentioned).
9. If MFA is assigned to the device, you'll need it at the time of executing the command.
10. To enable MFA delete: **aws s3api put-bucket-versioning --bucket bucket_name --versioning-configuration MFADelete=Enabled,Status=Enabled --mfa**

```
"arn:aws:iam::895094500025:mfa/root-account-mfa-device  
mfa_6_digit_code"
```

E.g.: `aws s3api put-bucket-versioning --bucket raunak1901 --versioning-configuration MFADelete=Enabled,Status=Enabled --mfa "arn:aws:iam::895094500025:mfa/root-account-mfa-device 074489"`

11. To check the aws configuration: `aws configure list`

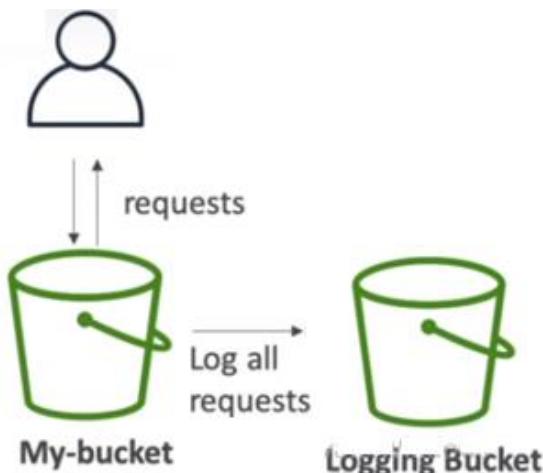
12. To check whether versioning is enabled: `aws s3api get-bucket-versioning --bucket raunak1901`

13. Try to delete the selected S3 object version with MFA authentication, but with other user: `aws s3api delete-object --bucket raunak1901 --key iron_man.png --version-id ZAI0RaXporU5Yfu_trOSCX_bQoc4kGNR -mfa "arn:aws:iam::895094500025:user/Raunak 123123"` Using other user will return **NotDeviceOwnerError**.

14. To show the list of bucket: `aws s3 ls`

S3 Access Logs

- For audit purpose, we may want to log all access to S3 buckets.
- Any request made to S3, from any account, authorized or denied, will be logged into another S3 bucket.
- That data can be analyzed using data analysis tool or Amazon Athena.



We make request to a bucket and that bucket has been enabled for logging into another logging bucket. All the request will get logged into this bucket.

LAB

1. Create 2 buckets with default settings.
2. Turn on Server Access Logging for the monitoring bucket under bucket properties.
3. Choose a target bucket in which the logs are going to be stored.
4. Hit Save.
5. Do a couple of operations and after few moments, we can see that S3 has all the logs.

S3 Replication (Cross-Region replication and Same-Region Replication)

- It enables asynchronous copying of objects across buckets in different AWS region.
- Buckets configured for cross-region replication can be owned by same AWS account or by different accounts.
- Versioning should be turned on for both buckets.
- It is enabled at a bucket level configuration.
- We need to specify proper IAM Permissions to S3.
- CRR use cases - compliance, lower latency access, replication across accounts.
- SRR use cases - log aggregation, live replication between production and test accounts.
- After activating, only new objects are replicated (existing objects are not copied).
- For Delete operations:
 - If we delete without version ID, it adds a delete marker and it is not going to be replicated.
 - If we delete with a version ID, it deletes in the source and not replicated.
- If bucket I has replication into bucket II, which has replication to bucket III then, the objects created in bucket I are not replicated to bucket III.

LAB

To enable cross-region replication:

1. Create a new bucket
2. Turn on versioning for this bucket.
3. Select the bucket for whom a copy is to be made.
4. Go to management
5. Select Replication
6. Select replication for Entire bucket or for some particular Prefix or Tags.
7. Select the destination bucket.
8. We can change the storage class for this bucket, for e.g. if we need to create this bucket as a backup and we are not going to use it frequently, select storage as S3-IA.
9. Create a rule for this bucket if you don't have one.
10. Save it.
11. Till now, we have created a bucket and allowed replication of the bucket. But if we check the bucket, the contents are still empty.
12. Copying the contents be like: `aws s3 cp --recursive s3://raunak1901 s3://technoronicssydneybucket`

Note: ‘recursive’ copies everything from source bucket to destination bucket.

13. The permissions are not copied when we replicate the bucket. Like, if the file is public in ‘raunak1901’ bucket, the default access level i.e. private is assigned to the bucket.
14. When we delete or rename an object in the source bucket, the original file remains in the destination bucket though the file gets removed or renamed respectively.

S3 Pre-Signed URLs

- Can generate Pre-signed URLs using SDK or CLI.
- They've default expiration time of 3600 seconds (1 hour) and can be configured with `--expires-in [TIME_BY_SECONDS]` argument.

- Users given a pre-signed URL inherit the permission of the person who generated the URL for GET/PUT.
- Example:
 - Allowing logged-in users to download premium video on S3 bucket.
 - Allow an ever changing list of users to download files by generating URLs dynamically.
 - Allow temporarily a user to upload a file to a precise location in our bucket.

LAB

1. Go to CLI.
2. Before generating pre-signed URLs, we need to configure CLI to generate a signature version. Type "aws configure set default.s3.signature_version s3v4". It allows us to have generated URL to be compatible with KMS encrypted object.
3. Run the command "aws s3 presign s3://bucket-name/file-name.ext --expires-in 200 --region eu-west-1". This gives us the pre-signed URL.

Lifecycle Management, IA, S3 & Glacier lab

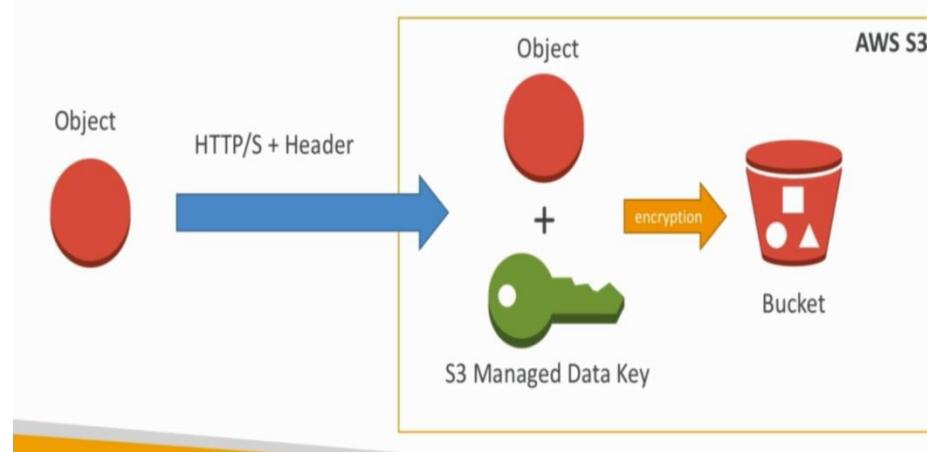
They are used to manage your objects so that they are stored cost effectively, configure the lifecycle. Like we want the objects that are not used frequently to be moved to S3-IA or Glacier after a year of creation. Or set an expiration date and S3 deleting objects when they get expired.

1. Go to S3 & create a bucket.
2. Versioning can also be used with Lifecycle management (not necessary).
3. Go to management -> Add lifecycle rule.
4. Define the lifecycle rule name. We can use add tags/prefixes so that the rule is applied to set of files.
5. If versioning is turned ON, we get an option to set the rule for current version and/or previous versions.
6. We will now add transitions rule for the object like move to IA or glacier after n number of days.

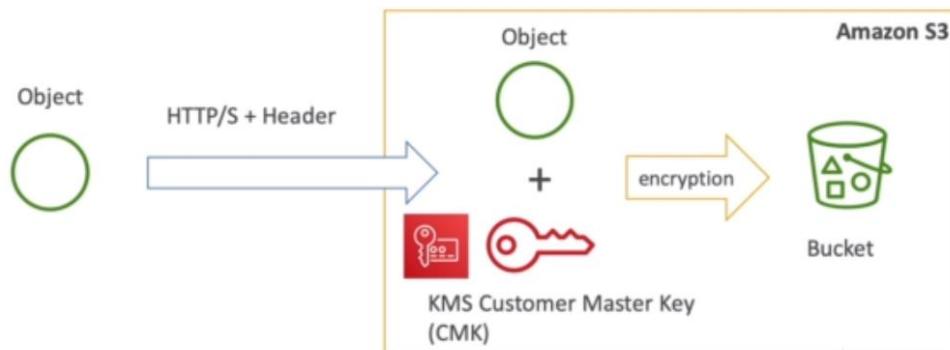
7. We can set expiration period (Permanently delete) for both previous & current version and we can select “clean up expired object delete markers & incomplete multipart upload”.
8. Object must be 128Kb in size.
9. We can move objects to S3-IA after minimum period of 30 days & archive to Glacier after 60 days.

Encryption

1. By default, all newly created buckets are private.
2. We can setup access control to buckets using:
 - a. Bucket policies
 - b. Access control lists.
3. S3 can be configured to create access logs which log all request made to S3 bucket. This can be don't to another bucket.
4. The encryption can be set up while uploading a file to AWS S3 bucket in the “Set permissions” page. The 2 options available here are SSE-S3 & SSE KMS. SSE-C is not available as we need to provide our own data key & Client side encryption is not provided since we need to encrypt the data on our machine. If we need to set encryption to bucket, go to bucket's properties and select “Default encryption” & specify the encryption type.
5. There are 2 types of encryption:
 - a. In Transit: AWS S3 exposes HTTP endpoint for non-encrypted data & HTTPS endpoint for encrypted data. It is simply when we are transmitting information to & fro from the bucket. It is preferred to secure using SSL/TLS encryption. Encryption in flight is also known as SSL/TLS.
 - b. At REST:
 - i. Server side Encryption
 1. **S3 managed keys – SSE-S3:** Each object is encrypted with a unique key. It uses 256-bit Advanced Encryption Standard (AES-256). It is handled & managed by AWS. Object is encrypted server side. To make it work, we must set a header when we send the data to Amazon S3. The header is: “**x-amz-server-side-encryption**”:“**AES256**”.



E.g.: We have object and we want to put it in Amazon S3 bucket & we want to encrypt with SSE-S3. So, we are going to make an HTTP request & and the SSE-S3 header to it. So, we can see our object now, & due to encryption it is going to create a managed data key which is managed by S3. Using the object & the key, it will be encrypted & then it'll be put to Amazon S3 bucket

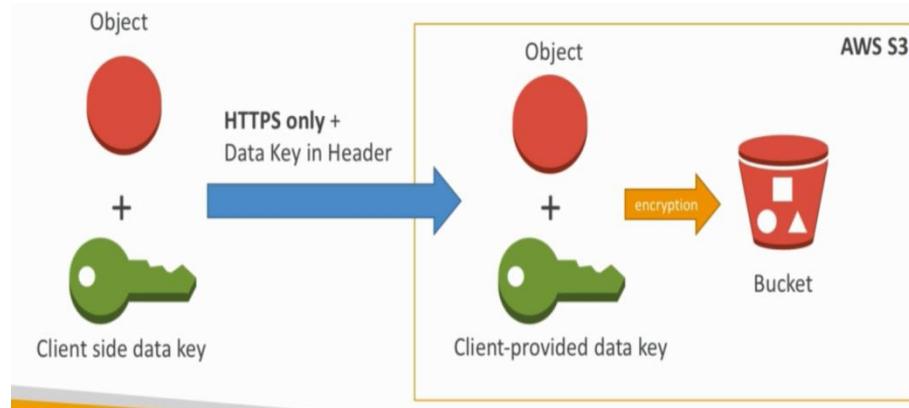


2. AWS key management service, Managed keys – SSE-KMS: The encryption using keys is handled & by managed by KMS. It provide with envelop encryption i.e., it provide us with data encryption key & this provide added protection to the unauthorized access to the objects in S3. It also provide us more control over the keys and with audit trail (record that when our keys were used & who used those keys). It gives the transparency that who is decrypting what & when). The object is encrypted at the server side.

We also have an option to create & manage encryption keys. The difference between AES256 & KMS is that in KMS, keys are managed on customer side. We must set the header: “`x-amz-server-side-encryption`”:“aws:kms”.

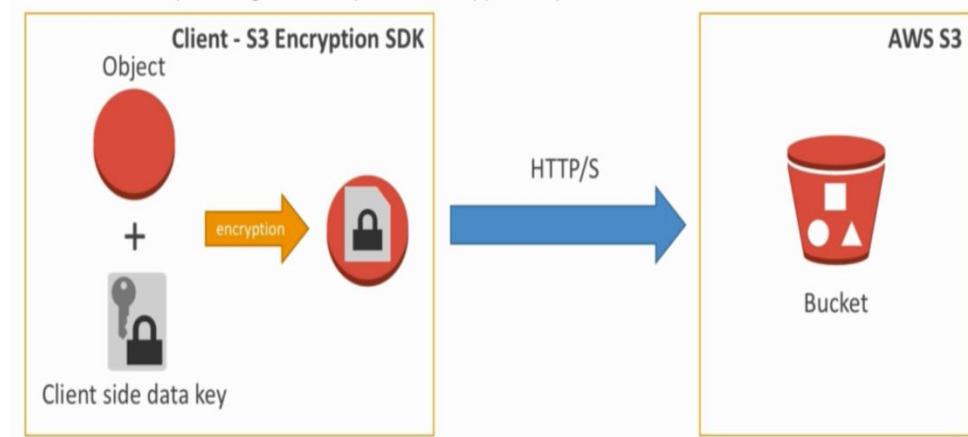
3. Server side encryption with customer provided keys – SSE-C.

The data keys are managed by the customer outside AWS & Amazon does not store the keys that we provide. Also, in this case HTTPS must be used. Encryption keys must be provided in the HTTP headers, for every HTTP request made.



The customer generated key is passed to AWS S3 & after encryption, it is thrown by AWS.

ii. Client Side Encryption:



We need to use library such as Amazon S3 Encryption client & client must encrypt data themselves before sending to S3. Also, they need to decrypt the data themselves when retrieving from S3. Customer fully manages the keys & encryption cycle.

LAB - I

1. Go to S3 under Services.
2. Upload a file.
3. Under Set properties tab, select encryption to be Amazon S3 master-key/AWS KMS master key.
4. If we select Amazon S3 master-key, we do not need to provide any key but if we select AWS KMS master key, we need either to provide our own custom keys or we can use the CMK service which provides aws/s3 key which is dedicated to Amazon S3.

LAB - II

1. Go to bucket properties.
2. There's an option of Default Encryption which automatically encrypt object when they are stored in S3.

The old way to enable default encryption was to use a bucket policy and to refuse any HTTP command without proper headers:

```
{  
    "Version": "2012-10-17",  
    "Id": "PutObjPolicy",  
    "Statement": [  
        {  
            "Sid": "DenyIncorrectEncryptionHeader",  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::<bucket_name>/*",  
            "Condition": {  
                "StringNotEquals": {  
                    "s3:x-amz-server-side-encryption": "AES256"  
                }  
            }  
        },  
        {  
            "Sid": "DenyUnEncryptedObjectUploads",  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::<bucket_name>/*",  
            "Condition": {  
                "Null": {  
                    "s3:x-amz-server-side-encryption": true  
                }  
            }  
        }  
    ]  
},
```

The new way is to use the "default encryption" option in S3. Also, we should note that Bucket policies are evaluated before "default encryption".

3. Choose either AES-256 or AWS-KMS to encrypt the objects.
4. Click on Save.
5. Click on upload.
6. Add an file and select None in Encryption section under properties Tab.
7. Click on Upload.
8. After encryption we can see the property of the file to be encrypted.

Security

There are 2 types of security:

1. User based

- a. IAM policies: we can restrict what the user can do through Amazon S3. It handles which API calls should be allowed for a specific user from IAM console.
2. Resource based
- a. Bucket policies
 - b. Object Access Control List
 - c. Bucket Access Control List

Note: An IAM principal can access an S3 object if:

- The user IAM permission allow it or
- The resource policy allow it.
- We also need to make sure that there's no explicit DENY. If users through IAM is allowed to access the S3 bucket but the bucket policy is explicitly denying the user to access it then we'll not be able to access it.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicRead",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::examplebucket/*"
      ]
    }
  ]
}
```

- a) Bucket policies: it is a JSON based policy. It has 4 major elements:
 - a. Resources: buckets & objects to which the policy applies to.
 - b. Actions: Set of API to allow or deny.
 - c. Effect: Allow or deny
 - d. Principal: the account or user to apply the policy to.
- b) Use of Bucket Policy:
 - a. Grant public access to the buckets
 - b. Force objects to be encrypted at upload.
 - c. Grant access to another account.
- c) Bucket settings for Block Public Access

- a. Block public access to buckets and objects can be granted through:
 - i. New access control list
 - ii. Any access control list
 - iii. New public bucket or access point policies.
 - b. We can block public and cross account access to bucket and objects through any public bucket or access point policies.
 - c. These settings were created to prevent company data leaks.
 - d. It can also be set account level.
- d) Networking:
- a. S3 supports VPC endpoints which means the instances in VPC that don't have internet access can talk through internal network
- e) Logging & audit:
- a. S3 access logs & audit can be stored in S3 buckets. We should not store the logs of S3 in the same bucket because there may occur recursion which can increase the file size in GB(s).
 - b. API logs can be stored in CloudTrails.
- f) User security
- a. MFA can be required in versioned buckets to delete objects.
 - b. Pre-signed URLs (URL that're valid for limited time) can be used (ex: premium videos).

LAB

1. Go to S3 under services
2. Select your bucket
3. Go to permissions -> Bucket policy
4. Click on “policy generator” below the editor.
5. Click on S3 bucket policy under “select type of policy”.
6. We'll deny 2 things, first the headers which do not contain header for encryption keys & headers which do not contain right value for encryption.
7. So select ‘Deny’ from ‘Effect’.
8. Principal be ‘*’ i.e., for all users.
9. Select action “PutObject”.
10. ARN can be found out under the bucket policy and add ‘/*’ at the end.

11. Click on “Add conditions”, we are specifying here permissions setting the condition to be null, selecting the key to be s3:x-amz-server-side-encryption and value to be true.
12. Click on “add condition”.
13. Now let’s add a second statement.
14. Follow steps 5-10 & now click on “Add conditions”. Set condition as: ‘StringNotEquals’, key should be s3:x-amz-server-side-encryption, & value should be ‘AES256’ & click on add condition.
15. Click on generate policy.
16. Copy the JSON created & paste it in the bucket policy & hit ‘Save’.
17. Try to upload a file in the bucket without selecting any encryption in the Permission page and we get an error.
18. Now, select “Amazon S3 master key” in encryption & upload, it gets uploaded.

Storage Gateway

It is service that connects an on-premises software appliance with cloud-based storage to provide seamless & secure integration between an organization’s on premises IT environment & AWS infrastructure. The service enables you to securely store data to the AWS cloud for scalable & cost-effective storage.

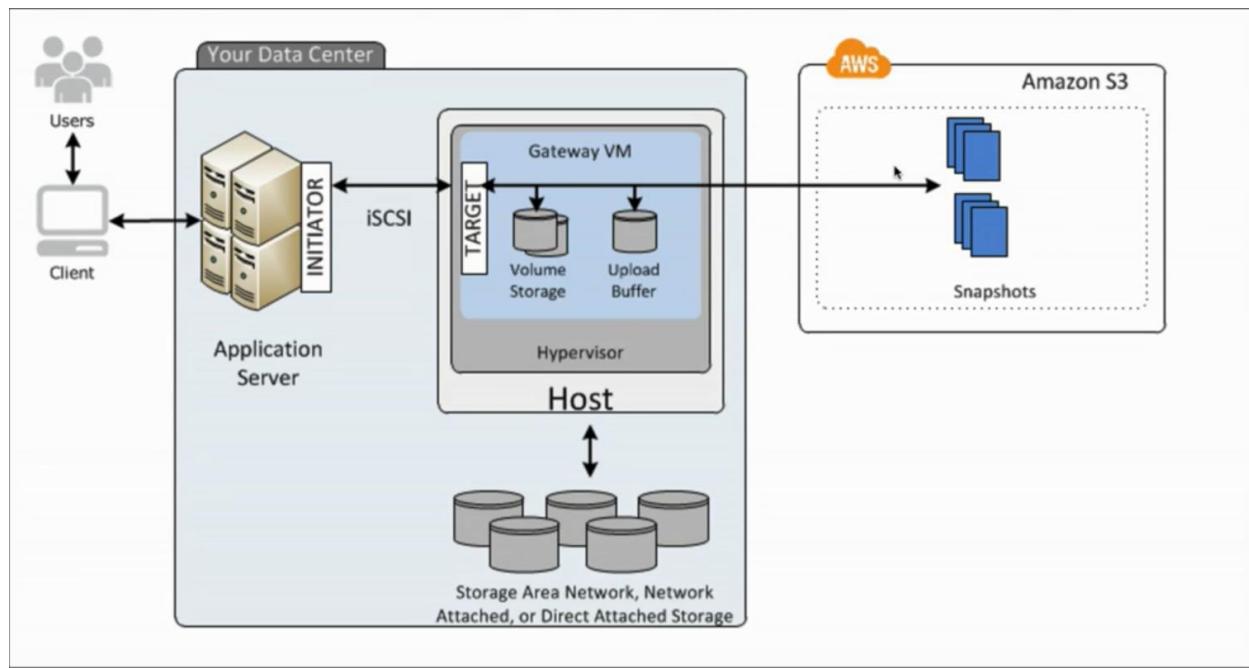
AWS Storage Gateway’s Software appliance is available for download as a Virtual Machine image that you install in your host datacenter. Storage Gateway supports either VMware ESXi (Elastic Sky X Integrated) or Microsoft Hyper-V.

Once the gateway is installed & associated with AWS account through the activation process, you can use AWS Management Console to create the storage gateway option that is right for you. Types of Storage Gateways:

- 1) **File Gateway (NFS – Network File System)**: It is where we store flat files in S3 keeping storage cost minimum. It allows us to store PDF’s, Word files, pictures, videos. Files are stored as objects in S3 buckets, accessed through NFS. Ownership, Permissions & timestamp are durably stored in S3 in the user-metadata of the object associated with the file.

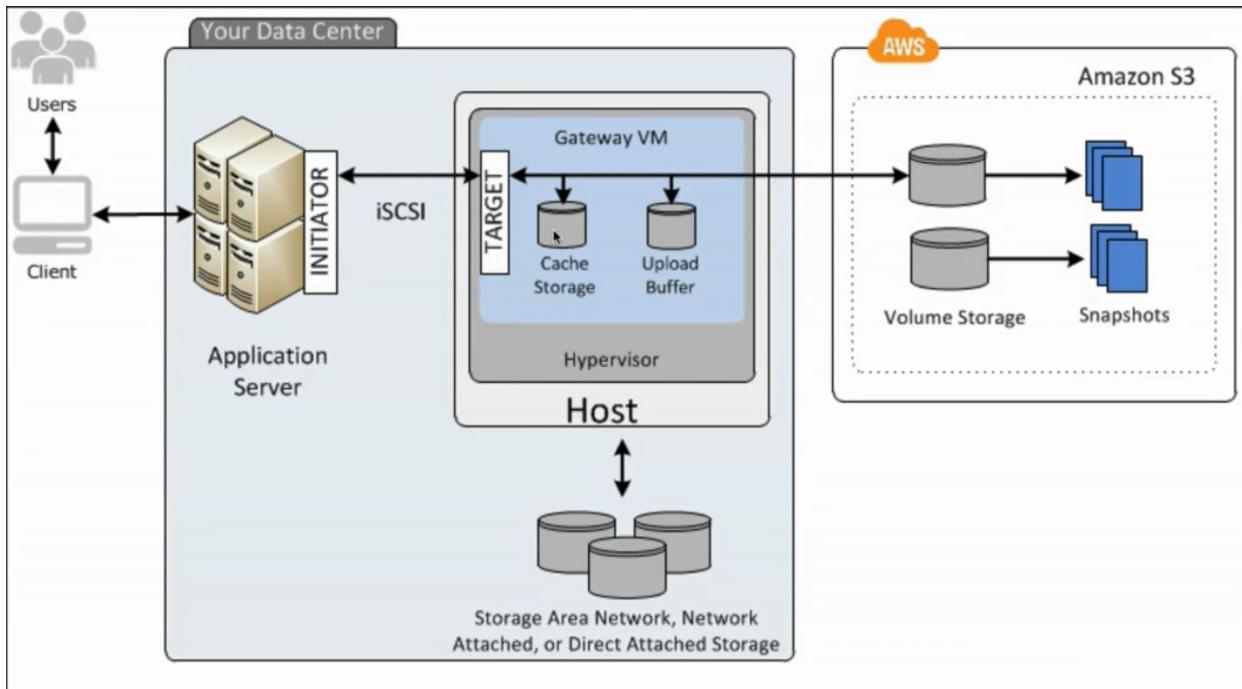
Once the objects are transferred to S3, they can be managed as native S3 objects, and bucket policies such as versioning, lifecycle management & cross region replication apply directly to objects stored in your bucket.

- 2) **Volume Gateway (iSCSI):** It uses block based storage. Here we can run OS, can be used as virtual Hard disk or as a VM running on. Data written to these volumes can be asynchronously backed up as point in time snapshots of our volumes & stored in the cloud as Amazon EBS (Elastic block store – It's a virtual hard disk attached to EC2 instance). It can be considered as snapshots. Snapshots are incremental backups that capture only changed blocks. All snapshot storage are compressed to minimize storage charges. It consists of 2 types:



- a) **Stored Volumes:** It is where we store entire copy of dataset on premise, while asynchronously backing up data to AWS cloud. Stored volumes provide our on-premise application with low latency access to their datasets, while providing durable, off-site backups. We can create storage volumes & mount them as iSCSI devices (It can be an end node such as storage device, or it can be an intermediate device such as bridge between IP & Fiber channel devices) from your on-premise application servers.

Data written to stored volumes is stored on our on premise storage hardware & it is backed up to S3 in the form of EBS snapshots. The size for stored volume ranges from 1 GB to 16 TB.



- b) Cached Volumes: It is where we store most recently accessed data on premise. It let you use S3 as primary data storage while retaining frequently accessed data locally on our storage gateway. Cached volume minimize the need to scale our on premise storage infrastructure while providing our application with low latency access to their frequently accessed data. We can create volumes up to 32 TB.
- 3). Type Gateway (VTL): It is a backup & archiving solution. It allows us to create virtual tapes and then to S3. Later we can use policy tapes to send them to Glacier. It uses popular backup application like NetBackup, Backup Exec, Veeam, etc.

Snowball

AWS Import/Export Disk accelerates moving large amounts of data into & out of the AWS cloud using portable storage devices for transport. It can import to S3 & export from S3. It transfers our data directly onto & off of storage devices using Amazon's high speed internal network & bypassing the internet. Using

Snowball, addresses common time challenges with large scale data transfers including high network costs, long transfer times, & security concerns. There are 3 types of snowballs:

1. Snowball
2. Snowball edge
3. Snowmobile



Snowball: Snowball is a petabyte-scale data transport solution that uses secure appliances to transfer large amounts of data into & out of AWS. Amazon gives us an appliance, we load data into it and send that to Amazon. Later, the data is exported to S3. It is simple, fast, secure & just 1/5th the cost of high speed internet. In US, Snowball comes in 2 sizes: 50 TB & 80 TB. All regions except US have 80 TB snowballs.

Snowball uses multiple layers of security. They are designed with temperature resistant layers. It uses AES-256 encryption. They have industry standard Trusted Platform Module (TPM) which is designed to ensure security & full chained custody of our data. It comes with kindle & it can be tracked wherever it is a particular time. Once the data transfer job has been processed & verified, AWS performs a software erasure of the Snowball appliance.

Snowball edge: It is a 100 TB data transfer device with on-board storage & compute capabilities. It is like AWS datacenter that we can bring on our premise. We can also run lambda functions & got S3 storage too. It can be used to move large amounts of data into & out of AWS, as a temporary storage tier for large local datasets, or to support local workloads in remote or offline solutions. So, it helps us ensuring our application continue to run even when they are not able to access the cloud.

E.g.: Airlines engine manufacturers can pull Snowball edges onto Airplanes & collect data about how that aircraft engine is running & when airplane lands, we can take the snowball edge out & ship it back to the data center.



Snowmobile: It is a 45 feet long ruggedized massive shipping container on the back of a huge truck. It contains 4 Petabyte or Exabyte amount of data. Currently, it has 100 Petabyte capacity. It makes easy to move massive volumes of data to the cloud, including video libraries, image repositories, or even complete data migration.

To get a snowball:

1. Go to Migration under AWS services.
2. Click on Snowball
3. Click on “create a job”
4. Select your country.
5. Fill in your address.
6. Give your Job name & select storage.

7. Give your bucket name in which data is to be moved.
8. Create an IAM role
9. Create SNS topic & enter your mail address.
10. Review the form & “Create the job”.

To start the snowball:

1. After the job has been created, you'll get credentials.
2. Download the manifest file & move it to command line directory file.
3. Plug in the power & Ethernet cable, you'll get an IP address.
4. Open command prompt & start snowball using the command:
`./snowball start -i 192.168.22.22 -m manifest_file_name -u 16_digit_client_unlock_code`
5. To copy a file: `.snowball cp hello.txt s3://bucket_name`

Transfer Acceleration



It uses the CloudFront Edge Network to accelerate our upload to S3. Instead of uploading directly to your S3 bucket, we can use distinct URL to upload directly to an edge location which will then transfer that file to S3. Here, we're minimizing the public internet and maximizing the private AWS network. So, we need to change the endpoint, not the code. You'll get a distinct URL to upload to a URL like: acloudguru.s3-accelerate.amazonaws.com

1. Create a bucket with default properties.
2. Go to its properties.
3. Go to Transfer Acceleration & enable it.
4. We get an endpoint like this:

technoronicstransferaccelrn.s3-accelerate.amazonaws.com

5. We can compare our data transfer speed in between direct upload speed to transfer accelerated upload speed in different regions.

Creating static website using S3

S3 can host static websites & have them accessible on the www. The website URL will be:

1. <bucket_name>.s3-website.<AWS_region>.amazonaws.com
Or
2. <bucket_name>.s3-website-<AWS_region>.amazonaws.com

If we get 403 forbidden error, make sure the bucket policy allow public reads.

We can use S3 to host static websites. They are serverless. They are very cheap & scales automatically.

LAB

1. Create a bucket
2. Go to properties
3. Select Static website hosting
4. For now, select “Use this bucket to host a website”.
5. Go to Permissions, and make sure the rules which give public access to the bucket has been unchecked (last 2).
6. Define the name of index document & Error document.
7. Upload an index.html & error.html (with content).
8. Make sure, you’ve made the content public, otherwise the end users will get an error.
9. We also need to create bucket policy.
10. Select type of Bucket policy to be S3 bucket policy.
11. Select effect to Allow.
12. Select Principal to be *. It means policy applies to anyone.
13. Select service to be S3.
14. Select Actions to be GetObject.
15. Copy the ARN of the bucket provided in the permissions Tab.
16. Add '/*' at the end.
17. Click on Add statement.
18. Generate the policy.
19. Copy the generated JSON and paste it inside the Bucket policy tab.

20.Hit on Save.

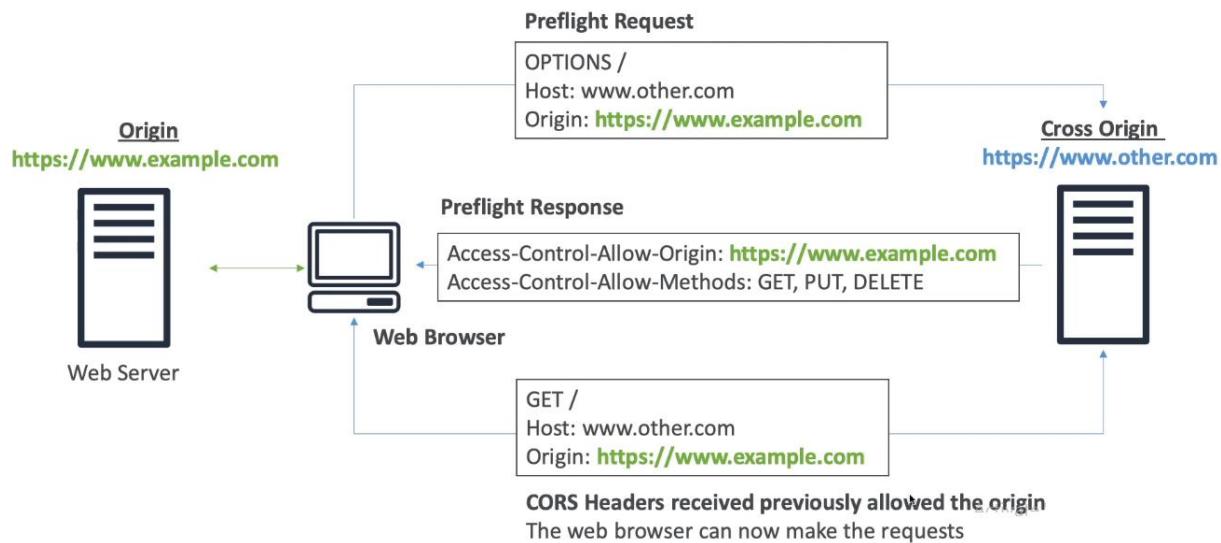
21.Click on the below link to view your website.

22.<http://technoronicks3staticwebsite.s3-website.ap-south-1.amazonaws.com/>

CORS

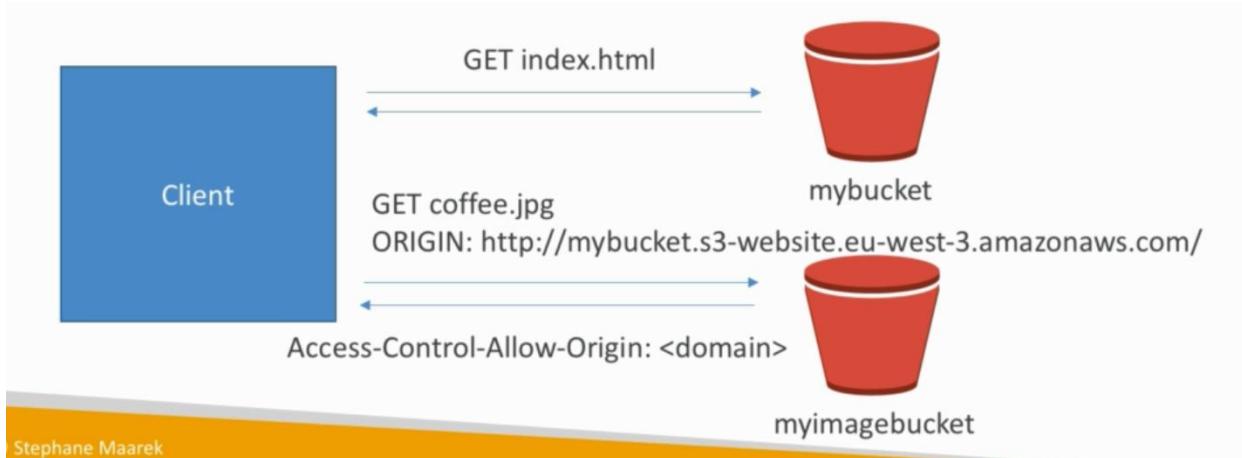
Cross Origin Resource Sharing limits the number of websites that can request files in S3 (limits cost). If we need to enable data from another S3 bucket, we need to enable S3 CORS.

- An origin is a scheme (protocol), host (domain) and port.
- Web-browser based mechanism to allow requests to other origins while visiting the main origin.
- Same origin: <http://example.com/app1> & <http://example.com/app2>.
- Different origin: <http://example.com> & <http://other.example.com>
- The request won't be fulfilled unless the cross origin allows for the request, using the CORS headers.



The web browser visits our first web-server (<http://example.com/>) and since it is the first visit we do, it is called the origin. There's a second web-server k/a cross-origin since it has a different URL (<https://www.other.com>). the web-browser visits our first origin and it is going to be asked from the files that're uploaded from the origin to make a request to the cross-origin. Web-browser makes a pre-flight request and it asks the cross-origin that if it allowed to do a request on it saying that the website <http://example.com> is sending me to you,

and can I make a request to your website. Cross-origin allows the web browser and authorizes some of the method like GET, PUT, and DELETE. The web-browser makes a GET request



Let us take an example, we've 2 buckets: mybucket&myimagebucket which contain static websites & pictures related to index.html respectively. Client connects to mybucket & request for index.html file & in response he gets it. It happens that in index.html file, the file request for an image in myimagebucket.

So the client says here is my origin: <http://technoronicks3staticwebsite.s3-website.ap-south-1.amazonaws.com/>. But the myimagebucket denies to share the image file unless the origin is approved. So, we need to enable CORS headers in myimagebucket to allow any request that comes from above origin. If the CORS is not set properly then client get 403 forbidden error.

Points to remember

- If a client does make a cross-origin request on our S3 bucket, we need to enable the CORS headers correctly.
- We can enable CORS headers for all origin (*) or for a specific origin.

S3 Performance

1. Behind the scene, each object goes to S3 partition & for the best performance, we want the highest partition distribution. Each partition is basically a server & the more server is involved, the higher throughput we get.

2. We can have random characters before the key name to optimize performance. We can add 4 random characters before the key like:
 - a. <my_bucket>/**5r4d**_myfolder/myfile.txt
 - b. <my_bucket>/**a914**_myfolder/myfile.txt

This forces Amazon to partition data correctly & put on different partition.
3. It was recommended not to use dates to prefix keys otherwise the partition would be very similar.
 - a. <my_bucket>/**2020_03_01**_myfolder/myfile.txt
 - b. <my_bucket>/**2020_03_02**_myfolder/myfile.txt
4. If we use SSE-KMS encryption, we may be limited to our AWS limits for KMS usage.
5. Using caching for frequently accessed contents.
6. Amazon S3 automatically scales to high request rates, latency 100-200 ms.
7. Our application can get at least 3500 PUT/COPY/POST/DELETE and 5500 GET/HEAD request per seconds per prefix in a bucket.
8. There's no limit to number of prefixes in our bucket.
9. Example (object path - Prefix):
 - a. Bucket/folder1/sub1/file - /folder/sub1
 - b. Bucket/folder2/sub2/file - /folder/sub2
 - c. Bucket/1/file - /1/
 - d. Bucket/2/file - /2/

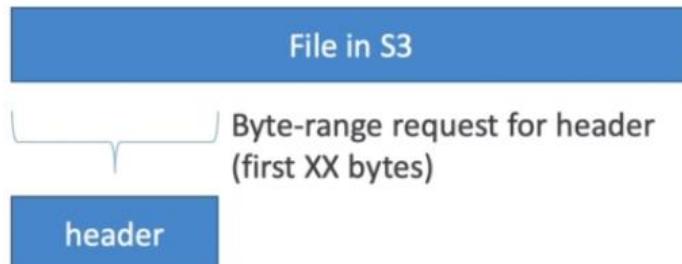
If we spread reads across all 4 prefixes evenly, we can achieve 22000 request per second for GET and HEAD.

10.S3 - Byte range fetches

- a. Parallelize GETs by requesting specific byte ranges.
- b. Better resilience in case of failures.
- c. It is used to **speed up downloads** or can be used to **retrieve only partial amount of the file**.
- d. All the request to fetch the file are made in parallel.



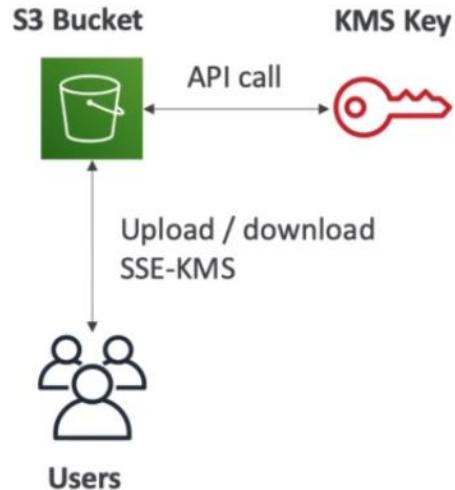
Let's have a large file in S3. We want to request the first part, second part and then the N part. So we request all these part as specific range of the file and all these request can be made in parallel, hence speeding up the downloads.



Talking about the second use case is to only retrieve a partial amount of the file. For ex: We know that first 50 bytes contain header, then we can just issue a header request with a byte-range request for the headers.

KMS Limitation

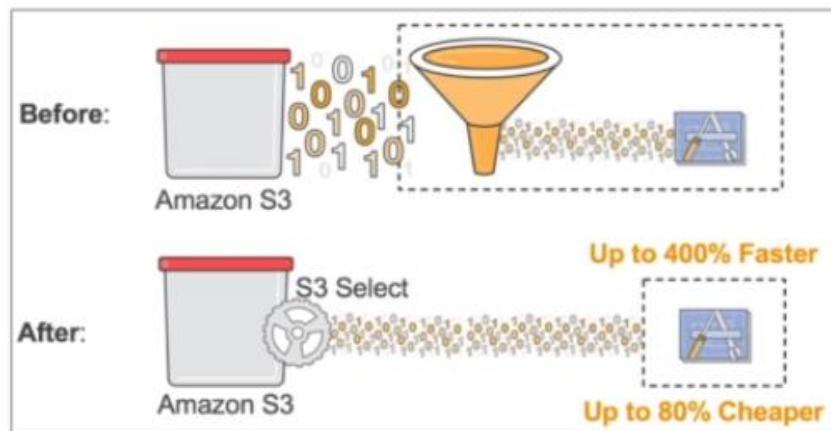
- If we use SSE-KMS, we may be impacted by the KMS limits.
- When we upload a file, it calls GenerateDataKey KMS API.
- When we download a file, it calls the Decrypt KMS API.
- By default, KMS has a quota per second of 5500, 10000, 30000 request per second which is region based and it cannot be changed. So, if we've more than 10,000 request per second in a specific region which only supports 5500 request per second for KMS, then we'll be throttled.



Our users connect to the S3 bucket, and they want to upload/download a file using SSE-KMS encryption. S3 bucket will make an API call to either GenerateDataKey or Decrypt to a KMS key.

S3 Select & Glacier Select

- Retrieve less data using SQL by performing server side filtering.
- Can filter Rows & Columns (Simple SQL statements).
- Less network transfer, less CPU cost client-side.

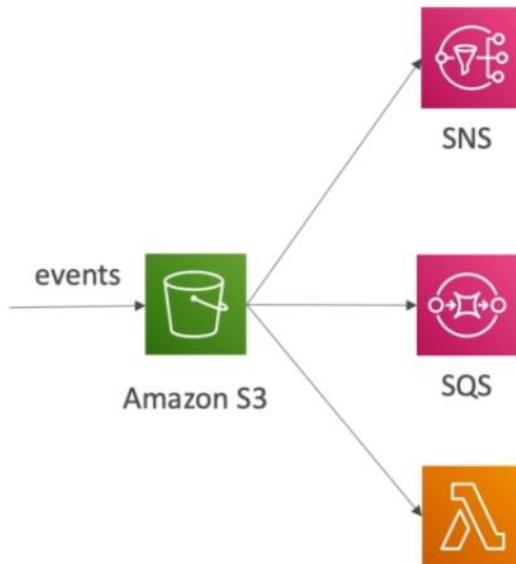


Earlier S3 sends all the data to our application and then we filter it application side but now, we filter the data from S3 using S3 select and it only gives us the data we need. It is 400% faster and 80% cheaper since there's less network traffic, less CPU and the filtering happens server side.

S3 Event notifications

Let's suppose some event happens in our S3 buckets like creation, removal, restoration or replication of object and we need to react to all these events. So, we can create rule and we can apply filter to object type (like only for *jpg files).

Use case: Generate thumbnails of images uploaded to S3. The possible targets for S3 event notifications. It can be



1. SNS to send notifications in email.
 2. SQS to add messages into a queue.
 3. Lambda functions to generate some custom code.
- S3 event notifications typically deliver events in seconds but sometimes may take minute or longer.
 - If 2 writes are made to single non-versioned object at the same time, it is possible that only a single notification will be sent. Hence if we want to ensure that event notification is sent for every successful write, we can enable versioning.

LAB

1. Create a bucket with default properties.
2. Enable versioning.
3. Go to Events tab.
4. Click on Add notification and give your event a name.

5. Select the Events on which notification needs to be sent.
6. Select SQS Queue under Send to option.
7. Select Add SQS queue ARN under SQS.
8. Go to SQS under services.
9. Create a new SQS queue and give it a name.
10. Select it to be standard queue. Make sure we're setting up the queue where the bucket is.
11. Create Queue.
12. Under details tab, copy the ARN of the queue.
13. Under Permissions tab, click on Add a permission.
14. Select effect to "Allow".
15. Select Everybody under principal.
16. Under Actions, select SendMessage.
17. Click on Add permission.
18. Under SQS queue ARN, paste the above value.
19. Click on Save.
20. Upload an object.
21. Go to SQS queue and start polling for messages.

AWS Athena

- It queries data in S3 without loading it into a database directly using the SQL query language.
- It is a serverless service to perform analytics directly against S3 files.
- We can use SQL language to query the files.
- It has a JDBC or ODBC driver to connect BI tools to it.
- We get charged per query and amount of data scanned.
- It supports different types of languages such as: CSV, JSON, ORC, Avro and Parquet. (it is built on Presto, which is a query engine).
- Use cases: BI, Analytics, reporting, analyze & query VPC flow logs, ELB logs, CloudTrail, S3 Access logs, etc.

LAB

1. Go to Athena under Services.

2. Before running query, we need to setup query result location in AWS S3. Go to settings, and set the location.
3. Hit Save.
4. We can choose the DB to be default now for working on these queries.
5. So, we will first create a DB and point to the new DB.
6. Then we'll create a table. Search Google for "How to analyze S3 server access logs using Athena" or go to:
<https://aws.amazon.com/premiumsupport/knowledge-center/analyze-logs-athena/> and copy the entire query and replace the location of the logging bucket.
7. We can see that tables has been created and click on the down arrow and the different columns that has been created.
8. Click on 3 dots and click on Priview table.

S3 Object lock & Glacier Vault Lock

- S3 Object Lock
 - Adopt a WORM (Write Once Read Many) model.
 - Block an object version deletion for a specified amount of time.
- Glacier Vault Lock
 - Adopt a WORM model.
 - Lock the policy for future edits (can no longer be changed even by Admins).
 - Helpful for compliance & data retention.

6. Elastic Compute Cloud (EC2)

It is a web service that provide resizable compute capacity in the cloud. Amazon EC2 reduces the time to obtain & boot new server instances to minutes, allowing us to quickly scale capacity, both up & down, as our computer requirements change. They are basically the virtual machines in the cloud. It can be virtual windows, linux servers. Using this, we can change the configuration of processors, cores per processor, RAM, disk size requirements, whether to setup with tiered storage or SSD drives.

It consists of the capability of:

1. Renting virtual Machines (EC2).
2. Storing data on virtual drives (EBS).

3. Distributing load across machines (ELB).
4. Scaling the services using auto-scaling group (ASG).

Amazon EC2 changes the economics of computing by allowing you to pay only for the capacity that we actually use. It helps developers the tools to build failure resilient applications & isolate them from common failure scenarios. There are 2 types of virtualization:

1. HVM (Hardware Virtual Machine)
2. PV (Para Virtual)

Note: One major disadvantage with Para-virtualization: You need a region-specific kernel object for each Linux instance. Consider a scenario where you want to recover or build an instance in some other [AWS region](#). In that scenario, you need to find a matching kernel — which can be tedious and complex. Nevertheless, I can't say that this is the only reason that Amazon now recommends using the HVM virtualization versions of the latest generation of their instances: There are a number of additional recent enhancements in HVM virtualization which have improved its performance greatly.

We can connect to our EC2 instance using:

1. SSH
2. Putty
3. EC2 instance connect

[Launching an EC2 instance](#)

1. Go to Services & click on EC2. Before choosing an instance make sure you are in the region closest to you. It reduces the response time and hence fastens our requests.
2. Click on 'Launch instance'.
3. For now, select 'Amazon Linux AMI'.
4. Select and configure the instance details (or the machine type), It says about how powerful our machine is going to be, no. of vCPU we want to have, amount of memory.
5. We can configure the number of instances
6. We can select a network or else even we can create a new VPC (Virtual Private Cloud, it's nothing but a data center)

7. Select any of the Subnet. It allows us to select the availability zones of that region. One subnet has only one availability zone.
8. Choose one the tenancy types from Shared, Dedicated & Dedicated Host.
9. We can select either of the shutdown behavior between Stop or Terminate. Terminating an EC2 instance takes the virtual machine & no charges are further applied on instance usage. Also, EBS volumes will be detached & deleted. While if we STOP an instance, The VM is taken away and we are not charged for this instance any more but the attached EBS volumes are not deleted. The volume continue to persist in availability zones & standard charges for EBS volumes apply.
10. We can enable monitoring which include monitoring of CPU utilization, Disk I/O, Network in-out. It can be done at every 5 minute or 1 minute.
11. We can pass bootstrap script to our EC2 instance in Advance Details section.
12. After filling the details, we can Add Storage. The volume type in the storage section is ‘root’ which contains 3 types as studied i.e., General Purpose SSD, Provisioned IOPS SSD and Magnetic. We can Add New Volume to the instance.
13. We can notice that by default ‘Delete on Termination’ checkbox is ticked.
14. We can Add Tags to our EC2 instance (The ability to apply a specific tag to all non-essential EC2 instances would make it possible to quickly search and filter out critical instances that need to remain active from those that can be safely scheduled for shutdown so that we can lower the cost.)
15. Configure security groups allows us to choose protocol to open to a network traffic. It acts as a Firewall to our instance. Selecting ‘My IP’ from ‘Source’ group allows only a particular IP to access your web address. For now, select the Type to be SSH as we are going SSH into the instances.
16. On reviewing & launching we might see a screen which says “Select an existing key pair or create a new key pair” which wants to convey that AWS allows us to connect to our instances securely. A key pair consist of a public key that AWS store & private key file that we store.
17. After configuring the Key Pair, launch the EC2 instance.
18. We can view our instance.

Termination protection are turned off by default. If we want to terminate our EC2 instance, Go to your instance and then click on instance settings and on instance settings click on change termination protection and click on “Disable”.

System status checks – It verifies that our instance is reachable. If it fails there's issue with AWS power, networking or software system. In this case, we need to restart or replace the instance.

Instance status checks – It verifies that our instance's operating system is accepting traffic.

Monitoring enable us to monitor the monitor CPU utilization, network, etc. Detailed monitoring help us to monitor every minute.

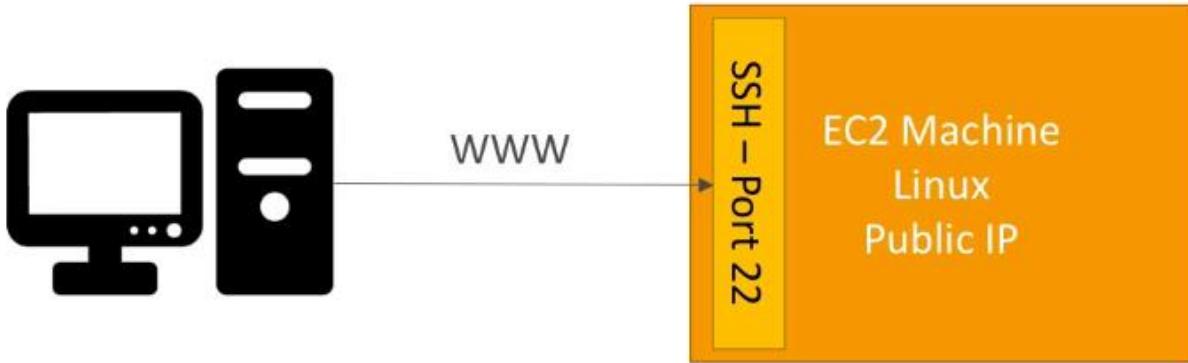
After launching the instance, if we go to the Description tab, we can see the public IP of our EC2 instance. Under Security groups, we can view the inbound rules, which gives us the information of which incoming ports to allow.

How to use Putty

```
~/aws-course ➤ ssh ec2-user@35.180.100.144
The authenticity of host '35.180.100.144 (35.180.100.144)' can't be established.
ECDSA key fingerprint is SHA256:gLqFnUlIDsBNQZFkmzJLGNRTtry2CbQ8L2N3ZUU0DTYQ.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '35.180.100.144' (ECDSA) to the list of known hosts.
ec2-user@35.180.100.144: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
x ➤ ~/aws-course ➤ ls
EC2Tutorial.pem ➤ identity
~/aws-course ➤ ssh -i EC2Tutorial.pem ec2-user@35.180.100.144
@@@@@@@@@@@WARNING: UNPROTECTED PRIVATE KEY FILE! @
Permissions 0644 for 'EC2Tutorial.pem' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "EC2Tutorial.pem": bad permissions
ec2-user@35.180.100.144: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
x ➤ ~/aws-course ➤ chmod 0400 EC2Tutorial.pem
~/aws-course ➤ ssh -i EC2Tutorial.pem ec2-user@35.180.100.144

          _\   _ )
         -|  (  /  Amazon Linux[2 AMI
         __\_\_|_|
```

Putty is used to SSH in to our EC2 instance. SSH is one of the most important function. It allows us to control a remote machine using a command line.



We have our EC2 machine & we've launched Amazon Linux 2 AMI on it with a public IP. Now, we want to access this machine. We need to add a security group allowing port 22 on it. Finally, our Machine will access over the web through that port 22 accessing the EC2 machine. Check that under security settings only the correct user has the full control over the .pem file & we should disable inheritance on it.

Putty KeyGen is used to convert .pem (the key pair file when we generated the EC2 instance) file to .ppk files.

1. Download Putty & Putty KeyGen
2. Open Putty KeyGen & Click on LOAD
3. Browse to the .ppm file downloaded which was created while launching an EC2 instance (step 16 in “Launching an EC2 instance”) & open it.
4. Save the file as .ppk extension.
5. Now, open PUTTY & in ‘Session’ Category, type “**ec2-user@35.154.32.14**” in the Host Name. Also, copy the Host Name to saved session for further use & then go ‘connection’ category & select ‘SSH’ and in that sub-section, click ‘Auth’ & browse to the file path where ‘.ppk’ is located.
6. Go back to session, click on ‘Save’.
7. Now, click on host name in “Saved session” & “Load” & hit ‘Open’.
8. Login as a super user – **sudosu**
9. Update the EC2 instance – **yum update -y**

Note: If we're launching EC2 instance directly, it shows permission denied error, we also need to provide the key. Even after using the key, we get an error that it is an unprotected key file. When we first download a file the permission is 0644 which is too open which can leak the private key. Since the private key is accessible by others, it says bad permission. To fix this, we do chmod 0400 and then we reference the key file. (Chmod is used to change the permission/access of the file system objects).

EC2 pricing

EC2 instance pricing (per hour) varies on these parameters:

1. Region you're in.
2. Instance type you're using.
3. On demand vs Spot vs Reserved vs Dedicated host.
4. Linux vs Windows vs Private OS.
5. We're billed by the second, with a minimum of 60 seconds.
6. Other factor such as storage, data transfer, fixed IP, public IP address, load balancing.
7. We do not pay for the instance if the instance is stopped.

AMI

An Amazon Machine Image is a special type of virtual appliance that is used to create a virtual machine with Amazon EC2 (Elastic Compute Cloud). It is software and the operating system that'll be launched on the server. AWS comes with base image such as:

1. Ubuntu
2. Fedora
3. RedHat
4. Windows, etc.

These image can be customized at runtime using EC2 user data (bash script). Even we can create our own image which is used to create our instances. AMIs can be built for Linux or Windows Machine. Custom built AMI can provide the following advantages:

1. Pre-installed packages needed.
2. Faster boot time (no need for long EC2 user data at boot time).

3. Machine becomes configured with monitoring/enterprise software.
4. Security concerns – control over the machines in the network.
5. Control of maintenance & updates of AMIs over time.
6. Active Directory Integration out of the box.
7. Installing our app ahead of time (for faster deploys when auto-scaling).
8. We can use someone else's AMI that is optimized for running an app, DB, etc.

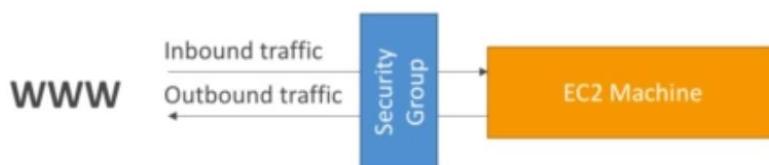
When we built an AMI, it is built for specific AWS region!!!

To choose the right instance type, we must check these 5 distinct 5 characteristics:

1. RAM (type, amount, generation)
2. CPU (type, make, frequency, number of cores)
3. The I/O (disk performance, EBS optimization)
4. Network (network bandwidth, network latency)
5. GPU

Security Groups

It is a virtual firewall & it controls traffic to out instances. They control traffic is allowed into or out of EC2 machines. When we launch our EC2 instance, we associate it with one or more security groups i.e., 1 instance can have multiple security groups. Also, we can have multiple instances within a security group & an instance can have multiple security groups.

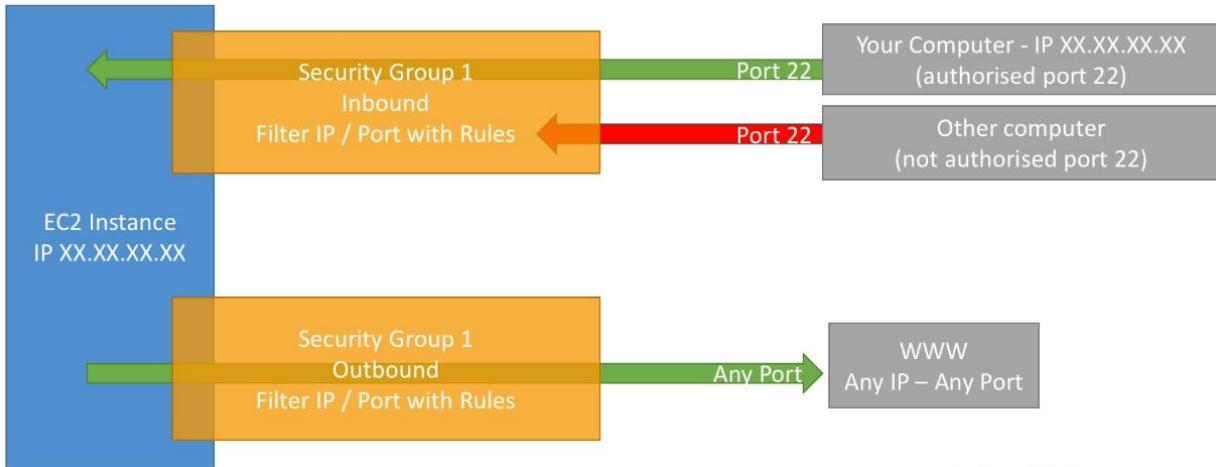


They regulate:

1. Access to the ports.
2. Authorized IP ranges – IPv4 & IPv6
3. Control of inbound & outbound network.

To install Apache

1. Open Putty
2. Run command as super user – `yum install httpd -y`

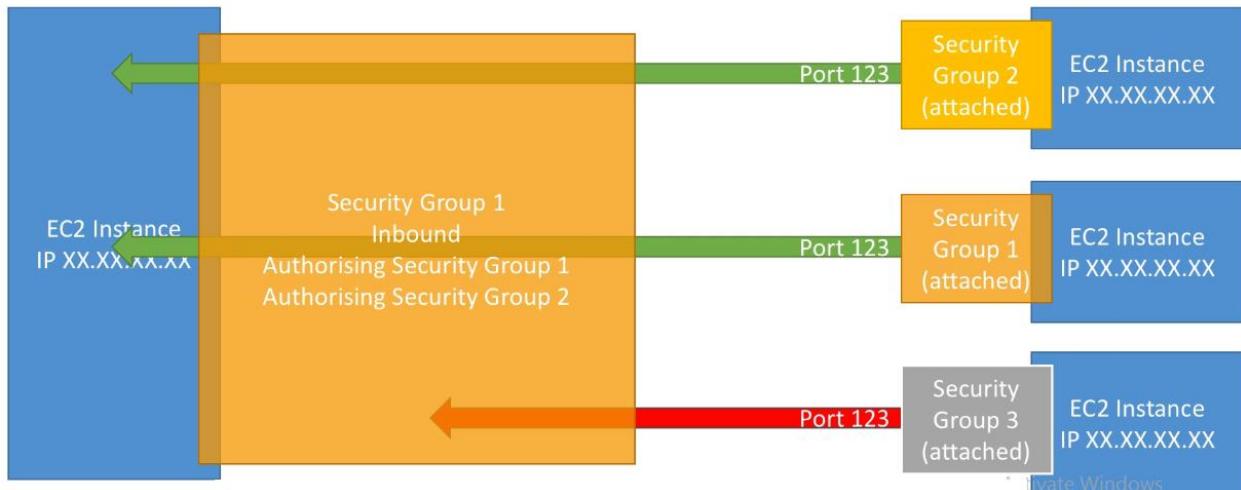


Let's suppose we've an EC2 instance and we have a security group with inbound & outbound rules logically separated. We've a computer having an IP which is authorized on Port 22. If it is not authorized, then our IP will be blocked by the security group. Now, let's suppose we have any IP and Port which wants to connect to our EC2 instance since all the outbound rules are open.

Points to remember

1. Security group can be attached to multiple instances.
2. Security groups can be locked down to a region/VPC which means they are not accessible in another AZ or VPC.
3. It lives outside EC2 – it means if the traffic is blocked, the EC2 instance won't see it even.
4. It is good to maintain one security group for SSH, as remote connections are most complicated things and they should be done correctly.
5. If our application is not accessible, then it is a security group issue.
6. If our application received a connection refused error, then it is an application error or it is not launched.
7. All inbound traffic is blocked by default & all outbound traffic is authorized by default.
8. If there're multiple security group attached to an instance, the rules get added.

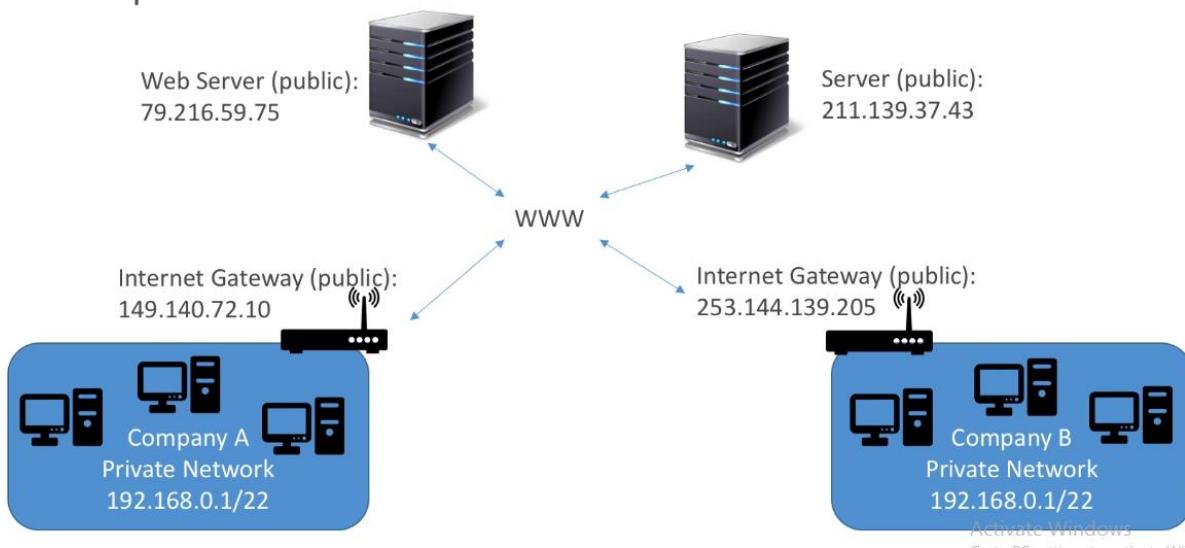
Referencing other security groups



We are referencing one security group from other security group. We've an EC2 instance attached to a security group and inbound rules says that Security Group 1 & Security Group 2 is authorized is authorized to connect to it. Now, if we launch other instances with security group 1 & 2 attached to it, we can directly connect to the EC2 instance from this EC2 instance. But if we've an EC2 instance having security group 3 which is not bounded to inbound rules, then it restricts the connection.

Private vs Public IP

1. Networking has 2 types of IP:
 - a. IPv4: 1.160.123.223 - IPv4 allows for 3.7 billion different address
 - b. IPv6: 3ffe:1900:4545:3:200:f8ff:fe21:67cf
2. Public IP can be identified on the internet whereas Private IP can be identified on private network only.
3. Must be unique across the whole world whereas Private IP must be unique across the private network but 2 different network may have the same IP.
4. Machines connect to the private IP using an internet gateway.
5. Only a specified range of IPs can be used as a private IP.
6. Our EC2 instance has 2 types of IP, Public & Private one. We cannot SSH using private IP since we are not in AWS network. But if we SSH using Public IP and when we look at the name of the instance, it says ec2-user@PrivateIP. Hence this private IP helps our instance identifiable on the private network of AWS



Let's suppose we have 2 servers having Public IP, so they can communicate with each other. Let's suppose we've a 3rd organization which has a private IP. So all computers inside that organization can talk to each other. If we attach an internet gateway, then these instances will also get access to other servers.

Elastic IP

1. When we stop & start an EC2 instance, it can change its public IP but private IP remains the same.
2. If we need a fixed public IP for our instance, we need an Elastic IP & it can be attached to one instance at a time.
3. An Elastic IP is a public IPv4 IP as long as we don't delete it.
4. We can attach one instance to one Elastic IP.
5. With an Elastic IP address, we can mask the failure of an instance or software by rapidly remapping the address to another instance.
6. We can only have 5 Elastic IP in our account (further we can ask AWS to increase it.)
7. Overall try to avoid using Elastic IP
 - a. They often reflect poor architectural design.
 - b. Use random public IP & register DNS name to it.
 - c. It'd be better to use a Load Balancer & don't use a public IP.

LAB

1. Go to Elastic IP under EC2 Services.
2. Click on "Allocate Elastic IP address."

3. Select the default settings for now & click on 'Allocate'.
4. Now, select the IP address to be associated, click on 'Actions' & navigate to "Associate Elastic IP address".
5. Select the instance name & click on 'Associate'.
6. If we want to release the Elastic IP, Right click on your instance settings as Elastic IP address incur lot of charged..=.
7. Click on Networking and click on Disassociate Elastic IP Address.
8. Click on Yes.
9. As we can see a new IP has been provided now.
10. We also need to release it from Elastic IP section. So go to Elastic IPs under EC2 dashboard.
11. Click on the Elastic IP.
12. Click on Actions and click on Release Elastic IP address.

Note: If we stop our machine now, our IPv4 is not going to be lost.

Apache web server

It is used to establish a connection between the servers and browser of website visitors (Chrome, Firefox, etc.) while delivering the files back and forth between them (client-server architecture).

Install Apache server

1. Before launching Apache, we'll update our environment through command:
`yum update -y`
2. `yum install -y httpd.x86_64`
3. `service httpd start` or `systemctl start httpd.service`
4. To start our Apache server start automatically whenever after EC2 instance gets booted up – `chkconfig httpd on` or `systemctl enable httpd.service`
5. `curl localhost:80` loads the content whatever the URL contains. It returns HTML page now. If we open the try to open the same page in web browser, it timeouts since Port 80 (HTTP) is not allowed in the inbound rules.
6. Edit inbound rules & reload the webpage, we can see the Apache Test page.
7. In the terminal, type `echo "Hello World!" > /var/www/html/index.html`
8. On page reload, we can see "Hello World!" displayed on the HTML page.

9. in the terminal type echo "Hello World from \$(hostname -f)" > /var/www/html/index.html. It returns "Hello World from ip-123.45.67.089.eu-west3.compute.internal".

EC2 User Data

- It is possible to bootstrap our instances using an EC2 User data script.
- Bootstrapping means launching the commands when machine starts.
- The script is only ran once at the instance first starts.
- EC2 user data is used to automate boot task such as:
 - Installing updates
 - Installing software
 - Downloading common files from the internet.
 - The EC2 user Data Script runs with the permission of that of a root user.

EC2 Options

- 1) On demand – allow us to pay fixed rate by hour (or by the second) with no commitment.
- 2) Reserved – provide us with the capacity reservation, & offer significant discount on the hourly charge for an instance. 1 year or 3 year terms.
- 3) Spot – unused EC2 instance that's available for less than on-demand price. These instances run until we terminate them or EC2 instances interrupt, or when Spot price (the hourly price for spot instance) exceeds your maximum price or when the demand for spot instance rises or when the supply for spot instance decreases.
- 4) Dedicated host – Physical dedicated servers dedicated for us. It can help reduce the cost by allowing us to use existing server bound software licenses.

On Demand

1. Highest cost since instance is up on demand & flexibility of Amazon EC2 without any upfront payment or long term commitment.
2. Application with short term, spiky or unpredictable workloads that cannot be predicted.

3. Application being developed or tested on Amazon EC2 instance for the first time.
4. We pay for what we use. (Billing is per second, after the first minute).
5. They're great for elastic workloads.

Reserved

1. Application with steady state or predictable usage.
2. Application that require reserved capacity.
3. Users able to make upfront payments to reduce their total computing cost even further.
4. Reservation period can be 1 or 3 years.
5. Recommended for steady state usage applications.
6. There are 3 types of Reserved instances:
 - Standard RI's (Up to 75% off on demand) – It enables us to modify the availability zones scope, network, platform, & instance size of RI. They can be sold in the reserved Instance marketplace.
 - Convertible RI's (Up to 54% off on demand) – We've the capability to change the instance family, instance type, OS, or tenancy associated with their RI. They cannot be sold in the reserved Instance marketplace.
 - Scheduled RI – They are available to launch within the time windows we reserve. It allows us to match our capacity reservation to a predictable recurring schedule that requires only fraction of days, week or month. E.g.: month end sale, etc.

The conclusion of Convertible vs. Standard Reserved Instances comparison is that if we've a stable workloads that'll remain stable for three years, the best option is Standard RI in order to maximize the discounts.

However, if our AWS environment is constantly changing, the best option is to purchase is Convertible RI to benefit from the price discounts &flexibility & let the Convertible RI Exchanger make recommendations about how our assets are to be managed.

Spot

- Can get a discount of up to 90% compared to On-demand.

- Instances we can lose any point of time if our max price is less than the current spot price.
- Most cost efficient instances in AWS.
- Useful for workloads that're resilient to failure.

They are used for:

1. Application that have flexible start & end times.
2. Application that're only feasible at low computing prices.
3. Users with urgent computing needs for large amounts of computing capacity.
4. Application like:
 - Batch Jobs that can be retried.
 - Data analysis that can be used with distributed computing.
 - Image processing, or any such applications that is possible to retry.

Note: They should not be used for critical jobs or databases since DB needs to be up and running for years. A good combination can be using reserved instances for our baseline capacity like running a web application for an year and we know we're going to run at least 10 instances for that application with On-Demand & Spot instances for peaks.

Dedicated hosts

1. We get a dedicated physical server to use.
2. We've full control of EC2 instance placement.
3. Visibility into underlying sockets/physical cores of the hardware.
4. Allocated for our account for a 3 period reservation.
5. Useful for regulatory requirements that may not support multi-tenant virtualization.
6. More-expensive.
7. Great for licensing which does not support multi-tenancy or cloud deployments or for the software that have complicated licensing model (BYOL - bring Your Own License).
8. It is good for the companies having strong regulatory or compliance needs that is we don't want our hardware to be shared by anyone.

9. Can be purchased on-demand (hourly).
10. Can be purchased as a reservation for up to 70% off the On-Demand price.

Characteristic	Dedicated Instances	Dedicated Hosts
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	X
Affinity between a host and instance		X
Targeted instance placement		X
Automatic instance placement	X	X
Add capacity using an allocation request		X

Dedicated Instances

1. Instances running on hardware dedicated to us only.
2. It may share the hardware with the other instances but within the same accounts.
3. We don't have the control of instance placement. We can only move hardware after Start/Stop.

Which host is right for us?

- On Demand: Coming & staying in the resort, whenever we like we pay the full price.
- Reserved: Like planning ahead & if we plan to stay for a long time, we may get a good discount.

- Spot instances: The hotel allows people to bid for the empty rooms and the highest bidder keeps the room. We can get kicked out any time if someone comes and pay higher price than us.
- Dedicated hosts: We book an entire building of the resort.

Pricing

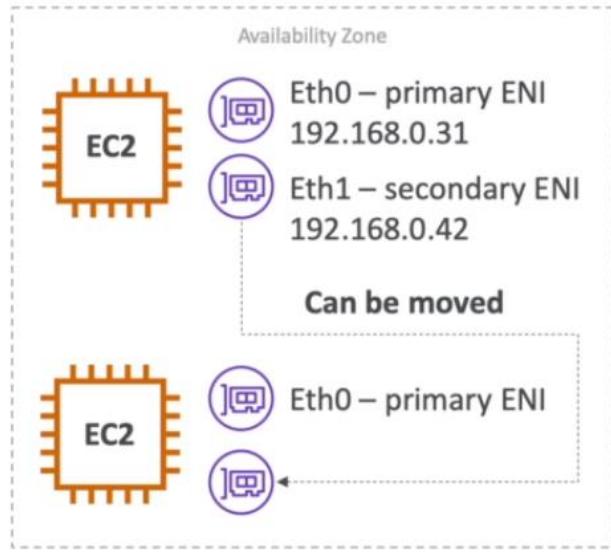
Price Type	Price (per hour)
On-demand	\$0.10
Spot Instance (Spot Price)	\$0.032 - \$0.450 (up to 90% off)
Spot Block (1 to 6 hours)	~ Spot Price
Reserved Instance (12 months) – no upfront	\$0.062
Reserved Instance (12 months) – all upfront	\$0.058
Reserved Instance (36 months) – no upfront	\$0.043
Reserved Convertible Instance (12 months) – no upfront	\$0.071
Reserved Dedicated Instance (12 months) – all upfront	\$0.064
Reserved Scheduled Instance (recurring schedule on 12 months term)	\$0.090 – \$0.095 (5%-10% off)
Dedicated Host	On-demand price
Dedicated Host Reservation	Up to 70% off

LAB

1. We can look for the different types of EC2 options in EC2 under Services section.
2. Click on **Reserved instances**.
3. Click on Purchase Reserved instances.
4. Select the Platform, Instance types, etc. and click on Search.
5. Click on the instance you want to purchase, click on Add to Cart.
6. We can view the total cost on the bottom left.
7. For **Scheduled Instances**, click on Scheduled Instances on EC2 Dashboard. These are only available in some of the selected regions, so if you don't find on the console, select some other region.
8. Click on Begin.
9. Fill in the specifications as per the requirement and hit "Find Schedules".
10. For **Dedicated host**, go to Dedicated host in EC2 dashboard.
11. Click on Allocate Dedicated Host.
12. Specify the instance as per the requirements and click on Allocate Host.

13. For **Spot request**, click on Spot Requests under EC2 dashboard. We can check the spot price history and we can see the price range is very stable. We can also see the difference between the difference in spot price & on-demand price.
14. Click on Request Spot Instances.
15. We can select Spot instance based on the required workload.
 - a. Load balancing workloads, if we want to have a web service.
 - b. Flexible workloads, if we want to have batch jobs.
 - c. Big data workloads, if we want to launch various types of instances to run some computations.
 - d. Defined direction workloads, which is a spot block saying if we want to book our instances for 1 hour to 6 hour.
16. Fill in the specification as per the requirement.
17. If we select Load Balancing workloads, we can choose Fleet request which allows us to choose the type of instances (remove the checkbox of Apply recommendations).
18. At the end we have the estimated price.
19. Hit Launch.
20. But if we want to request just **one spot instances**, click on instances from the EC2 dashboard.
21. Click on Launch instances.
22. Select the AMI and the instance type.
23. Under Instance details, click the checkbox which says "Request Spot Instances", it shows the current price of the Spot instances.
24. We can set the maximum price we want for the spot instance. But if the instance price goes above our price, then AWS reclaim our instance.
25. The persistence request gives us the possibility to choose what happens once our instance gets reclaimed. Should we stop our instance or Terminate it.
26. We can also choose the duration between our instance is valid under "Request valid to" section.
27. We can choose the Tenancy, either we can run our instances on a Shared Hardware or Dedicated one or on the Dedicated Host.

Elastic Network Interface



- It is a virtual network interface.
- They're Logical component in a VPC that represent a virtual card.
- These give EC2 instances access to the network though they're outside the EC2 instances as well.
- For example, we've an availability zone, and we've one EC2 instance and to it is attached is eth0 (our primary ENI) which provides our EC2 instance network connectivity (can be like a Private IP).
- Each ENI can have the following attributes:
 - Primary private IPv4 (referring to ENIs), one or more secondary IPv4. If we've another IPv4, we will have another IP address.
 - Can have one Elastic IP (IPv4) per private IPv4.
 - Or one Public IPv4, hence providing us a public or a private IP.
 - Can have one or more security groups attached to our ENI.
 - Can have a MAC Address attached.
- We can create ENI independently and attach them on fly on EC2 instances for failover.
- They're bound to a specific availability zone.
- Let's suppose, we've another EC2 instance which have another ENI attached to it, so we can move eth1 from the first EC2 instance to the second EC2 instance to move that private IP. It is very helpful for failovers.

LAB

1. Launch a EC2 instance selecting number of instances to be 2 under configure Instance details.
2. We'll run them in the same availability zone, selecting the Subnet to be a specific one.
3. Select other inputs to be default and Launch the instance.
4. On viewing the instance, if we see the switch to the Network Interfaces tab and clicking on the Interface ID and clearing the filters, we can see the two different Public & Private IP address.
5. We can even create our own network interfaces by clicking on "Create Network Interface".
6. Give it a name "My secondary ENI".
7. Provide the same subnet selected in step 2.
8. Attach the same security group as attached to the instances.
9. Hit on Create.
10. If we go to and check the interface, we can see there are 3 ENIs and the status of the newly created ENI is available while the other two are in-use.
11. Right click on the newly created ENI and click on Attach.
12. Select either of the instances & hit Attach.
13. Go back to instances & refresh the instances.
14. We can see that one of the instance is having two ENIs providing 2 different Private IP.
15. We can move this secondary ENI to the other EC2 instance.
16. Click on the network interface.
17. Select the Secondary ENI and click on Detach.
18. Again click on Attach and attach it to the other EC2 instance.

EC2 instances types

1. **General purpose instances:** It provide balance of compute, memory, & networking resources, & can be used for variety of workloads. Family: A1, M5, T2.
 - I. **A1** – Ideally suited for scale-out workloads that are supported by ARM ecosystem. Applications are: Web servers, Containerized micro services, Caching fleets, Distributed data stores, Application requiring Arm instruction set.

- II. **M5** – Provide ideal cloud infrastructure, offering a balance of compute, memory & network resources. Application are: Web & application servers, small & medium databases, Gaming servers, caching fleets, Running backend servers for SAP, Microsoft SharePoint, Cluster computing & other enterprise application.
- III. **T3** – They are balanced instances types. Provides a baseline level of CPU performance with the ability to burst to a higher level whenever required by our workload. But if the burst is too long, CPU becomes bad. Application are: Websites & web application, Code repositories, Development, build & test & staging environments, Microservices.

One CPU credit is equal to one vCPU running at 100% utilization for one minute. Other combinations of vCPUs, utilization, and time are also equal to one CPU credit; for example, one vCPU running at 50% utilization for two minutes or two vCPUs running at 25% utilization for two minutes.

2. **Compute optimized instances:** They are ideal for computer-bound applications that benefit from high-performance processors. They are well-suited for Batch processing workloads, media transcoding, High-performance web servers, High-performance computing, Scientific modelling, Dedicated game servers, ML. Family: C5
3. **Memory optimized instances:** These instances are designed to deliver fast performance that process large data sets in memory. Family: R5, X1.
 - I. R5– Memory intensive apps or Databases.
 - II. X1 – SAP/HANA, Apache Spark, etc.
4. **Storage optimized instances:** These are designed for workloads that require high, sequential read & write access to very large datasets on local storage. They are optimized to deliver tens of thousands of low-latency, random I/O operations per second to application. Family: D2, H1, I3.
 - I. **D2** – Fileservers, Data warehousing, Hadoop
 - II. **H1**– Data-extensive workloads, Application requiring sequential access to large amount of data, or application requiring high-throughput access to large quantities of data.

III. **I3** – No SQL DBs, Data Warehousing, High-frequency Online Transaction Processing (OLTP).

5. **Linux Accelerated Computing Instances:** It satisfies the need of high processing capabilities which provide access to hardware based compute accelerators such as Graphic Processing Units (GPUs) or Field Programmable Gate Arrays (FPGAs). It enable more parallelism for higher throughput on compute-intensive workloads. Family: F1, Inf1, P3, P2, G4, G3, G2.

- I. **F1** – Designed to accelerate computationally intensive algorithm, such as data flow or high parallel operation.
- II. **Inf1** – These instances uses AWS Inferentia machine learning inference chips.
- III. **P3** – Designed for general purpose GPU computing. Provide high-bandwidth networking, powerful, half, single & double precision floating-point capabilities, ideal for deep learning, computational fluid dynamics, seismic analysis, etc.
- IV. **G4** – Well suited for Graphic application like video transcoding, game streaming in cloud, etc.

Learning tech: DR MC GIFT PX.

D – Density

R – RAM

M – Main choice for general purpose apps

C – Compute

G – Graphics

I – IOPS

F – FPGA

T – Cheap general purpose

P – Graphics (think of pics)

X – Extreme Memory

T2 Unlimited

- It is possible to have an "Unlimited burst credit balance".

- We pay extra money if we go over credit balance, but we don't lose in performance.
- It is a new offering, so cost would be too high if we're not monitoring the health of our EC2 instances.

EBS

EC2 machine loses its root volume when they're terminated. **Unexpected EC2 terminations may happen from time to time (AWS will mail us), so we need to store our instance data somewhere safe on an external drive.** Amazon Elastic Block Storage allows to create storage volumes & attach them to Amazon EC2 instances. Once they're attached, we can create a file system on the top of these volumes, run a database, or use them in any other way you'd use a block device. **They're network drive, not physical drives so our EC2 volumes will be linked to EBS volumes using a network and there might be a bit of latency. It can be detached from an EC2 instance & attached to another one quickly.**

Amazon EBS volumes are placed in a specific availability zones (locked to an AZ which means an EBS volumes in us-east-1a cannot be attached to us-east-1b) where they're automatically replicated to protect you from a failure of a single component. We can have multiple EBS volumes attached to an EC2 instance. EBS volumes can be resized. We can only increase the EBS volumes size & IOPS (only in IO1 type). After resizing the EBS volume, we need to repartition our drive. EBS volumes are scoped to a specific AZ. We cannot attach volume of one AZ to another AZ.

EBS volumes have provisioned capacity means we can change the size, throughput, IOPS. Only GP2 & IO1 can be used as boot volumes. EBS volume types:

1. General Purpose SSD (GP2)

- I. It balances both price & performance.
- II. Ratio of 3 IOPS per GB with up to 10,000 IOPS & they've the ability to burst out to 3,000 IOPS for extended period of time for volumes at 3334 GiB& above.

2. Provisional IOPS SSD (IO1)

- I. Highest-performance SSD volumes for mission-critical low latency or high-throughput loads. Very expensive.

- II. Designed for I/O intensive applications such as large relational or NoSQL databases.
- III. Use if you need more than 10,000 IOPS
- IV. Can provision up to 20,000 IOPS per volume.

3. Throughput Optimized HDD (ST1) – Magnetic storage & used for sequential data. Low cost & are used for frequently accessed data, throughput-intensive loads.

- I. Big Data
- II. Data warehouses
- III. Log Processing
- IV. Can't be boot volumes

4. Cold HDD (SC1)

- I. Lowest Cost Storage for infrequently accessed workloads
- II. File Server
- III. Cannot be a boot volumes

5. Magnetic (Standard)

- I. Lowest cost per gigabyte of all EBS volume types that're bootable. Magnetic volumes are ideal for workloads where data is accessed infrequently, and applications where the lowest storage cost is important.

Note: We cannot mount 1 EBS volume to multiple EC2 instances, instead we use EFS. Also, on an EBS backed instance, the default action is for root EBS volume to be deleted when the instance is terminated.

LAB

1. Create an EC2 instance just adding a step to Add a volume of type EBS.
2. Select path to be /dev/sdb & size to be 2GiB for now (As a free tier eligible, we can get up to 30 GiB of free EBS volumes).
3. Select type to be general purpose SSD.
4. Disable delete on termination.
5. Add a security group which allows to SSH into our instance or create one.
6. After creating an instance, let's SSH into it.
7. Type lsblk in the cmd propmt. It shows all the attached drives (like xvda, xvdb, etc.).
8. sudo file -s /dev/xvdb

9. It'll return data. It means there's no file system on the device and we need to create one. To create a ext4 file system on the volume, run: sudo mkfs -t ext4 /dev/xvdb
10. Now, we need to mount our directory in the data folder. Run this command: sudo mkdir /data.
11. To mount data on the data folder, sudo mount /dev/xvdb /data.
12. Now doing lsblk, we can see that our xvdb drive has been mounted to the data folder.
13. To mount this EBS volume on every system reboot: sudo cp /etc/fstab /etc/fstab.orig.
14. sudo nano /etc/fstab and then we need to add a line.
15. We need to add a device name, a mount point, the file system type and some other info: /dev/xvdb/ /data ext4 defaults,nofails 0 2.
16. Exit and save.
17. sudo umount /data is used to un-mount data from EC2.

EBS volume type use cases

GP2

- Recommended for most workloads.
- System boot volumes.
- Virtual desktops.
- Low-latency interactive apps.
- Development & test environments.
- Size - 1GiB to 16TiB.
- Small GP2 volumes can burst 3000 IOPS.
- Max IOPs is 16000
- We get 3 IOPS per GB, means at 5334 we're at the max IOPS.
- The minimum number of IOPS provisioned is 100.

IO1

- Critical business application requires sustained IOPS performance, or more than 16000 IOPS per volume.
- Used for large DB workloads such as: Mongo DB, Cassandra, MS SQL server, etc.

- Size can range from 4GB - 16 TiB.
- Min IOPS provisioned is 100 & max is 64000 (Nitro instances) else MAX 32000 (other instance).
- The maximum ratio of provisioned IOPS to requested volume size (in GiB) is 50:1. Like if we request volume to be 10 GiB, then we can't provision of 600 IOPS. It will be $50 \times 10 = 500$ IOPS.

ST1

- Streaming workloads requiring consistent, fast throughput at a low price.
- Big Data, Data warehouse, Log processing.
- Apache Kafka.
- Cannot be boot volume.
- Size ranges from 500GB to 16TB.
- We get a max IOPS of 500.
- Maximum throughput of 500MB/s and can burst as well.

SC1

- Throughput-oriented storage for large volume of data that is infrequently accessed.
- Scenarios where the lowest storage cost is important.
- Cannot be boot volumes.
- Max IOPS is 250.
- Max throughput of 250MBPS.

EBS encryption

When we create an encrypted EBS volume, we get:

1. Data is encrypted inside the volume.
2. As we attach the encrypted EBS volume to an instance, communication between them is started at flight.
3. All the volumes created from the EBS volume are also going to be encrypted.
4. Snapshots created from encrypted EBS volumes are also encrypted.
5. Encryption has minimal impact on latency.
6. EBS encryption leverages keys from KMS.

EBS vs Instance store

- Some instance do not come with Root EBS volumes & they're called instance store. Volumes attached to these instance are physical whereas EBS is a network drive..
- The advantages are of Instance Store are:
 - Better IO performance.
 - Good for buffer/cache/scratch data/temporary content.
 - Data survives after reboots.
- The cons of Instance Store are:
 - On termination, instance store is lost.
 - We can't resize the instance store.
 - Backups must be operated by the user.
- Instance store is a physical disk attached to physical EC2 server where our EC2 instance is.
- It has very high IOPS because it is physically attached.
- EBS volumes are network based and there's networking between instances and hence there's limited IOPS.
- Instance store once attached, its size cannot be changed and there's a risk of data loss if hardware fails.

EBS-backed instances should fit most application workloads only when we need extremely high performance.

Points to remember:

1. Migrating an EBS volume across AZ means first backing it up (snapshot) & recreating to other AZ.
2. EBS backup use IO & we should not run them while the application is handling lot of traffic.

Reserved Instances

1. Go to EC2 dashboard.
2. Go to Instances & reserved instances.
3. Click on 'Purchase reserved instances'
4. Select the features you want to add.
5. Search the offerings as per the selected feature.

Note: Root volumes or the volume provided by Amazon added to EC2 instances cannot be encrypted. We can use a third party tool (such as bit locker) to encrypt the root volume, or this can be done creating AMI's in the console or using the API.

Let's create our first simple website.

1. Navigate to the folder where all our web files will be located using the command `cd /var/www/html`. Anything in this directory would be publically accessible.
2. Create a file name `index.json` by typing the command: `nano index.html`
3. Write a simple html page code to print 'Hello World'.
4. Save it & then open your browser & go to the IP shown in EC2 instance.
5. In the security group attached to that instance, check if the inbound rule allows port 80.
6. In the security group, there's inbound & outbound IP linked, If we even delete the outbound IP but if the same IP is in Inbound, the network traffic does not stops the traffic and the webpage gets opened which means if we add inbound rules, outbound rules are added automatically i.e., Security Groups are stateful.
7. All Inbound traffic rules is blocked by default.
8. We cannot block specific IP address using Security Groups instead use Network Access Control List.
9. We can specify allow rules, but not deny rules.

Upgrading EC2 Volume Types

1. We can't have EC2 instance in 1 availability zone & other EBS volumes in other availability zone. They need to be in the same AZ to be so that EBS can be mounted over EC2 instances.
2. Standard volume size can't be modified. It is pure magnetic & oldest storage volume that're available.
3. If we've one EC2 instance in one availability zone, and we wanna create another EC2 instance in other availability zone using the same Volume, we need to take a snapshot of that volume

- a. Go to services → EC2 → Volumes → Select the volume → Actions → Create Snapshot → Give description → Create
- b. Now, go to Snapshot & we can see that the snapshot of the volume has been created. The unique thing about this is now we can change the volume type and even availability zone.
- c. This volume can now be attached to another EC2 instance.
- d. Even we can copy the snapshot & change the Destination Region to another region.
- e. Once we copy a snapshot, we can create an image from it & later on boot it from EC2 instance. Go to Snapshot → Select the Snapshot → Actions → Create image.

Note: Snapshots are used for backups & Images are used to create new EC2 volumes from.

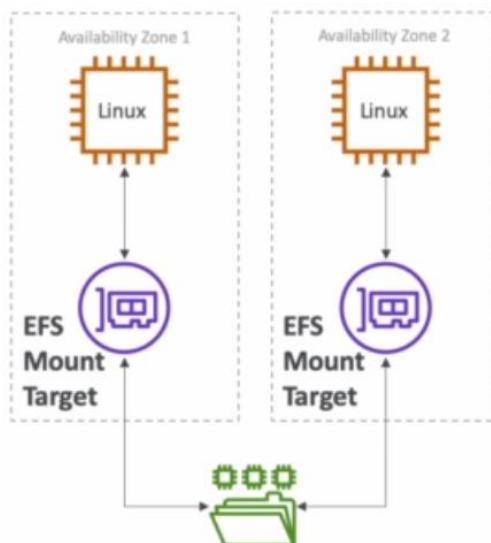
- f. The Image can be found out in AMI's section under Image.
- g. If there are multiple volume under an EC2 instance & after creating the snapshot, it's image & on launching it shows error then Go to Running Instances from EC2 dashboard → Image → Create Image → After the image has been created, remove the Volumes which are not required → Create image
- h. Even AMI's can be copied from one region to another.
- i. That's how we migrate one EC2 instance from one region to another.
- j. So, in short, Create a snapshot → Create an AMI from the snapshot → Move it to separate availability zones.

Note: By default, root device volumes can be terminated but the other EBS volumes won't be terminated.

4. Volumes exists on EBS & volumes are nothing but virtual hard disk.
5. Snapshots exists on S3, they are not directly visible as bucket or in bucket but they are stored on S3.
6. Snapshots are point in times copy of Volume.
7. Snapshots are incremental.
8. First snapshot may take time to create.

9. To create a snapshot for Amazon EBS volumes that serve as root devices, you should stop the instance before taking the snapshot. However, you can take the snapshot while the instance is running.
10. We can create AMI's from both Volumes and Snapshots.
11. You can change EBS volumes in the running state, including changing the size & storage type.
12. Snapshots of encrypted volumes are encrypted automatically.
13. Volumes restored from encrypted snapshot are encrypted automatically.
14. Snapshots can be shared, but only if they are unencrypted. These snapshots can be shared with other AWS accounts or made public.

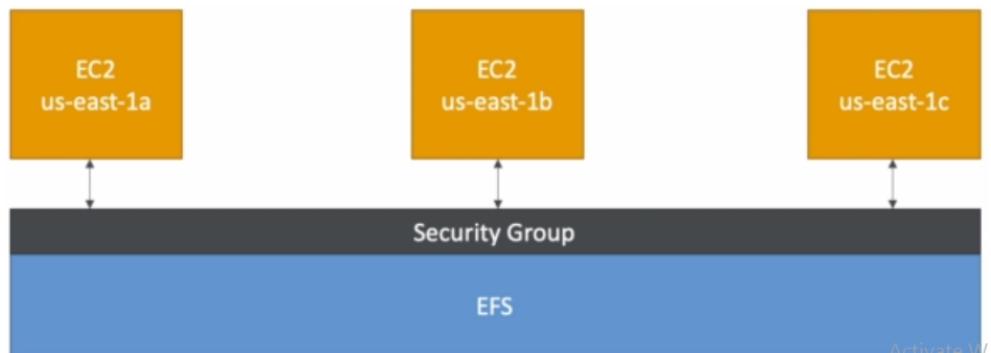
Elastic File System (EFS)



It is a file storage service for Amazon EC2, more specifically it is a Managed NFS (network File System) that can be mounted on many EC2. It is easy to use & provide simple interface. It is easy to use & provides a simple interface that allows to create & configure file system. With Amazon EFS, storage capacity is elastic, growing & shrinking automatically as you add & remove files, so your applications have the storage they need, when they need it.

- EFS works with EC2 instance in multi AZ (EBS was locked into single availability zone).
- Supports the Network File System Version 4 (NFSv4) protocol. This protocol is a standard way to mount network drive.

- We only pay for what we use. We don't need to buy 8 Gig initially. It costs 30 cents per Gig (3 times cost of GP2).
- Can scale up to Petabytes (Highly available & scalable).
- Supports thousands of concurrent NFS connections.
- Data is stored within multiple availability zones within a region.
- EFS is block based storage.
- Read after write consistency.
- Use cases: content management, web sharing, data sharing, Wordpress.
- To access the EFS file system, we need a security group.
- **EFS only works for Linux based AMIs (not for Windows).**
- Encryption at rest using KMS.
- File system scales automatically, pay-per use, no capacity planning.



Let's suppose this is our EFS and there's a network file system and we attach a security group to it to manage incoming connections. There are multiple instances attached to it in different AZs and they are all mounting the same NFS and EFS to their File system and they'll access the same files (EBS was linked to 1 instance at a time and data was not shared between multiple EC2 instances).

Performance & Storage Class

- EFS Scale
 - 1000s of concurrent NFS clients which means 1000s of EC2 instances mounting the same NFS drive at the same time.
 - It has a massive scaling of 10GB+ /s throughput.
- Performance mode (set at EFS creation time)
 - General Purpose (default). Latency sensitive use case (web server, CMS, etc.)

- Max I/O - higher latency, throughput, highly parallel (big data, media processing).
- Storage Tiers
 - We've a Lifecycle management feature - moving file after N days).
 - Standard: for frequently accessed files.
 - Infrequent access (EFS-IA): cost to retrieve files, lower price to store.
We pay a retrieval fee when we get back these files.

LAB

1. Go to S3 under Services.
2. Select EFS.
3. Click on “Create File System” to escape the customization.
4. Select on Customize.
5. Choose Automated backups.
6. Under Lifecycle management, we can say if the file is not accessed for 30 days, we can put it to EFS-IA to save cost.
7. Choose the performance mode between General purpose & Max I/O. General Purpose is for most file system. Max I/O performance mode is for tens, hundreds or thousands of EC2 instances are accessing the file system. Select Performance mode to be General Purpose for now.
8. Select the output throughput modes. It is of 2 types: Bursting & provisioned.
 - a. Bursting: EFS scales as the file system stored in the standard storage class grows. File-based storage are typically spiky, driving high level of throughput for short period & low level of throughput for rest of the period. File system can burst to 100 MiB/s i.e., 10 TB file can burst $10\text{TiB} * 100\text{MiB/TiB}$.
 - b. Provisioned: Available for applications with high throughput to storage or with requirements those allowed by bursting throughput modes.
9. Select the VPC, availability zones, Subnet, IP address & Security Groups. For each AZs we need to define a security group.
10. Go to EC2 console.
11. Click on Security groups and click on Create (for now, don't add any inbound rules).
12. We can choose to enable the encryption.

13. Review & Create File System.
14. Now go to Instances under EC2 service.
15. Create 2 instance with two different subnets (in Configure instance section) and create a new security group.
16. Go to Load Balancer & create a classic load balancer with default settings.
17. Add both the instances created in step 10.
18. Review & create.
19. The inbound rules in step 11 now should allow Security group of the EC2 instances in step 15. Its type should be NFS

Note: The 2 instances created in step 10 must be under same security group like that of EFS volume.

20. Launch the EC2 instances in Putty & login as super user (for both the instances).
21. Install apache using the command: `yum install httpd -y` & start the service: `service httpd start`.
22. Go to root directory: `cd /`
23. To mount our instances, go to EFS under service & click on your service.
24. Click on “EC2 mount instructions”.
25. We need to install EFS package in our EC2 instance: `sudo yum install -y amazon-efs-utils`
26. Create a new directory: `sudo mkdir refs`
27. Mount the file system using NFS client: `sudo mount -t nfs4 -o nfsvers=4.1,rsiz=1048576,wsiz=1048576,hard,timeo=600,retrans=2,noresvport fs-1bab0eca.efs.ap-south-1.amazonaws.com:/ /var/www/html`
28. Go to html directory using `cd /var/www/html` & create a file in any of the terminal (for e.g.: `nano index.html` & write some content to it & save it).
29. Later, we can see that the same file exist in other terminal if we go to html directory & run the `ls` command.
30. We can see the health status of the instance in the Load Balancers under EC2 Service by clicking on the load balancer that we created in step 11 under Status section of the Instances column. The status should be “InService”.

31. Copy the DNS name under the Description section of Load Balancer under EC2 service & run it on browser. We can view the content of the html web page that we created.

Note: So basically we don't need to copy every time our code from one instance to other neither run bash scripts. EFS provisions to copy the code automatically from one instance to all the another instances.

The main difference between EBS & EFS is that EBS can be mounted to single EC2 instances while EFS can mounted to multiple EC2 instances.

Use case: It can be used as a file server, hence used to create centralized repositories. So we can have multiple EC2 instance connected to the same EFS volume and all accessing the same files. Later, we can apply user-level & directory level permissions and that'd be universal across all the EC2 instances.

RAID volumes & Snapshots

RAID

RAID – Redundant Array of Independent Disks. It puts a bunch of disk together & sums up in single disk at now it acts as one disk to the Operating disk.

1. RAID 0 – Striped, No redundancy, Good performance. The bad thing is that since volume is made from multiple volumes and if any disk fails, we will lose entire RAID array.
2. RAID 1 – Mirrored, Redundancy. The good thing about this is if one disk fails, RAID array still can be used using other disk.
3. RAID 5 – Here we get 3 disk or more and we write parity to another disk.
4. Note: AWS discourages from making RAID 5 volume on EBS. Parity is a checksum. So, if one of the disk volume, we can rebuild the RAID array using the checksum. It is basically a mathematical formulae which tells what the missing data is.
5. RAID 10 – It is combination of RAID 0 & RAID 1. It is striped & mirrored, Good redundancy & good performance.

RAID array are used when we are not getting the Disk IO that we require and we have a single volume & it has been provisioned to its maximum size and we still get a higher disk IO, so we add multiple volume and we create a RAID array

to give the disk array that is required. The most common RAID array used is RAID 0 or RAID 10. This way we put multiple volume to create a single volume in a redundant array of independent disk.

1. Go to security group
2. Select your security group & Click on ‘Edit’ inbound rules.
3. Add RDP protocol & in ‘Source’ section, select ‘My IP’ or ‘Anywhere’.
4. Click on ‘Save’.
5. Go to ‘Instance’ & ‘Launch Instance’.
6. Select AMI as “**Microsoft Windows Server 2012 R2**”.
7. Select default features in ‘Choose Instance Type’ & ‘Configure Instance’.
8. In Add Storage, “Add New Volume” & add 4 General Purpose SSD volumes.
9. Select your Configured Security Group.
10. Launch the Instance by adding a Key Pair.
11. Go to ‘Instance’ & Select your Instance.
12. Under ‘Actions’ select “Get Windows Password” & select the .pem file you’ve downloaded in step 10.
13. Decrypt the password & Copy the “User name” & “Password” to your file.
14. Open Remote Desktop Connection in your mobile & enter the IP Address of your new EC2 instance, & the Username, Password saved in step 13.

Note: The steps mentioned below are operated in Remote VM

15. After the machine got booted up, we can open File Explorer to check the volumes created. As we can see, there are 4 volumes addition to a root a volume created in step 8.
16. To work on RAID array, go to Disk Management & Select the Volume, Right click on it, Delete the Volume.
17. Do the same step 16 for all the 4 volume.
18. Right click on unallocated volume & we can see for creation of different volume type.
19. Click on “New Striped Volume”. Add all the disk available & Click on ‘Next’ & ‘Finish’.

20. Now, the problem is that on taking the snapshot, the snapshot excludes the data held in the cache & by the application and the OS. This tends not to matter on a single volume, however using multiple volumes in a RAID array, this can be a problem due to interdependencies of arrays. The solution to this is taking Application Consistent Snapshot. It can be done using these steps:

- a. Stop the application from writing to the disk.
- b. Flush all the cache to the disk.

It can be done using either any of these steps:

- a. Freeze the file system.
- b. Unmount the RAID array.
- c. Shutting down the associated EC2 instance (easiest way).

Snapshots

EBS volumes can be backed up using snapshots. Snapshots only take actual space of the blocks on the volumes means if there's 5GB data on 100GB drive, then snapshot will only be of 5GB. Snapshots are used for:

1. Creating backups
2. Resizing the volume
3. Changing the volume type
4. Encrypting the volume

Create an AMI

To create a snapshot, the best practice is to first stop an instance.

1. Go to Instance
2. Select your Instance & click on 'Actions' and 'Stop' the instance.
3. Go back to 'Volume'
4. Select your Volume & click on 'Actions' and create a Snapshot.
5. Let's copy the snapshot to another Region & Encrypt it while copying.
6. Now let's go to the Region selected in step 5 & create an Image of the snapshot.
7. Now go to AMI's and select your Image & click on 'Launch'.

8. Set the defaults and we can see in “Add storage” section, the Root volume type is ‘Encrypted’ as we did it in step 5 as Snapshots of encrypted volume are encrypted automatically.
9. Add some Tag to it.
10. Launch the instance.

Note:

- We can also look for different AMI's in the AWS marketplace.
- Deregistering an AMI does not allow to launch any new instances from it.
- Volume restored from encrypted Snapshots are encrypted automatically.

AMI Types

Root volume are the volume from where we boot our Operating System from. It is of 2 types:

1. EBS backed
2. Instance store backed

We can select our AMI based on:

1. Region
2. Operating System
3. Architecture (32-bit or 64-bit)
4. Launch Permission
5. Storage for Root Device (Root Device Volume)
 - a. Instance store (EPHEMERAL Storage)
 - b. EBS backed volume

Instance Store Root Device

1. Launch an instance from community AMI's
2. Select Instance store from Root Device Type. Coming to the “Add storage” option, we can attach additional storage volumes but after launching the instance, only additional EBS volume can be attached not the instance store volumes. It can neither be encrypted.

3. The difference between these two volumes are that Instance Store can't be stopped. They can only be Rebooted or Terminated while EBS Root Device can be stopped.
4. Hypervisors are virtual machine monitor (VMM) that enables numerous virtual operating system to simultaneously run on a computer system. If hypervisor is in a failed state, we can stop the instance & start it again. Also, we can move the Virtual machine around the AWS datacenter by starting & stopping the EC2 instance. And these things are not possible with the help of instance store. If the instance fails, we lose that instance.
5. Instance store volumes can't be detached & hence cannot be added to another EC2 instance later on while EBS volume can perform such operation.
6. By default, both ROOT volumes will be deleted on termination, however with EBS volumes, you can tell AWS to keep the root device volume.

7. Load Balancers

Scalability & High Availability

- Scalability means application/system can handle great Loads by adapting.
- There're 2 kinds of scalability:
 - Vertical scalability
 - Horizontal scalability

Vertical Scalability

- Vertical scalability means increasing size of the instance (scale up/down).
- Take an example of a junior operator who just got hired. He can take only 5 calls per minute. We have a senior operator which is much greater & he can take 10 calls per minute. So we scaled junior operator to senior operator who is faster & better.
- For example, we've an instance running on t2.micro & we want to run it on t2.large.
- Vertical scalability is very common for non distributed system such as database.

- RDS, ElastiCache are services that can scale vertically.
- There is a hardware limit that how much we can vertically scale.

Horizontal scalability

- It means increasing the number of instances/systems for our application (scale out/in).
- Let's take an example who is being overloaded. In this case, we'll be hiring a second operator or n number of operators.
- Horizontal scaling means distributed system.

High Availability

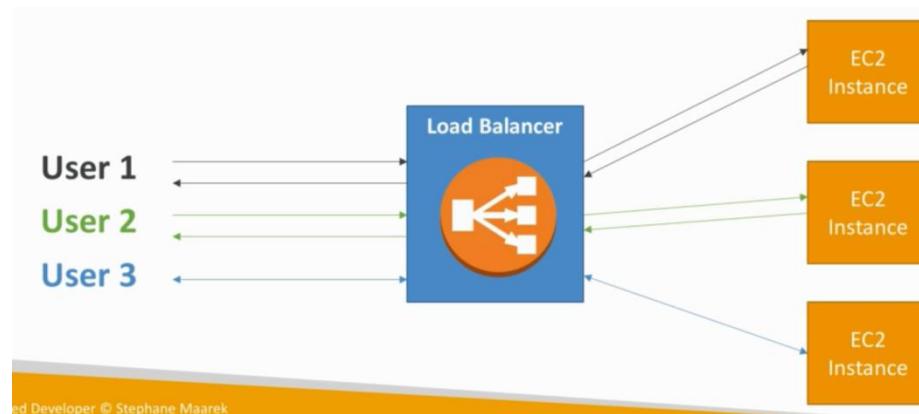
- High availability usually goes hand in hand with Horizontal scaling.
- High availability means running our application/system in at least 2 data centers.
- The goal of high availability is to survive data center loss.
- High availability can be passive (for RDS Multi AZ for example).
- The availability can be active (for horizontal scaling).

Introduction to Load Balancer

It is a virtual appliance that is going to spread the load of our appliance across different web servers. When users connect now, they don't connect to the EC2 instance, instead they connect to the load balancers. The load balancer now redirects traffic to the EC2 instance. The EC2 instance sends back a response to the Load Balancer which is forward to the user.

Here, the first user goes to first EC2 instance through the Load Balancer and the EC2 instance sends back a response to the Load Balancer which is forward to the user. When User 2 does the same mechanism, User 2 gets served in the backend by the EC2 instance & User 3 by the 3rd EC2 instance. The user just interface with the single point of entry which is Load balancer and in the backend we can scale our EC2 instance, many of them serving the traffic.

Usage



- Spread Loads across multiple downstream instances.
- Spreads a single point of access (DNS) to our application. So we just need to know the single point of access i.e., the hostname of Load Balancer & not to know about all the backend EC2.
- It seamlessly handles failure of Downstream instances through health checks.
- Does regular health checks on our instances.
- High availability across multiple AZs.
- It separates public traffic coming from users to the Load Balancer with the private traffic from Load Balancer into our EC2 instances.
- An ELB (EC2 Load Balancer) is a managed Load Balancer.
 - AWS guarantees that it'll be working.
 - AWS takes care of the upgrades, maintenance & high availability.
 - Provides few configuration knobs
- It cost less to setup our own Load Balancer but it'll be a lot more effort at our end.
- EC2 managed Load Balancer has many other services/offerings.

Health Checks



- They enable the Load Balancer to know if the instances it forward traffic to are available to reply to requests.

- The health check is done on a port and a route.
- If the response is not 200 OK, then the instance is unhealthy.
- They occur every 5 seconds which can be configured.

Points to remember:

- Classic & Application Load balancer supports SSL certificates & provides SSL termination.
- Load balancer has a static host name & they should not be resolved & use underlying IP.
- Load balancers scale but not instantaneously, so if we want it instantaneous we need to contact AWS for “warm up”.
- 4XX (e.g. 400) errors are client induced errors whereas 5XX are application induced errors.
- Load Balancer Errors 503 means Load balancer are at target, or no registered targets. It is overloading.
- If Load Balancer doesn't connect to an application, we should check the security group.
- Monitoring
 - ELB access logs will log all access requests (so we can debug per request).
 - CloudWatch Metrics gives us aggregate statistics (ex: connections count)
- We can setup 2 types of Load Balancers on AWS, an internal Load Balancer (it is private within our Account and it cannot be accessed from the Public Web) or an external Public Load Balancer or ELB which will allow our users to access our website, etc through the Load Balancer.

Load Balancer Security Groups



- Security is that we've our users talking to our Load Balancer talking to our EC2 instance.
- Users may be talking to our Load Balancer through HTTP/HTTPS.
- We'll setup a security group like this.

Load Balancer Security Group:

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	0.0.0.0/0	Allow HTTP from an...
HTTPS	TCP	443	0.0.0.0/0	Allow HTTPS from a...

- There's going to be HTTP traffic between Load Balancer & EC2, but this time our traffic is strictly restricted to our Load Balancer which means our EC2 instance expects only Load Balancer to send traffic to it.

Application Security Group: Allow traffic only from Load Balancer

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	sg-054b5ff5ea02f2b6e (load-b	Allow Traffic only...

- So, we've Application Load Balancer which only allow traffic from the Load Balancer. When we look at the source in this security group, it represent the ID of the Load Balancer security group

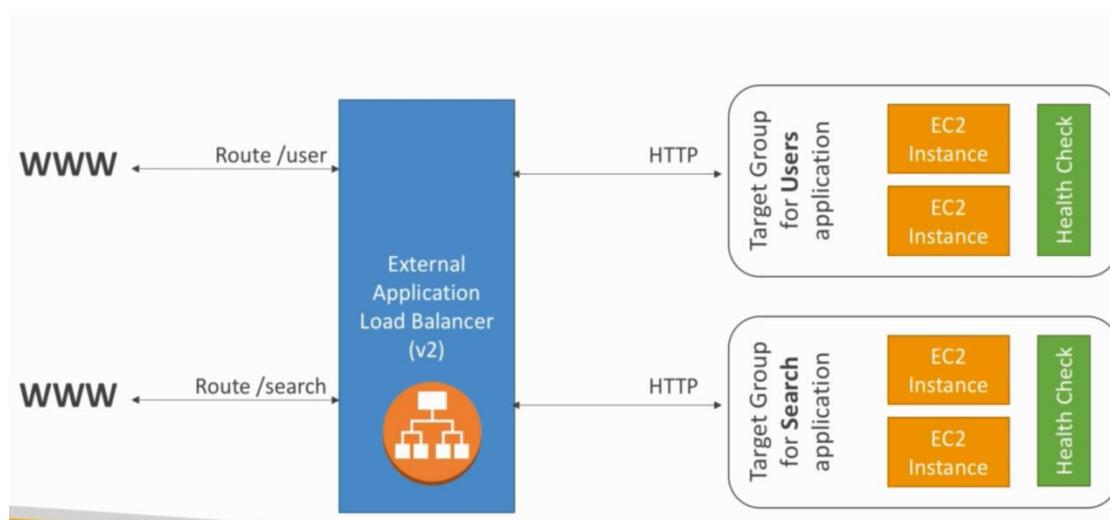
Types

There are 3 type of Load Balancers: Application Load Balancers, Classic Load Balancers & Network Load Balancers.

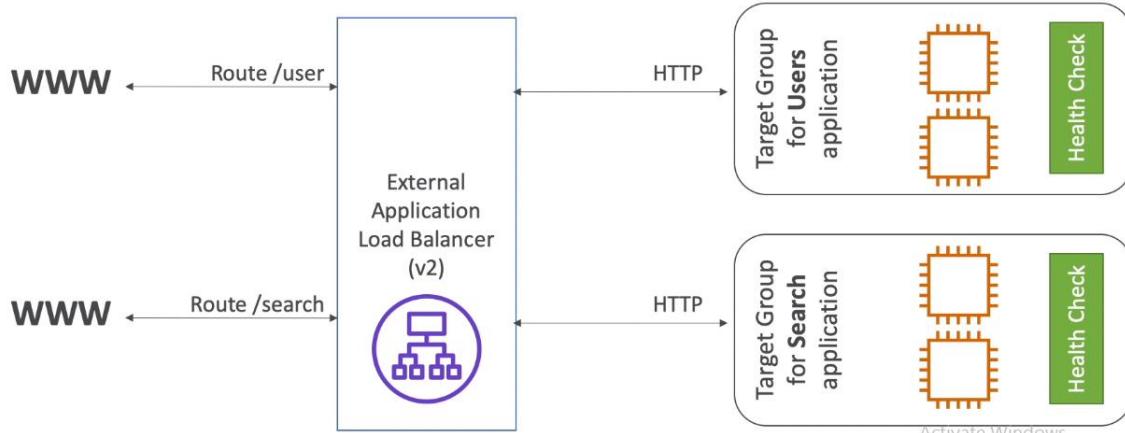
Application Load Balancer

1. It works at layer 7 i.e., at Application layer.
2. It is preferred for HTTP/HTTPS and it also supports HTTP2 and Websockets.
3. It allows us to handle multiple HTTP applications across machines (called target groups) & multiple applications on the same machine (Example:- containers) which saves more money when compared to Classic Load Balancer.
4. It also redirects traffic from HTTP to HTTPS.

5. Routing can be done cleverly using Application Load Balancers. It also supports root routing. There's a routing based on different target groups.
- Routing based on the path of the URL (ex: abc.com/users & abc.com/posts).
 - Routing based on hostname in the URL (ex: one.example.com & other.example.com).
 - Routing based on query strings & headers (ex: abc.com/user?id=123&order=false)



Load balancers allow us to do load balancing based on route and hostname in URL. They are awesome for microservices & container-based applications like Docker & Amazon ECS because it has a port mapping feature which allows us to redirect to a dynamic port in ECS instances. In comparison, we need multiple CLBs per application, then we need a single ALB per application.



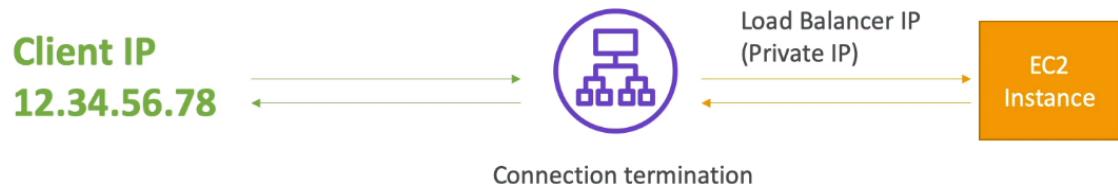
Here we've an External Application Load Balancer which is public facing & behind it we've our first target group made of EC2 instances which is going to be routing for the Route/user.

We've a second target group made up of EC2 instances again which is going to be our search application and there's going to be a health check with it with a health check. It is going to be routed through Rules for Route/search. Both the application are behind the same ALB which knows intelligently to route to which of these target groups based on the route that is going to be used in the URL.

Points to remember:

- Stickiness can be enabled at a target group level i.e., same request goes to same instance, it can be done by enabling cookies.
- ALB supports HTTP/HTTPS & Websockets protocols.
- The application servers don't see the IP of the client directly.
 - The true IP of the client is inserted in the header X-Forwarded-For (it is a header for identifying the originating IP address of a client connecting to a web server through an HTTP proxy or a Load Balancer).
 - We can also see the port (X-Forwarded Port) & protocol (X-Forwarded Protocol)
- We get a fixed hostname (XXX.region.elb.amazonaws.com)
- The application load balancer don't see the IP of the client directly.

- The true IP of the client is inserted in the header X-Forwarded-For.
- We can also get Port (X-Forwarded-Port) and Proto (X-Forwarded-Proto).



- It means our client IP is directly talking to our Load Balancer which performs connection termination and when Load Balancer talks to our EC2 instance, it is going to use the Load Balancer IP which is a private IP in our EC2 instance. For the EC2 instance to know the client's IP, it'll have to look at these extra headers in our HTTP request which are called X-Forwarded -Ports & -Proto.

Target Group

- They can be EC2 instances which are managed by Auto Scaling group.
- Then can be ECS task.
- They can be Lambda functions - HTTP request is translated into JSON event.
- It can be IP address - must be private IPs.
- ALB can route to multiple target groups.
- Health checks are done at the group level.

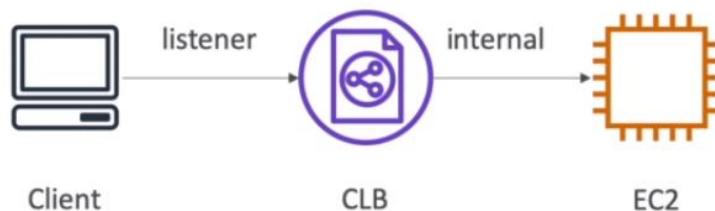
LAB

1. Go to EC2 under ‘Services’.
2. Click on “Load Balancers”.
3. Select Application Load Balancer.
4. Give a name to your ALB and IP address type IPv4.
5. Select the scheme, “internet-facing” is for public IP & ‘internal’ for private.
6. Select your availability zone (for now select all 3) & Security Group.
7. Select the target type (Instance, IP or Lambda functions) in “Configure Routing” options.

8. Customize the health check.
9. Register two of the instance under target group.
10. Launch the instance.
11. After the ALB gets launched, browse the DNS name.
12. Now, the instance is publicly accessible through its IP address but if we want that these instances should be communicated only through ALB launched in step 26, then go to the security group attached to that instance & click on Inbound rules. In the source section, instead of “0.0.0.0/0”, it will come from the security group created for the ALB.
13. Create another target group and add the third instance to this group.
14. Go back to Load Balancer and navigate to Listeners and edit the rules.
15. Click on view/edit rules.
16. Here we can see the default action is forwarding the request to our first target group.
17. Click on edit to insert a new rule & add the condition like based on Path, Host header, Source IP, etc.

Classic Load Balancer

- It works at layer 4 i.e., It does its routing decisions on TCP layer or it sometimes does layer 4 routing.
- It supports HTTP, HTTPS & TCP. The Health checks are TCP or HTTP based. Also, we've a fixed hostname XXX.region.elb.amazonaws.com.



- We've a Classic Load Balancer talking to our client on a HTTP listener and internally CLB will be redirecting the traffic to EC2 instances
18. Login to your SSH using Putty.
 19. Change to Super user.
 20. Create a healthcheck.html file using the command `nano healthcheck.html` & writing just “This instance is healthy”.

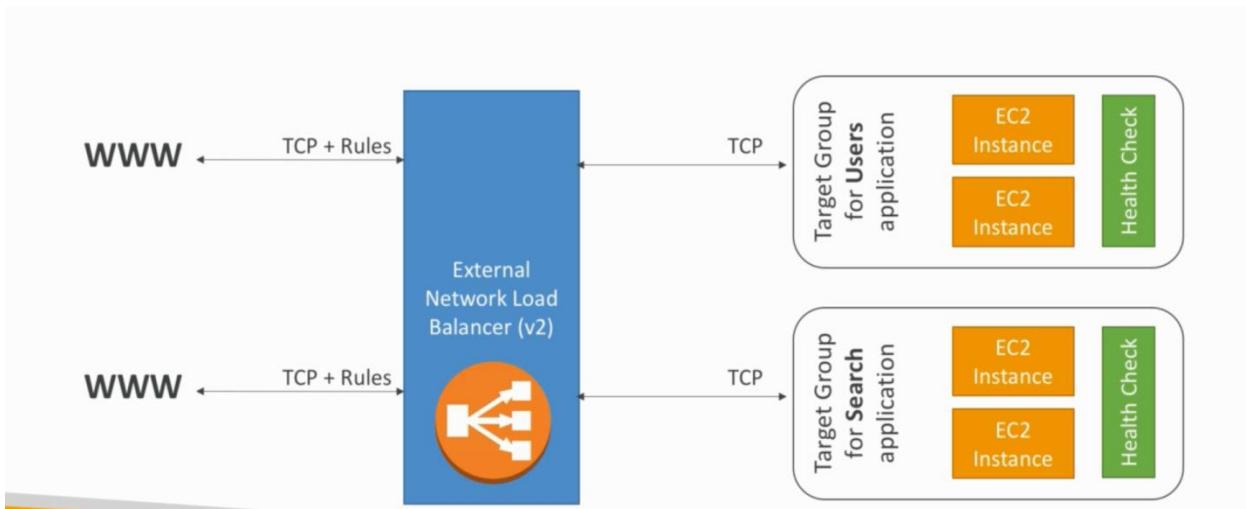
21. Go to EC2 under 'Services'.
22. Click on "Load Balancers".
23. Create a Classic Load Balancer.
24. Give it a name.
25. Add Port 80 to the security group.
26. Leave the default settings. Under configure health check, change the Ping Path to 'healthcheck.html' file.
 - Response Timeout – Time to wait when receiving a response from the health check. (2 sec to 60 sec). For now, take it to 5 sec.
 - Interval – Amount of time between health check (5 sec to 30 sec). For now, take it to 10 sec.
 - Unhealthy threshold – Number of consecutive health checks failures before declaring an EC2 instance unhealthy. At first, if the instance fails to launch, it tries to re-launch itself after 'Interval' time and again (if we set the threshold to 2) if it fails, the instance is declared as unhealthy.
 - Healthy threshold – Number of consecutive health checks success before declaring an EC2 instance healthy. For now, take it to 5.
27. Add Response Timeout, Interval, Healthy threshold & unhealthy threshold.
28. Add an EC2 instance to it & tags as ELB cost money, so add a tag to it so we can identify whether this ELB is in running state.
29. Review & create.
30. Now go to "Load Balancers" & under that go to 'Instance' Section, we can see that our instance status is in "In Service".
31. We can see that our EC2 instance is accessible from the public IP as well as from the Load Balancer. But, we want our instance to be only accessible from Load Balancer. So, go to Security groups under EC2 dashboard.
32. If we look at the inbound rules we can see our EC2 instance is open to PORT 80 and source is 0.0.0.0/0 which says it can be accessed from anyone from anywhere. So, we'll change it to the security group assigned to the Load Balancer. Hence, we can change its description too.
33. Hit Save.
34. Now, if we refresh the page which was accessed through EC2 load balancer, it times out.

- 35.Create 2 more similar EC2 instances one of the same AZ and other in different AZ within the same region (we can use the option "Launch more like this.")
- 36.Refresh the page (IP of the Load balancer). We can see the IP getting changed every time. It means our Load Balancer is getting responses from the 3 EC2 instances.
- 37.Go to Load Balancer and attach the newly created EC2 instance to Load Balancer.
- 38.Click on the security group created for the EC2 instance and we
- 39.Now, open Putty & remove the 'healthcheck.html' file by typing in the command `rm -rf healthcheck.html` as a Super User.
- 40.As soon as we remove the file, we can see the Instance status has been changed to "Out Of Service" (after 5+5+2 sec) coz it looks for the file `healthcheck.html`.
- 41.Let's add that file again `echo "I'm healthy ELB instance" > healthcheck.html`
- 42.Instances monitored by ELB are reported as InService, or OutOfService.
- 43.Once an EC2 instance is out of service, the ELB will not send traffic to EC2 instance coz it's failing the health check & presuming the EC2 instance has gone down.
- 44.Try to paste the DNS name of your Load Balancer in the browser, it points to our `index.html` file.
- 45.The difference between the ELB & EC2 instance is that we get a public IP for an EC2 instance but we don't get it for an ELB. There's a public IP for ELB but Amazon manages it for us, they wants to use the DNS name since the public IP address can change.

Network Load Balancers

- (Layer 4) - They're not included in the free-tier.
- It supports TCP, TLS (secure TCP) & UDP.
- It forwards TCP & UDP traffic to our instances.
- It allows millions of request per request.
- Latency is too low ~100ms (vs 400 ms for ALB).

- NLB has one static IP per AZ, and supports assigning Elastic IP (helpful for white-listing specific IP). The ALB or CLB does not have a static IP rather they've a static hostname.
- NLB are used for extreme performance, TCP or UDP traffic.



Our target group just like before can be EC2 instances but here TCP based target groups will reach our EC2 instances.

- a. It forwards TCP traffic to our instances.
- b. Handles millions of requests per second as they've high performance.
- c. They support for Static IP or Elastic IP
- d. They've less latency (approx. 100ms which is approx. 400ms for ALB) so they should not be the default load balancer we should choose.
- e. They can directly see the client IP.

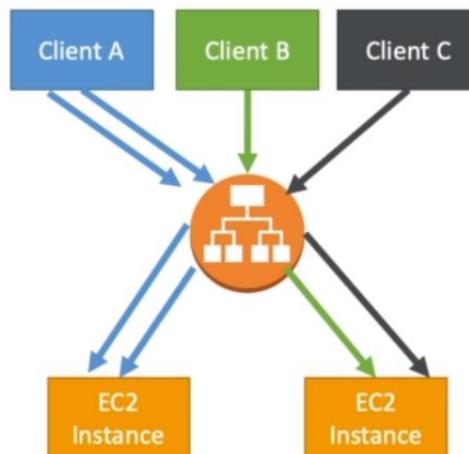
LAB

1. Create a Load Balancer as created for Application Load Balancer.
2. Now if we ping the URLs, we are not directed to the website. This is because the security groups attached to it has the inbound rule to allow traffic to port 80 from other port.
3. So go to either of the instance.
4. Click on the security group.
5. Click on inbound rules and edit.
6. Add a rule for Custom TCP on PORT 80, allowing traffic from anywhere (0.0.0.0/0).

- Now if we refresh, we can see the HTML page being displayed.

Load Balancer Stickiness

- It is possible to implement stickiness to Load Balancer so that the same client is always redirected to the same instance.
- This works for Classic Load Balancer & Application Load Balancer.
- Working: There's a cookie which is used for stickiness has a expiration date we control and as long as cookie remains the same, then the Load Balancer redirect to the right instance.
- Enabling stickiness may bring imbalance to the Load Balancer



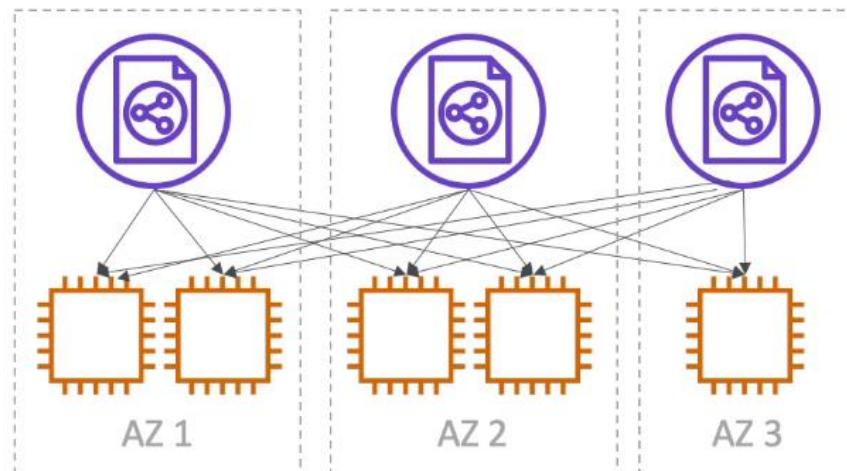
- Here is our Load Balancer & we've 2 EC2 instances. Our first EC2 instance talk to the Load Balancer and if the same client talks again to the Load Balancer with the stickiness enabled then EC2 instance will receive the same result. And similarly for Client B.

LAB

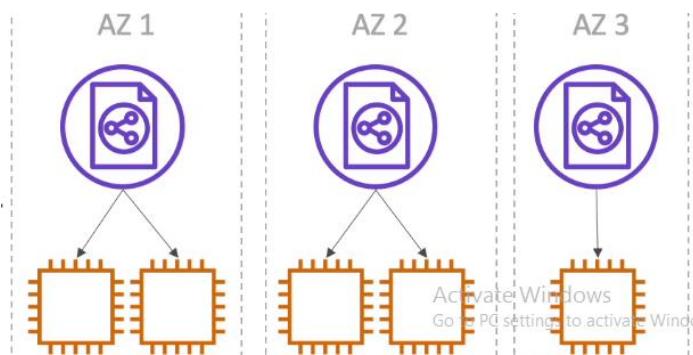
- Go to the target group attached to the Load balancer.
- Down below, we can find Stickiness section.
- Click on Edit attributes & set the Stickiness to be Enabled.
- Select the duration as 120 seconds for now.
- Now if we refresh our ALB IP, it does not change its IP, rather sticks to the same IP address.
- This will happen for 2 minutes i.e., unless our cookie expires and then it will target to our other underlying EC2 instances.

Cross Zone Load Balancing

- Each Load Balancer instance distributes evenly across all registered instances in all AZ.



- Let's suppose we've 3 availability zones having 5 EC2 instances. Consider, we've deployed our Load Balancer and it is deployed across the 3 availability zone. Hence through Cross Zone Load Balancer, our first instance in the first AZ will be distributing traffic to all the EC2 instances. And this happens with the other 2 instances as well. Hence, it is the best way to spread the load across all the instances.

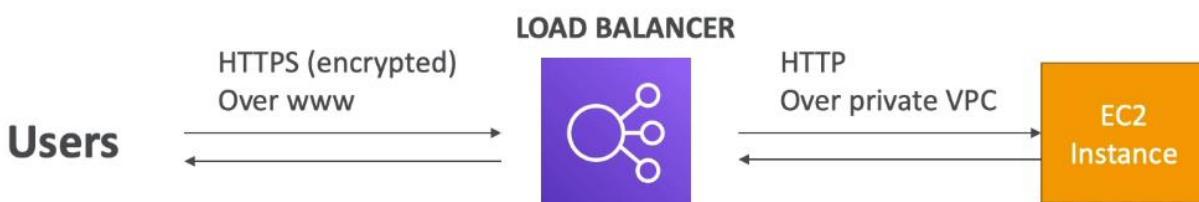


- If we don't have cross zone load balancing, then it is a different kind of Load Balancing which would work but not across regions.
- Classic Load Balancer
 - Disabled by default.

- No charges for inter AZ data if enabled. If our data goes from one AZ to another AZ, then we get charged for it. But if we enable it for CLB, then we'll get charged for it.
- Application Load Balancer
 - This is always on and it cannot be disabled.
 - No charges for inter AZ data.
- Network Load Balancer
 - Disabled by default.
 - You pay charges (\$) for inter AZ data if enabled.
- For the LAB section, we go to Load Balancer page & choose our Load Balancer. Under Description Tab, we can see cross-zone Load Balancing. We can change that setting & hit save. It will enable Load Balancing against all the EC2 instances that are attached to the Load Balancer.

SSL/TLS

1. An SSL Certificate allows traffic between our clients and our Load Balancer to be encrypted in transit (in-flight encryption).
2. SSL refers to secure socket layer, used to encrypt connections.
3. TLS refers to transport layer security, which is newer version of SSL.
4. Nowadays TLS certificates are mainly used but people still refer SSL.
5. Public SSL certificates are used by Certificates Authorities (CA), like GoDaddy, DigiCert, etc.
6. SSL certifications have an expiration date (which we can set) and must be renewed.



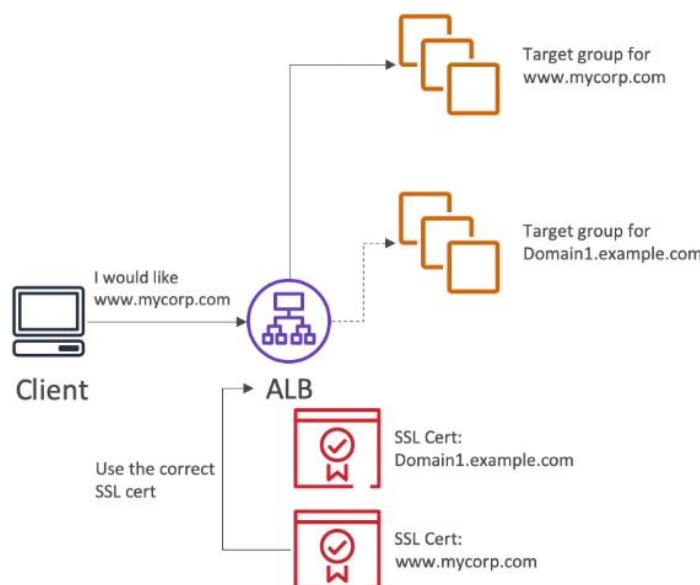
Users connect to HTTPS and it connects over public internet to the Load Balancer. Internally Load Balancer does SSL certificate termination and in the backend it talks to our EC2 instance using HTTP. The traffic goes to the VPC which is a private traffic which is somewhat secure.

- The Load balancer uses an X.509 certificates (SSL/TLS server certificate).
- We can manage these certificates using ACM (AWS Certificate manager).

- We can create/upload our own certificates alternatively.
- When we set a HTTPS Listener:
 - We must specify a default certificate.
 - We can add an optional list of certificates to support multiple domains.
 - Clients can use SNI (Server Name Indication) to specify the hostname they reach.
 - We can set a specific security policy if we wanted to support older version of SSL/TLS called legacy clients.

Server Name Indication (SNI)

- SNI resolves the problem of loading multiple SSL certificates onto one web server (to serve multiple websites).
- It is a newer protocol which requires the client to indicate the hostname of the target server in the initial handshake.
- The server will then find the correct certificate, or return the default one.
- It only works with ALB & NLB (newer generation), CloudFront.
- It does not work with Classic Load Balancer. So if we see multiple SSL certificates on our Load Balancer, we should think of ALB or NLB.



We've ALB and 2 target groups and ALB routes to these target group based on some rules and the rules may be directly linked like hostname. Hence ALB has

2 SSL certificates corresponding to the target groups. Now, client connects and says that he want to connect to www.mycorp.com and that is a part of Server Name indication and ALB says let me use the correct SSL certificate to fill that request. Hence, ALB takes the right SSL certificate and encrypts the traffic and the rules that it knows to redirect to correct target group.

ELB - SSL certificates

- Classic Load Balancer
 - Supports only one SSL certificate.
 - We need to use multiple CLB for multiple hostname with multiple SSL certificates.
- Application Load Balancer
 - Supports multiple listeners with multiple SSL certificates.
 - It uses SNI (Server Name Indication) to make it work.
- Network Load Balancer
 - Supports multiple listeners with multiple SSL certificates.
 - It uses SNI to make it work.

LAB

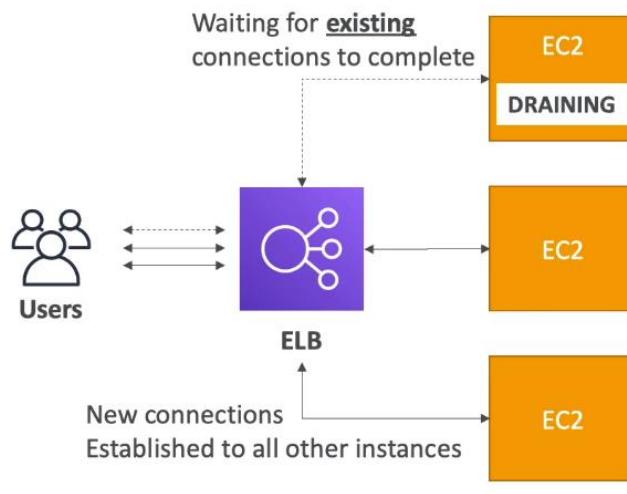
1. Go to EC2 under Services.
2. Click on any **Classic Load Balancer**.
3. Select a Load Balancer and navigate to Listeners Tab.
4. Edit an add a HTTPS listener.
5. Click on change under Cipher column which is the protocols we need to support.
6. Click on Change under SSL certificate and we can import it directly which can be manually encoded. Or we can choose the certificates from ACM.
7. Click on Save.
8. For **Application Load Balancer** click on Add under listener section.
9. Select the port to be HTTPS.
10. Select default action to be forwarding it to the target group.
11. Select a Security policy and a Default SSL certificate. Now we can have each rule or different SSL certificates and that'd allow us using server name indication, SNI to have multiple SSL certificates on different target groups.

12. For NLB, we can add a listener as the steps used for ALB.

13. This time we have TLS for secure TCP,

Connection Draining

- It has 2 different names based on which Load Balancer we're considering.
- Feature naming. If we use
 - CLB, then it's called connection draining.
 - Target Group, then it's called Deregistration Delay.
- It is the time to complete "in-flight request" while the instance is de-registering or unhealthy.
- As soon as the instance is in draining mode will stop sending new request.
- Suppose, we've an ELB and we've 3 EC2 instances behind it are our users are accessing first EC2 instance through the ELB. If it turns out that our EC2 instance is being terminated/unhealthy so it goes to draining mode. During the draining mode, the existing connection will be waited for the duration of the connection draining period to be completed (default is 300 seconds). If there's any new connection made by the users into the ELB to other EC2 instances available. The de-registration delay is 300 seconds but we can set the period in between 1 second n 3600 seconds. We can disable it setting the value to 0.



- Ex: We've a web application that does short request (between 1-5 seconds) and we set our de-registration delay to value such as 10, 20 seconds. But if our EC2 instances are slow to respond, it may take minutes as it's have to

do lots of data processing, then we need to set the connection draining a bit higher. And if we don't want to implement it, we can disable it and in that case if a connection is dropped while our EC2 instance is being killed then users retrieve an error and then users need to retry the request until that succeeds by being redirected to a new EC2 instance.

Command Line Interface (AWS CLI)

It is a tool that pulls all the service together in one central console, giving us easy control of multiple AWS services with a single tool.

AWS CLI Configurations



Our computer access the AWS network in our registered accounts using the CLI at the command Line Interface over World Wide Web. When we connect the computer to the CLI and the AWS accounts, it checks for credentials and permissions.

1. Download it from web browser & install it.
2. Run `aws --version` in CMD to confirm successful setup
3. Launch an EC2 instance with default settings.
4. Go to IAM & create a user.
5. Select Access Type as Programmatic Access.
6. Assign any existing policy & Create User.
7. Download .CSV file.
8. Login using CMD or Putty.
9. The File are stored in AWS directory.
10. To go to this directory, type `cd .aws`
11. Go to credentials file, type `nano credentials`.

10. To fetch the detail of ec2 instances running, run `aws ec2 describe-instances`. Using this command, we can even get the details of the terminated services.
11. To terminate an instances, run `aws terminate-instances--instance-ids InstanceID`

Note: It's always a better habit to delete the users if the related EC2 instances has been terminated. It adds a security risk to your instances. If we've multiple AWS account, & we want to access in parallel to the current account, then firstly configure it by typing: `aws configure --profile my-other-aws-account`.

When we check the credentials, we can see the credentials for 2 accounts.

Now to use it for e.g.:

- `aws s3 ls` executes for default profile.
- `aws s3 ls --profile my-other-aws-account` executes for the different profile .

AWS CLI Dry runs

Sometimes we may just need to test the policies, just to make sure we've the permissions. Also, some commands like creating an EC2 instances may become quite expensive when we run it. So some of these commands a --dry-run option and it simulate API calls. If the commands have the permission, they'll just not run the actual command & if there's no permission, we get denied access.

Sample command to check whether our EC2 instance is capable of starting other eC2 instance: `aws ec2 run-instances --dry-run --image-id (AMI id) --instance-type (Instance type like t2.micro)`.

If the commands succeeds, we get a message that Request would have succeeded but DryRun flag is set else we get an message that an error occurred while running the action.

AWS CLI STS Decode errors

When we run API calls and if they fail, we get a long error message. This messages can be decoded using STS command line & we need to run `sts decode-authorization-message` to decode the messages. To decode the aws message, run

```
aws sts decode-authorization-message --encoded-message  
(encoded message)
```

LAB

For Decode Authorization message to work,

1. we need to attach policy
2. choose the Service to be STS.
3. Under Actions tab, add DecodeAuthorizationMessage.
4. Review and Save.
5. Re-run the command to decode message.
6. We get a JSON content in response.
7. Either format the JSON to get a better view of the message or Format it in VS Code.

IAM Roles

Storing the credentials of an EC2 instance is a security risk. It goes hectic if we've thousands of EC2 instances and if all the credentials stored inside a single file gets hacked. Even if we get to know about it later, then we need to change the other 999 user's credentials. Hence, the use of ROLES comes to picture.

We can create roles for different AWS services, another AWS account, Web Identity (a social identity that an internet user establishes in other communities & website which stores data in Dynamo DB) or SAML federation (used to create temporary AWS credentials to provide access to AWS resources).

1. Create a Role for 'EC2' service under IAM.
2. Select policy 'AmazonS3FullAccess'
3. Give ROLE a name.
4. Now, launch an EC2 instance with IAM role as created in the above steps (all the settings to be default).
5. After the instance is launched, select the instance from the list & click on 'Actions'.
6. We can see in "Instance setting list", "Attach/Replace IAM Role". An EC2 instance can have only one IAM role attached per instance.
7. Now, if we login into user as root & type cd .aws and type in below commands:

```
[root@ip-172-31-6-34 ec2-user]# cd ~
[root@ip-172-31-6-34 ~]# ls
[root@ip-172-31-6-34 ~]# cd .aws
bash: cd: .aws: No such file or directory
[root@ip-172-31-6-34 ~]# aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]: eu-west-1
Default output format [None]:
[root@ip-172-31-6-34 ~]# cd .aws
[root@ip-172-31-6-34 .aws]# ls
config
[root@ip-172-31-6-34 .aws]#
```

8. We will not find the ‘credential’ file as we’ve create a ROLE.
9. The best practice to copy a bucket from a region to other is giving explicitly the region name `aws s3 cp --recursive s3://source_bucket_name /home/ec2-user --region eu-west-2`

Policy Simulator

- It is used to test policies based on users/groups/roles.
- To test your policy based on roles, choose any of the role created under your account.
- Under Service, select the service you want to simulate.
- Under Action, select the action like "GetObject".
- Click on Run Simulation.
- It responds us with either Allowed/Denied.

Bash Scripting

Bash shell script is a computer program written in Bash programming language. It is used to interact with the operating system. Any command that we run from a command line can be used in a bash script. In AWS, we’ll use Bash scripting when our EC2 instance first starts up.

1. Create a S3 bucket with an index.html file.
2. Create a role with S3FullAdminAccess
3. Go to Instances & create an instance with the role created in step 2.
4. In “Configure Instance” section, in the “Advance details” we’ll write our bash script:

Note: It starts with “#!” & then the path to the interpreter (it interprets our bash commands & runs them sequentially at root level when VM starts up).

The bash script says:

```
#!/bin/bash
```

```
yum update -y
```

Launch the EC2 instance.

Go to Putty, login using this instance's IP & run the command:

yum update -y, we'll notice & no package has been updated and the CLI says “No packages marked for update”.

```
aws s3 cp s3://bucket_name/file_name /var/www/html
```

5. Launch the instance with further default settings.

Instance Metadata

- It is the data about the instance that we can use to manage or configure the running instances. It allows EC2 instances to “learn about themselves” without using IAM Roles for that purpose. E.g.: if we run web servers for various small business, they all can use the same generic AMI & retrieve their content from the Amazon S3 bucket that we specify in the user data at launch. To add a new customer at any time, create a bucket for him, add their content & launch your AMI with the unique bucket name provided in your code in the user data.
- We can retrieve IAM role name from meta data, but we cannot retrieve the IAM policy).
- The **meta-data IP** is: <http://169.254.169.254>(This IP does not work from browser, it runs from our EC2 instances).
- To get meta-data (say AMI ID), user data, run from putty, curl <http://169.254.169.254/latest/meta-data/ami-id>
- We can get our meta-data using command:
curl<http://169.254.169.254/latest/meta-data/> (do not forget to add a '/' at end. When it does not end with a '/', it means there's more to it.)
- To write our IP address in a file, curl <http://169.254.169.254/latest/meta-data/public-ipv4>> mypublicip.html & if we run ls command then we can see

there a file exist named mypublicip.html and we can even see the data inside by running `nano mypublicip.html`

- We've not authorized with IAM role to get this information.
- <http://169.254.169.254/latest/meta-data/iam/security-credentials> returns the Role Name.
- <http://169.254.169.254/latest/meta-data/iam/security-credentials/RoleName> returns with the access key, Secret Access key and a token. These are short lived credentials and at end, there's an expiration date usually an hour. This is how our EC2 instance get temporary credentials through the IAM Role.

AWS Limits

- API rate limits
 - DescribeInstances API for EC2 has a limit of 100 calls per second.
 - GetObject on S3 has a limit of 5500 GET per second per prefix.
 - For Intermittent Error: we implement exponential backoff.
 - For Consistent Errors: Request an API throttling limit increase.
- Service Quotas (Service limits) - it describes how many resources we can run. For ex:
 - Running On-Demand standard Instances: we can run up to 1152 vCPUs.
 - We can request a service limit increase by opening a ticket.
 - We can request a service quota increase by using the Service Quotas API.

AWS SDK

It is used when we desire to perform actions on AWS not using CLI but using code, we use Amazon SDK. Official SDKs are:

1. Java
2. .NET
3. Node.js
4. PHP
5. Python (boto3/botocore)
6. Go

7. Ruby
8. C++

These SDK are really useful when we're coding against services such as DynamoDB. In fact, the AWS CLI uses Python SDK (boto3). If we don't define or configure default region, then us-east-1 will be chosen by default by our SDK to issue API calls.

[Security Credentials](#)

It is always recommended to use the **default credential provider chain**.

The CLI will look credentials in this order:

1. Command line options --region, --output, --profile.
2. These has a priority over the second place, it looks in the environment variables: AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, and AWS_SEESION_TOKEN. These will take precedence of we don't set any Command Line options.
3. Then we look into CLI credentials file - aws configure,
4. ~/.aws/credentials on Linux/Mac & C:/Users/user/.aws/ credentials on Windows.
5. It then looks in the container credentials like if we have an ECS task, then it'll look at the container credentials.
6. If we've EC2 Instance profiles, then it'll look at the EC2 profile credentials.

It works seamlessly with our

1. AWS credentials (~/.aws/credentials) at our computer or on premise.
2. Instance Profile credentials using IAM roles.
3. Environment variables.

Best practice is for credentials, is to use IAM roles if we are working from within AWS Services.

[Default credentials provider chain](#)

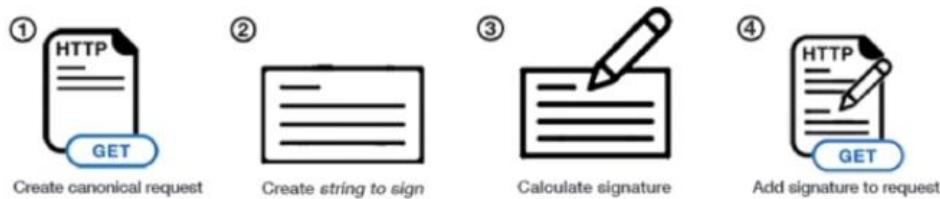
The Java SDK (example) will look credentials in this order:

1. Environment variables - AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY.
2. Java system properties - aws.accessKeyId and aws.secretKey.

3. The default credentials profiles file - ex at: `~/.aws/credentials`, shared by many SDK.
4. Amazon ECS container credentials - for ECS containers.
5. Instance profile credential - used on EC2 instance.

Signing AWS API request

- When we call AWS HTTP API, we sign the request, so that AWS can identify myself, using our AWS credentials (access key and secret key).
- Some of the request to Amazon S3 don't need to be signed.
- If we use the SDK or CLI, the HTTP requests are signed for us.



- We should sign an AWS HTTP request using Signature v4. (Sig v4)
- Examples:
- HTTP Header option

```
GET https://iam.amazonaws.com/?Action=ListUsers&Version=2010-05-08 HTTP/1.1
Authorization: AWS4-HMAC-SHA256 Credential=AKIDEXAMPLE/20150830/us-east-1/iam/aws4_request,
SignedHeaders=content-type;host;x-amz-date,
Signature=5d672d79c15b13162d9279b0855cfba6789a8edb4c82c400e06b5924a6f2b5d7
content-type: application/x-www-form-urlencoded; charset=utf-8
host: iam.amazonaws.com
x-amz-date: 20150830T123600Z
```

- Query String option (ex: S3 pre-signed URLs)

```
GET https://iam.amazonaws.com?Action=ListUsers&Version=2010-05-08&
X-Amz-Algorithm=AWS4-HMAC-SHA256&
X-Amz-Credential=AKIDEXAMPLE%2F20150830%2Fus-east-1%2Fiam%2Faws4_request&
X-Amz-Date=20150830T123600Z&X-Amz-Expires=60&X-Amz-SignedHeaders=content-type%3Bhost%3Bx-amz-date%3Bx-amz-signature
X-Amz-Signature=37ac2f4fde00b0ac9bd9eadeb459b1bbe224158d66e7ae5fcadb70b2d181d02 HTTP/1.1
content-type: application/x-www-form-urlencoded; charset=utf-8
host: iam.amazonaws.com
```

[Activate Windows](#)

Exponential Backoff

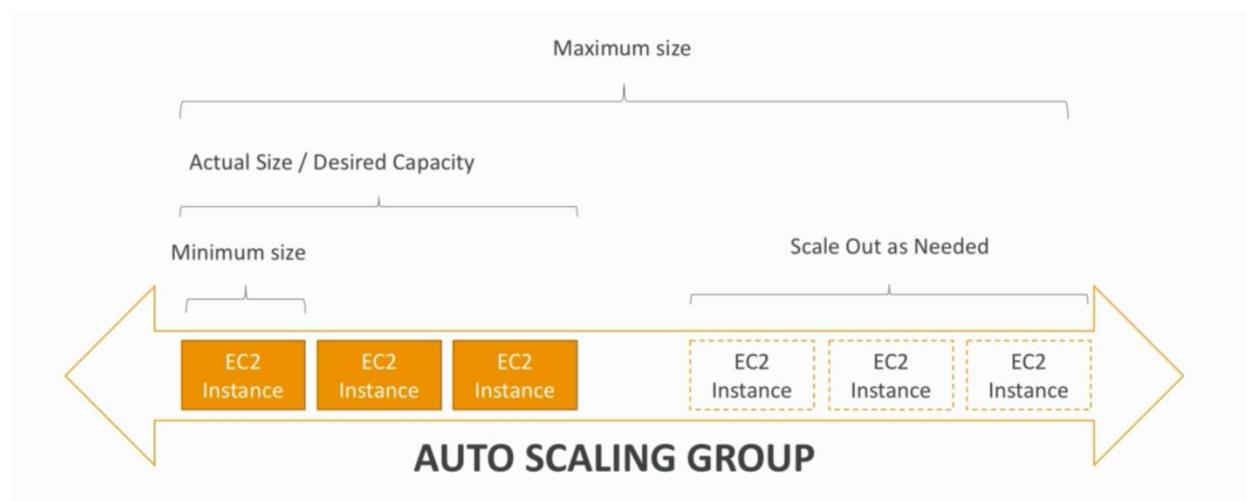
- If we get ThrottlingException intermittently, we use exponential backoff.
- It is a retry mechanism that is included in SDK API calls.
- If we're using the API by ourselves, without any SDK then we must implement it at our end.

- Any API that fails because of too many calls needs to be retried within Exponential Backoff. These are for rate limited API. After failure, our 1st API call waits suppose for 10ms, 2nd one waits for 20ms, other one at 40ms. If our API calls keep on failing, we'll wait twice as longer as previous call and these ensures, we don't overload our API by trying every milliseconds.

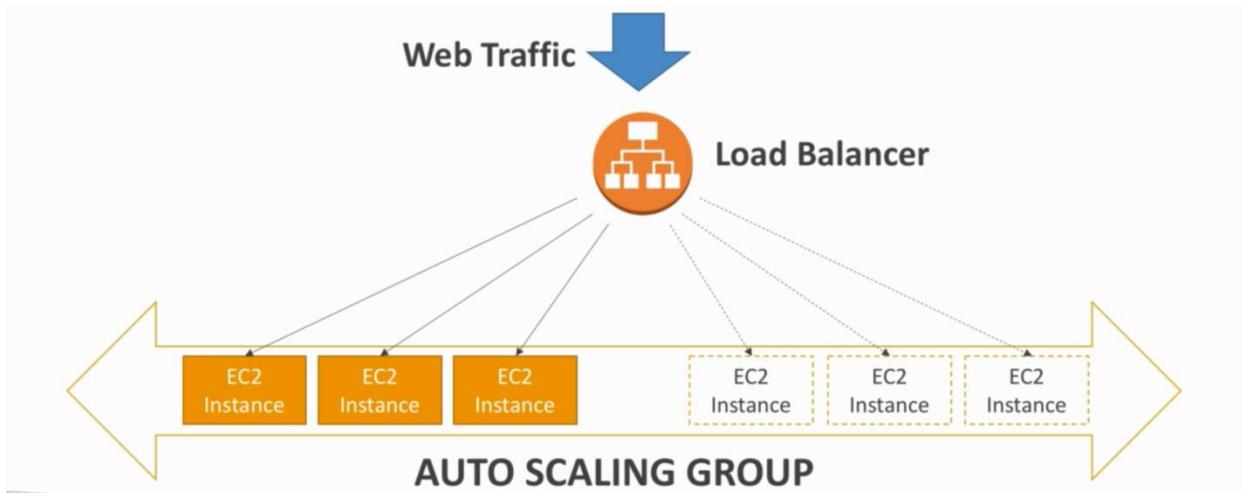
Auto scaling Groups

In real life, the load on our website & application can change. The goal of auto-scaling group is to:

- Scale out (add EC2 instances) to match an increase load.
- Scale in (remove EC2 instances) to match the decrease load.
- Ensure we've minimum & maximum number of machines running.
- Automatically register new instances to Load balancer.



We've a minimum size which we'll have for sure running into this auto scaling group. We've a desired size which is the current number of instances running & maximum size is the number of instances can be added to scale out if needed when those load goes up.



We've a load balancer and an auto scaling group & load balancers knows how to connect to these ASG (Auto scaling group) instances. If our ASG scales out, so if we add EC2 instances then Load balancer automatically register these targets (EC2 instances) & perform health checks & directly reach traffic back to them. This way, Load Balancer & auto scaling group work hand in hand.

ASG has following attributes:

1. A launch configuration
 - a. AMI + Instance type
 - b. EC2 user data
 - c. EBS volumes
 - d. Security Groups
 - e. SSH Key Pair
2. Min size/Max size/Initial Capacity
3. Network + Subnet Information
4. Load Balancer Information
5. Scaling policies

Auto Scaling Alarms

It is possible to scale an ASG based on CloudWatch alarms. The alarm is based on some metrics. When the metrics goes up, it alarms us to scale out i.e., to increase the instances & vice-versa. We can define the metrics on the basis of:

1. Target average CPU usage.
2. Number of requests on the ELB per instance.

3. Average network in &
4. Average network out.

We can also scale on the basis of custom metric. (Ex: number of connected users).

To do this,

1. We'll create custom metric from our application on EC2 & send it to CloudWatch using the PutMetric API.
2. Create CloudWatch alarm to react to high/low values to that metric.
3. These alarm trigger the scaling policy for ASG.

Points to remember:

1. Scaling policies can be based on CPU network and can be even on custom metrics or based on schedule (if we know our visitor pattern that there's load at 9:00 A.M in the morning).
2. ASG uses Launch configuration or Launch templates (newer).
3. If we update an ASG, we need to define new launch configuration/launch template.
4. If IAM roles are attached to ASG, it gets automatically assigned to EC2 instances.
5. ASG are free. We only need to pay for the underlying resource being launched.
6. If we've instances running under ASG & if they get terminated for whatever reason, ASG restarts them automatically. Extra safety!
7. ASG can terminate instance marked as unhealthy by a Load Balancer.

LAB

1. Create a basic html file & upload it to a bucket (e.g.: technoronicks3staticwebsite).
2. Go to Launch Configuration on EC2 Services list under Auto-Scaling section. We can choose a launch template or a launch configuration (Launch template is the newer version while Launch configuration is the older version. Launch template allow us to use a spot fleet of instances and launch configuration allow us to use just one instance type).
3. Give a name to the Auto Scaling group.
4. Create a launch configuration with default settings & bash script.

5. Now click on the button “Create an Auto Scaling using this launch configuration”
 6. Assign a name to this group, with the number of instances we wanna start with. E.g.: if we assigned 3 instances and if there are 3 subnets (availability zones) in that region, then each instance will be launched in one of the subnets.
 7. While configuring the auto-scaling policies, we can choose between “Keep this group at its initial size” or “Use scaling policies to adjust the capacity of the group”.
 8. If we choose “Use scaling policies to adjust the capacity of the group”, we get an option to:
 - a. Scale out instances between n numbers of instances.
 - b. Choose the metric type (condition on which our instances would be scaled) & its target value.
 - c. Set the time taken by the instance to warm-up.
- OR
- d. We can select “Scale the Auto Scaling group using step or simple scaling policies”
 - e. In this, we can further edit when to add or remove (scale out or scale in) the instance.
9. Health Check Grace Period is the length of time an auto scaling waits before checking the instance’s health status.
 10. Launch the group with further default settings.
 11. For a Launch template, Click on "Create a launch template".
 12. Give it a name.
 13. Choose the type of instance and the type of key pair.
 14. Choose Networking platform to be VPC for now & a security group.
 15. Leave other settings as default and in user data use the previously used user data.
 16. Hit on create a Launch template.
 17. Refresh the Launch template section in ASG page and add the newly created launch template.
 18. Click on Next.

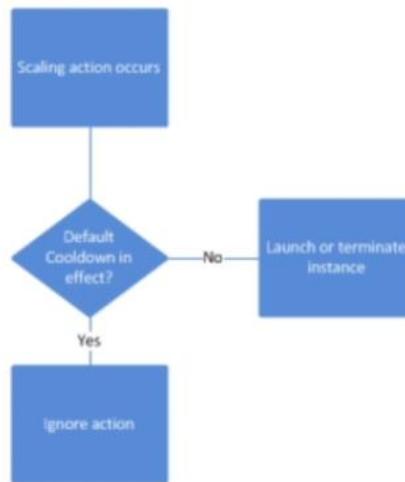
- 19.Under purchase option, we have option of choosing in between the On-Demand or combination of both of them (Combination is helpful when we've a hybrid fleet).
- 20.Select all the 3 subnets to launch the EC2 instance.
- 21.Click on Next.
- 22.Enable the Load Balancing.
- 23.Select the target group.
- 24.We've 2 types of Health Checks:
 - a. EC2 health checks - it means if the EC2 instance fails, then it'll be replaced.
 - b. ELB health checks - it means if the ELB health checks from the target group does not pass then the application of Auto Scaling group will terminate that instance and recreate a new one.
- 25.Click on Next.
- 26.Click on Desired capacity to be one, minimum to be one & maximum capacity to be 3.
- 27.Click on Next and other settings to be default.
- 28.Review and create.

Scaling Policies

1. Target tracking Scaling
 - a. Most simple and easy to setup.
 - b. Example: I want the average ASG CPU to stay around 40%, it means if we're over that instances then it'll provision more instances and if we're under that CPU, it'll start scaling in i.e., terminating the instances.
2. Simple/Step scaling
 - a. We setup a CloudWatch Alarm and when it is triggered (ex: CPU>70%), then add another unit or a couple of them.
 - b. We setup a CloudWatch Alarm and when it is triggered (ex: CPU<30%), then remove another unit or a couple of them.
3. Scheduled Actions
 - a. We anticipate a scaling based on usage pattern.
 - b. Ex: Increasing the min capacity to 10 at 5 P.M on Fridays.

Scaling Cooldowns

1. The cooldown period helps to ensure that Auto Scaling group does not launch or terminate additional instances before the previous scaling takes effect.
2. In addition to default cooldown for ASG, we can create cooldowns that apply to a specific scaling policy.
3. Scaling specific cooldown period overrides the default cooldown period.
4. Common use of scaling specific cooldown is with a scale-in policy: A policy that terminates instances based on specific criteria or metric. Because it terminates instances, Amazon EC2 needs less time to determine whether to terminate additional instances.
5. If the default cooling period of 300 seconds is too long - we can reduce costs by applying a scaling-specific cooldown period of 180 seconds to the scale-in policy to terminate instances a bit faster.
6. If our application is scaling up & down multiple times each hour, we need to make sure to modify the ASG cool-down timers and the CloudWatch Alarm Period that triggers the scale in.



7. The flowchart says is there scaling action occurring, if we're within the cooldown period then we can ignore the action & if not we can launch or terminate instances.

LAB

1. Got to the your ASG under EC2 Console.

2. Navigate to auto-scaling tab. Here we can define either a scaling policy or a scheduled action.
3. Click on Actions under **Scaling Policy** and click on Add Policy.
4. For now, select the policy type to be **Target Tracking Scaling**.
5. Set a Scaling policy name.
6. Choose the Metric type to be Average CPU utilization.
7. Set a target value lets be 50.
8. Set the cooldown to be 300 (by default).
9. We can also disable scale-in, if we just want to create a scale out policy.
10. Hit on Create.
11. In ASG, select the desired capacity to 2 & update.
12. Now, on basis of scaling policy since the CPU utilization is less than 50, so it will terminate one of the instance.
13. Now, choose **Scaling policy to be step scaling type**.
14. Give it a name.
15. Create a CloudWatch alarm and if it triggers, then add/remove/set to a capacity units of 'n' or % of group.
16. Also, add the time which it'll wait for another scaling activity.
17. For **Scheduled actions**, click on Create Scheduled actions and give it a name.
18. Select the date & time.
19. Set the Desired, Minimum & Max capacity.
20. Hit on Create.

Placement Groups

It is a logical grouping of instances within a single availability zone. Using placement group enables application to participate in a low latency, 10 GPBS network. Placement groups are recommended for application that benefit from low latency, high network throughput, or both. We use places like where we need Grid Computing or we are working on Cassandra as backend database (where high network throughput is required or low latency between different Cassandra nodes).

1. Placement group can't be created at multiple availability zones.
2. The name we specify for a placement group must be unique within AWS account.

3. Only certain types of instances can be launched in a placement group.
(Compute optimized, GPU, Memory Optimized, Storage Optimized)
4. AWS recommend homogenous (instances having same size & same family) instances within placement groups.
5. Placements groups can't be merged.
6. You can't move an existing instance into a placement group. We can create AMI from an existing group & then launch new instance from the AMI into a placement group.

Cloud History

AWS is infrastructure as a code. We can provision our VM anywhere on the globe.

IAAS

It runs on a physical machine (to us its virtual machines), so we've to manage the OS. If the disk corrupts, we need to reinstall OS

PAAS

In amazon world it's Elastic Beanstalk. We just upload our code & Amazon take a look at your code & provision underlying resources for us & create web service for us.

Containers

They're isolated & light weight. We need to deploy them to a server & we need to worry the container running & scaling them to response to load.

Lambda

Theory

They are serverless. We don't need to manage our data centre nor IAAS, PAAS or Containers. We don't have to worry about OS, patching, scaling, etc. We just need to worry about our code. Take your code, upload it to Lambda, then configure & event triggers which is going to trigger our Lambda functions. So Lambda encapsulates:

1. Data centre
2. Hardware
3. Assembly Code/ Protocols

4. High Level languages
5. Operating system
6. Application Layer/AWS APIs

- Lambda event can trigger another Lambda event.
- Lambda functions are independent, 1 event = 1 function.
- Architectures can get extremely complicated, AWS X-Rays can allow you to debug what is happening.
- Lambda can do things globally, we can use it to backup S3 buckets to other S3 buckets.
- It scales automatically. We can use Lambda in following ways:
 1. As an event driven compute service where AWS runs our code in response to events. These events could be changes to data in Amazon S3 buckets or an Amazon Dynamo DB table.
 2. As a compute service to run our code in response to HTTP request using Amazon API gateway or API calls using AWS SDKs.

Pricing of Lambda function is based on

1. Number of requests: First 1 million requests are free & thereafter \$0.20 is charged per request.
2. Duration: It is calculated on the basis from the time your code begins to execute until it returns or otherwise terminates, rounded up to nearest 100ms.
3. Memory: It depends on the memory you allocate to your function. We are charged \$0.00001667 for every GB-second used.

Note: The maximum duration to execute your function is 5 minutes. If function goes above that period then we need to break that function into smaller one.

LAB

1. Go to your console.
2. Go to Lambda under Compute.
3. Click on Create Function.

4. Click on “Author from scratch”
5. Give the function a name & select its runtime language.
6. Create a new role from AWS policy templates giving it a name & selecting a policy template.
7. Go to your text editor & copy the code:

```
def lambda_handler(event, context):
    # TODO implement
    print("In lambda handler")
    resp = {
        'statusCode': 200,
        'headers': {
            'Access-Control-Allow-Origin':
            "*",
        },
        'body': 'Raunak Rhishabh'
    }

    return resp
```

8. The triggers are: API gateway, AWS IoT, Alexa Skill Sets, Application Load balancer, CloudWatch Events, CloudWatch Logs, Code Commit, Cognito Sync Trigger, Dynamo DB, Kinesis, S3, SNS, and SQS.
9. Add API Gateway as trigger with default settings with security as IAM.

7. DNS

- It is used to convert human friendly domain names such as (<http://wikipedia.org>) to Internet Protocol (such as <http://124.25.36.58>).
- The IPv4 is a 32 bit field & has over 4 billion different addresses.
- IPv6 was created to solve this issue & has an address of 128 bits. (340 undecillion or 340×10^{36}).
- The last word in the domain name represents the ‘top level domain’, e.g.: .com, .gov. The second word in the domain name represents “second level domain name”, e.g.: .co.uk.

- Since all the Domain name needs to be unique, there needs to be a way to be organized so that domain names are not duplicated. This is where domain name registrar comes in. A registrar is an authority who can assign domain names directly under one or more top level domains. These domains are registered with InterNIC, a service of ICANN, which enforces uniqueness of domain names across the internet. Each database is registered in central database known as WhoIS database.
- In AWS, the most common records are:
 - A: URL to IPv4
 - AAAA: URL to IPv6.
 - CNAME: URL to URL
 - Alias: URL to AWS resource.

SOA records

An SOA record is Start of Authority record. It is a type of resource record in the Domain Name System containing administrative information about the zone, especially regarding zone transfers.

It stores information about:

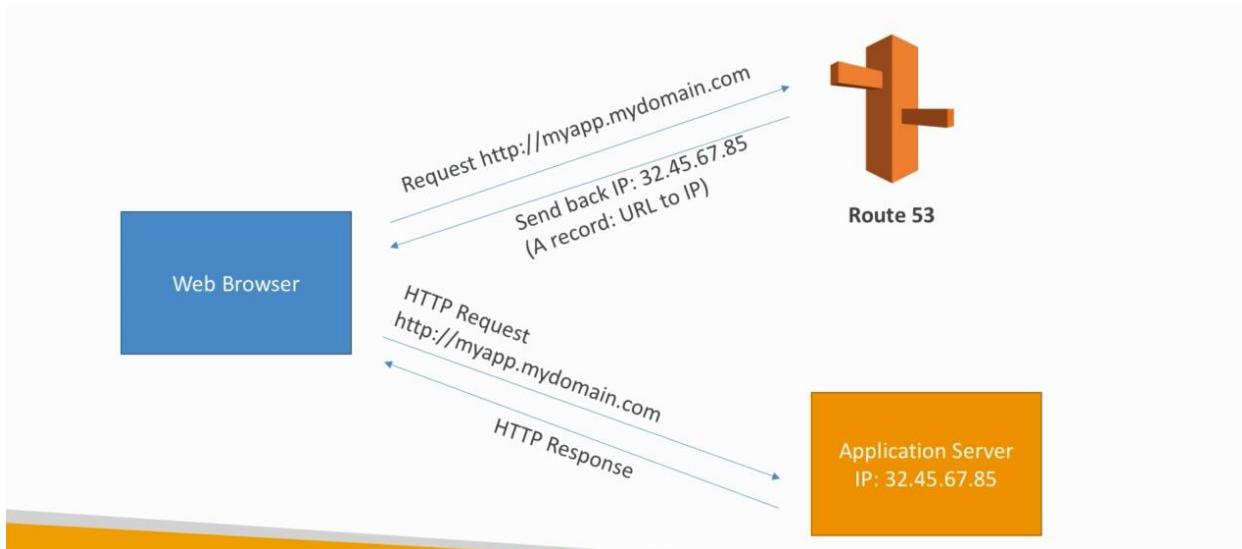
1. The name of the server that supplied the data for the zone.
2. The administrator of the zone.
3. The current version of the data file.
4. The number of seconds the secondary name server would wait before checking for updates.
5. The number of seconds the secondary name server should wait before retrying a failed zone transfer.
6. The max number of seconds that a secondary sever can use data before it must be either refreshed or expired.
7. The default number of second for the time to live on resource records.

NS records

It stands for name server records & are used by top level domain name servers to direct traffic to content DNS servers which contains authoritative DNS records.

A records

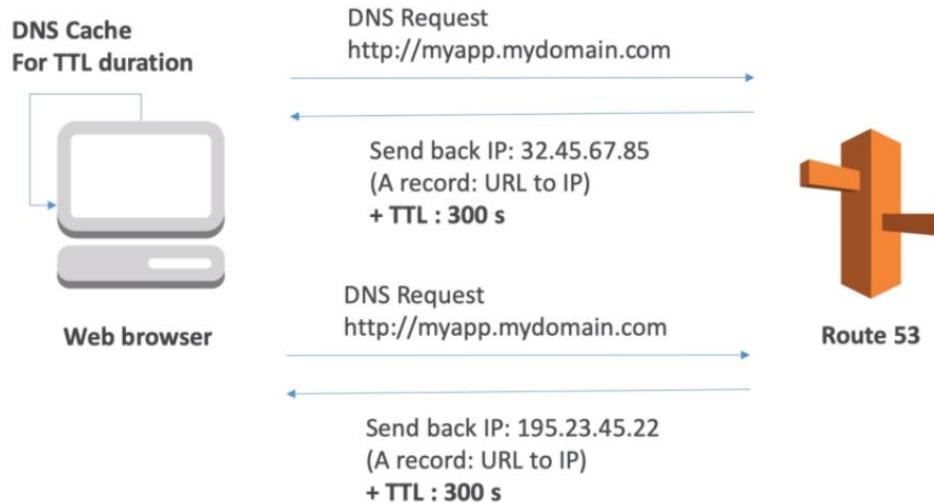
An ‘A’ record is the fundamental type of DNS record & the “A” in the record stands for “Address”. The A record is used by the computer to translate the name of the domain to IP address. For e.g.: <http://www.google.com> to <http://35.266.47.552>. Elastic Load Balancer never has a IPv4 or IPv6 address. They just have a DNS name.



Let's suppose we want to access an application server using web browser but we don't know the IP. So we use Route53 to browse to myapp.mydomain.com. Route53 says that since it's an A record, so it maps the URL to an IP & sends back the IP: 32.45.67.85. Now, the web browser makes the HTTP request using the same Domain name but this time it goes directly to the application server & the application server sends back the HTTP response. For the next time, the IP address is cached so my web browser directly to Application server.

TTL

It is a way for web browsers and clients to cache the response of a DNS query so that our DNS does not get overloaded.



We're going to make a DNS request from `myapp.mydomaain.com`, then Route53 sends back the IP and TTL. The web browser will cache the DNS request and the response for the TTL Duration and as soon as see the request, it is going to be valid for 300 seconds (we can set this value as per our choice). It means if we make a DNS request in between next 300 seconds, then the web browser will look internally in Cache and will not ask Route53 again. After this TTL happens, and if something changes on Route53 side (like IP) then again our cache will be updated. Also, if anything changes on Route53 DNS record, it does not means all the clients will see the change right away. High TTL can be like 24 hours where there's Less traffic on DNS & we get Possibly outdated records and low TTL is fir case having high traffic on DNS and there the record are outdated for less time. Also, it will be easy to change the records. **It is mandatory for each DNS to specify a TTL.**

The length that DNS record is cached on either the Resolving Server or the users own local machine equals the value of "Time to Live" in seconds. The lower the time to live, the faster the DNS records changes to propagate throughout the internet.

LAB

1. Go to Route53 under Services.
2. Navigate to hosted zones and create a Record sets.
3. Give it a name.
4. Select the type to be A.
5. Set the TTL value to be 120 seconds for now.
6. In the value section, provide the public IP of the EC2 instance.

7. Hit on Create.
8. Go to command prompt.
9. Type nslookup www.yourDNSname.com. It returns back the Server, Name, Address.
10. After 120 seconds, the Address will be updated to a new one.
11. In between 120 seconds time period, update the Value section with a new IP and in the meantime type nslookup www.yourDNSname.com. It returns back the same Server, Name, Address.
- 12.

C Names

If we've an AWS Resource (it can be a Load Balancer or CloudFront), it'll expose a AWS hostname. Like if we've a Load Balancer (lbl-1234.us-east-2.elb.amazonaws.com) and we want to expose our application as myapp.mydomain.com) and it should point to our Load Balancer.

A Canonical name can be used to resolve one domain name to another. For e.g.: If we've mobile website having domain name <http://m.acloudguru.com> that is used for when users browse to our domain name on their mobile devices. We may also want the name <http://mobile.acloudguru.com> to resolve the same address. It points a hostname to an any other hostname. It is only for non-root domain name. (abc.com is a root domain name and mail.abc.com is not a root domain name). Any domain name is a root domain name under which we can further create sub-domain and organize our file structure.

Alias records

- It points a hostname to an AWS resource like app.domain.com to blabla.amazonaws.com).
- It works for Root domain and non-root domain.
- It is used to map resources record set in your hosted zone to Elastic Load Balancers, CloudFront distributions, or S3 buckets that're configured as websites. Alias record work like CNAME record in that you can map one Domain name to another target DNS name.

- A CNAME can't be used for naked domain names (zone apex records). We can't have a CNAME for <http://acloudguru.com> it must be a record or an Alias.
 - It can save our time because Amazon Route 53 automatically recognizes changes in the record set because the Alias record set refers to. For ex: If an alias record set points to an ELB at lb1-1234.us-east-1.elb.amazonaws.com. If the IP address of the load balancer changes, Amazon Route 53 will automatically reflect the changes in their DNS answers for example.com without any changes to the hosted zone that contain resource record sets for example.com.
-
- ELB do not have Pre-defined IPv4 address, we need to resolve them using a DNS name. We are given a DNS endpoint.
 - When we make a request to Route 53 for a DNS record, we're going to be charged for that DNS record for that CNAME. If we are going to make request in its Alias record, we won't be charged as Alias record are free of charge.
 - They have capability of native health checks.
 - Given the choice, always choose Alias record over CNAME.

LAB

1. Go to Route53 under Services.
2. Navigate to hosted zones and create a Record sets.
3. Give it a name.
4. Select type to be CNAME type.
5. Don't choose to be Alias record type.
6. Copy the domain name of the Load Balancer and Paste it under the value section.
7. Hit on Create.
8. If we navigate to the URL using the domain name created in Step3, we'll be redirected to the DNS of the Load Balancer.
9. Now, since a Load Balancer is an Amazon resource, so we can use the Alias Record and use it AWS more efficiently.
10. Click on Create Record Set.

11. Give it a name.
12. Select type to be A (Alias record).
13. Choose Alias to be Yes.
14. Copy the DNS of the Load balancer under Alias Target.
15. Click on Create.
16. We can evaluate the Target health and it balances Load Balancer health checks.
17. If we leave the Name section to be blank and create a CNAME record, it gives an error saying Set of type CNAME with DNS Name is not permitted in the Apex zone. So, the only way to do it is using Alias Record and this time it does not give error.

8. Route 53

- It is a managed DNS (Domain Name system).
- It is a scalable and highly available Domain Name System Service.
- It intends to give business & developers a reliable way to direct end users to applications or basically.
- Route53 can use:
 - Public domain names that we own.
 - Private domain name that can be resolved by our instances.
- It is a collection of rules & records which help the clients understand how to reach the servers through Domain Name. The different routing policy are:
 - a. Simple: It is the default routing policy when you create a new record set. This is the most commonly used when you've a single resource that performs a given function for our domain. E.g.: one web server that serves content for the <http://aclouguru.com> website.
 - b. Weighted
 - c. Latency
 - d. Failover
 - e. Geolocation
- Route53 has advanced features such as:
 - Load Balancing (through DNS - aka Client Load Balancing).
 - Health checks

- Routing policy, such as Simple, Failover, Geolocation, Latency, Weighted, Multi-value.
- It is a global service and does not require any region selection.
- We need to pay \$0.50 per month per Hosted zone.

LAB

1. Go to Networking & Content Delivery under Services.
2. Click on Route 53.
3. Click on “Get started” under Register domains.
4. Choose a domain name & save the contact details for the Domain.
5. Accept the terms & conditions and hit on complete purchase.
6. After the registration process is complete, click on Hosted zones and click on your Domain name, we can see the NS records & SOA records.
7. Create 2 EC2 instances in different regions with default settings (providing PORT 22 & 80 in the security group), just provide bash script in “Configure Instance “section.
8. Create a Application Load Balancer with Ping path set to ‘/index.html’, interval equals 10 second, Unhealthy threshold equals 2 & Healthy threshold equals 3.
9. Add the EC2 instance & review and create the ELB.
10. Go to Load Balancer & ping the DNS name at a fast rate, we can see that ELB balances both the instances equally.
11. Similarly, create an instance in another region (mostly a region far away from the previous one created in step 7) & add a Load Balancer to it.
12. Go to Route53 under Networking & Content Delivery services.
13. Click on your hosted zone & then click on your Domain name.
14. By default there will be 2 record sets created.
15. Click on “Create Record Set”.
16. Give a name, select type to be IPv4 address, select Alias to ‘Yes’ & select one the ELB. For now, do not click on the Evaluate Target health.
17. A new ‘A’ record will be created at it points to DNS name of the Elastic Load Balancer.
18. Note: DNS of ELB does not have IPv4 address. We can only resolve an ELB by going to its DNS name.

19. Here (Simple Route53 is), hellocloudguru.com is resolving 2 EC2 instances that are behind ELB.
20. Go to command prompt.
21. Type nslookup www.yourDNSname.com. It returns back the Server, Name, Address.

Simple

- It is the default routing policy when you create a new record set.
- This is the most commonly used when you've a single resource that performs a given function for our domain.
- We've a web browser and we request a DNS name. Route53 replies that it is an A record and the IP is 11.22.33.44. so, we use it basically to redirect to a single resource.
- We cannot attach health checks to check simple routing policy.
- If multiple values are returned, a random one is chosen by the client.

LAB

1. Go to create Records under Hosted zone.
2. Create a new A type record.
3. Enter an IP address of the EC2 instance.
4. Choose Routing policy to be Simple.
5. Hit "Save Record Set".
6. Now if we browse the URL (provided under Name section), it redirects to the EC2 instance's IP.
7. For little bit more complexity, add another IP in the Value section separated by a line. When the web browser will query for the URL (provided under Name section), it'll receive 2 addresses back and then browser choose which IP to go. It is called client side load balancing which is done by the web browser.
8. Now if we browse the IP, we'll get response from one of the EC2 instance for the time period we've set under TTL.

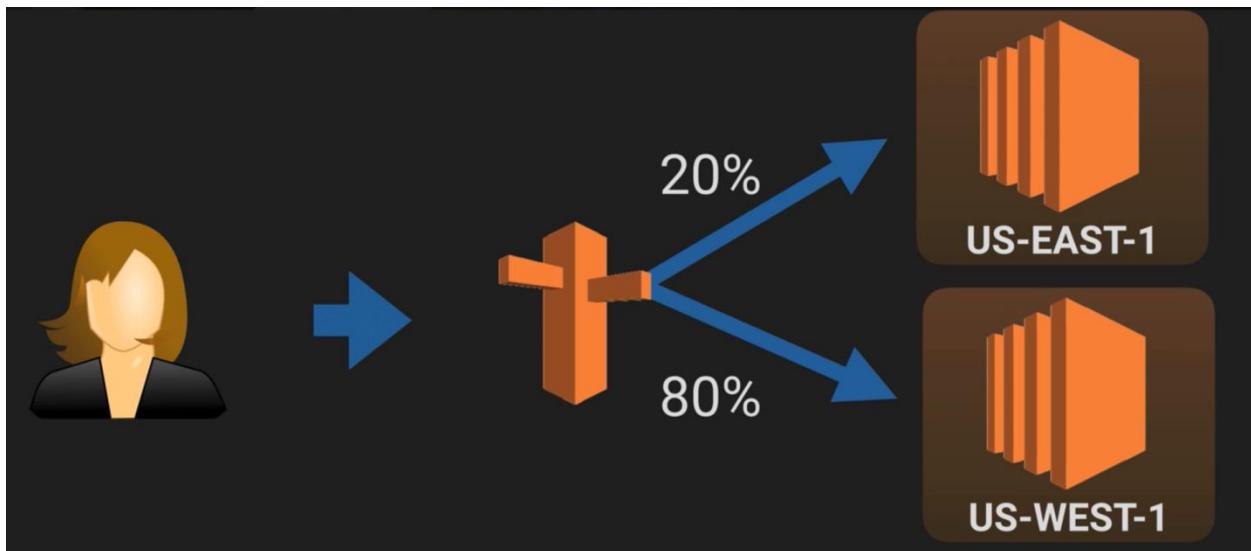
Weighted Routing

- Weighted routing policy allow us to split our traffic based on different weights assigned. It controls the % of the request that'll go to specific endpoint.
- If a user goes to a URL, it is going to resolve that DNS name to Route53 and Route53 can configure waiting & it can direct a part of traffic to one region & other part to different region.
- Helpful to test 1% of traffic on new app version.
- Helpful to split traffic between 2 regions.
- Can be associated with health checks, so if an EC2 instance is not working properly, no traffic will be directed towards it.

For understanding, let's take a scenario. If we know that from ABC region we've more sales coming & from region XYZ the sale is only 20%. So we need to optimize the balancers accordingly. If we want less traffic to our testing website & more focus on production website, so we can configure accordingly.

It is not necessary that we've to select two different regions. We can select same region too.

Route53 is a Global Service i.e., it does not matter whether we are in Sydney or Mumbai.

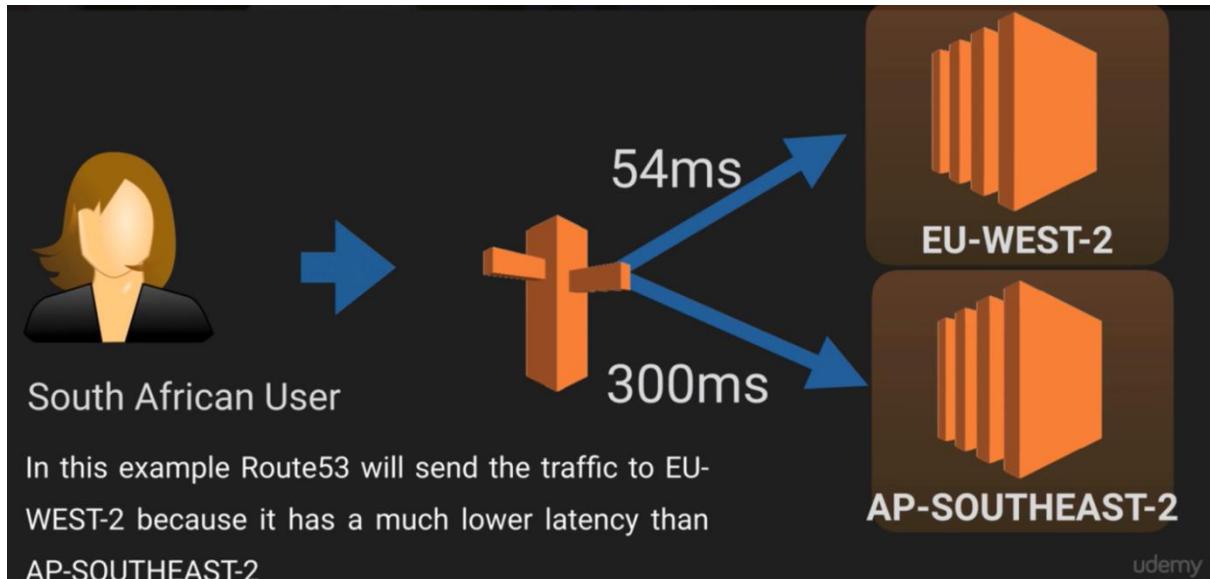


LAB

1. Go to Route53 under “Network & Content Delivery” services.
2. Click on your Hosted Zone & DNS name.
3. Click on create Record Set.
4. Select type to be A type.
5. Give it a name.
6. Set Alias to ‘No’.
7. Provide the IP of the EC2 instance.
8. Select Routing Policy to be weighted.
9. Decide the weight (in numeric to your Alias Target set in step 6) & give it a Set ID.
10. For now, do not associate with the Health check.
11. Now, create another record set in another region (Make sure we enter the same name while creating Record Set).
12. Provide a different IP under Value section.
13. Select Routing policy to be weighted and Weight to 30 & give it a Set ID.
14. Ping the ELB DNS name & we’ll see that for most of the time, the site routes the website having higher weight. The IP refreshes only after we have set TTL period.

Latency

- Latency based traffic allows us to route our traffic based on the lowest network latency for our end users i.e., which region is going to give them the fastest response time.
- It is helpful when Latency is a priority for the users.
- Latency is going to be evaluated in terms of user to designated AWS Region which means users of Germany may be directed to US if that is the lowest latency.



To use latency based routing, we create a latency resource record set for the Amazon EC2 (or ELB) resource in each region that hosts our website. When Amazon Route 53 receives a query for our site, it selects a latency resource record set for the region that gives the user the lowest latency. Route 53 then responds with the value associated with that record set.

In this example, Route53 will send the traffic to EU-WEST-2 because it has much lower latency than AP-Southeast-2.



Let's take another example. We have different users in different parts of the world. Also, there's 2 different instances located, one in US & another in Sydney.

So based on Latency routing policy, the 4 users on the Left hand side will go to US while the other 2 will be directed to Australia.

LAB

1. Go to Route53.
2. Go to your Hosted zone.
3. Create a new record.
4. Provide it with an IP of an instance.
5. Select Alias to 'Yes' & Routing Policy to 'Latency'.
6. Select the region where that IP belonged to & hit 'Create'.
7. Similarly, create another record in a different region with Routing Policy set to 'Latency'.
8. Now go to your DNS name, one directly to Mumbai region & other using VPN of that region selected while configuring the Latency. Difference of the latency between two regions can be easily noticed.

Health Checks

If an instance is unhealthy, Route 53 will not send traffic to that EC2 instance.

- An instance is termed unhealthy if it fails 3 health checks in a row and termed healthy if it passes 3 health checks in a row.
- The default health check is 30 seconds. But faster health checks can be set up (10 seconds) but at a higher cost.
- AWS launches 15 health checker in the background, which will check the endpoint health.
- If we calculate using the above 2 points, we would get a request every 2 seconds.
- We can have HTTP, TCP, and HTTPS health checks. When we use HTTPS health check, we will not get SSL certificate verification.
- We can integrate these health checks with CloudWatch if we wanted to.
- Once we have these health checks defined in Route 53, they can be directly linked to Record sets and DNS Queries and it'll change the behavior of Route 53.

Note: A health check monitors the health of our end points.

LAB

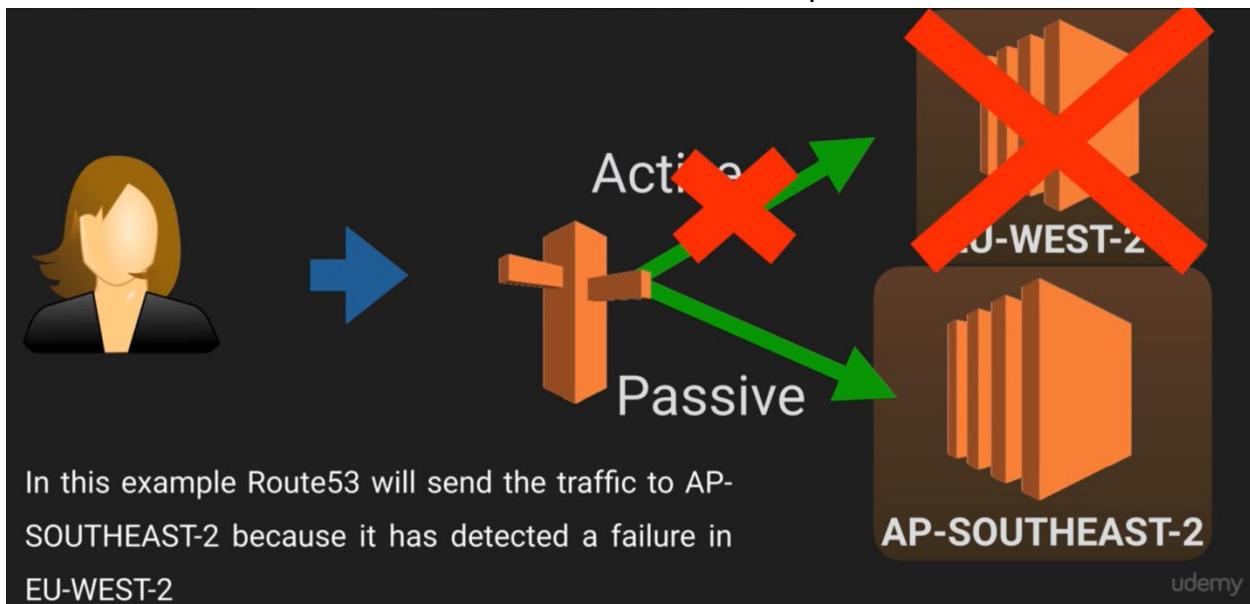
1. Go to Load Balancer in EC2 section under 'Compute' Services.
2. Select the Load Balancer for which further configuration is going to be made and copy its DNS name.
3. Now, go to Health Checks under Route53. Health check can be based on IP address or Domain name.
4. Give it a name.
5. Select 'Endpoint' in 'What to monitor'.
6. Select IP address under "Specify endpoint by".
7. Provide the IP address of the EC2 instance.
8. Keep the path just to be '/'.
9. Under Advanced configuration, we can set the request interval to be 30 seconds.
10. Keep the failure threshold to be 3 (3 failures equal failure for health checks).
11. We can enable the latency graph enabled if we want to see the Latency of these health checks.
12. We can invert the health check status, saying a healthy check is unhealthy.
13. We can configure the health checks to be from specific regions or just choose Recommended list.
14. We can check the pricing option (Health check for first 50 AWS endpoints are free).
15. Click on Next.
16. We'll not create an alarm for now.
17. Click on Create Health Check.
18. We'll create other similar health check other IP addresses.

Failover Routing Policy

These are used when we want to create an Active/Passive setup. For e.g.: we may want our primary site to be in Region 1 and secondary Disaster Recovery site in Region 2. Route53 will monitor the health of our primary site using a health check. If the health check fails, Route53 is going to send all our health check to Sydney.

- An instance is termed unhealthy if it fails 3 health check in a row and termed healthy if it passes 3 health check in a row.
- The default health checks is 30 seconds. But faster health checks can be set up (10 seconds) but at a higher cost.
- AWS launches 15 health checker in the background, which will check the endpoint health.
- If we calculate using the above 2 points, we would get a request every 2 seconds.
- We can have HTTP, TCP, and HTTPS health checks. When we use HTTPS health check, we will not get SSL certificate verification.
- We can integrate these health checks with CloudWatch if we wanted to.
- Once we have these health checks defined in Route 53, they can be directly linked to Record sets and DNS Queries and it'll change the behavior of Route 53.

Note: A health check monitors the health of our end points.



Lab

1. Click on Create Record set.
2. Select TTL to be 60 seconds.
3. Provide an IP address.
4. Select Routing Policy to be Failover.
5. Select Failover Record Type to be Primary.

6. Select Yes while associating with Health Check otherwise it gives an error.
7. Associate it with a Health Check. Make sure we provide the same IP of the Region in which we are associating our health check to.
8. Click on Create.
9. Create another record for another EC2 instance with Failover Record Type to be Secondary.
10. This time it is not mandatory to associate it with a health check.
11. Click on Create.
12. Go to the URL which we provided under name section. We can see, it gets directed to the IP of the primary instance.
13. Now, stop that EC2 instance. It makes the Health check fails.
14. If we navigate to Health Checks, we can see that one of the Health Checks goes unhealthy.
15. Under Monitoring, we can see the Health Check graph that report to the endpoint goes down to 0%.
16. If we go back to URL and refresh the page, we get the reply from the IP address provided under secondary EC2 instance.
17. Go to Load Balancer in EC2 section under 'Compute' Services.
18. Select the Load Balancer for which further configuration is going to be made and copy its DNS name.
19. Now, go to Health Checks under Route53. Health check can be based on IP address or Domain name.
20. Give it a name.
21. Select 'Endpoint' in 'What to monitor'.
22. Select 'Domain Name' while specifying the endpoint.
23. Copy the DNS of your ELB (mentioned in Step 2) in Domain Name.
24. Under Advanced configuration, select "Request Interval" to be say 10 seconds. Select Failure threshold to be 1 for now.
25. Click on 'Next' leaving all other configuration setting to be default.
26. We can choose to create an alarm or not.
27. Similarly create one more Health Check for another Domain name (for e.g.: <http://acoudguru.com>) with other configuration same as previous one.
28. Now, go to Hosted Zone & select your DNS name.
29. Create a record setting Alias to 'Yes' and Alias Target to one of the region.
30. Set Routing policy to Failover.

31. Choose Failover Record Type to be Primary.
32. Select "Evaluate Target Health" to 'Yes' & "Associate with Health Check" to 'Yes'.
33. Associate a Health Check to this record (the one which has Domain Name set to ELB).
34. Similarly create a second health record setting our Failover Record Type to be secondary leaving all others option to 'No'.
35. Go to Web Browser & enter the Domain Name. (<http://acloudguru.com>)
36. We can see the site is loading without any failure. Now stop the primary & secondary instances in ELB under EC2 services on by one from both the region.

Geo Location Routing

- It helps us to route our traffic will be sent on the basis of geographic location of the users (i.e., the location from which DNS queries originate).
- For e.g.: We might want to route all our queries from Europe to be routed to a fleet of EC2 instances that're specially configured for our European customers. These servers may have a local language of our European customers & all prices are displayed in Euros.
- We should create a "default" policy in case there's no match on location.



LAB

1. Go to Route53 under “Network & Content Delivery”.
2. Go to your hosted zone & create a record.
3. Provide a URL name (non-root).
4. Provide an IP of an EC2 instance under value section.
5. In Routing Policy, select Geolocation.
6. Select your location (from where the traffic is coming from).
7. Leave further configuration to ‘NO’. So now all the IP coming from the location (set in step 6) will go the IP provided in step3.
8. Create another 2 record with different Geolocation (set one of the Location to default and one in another region and all of them having different IP).
9. Now route to the name provided in step 3, we can see the page gets opened based on our current location.
10. Connect to a VPN to a location that is provided in step 8 and route to the same URL and it replies back with the nearest corresponding IP address location.
11. Connect to a different VPN and again refresh the page, it lands us to the page with of default location.

Points to remember:

1. Route53 can use public domain names (abc.com)
2. Private domain names (abc.internal)
3. Route53 has advances features such as:
 - a. Load balancing (through DNS-also called client side load balancing)
 - b. Health checks (limited)
 - c. Routing policy

Multi Value

- It is used when we route traffic to multiple resources.
- It is used when we want to associate a Route 53 health check with records.
- It is improvement over simple routing policy.
- It returns 8 healthy records for each multi value query.
- It is not a substitute for ELB (as done a sort of Load Balancing on the Client side).

Name	Type	Value	TTL	Set ID	Health Check
www.example.com	A Record	192.0.2.2	60	Web1	A
www.example.com	A Record	198.51.100.2	60	Web2	B
www.example.com	A Record	203.0.113.2	60	Web3	C

- If one of the instances will stop serving traffic, Route53 will not set the value of that to the clients but the other 2 will still be happening.

LAB

1. Go to Create Record Set under hosted zone.
2. Provide a IP address of an EC2 instance under value section.
3. Select the Routing Policy to be Multivalue answer.
4. Associate it with a Health Check of the corresponding location.
5. Click on Create.
6. Create another 2 records with different IP and its corresponding location and choosing Routing policy to be Multivalue answer.
7. So now, we've 3 records of Multivalue answer type.
8. Now, when we run the query nslookup www.yourDNSname.com. and we get back the 3 IP or dig www.yourDNSname.com. if any of the instance is down, it would route us to either of the 2 IP. Hence kind of Load Balancing on the Client side.

9. Elastic Beanstalk

It helps us to deploy our application safely & predictably. It is Platform as a Service (PaaS) and a developer centric view of deploying an application on AWS. Beanstalk is free but we pay for the underlying instances. It helps us in:

1. Managing infrastructure.
 - a. Instance configuration/OS will be managed by beanstalk.
 - b. Deployment strategy is configurable, but is performed by Beanstalk.
2. Deploying code is only the responsibility of the developer.
3. Configuring all database, load balancers, etc.
4. Scaling concerns.

Overview

Elastic Beanstalk is a

- Managed service
 - It performs instance configuration/OS.
 - Deployment strategy is configurable but performed by Beanstalk.
- Just the application code is the responsibility of the developer.

Three architecture models:

1. Single Instance deployment: good for development (Contains an EC2 instance + an elastic IP + an auto-scaling group + optional additional DB and all this under one AZ), so DNS address maps directly to Elastic IP.
2. LB + ASG for production or pre-production environment.
3. ASG: only for non-web apps in the production (spans across multiple AZ, in each AZ there can be multiple EC2 instances + optional RDS + ELB talks directly to ASG and will connect to EC2 instances) and ELB exposes a DNS name which'll be wrapped by Elastic Beanstalk DNS name

It has 3 components:

1. Application
2. Application version: each deployment
3. Environment name



We can deploy application version to environments & can promote application version to the next environments. There's a rollback feature to previous application version. We've a full control over lifecycle of environments. It has support for many platforms:

1. Go
2. Java SE
3. Java with Tomcat
4. .NET on windows server with IIS
5. Node.js

6. PHP
7. Python
8. Ruby
9. Packer builder
10. Single container Docker
11. Multi container Docker
12. Preconfigured Docker
13. Also, if any platform is not supported, we can write our own custom platform

LAB - I

1. Go to “Elastic Beanstalk” under services.
2. Give the application a name.
3. Choose a platform
4. For now, select “Sample Application”
5. Add a tag & hit create.
6. It says our application will be using a S3 bucket and if we navigate to S3, we can observe a new bucket has been created which contain few files/folders.
7. Go back to Elastic Bean Stalk, click on newly created environment and click on “Events” on the list.
8. We can see all the list of events that happened in our Elastic Bean Stalk, creation of bucket, security group, Elastic IP.
9. Click on the URL shown in the environment page created in step 5.
10. We can see that our environment is running on the dedicated environment in the AWS cloud.
11. The reason we have one EC2 instance in elastic IP is because we’re running the application in the simplest development mode and if we update our instance, the Elastic IP moves across instances.
12. We can download the logs under Logs section by requesting AWS from the console.
13. We can see the Heath Check of the application and we can monitor many other configurations.
14. Other things like monitoring, Alarms, managing updates for BeanStalk automatically (& it can be configured for OS) are also available.

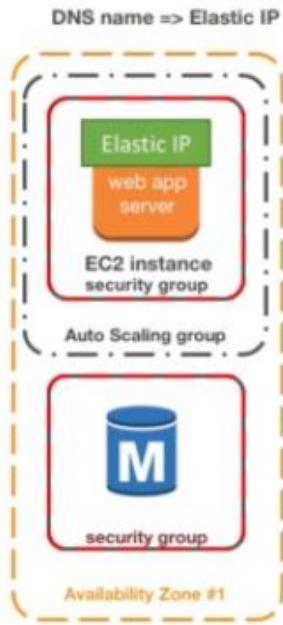
15. We can see the Application versions by clicking Application version under our environment.

LAB – II

Let us create a web server environment like a production environment. The option to choose Web Server environment or Worker Environment comes when we've already created a Beanstalk environment.

1. Click on “Create a new environment”
2. Select Web server environment.
3. Give your environment a name, choose a domain name.
4. Choose a platform.
5. For now, click on sample application.
6. Click on configure more options.
7. If we choose Configuration as “High availability”, we can observe many things changes such as Capacity, Load Balancer, etc.
8. AWS provides us the feature of doing Custom Configuration. Here we can configure our software (Proxy Server,), Root Volume Type, Security groups, Environment type, Number of instances, Instance types, AZs, Scaling metrics, etc.
9. We can externalize RDS to our Beanstalk environment, as the DB will remain even if we delete our Elastic Beanstalk environment.
10. Hit create & select “Low cost” for now to be in free-tier.

Deployment options for updates

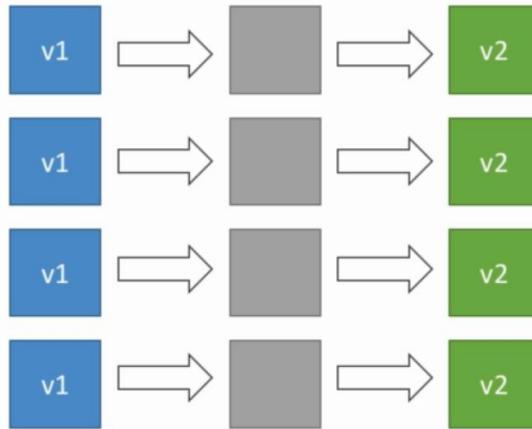


We've seen single instance deployment which is good for development where we get one EC2 instance, one Elastic IP, one Auto-Scaling group, a DB connection and all of them in one AZ. Also, the DNS name maps directly to the Elastic IP.

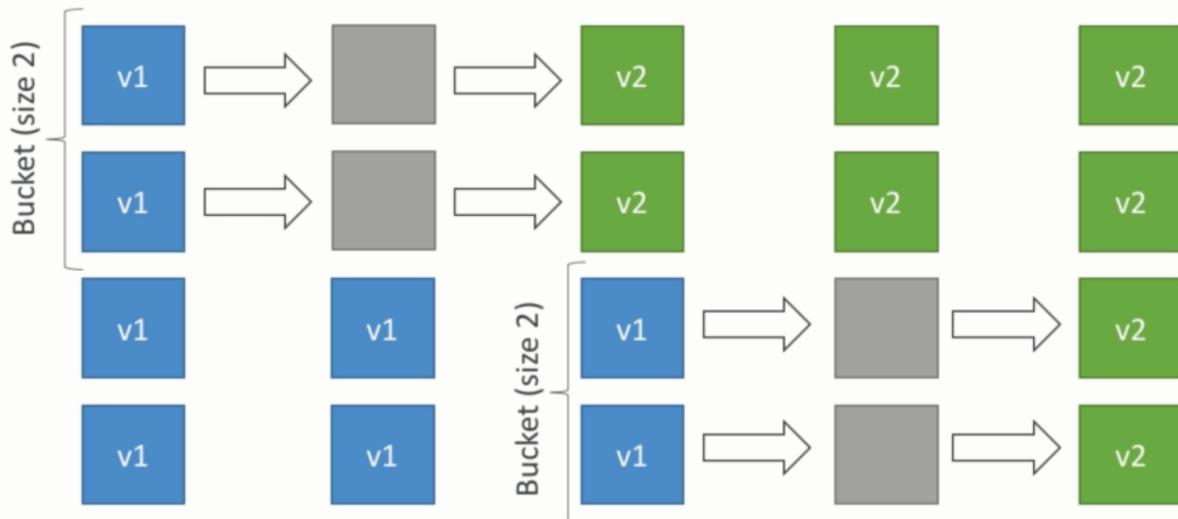


The second setup is with Load Balancer (in some cases Load Balancer is missing) and it is great for production type of deployments. We've ASG which spans across multiple AZ. Each AZ has several instances with their security group and they talk to an RDS (one master and a standby databases). The Elastic Load Balancer talks to ASG and will connect to all EC2 instances. The ELB exposes a DNS name which will be wrapped by the Elastic Beanstalk DNS name.

1. All at once (deploy all in one go): These are fastest but instances are not available to serve traffic for a bit (downtime).
2. Rolling: Update a few instances a time, and then move on to the next bucket once instance is healthy.
3. Rolling with additional batches: it is like Rolling, but it spins up new instances to move the batch (so that the application is still available and always operating at full capacity).
4. Immutable: Spins up new instances in a new ASG, deploys version to these instances, and then swaps all instances when everything is healthy.



- All at once:** Consider 4 EC2 instances and they all run the version one, now we are going to deploy v2. First Elastic Beanstalk stops the application on all our EC2 instances (in grey as they don't run anything) and we're running the new V2 because Elastic Beanstalk will deploy V2 to these instances. It's very quick deployment, the application has downtime (grey ones) and with this setup there's no additional cost.



- Rolling:** Application is running below capacity size (in stage 2, 3 or 4) and we can set the bucket size which is defined by bucket size (for now consider bucket size to be 2). The first 2 instances will be stopped (grey ones) but still we've 2 instances running V1. Then first 2 instances will be updated, so

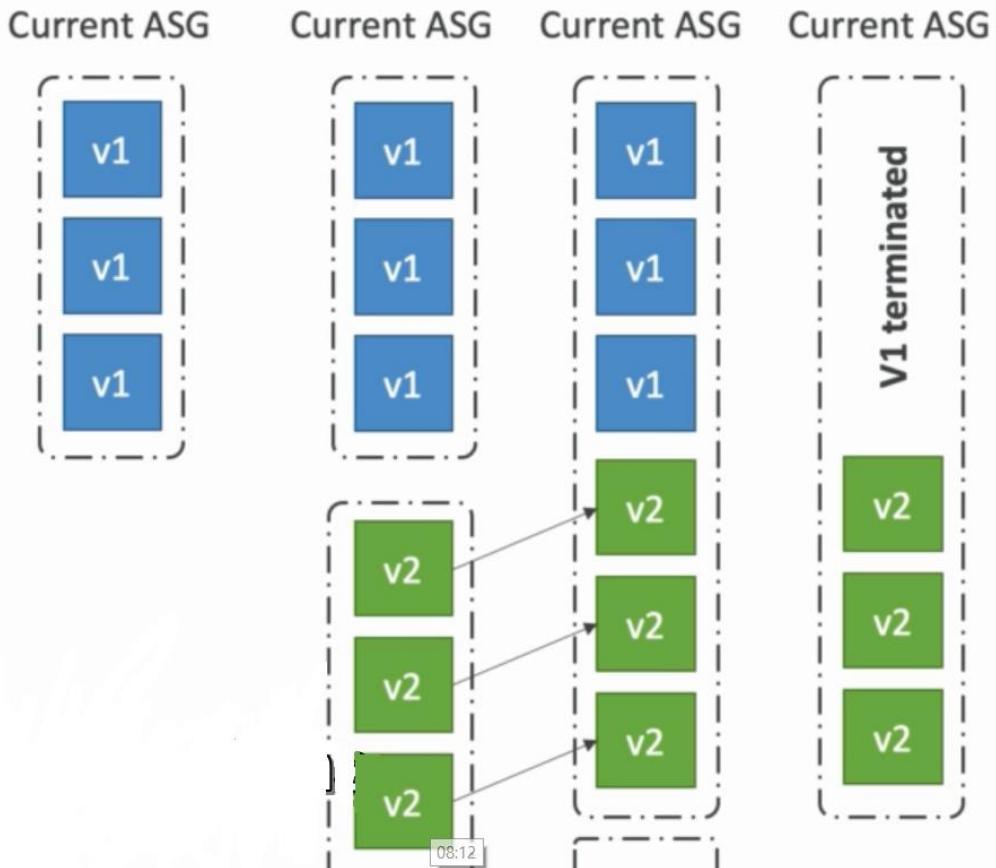
they'll be running V2 and then we will roll on to the next bucket or to the next batch & hence termed as rolling. Now it will have some downtime & will get updated to V2. At end, we'll have all instances updated to V2. So application is running both version simultaneously during deployment and there's no additional cost. If bucket size is small & instance count are more, then it'll take time to undergo this process.



3. Rolling with additional batches: Application is not running under capacity. Here we run at capacity & we set the bucket size and our application runs at both version simultaneously but at a small additional cost and that's additional batch which is removes at the end of deployment. The deployment is long it is a good way to deal with production.

We've 4 EC2 instances, and we're going to deploy 2 new EC2 instances which'll have V2 version on it. So Elastic Beanstalk created 6 instances from 4 instances. In next step, the first 2 bucket gets stopped and application is updated to V2 and the rolling process repeats again and finally all the application gets updated to V2. At the end, we've 6 EC2

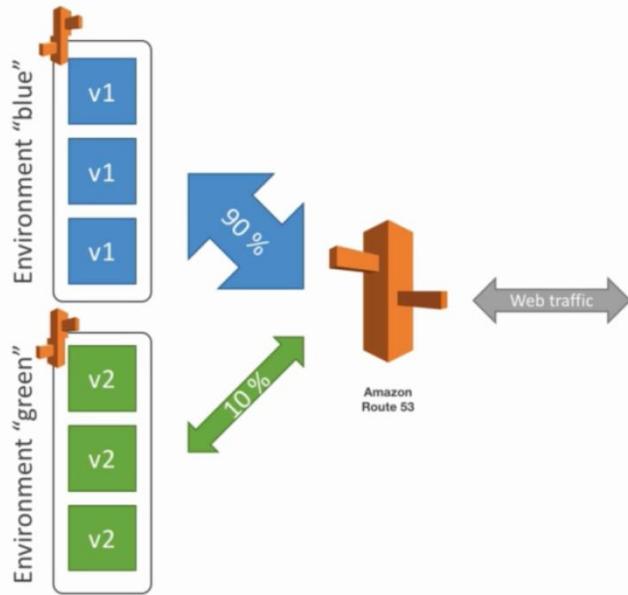
instances running V2 and at end additional batches get terminated and taken away.



4. **Immutable:** These deployments have zero downtime. The new code is deployed to new instances instead on previous instances as in last deployment methods. These instances come from temporary ASG. There's a high cost because we double the capacity as we get a full new ASG and it's the longest kind of deployment. As a bonus, we get a quick rollback in case of failures because to mitigate the failure we just have to terminate and Elastic Beanstalk will terminate the new ASG. So, it's a great choice for production if we're willing to take a little more cost.

We've 3 instances running at V1 and then we'll have new temporary ASG created. At first, ASG will launch 1 instance on it just to make sure if it works and if it works and passes the health checks, it is going to launch all the remaining ones. When the process is complete, it is going to merge the

temporary instances to the current ASG and when this is done & the temporary ASG is empty, then current ASG terminates



5. Blue/Green: it is not a direct feature of Elastic Beanstalk, it has zero downtime & release facility, more testing, etc. We just deploy a new stage environment (Elastic Beanstalk environment) and we'll deploy new V2 there. The last deployment strategies within the same environment but here we create a new environment and these environment can be validated in our time and can also be roll back if there's any issues and we can use Route53 to prevent the traffic from going to the two directions by setting the weighted policy and redirect a small traffic to the stage environment and then swap the URL of test environment with the new one.

Summary table:

Deployment Methods

Method	Impact of Failed Deployment	Deploy Time	Zero Downtime	No DNS Change	Rollback Process	Code Deployed To
All at once	Downtime	⌚	X	✓	Manual Redeploy	Existing instances
Rolling	Single batch out of service; any successful batches prior to failure running new application version	⌚⌚⌚+	✓	✓	Manual Redeploy	Existing instances
Rolling with additional batch	Minimal if first batch fails, otherwise, similar to Rolling	⌚⌚⌚+	✓	✓	Manual Redeploy	New and existing instances
Immutable	Minimal	⌚⌚⌚⌚	✓	✓	Terminate New Instances	New instances
Blue/green	Minimal	⌚⌚⌚⌚⌚	✓	X	Swap URL	New instances

LAB

1. Go to Elastic Beanstalk under Services.
2. Create a new application, by choosing an application name and a platform.
3. Click on “Configure more option” and then click “Modify” on “Rolling updates and deployment”.
4. Choose a deployment policy.
5. Select a batch size, the percentage of the fleet or a fixed number of instances getting updated. For e.g., if we select the percentage to be 30%, then 70% of the fleet will always be up and if we say fixed, and define the number to “1”, then only one instances can be taken down at a time.
6. When we choose “Rolling with additional batch”, we make sure all the application is running at capacity and bigger the batch size, more the cost.
7. If we choose immutable, we can't choose everything here since a new auto scaling group is created.
8. We can define the configuration updates happen by choose the type of rolling updates being happening.
9. We can select the health checks, healthy threshold and command timeout while deploying the Elastic Beanstalk.
10. Go to the web browser and search for sample web application in your happy language(https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Related_Resources.html)
11. Download the zip code.
12. Make few changes like the color code.
13. Upload the zip to Elastic Beanstalk application created in LAB-1.
14. After deployment is finished, we can see the successful change.
15. In the Event section, we can check out for the logs.

Elastic Beanstalk CLI

We can install an additional CLI called “EB CLI” which makes working with Beanstalk from CLI easier. Basic commands are:

1. eb create
2. eb status
3. eb health
4. eb event

5. eb logs
6. eb open
- 7. eb deploy**
8. ebconfig
9. eb terminate

It is helpful for automated deployment pipelines.

Beanstalk lifecycle policy

1. Elastic Beanstalk can store at most 1000 application versions.
2. If we don't remove the old version, we won't be able to deploy any more.
3. To phase out old application versions, we use a lifecycle policy
 - a. Based on time (old version are removed)
 - b. Based on space (when we've too many versions)
4. Currently using version can't be deleted.
5. If version get deleted, we've an option not to delete underlying source bundle in S3 and it allows to prevent data loss and restore that versions if we want in future.

LAB

- a. Go to Elastic Beanstalk application created
- b. Click on "Load Configuration" under 'Actions'.
- c. Click "Application versions".
- d. Select the application version & click on 'Settings'.
- e. Now we can see the lifecycle settings.
- f. We can enable it by clicking on the checkbox.
- g. We can remove the version by either choosing the number of application version or by selecting number of days after which the application version would get deleted.
- h. 'Retention' option helps to keep or delete the source bundle from S3.
- i. The 'Service Role' allow us to choose the role which will delete this versions.
- j. Hit 'Save'.

Elastic Beanstalk extension

- When we create a zip file, it contains the code that must be deployed to Elastic Beanstalk but we can also add ebextensions.
- All the parameters set in the UI can be configured with code using files.
- Requirements:
 - All these files must be in the .ebextension/ directory in the root of the source code.
 - It must be in the YAML or JSON format.
 - Extension of the file must end with .config
 - We can modify some default settings using: option_settings.
 - We've the ability to add resources such as RDS, ElastiCache, DynamoDB, etc... that cannot be set using Elastic Beanstalk console.
- Resources managed by .ebextension gets deleted if the environment goes away.

```

.ebextensions ← ebextension directory
environment-variables.config
Icon
app.js
cron.yaml
Icon

option_settings:
# see: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/command-options-general.html#command-options-general-elasticbeanstalkapplicationenvironment
aws:elasticbeanstalk:application:environment:
  DB_URL: "jdbc:postgresql://rds-url-here.com/db"
  DB_USER: username
  Environment
  variables

```

In our code directory (.ebextension directory), we've placed the .config file. Under option settings, we've set the AWS Elastic Beanstalk environment variables (if later we want to connect to an RDS DB, then we need to set the environment variables this way). To make it work, we need to zip the folder and then go to your Application under Elastic Beanstalk, navigate to your environment and click on "Upload & Deploy" and upload the zip file.

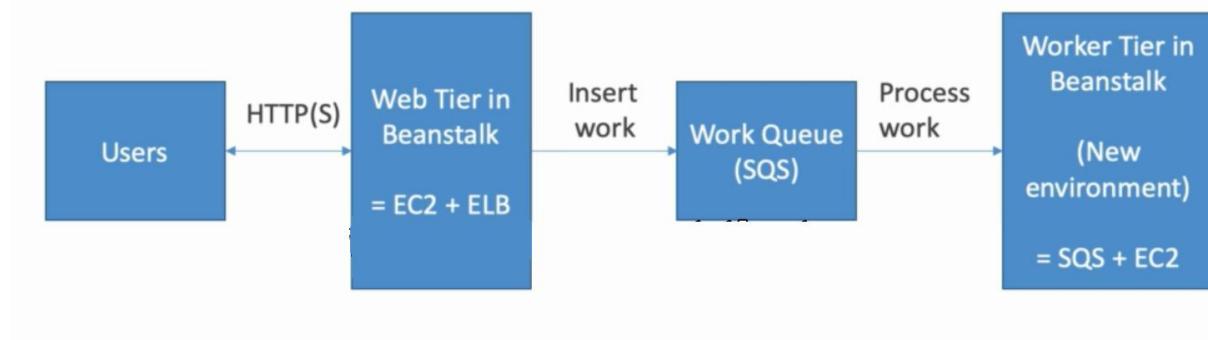
Name	Value
DB_URL	jdbc:postgresql://rds-url-here.com/db
DB_USER	username

Once the environment has been deployed, go to software -> Edit -> Environment properties, we can see the environment variables got set automatically.

Under the hood, Beanstalk relies on CloudFormation

Web Server vs Worker Environment

If our application perform task that are long to complete, offload these task to a dedicated environment. By doing this, we decouple our application into 2 tiers, webserver tier & worker tier. Ex of a long task can be: processing a video, generating a zip file, etc. We can define some period task in cron.yaml file.



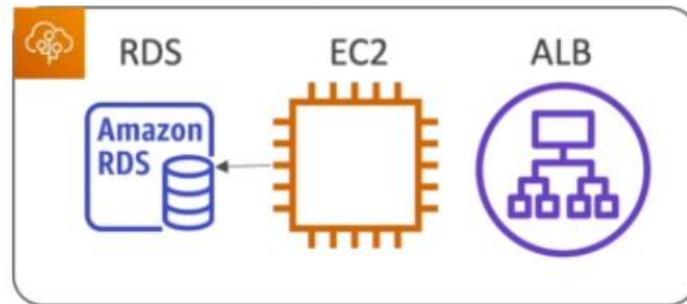
Our users talk with HTTP or HTTPS to our web tier in Beanstalk which is made up of EC2 & ELB. If some task need to be performed, we insert it into Work Queue which is SQS. Now, the work queue is processed by worker tier in Beanstalk which is a separate new environment which comprises of work tier & EC2 instances.

LAB

1. Go to ‘Actions’.
2. Click on “Create environment”.
3. Select “Worker environment”.

RDS with Elastic Beanstalk

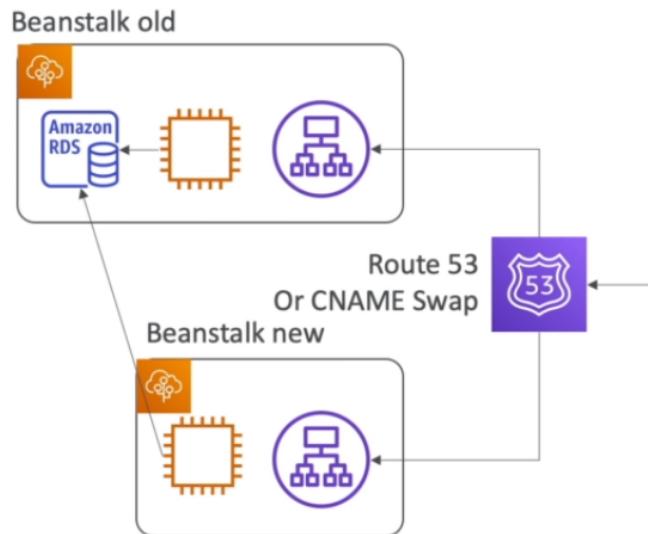
Beanstalk with RDS



RDS can be provisioned with Elastic Beanstalk, which is great for dev/test.

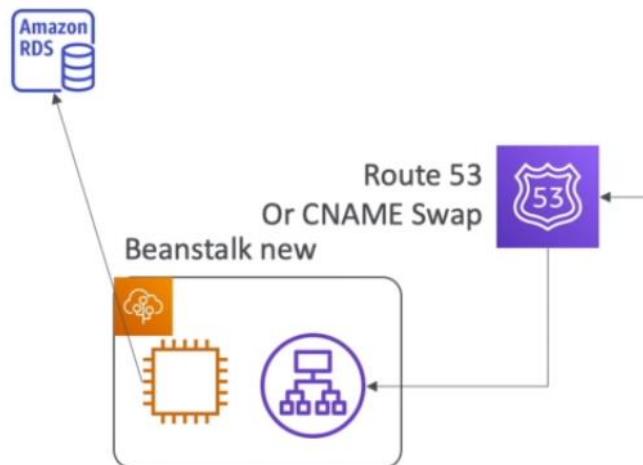
- This is not great for production environment as if we delete our Beanstalk environment, our RDS DB will get deleted since they are tied to each other.
- The best for prod is to separately create an RDS DB & provide a connection string to the Elastic Beanstalk.

Let's learn to migrate if we by mistake couple our RDS DB to Beanstalk and we want to decouple and migrate it to standard RDS:



1. Take an RDS DB snapshot.
2. Enable deletion protection in RDS, it prevents RDS DB from being terminated.

3. Create a new environment, now without an RDS but pointing to existing old database.
4. So now we've 2 Beanstalk environments and 1 RDS that is protected against deletion.
5. We are now going to perform blue/green deployment and swap the new & old environment.



6. Terminate the old environment, but now the RDS won't get deleted. We'll delete the CloudFormationtask stack because it'll be in delete failed state.

Docker

Single Docker

- It is possible to run our application as single Docker Container.
- For this we provide:
 - Dockerfile: Elastic Beanstalk will build & run Docker container.
 - Dockerrun.aws.json (v1): It is a definition and we describe where the already built Docker image is and it could be in:
 - ECR repository
 - Docker Hub
 - So we define the
 - Image
 - Ports
 - Volumes
 - Logging

- Beanstalk in Single Docker Container does not use ECS.

Multi-Docker

- Multi-Docker helps run multiple containers per EC2 instance in Elastic Beanstalk.
- It'll create
 - ECS Cluster
 - EC2 instances configured to use the ECS Cluster
 - Load Balancer (in high availability mode).
 - Task definition and execution.
- For these, we need to write Dockerrun.aws.json (v2) at the root of the source code.
- This file generates ECS task definition.
- Our Docker image must be pre-built and will be stored in Docker hub or ECR or other repositories.

Points to remember

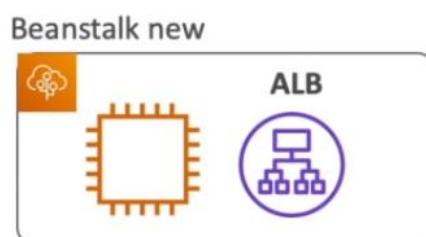
1. Elastic Beanstalk work on the basis or relies on CloudFormation (an AWS service). For every application we create in Elastic Beanstalk, we see that a CloudFormation stack is been created and it triggers all the changes that happens in our environment.

We can select our stack and click on “Resources” tab to view the resources that’re being created.
2. When we deploy our application, we describe the dependencies of our application and it can be requirement.txt file for Python, package.json for Node.js., then we package the code as zip & the zip is uploaded to EC2 machines (the zip files also acts as a method to keep our files secure). Then, EC2 machine resolve the dependencies. These deployment can be slow if the dependencies are big.
3. We can clone the Elastic Beanstalk environment which is helpful for deploying a "test" version of our PROD application. In this case, all the resources and configuration are preserved which includes:
 - a. Load Balancer types and configuration.

- b. RDS DB type (configuration of DB is preserved but the data is not going to be preserved).
- c. Environment variables.

After cloning the environment, we can change the settings.

For this, go to your environment under Elastic -> Environments and under Environment actions, select "Clone Environment"



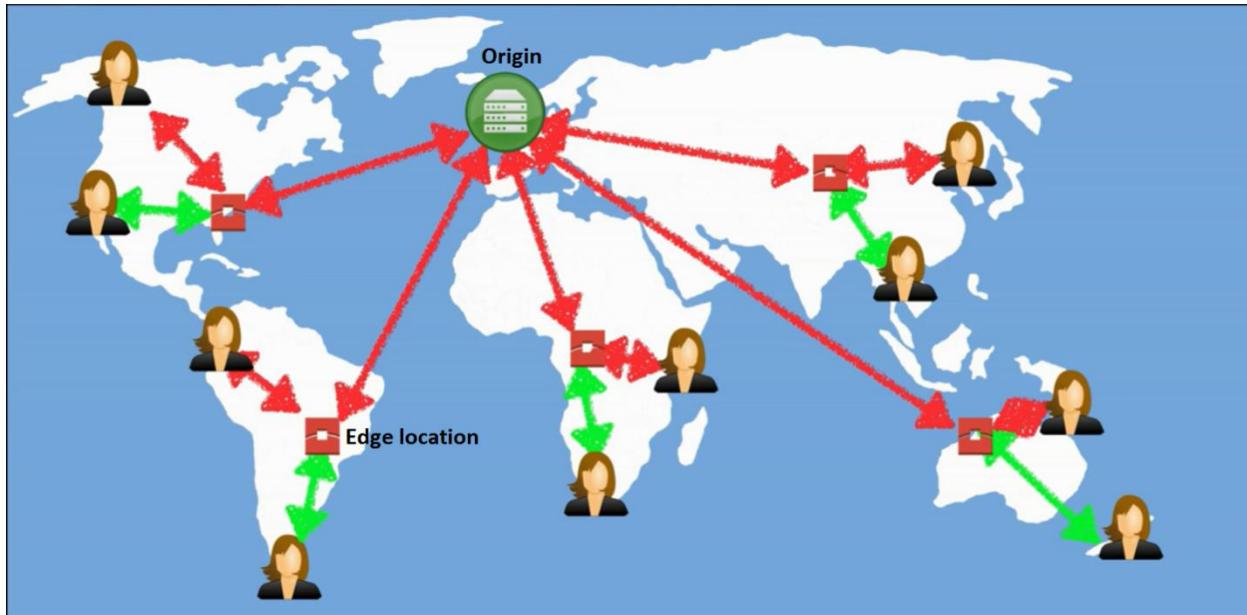
4. Elastic Beanstalk migration

- a. After creating an Elastic Beanstalk environment, we cannot change the Elastic Load Balancer type (like CLB to ALB).
- b. In order to do so, we need to perform migration by
 - i. Creating a new environment with the same configuration except the Load Balancer (we cannot use clone feature as it'd copy the Load Balancer type and other configuration).
 - ii. Deploy the application onto new environment.
 - iii. Shift the traffic from old environment to a newer one, to do this we can perform CNAME swap or Route 53 to do a DNS update.
- 5. Optimization in case of lengthy deployments, package dependencies are available with the source code to improve performance and speed and they get uploaded to EC2 instances right away.
- 6. Elastic Beanstalk work with HTTPS.

- a. We need to load SSL certificate to the Load Balancer which can be done from the console under Elastic Beanstalk configuration.
 - b. It can be done from code. Ebextensions/securelistener-alb.config
 - c. SSL certificate can also be provisioned using ACM (AWS certificate manager) or CLI.
 - d. Security group must be configured to allow incoming port 443.
7. Beanstalk redirect HTTP to HTTPS.
- a. We need to configure our application instances to redirect HTTP to HTTPS (see in AWS repository).
 - b. Or we can configure our Application Load Balancer with a rule to redirect HTTP to HTTPS. (Make sure the health checks are not redirected giving 200 error)

9. CloudFront CDN

- A Content Delivery Network is a system of distributed servers (network) that deliver faster (improves READS) webpages & other web content to a user based on the geographic locations of the user, the origin of the webpage & content delivery server as the content is cached at the edge location.
- CloudFront also provides us DDoS (Distributed Denial of Service) protection, integration with Shield and Web Application Firewall.
- It can expose external HTTPS and can talk to internal HTTPS backends.
- CloudFront distributes the READS all around the world based on different Edge Locations and it improves latency. It reduces loads on main S3 bucket.



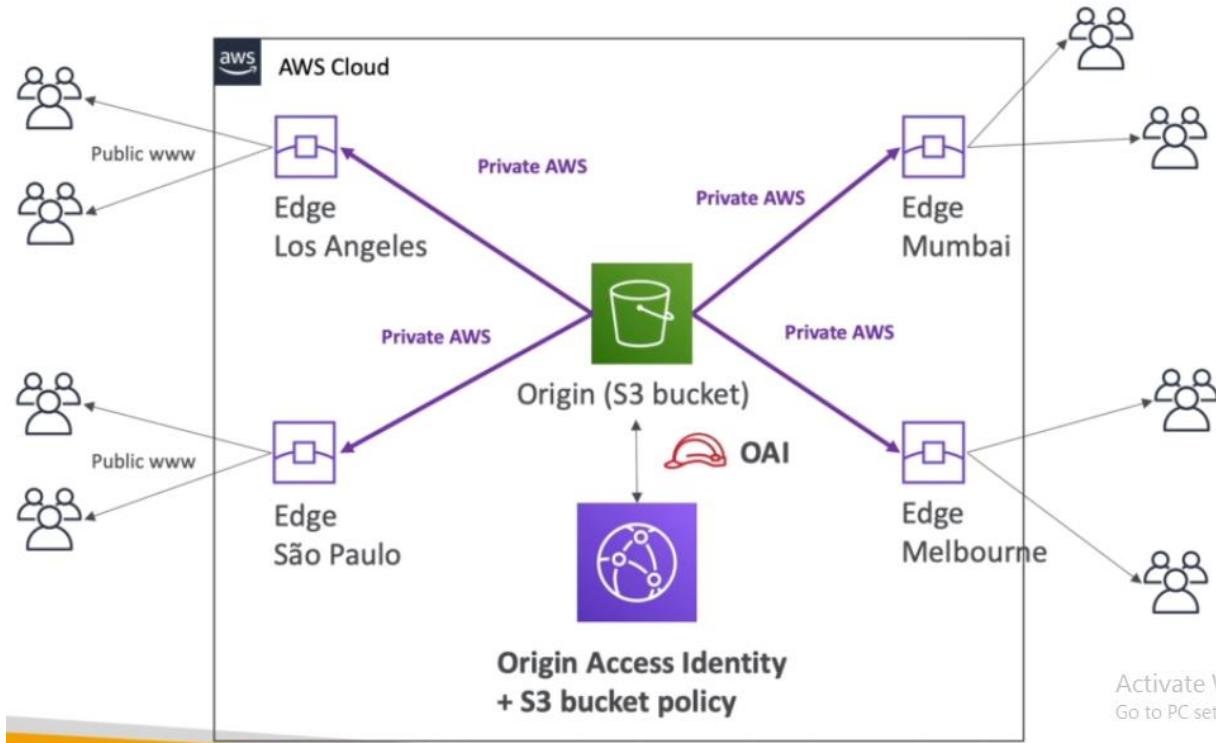
When the 1st user tries to access the files, the file get cached from the origin to the edge location. But now, if the 2nd user tries to access it, since the file is already routed so it takes less time to access the file.

Amazon CloudFront can be used to deliver the entire website, including dynamic, static, streaming & interactive content using a global network of edge location. Amazon CloudFront is optimized to work with other Amazon web services, like simple storage service (S3), EC2, Amazon Elastic Load Balancing, and Amazon route 53.

Amazon cloud also works seamlessly with any non-AWS origin server, which stores the original, definitive versions of your file.

Origins

- S3 bucket
 - For distributing files and caching them at the edge.
 - Enhanced security with CloudFront.
 - CloudFront can be used as an ingress (to upload files to S3).



Let's suppose we've a cloud and an origin as S3 bucket. Let's have an edge location at Los Angeles and users want to read data from there, so Edge location fetches the data from S3 bucket over private AWS network and return the result from Edge Location. The idea is that inorder to access the S3 buckets, the edge location of the CloudFront uses an OAI (Origin Access Identity - an IAM Role for CloudFront origin) to access S3 bucket. Thn the bucket allows to files to CloudFront.

- Custom Origin
 - Application Load Balancer
 - EC2 instance.
 - S3 websites (must first enable the bucket as a static S3 website).
 - Any HTTP backend we want.



Compare to S3 bucket, in this case the security changes a little bit. We've EC2 instances which are public since they must be publically accessible from HTTP standpoint. The users access the Edge location and edge location access our EC2 instance and edge location traverses the security group. Hence, the security group must allow the list of IPs of CloudFront edge locations into the EC2 instance.



If we use ALB as an origin, we've a security group for the ALB and the ALB must be public so that it is accessible by CloudFront. But the backend EC2 instances now can be private. The security group of the EC2 instances should allow SG of Load Balancer. Edge location are public locations and it needs to access the ALB through public network, hence SG of ALB must allow public IP of Edge location.

Distributions

The name given to the CDN which consists of collection of Edge locations. There are 2 types of distributions:

1. Web distributions – Typically used for websites.
2. RTMP – A real time messaging protocol. Used for media & streaming between a Flash player & server.

Note: Edge location are not only for READ, we can even write new files to Edge locations. Then the object will be put back to the original Origin servers. Objects are cached for TTL (time to live) and after setting the TTL period, it'll expire. We can clear cached object but you'll be charged for this.

CloudFront vs. CORS

- CloudFront
 - Global Edge Network
 - Files are cached for a TTL
 - Great for static content that must be available everywhere.

- S3 CORS
 - It must be setup for each region where we want replication to happen.
 - Files are updated in near-real time.
 - READ only
 - Great for dynamic content that needs to be available at low-latency in few regions.

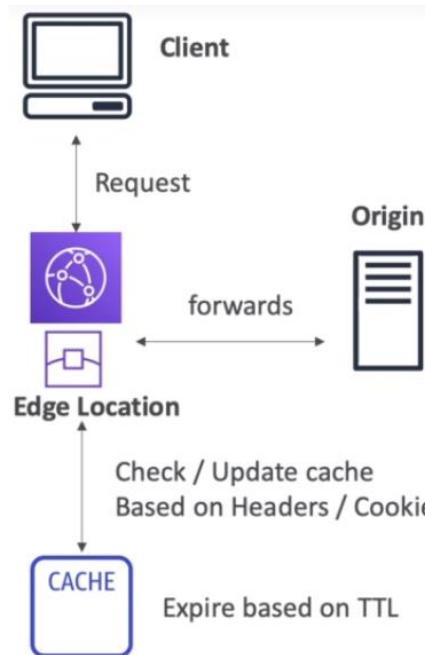
LAB

1. Create a bucket with default settings.
2. Upload files on to bucket.
3. Go to services, under Network & Content Delivery, select “CloudFront”.
4. Click on Create Distribution.
5. We can choose Web or RTMP.
6. Click on Get Started (For now, let's select Web distribution).
7. Select Origin Domain Name, it is auto-populated which get it data from bucket names, select the bucket name created in step 1.
8. Now select the origin ID, it is used to differentiate multiple distribution within the same origin. It is just a description for the origin.
Note: We can have multiple distributions within the same origin.
9. Restrict public access block the public access to bucket through the URL (bucket DNS). All request must go through the CloudFront now.
10. Create a new Origin Access Identity.
11. Say "Yes, Update Bucket policy" under Grant Read permission on Bucket.
12. Origin path – if we want CloudFront to request your content from a directory in your Amazon S3 bucket or your custom origin, write the directory name (& this will be your origin path). It begins with '/'. CloudFront appends the directory name to the value of the Origin Domain Name when forwarding the request to your origin. E.g. myawsbucket/production.
13. Path-pattern – It forwards all the request to the origin. To change the behavior or the routing for other request (for e.g. *.jpg'), add more cache behaviors after you create the distributions.
14. We can set the TTL options which is minimum of 86400 sec (24 hr.) & max of 292 days.

15. We can also forward cookies, cache, query string forwarding.
16. Restricted viewer access lets you to forward URL to some particular/selected person.
17. We can even upload our own custom SSL certificates.
18. We can set the Viewer protocol policy like both HTTP & HTTPS, or only HTTPS or redirecting the HTTP request to HTTPS.
19. We can allow various HTTP method so that we can give different type of access to the users like GET, HEAD, or GET, HEAD, OPTIONS, or GET, HEAD, OPTIONS, PUT, POST, HEAD, DELETE.
20. d1mpbuaq23ng64.cloudfront.net/megan_fox.jpg
21. We can set geographical restrictions to our web content & further we've 2 option of 'white listing' & 'black listing'.
22. We also can disable the distribution which may take even up to 15 minutes.
23. If we look at the bucket policy, it is allowed to GetObject.
24. If we try to make the files inside the bucket publically accessible, we get access denied error, so we need to change the permissions of the file and the bucket to unblock public access.
25. Go to CloudFront URL, add a slash '/' and the file name. we can see the CloudFront URL is later redirected to AWS S3 bucket URL, and in order to get direct access, we need to wait 3-4 hours.
26. Now we can remove the public access from the files and remove Read Object and apply all settings that blocks public access under Bucket permissions.

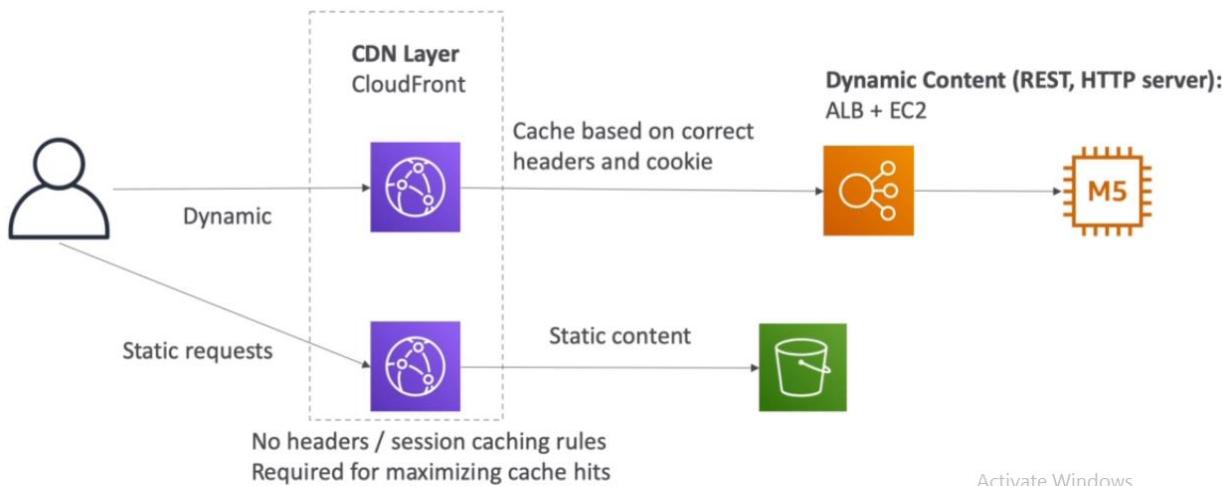
Caching

- Cache is based on
 - Headers
 - Session Cookies
 - Query String Parameters
- The cache lives at each CloudFront edge location.



We've a client making request to the CloudFront Edge location and the cache will be checked on the base of values of headers, query string parameters and the cookies. The cache will have expiry based on TTL. If the value is not in the cache, then the HTTP request will be forwarded to the origin and then the cache will be filled by the query response.

- We're minimizing the request to the origin by maximizing the amount of cache hits.
- We can control the TTL (0 seconds to 1 year).
- We can invalidate part of cache using the CreateInvalidation API.



In CloudFront, we can separate our static and dynamic distributions. It means our static request are going to go through CloudFront into our static content holder (S3 bucket). When we've these request, no headers/caching rules is applied and hence all the static content is going to be cached in CloudFront, hence maximizing the cache hits.

In case, when we've dynamic content, it is going to travel through CloudFront, a different distribution. But this time, it is going to be forwarded to an HTTP server (ALB + EC2 instance)

LAB

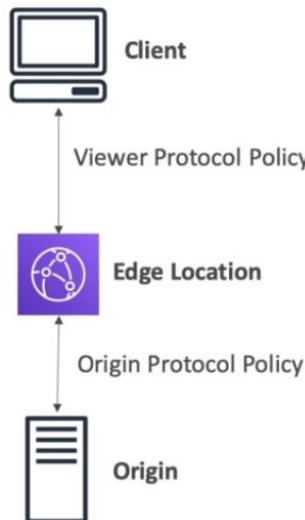
1. Browse the URL that we got in the last lab.
2. In order to check the TTL settings, go to CloudFront and navigate to Distributions.
3. Go to Behaviour tab, click on the behaviour & edit.
4. We can view the default TTL to be 86400. So, for this period our data is going to be cached in the CloudFront.
5. If we've a file in S3, which is accessed through CloudFront URL and if we make any changes to it, then changes are not going to be reflected though it gets reflected using the pre-signed URL.
6. So, we are going to perform invalidation. By invalidating, we are going to remove them from the CloudFront Edge caches.
7. Navigate to Invalidations tab under Distributions in CloudFront.
8. Click Create Invalidations.
9. Under Object Paths, write '*' and click on Invalidate. '*' invalidates everything here.
10. While the Status is in progress, CloudFront talk to all the Edge around the Globe, and tell them to flush their caches based on invalidation.
11. Once the status is completed, on page refresh we'll get the updated content of S3.

Security

- The first bit of security is Geo-restriction, which restricts user based on Geo-Location and there we can define which countries are banned or

approved from accessing CloudFront location. The country is determined using 3rd party GeoIP database.

- The second way of enabling security on CloudFront is HTTPS.



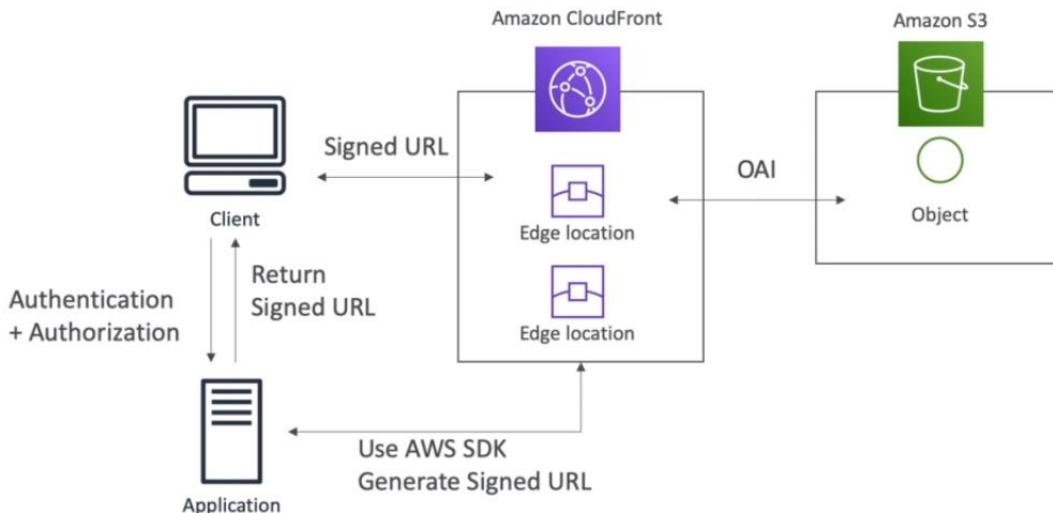
The client is talking to the Edge location which talks to our origin. We can setup 2 types of security:

- i. Viewer Protocol Policy: It is between our clients and edge location. The traffic going to the CloudFront can be encrypted by:
 - a. We can redirect HTTP to HTTPS.
 - b. Or to use HTTPS only.
- ii. Origin Protocol Policy (origin can be HTTP or S3): we can set it up to be:
 - a. HTTPS only
 - b. Or Match Viewer (HTTP -> HTTP & HTTPS -> HTTPS)
- Note: S3 bucket "websites" does not support HTTPS.

[CloudFront Signed URL/Cookies](#)

- Let's suppose we've a CloudFront distribution and we want to keep it private and want to give paid shared access to premium users over the world and we also want to know who has access to CloudFront distribution.
- For this, we can use CloudFront Signed URL/Cookie.
- We need to attach a policy with:
 - Including the URL expiration time.

- Including the IP ranges to access the data from.
- Trusted signers (telling which AWS account can create signed URLs for our users).
- Signed URL = access to individual files (one signed URL per file).
- Signed Cookie = access to multiple file & cookie can be reused (one signed cookie for many files).



We've CloudFront distribution and it has bunch of Edge Locations. We can access S3 buckets through OAI (Origin Access Identity) for full security. It means object in our S3 bucket cannot be accessed by anything else but CloudFront. Still, we want to give access to the people through CloudFront. Now, the client is going to authorize and authenticate our application and we've to code to that application. Our application uses AWS SDK to generate signed URL from CloudFront. It returns signed URL to the client and then the client uses the signed URL to get the data and files/object directly from CloudFront. These work both for signed URL/Cookie.

CloudFront Signed URL vs. S3 Pre-Signed URL



- CloudFront Signed URL:
 - It allows access to the path, no matter the origin. So, it not just work for S3 as an origin, but also for HTTP, backend, etc.
 - It is an account wide key-pair, so only the root can manage it.
 - It can be filter by IP, path, date and expiration.
 - We can leverage the caching features out of CloudFront.

We've a client using the signed URL onto our CloudFront Distribution and CloudFront Distribution talks to the origin (here, it is an EC2 instance).



- S3 Pre-signed URLs
 - It issues the request to the person who pre-signed the URL.
 - Uses the IAM key of the signing IAM principal.
 - Limited lifetime.

9. Databases

A structured set of data held in a computer that allows data to be easily accessed, manipulated & updated.

Relational Database

It uses a set of formally described tables from which can be accessed or reassembled in many different ways without having to organize the database tables. With a relational database, we can quickly compare information because of the arrangement of data in columns. Its types are:

1. SQL Server (imp.)
2. Oracle
3. MySQL server
4. PostgreSQL
5. Aurora (imp.)
6. Maria DB

Non-Relational Database

It is a database that does not use the tabular schema of rows & columns found in most of the traditional database systems. It has advantage of the ability to store and process large amounts of unstructured data. As a result, it can process any type of data without needing to modify the architecture. SO, creating & maintaining a NoSQL database is faster and cheaper. E.g.: DynamoDB – No SQL The structure is like that we have a database & inside that database we've a collection. Inside a collection we've a document & inside that document we've a key value pairs. Comparing both types of DB, Table of Relational DB is similar to Collection of Non-Relational DB. Similarly, Row can be compared with Document & Fields equals Key Value Pairs. We can create new Database & we can have new collection in there and when we add new document, we can add as many key pairs as we want.

```
{
  "_id": "51262c865ca358946be09d77",
  "firstname": "John",
  "surname": "Smith",
  "Age": "23",
  "address": [
    {"street": "21 Jump Street",
     "suburb": "Richmond"}
  ]
}
```

Data Warehousing

It is used for business intelligence. It is a system used for reporting and data analysis. They are the central repositories of integrated data from one or more

heterogeneous sources. Used to pull large & complex datasets. Usually used by the management to do queries on Data such as current performance and targets, etc. Tools like Cognos, Jaspersoft, SQL server reporting services, Oracle Hyperion, SAP NetWeaver.

[OLTP vs OLAP](#)

Online Transaction Processing differs from Online Analytics processing in terms of the queries we run.

Example of OLTP: Order_no. 5648 (Select * FROM order_no. 5648)

It pulls up the row of data such as Name, Date, Delivery address, Delivery status, etc. Another use case is OLTP is used in online shopping where we are using OLTP transaction.

Example of OLAP: If we want to pull the data for Net Profit of EMEA & Pacific from the Digital Radio product. So, it is going to pull a large number of records. So we need to calculate:

- i. Sum of Radios sold in EMEA
- ii. Sum of Radios sold in Pacific
- iii. Unit cost of Radio in each region.
- iv. Sales price of each radio.
- v. Difference between Sales price and unit cost.

Querying of these calculation puts lots of stress on the DB. And we will not run on our production Database i.e., on our online store as it will hammer the DB. We want to be ticking along doing all the OLTP transaction. So we should transfer all our production Database into OLAP database or Data warehouse where we will run these queries where it will not hamper our production database. They are designed quite differently from the database perspective & infrastructure layer.

[RDS Backups, Multi AZ & Read replicas](#)

[RDS](#)

Relational Database is a managed DB service for DB which use SQL as a query language. It allows us to create databases in cloud which are managed by AWS. Databases that can be created are:

1. PostgreSQL
2. Oracle

3. MySQL
4. Maria DB
5. Microsoft SQL Server
6. Aurora

Advantage of RDS versus deploying DB on EC2

1. Managed service
2. OS patching
3. Continuous backups & restore to specific timestamp.
4. Monitoring dashboards
5. Multi AZ setup for disaster recovery.
6. Maintenance windows for upgrades.
7. Scaling capability.

Note: the disadvantage is we cannot SSH into EC2 instance.

[RDS Backup](#)

There are 2 different type of backups:

1. Automated backups
2. Database snapshots

[Automated backups](#)

- Backups are automatically enabled in RDS.
- Daily full backup of database (during the maintenance window).
- Transaction logs are backed-up every 5 minutes.
- Have the ability to restore to any point in time (from oldest backup to 5 minutes ago).
- They allow us recover our database to any point of time within a “retention period”. Retention period can be between 1 to 35 days & default is 7 days.
- It takes full daily snapshots and will take full daily snapshots & will store transaction logs throughout the day. While recovery, AWS will choose the most recent snapshot & will apply transaction logs to the day. This allows us to do point in time recovery down to a second, within a retention period.

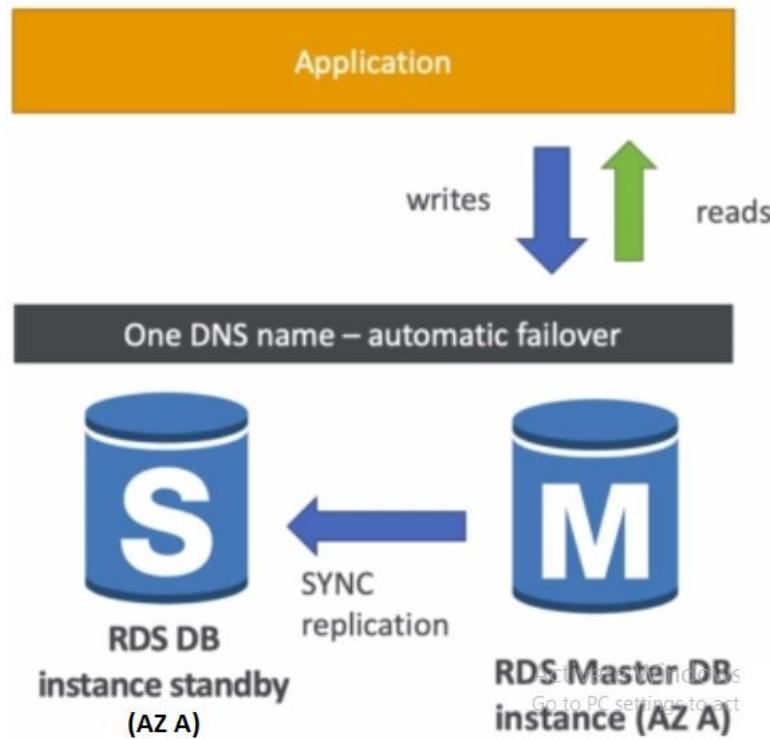
- Automated backups are enabled by default. The backup data is stored in S3 & we get free storage space equal to the size of our database. So, if we've an RDS instance of 10 GB, we get 10 GB worth of storage.
- Backups are taken within a defined window, storage I/O may be suspended while our data is being backed up & we may experience elevated latency.

Database Snapshots

- They are done manually (i.e., user initiated).
- They are stored even if we delete the original RDS instance, unlike automated backups.
- The retention of backup for as long as we want.

Whenever we restore automated backups or database snapshots, the restored version of the snapshot will be new RDS instance with a new endpoint.

Multi-AZ



Let's say we've ELB with 3 EC2 instances behind it. All these instances are connected to a primary RDS instance in any one of the AZ. This RDS instance is going to use synchronous replication in another AZ. If one of the AZ fails, AWS will

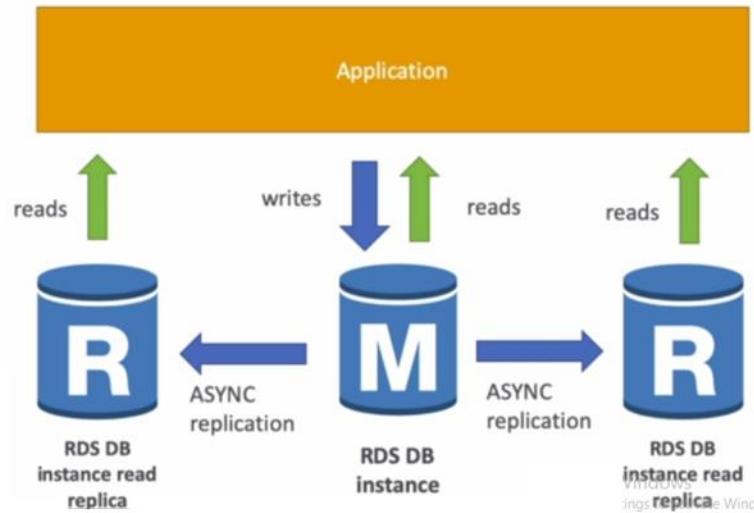
automatically fails to another AZ using the same endpoint, so we don't need to change the connection string.

It allows us to have an exact copy of our production DB in another availability zones. AWS handles the replication, so when our production DB is written to, this will automatically be synchronized to the stand by database. In the event of planned DB maintenance, DB instance failure, or an AZ failure, Amazon RDS will automatically failover to standby so that DB operation can resume quickly without administrative intervention. We get one DNS name. we get a failover in case of loss of AZ, loss of network, instance or storage failure and in that case our standby DB will become the new master. We do not need to do any manual intervention in our app as long as it tries to keep on connecting to our DB automatically, and at some point it will failover to the standby.

It is for Disaster recovery. It is not primarily used for improving performance or scaling. For performance, we need Read Replicas. Multi AZ DB is available for SQL server, Oracle, MySQL Server, PostgreSQL, and MariaDB.

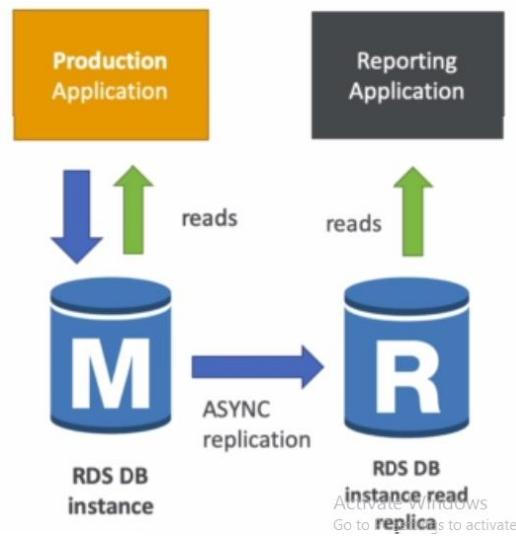
Read Replicas

It helps to scale our Reads. Let's have ELB with 3 EC2 instances behind it. All these instances are connected to an RDS instance. Read Replicas uses asynchronous replication (Synchronous replication means writing the data to primary storage & the replica simultaneously while asynchronous replication copy the data to the replica after the data is written to the primary storage) & it creates exact copy of our database & this can only be read from. Let's say our main DB has lots of queries happening & have high workload. So we can change the connection string (we get 3 connection string, 1 for the master & 2 for the replicas) of the 3 different EC2 instances to the new replicated RDS instances. We can have Read-Replicas of Read-Replicas (copies of copied RDS instances) & we can have 5 copies (Read Replicas) of our main database. Use case can be as testing or development environment purposes of main DB keeping the production environment untouched.



Let us think of an application and we have an RDS instance. Our application perform reads and writes to our DB instance. Let's suppose we want to scale the reads as our main DB instance cannot scale enough and it receives too many requests. Say, we've 2 more RDS instance read replica and there'll be asynchronous replication between main RDS instance & 2 read replicas.

Use Case



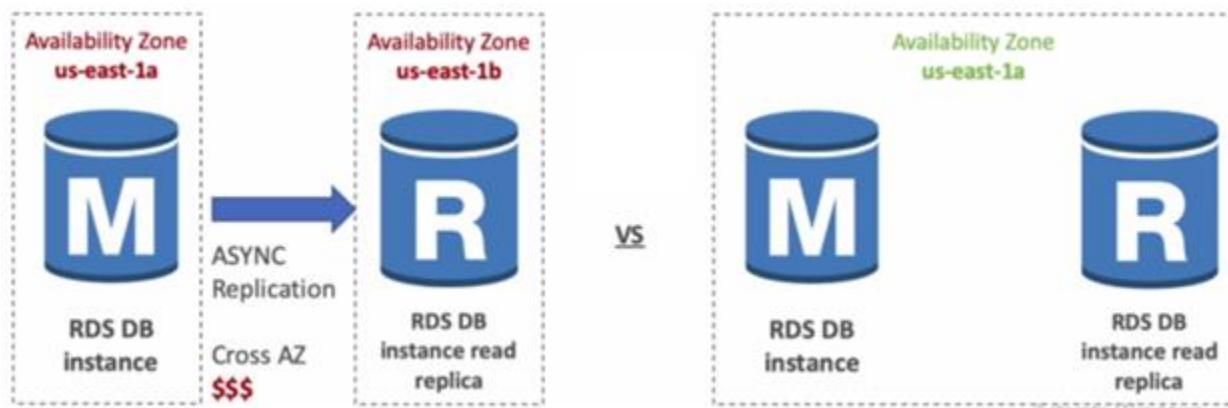
Let's suppose we have a production database, and it is taking on a normal load. Our production DB is having read & write to our main RDS DB instance. A new team comes and says that they want to run some reporting and analytics on top of our data. If we are going to plug in the application on our main instance, then we might overload our DB and possibly slow down the prod application. So, we'll

create a Read Replica to run our new workload there. So our reporting application will read from Read Replica and run analytics there.

The database supported for Read-Replicas are:

1. MySQL
2. PostgreSQL
3. MariaDB

Network Cost



- In AWS, there's network cost when data goes from one AZ to another.
- As we'll replicate data from us-east-1a to us-eas-1b, asynchronously, then it'll cost lot of money. But if we want to reduce the cost, we'll keep our data within same AZ (network cost is free).

Important point to remember:

- We can have up to 5 Read Replicas.
- The Read Replicas can be within the same AZs, Cross AZs or Cross region.
- Read Replicas are used for scaling not for Disaster recovery.
- Read Replica are read only, we can't write to Read-Replica.
- Automatic backups must be turned on in order to deploy read replica.
- We can have read replicas of read replicas but since it is asynchronous replication, it creates a latency because it is going to sync over read replica & that is going to sync over to read replica to next read replica.
- Each Read Replica is going to have its own DNS endpoint.
- We can't have Read Replica which have multi AZ turned on, i.e., if we create a Read Replica it should not be multi AZ copy, it's not going to be 2

copy of Read Replica. It's going to be single copy & it is going to be in whatever AZ we choose.

- We can create Read Replicas of Multi AZ source DB.
- Read Replicas can be promoted to their own DB. This breaks their own replication. So, we can take one of these replicas and we want to be its own database now and take rights. Now, this replicated DB can be written. Use case can be if we are using a replicated DB for testing or development purpose, & we want to write something to it now.
- Application must update the connection string to leverage read replicas. We can have Read Replica in a second region for MySQL & MariaDB but not for PostgreSQL or other DB.

Steps to create a Read Replica:

1. Go to RDS under Services.
2. Click on database & select your Database.
3. Click on Actions & select “Create Read Replica”. Here we can change the DB instance class (size of DB), storage type, AZ, etc.
4. Give your DB Instance identifier a name and for now leaving all the settings to default.
5. Hit ‘Create Read Replica’

Note: DynamoDB offers “push button” scaling, meaning that we can scale our DB on a fly, without any downtime but in RDS, it is not so easy & we usually have to use a bigger instance size or to add a read replica.

LAB

Note:

- DynamoDB is Amazon’s NoSQL database. (Knowing about DynamoDB is imp.)
- Redshift is Amazon’s Data Warehouse. It is used for online analytics processing. The use case is like if we’re running really complex queries & we don’t want to run on our production DB. We basically put a copy of our production DS & put it RedShift cluster & we hammer our RedShift cluster. So, our production DB does not get load performance hits.

1. Go to Databases under Services & click on RDS.
2. Click on “Create database”
3. Click on Standard Create select MySQL DB for now.
4. Do the credential settings.
5. Select the DB instance size.
6. Select storage type & your allocated storage.
7. Create a new security group for your DB and just assign a name for now, it'll be unique across region.
8. Configure your connectivity settings. Make the DB publically accessible for now, as we are going to connect through SQL Electron).
9. With other option set to default, launch your DB.
10. Click on your DB after it's created.

Note: We can use a DB client - SQLElectron to run small DB programs. We need to install it on our computer.

11. Go to Security settings and select your security group & edit its inbound rules.
12. Open SQLElectron, give it a name, DB type, Server Address (we can find it under Connect Block with the endpoint) and click on Test.
13. Click on the IP address, and after removing add your security group (like MyWebDMZ just by typing the sg, it will show all the names of the security groups).
14. Create a new EC2 instance with default configuration with a bash script.

```
#!/bin/bash
yum install httpd php mysql -y
yum update -y
chkconfig httpd on
service httpd start
echo "<?php phpinfo(); ?>" > /var/www/html/index.php
cd /var/www/html
wget https://s3.eu-west-2.amazonaws.com/acloudguru-example/connect.php
```

15. Leave everything as default and Review & Launch your instance.
16. Login into Putty using SSH

17. Go to your instance IP address & later on IP.address/connect.php
 18. Go to your RDS dashboard & copy your Endpoint
 19. In Putty, open your connect.php file using command nano connect.php
 20. Update the tags with your credentials and save & reload the connect.php page.
 21. To create a read Replica, go to 'Actions' in 'Instances' Section under RDS.
 22. Click on Create Read Replica.
- Note: For RDS, port 3306 should be open.

RDS security

1. RDS DB are usually deployed within a private subnet, not in a public one.
2. RDS security works by leveraging security groups (the same concept as for EC2 instances) – it controls which IP can communicate with RDS.
3. We can attach IAM policies who can manage AWS RDS using RDS API like who can create DB, delete DB or create Read Replicas.
4. Traditional username & password can be used to login to the database or we can also enable it using IAM users & it can be used for MySQL/Aurora.
5. IAM based authentication can be used to login into RDS MySQL & PostgreSQL.

Encryption

- Encryption at rest is supported for MySQL, Oracle, SQL server, PostgreSQL & MariaDB.
- Encryption is done using AWS KMS– AES 256 encryption.
- Encryption has to be defined during launch time.
- If the master is not encrypted, the Read Replicas cannot be encrypted. Once our RDS instance is encrypted, the data stored at REST in the underlying storage is encrypted, as are its automated backups, read replicas, and snapshots.
- We can also use SSL certificates to encrypt data in RDS in flight (over the network). For this, we need to provide SSL options with trust certificate when connecting to Database.
- We can enable Transparent Data Encryption (TDE) available for Oracle and SQL server.

- Snapshots of un-encrypted RDS DB are unencrypted.
- Snapshots of encrypted RDS are encrypted.
- We can copy a snapshot of an unencrypted and enable encryption to make it into an encrypted one. Then we can restore the DB from the encrypted snapshot and finally migrate applications to the new DB and delete the old DB.

At the present time, encrypting an existing DB instance is not supported. To use Amazon RDS encryption for an existing DB, create a new DB instance with encryption enabled & migrate your data into it.

To enforce SSL in:

1. PostgreSQL: Run `rds.force_ssl=1` in the RDS console.
2. MySQL: Within the DB: Grant usage on *.* TO 'mysqluser'@% Require SSL;

To connect using SSL:

1. Provide the SSL trust certificates.
2. Provide SSL options when connecting to the database.

For manual restore:

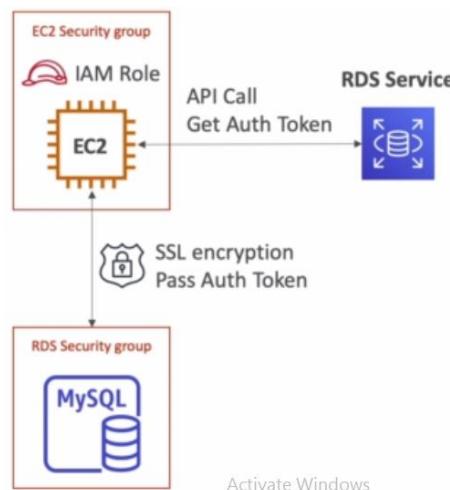
1. Go to RDS under services.
2. Click on Snapshots.
3. Select the snapshot to be restored.
4. Give your instance a name and check all other options & restore.

For point in time restore:

1. Go to RDS under services.
2. Click on 'Databases'.
3. Select your DB & choose "Restore to point in time" under 'Actions' tab.
4. We have again an option to change our DB instance class, Multi AZ deployment, etc.
5. The technique mentioned in step 4 is a way to increase the size of RDS instance or scaling up RDS instance. Take a snapshot of your DB & later change the instance class.

IAM Authentication

- IAM DB authentication works with MySQL and PostgreSQL.
- For authentication, we don't need a password just an authentication token obtained through IAM & RDS API calls.
- Auth tokens has a lifetime of 15 minutes.



We have our EC2 Security group and EC2 instance. We have a MySQL DB in other security group. EC2 instance will have an IAM Role issuing API calls to RDS service to get an Authentication token. It passes the token while connecting securely to MySQL DB. It makes sour that Network IN/OUT must be encrypted using SSL. IAM manages users centrally rather than managing users from DB.

Points to Remember

It is our responsibility to:

- Check the ports /IP /Security group's inbound rules in DB's Security groups.
- Take care of In-Database user creation & permission or manage through IAM.
- Creating a DB with/without public access that whether it is going to be in a public or a private subnet.
- We need to ensure parameter groups or DB configured to allow only SSL connections, making sure encryption is happening.

AWS responsibility:

- There's no SSH Access.

- No manual DB patching.
- No manual OS patching.
- There's no way to look at the underlying instances

DynamoDB

It is a fast & flexible NoSQL database service for all application that need consistent, single digit millisecond latency at any scale. It is a fully managed database & supports both document and key-value data models. Its flexible data model & reliable performance make it a great fit for mobile, gaming, ad-tech, IoT, and many other applications.

- Stored on SSD storage.
- Spread across 3 geographically data centers.
- Eventual consistent reads (Consistency across all copies of data is usually reached within a second. Repeating a read after a short time should return the updated data – Best Read performance).
- Strongly consistent reads (returns a result that reflects all writes that received a successful response prior to the read).

Use case to understand the above 2 points: If we've an application that goes to write something to DynamoDB table, does it need the information to be immediately available within less than a second, if so we are going to need "Strongly consistent reads". If it can wait to propagate that write to the 3 different data centre, or it can wait for a second before that data can be read we can move with the "Eventual consistent reads" model.

Pricing

- Provisioned through capacity.
 - Write throughput of \$0.0065 per unit for every 10 units.
 - Read throughput of \$0.0065 per unit for every 50 units.
- Storage cost of \$0.25 GB per month.

Pricing example

Let's assume our application needs to perform 1 million writes & 1 million reads per day, while storing 3 GB of data.

First we need to calculate, how many reads & writes we need per second. 1 million can be averaged to $1,000,000 / 24 \text{ hours} / 60 \text{ minutes} / 60 \text{ seconds} = 11.6$ reads or write per second.

A DynamoDB Write capacity unit can handle 1 read & write per second, so we need 12 Read & Write Capacity Units.

With Read Capacity units, we're billed in blocks of 50 & with write capacity units we're billed with blocks of 10.

To calculate Write Capacity units = $(0.0065 / 10) * 12 * 24 = \0.1872

To calculate Read Capacity units = $(0.0065 / 50) * 12 * 24 = \0.0374

Real time use case: DynamoDB can be expensive for Writes but it's extremely cheap for Reads. If we've DB or production environment, that have lots of reads & very little writes & it needs to be really scalable & need to have a good performance & if it does not need any SQL joint Query & can run on NoSQL database, then DynamoDB can be a good choice.

Setting up DynamoDB table

1. Go to DynamoDB under 'Database' section in services.
2. Click on "Create Table".
3. Give table a name & set its primary key.
4. Use default settings for now & hit "create".
5. Let's add more item to our table, go to 'Items' & create an item (item is basically a Row in a Table).
6. Click on the '+' button to open action menu.
7. Let's append a String (say the field be 'FirstName' with any value of your choice).
8. Click on 'Save'.
9. Similarly, create a second column with another set of data.
10. Metrics can used the logs of our DB. We can watch our Read & Write provisioned & consumed units. If consumed goes greater than provisioned units, we can add units on the fly under 'Capacity' section & this is the benefit of DynamoDB. It also help us to pay only what we consume, so it's a good habit to monitor these metrics weekly.

11. So, comparing with RDS, DynamoDB can be scaled on fly whereas RDS has a downtime & in RDS we first take a snapshot, then we need to increase the instance size.

Note: Reserved capacity is a billing feature that allows you to obtain discounts on our provisioned throughput capacity in exchange for one-time upfront payment & committed to a minimum monthly usage level. Reserved capacity applies within a single AWS region & can be purchased for 1-year or 3-year term.

RedShift

Amazon RedShift is a fast, fully managed, petabyte-scale data warehouse service in the cloud. Customers can start for just \$0.25 per hour with no commitments or upfront costs & scale to a petabyte or more for \$1000 terabyte per year, less than one tenth of most other data warehousing solutions.

Let us have a look back to an example of OLAP that we took under Data warehousing (Pg. 71). It was making lots of queries to calculate Net profit (basically in deep, it was querying too many columns to get the result). So, data warehousing is adding all about columns, it's not about individual tables in a record set. It's all about adding whole bunch of columns. Data warehousing databases use different type of architecture both from a database perspective & infrastructure layer.

RedShift Configuration

- Single Node (can have 160GB of data)
- Multi-Node (let's suppose we need to scale up from single node)
 - Multi-Node configuration consists of a Leader Node (it manages client connection & receives queries) &
 - Compute Node (it stores & performs queries & computations). It has up to 128 compute nodes.

RedShift – 10 times faster

Columnar data storage: Instead of storing data as a series of rows, Amazon Redshift organizes data by columns. Unlike row-based systems, which are ideal

for transaction processing, column based system which are ideal for data warehousing and analytics, where queries often involve aggregate performed over large data sets. Since only, the columns involved in the queries are processed & columnar data is stored sequentially on storage media, column based media require far fewer I/O's, gently improving query performance.

It is faster because Storage systems has to pull the data out from physical disks drives, which stores information magnetically on spinning platters using read/write heads that move around to find the data that users request. The less the heads to move, the faster the drives performs, minimizing seek time, and system can deliver that data faster.

Advanced compression: Columnar data can be compressed much more than row-based data stores because similar data is stored sequentially on disk. Let us understand this by an example, if we've got your table in which rows are like FirstName (string), LastName (string), ID (numeric), date, etc. having different data type & in order to compress a row with all different data types, we're not going to get a really good compression. So, if we start a compression based on columns, i.e., just dates, just ID (numbers), name (varchar), we can compress that data a lot better.

Amazon RedShift employs multiple compression techniques & can often achieve significant compression relative to traditional data stores. In addition, Amazon RedShift does not require indexes or materialized views and so use less space than traditional database systems. When loading data into an empty table, Amazon RedShift automatically samples our data & selects most appropriate compression scheme.

Massive parallel processing (MPP)

Amazon RedShift automatically distributes data and query across all nodes. It makes it easy to add nodes to our data warehouse & enables us to maintain fast query performance as our data warehouse grows.

RedShift Pricing

The pricing depends upon:

1. Compute Node Hours (total number of hours we run across all our compute nodes for the billing period. We're billed for 1 unit per node per hour, so a 3 node data warehouse cluster persistently for an entire months would incur 2,160 instance hours. $3 \text{ (nodes)} * 24 \text{ (hours)} * 30 \text{ (days)}$. We're not charged for leader node hours, only compute node will incur charges).
2. Backup
3. Data transfer (only within a VPC, not outside it).

RedShift Security

- Everything that is communicated to RedShift is transit using SSL.
- It is also encrypted at REST using AES-256 encryption.
- By default, RedShift takes care of the key management for us.
 - However, we can manage our own keys through Hardware Security Modules (HSM) &
 - Using AWS KMS.

RedShift availability

- It is only available in 1 AZ (since RedShift is not used in Production environment, it is used in Business Intelligence where we can run our reports & queries).
- Can restore snapshots to new AZ's in the event of an outage (period when power supply or other service is not available).

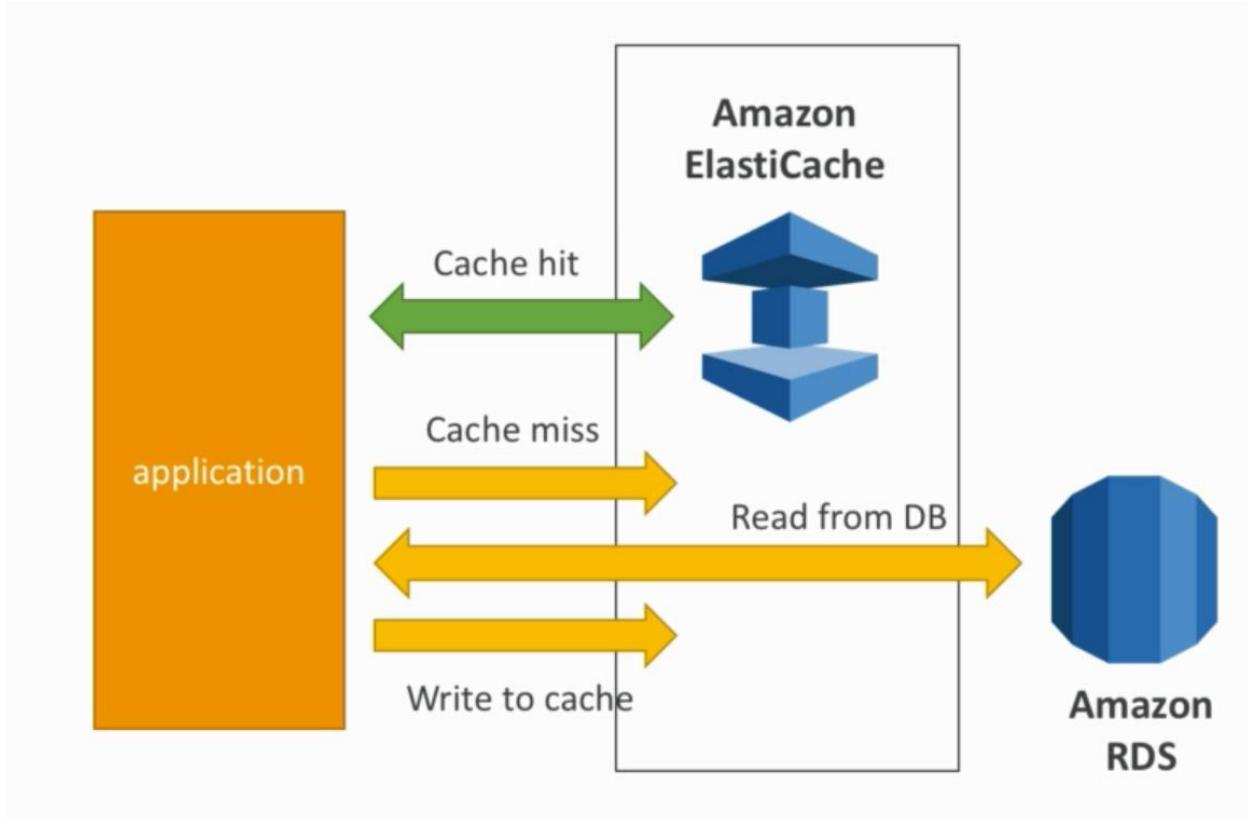
ElastiCache

(Imp)

- It is a web service that makes it easy to deploy, operate, & scale-in memory cache in the cloud.
- The service improves the performance of the web application by allowing us to retrieve the information from fast, managed, in-memory caches, instead of relying entirely on slower-disk based database. In one word, we can say ElastiCache caches things.
- Let's take an example: We've an online store where we sell cars, a particular sports car is searched more. So instead of making the frontend

querying the DB many time, we can cache it up & reduce the load on our DB. Hence instead of running the query every time, we can cache the information so that our web server gets it from Elastic cache rather than getting it from the database server & this frees our database server to return more valuable information.

- It can be used to significantly improve latency & throughput for many read heavy application workloads (such as social networking, gaming, media sharing, & Q&A portals) or compute intensive workloads (such as recommendation engine).
- Caching improves application performance by storing critical pieces of data in memory for low-latency access. Cached information may include the result of I/O intensive database queries or the results of computationally intensive applications.
- Helps making an application stateless by storing the state.
- Write scaling using Sharding.
- Read Scaling using Read Replicas.
- It has multi AZ with failover capability.
- DB cache helps release loads in RDS.
- Cache must have a validation strategy to make sure only the most current data is in the cache.



Our application communicates to RDS. Application queries ElastiCache, if not available, get it from RDS & stores it in ElastiCache. It is termed as cache hit when we get into ElastiCache and it works. Now, we've an application, it has a cache hit and we get a data from ElastiCache. Sometimes application request data, but it doesn't exist. This way it's a cache miss. In this case, here our application needs to query dB. Our application should be programmed such as it writes back to the cache. Also, cache must come with a validation strategy to make sure only the most current data is used in the cache. And the idea is that, if same/another application asks for the same query, this time it'll be a cache hit.

Types

1. Memcached: It is an in-memory object store & widely adopted memory caching system. ElastiCache is protocol compliant with Memcached, so popular tool that we use today with existing Memcached environment will work seamlessly with the service. It is a multi-node for partitioning of data (Sharding). If memcached node goes down, then the data is lost. There's no backup/restore feature. It is a multi-threaded architecture, so a part of data

is going to be on one shard and another part of the cache is going to be on another shard.

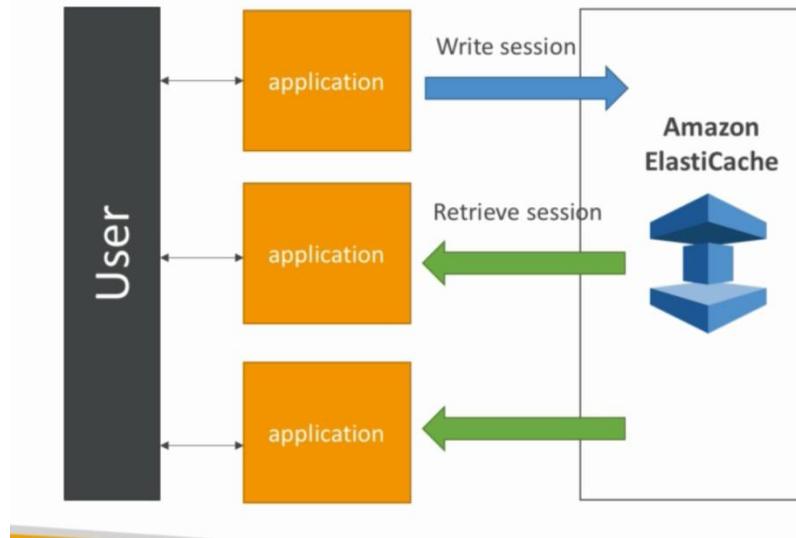


- a. Cache doesn't survive the reboots.
- b. Use cases:
 - i. Quick retrieval of objects from memory.
 - ii. Cache often accessed objects.
- 2. Redis: It is more popular than Memcached & is in-memory key-value store that supports data structure such as sorted sets & lists.



- a. It has multi AZ feature with auto-failover means if one AZ fail, we can failover automatically to another one.
- b. We can enhance Read scales by creating Read Replicas and hence we have high Reads and High availability.
- c. We can backup and restore our Redis Clusters.
- d. It has super low latency (sub milliseconds).
- e. Cache survive reboots by defaults (persistence)
- f. Publish/Subscribe capability for messaging.
- g. ElastiCache supports Master/Slave replication which can be used to achieve cross AZ redundancy.
- h. Support for Read Replicas.

User Session store



Let's suppose user logs into any of the application & the application writes the session data into ElastiCache. Now, the user hits another application and now the application needs to know our user is logged in & this way all the instances can retrieve the data & user doesn't have to re-login. This helps to relieve the load of the database & to share some state so the user session is stored into a common ground in which the application can be stateless & retrieve and write the application in real time.

LAB

1. Go to ElastiCache under Services.
2. Click on Get Started now.
3. We will select our cluster engine to be Redis for now.
4. Give it a name and description.
5. Select version compatibility to be latest.
6. The standard port for Redis is 6379.
7. Select the parameter group to be default.
8. Select the node type to be t2.micro.
9. Set the number of Replicas to be zero. If we select it two be say 2, then there's Multi-AZ with Auto-Failover option.
10. We need to create a subnet group, create a new one and give it a name.
11. Select either of the subnets.

12. We can set an authentication token and this will be needed by application to connect to Redis.
13. We can Enable automatic backups and it cannot be enabled for Memcached.
14. Click on Create.

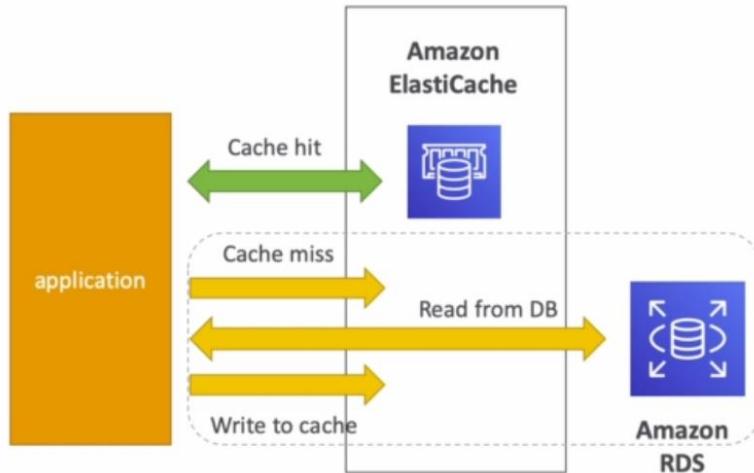
Points to remember

- ElastiCache is a good choice if our database is particularly read heavy and not prone to frequent changing.
- RedShift is a good choice if our database is feeling stress because management keep running OLAP transactions on it etc.
- Elastic help make our application stateless.
- It has write scaling option using Sharding (it is a type of database partitioning that separates very large database into smaller, faster & more easily managed parts called data shards) & read scaling using Read replicas.
- Multi AZ with failover capability.
- AWS takes care of OS maintenance/patching, optimizations, setup, configurations, monitoring, failure recovery & backups.
- Redis have more industrial RDS type feature while Memcached is going to be pure cache that's going to live in memory.
- Since it has persistence, we can use it as a DB, cache, and message broker.

Caching implement consideration

- Caching is not for every type of Dataset. Data may be out of date, so we may have eventually (after a long delay) consistency.
- If our data is changing slowly, and only few keys are frequently needed, caching is effective. But if our pattern is changing rapidly, and we need all the key space in our data set then caching is not effective.
- The data should be well structured for returning the queries to reduce response time so that caching can happen effectively.

Lazy Loading/Cache-Aside/Lazy Population



If our application wants to read something, it goes to ElastiCache and if returns result, it is called a Cache Hit. If there's no result, it's called a Cache miss. Hence we read the from DB (Amazon RDS). Now the application has the correct data and it goes to write the data to the cache to make sure if other application who request the same data will go to directly to the Cache hit.

Pros

1. Only requested data is cached (the cache isn't filled up with unused data).
2. In case the cache gets wiped or Node failures are not fatal, then the latency is going to be increased because the Reads needs to go to RDS and then be cahed.

Cons

1. Cache miss penalty that results in 3 round trips, noticeable delay for the request.
2. Stale data: Data can be updated in the DB and outdate in the cache.

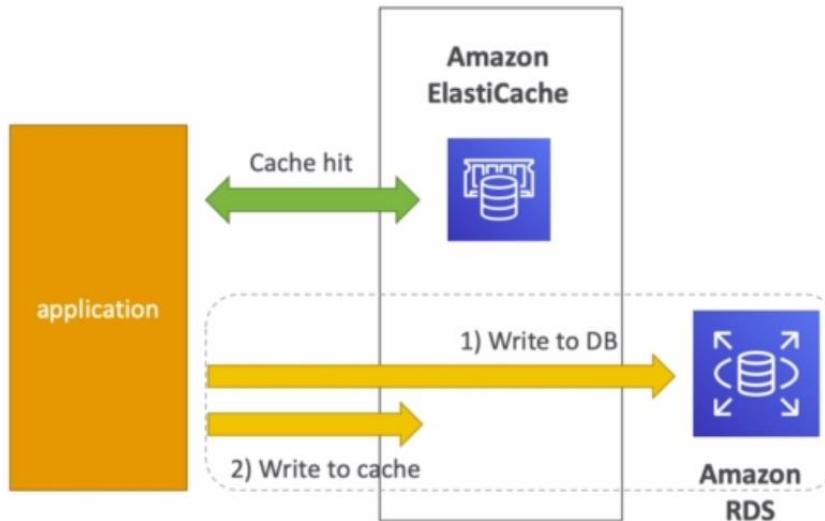
Python Pseudocode

```

1  # Python
2
3  def get_user(user_id):
4      # Check the cache
5      record = cache.get(user_id)
6
7      if record is None:
8          # Run a DB query
9          record = db.query("select * from users where id = ?", user_id)
10         # Populate the cache
11         cache.set(user_id, record)
12         return record
13     else:
14         return record
15
16 # App code
17 user = get_user(17)

```

Write Through



We Add or Update the cache when database is updated. If our application wants to read something, it goes to ElastiCache and if returns result, it is called a Cache Hit. When there is a write happening to RDS, when our application modifies the Amazon RDS then the application writes to the cache.

Pros

1. Data in cache is never stale, Reads are quick. Whenever there's a change in RDS, there's a change in cache.
2. We get a Write penalty this time when we need to write to the DB as each write require 2 calls, one to the DB & other one to the Cache. Last time we

need to do 3 network calls in total in Read Penalty. And it takes more time to Read than to Write.

Cons

1. There's a missing data unless the database is updated or added. We can implement Lazy Loading i.e., if we do not find the data in the Cache we can look in the Database.
2. We get a Cache-churn. If we a lot of Data in RDS, there's going to be lot of data in ElastiCache but there's a chance that this data is never going to be read. Hence this can be problematic when the cache is too small.

Python Pseudocode

```

1  # Python
2
3  def save_user(user_id, values):
4
5      # Save to DB
6
7      record = db.query("update users ... where id = ?", user_id, values)
8
9      # Push into cache
10
11     cache.set(user_id, record)
12
13     return record
14
15 # App code
16
17 user = save_user(17, {"name": "Nate Dogg"})

```

Activate Ven

Cache Evictions and Time-to-Live

Our cache has limited size, hence cache eviction removes the data out of our cache. Cache eviction can occur in 3 ways:

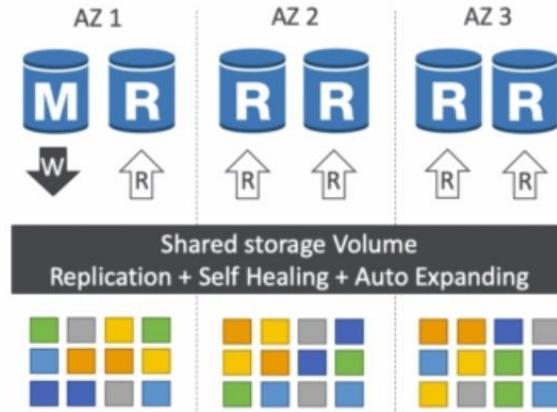
1. We can delete item explicitly in the cache.
2. Item is evicted because the memory is full and it is not recently used (LRU - Least Recently Used).
3. We can set Time-To-Live saying that the cache can live for 5 minutes.
4. TTL can range up to from seconds to hours or days.
5. If too many evictions happen due to memory, then we can update the cache size.

Aurora

- It is a database engine that is available in RDS.
- It is not open sourced as it is a proprietary technology from AWS.
- It runs only on AWS infrastructure not like MySQL which runs like can be locally installed on our device. It is PostgreSQL, MySQL compatible which means our drivers will work as if Aurora was a PostgreSQL or MySQL database). We can run the databases which we were previously running on MySQL on Aurora databases.
- It is a MySQL compatible, relational database engine that combines the speed & availability of high-end commercial databases with the simplicity & cost-effectiveness of open source databases.
- Amazon Aurora database provides up to 5 times better performance than MySQL at a price point one tenth that of a commercial database while delivering similar performance & availability. It also provides 3x performance of PostgreSQL on RDS.

Aurora Scaling

- Starts with 10 GB, Scales in 10 GB increment to 64 TB.
- Compute resources can scale up to 32vCPUs & 244GB of memory.
- It does have a downtime while scaling. We do scaling using a maintenance window. Though it is quite fast in terms of scaling.
- Aurora maintains 2 copies of our data in each AZ with minimum of 3 AZ. So, there are 6 copies of our data.
- Aurora is designed to transparently handle the loss of up to 2 copies of data without affecting the database write availability (i.e., if one AZ is down, it is fine) & 3 copies of data without affecting read capability.
- Aurora storage is also self-healing. Data blocks and disks are continuously scanned for errors & repaired automatically.
- Storage is striped across 100s of volume.



We've 3 AZs which have a shared storage volume. It has replication, self-healing and auto expanding. Like if we write any data (any of the colored blocks), we can see 6 different copies in 3 AZ. There's only one instance that takes WRITE. If the master does not work, the failover will occur in less than 30 seconds on average. On top of the Master, we can have 15 Aurora Read Replicas serve reads and any of these Read Replicas can become the master if Master fails. These Read Replicas supports Cross-Region Replication.

Aurora Replicas

There are 2 types of replicas available:

1. Aurora replicas (these are separate Aurora DB from our original Aurora DB & we can have up to 15 Aurora Replicas)
2. MySQL read replicas (we can have up to 5 MySQL read replicas)

In case we lose our primary Aurora database, failover automatically occur to our Aurora replica but it does not occur to our Aurora Read Replica. Failover in Aurora is instantaneous. We get high availability on Aurora.

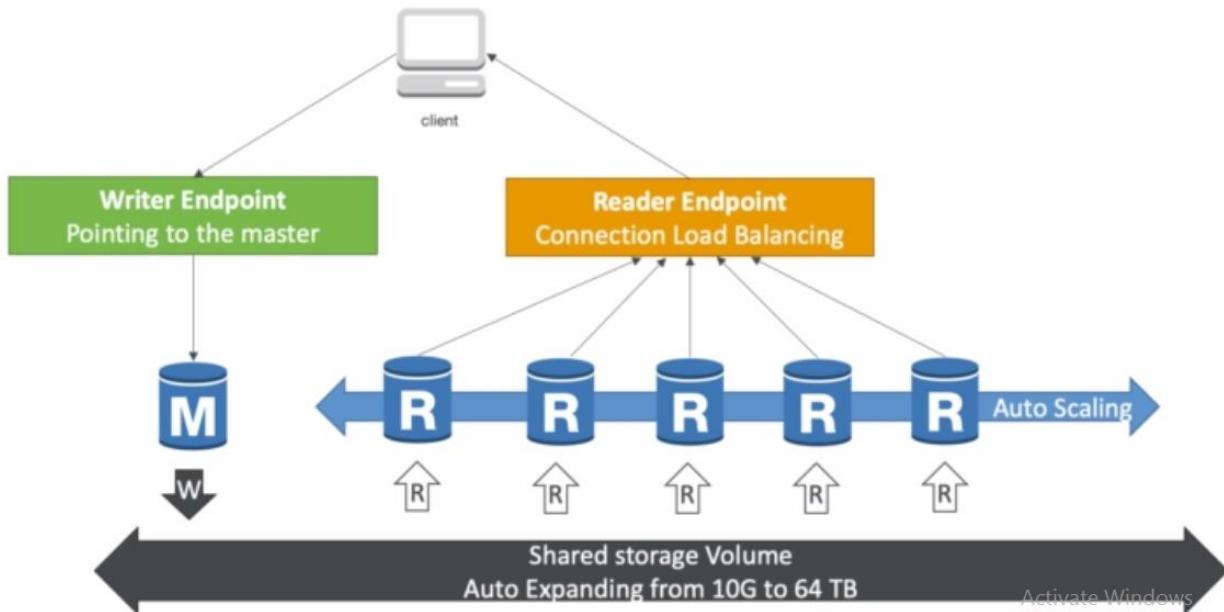
Note: these are quite largely paid instance cost about 5-10\$ (costs 20% more than RDS), so always take a note that these instance are not left running if not in use.

1. Steps to create Aurora database:
2. Go to services and go to RDS
3. Go to database and create one.
4. Select Amazon Aurora
5. The DB size now starts with r4.large, no t2 micro.

6. Under additional configuration, select the failover priority and we can have up to 15 Aurora replica. If we select tier-0 as failover priority, it'll have highest priority.
7. Give your cluster identifier & database your name.
8. Leave everything as default & launch the instance.
9. We can see that Multi-AZ is set to 'No'.
10. To allow failover, create aurora replica by selecting the database and clicking on Instance Actions.
11. Select instance size & your source DB name.
12. Give DB instance source identifier a name.
13. Leave everything as default & create your Aurora replica.
14. This replication role is a reader & we can only read from the DB.
15. The connection string can be found out under "cluster endpoint" section.
16. Now, we can see that Multi-AZ is set to "2 zones".
17. If the original DB instance fails, we don't need to update the connection string in the file/program wherever it is used. Amazon replaces the connection string with the connection string of the replica DB.

DB Cluster

It defines how Aurora works when we have clients. How we interface with all these instances.



We have a shared storage volume (Auto expanding from 10GB to 64 TB). The master writes to the storage, and since the master can write & failover so Aurora provides us with a writer Endpoint which always points to Master. Even if our master fails, the writer Endpoint automatically writes to the right instance. we can have auto-scaling on top of these Read Replicas (up to 15). Since we've auto-scaling, it can be tough for our application to keep track of these Read Replicas. How to connect to them or their URLs. We've a Reader Endpoint and it has the same feature as Writer Endpoint. It helps in connection Load Balancing and it connects automatically to all the Read Replicas. So, anytime the client connects to the Reader Endpoint, it will get connected to Read Replicas and Load Balancing will be done this way. We need to make sure that Load balancing occurs at connection level, not at statement level.

Features

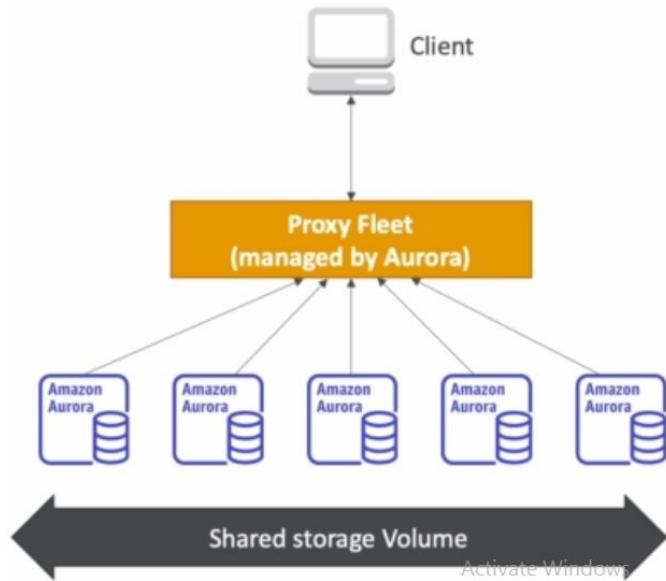
1. Automatic fail-over.
2. Backup and Recovery.
3. Isolation and security.
4. Industry Compliance.
5. Push-button scaling.
6. Automated patching with zero downtime.
7. Advanced monitoring.
8. Routine maintenance.
9. Backtrack: restore data at any point of time without relying on backups.
Like we say we want to go back at 4:00 P.M, it works as well.

Security

- Similar to RDS, since it is similar to same engine.
- Encryption at REST using KMS.
- Automated backups, Replicas, Snapshots that're already encrypted.
- Encryption in flight using SSL (enforcing like in MySQL and Postgres).
- Possibility to authenticate using IAM token (same method as RDS).
- We need to manage the security group for protecting the instance.
- We cannot SSH to our instance.

Serverless

- It is an automated instantiation and auto-scaling based on actual usage.
 - It is good for infrequent, intermittent, or unpredictable workloads.
 - No capacity planning needed.
 - We pay per second, and hence can be more cost-effective.



We've a shared storage volume and a client who wants to access our Client's DB but it is serverless. In the backend, there's going to be an Amazon Aurora created by Aurora Serverless and there's a proxy fleet managed by Aurora that our client will connect to and then will transfer to our Amazon Aurora Database. When we get more load, more Amazon databases will be created for us and if we don't have any usage, it will scale down to zero.

Global Aurora

- The first way is to have Aurora Cross Region Read Replicas.
 - They're useful for disaster recovery.
 - Simple to put in place.
 - There's another way to do so by using Aurora Global Databases (recommended):
 - We've 1 primary region where all the Reads & Writes happen.
 - We've up to Read only regions where the replication lag is going to be less than a second.
 - We can have up to 16 Read Replicas per secondary region.

- Helps in decreasing latency.
- If we want to promote any another region for recovery since main region has massive outage, then we've an Recovery Time Objective of 1 minute. It means in less than a minute, our secondary DB will become primary and will be ready to take on WRITES.



We've us-east-1 which is our primary region. There we've Aurora DB, and application does Read & Write to it. Then, we define a secondary region in us-east-1. Hence, we define an Aurora Global DB which performs replication up to 1 second lag (asynchronous replication). Now, our application from eu-east-1 can read from this DB.

LAB

1. Go to RDS under Services.
2. Select DB creation method to be standard.
3. Select engine to be Aurora.
4. Under Edition, we'll choose Aurora with MySQL compatibility.
5. We will select Database Location to be Regional for now.

6. Under Database features, we will select one writer and multiple readers. Serverless helps to scale automatically on the basis of workloads.
7. Choose the templates to be production.
8. Set the credentials.
9. Select the DB instance size as per requirement.
10. Under availability & durability, we can choose Multi-AZ deployment or Say no to it since Aurora itself is across multi-AZ.
11. Under connectivity, we can choose the Default VPC.
12. Under additional configuration, we can choose the DB instance identifier and the Initial DB name.
13. Backups helps to take snapshots and we can set retention period between 1 to 35 days.
14. Under maintenance, we can choose to enable Delete protection.
15. Hit on Create Database.

Note: We've a reader and writer endpoint to read & write DB and this is the preferred way rather connecting to the database and then connecting to instance's endpoint. To add auto-scaling to our DB, we'll go to Actions, and select "Add Replica auto scaling" to add elasticity. Select the Target value after which Aurora Replicas will be created automatically. We can also set the min and max capacity of our clusters and Add the policy.

10. VPC

- We can say VPC (Virtual private cloud) as a virtual data center in a AWS cloud that allows to deploy resources within it.
- Amazon VPC lets us provision logically isolated section of AWS cloud where we can launch AWS resources in a virtual network that we define. We've complete control over our virtual networking environment, including selection of our own IP address range, creation of subnets, & creation of route tables & network gateways.
- It is a region specific resource.

We can easily customize the network configuration for our Amazon Virtual Private Cloud. For e.g.: We can create a public-facing subnet for our web servers that has access to the internet, & place our backend system such as databases or

application servers in a private facing subnet with no Internet access. We can leverage multiple layers of security including security groups & network access control list, to help control access to Amazon EC2 instances in each subnet.

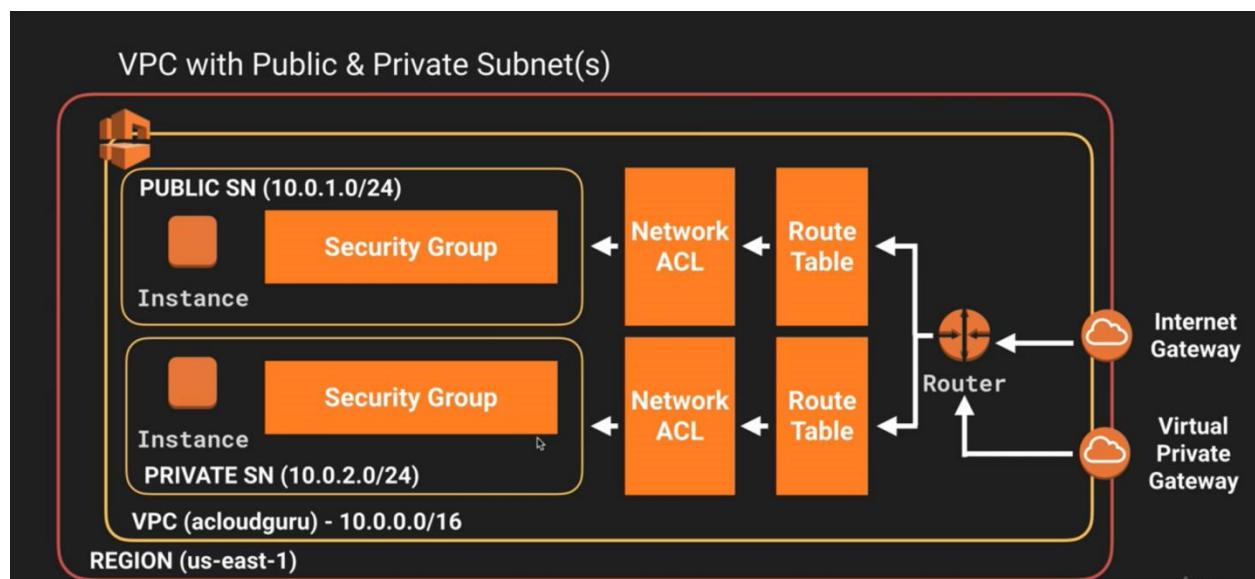
Also, we can create a Hardware Virtual Private Network (VPN) connection between our corporate datacenter & our VPC and purchase the AWS cloud as an extension of our corporate datacenter.

Public subnet usually contains:

1. Load balancers
2. Public authentication layers
3. Files
4. Static websites

Private subnets contains:

1. Web application servers
2. Databases

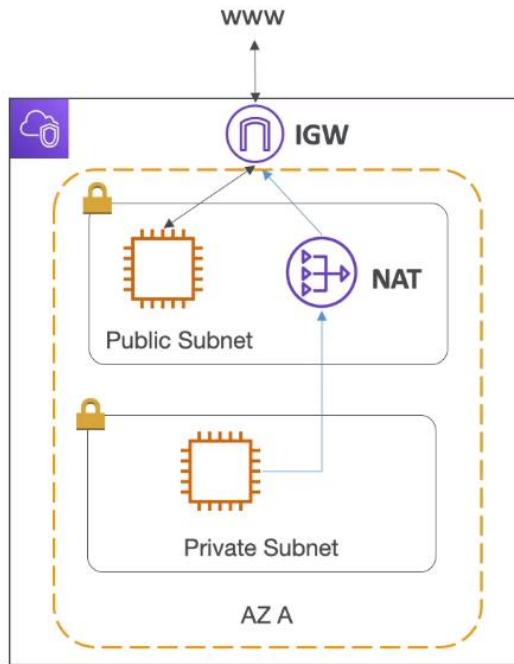


There are 2 ways we can connect into this VPC:

1. Internet Gateway (provides internet access)
2. Virtual Private Gateway (connection is established using VPN)

Now, the traffic is sent to a router. The traffic is sent through the Route Table (Route table is a data table that is connected to a router or network hosts that lists the routes to particular network destinations) through a network ACL (a layer of security that acts as a firewall for controlling traffic in & out of the subnet) to a private subnets (accessible via VPN) or public subnets (accessible via internet).

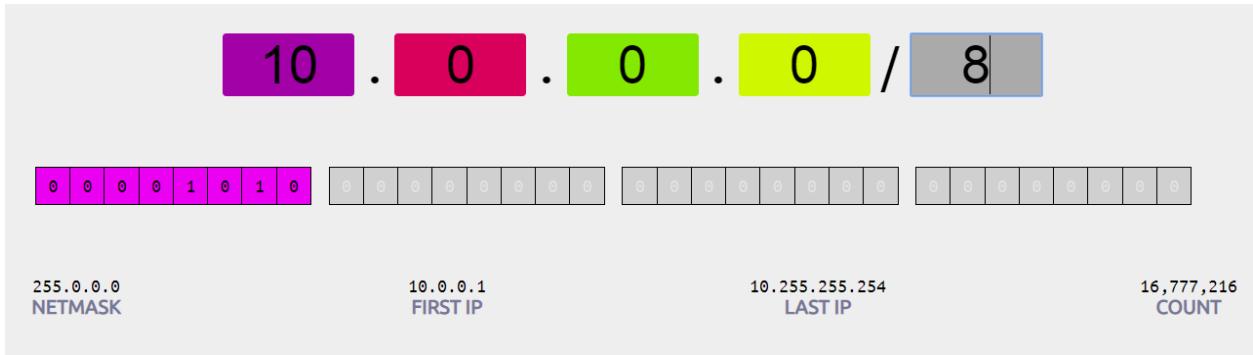
We can go into public VPN via internet & later SSH into private that's what Jump box or bastion service are. These subnets have security group which protect our instances.



Here we've 2 subnets, a public and a private one. Resources under public subnets are accessible through internet using Internet Gateways. Whereas Private subnet are not accessible to the internet. Nat gateways/instances are used to provide net and access to internet while remaining private. For that we deploy a NAT gateway/instance in a public subnet and then we create route from the private subnet to the NAT instance/gateway. Since the NAT has a route to Internet Gateway since it is in a public subnet and hence the private subnet can access through the NAT all the way to the internet.

Amazon let us use bunch of internal IP address ranges:

1. 10.255.255.255 (Prefix – 10/8) – this'll give the highest address range since '8' indicates first 8 blocks of the 32 blocks has been occupied & the IP address count limits to 1,67,77,216. The prefix highest limit which can be accepted in AWS is (10/28).



2. 172.16.0.0 – 172.31.255.255 (Prefix – 172.13/12) – this'll give mid address range.
3. 192.168.0.0 – 192.168.255.255 (Prefix – 192.168/16) – this'll give lowest address range.

Note: www.cidr.xyz gives an interactive IP address & CIDR range visualizer.

We can have multiple VPC inside a region.

Note: Networking has 2 sort of IPs: IPv4 & IPv6. IPv4 is still the most common one & it allows 3.7 billion different address in public space. IPv6 is newer & can solve problems for IoT.

By default, our EC2 instance comes with a private IP for internal AWS network & a public IP for access to the internet. We can't SSH using private IP because we are not on the same network.

Applications

1. We can launch instance of our choice using VPC.
2. Custom IP address ranges can be assigned in each subnet.
3. Route tables can be configured between subnets (like which subnets are allowed to speak to other subnets).
4. Create Internet gateway & attach it to VPC (note: we can have one internet gateway per VPC).
5. Much better security control over AWS because we use subnets to block IP addresses (Subnet network access control list). Instances can be moved to private subnets to stop people from accessing them.
6. Instance security groups can be deployed inside our VPC's

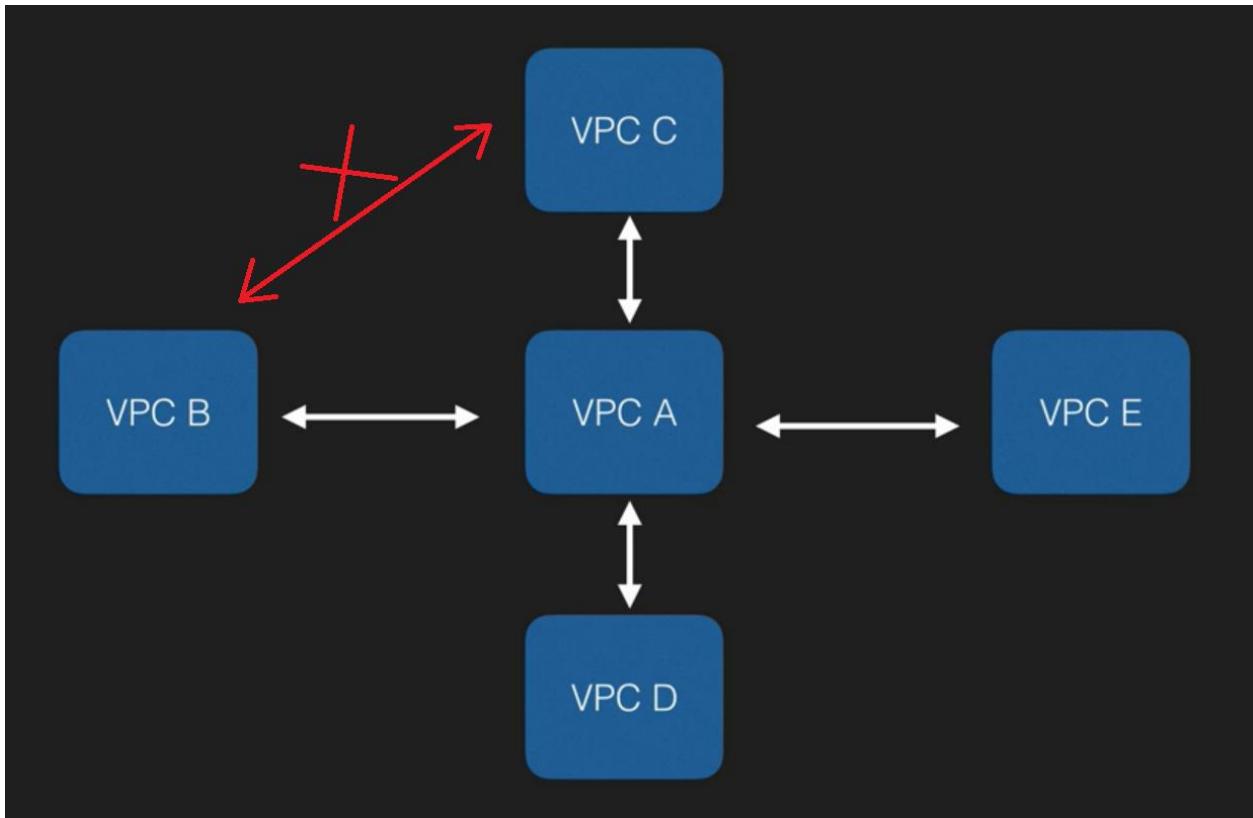
Default VPC vs Custom VPC

1. Default VPC is user friendly, allowing us to immediately deploy instances.

2. All subnets in default VPC have a route to internet.
3. Each instance has both a public & private IP address while a custom VPC does not have a public IP address.

VPC peering

1. It enables to establish connectivity between VPC & other structures.
2. Say, we've 2 different VPC in either 2 different accounts/regions and we want to connect them together privately as if they're a part of the same network.
3. To make sure, these VPC can be connected we need to make sure that the IP ranges defined in the VPC are different and not overlapping.
4. VPC Peering connection is not transitive (these must be established for each VPC that needs to communicate one another).
5. With 1 VPC, we can have multiple VPC's. We can peer one VPC to another, basically it allows us to connect one VPC with another via direct network route using private IP address.
6. Instances behave as if they are on same private network. Use case can be considered as we can communicate one instance inside a one subnet inside one VPC with another instance under another subnet under another VPC.
7. We can peer one VPC's with another AWS account as well as with other VPC's in the same account.
8. Peering is in a star configuration, i.e., 1 central VPC peers with 4 others. There's no transitive peering i.e., here, VPC C can communicate with VPC A but not with VPC C.



Note: Security group are stateful i.e., if we open an port in inbound rule, that port opens up in outbound rule automatically, but Network Access Control List (Network ACL) are stateless i.e., if we've to open both inbound & outbound port.

LAB

1. Go to VPC under ‘Networking & Content delivery’.
2. We can see the Subnets, Route tables, Network ACL, Security Groups created by default under our availability zones.
3. Click on “Create VPC” & give it a name & a CIDR range.
4. Select either of the option to work with IPv6 CIDR block to work with IPv6.
5. Select the tenancy, whether we want to work with dedicated hardware & these will not be used by any of the AWS customers or select ‘Default’.
6. Hit ‘Create’.
7. As soon as we create a VPC, we can see a route table, Network Access Control Lists (Network ACL), Security group.

8. To create a VPC structure, we need to create subnet (refer to the above diagram, since it is not being created by default).
9. So now, create a subnet which include a ‘Name tag’, select the VPC which we created in step 6.
10. Select the availability zone & range of IPv4 CIDR block.
11. Hit ‘Create’.

Note: when we look available IPv4 section of the ‘Subnets’, we see there’s only 251 available IP addresses but according www.cidr.xyz, there should be 256. The reason for this is because the first 4 & last IP addresses is not available for our use and cannot be assigned to an instance.

- a. 10.0.0.0 is reserved by the network address.
 - b. 10.0.0.1 is reserved by the AWS for VPC router.
 - c. 10.0.0.1 is reserved by AWS for DNS.
 - d. 10.0.0.3 is reserved by AWS for future use.
 - e. 10.0.0.255 is reserved for network broadcast address.
12. Create another subnet in a different availability zone.
 13. Referring to the above diagram, we’ve to now add an internet connectivity.
 14. Go to “Internet Gateways” under VPC dashboard.
 15. Give it a name & hit on ‘Create’. We can see that it shows state as ‘Detached’.
 16. So, select your internet gateway & click on ‘Actions’ & under that click on “Attach to VPC”. Select the VPC created in Step 6.

Note: we can attach single VPC to multiple Internet Gateway.

17. Go to Route Tables.
18. Create one route table & add internet gateway to it by selecting the route table & then click on ‘Routes’ & edit a route ‘0.0.0.0/0’ & target should be “Internet Gateway”. This’ll allow all internet traffic that comes within this route will be a public subnet.
19. To add a route out IPv6, go to edit route & add a route ‘::/0’ with the same target “Internet Gateway”. This’ll allow us to give access to internet accessibility to IPv6 for any subnet associated with the route table.
20. Associate a subnet with the route table under “Subnet association” section.
21. Add a subnet to your route table & hit on ‘Save’.

- 22.Under ‘Subnet’ section, when we click on the either of the subnet created above, we can see that “Auto-assign public IPv4 address” is set to ‘No’ which in our case we won’t be able to access that EC2 instance. So, select the above subnet & click on ‘Actions’ & then choose “Modify auto-assign IP settings” & click on the checkbox “enable auto-assign public IPv4” address.
- 23.Go to EC2 under Services.
- 24.Launch a default “Amazon Linux AMI” with default settings except select the newly created public VPC under ‘Network’ under “Configure Instance”.
- 25.Create a new security group with ‘SSH’ & ‘HTTP’ type.
- 26.Launch another EC2 instance with private VPC under ‘Network’ under “Configure Instance”.
- 27.Add a default security group, since we don’t want this instance to be publicly accessible.
- 28.We can note that the public VPC has been assigned an IP address automatically.
- 29.Create a Security Group adding SSH, MySQL/Aurora, HTTP, HTTPS, All-ICMP (this allows to ping our IP in private security group or to private subnet from public subnet) & hit ‘Create’.
- 30.Now, change the security group of the private EC2 instance to the newly created security group by selecting the EC2 instance & click on Instances, choose ‘Networking’ & click on “Change Security group”.
- 31.If we run any command like “yum update” in our private subnet, it does not give any result as it is not connected to the internet.
- 32.Launch a new EC2 instance selecting any of the ‘NAT’ instance from the security group with default configuration except under the security group, we need to select our VPC and under subnet, select the public one.
- 33.After the instance has been created, select that instance & click on actions. Under Actions, click on ‘Networking’, & click on “Change Source/Destination check”& disable it.
- 34.Note: Each EC2 instance performs source/destination check by default. It means instance must be source or destination of any traffic it sends or receives. However, a NAT instance must be able to send and receive traffic when the source or destination is not itself. Therefore, we must disable source/destination check on NAT instance.

- 35.Go to Route tables, select the Route table which has access to Internet gateway. (We can check it under associated subnets, as no subnets have been assigned to it).
- 36.Now login to the public & then to the private instance. We can see that we can run update command.

Note: The NAT instance has got a ton of bottle-neck as this is a single instance in a single availability zone. It won't have so much network throughput as instance type is t2.micro. It is relying on a single OS, so if it crashes we are going to lose all our servers behind the private subnet. To resolve this, we can put in multiple availability zone, multi-scaling groups, & multiple route out to the internet.

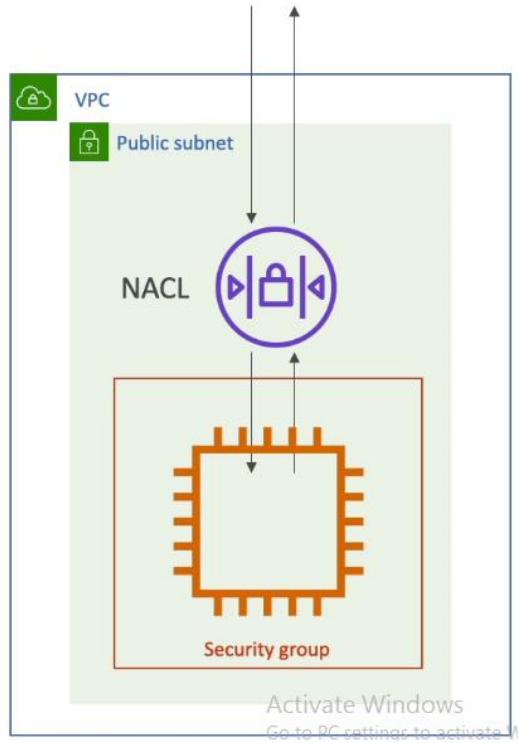
- 37.Go to VPC, we can see we've "NAT Gateway" which works on IPv4 & "Egress Only Internet gateways" which works on IPv6.
- 38.Go to NAT Gateway under VPC & hit on "Create".
- 39.Select the public subnet & we need to have an Elastic IP address. So, click on "Allocate Elastic IP address".
- 40.Hit on "Create a NAT Gateway" (takes 10-15 minutes to get available).
- 41.Go to route tables, click on the default route table, under 'Routes' we can see it is written as "blackhole" because the ENI (Elastic Network Interface) has been deleted. So, remove this route.
- 42.Add a route, and Target should be NAT gateway that we created in step 40.
- 43.Hit 'Save'.
- 44.Now, we can download or update our package because we've access to the internet.

Note: We can use Bastion sever with NAT-instances but not with NAT gateways, well Bastion servers are used to RDP or SSH into instances in private subnets. It can be done using out Terminal window or Putty. It is a good practice not to store private keys into public instance, rather store it on client devices. We should prefer to use NAT gateways rather NAT instances are harder to manage. NAT gateways are managed by AWS whereas NAT instances are self managed.

Exam Tips

1. NAT instance must be in a public subnet.
2. There must be a route out of a private subnet to the NAT instance, in order for this to work.
3. The amount of traffic a NAT instance supports depends upon the instance size. If we're bottlenecking it, increase the instance size.
4. We can create high availability using Auto-scaling groups, multiple subnets in different AZs, and a script to automate failover.
5. NAT instances are always a security group.
6. NAT gateways are preferred by enterprises.
7. They can scale up to 10Gbps.
8. We don't need to patch an OS, Anti-virus or worry about security groups.
9. Automatically assign a public IP address when they're being created.
10. Always remember to update the route tables & point them to the NAT gateways.
11. We should not keep NAT gateways in only one AZ, coz this may cause problems in AZ failure.
12. We don't need to disable Source/Destination Checks on NAT gateways.
13. NAT gateways are more secure than NAT instances.
14. We can only associate one subnet to one Network ACL, not multiple Network ACLs.

Network Access Control Lists vs. Security Groups



NACL

- It is a firewall which controls traffic to and from subnets.
- It can have allow and deny rules.
- They are attached at the subnet level.
- Rules only include IP addresses, like saying all the traffic coming from this IP address is allowed/denied.
- It is the first defense mechanism of the public subnet.
- The traffic coming to & from the internet is going first through the network ACL.

Security Groups

- It is a firewall that controls traffic to and from the ENI / EC2 instance.
- It only has ALLOW rules.
- Rules include IP addresses and other security groups.

Security Group	Network ACL
Operates at the instance level	Operates at the subnet level
Supports allow rules only	Supports allow rules and deny rules
Is stateful: Return traffic is automatically allowed, regardless of any rules	Is stateless: Return traffic must be explicitly allowed by rules
We evaluate all rules before deciding whether to allow traffic	We process rules in number order when deciding whether to allow traffic
Applies to an instance only if someone specifies the security group when launching the instance, or associates the security group with the instance later on	Automatically applies to all instances in the subnets it's associated with (therefore, you don't have to rely on users to specify the security group)

So, we attach a security group to our EC2 instance and now the traffic can flow all the way through to the EC2 instance.

1. Go to Network ACL under VPC.
2. “Create a network ACL” & deploy it under a VPC.
3. Hit ‘Create’.
4. When we create a private Network ACL, by default everything is denied under Inbound & Outbound rules.
5. An Ephemeral port is temporary hub used for Internet Protocol communication. They are allocated automatically from a pre-defined range by the IP stack software.
6. Add inbound and outbound rule by allowing it to HTTP, HTTPS & SSH with rule number 100, 200, 300 respectively.
7. We can see that if we SSH into the terminal using its IP and add an index.html page to it, then it is opened in the browser.
8. Now, add another rule 101, denying the HTTP port, still the webpage is opened in the browser but if we change the rule number from 101 to 99, we can see that page fails to load & time out and this change is instantaneous.

Note: So, Network ACL can be used to block specific IP address, IP address range.

: Since Network ACL comes before Security group, so even we allow the same rule in Inbound rule of security group, it block the IP address.

Exam Tips

1. VPC comes with a default Network ACL, and by default it allows all inbound & outbound traffic.
2. We can create custom Network ACL & this by default it denies all inbound & outbound traffic.
3. Each subnet in our Network ACL must be associated with a Network ACL, if it is not associated with any Network ACL then the subnet gets automatically associated to default Network ACL.
4. We can only associate one subnet with one Network ACL, however when we associate a network ACL with a subnet, the previous association is removed.
5. Network ACL contains a numbered list of rules that is evaluated in order, starting with the lowest number rule.
6. Network ACL are stateless. They have separate inbound & outbound rules, and can either allow or deny traffic.
7. We can block IP addresses using Network ACL not using Security group.

Custom VPC's & ELB

1. Go to EC2 under Services & click on Load Balancers.
2. We should 90% prefer to use Application load balancer over Network Load Balancer unless we need ultra-high performance or static IP address with our application.
3. Click on “Create load balancer” & choose Application Load Balancer.
4. Select everything to be default, select the newly created VPC. Also, the page says, we can specify only one subnet per AZ and we must specify subnets from at least 2 AZ to increase the availability of our load balancer unless this can't be up.

VPC Flow logs

- It's a feature that enables us to capture information about the IP traffic going to & fro from your network interface in our VPC.

- It is stored using CloudWatch logs. After we've created a Flow log, we can view & retrieve its data in Amazon CloudWatch.
- It helps to monitor & troubleshoot connectivity issues. Example:
 - Subnets to internet.
 - Subnets to subnet
 - Internet to subnet
- Captures network information from AWS managed interfaces too. ElastiCache, Load balancers, RDS, Aurora, etc.
- It can be created at 3 levels:
 1. VPC – Captures all Elastic Network Interface traffic within VPC.
 2. Subnet – Captures EC2 instance & ENI within particular subnet.
 3. Elastic Network Interface level.

LAB

1. Go to VPC under Services.
2. Select your VPC & click on 'Actions' & choose "Create flow log".
3. The Filter menu can be used to set the logs of accepted, rejected or all the traffic flow information from the network interfaces associated with the resources.
4. Under 'Destination', click on "Send to CloudWatch" logs.
5. Select a 'Destination', a new one can be created by going to CloudWatch & Services & then select "Log groups", hit on 'Action' & select "Create Log group".
6. Create a IAM role, click on "Set up permissions" which will take us to a new page.
7. Click on "Allow".
8. Click on 'Refresh' tab & we can view the newly created IAM role.
9. Hit on 'Create'.
10. We can stream these logs to AWS Lambda or Amazon Elasticsearch service by clicking on the "Log group" & then click on 'Action' & selecting your option.

Note: If we stream our logs to AWS Lambda then we can have our environment proactively react to something going inside our VPC when

someone tries to attack our EC2 or Elastic Network Interfaces, also we can export the data to S3.

Exam Tips

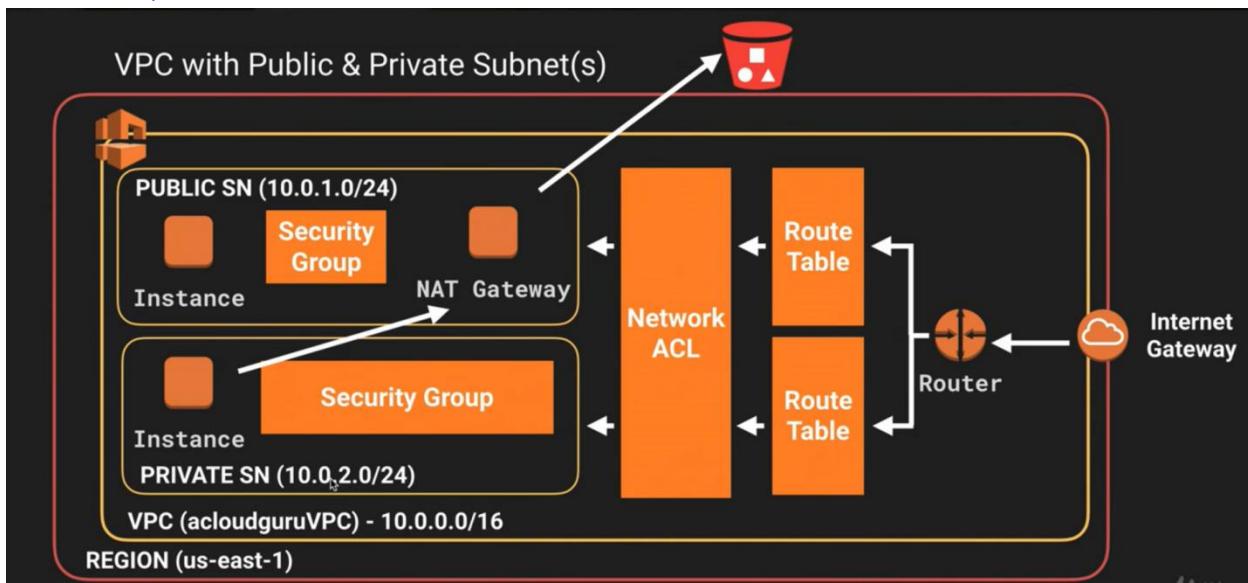
1. We cannot create flow logs for VPC that are peered with our VPC unless the peer VPC is in our account so they've to be within same account.
2. We cannot tag a flow log because probably their value changes.
3. Once a flow log is created, we cannot change its configuration. For e.g., we cannot associate a different IAM role with a flow log.
4. Not all the IP traffic is monitored. The traffic is **not** monitored for:
 - a. Traffic generated by instances when they contact DNS server. If we use our own DNS server then all the traffic to that DNS Server is logged.
 - b. Traffic generated by Windows instance for Amazon Windows license activation.
 - c. Traffic to & from 169.254.169.254 for instance metadata.
 - d. DHCP traffic.
 - e. Traffic to the reserved IP address for the default VPC router.

NAT vs Bastion

NAT instance are used to route internet traffic to EC2 instances that're in private subnets. These instance are able to connect out through to the internet but people within the internet cannot use SSH or RDP to connect via NAT into private instances. To do this, we need to have a Bastion host or a Jump box that basically allow us to SSH or RDP into our Bastion & then initiate a private connection over the private network to our instance to administer them using SSH or RDP. The idea is that instead of using multiple hardened fleet of EC2 instances of a security purposes, we can have one Bastion & then we can access all our instances in private subnets through Bastion.

NAT instance is always behind a security group whereas NAT gateway is independent of a Security Group.

VPC end points



Consider the diagram in which our instance is sitting in a private subnet & it is behind NAT gateway & if we want to send files to S3, the NAT gateway / VPC endpoint gateway is going to traverse through the internet, it will go to a public endpoint.

If we want our traffic to go directly to S3, not through NAT gateway, so we need to create an internal gateway instead of a NAT gateway.

1. Go to IAM under services.
2. Go to ‘Roles’ and click on “Create role”.
3. Select EC2 & select case “Allow EC2 instance to call AWS services on behalf”.
4. Add permission “AmazonS3FullAccess”.
5. Set the tag, Give the Role a name and hit on “Create Role”.
6. Go to EC2 under Services.
7. Select the previously launched private instance under ‘Instances’.
8. Click on actions, navigate to instance settings & click on “Attach/Replace IAM Role”. Select the role created in step 5 and click on ‘Apply’.
9. Go to VPC under Services and click on “Network ACLs”.
10. Go to VPC Network ACL and associate the public subnet with it.
11. Login to your private instance from your public instance.
12. Now, go onto VPC under Services, click on Endpoints & create one.
13. Endpoints come in 2 different varieties: Interface & Gateway.

Note:

- i. A VPC endpoint securely allow us to connect our VPC to another service using a private network instead of a public network.
- ii. An interface endpoint is powered by ‘PrivateLink’ & uses elastic network interface as an entry point to the traffic destined to the service.
- iii. A gateway endpoint serves as a target for a route table for the traffic destined for the service.
- iv. It gives us enhanced security and lower latency to access AWS services.

14. Select the type as ‘Gateway’ & your service name (it is name as your region name)

15. Select the VPC created above & the private Route Table ID.

16. Click on “Create endpoint”.

17. Login to your private SSH & run command - `aws s3 ls`, we can view the list of all s3 buckets.

Site to Site VPN



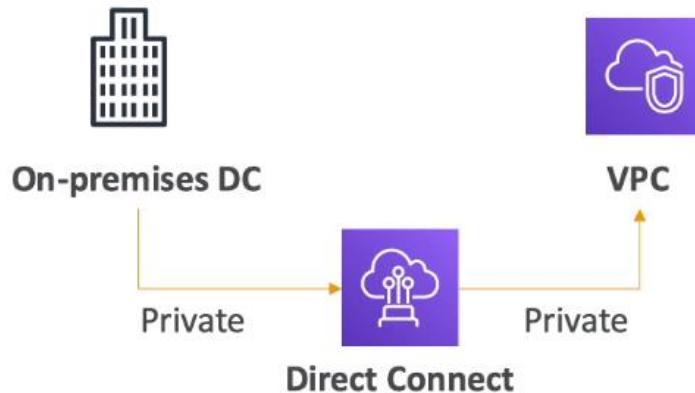
Suppose we want to establish connectivity between our On-premise data center (ex: office) to the Cloud VPC. The first way is site to site VPN.

- It helps in connecting on-premise VPN to AWS.
- The connection is automatically encrypted.
- It goes over the public internet.

Here, we are establishing connection between on-premise data center and VPC and that goes over the public internet.

Direct Connect

It also helps to establish connectivity between our On-premise data center (ex: office) to the Cloud VPC. And this is the another way of doing so.



- It establishes a physical connection between on-premise apps & AWS .
- The connection is private & it'll not go over the public internet. Hence it is going to be secure & fast.
- Since it is a private line to our VPC, it takes at least a month to establish because there's some work that needs to happen.

Note: If we use Site-to-Site VPN or Direct Connect, they cannot access VPC endpoints. VPC endpoints are used to access AWS resources privately within the VPC not by connecting on-premise Data center.

VPC Clean-up

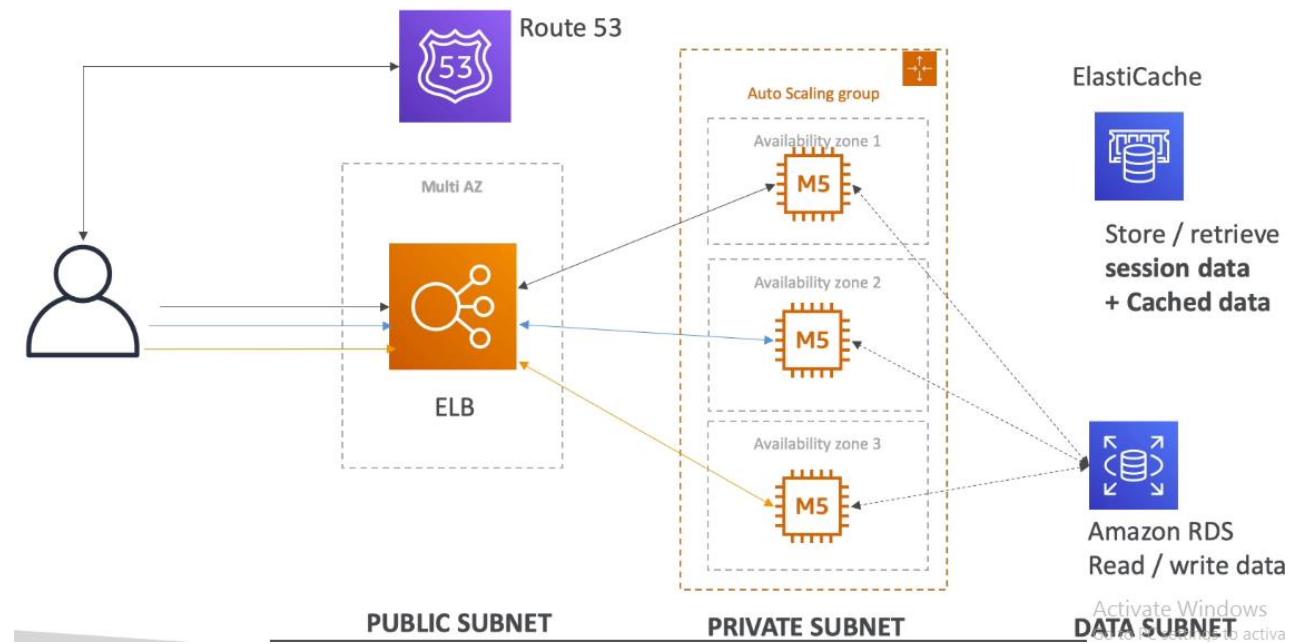
1. Go to EC2 under Services.
2. Terminate the instance created.
3. Go to VPC under Services & navigate to NAT gateways.
4. Delete the one you created.
5. Navigate further to Internet Gateway & select “Detach from VPC” & Delete it later.
6. Delete the endpoint.
7. Go to “Your VPC”, & click on your VPC & delete it.

Exam Tips:

1. NAT instances must be in public subnet.
2. There must be route out of the private subnet to the NAT instance, in order for this to work.
3. The amount of traffic a NAT instance could support depends on the size of the instance size. If we're bottlenecking, we need to increase the instance size.

4. We don't need to disable source/destination check on NAT Gateways.
5. NAT Gateways are more secure than NAT instance.
6. Gateways are much more resilient & durable because they don't rely on single ENI.
7. Inside our VPC, we have subnets and subnets allow to partition our network inside the VPC. Subnets are defined under the availability zone level.
8. There's one default VPC per AWS region.
9. Subnets are tied to AZ, they represent network partition of the VPC.
10. Internet gateway provides Internet at the VPC level to the public gateway.
- 11.t

Example: A 3-tier architecture



Our user wants to access web application, and we've designed an Elastic Load Balancer which is spread across the availability zone. Since the ELB is going to be publicly accessible, so it needs to be deployed in a public subnet. To access the Load Balancer, we need to do a DNS Query to know where it is so, we'll use Route 53. Hence our user will be talking to our ELB.

The ELB is going to spread the traffic onto the EC2 instances, so they're going to sit in Auto-Scaling group. But this time the ASG need not to be publicly accessible from the internet, only they're going to be accessible through the ELB.

So, we are going to deploy it in a private subnet. There're 3 AZ containing EC2 instances in all of them. The ELB is going to send traffic from public subnet to the private subnet using Route tables.

We need to have dataset somewhere, so we'll having a second private subnet known as Data subnet. This data subnet is in the 3rd Tier. We're going to have Amazon DB which is helpful in Reading/Writing data. Hence the EC2 instance is going to connect RDS. ElastiCache cache data from RDS (like restoring the session data of EC2 instance).

LAMP Stack on EC2

LAMP stands for Linux Apache MySQL PHP

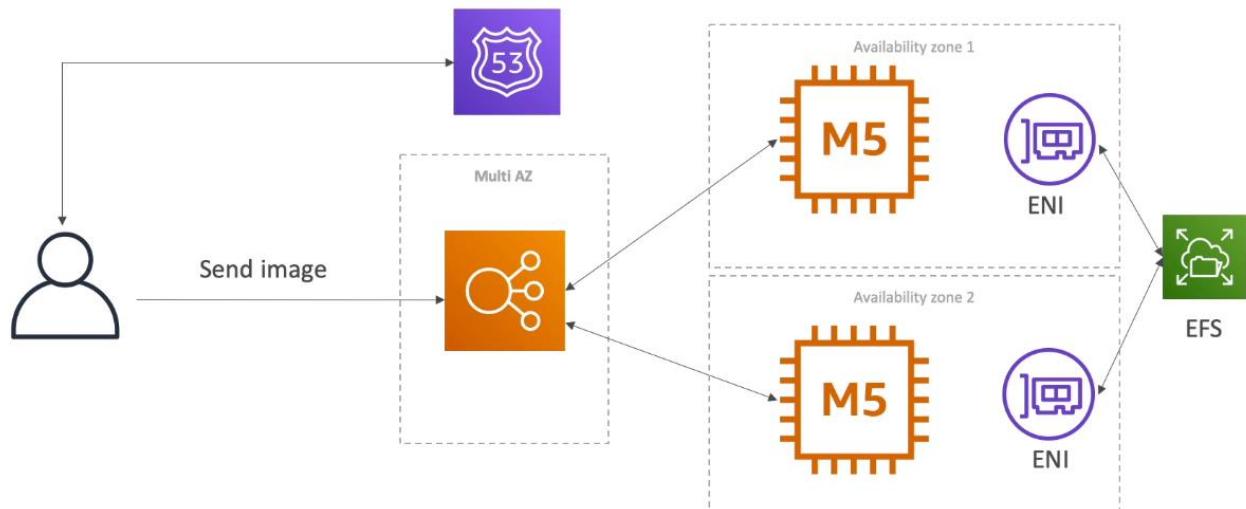
Linux is going to be OS for EC2 instances.

Apache is the webserver that is going to be run on Linux (EC2).

MySQL is the database on RDS.

PHP is the application logic how to render web pages (running on EC2).

On this LAMP Stack, we can add Redis/Memcached (ElastiCache) to include caching tech. To store our application data & software, we can use the EBS drive attached to our EC2 instances



WordPress also work on 2 tier architecture, the Load Balancer tier and the application tier. If users share images to EC2 through the Load balancer and hence EC2 instance needs to share these images with all other EC2 instance and for this the perfect use case is EFS which is a network drive which creates Elastic

Network Interfaces in each AZ. Hence our EC2 instances can store images on EFS and all the other EC2 instances will have access to these images.

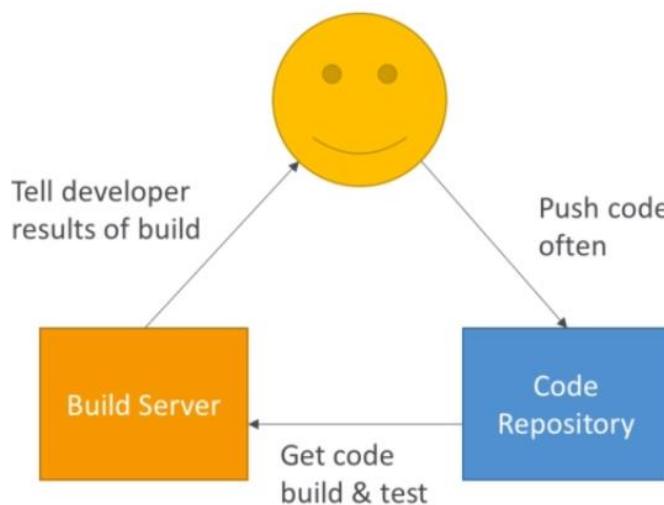
11. SQS – Simple Queue Service

It was the first AWS service that was publicly available. It is a web service that gives us access to message queue that can be used to store messages while waiting for the computer to process them.

It is a distributed queue system that enables web services applications to quickly & reliably queue messages that one component in the application generates to be consumed by other application. It is a temporary repository for messages that're awaiting processing.

12. CICD

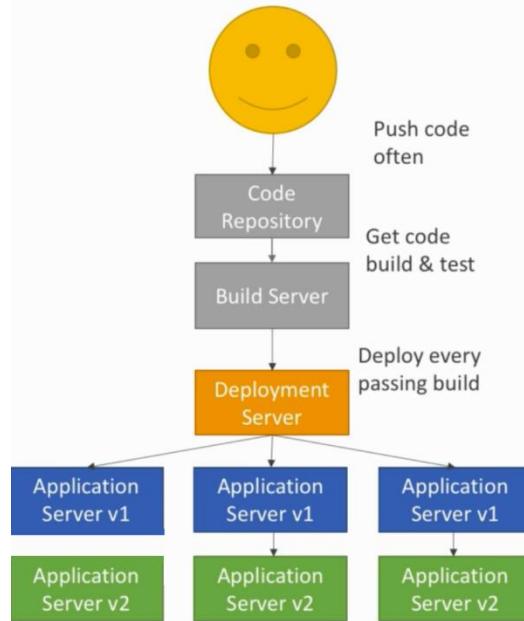
Continuous Integration



- It helps us to push our code on repository (GitHub, CodeCommit, Bitbucket, etc.) and have it deployed to AWS automatically and in right way.
- Tested/build before deployment (CodeBuild, Jenkins CI, etc. and later the developer gets the feedback about the test that have passed/failed).
- With possibility to go into different stages (dev, test, pre-prod & prod after fixing the bugs early and deploy faster)

- With manual approval where needed and deploy often.

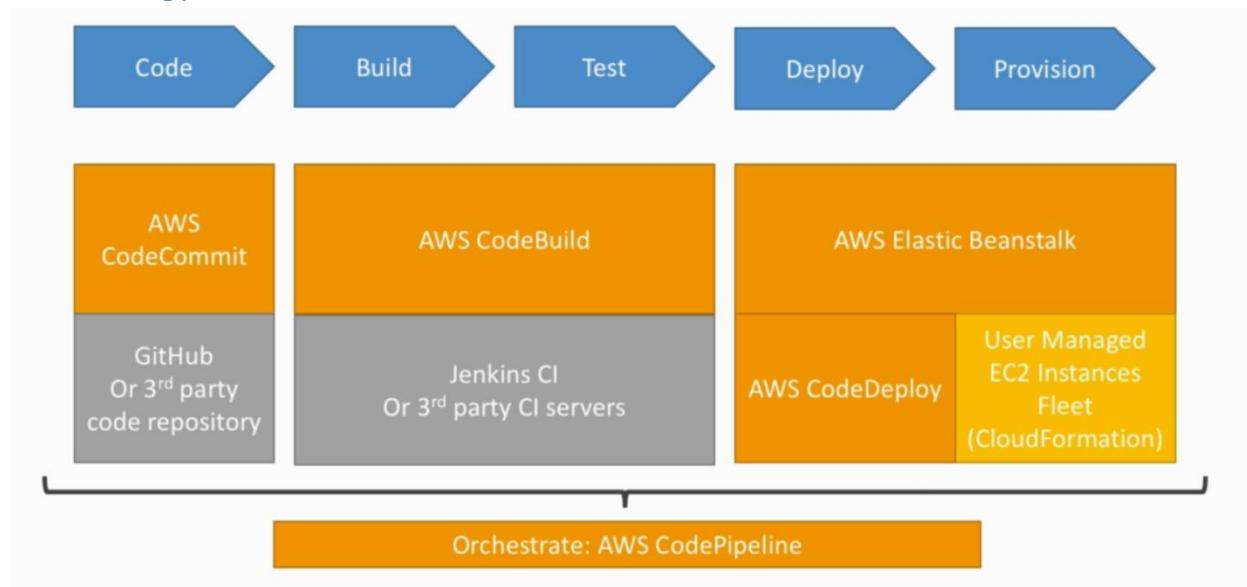
Continuous Delivery



- It ensures that the software can be released reliably whenever needed.
- Ensures deployments happen often and are quick.
- Decrease the release duration from months to everyday (or even 5 times a day).
- It usually means automated deployment.
- It means automated deployment using:
 - CodeDeploy
 - Jenkins CD
 - Spinnaker, etc.

We push the code, the build server build & test from Continuous Integration. From Deployment Server, it deploys every build that passes. The application which was running version v1, the deployment server using the scripts makes it go to version 2.

Technology stack



Code Commit

Version control is the ability to understand various changes that happened to the code over time which can be controlled using Git, SVN, etc. A Git is a central repository server. The benefits of code commit are:

1. Collaborate with other developers
2. Make sure code is backed up somewhere
3. Make sure it is fully viewable & auditable.

Since the Git repositories can be expensive and some industry includes paid private repositories as well as free public repositories. AWS CodeCommit provides it us at very low cost. It offers:

- Private Git repositories
- No size limit on repositories
- Fully managed, highly available
- Code only in AWS Cloud account which increases security and compliance.
- Security by using encryption, access control, etc.
- It is integrated with Jenkins, CodeBuild, etc. so we are not locked with AWS.

Code Commit Security

- Interaction are done using Git using standard Git commands.
- Authentication in Git:

- SSH Keys: AWS users can configure SSH Keys in their IAM console
- HTTPS: It is also done using the same IAM console and can be done using CLI authentication helper or generating HTTPS credentials.
- MFA can be enabled for extra security.
- Authorization on Git
 - We can use IAM policies and manage user/roles to right repositories.
- Encryption
 - Repositories are automatically encrypted using KMS.
 - Encrypted in transit (can only use HTTPS or SSH)
- Cross Account Access
 - Never share SSH keys
 - Never share AWS credentials
 - Use IAM roles and use AWS STS (cross account access API, called AssumeRole)

CodeCommit notifications

CodeCommit notification can be triggered using AWS SNS or AWS Lambda or AWS CloudWatch Event rules. Use case for SNS/Lambda notification are:

- Deletion of branches
- Trigger for push that happen in master branch
- Notify external build system
- Trigger AWS lambda function to perform codebase analysis (maybe credentials got committed in the code).

Use case for CloudWatch Event rules are:

- Trigger for code commit (create/update/delete or commented)
- Commit comment events
- CloudWatch event rules goes into SNS topic.

CodeCommit vs GitHub

Similarities

1. Both are Git repositories
2. Both support code review.
3. It can be integrated with AWS CodeBuild.
4. Both support HTTPS and SSH method of authentication.

Differences

1. Security
 - a. GitHub is administered by GitHub users while
 - b. CodeCommit are administered by IAM users & roles.
2. Hosting
 - a. GitHub is hosted by GitHub, a 3rd party.
 - b. GitHub Enterprise are self-hosted on our servers which means we need to manage the servers while
 - c. AWS CodeCommit are managed & hosted by AWS.
3. UI
 - a. GitHub is fully featured while
 - b. CodeCommit UI is minimal

LAB – CodeCommit

1. Go to CodeCommit under Services.

Note: we can see in the UI, there's CodeBuild, CodeDeploy & CodePipeline available for us in the same page.

2. Click on “Create Repository”.
3. Hit ‘Create’.
4. Choose the file to upload, provide an author name & an email id.
5. Hit on “Commit changes”.
6. And after committing the code changes, we can see view the commit id.
7. Under settings tab, go to ‘Notification’ sub-tab.
8. Hit on “Create notification rule”
9. Give a name to the notification, select the events that are going to trigger notifications.
10. Select SNS topic and hit on ‘Submit’.
11. Click on ‘Trigger’ Tab.
12. Choose a Trigger name & select the list of events for which triggers are going to hit.
13. Select the branch name to which it is going to apply.
14. Select the service that is going to be triggered.

15. Hit on “Create trigger”.

Notes:

1. The “**Pull request**” is for the coders they work in different branches, and they want to merge their code into the master branch.
2. “Create Pull request” can review the code that are committed.
3. “Commit” helps us view the commit history in our CodeCommit Repository.
4. “Branches” helps us view all our branch, the default is our master branch and we can create a branch by clicking on “Create branch”.
5. Settings:
 - a. We can rename our repository. Here we get a repository ID & ARN.
 - b. Under Notification tab, we can set up notifications so that repository user can receive emails about repository events. These events can be pull request update, pull request comment, commit comments event.
 - c. Under ‘Trigger’ tab, we can create a Trigger for CodeCommit repository, so that events in the repository invoke a Lambda functions or Amazon SNS.

Now, we are going to learn how to commit our code without using Create file or Upload file.

16. Go to IAM under services.
17. Go to Users.
18. Select your users & go to ‘Security credentials’ tab.
19. Once we scroll down, we can see SSH keys for AWS CodeCommit & HTTPS Git credentials for AWS CodeCommit.
20. Click on “Generate credentials” under HTTPS credentials for CodeCommit. We can generate up to 2 username & password directly to use HTTPS to connect directly to CodeCommit.
21. Go to CodeCommit under Services.
22. Click on “Clone URL” under ‘Code’ section and select “Clone HTTPS”.
23. Install Git on your machine.
24. Type `git--version` & check if Git is properly installed.
25. Now type, `git clone URL`(URL copied in step 22).
26. Type in the credentials generated (in step 20).

Note: To reset git credentials

Go to Control Panel > User Accounts > Credential Manager > Windows **Credentials**. You will see **Git credentials** in the list (e.g. `git:https://`). Click on it, update the **password**.

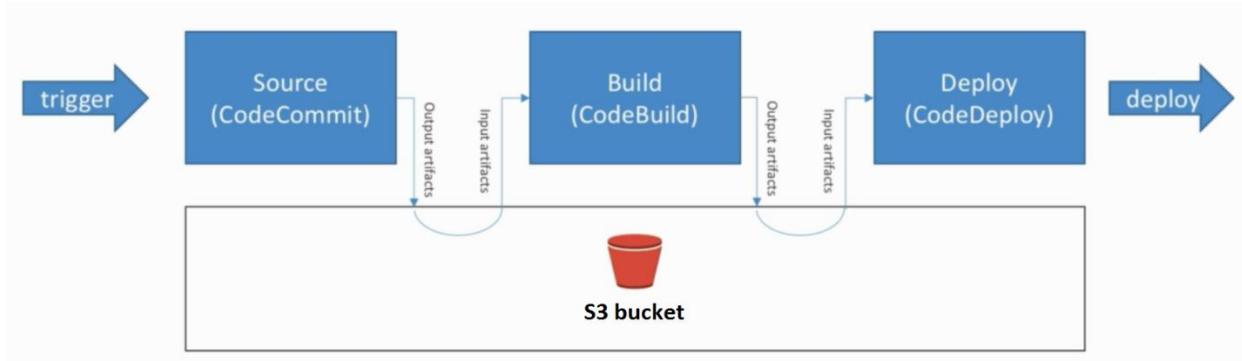
- 27.Type in the `ls` command & we can view our new folder.
- 28.Type `cd folder_name`.
- 29.Type `ls`& all the contents inside the folder can be viewed.
- 30.To copy the files, type `cp //path//to dir path//to//newDir`
- 31.To add copied files to your directory, run: "`$ gitadd`"
- 32.Check your file's status: `$ git status`.
- 33.Commit your files: `$ git commit -m "Comments"`
- 34.To push the file to your server: `$ git push`
- 35.Configure your email id: `$ gitconfig --global user.email "raunak.rtu@gmail.com"`
- 36.And then run commit command to commit those files, `$ git commit -m "New files added"`
- 37.Finally, `$ git push` command uploads the files (check-in) the files to the repository.
- 38.We can even edit these files in the console.
- 39.Go to 'Commit' to view the log of changes done.

CodePipeline

It is a fully manages continuous delivery service that helps to automate our release pipelines for fast & reliable application & infrastructure updates.

CodePipeline Artifacts

They are bunch of files that are passed & stored through Amazon S3 on to the next stage.



Let's imagine Amazon S3 bucket & then we get a **trigger**, hence our source (**CodeCommit**) gets triggered (example: we pushed some code & this source will **send everything to Amazon S3** which is called the source output artifacts). These artifacts slowly comes through Amazon S3 buckets to go into the **build stage** so that **CodeBuild will get the artifacts**, it will get code from the source & build it and **after building it may also generate artifacts**. It may be like generated binaries, or zip files and **put this right back to Amazon S3 bucket**. This output will be **piped up the way to the deploy stage** to **CodeDeploy**. Now, CodeDeploy as the output of our build stage for example the zip file. They are able to just **deploy** your files.

CodePipeline Troubleshooting

1. Whenever there is a state change in the pipeline, it generates AWS CloudWatch events. These events can trigger a SNS notification.
Ex: when we create events for failed pipelines or cancelled stages.
2. If CodePipeline fails a stage, our pipeline stops & we can get information in the console.
3. AWS CloudTrail can be used to audit AWS API calls.
4. If pipeline can't perform an action, make sure the IAM service role attached does have enough permission policy.

LAB

1. Go to CodeCommit under Services. (Make a note that a code is present in the git repository & it is not zipped also, make sure that other than codes no other files are present like image or other).
2. Navigate to CodePipeline & under that choose Pipelines
3. Create a new pipeline.

4. Specify it a name.
5. Leave everything default & hit on ‘Next’.
6. Choose ‘Source Provider’. This is where the input artifacts are stored for our pipeline. For now, choose CodeCommit but we’ve options like ECR, S3, BitBucket, Github.
7. Choose a repository name. (Create one if you’ve not created previously).
8. Select your branch name.
9. Leave other options as default & hit on ‘Next’.
10. Add a build stage, select a build provider & fill other stage (for now, we’re skipping this step).
11. Choose one of the deploy provider.
12. Choose a region & select your application name.
13. Select an environment (Create one if there’re no options).
14. Hit on “Create Pipeline”.
15. Go to Pipeline, we can see the source running.
16. Go to Elastic Beanstalk, we can see that Beanstalk is updating our environment (the environment we chose in step 13).
17. After the build & deployment is done, our Beanstalk environment application gets updated.
18. To delete a file from repositories:
 - a. Navigate to that directory using `cd path/to/file`
 - b. Remove the file using: `rm filename`
 - c. Add the file to the list: `git add .`
 - d. Commit your changes: `git commit -m "comment"`
 - e. Push the changes: `git push`
19. Let’s add another stage (deployment stage) to our pipeline, click on edit in the pipeline page created.
20. Click on “+ Add stage” & give it a name & hit on “Add stage”.
21. Click on “+Add action group”.
22. Set a name of your choice.
23. For now, under “Action provider”, choose “Manual Approval”.
24. Leave the optional settings for now, click on “Done”.
25. Add another action group sequentially (even we can add it parallelly).

Note: Stages have multiple action groups.

26. Choose a name of your choice & under action provider set “AWS Elastic BeanStalk”.
27. Select Input artifacts to be “SourceArtifacts”.
28. Hit on Done & Save it from the top.
29. Make few code changes & commit.
30. We can see that code builds, then deploys, waits for manual approval & again deploys to production environment.

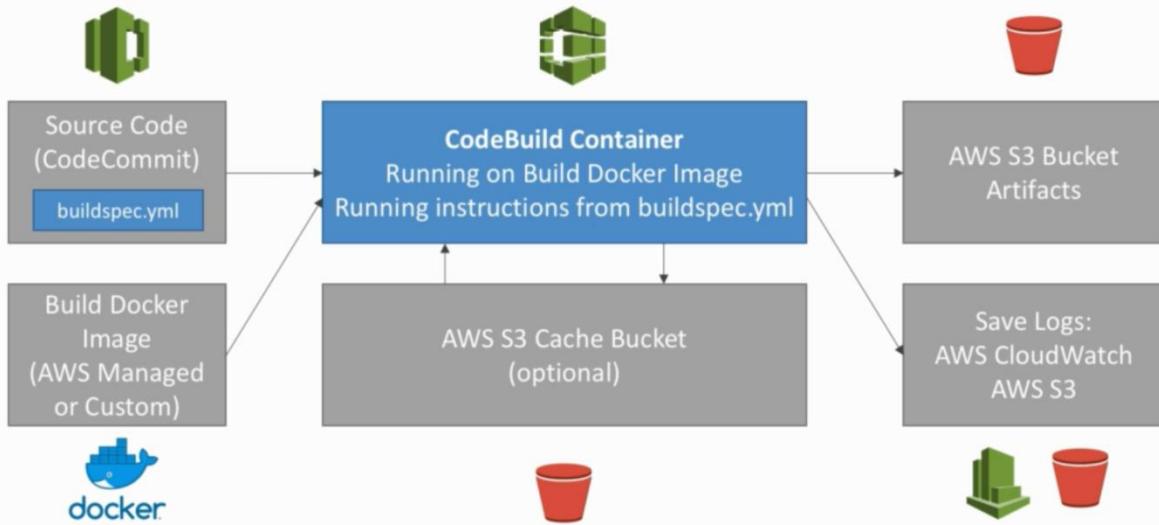
CodeBuild

- It is a fully managed build service and an alternative to other tool such as Jenkins.
- It scales continuously (no servers to manage or provision – no build queue). As build request comes in, CodeBuild builds it for us.
- It is pay for usage: the time it takes for complete build (not like Jenkins, when it sits idle or when there's no task, then we need to pay for it).
- Leverages Docker under the hood for reproducible builds and it has the possibility to extend capabilities leveraging our own base Docker images.
- It is secure as it is integrated with KMS for encryption of build artifacts, IAM for build permissions, and we can run CodeBuild within our own VPC for network security, CloudTrail for API calls logging.
- It can source code from GitHub/ CodeCommit/ CodePipeline/ S3, etc.
- Build instruction can be defined in code (buildspec.yml file)
- Output are logged into S3 or AWS CloudWatch logs.
- Metrics to monitor CodeBuild statistics.
- CloudWatch alarms can be used to detect build fails & trigger notifications.
- Trigger SNS notifications.
- CodeBuild can be reproduced locally in case of errors.
- Builds can be defined with CodePipeline or CodeBuild itself.

Working

We've a **source code** (buildspec.yml file) which is **CodeCommit** for us. We also need a **Build Docker image** (either of Amazon managed Docker image or it can be custom). There's going to be a **CodeBuild Container** which runs as soon as the code is triggered. It uses source image from the run & run instructions comes from

`buildspec.yml` file. There's optional build **S3 Cache bucket** which is used to cache multiple dependencies or artifacts, we can pull in this cache when our build starts & increase the performance. After successful finish of CodeBuild, it passes the output to **AWS S3 bucket**. The logs are sent to **AWS CloudWatch AWS S3**. If the build succeed, the cache files are put back to S3 buckets.



BuildSpec

- `Buildspec.yml` must be at the root of our code (not within subdirectory).
- Define environment variables:
 - It can be plain text variables.
 - Secure Secrets: It can be SSM parameter store. But we don't ever store secrets in our code.
- Phases:
 - Install: to install dependencies what we need for build.
 - Pre-build: Final command to execute before build.
 - Build: actual build commands.
 - Post-build: Finishing touches (zip output for example).
- Artifacts: what we upload to Amazon S3 (encrypted with KMS).
- Cache: Define here which file to cache (usually dependencies) to S3 for future build speed -up.

Local Build

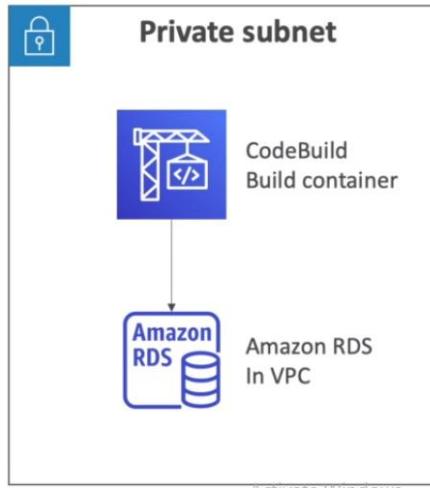
- It can be performed in case of deeper trouble shooting beyond the logs.

- We can run CodeBuild locally, but we need to install Docker for this.
- We also need to leverage CodeBuild Advance.

LAB

1. Go to CodePipeline under Services.
2. Navigate to build projects.
3. Hit on “Create build project”.
4. Set your project a name.
5. Under Source, select AWS commit & choose the repository.
6. Choose the OS.
7. For runtime, choose ‘Standard’.
8. Use the latest image & under version, use the latest version.
9. Click on “New Service Role”. CodeBuild needs to run on some actions like pulling the code from CodeCommit, It’s important to create an IAM service role.
- 10.Under additional configuration, we can set Timeout, Queued Timeout, Certificates. Sometimes CodeBuild needs to access resource within VPC. (None for now).
- 11.We can specify the size of the Docker compute. If we need to run a lot of test & if we need a lot of memory, we can choose higher memory with more number of vCPUs.
- 12.We can add environment variables, encryption type, etc.
- 13.Under Buildspec (it stands for Build specification), use the default one for now which is created by AWS.
- 14.Under Artifacts, we can either push the Artifacts to S3 or choose “No artifacts”.
- 15.We can also set the encryption & select cache type for better performance.
- 16.Select the logs either to be pushed in CloudWatch logs or S3 logs.
- 17.Hit on “Create build project”.
- 18.To run the build, click on “Start build”.
- 19.Select the branch, commit ID (optional) & configure other options (not for now) click on “Start build”.
- 20.We can go to lastic Beanstalk & check the URL is working or not.

CodeBuild in VPC



- By default, CodeBuild containers are launched outside our VPC, therefore they cannot access resources in our VPC.
- If we've a private subnet and an Amazon RDS DB and if we launch a CodeBuild by default then it cannot access Amazon RDS DB coz CodeBuild is not launched in VPC.
- We can specify a VPC configuration with:
 - VPC ID
 - Subnet IDs
 - Security Group IDs
- As soon, as we do the above step, our build can access resources in our VPC.
- Now, CodeBuild is in VPC and can access RDS if the security group rules allow.
- Use cases: Integration test, data query, internal load balancers. (as they are privately deployed in our VPC).
- Go to Services -> CodeBuild -> Build projects -> yourBuildProject. Click on Edit and select environment. Under Additional configuration, select the VPC, subnets, Security group and click on Validate VPC setting.

CodeDeploy

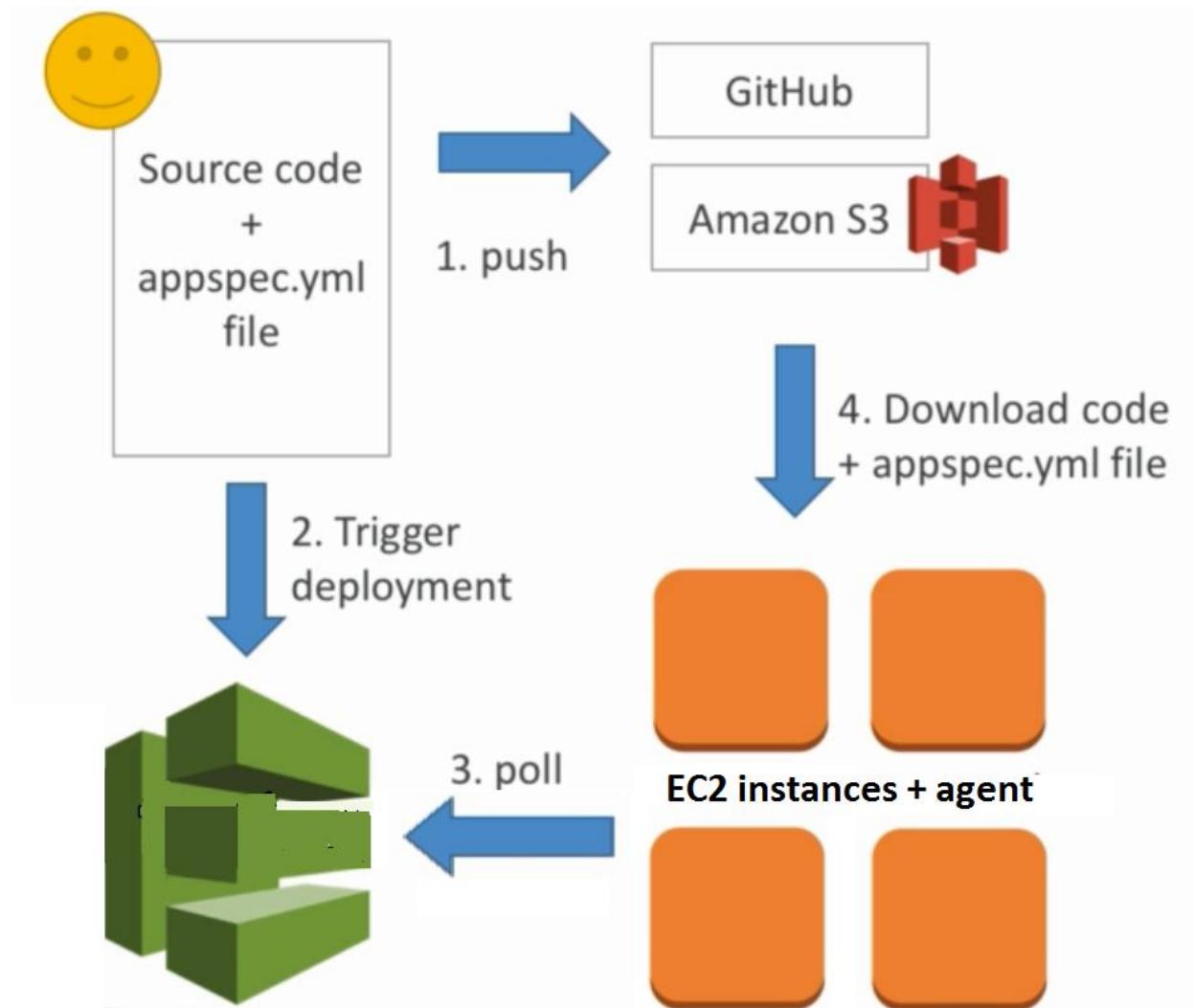
It comes to scenario when we want to deploy our application to many EC2 instances provided these instances aren't managed.

To make CodeDeploy work:

Requirements

1. Each EC2 machine (or on premise machine) must be running CodeDeploy agents.
2. Agent continuously polls AWS CodeDeploy for work to do.
3. CodeDeploy sends appspec.yml file
4. Application is pulled for S3 or GitHub.
5. EC2 will run the deployment instructions.
6. CodeDeploy agent will report of success/failure of deployment on the instance.

Working



1. Source code + appspec.yml file pushes the code to GitHub/Amazon S3.
2. Triggers deployment.
3. Since EC2 instances continuously polls, it will realize that deployment has been triggered & it's time to deploy.
4. So they'll download the code & appspec.yml file onto the EC2 instances & agent takes care of whatever is in the appspec.yml file to deploy our application correctly.

Points to remember

1. EC2 instances are grouped by development group (dev/test/build)
2. CodeDeploy can be chained into CodePipeline & use artifacts from there.
3. CodeDeploy can re-use existing setup tools, works with any application, auto scaling integration.
4. Blue/Green deployment works only with EC2 instances.
5. It supports for AWS Lambda deployments.
6. CodeDeploy only deploys our application.

Primary components

1. Application: Unique name
2. Compute platform: EC2/ on-premise or Lambda
3. Deployment configuration:
 - a. EC2: specify minimum number of healthy instances for the environment
 - b. AWS Lambda: specify how traffic is routed to your updated Lambda functions versions.
4. Deployment group: group of tagged instances.
5. Deployment type: In-place deployment or Blue-Green deployment.
6. IAM instance profile: need to give EC2 instance the permission to pull from Github.
7. Application revision: application code + appspec.yml file
8. Service Role: Role for CodeDeploy to perform what it needs.
9. Target revision: Target deployment application version.

CodeDeployAppSpec

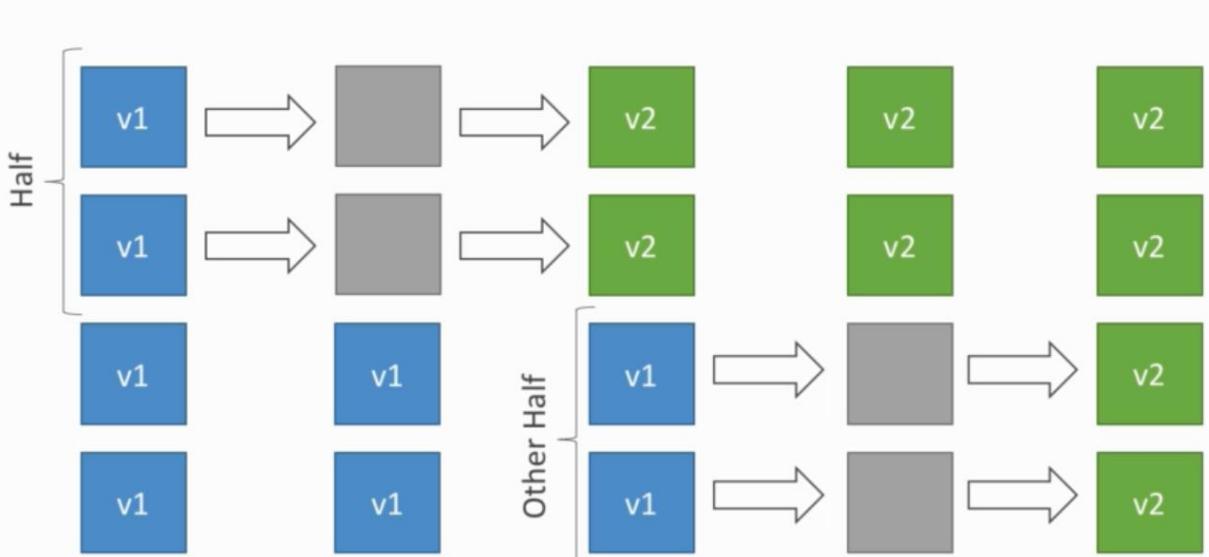
1. File section: how to source & copy the files from S3/ GitHub to filesystem.
2. Hooks: Set of instructions to so to deploy the new version(hooks can have timeouts). The order of hook is:

Event	Start time	End time	Duration	Status
ApplicationStop	Sept 26, 2018 7:51:29 AM UTC	Sept 26, 2018 7:51:29 AM UTC	less than one second	Succeeded
DownloadBundle	Sept 26, 2018 7:51:30 AM UTC	Sept 26, 2018 7:51:30 AM UTC	less than one second	Succeeded
BeforeInstall	Sept 26, 2018 7:51:31 AM UTC	Sept 26, 2018 7:51:32 AM UTC	2 secs	Succeeded
Install	Sept 26, 2018 7:51:33 AM UTC	Sept 26, 2018 7:51:33 AM UTC	less than one second	Succeeded
AfterInstall	Sept 26, 2018 7:51:34 AM UTC	Sept 26, 2018 7:51:34 AM UTC	less than one second	Succeeded
ApplicationStart	Sept 26, 2018 7:51:35 AM UTC	Sept 26, 2018 7:51:35 AM UTC	less than one second	Succeeded
ValidateService	Sept 26, 2018 7:51:36 AM UTC	Sept 26, 2018 7:51:36 AM UTC	less than one second	Succeeded
BeforeAllowTraffic	Sept 26, 2018 7:51:49 AM UTC	Sept 26, 2018 7:51:49 AM UTC	less than one second	Succeeded
AllowTraffic	Sept 26, 2018 7:51:50 AM UTC	Sept 26, 2018 7:52:11 AM UTC	21 secs	Succeeded
AfterAllowTraffic	Sept 26, 2018 7:52:12 AM UTC	Sept 26, 2018 7:52:12 AM UTC	less than one second	Succeeded

- a. ApplicationStop
- b. DownloadBundle
- c. BeforeInstall
- d. AfterInstall – Application cleanup/ Launch a server
- e. ApplicationStart
- f. ValidateService – a Health check, to see that the application is working correctly.

Deployment configuration

1. Configs
 - a. One at a time: one instance at a time, if one instance fails then deployment stops.



- b. Half at a time
- c. All at once: Quick, but no healthy host. Good for development
- d. Custom: say minimum healthy host = 75%

2. Failures

- a. Instances stay in “failed state”.
- b. New instances be first deployed to failed state instances.
- c. To rollback, re deploy old deployment or enable automated rollback for failures.

3. Deployment Targets

- a. Set of EC2 instances with tags
- b. Directly deploy to ASG
- c. Mix of ASG/Tags, so we can build deployment segments.
- d. Customization in scripts with “Deployment Group Name” environment variables.

LAB

Firstly, we need to create an IAM role.

1. Go to IAM -> Create Role
2. Select “CodeDeploy”.
3. Under Case – Select CodeDeploy
4. Hit on Next
5. Select AWSCodeDeployRole& Hit on “Create Tags”
6. Write down your tags & hit on “Review”.

7. Give your role a name & hit on “Create Role”.

Secondly, we need to create an EC2 service role since an EC2 instance will be running with CodeDeploy agent & an EC2 instance needs to pull data from S3, so it needs an S3 access.

8. Go to “Create Role” under Roles.
9. Select EC2 and hit on “Add permissions”.
10. Select “AmazonS3ReadOnlyAccess”.
11. Set Tags & Review
12. Give the role a name.
13. Hit on “Create Role”.

Go back to CodeDeploy

14. Go to Applications under CodeBuild
15. Hit on “Create Application”.
16. Give your application a name & choose a Compute platform. (For now, EC2/On-premises).

Now, we need to create a Deployment Group but before that we need to create an EC2 instance, where we'll deploy an application.

17. Go to Services under EC2.
18. Hit on “Launch instance”.
19. Select “Amazon Linux 2 AMI”.
20. Under IAM role in Configure Instance details, select the role created in step 13.
21. Select other settings to be default, except under “Security Group” add “HTTP”.
22. Hit on Launch after creating a code pair.
23. SSH into the machine.
24. For it to work with CodeDeploy service, we need to install CodeDeploy agent on this machine.
25. Run the commands:
 - a. sudo yum update
 - b. sudo yum install ruby

- c. Download the agent: \$ wget<https://aws-codedeploy-eu-west-3.s3.eu-west-3.amazonaws.com/latest/install>
- d. Install it: chmod +x ./install

Note: To confirm install is successful, run ‘ls’ command, if ‘install’ comes in green, it is installed properly.

- e. To execute: sudo ./install auto
- f. To check, if our CodeDeploy agent has started: sudo service codedeploy-agent status

Deployment group: These are set of EC2 instances that we’re going to deploy our application to. It means we can have several group in our EC2 instances, like development instances, product instances or 5% of the instances like trial & rest as production.

26. Go to Applications under CodeDeploy (under Services).
27. Click on “Create Deployment Group”.
28. Enter the name of the deployment group.
29. Enter the role created in step 13.
30. Choose the deployment type to be “In-place” right now.
31. Under Environment Configuration, select Amazon EC2 instances (we can also select a group of them).
32. Set the same tag name as specified for EC2 instances created in step 18. We can add multiple tags under this, & all tags coming under this group will be a part of this Deployment groups.
33. Under Deployment Settings, select different strategies to deploy your application, for now select “AllAtOnce”. Even we can create our own Deployment configuration.
34. We can also have a load Balancer, but we’ll disable it for now.
35. Hit on “Create Deployment Group”.
36. Go back to the application & select the “Deployments” tab.
37. Hit on “Create Deployment”.
38. Select the name of the group created above in step 35.
39. Under Revision Type: select “My application is stored in Amazon S3”.

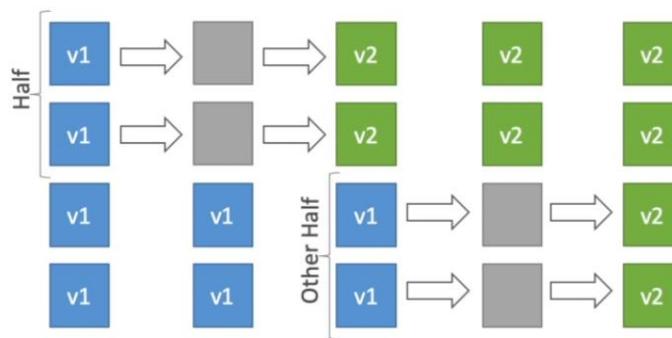
But before that we need to put our application in S3 bucket. So we are going to create it.

1. Go to S3 under Services.
2. Create a bucket, specify it a unique name & hit on ‘Create’.
3. Upload the SampleApp_linux.zip file to the bucket.
4. Click on the file & click on “Copy Path”.
5. Leave other options as default & hit on “Deployment”.

CodeDeploy for EC2 and ASG

EC2

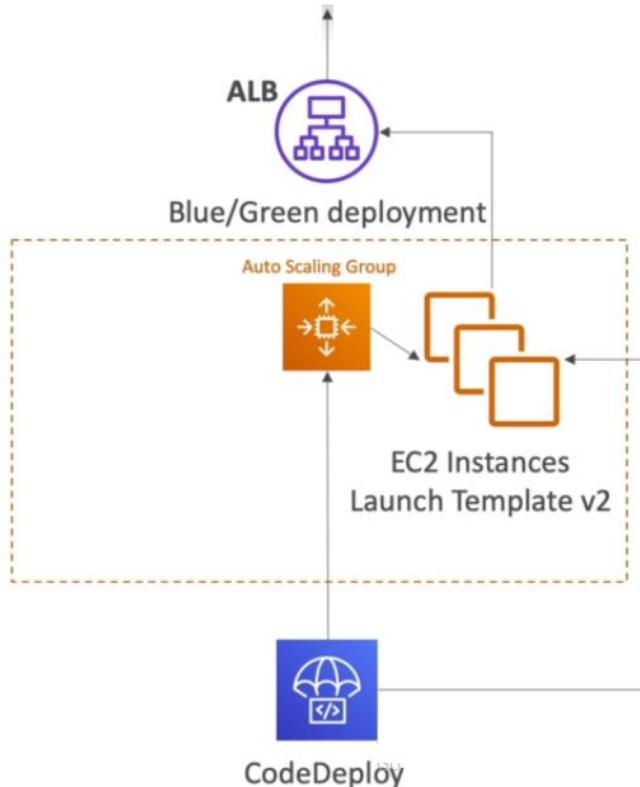
- We need to define how to deploy application using appspec.yml + deployment strategy



- It does in-place update to our fleet of EC2 instances.
- We can use hooks to verify deployment after each deployment phase.

ASG

- Here we perform In place update
 - It is used to update the current EC2 instances.
 - If ASG create instances during deployment, the ASG will get the deployments automatically.
- Blue/Green deployment:
 - It creates a new ASG and settings are copied over.
 - We can choose how long to keep the old instances.
 - To make Blue/Green deployment, we must use ELB on top of ASG.



We've ASG with EC2 instances on Launch Template 1. CodeDeploy deploys to these instances. Then we've Blue/Green deployment with ASG or new EC2 instances with Launch Template v2 will be launched. Then, CodeDeploy makes the deployment to these instances automatically which will be linked to ALB. Hence, the ALB serves both instances at the same time. We can choose how long to keep the old instances and after that previous instances will be terminated.

Rollback

- We can specify automatic rollback options.
- Rollbacks when deployment fails.
- Disable rollbacks in case we do not perform rollback for this deployment.
- If rollback happens, then CodeDeploy redeploys to last good revision as new deployment and in that case we get a new version ID.

CodeStar

- It is an integrated solution that regroups: GitHub, CodeCommit, CodeBuild, CodeDeploy, CloudFormation, CodePipeline, CloudWatch.
- It helps us quickly create “CICD-ready projects for EC2, Lambda Beanstalk

- Supported languages: Python, Ruby, C#, Go, HTML5, Node.js, PHP
- Issue tracking integration with: JIRA/GitHub issues
- Ability to interact with Cloud9 to obtain a Web IDE (not all regions)
- One dashboard for all components.
- It's free service, we are going to pay for the underlying usage
- It has limited customization, so can kick off early

LAB

1. Go to CodeStar under Services
2. We may need to change the region
3. Click on “Start project”
4. Click on “yes, create role”.
5. Select programming language of your choice & under services, check Elastic BeanStalk for now.
6. Select Web Application.
7. Select repository to be AWS CodeCommit.
8. Keep other settings to be default
9. Choose a key pair.
10. Give a name & email id of your choice.
11. Under setting up how to edit code, “Skip” for now.

The code gets sourced, build & deploy. CodeStar is linked to all underlying services. If we click on code, it navigates to source code under repositories & similarly for Build, Deploy, Pipeline, etc.

Under extension we can integrate with GitHub issues or Atlassian JIRA

13. CloudFormation

It provides a simple way to create & manage a collection of AWS resources by provisioning & updating them in a predictable way. It enables us to manage our complete infrastructure or resources in a text file.

Introduction

Infrastructure as Code

IaaS on AWS is done using CloudFormation. Currently, a lot of manual work has been done like code build, code deployment, creating environment which was later balanced by Elastic Beanstalk. These manual work gets tougher to reproduce:

1. In another region.
2. In another AWS account.
3. Within the same region if everything gets deleted.

To reduce this load, all the infrastructure was made in code format. The code is deployed & in turn it creates/ update /delete our infrastructure.

CloudFormation is a declarative way of outlining our AWS Infrastructure, for any resources. For example, within a CloudFormation template, we say:

1. I want a security group.
2. I want two EC2 machines using this security group.
3. I want two Elastic IPs for these EC2 machines
4. I want a S3 bucket.
5. I want a Load Balancer (ELB) in front of these machines,

Then CloudFormation creates these for us in the right order with exact configuration.

The advantages are:

- No resources are manually created, which is excellent for control.
- The code can be version controlled for example using Git.
- Changes to infrastructure are reviewed through code.
- Productivity
 - Ability to destroy & re-create an infrastructure on the cloud on the fly.
 - Automated generation of Diagram for our templates.
- Separation of concern: create many stacks for many applications & many layers. Ex:
 - VPC stacks
 - Network stacks

- App stacks
- Leverages existing templates on the web.
- Leverages the documentation.

Cost

Each resource within the stack is staged with an identifier so, we can easily see how much a stack costs us.

We can estimate the costs of our resources using the CloudFormation template.

Savings strategy: in Dev, we can delete the automation of templates at 5 P.M & recreate at 8 A.M safely.

Working

- Templates have to be uploaded in S3 & then referenced in CloudFormation.
- To edit the template, we can't edit the previous one we've to re-upload the new version of the templates to previous ones.
- Stacks are identified by name.
- Deleting the stack delete the every single stack artifacts that was created by CloudFormation.

Deploying templates

- Manual way
 - Editing templates in the CloudFormation Designer
 - Using the console to input parameters.
- Automated Way
 - Editing the templates in a YAML file.
 - Using the AWS CLI interface to deploy templates.
 - Recommended way when we fully want to automate our flow.

Building Blocks

Template components:

1. Resources: These are the AWS resources declared in the template. (Mandatory) they can be EC2 machines, Elastic IPs, Security group, Load Balancers.
2. Parameters: They are dynamic input for our templates.

3. Mappings: These are the static variables for our templates.
4. Outputs: References to what has been created.
5. Conditionals: List of conditions to perform resource creation.
6. Metadata

LAB

1. Go to CloudFormation under ‘Services’.
2. Under Stack, go to “Create Stack”.
3. Select “Template is ready” option
4. Under specify template – Click on “Upload a template file” & hit on “Choose a file”.
5. Upload the just-ec2.yaml file & click on “Next”. (Make sure that modifications have been made according to the availability zone & other details).
6. Choose a Stack name of your choice & hit on “Next”.
7. We can attach an IAM role to CloudFormation to do the modifications. If nothing is set, CloudFormation uses permissions based on our credentials defined in our account.
8. We can define Rollback in case of failure.
9. We can set the notification options to notify us whatever happens to our CloudFormation tasks.
- 10.Under Stack creation options, we can check the Termination option so that no one deletes the CloudFormation.
- 11.Timeout tells if we don’t want the task to continue, we just want the it to time out after fail.
- 12.Hit next after applying the settings.
- 13.After that we get the template URL which was uploaded to Amazon S3 & we can even get the estimate of the cost.
- 14.Review the options & click on “Create Stack”.

We can look down the events tab. It has a time stamp, status & a type. If we look into the logs, it says firstly we created a CloudFormation Stack & it was User Initiated. Then it logs that we are trying to create an EC2 instance. Check for EC2 instances under Services, we can see that a new instance has been launched. If we look at the tags of it, there’s a logical id, stack id & a stack name.

'Resources' tab tells about the summary of the events happening. Under 'Template', we can view the CloudFormation designer.

Now, we're going to update our stack. It'd be having an EC2 instance with more Security Groups. We'll be creating Elastic IP. Let's go back to 'Stack'.

- 15.Go to CloudFormation under 'Services'.
- 16.Click on 'Stack' & choose your Stack.
- 17.Click on 'Update' button.
- 18.Click on "Replace current template" & "Upload a template file" downloaded ec2-with-sg-eip.yaml.
- 19.Update the instance.
- 20.This time if we navigate to the EC2 instance, we can see that previous instance is been terminated & new instance is been created. Also, 2 new Security groups & a new Elastic IP gets created.
- 21.If we want to get rid of these, we will not shut each instance one by one, instead we delete the whole stack. The resources will get deleted in the right order.

YAML

```

1 invoice:      34843
2 date   :      2001-01-23
3 bill-to:
4     given  :  Chris
5     family :  Dumars
6     address:
7         lines: |
8             458 Walkman Dr.
9                 Suite #292
10            city   : Royal Oak
11            state  : MI
12            postal : 48046
13 product:
14     - sku        : BL394D
15     quantity    : 4
16     description : Basketball
17     price       : 450.00
18     - sku        : BL4438H
19     quantity    : 1
20     description : Super Hoop
21     price       : 2392.00

```

- They have key value pairs (line 1)
- Nested objects can be formed (line 6)
- Support Arrays (line 13)
- Multi-line string (line 7)
- Can include comments.

Resources

They are the core of CloudFormation template. They represent different AWS component that'll be created & configured. They are declared & can reference each other. There are 224 type of resources.

Resources Type identifiers are of the form: **AWS::aws-product-name::data-type-name**.

Here we can find the list of resources:

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html>

When we create resources, it must have a Type & a Properties & all these goes under code block 'Resources'

Points to remember

- We cannot create a dynamic amount of resources. Everything in the CloudFormation template has to be declared. We cannot perform CodeGeneration here.
- Almost all of the services are supported.
- We can work around that using AWS lambda custom resources.

Parameters

It is a way to provide input to our AWS cloudFormation template. Parameters are important to know about if:

- We want to reuse our templates across the company.
- Some inputs cannot be determined ahead of time. For e.g.: the key pair we are going to link to our instances.
- They are powerful, controlled & can prevent errors from happening in our templates, thanks to types.
- They can be used if the CloudFormation resource configuration likely to change in near future. We don't have to re-upload a template to change the content.
- Parameter can be controlled by these settings.
 - Type:
 - String
 - Number
 - CommaDelimitedList
 - List<Type>
 - AWS Parameter (to help catch invalid values – match against existing values in AWS account).
 - Description
 - Constraints
 - ConstraintDescription (String)
 - Min/Max Length
 - Defaults
 - AllowedArrays (array)
 - AllowedPattern (regexp)
 - NoEcho (Boolean) – If we want to hide the contents.

How to reference a parameter

- The `Fn::Ref` function can be leveraged to reference the parameters.
- Parameters can be used anywhere inside the template.
- The shorthand for Reference in YAML is `!Ref`.
- Reference can be used both to reference Parameters & Resources.
 - If we reference a parameter, it returns the value of the parameter.
 - If we reference another resource in our CloudFormation template, it returns the physical ID of the underlying resource. For e.g.: if we reference an EC2 instance, we get the EC2 instance ID.

Pseudo Parameters

AWS offers us pseudo parameters in any CloudFormation template. These can be used anytime & we get a bunch of values we may want to retrieve.

Reference Value	Example Return Value
<code>AWS::AccountId</code>	1234567890
<code>AWS::NotificationARNs</code>	[arn:aws:sns:us-east-1:123456789012:MyTopic]
<code>AWS::NoValue</code>	Does not return a value.
<code>AWS::Region</code>	us-east-2
<code>AWS::StackId</code>	arn:aws:cloudformation:us-east-1:123456789012:stack/MyStack/1c2fa620-982a-11e3-aff7-50e2416294e0
<code>AWS::StackName</code>	MyStack

Mappings

Mappings are fixed variable in our CloudFormation template. They're very handy to differentiate between environments (Dev vs. Prod), regions (AWS regions), AMI types. All the values are hard coded within the template.

```

Mappings:
  Mapping01:
    Key01:
      Name: Value01
    Key02:
      Name: Value02
    Key03:
      Name: Value03
  
```

```

RegionMap:
  us-east-1:
    "32": "ami-6411e20d"
    "64": "ami-7a11e213"
  us-west-1:
    "32": "ami-c9c7978c"
    "64": "ami-cfc7978a"
  eu-west-1:
    "32": "ami-37c2f643"
    "64": "ami-31c2f645"
  
```

Fn::FindInMap – Accessing mapping values

```

AWSTemplateFormatVersion: "2010-09-09"
Mappings:
  RegionMap:
    us-east-1:
      "32": "ami-6411e20d"
      "64": "ami-7a11e213"
    us-west-1:
      "32": "ami-c9c7978c"
      "64": "ami-cfc7978a"
    eu-west-1:
      "32": "ami-37c2f643"
      "64": "ami-31c2f645"
    ap-southeast-1:
      "32": "ami-66f28c34"
      "64": "ami-60f28c32"
    ap-northeast-1:
      "32": "ami-9c03a89d"
      "64": "ami-a003a8a1"
Resources:
  myEC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", 32]
      InstanceType: m1.small
  
```

We use **Fn::FindInMap** to return a mapped value from a specific key.

Syntax: !FindInMap [MapName, TopLevelKey, SecondLevelKey]

Mapping vs Parameters

Mappings are great when we know values in advance that can be taken. They allow safer control over template. They can be deduced from variables such as:

- Region

- Availability zone
- AWS account
- Environment

Whereas Parameters are used when the values are really user specific.

Outputs

- The output section declares optional output values that we can import into other stacks (if we export them first!)
- We can view the outputs in the AWS console or using the AWS CLI.
- They're very useful for example if we define a network CloudFormation, and output of the variables such as VPC ID & subnet IDs and later we can use this in other CloudFormation template.
- It is the best way to perform some collaboration cross stack, as we let expert handle own part of the stack.
- We cannot delete a CloudFormation stack if its output is being referenced by other CloudFormation stack.
- E.g.:

Creating a SSH Security Group as part of one template. Later, the value of Security Group ID will be used by other template. “Export” block is used to export the value & later we will import it.

```
Outputs:
StackSSHSecurityGroup:
  Description: The SSH Security Group for our Company
  Value: !Ref MyCompanyWideSSHSecurityGroup
  Export:
    Name: SSHSecurityGroup
```

After specifying the Export block, the “StackSSHSecurityGroup” is going to be exported as a Name “SSHSecurityGroup”. To import the value, we'll use a **Cross Stack Reference**.

We'll be creating a second template that leverages the security group.

```

Resources:
MySecureInstance:
  Type: AWS::EC2::Instance
  Properties:
    AvailabilityZone: us-east-1a
    ImageId: ami-a4c7edb2
    InstanceType: t2.micro
    SecurityGroups:
      - !ImportValue SSHSecurityGroup

```

For this, we use the `Fn::ImportValue` function. We can't delete the underlying stack until all the reference are deleted too.

Conditions

They're used to control the creation of resources or outputs based on some conditions. We can define condition based on our choice, but most common ones are:

- Environment (dev /test/ prod): if we are in a specific environment, then create then create or don't create the resource.
- AWS region
- Any parameter value

Each condition can reference another condition, parameter value or mapping.

Syntax:

Conditions:

```
ResourceName: Equals [ !Ref EnvType, prod ]
```

E.g.:

```

Conditions:
| CreateProdResources: !Equals [ !Ref EnvType, prod ]

```

The logical ID is for us to choose. It's how we name (CreateProdResources, it can be changed) the condition. The intrinsic function can be any of the following:

- `Fn::And`
- `Fn::Equals`
- `Fn::Not`

- Fn::Or

E.g.:

Conditions can be applied to resources/ Outputs/ etc.

Resources:

MountPoint:

Type: "AWS::EC2::VolumeAttachment"

Condition: CreateProdResources

Intrinsic Function

- Ref
- Fn::GetAtt
- Fn::FindInMap
- Fn::ImportValue
- Fn::Join
- Fn::Sub
- Condition Functions (Fn::If, Fn::Not, Fn::Equals)

Fn::GetAtt

- Attributes can be attached to any resources we create.
- To know the attributes of the resource, the best place is to look at is the documentation. (Go to template resource type reference: <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html> -> choose any of the resource type -> go to its Return Values & Ref returns the ID of the instance & Fn::GetAtt we are able to see which values for the specified resource).

```
Resources:
  EC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: ami-1234567
      InstanceType: t2.micro
```

- For e.g.: if we want to get the AZ of an EC2 machine, we've our resource blog with an ImageId & an instance type.

```
NewVolume:
  Type: "AWS::EC2::Volume"
  Condition: CreateProdResources
  Properties:
    Size: 100
    AvailabilityZone:
      !GetAtt EC2Instance.AvailabilityZone
```

We create an EBS volume, for this we are going to create a new volume under resources & declare Type as EC2 volume. Under Properties we can see the AvailabilityZone is using the GetAtt function: EC2Instance.AvailabilityZone, so now EC2 instance is coming straight out of this name on the first image.

Fn::Join

- Join values with delimiter
- Syntax: !Join[delimiter, [comma-delimited list of values]]
- E.g.: to produce “a:b:c”, we use --> !Join [“:”, [a,b,c]]

Fn::Sub

- Fn::Sub or !Sub as a shorthand is used to substitute variables from a text. It's a very handy function that allows us to fully customize our templates.
- For e.g.: we can combine Fn::Sub with References or AWS Pseudo variables.
- String must contains a \${VariableName} & will substitute them.
- Syntax: !Sub
 - String

- {Var1Name: Var1Value, Var2Name: Var2Value}

Rollbacks

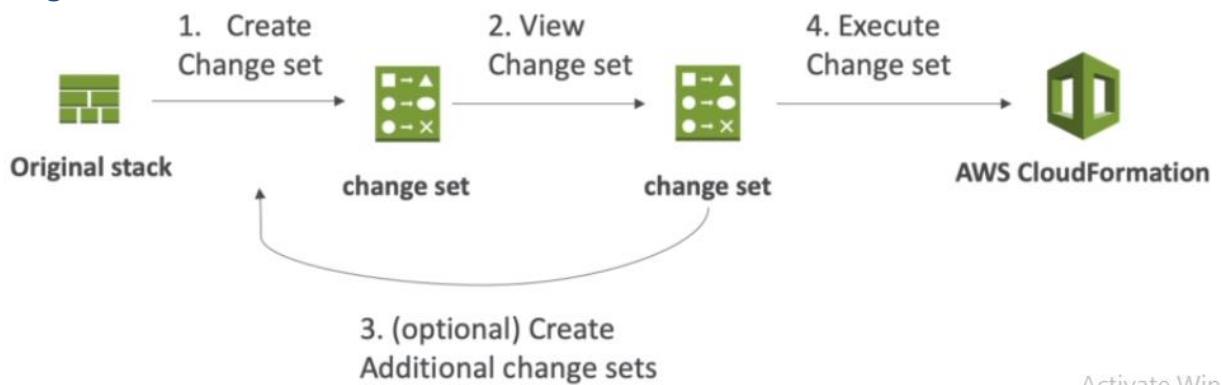
- Stack Creation Fails:
 - Default: everything roll backs (gets deleted). We can view the log file.
 - We also have an option to disable the option & troubleshoot what happened.
- Stack Update Fails;
 - The stack automatically rolls back to previous working state.
 - We also have the ability to see the log & error messages.

LAB

1. Create a Stack (as done in LAB of CloudFormation)
2. After successful creation of Stack, Click on 'Update'.
3. Upload a new YAML file with change in any of the undesired values.
4. Hit on 'Create'.
5. We can see under Events that Update has failed & Update Rollback in progress.

Changeset, Nested Stack, and stack set

Changeset



- Changeset won't say if the update will be successful.

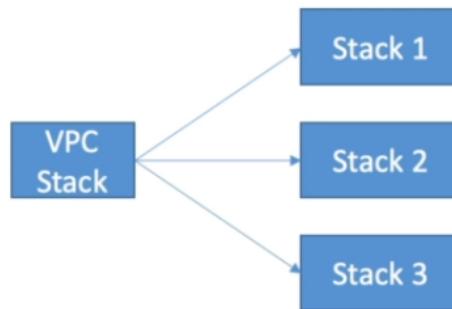
Nested Stack

- They're part of CloudFormation stacks as a part of other stacks.
- They allow us to update and isolate repeated patterns/common components in our separate stack and call them from other stack.

- Example:
 - A Load-Balancer configuration that is re-used
 - Security group pattern that is re-used.
- Nested stack are considered best practices.
- To update the nested stack, always use the parent (root stack).

Cross stack vs. Nested stack

- Cross stack
 - They're helpful when stacks have different life cycles.
 - We use Output Exports and Fn::ImportValue.
 - When we need to pass export values to many stacks (VPC ID, etc.)



The VPC stack is going to be reused across many different stacks and the value needs to be exported.

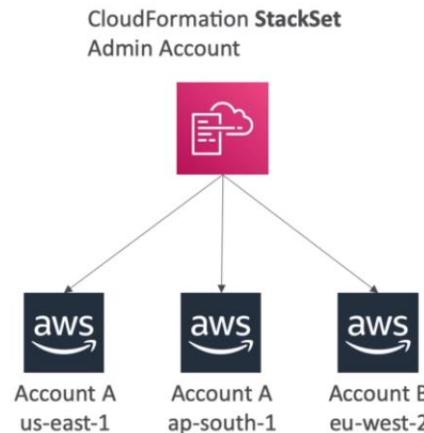
- Nested stack
 - Helpful when component must be re-used.
 - Ex: re-use how to properly configure an ALB.
 - The nested stack is only important to the higher level stack.



We've an application stack . there we've RDS nested stack, ASG stack, and ELB stack. We then create a second nested stack, with same nested stack.

StackSet

- It is used to create, update or delete stack across multiple accounts and regions with a single operation.
- Administrator account to create StackSets.
- Trusted accounts to create, update, delete stack instances from stacksets.
- When we update a stackset, all associated stack instances are updated through all accounts and regions.



We create a stackset in admin account, and it gets applied to different accounts in different regions.

14. Monitoring

Monitoring is important because we can troubleshoot & stop environmental damage. It helps in internal monitoring so that:

- We can prevent issues before they happen
- Performance & cost
- Trends that how things scale
- Learning to improve using trends.

Monitoring in AWS can be done using:

- CloudWatch: It monitors on the basis of
 - Metrics: Collect & track key metrics.
 - Logs: Collect, monitor, analyze & store log files.
 - Events: Send notifications when certain events happen in AWS.
 - Alarms: react to real-time to metrics/events.
- AWS X-Ray:

- It allows us to troubleshoot application performance & errors.
- Distributed tracing of Micro services. It means if we've a lot of services, calling one another or if we've a lot of components interacting with one another then we can see how application makes calls & how long it takes and it can be traced.
- CloudTrail:
 - Allows us to do internal monitoring.
 - Audit changes made to AWS resources by users.

CloudWatch

Dashboard

1. Go to ‘CloudWatch’ under Management & Governance section.
2. Go to Dashboard & create one.
3. For now, let us create a Text Widget and it uses Markdown language.
4. Create a widget of your choice.
5. Now let’s add a line widget.
6. To configure the widget, choose any of the metrics created till now.
7. Select EC2, & select Pre-Instance metrics.
8. Here, the metric name is important to remember. It’s basically CPU, Disk, Metadata, Network & Status related.
9. Select “CPU Utilization” for now & click on “Create Widget” & ‘Save’.
10. Add some more widget to your Dashboard.
11. Add “Stacked Area”

Alarm

- It helps us to notify when particular threshold are hit.
- Alarms can be in 3 states:
 - OK: when alarm is not doing anything.
 - INSUFFICIENT DATA: when we don’t send enough data for alarms.
 - ALARM: when threshold is being passed.
- Period: It is the length of the time to evaluate the metric. In High resolution custom metric, we can choose 10sec or 30sec as the evaluation period of our alarm.

LAB

1. Go to Alarm under “Management & Governance”.
2. Create an Alarm.
3. Select a metric, of which you want to create an Alarm.
4. Add a condition to it.
5. Add a threshold value to your Alarm
6. Choose the list of email to whom you want to send Notifications.
7. Add an Alarm Name.
8. Preview & Create.

Events

It indicates changes in our AWS environment. AWS resources generate events when their state changes. When resources changes their state, they automatically send events to Event Stream. We then create certain rules that matches selected events in stream and route them to targets to take action. We can also use rules to take actions on a pre-determined schedule. We can either have a

- Schedule so we can define Cron Job to schedule events in CloudWatch so that we can trigger little notification on demand.
- Event pattern in CloudWatch events which allow us to define rules like if a service does something we’re going to trigger an event. E.g.: CodePipeline state changes.
- Trigger may be Lambda function, SQS/ SNS/ Kinesis.
- CloudWatch events creates a small JSON documents & will give you information about the change.

LAB

1. Go to CloudWatch under Services.
2. Click on Events.
3. Create a rule. We can select either “Event pattern” or ‘Schedule’.

Note: Schedule helps us create an event every n number of minute or we can setup a Cron expression. After completing the details, an JSON is produced which can be seen in the “Show sample events”

4. Targets can be also be added.
5. After selecting “Event pattern” in step3, select Service name to be CodePipeline.
6. Select either of option in the event type.
7. Select “Specific state(s) for now & select the state to be ‘Failed’ in the dropdown options.
8. We can preview the event pattern which can be copied to clipboard or edited if we need to add some more details.
9. Select target to be SNS topic. Select a Topic (need to be previously created).
10. Configure the input.
 - a. Matched event: if we pasted the entire events (from step 8) that will be matched into the alarm.
 - b. We can either have different optimization if we want to modify the events.
11. Click on “Configure details”
12. Give the rule a name

Logs

It help us to create & store logs from our resources, services, and application & application can send logs to CloudWatch using the SDK. It helps us to visualize & correlate our logs and metrics for full operational visibility. CloudWatch logs can go to:

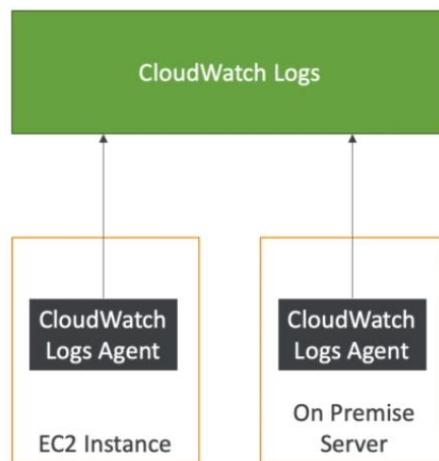
- Batch exporter to S3 for archival.
- Stream to ElasticSearch Cluster for further analytics.
- CloudWatch logs can use filter expressions if we want to search through them.
- Architecture:
 - Log groups: arbitrary name, usually representing an application.
 - Log stream: instances within application/ log files/ containers.
- Can define log expiration policies (can be set to never expire, 30 days, etc.)
- AWS CLI can be used to tail the CloudWatch logs.
- To send CloudWatch, make sure IAM permissions are correct.
- For security, all the logs can be encrypted using KMS at the group level.

LAB

13. Go to CloudWatch under Services.
14. Click on Logs.
15. Click on any of the log group to view the logs.
16. We can set the retention policy to never expire, or set to custom number of days.
17. If we click on the log group, & click on action then we get the option to Export the data to S3. Even, we can stream the data to AWS Lambda or Amazon ElasticSearch Service.
18. We can create & delete a Log stream.
19. We can even search in the log files.

Agents & Log agents

1. By default, no logs from EC2 machine will go to CloudWatch.
2. We need a CloudWatch agent to push the log files.



Our EC2 instances will have CloudWatch logs agent, for example sending the logs to CloudWatch logs. For it to work, our EC2 instances must have an IAM Role.

3. The CloudWatch log agent can be set-up on On-premise servers.
4. CloudWatch logs agent is the older one and can send logs to CloudWatch logs whereas CloudWatch unified agent, it is the newer one and can collect additional system level metrics such as RAM, processes, etc. It can also send logs to CloudWatch logs. It is a centralized configuration using SSM parameter store.

5. It provides logs for CPU, Disk metrics, Disk IO, RAM, Netstat, Processes, Swap space.
6. It can be collected directly on EC2 instances/Linux server.

Metrics

To view metrics directly, we go to Metrics section in CloudWatch. Instead of creating Dashboards, Widgets & things, we can view all in Metrics & CloudWatch provides metrics for every services in AWS.

Note:

- CloudWatch provides metrics for every services in AWS.
- Metric is variable to monitor (CPU Utilization, NetworkIn, etc).
- Dimension is an attribute of the metric (instance id, environment, etc.)
- We can have 10 dimension per metric.
- Metrics have timestamps.
- By default, our EC2 instances have metrics every 5 minutes.
- With detailed monitoring (for a cost), we get data every minute.
- If we want to more prompt scale, we can use detailed monitoring.
- Free tier allows for 10 detailed monitoring metrics.
- EC2 memory usage is by default not pushed (must be pushed from inside the instance as a custom metric).
- We can define & send our own custom metrics to CloudWatch
- We have the ability to use dimension to segment metrics such as
 - a. Instance.id
 - b. Environment.name

We've 10 different dimension for these metrics.

- Metric resolution
 - a. Standard resolution is 1 minute
 - b. High resolution: Up to 1 second (**StorageResolution** API parameter is used to call High resolution parameter) – Higher cost.
- To send a metric to CloudWatch, we need to use **PutMetricData** API.
- Exponential backoff can be used in case of throttle errors. The idea behind exponential backoff is to use progressively longer waits between numbers

of retries for consecutive error responses. It can be based on operation being performed as well as local factors such as network latency.

- The variable in the metrics can be variable to monitor for each services (like CPU utilization, NetworkIn, etc.)
- Metrics belongs to Namespaces. A namespace is a container for CloudWatch Metrics & metrics in different namespace are isolated from each other so that metrics from different application are not mistakenly aggregated into same statistics.
- Dimension is an attribute of a metric. It is a name/value pair that is part of identity of the metric. For e.g. we may have CPU utilization of a metric but the dimension of it may be instanceID because we have many CPU utilization of our instances.
- We can have 10 dimension per metric.
- Metrics have timestamps.
- Can create CloudWatch dashboard of metrics.

The difference between CloudWatch & CloudTrail is that CloudWatch is for logging & monitoring whereas CloudTrail is for auditing. CloudTrail monitor entire AWS environment. If we create a new User or Role, or a S3 bucket, that's all logged through CloudTrail. So, CloudWatch is for performance monitoring within AWS and CloudTrail is for auditing, it creates an audit trail that what people are doing with our AWS account.

Metric Filters

- CloudWatch can use filter expression for:
 - Finding specific IP inside our log.
 - Count occurrences of a particular word inside log.
 - Can be used to trigger alarm if we reach the threshold.
- These filter do not retroactively filter data. Filter only publish the metric data points for event that happen after the filter was created.



We've CloudWatch logs agent on EC2 instance streaming into CloudWatch logs. The logs create a metric filter. Suppose, it defines number of errors in the log and if we reach a threshold, alarm gets triggered and it sends data to SNS topic.

EventBridge

- It is the next evolution of CloudWatch events.
- When we use the CloudWatch events, by we use the Default Event bus which is generated by AWS Services.
- EventBridge has multiple buses:
 - Partner Event Bus: It receives events from SAAS service/Application, not our AWS Services.
 - Custom Event Bus: Our application can publish their own events.
- Events buses can be accessed by the cross accounts
- EventBridge can analyze the events in the bus and infer the schema (how the data is structured).
- It allows us to generate code for our application. The application knows in advance how data is structured in the Event Bus.
- Schema can be versioned.

EventBridge vs. CloudWatch

- EventBridge builds upon and extends CloudWatch Events.
- It uses the same service API and endpoint, and the same underlying infrastructure.
- EventBridge allows extension to add event buses to our custom applications and third-party SAAS apps.
- EventBridge has a Schema Registry Capability.
- EventBridge has a new capabilities.
- Overtime, CloudWatch Events will be replaced with EventBridge.

X-Ray

When we do debugging in production, the good old way is:

- Test locally
- Add log statements everywhere
- Re-deploy in production

And from the logs, we can find out what is breaking. But, it's not better way of doing things. Also, formats of the log files generated by the application using CloudWatch & analytics is hard.

- So, if we've just single big application, debugging is easy, while when it comes to distributed services & we've multiple micro-services running then it becomes a nightmare.
- X-Ray gives us a visual analysis of application.

Advantages



- We can troubleshoot the performance of our application & identify bottlenecks.
- We can understand the dependencies in our micro-services architecture & identify how our micro-services interact with one-another.
- We can pinpoint that which service is giving us issues.
- We can review how each request is behaving.
- We can also find errors & exception (as we can see in some of the circle, there's orange part too).
- Are we meeting the time service level agreement?
- We can understand which service really slows down.
- We can identify which users are impacted by our errors.

Compatibility

AWS is compatible with many services:

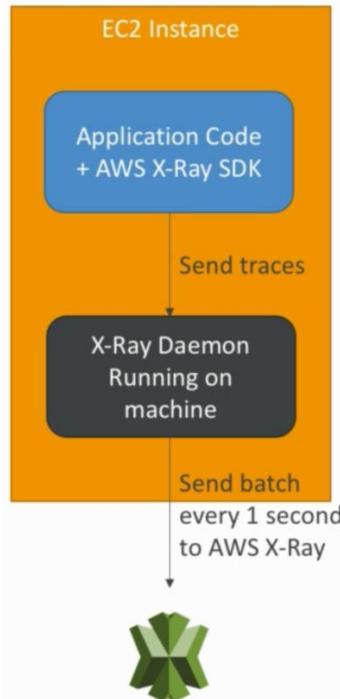
- AWS Lambda
- Elastic Beanstalk
- ECS
- ELB
- API Gateway
- EC2 instances or any application server.
- Or even something which we have on-premise.

Working

X-Ray leverages tracing, and tracing is an end to end way to follow a “request”. When we make a request, each component that is dealing with the request add its own trace.

- Trace is made up of segment & segments are made up of sub-segments.
- Notes or comments can be added to traces to provide extra information.
- We've the ability to trace:
 - Every request
 - Or a sample request (as we may be interested to get only a % of total request or at a particular rate like only 5 requests per minute).
- Security:
 - We can use IAM for authorization
 - KMS for encryption at rest.

Enabling X-ray



1. Our code (Java, Python, Go, Node.js, .NET) must import the AWS X-Ray SDK.
 - a. Very little modification is needed in the code.
 - b. The application SDK will then capture:
 - i. HTTP/ HTTPS request
 - ii. Database calls (MySQL, PostgreSQL, DynamoDB)
 - iii. Calls to AWS Services.
 - iv. Queue calls (SQS).
2. Install the X-Ray Daemon or enable X-Ray AWS configuration.
 - a. X-Ray Daemon works as a low level UDP packet interceptor (Linux/ Windows/ Mac.) Daemon is process that runs in background and performs specified operations at pre-defined times or in-response to certain events.
 - b. If we've AWS Lamda /other AWS services that already have integration with X-Ray, then they runs Daemon for us.
 - c. Each application must have the IAM right to write the data to X-Ray.

Output

1. X-Ray service collects data from all different services.
2. Service map is computed from all the segments & traces.
3. X-ray is graphical, so even non-technical people can understand & troubleshoot it.

Troubleshooting

- If X-ray is not working on EC2
 - Ensure that EC2 IAM role has proper permissions.
 - Ensure the EC2 instance is running the X-Ray daemon.
- To run on AWS Lambda:
 - We need to ensure that it has an IAM execution role with proper policy (AWSX-RayWriteOnlyAccess).
 - Ensure that X-Ray is imported in the code.
 - Ensure that X-ray integration is enabled on AWS Lambda

LAB

1. Go to X-Ray under Services. (for now, select region as ‘N. Virginia’)
2. Hit on “Get started”.
3. Select “Instrument your application”.
4. Click the language of your choice.
5. Under implementation,
 - a. Firstly, we need to set the environment.
 - b. Later, we need to run the AWS X-Ray daemon as X-Ray SDK does not send trace directly to AWS X-Ray to avoid choking. We need to run Daemon & Daemon uploads them in batches & to this, we can see the instruction how to run on different environments like EC2 Linux, EC2 Windows Server, Amazon ECS, Elastic BeanStalk& Lambda.
 - i. In Linux machine, there’s just a ‘curl’ to download the RPM file (RPM stores installation packages on Linux OS)
 - ii. To install X-Ray on Beanstalk, as we remember Beanstalk can be extended & .ebextensions files can be used & inside .ebextension directory we’ll have xray-daemon.config file which will have 3 lines of code which says x-ray is enabled.

```
option_settings:
  aws:elasticbeanstalk:xray:
    XRayEnabled: true
```
6. For now, go back to step3 & select “Launch a sample application”.
7. Click on “Launch sample application”

8. Choose “Template is ready” and while specifying the template & keep the default URL & hit on ‘Next’.
9. Select any of the Subnets & Parameters & hit ‘Next’.
10. For now, we don’t need to configure any of the stack option, so hit on ‘Next’.
11. Under ‘Capabilities’, check the checkbox to acknowledge that the CloudFormation will create IAM resources.
12. Hit on ‘Create Stack’.

We can see the ‘Template’ tag that it contains 2 parameters, VPC & Resources. For the resources, an IAM role & a policy is created. A ‘Ref’ function links the policy to the role. Then we create an Elastic Beanstalk application.

13. Go to ‘Output’ tab & we can see that 4 application is been created. Hit on the IP address.
14. Click on the start button on the page. It is going to generate signups (10 signups per minute, i.e., one signup per 6sec. each minute there’ll be a duplicate signup & it should trigger an error).
15. After completing all the request, click on ‘Done’.
16. After computation, a map is being created. We can view that the frontend is calling a meta-data service (169.254.169.254), a DynamoDB table & calling to AWS SNS for notification.
17. We can see the orange part in the frontend & DynamoDB.
18. Click on EC2 instance.
19. A distribution graph is generated & let’s click on the Errors (orange box) & click on view traces. (The Green represents ‘OK’, Red represents ‘Fault’, & Violet represents ‘Throttle.’)
20. We can see that there are 2 errors, the sign-up URL & the icon (as the chrome tab has no icon).
21. Click on the “Signup URL”, we can see the Trace list that the Response was 409 & Response time was 30ms.
22. Click on either of the Traces. We can see that the frontend made a request to a DynamoDB & the POST for the request was signup.
23. Click on the response code ‘400’ & lot many other information comes up.
24. Similarly, we can look up at the other traces in any of the 4 services.
25. Delete the stack after use.

API

- X-Ray Write API: It writes data into the X-Ray service.

```

"Effect": "Allow",
"Action": [
    "xray:PutTraceSegments",
    "xray:PutTelemetryRecords",
    "xray:GetSamplingRules",
    "xray:GetSamplingTargets",
    "xray:GetSamplingStatisticSummaries"
],
"Resource": [
    "*"
]

```

arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess

- PutTraceSegments: Uploads segments documents to AWS X-Ray
- PutTelemetryRecords: It is used by the X-Ray Daemon to upload info like:
 - SegmentRecieveCount
 - SegmentRejectedCounts
 - BackendConnectionErrors
- GetSamplingRules: When we change the sampling rules in the UI, all the X-Ray Daemons gets automatically updated to know when to send data into X-Ray. For or sampling rules to know how Daemons are changing, GetSamplingRules authorization and permissions are necessary.
- GetSamplingTargets
- GetSamplingStatisticsSummaries
- The X-Ray Daemon needs to have IAM policy authorizing the correct API calls to function correctly.
- X-Ray Read API: it is a managed policy for reading.
 - GetServiceGraph: it is used get the main graph in the console.

- BatchGetTraces: It retrieves a list of traces specified by ID. Each trace is a collection of segment documents that originates from a single request.
- GetTraceSummaries: retrieve IDs and annotations for traces available for specified time using filters. To get full traces, pass the trace IDs to get BatchGetTraces.
- GetTraceGraph: Retrieves a service graph for one or more specific trace IDs.

Exam-tips

1. The X-Ray daemon / agent must have a configuration to send traces to the account & also cross-account.
 - a. To make this work, we need to make sure that the IAM Role are configured correctly & the agent will assume the role & send it back to the account.
 - b. We need this because we could have a central account for all our application tracing across all our AWS sub-accounts which is a common use case & it's just a configuration in X-Ray daemon and it is all about IAM permissions.
2. Instrumentation means the measure of product performance, diagnose errors and to write trace information.
3. To instrument our application code, we need to use X-Ray SDK.

Example for Node.js & Express

```
var app = express();

var AWSXRay = require('aws-xray-sdk');
app.use(AWSXRay.express.openSegment('MyApp'));

app.get('/', function (req, res) {
  res.render('index');
});

app.use(AWSXRay.express.closeSegment());
```

4. Application code can be modified to customize & annotate the data that SDK sends to X-Ray using interceptors, filters, handlers, middleware, etc.
5. Segments: It is what we see in the UI. Each application/service sends them.

6. Subsegments: If we need more details in our segment.
7. Trace: Segments collected together to form an end-to-end trace.
8. Sampling: in case we are flooding the AWS X-Ray service with too much of data, Sampling can be used. Sampling is used to decrease the amount of data to X-Ray to reduce the cost. Even we can define, we want to send only 5% of traces to X-Ray or just a sample of them known as sampling.
 - a. By default, the X-Ray SDK records the first record each second known as reservoir which ensures that at least one trace is recorder per second as long as the service.
 - b. 5% is the rate at which additional request beyond the reservoir size are sampled.
 - c. For ex: if we set the reservoir size to 50, and sampling rate to 10% then if 100 request per second per seconds match the rule, the total sampled is 55 requests per second.
 - d. We can create our own rules with reservoir and rate

Example Higher minimum rate for POSTs

- Rule name – **POST minimum**
- Priority – **100**
- Reservoir – **10**
- Rate – **0.10**
- Service name – *
- Service type – *
- Host – *
- HTTP method – **POST**
- URL path – *
- Resource ARN – *

- e. We can send 10 request per second to X-Ray and 10% of the other request will be sent.

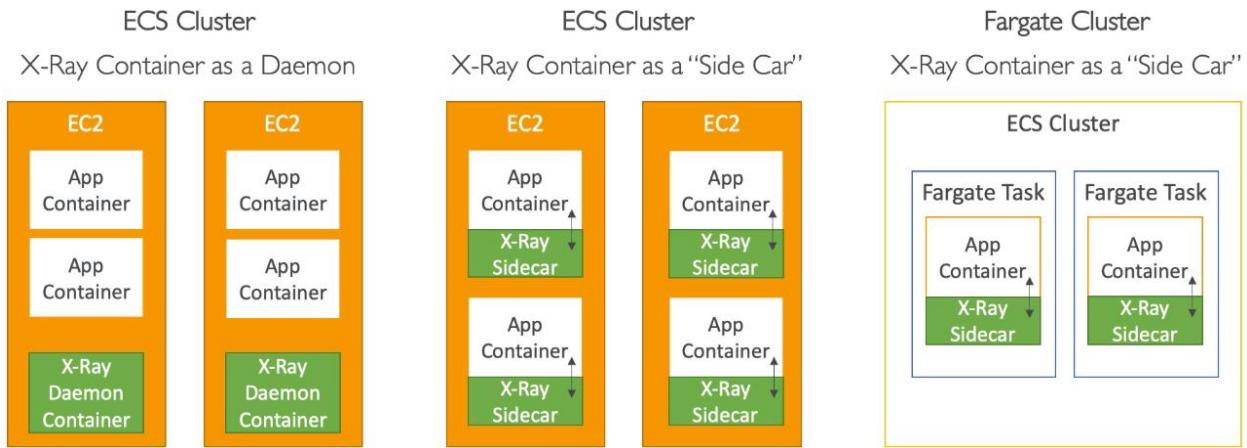
Example Debugging rule to trace all requests for a problematic route

A high-priority rule applied temporarily for debugging.

- Rule name – DEBUG – history updates
- Priority – 1
- Reservoir – 1
- Rate – 1
- Service name – Scorekeep
- Service type – *
- Host – *
- HTTP method – PUT
- URL path – /history/*
- Resource ARN – *

- f. For debugging, we send all request, not losing any traces.
- g. If we change the sampling rules in the console, we do not need to restart our applications, neither we to do anything in SDK. The Daemon send the right amount of data in the X-Ray service as it knows how to get these sampling rules.
9. Annotations: key value pairs that can be added to our traces & segments & it is used to index them.
10. Meta-data: they are also key-value pairs but they are not indexed & not used for searching.
11. The X-Ray daemon/agent has a config to send traces to cross account.
 - a. Make sure the IAM role must be correct to send traces to X-Ray.
 - b. It allows to have a central account for all our application tracing.
12. Different mode of running the X-Ray is
 - a. EC2 or On-Premise:
 - i. In this case, our Linux system must run the X-Ray Daemon.
 - ii. Also, if it is an EC2 instance we need to attach proper IAM role.
 - iii. If it is an On-premise instance, we need to make sure that some AWS credentials are loaded into that machine.
 - b. If we use X-Ray with AWS Lambda:
 - i. We need to make sure that X-ray integration is ticked on Lambda. (Lambda runs Daemon for us.)
 - ii. IAM role in this is a Lambda role.
 - c. X-Ray with BeanStalk
 - i. We need to set configuration on BeanStalk console.

- ii. Or we can use BeanStalk console (.ebextension/xray-daemon.config)
- iii. Make sure IAM policy are configured correctly.
- iv. Make sure application service is instrumented with X-Ray SDK.
- d. X-ray on ECS /EKS /Fargate (Docker):



- i. We need to create a Docker image that runs Daemon or use the official X-Ray official Docker image.
- ii. We need to ensure that port mappings & network settings are correct & IAM task roles are defined.
- iii. On ECS Cluster, to run Daemon is by using container as Daemon itself. Say, we've 2 EC2 instance in our ECS Cluster, and we manage those EC2 instances and we're going to run Daemon container on every instances. So, we can launch App containers on the EC2 instances
- iv. On ECS Cluster, X-Ray container as a side car.
- v. On Fargate Cluster, X-Ray container as a side car: Here we do not have control over the instances. So we cannot use the X-Ray Daemon container. We've to use the X-Ray Container as a Side-Car pattern.

ECS + X-Ray: Example Task Definition

```
{
  "name": "xray-daemon",
  "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/xray-daemon",
  "cpu": 32,
  "memoryReservation": 256,
  "portMappings" : [
    {
      "hostPort": 0,
      "containerPort": 2000,
      "protocol": "udp"
    }
  ],
  {
    "name": "scorekeep-api",
    "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/scorekeep-api",
    "cpu": 192,
    "memoryReservation": 512,
    "environment": [
      { "name" : "AWS_REGION", "value" : "us-east-2" },
      { "name" : "NOTIFICATION_TOPIC", "value" : "arn:aws:sns:us-east-2:123456789012:scorekeep-notifications" },
      { "name" : "AWS_XRAY_DAEMON_ADDRESS", "value" : "xray-daemon:2000" }
    ],
    "portMappings" : [
      {
        "hostPort": 5000,
        "containerPort": 5000
      }
    ],
    "links": [
      "xray-daemon"
    ]
  }
}
```

- The X-Ray Daemon is going to be running and the container port 2000 is mapped to the EC2 instance and the protocol is UDP.
- The environment variable called AWS_XRAY_DAEMON_ADDRESS. It helps the X-Ray SDK to find the X-Ray Daemon.
- We also need to link the above 2 things together from a networking standpoint.

CloudTrail

- It provides governance, compliance, and audit for our AWS account.
- CloudTrail is enabled by default.
- We can get the history of events/ API calls made within our AWS account through our:
 - Console
 - SDK
 - CLI
 - AWS Services
- We can put from CloudTrail into CloudWatch logs.
- If a resource is deleted in AWS, look into our CloudTrail first.

LAB

1. Go to CloudTrail under Services.
2. Click on any of the events.
3. Click on “View event”.
4. We can see a JSON data giving detailed log of event.
5. We can also see the event history & even filter the logs by different category.

CloudTrail vs CloudWatch vs X-Ray

- CloudTrail:
 - Audit API calls made by users/ services/ AWS console
 - Useful to detect unauthorized calls or root cause of changes
- CloudWatch:
 - It metrics overtime for monitoring
 - Stores application logs
 - Alarms to send notification in case of unexpected metrics.
- X-Ray:
 - Automated trace analysis & central service map visualization.
 - Latency, error & fault analysis.
 - Request tracking across distributed system.

15. AWS Integration & Messaging

If we want to deploy multiple applications, they need to communicate. Our service need to share information & share data. There are 2 patterns of application communication out there:

1. Synchronous communication (application to application like if provide a buying service then we need to provide shipping service).
2. Asynchronous or event based communication (application to queue to application). The applications are not directly connected.

Synchronous between applications can be problematic if one service bury down another due to sudden spikes of traffic or other condition. For e.g.: if we've a video encoding service & we need to encode 1000 videos, but usually it's 10, our

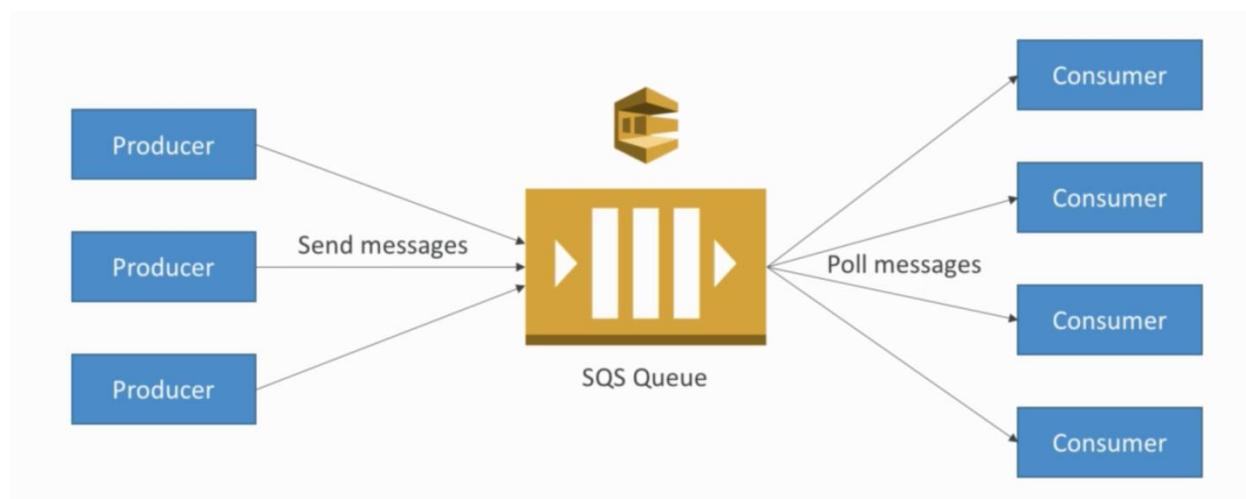
encoding service is going to be overwhelmed, we are going to have outages (a period when service is going to be unavailable).

In this case, it's better to decouple our application, & the decoupling layer will scale for us using:

1. SQS (Simple Queue Service): Queue model
2. SNS (Simple Notification Service): pub/sub model
3. Kinesis: Real time streaming model like big-data.

These 3 things scale independently from our application.

SQS



SQS takes messages & messages will be sent by producers. We can have as many producers as we want & on the other end we are going to have consumers & consumer polls messages from the SQS Queue. Like our buying service was Producer & Shipping service is Consumer. Also, we can have as many consumer as we want. Also, number of producers need not to be same as number of producers, so these can be scaled independently.

Standard Queue

- Oldest & most stable AWS service.
- Fully managed, we don't need to setup a queue, just create a queue & it'll be done.
- Scales from 1 message to 10000 messages per second.

- Default retention of message is 4 days, with a maximum of 14 days inside queue. So we need to consume the message within this period else it'll be gone. No limitations on number of message that can be set up in a queue.
- Low latency (<10ms on publish & receive).
- There's horizontal scaling in terms of consumers.
- Can have duplicate messages (at least once delivery, occasionally).
- Can have out of order message
- Limitations of 256KB of message can be sent.

Producing Messages

- Messages up to 256 KB are sent to SQS by producers using the (SendMessage API)
- The message persists in SQS queue until consumer deletes it.

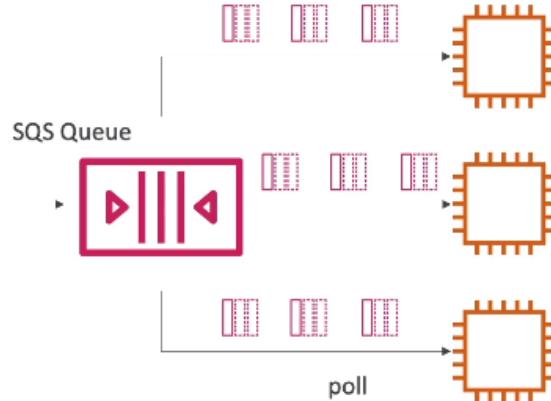
Consuming Messages

- Consumers are applications that we've to write with some code (like running EC2 instances, servers, or AWS Lambda).



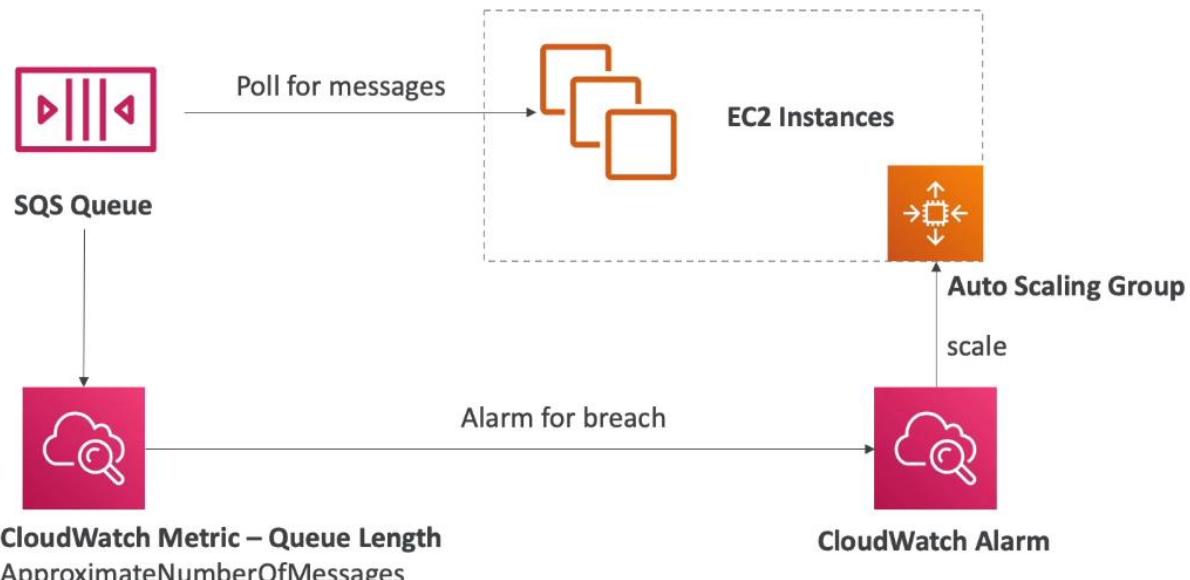
- Our queue has a consumer and it polls for message (receive up to 10 messages at a time).
- The consumer has responsibility to process these messages like inserting orders in DB
- Later, the messages are deleted using DeleteMessageAPI do that these messages are not consumed by the other consumer later.

Multiple EC2 Instances Consumers



- Consumers receive and process message in parallel.
- If a message is not processed by a consumer, it'll be received by another consumer and hence we've at least once delivery.
- If we need to increase the throughputs due to more messages then we can add consumers and do horizontal scaling.

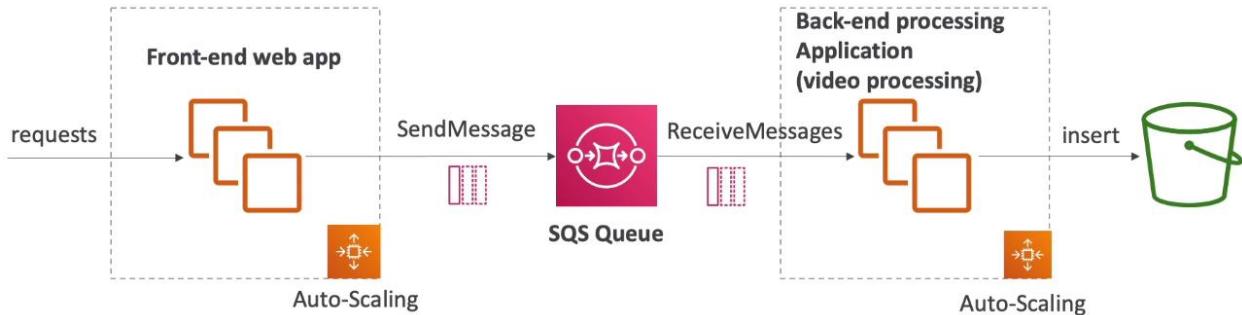
SQS with ASG



- Consumers will be running on EC2 instances inside ASG and they'll be polling for messages from SQS queue.
- ASG needs to auto-scaling on some kind of metric & metric is Queue Length. It's called ApproximateNumberOfMessages which is a CloudWatch metric available in any SQS queue.

- We can setup an alarm that whenever queue length goes above certain level, then setup a CloudWatch Alarm which increases the capacity of ASG

SQS to decouple between application tiers

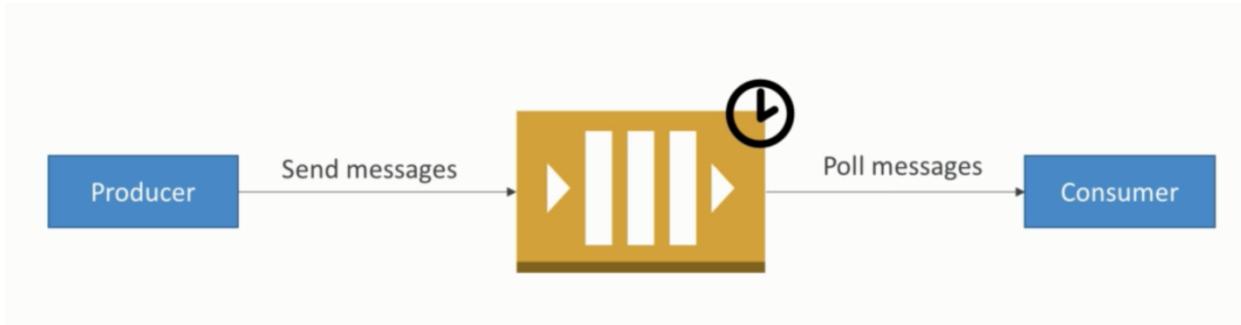


- The Front-end web app processes videos. We can have application which takes request and processes it into S3 bucket, but it may very long to do which may slow down our websites.
- Instead, we'll decouple our application, where the request of processing the file and the actual processing of a file can happen in 2 different applications.
- Whenever a file process request is received, a message is sent to SQS queue.
- The second backend tier which will be in its own ASG receives this messages, processes it and insert them in S3 bucket.

Security

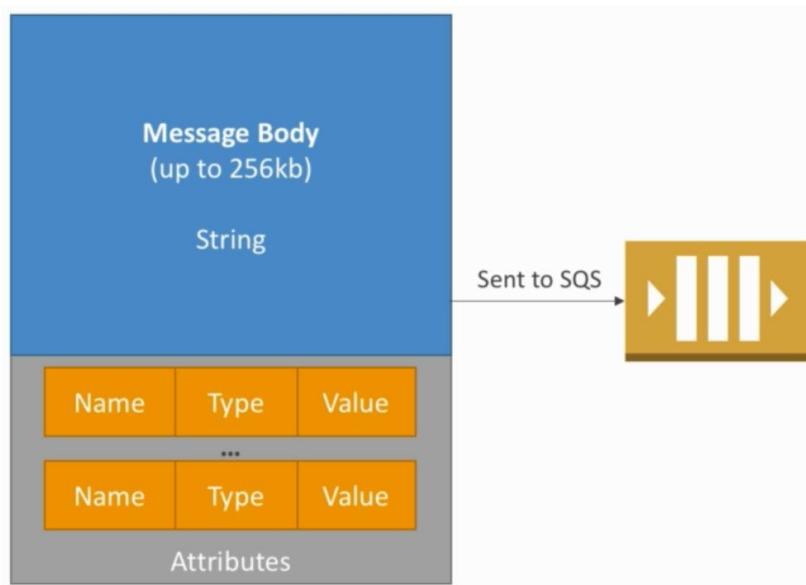
- Encryption
 - In flight encryption using HTTPS API
 - At-rest encryption using KMS keys.
 - Client side encryption.
- Access controls: IAM policies to regulate access to the SQS API.
- SQS Access Policies
 - Useful for cross account to access SQS queues.
 - Useful for allowing other services (SNS, S3) to write to SQS queue.

Delay Queue



- Delay message (consumer's don't see it immediately) up to 15 minutes. The message remains in SQS for n minutes.
- Default is 0 seconds.
- Can set a default at queue level.
- Can override the default using DelaySeconds parameter.

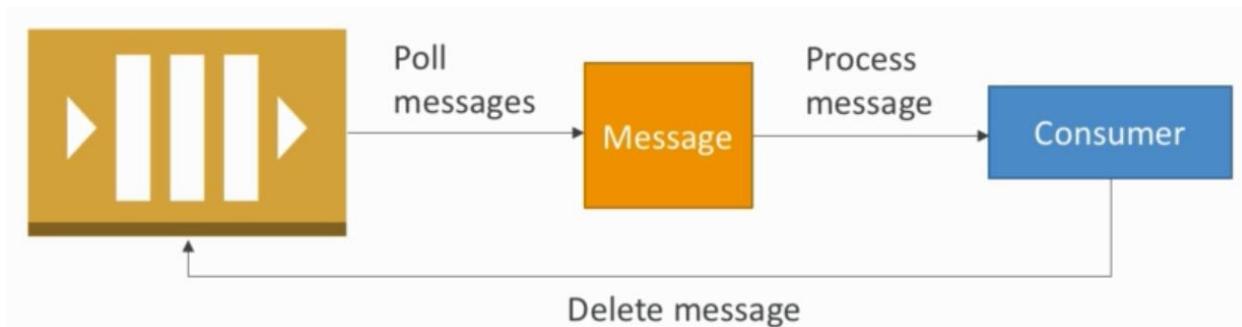
Producing messages



- We need to define the body, and the body is 256KB & has to be a string.
- Message attributes can be added or metadata –optional.
- Message attribute has a key value pair, which has a name, type & value.

- We can also provide a delay delivery – optional.
- Now, the message is sent to SQS.
- We get back a
 - Message identifier from SQS (an ID).
 - MD5 hash of the body (used for secure cryptography).

Consuming messages



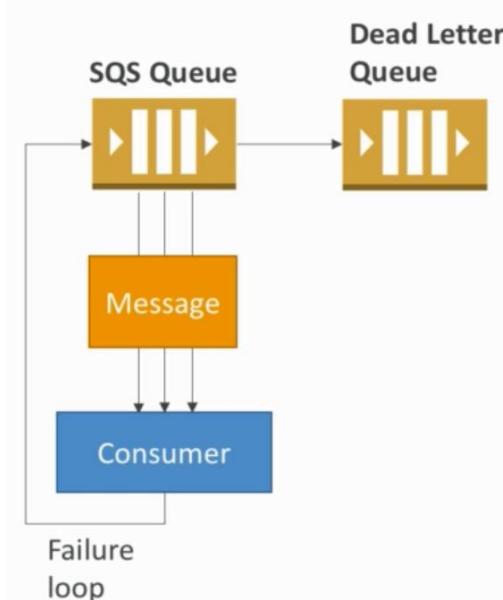
- Consumers don't get the messages pushed straight to them.
- They need to poll SQS for messaging (can receive up to 10 message at a time).
- Consumers have a duty to process the message(s) that they receive within visibility timeout. Processing the message means for e.g. the consumer will insert the message into a shipping DB & the message will live its life there.
- After processing the message, they can delete the message using message ID that was received, & something called the receipt handle.

Visibility options

- When a consumer polls a message from a queue, the message is invisible to other consumers for a defined period which is known as Visibility Timeout.
 - Visibility timeout can be set between 0sec to 12hours (default 30sec).
 - If the visibility is too high (15 minutes) & consumer fails to process the message.
 - If too low (30sec) & consumer needs time to process the message (2minutes), another consumer will receive the message & message will be processed more than once.

- **ChangeMessageVisibility API** helps to set the time to process a message.
- **DeleteMessage API** tells SQS the message was successfully processed.

Dead Letter Queue



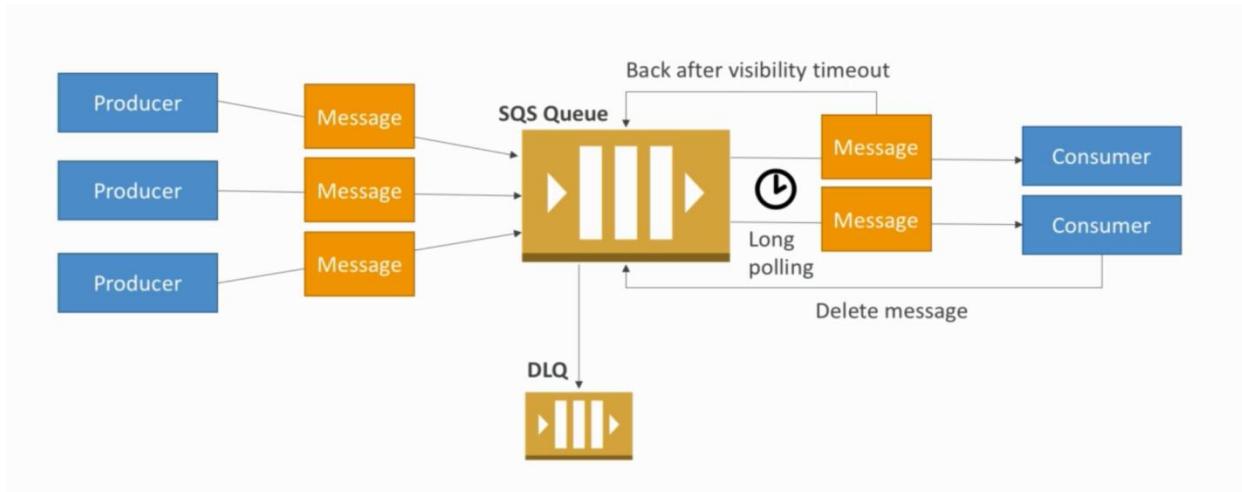
- If the consumer fails to process within the Visibility Timeout, the message goes back to the queue & the other consumers can get it.
- We can set a threshold of how many times a message can go back to the queue – it is called a “redrive policy”
- After the threshold is exceeded, the message goes into a Dead letter queue (DLQ).
- We have to create a DLQ first assign it DLQ.
- Make sure to process the message in the DLQ before they expire.

Long Polling

- When a consumer request message from a queue, it can optionally wait for message if there are none in the queue like it can wait & say I'm willing to wait 20sec until we have one. This is known as Long Polling.
- Long Polling decreases the number of API calls made to SQS while increasing the efficiency & latency of application.

- The wait time can be configured between 1 to 20sec & 20 second is preferable.
- Long Polling is preferable to short polling.

SQS message consumption flow diagram



LAB-I Console Hands On

1. Select “Standard Queue”.
2. Give Queue a name.
3. Hit on “Configure Queue”
4. Create the Queue

Note:

- We can send, view/delete message.
- We can even reconfigure the Queue (its parameter).
- Purge queue, it deleted the messages in the queue completely.
- Delete queue helps to delete the queue altogether.
- “Messages available” helps to know how many messages are available to be processed.
- “Messages in flight” tells how many messages are pulled by consumer but not deleted yet.

5. Click on “Queue actions” & hit on “Send message”.

6. Type a sample message & even we can delay the message up to 15 minutes.
7. Click on “Message attribute” tab & AWS SQS allows us to add metadata like timestamp, geospatial data, signatures & identifiers) as each message can have 10 attributes.
8. Click on “Send message”. We get Message identifier & MD5 of the body
9. We can see that 1 message is available in the queue.
10. Right click on the column & select “View/Delete Messages”.
11. We get a few warnings like:
 - a. Message will come from the front of the queue unless other application are reading from the queue.
 - b. Message displayed in the console will not be available to other applications until the console stops polling for message.
 - c. The console drops polling for message as the specified number of seconds has elapsed, the requested number of message have been received or the “Stop Now” Button has been pressed.
 - d. Deleting a message from queue permanently removes it from the queue.
12. Click on “Start Polling for Messages”.
13. We can see our first message that we typed in step 6. We can see its size, time log when it was sent, and how many time it was sent. If we fail to receive the message, and receive it again, the process count will be two.
14. Click on “Stop now” and hit on “Start polling for Messages”, then we can see the message count is 2.
15. Clicking on “More details” allows us to see the Message ID, Size, MD5 hash ID, the sent & receive timings, number of times the message has been received and number of attributes.
16. Once we delete the message from the SQS queue, it will never be seen back.
17. After deleting the message, if we go back to the console we can see that there's everything default.

LAB-II Dead Letter Queue

1. Go to SQS under Services.
2. Click on Create Queue

3. Give the queue a name.
4. Configure your settings, and hit on Create.

We need to set the re-drive policy on our First Queue to talk & send message to our first queue.

5. Go to console, and click on the first queue we created, select “Redrive policy” tab. We can see, nothing under the tab.
6. Click on your queue and hit on “Queue Actions” and select configure queue.
7. Check on the “use Redrive policy” checkbox.
8. Give the name of the existing queue that’ll serve as a Dead letter Queue.
9. Set the number to maximum number of times a message can be received before it is send to Dead Letter Queue.
10. Hit on “Save Changes”.
11. Now send the message from the first queue & close.
12. Now consume the message by clicking on “View/Delete Message” & click on “Start Polling for Messages”.
13. Receive the message once by repeating the step13.
14. After 2 times, when we start polling the message does not appear in the “view/Delete Message” of our first queue, click on the Dead Letter Queue & hit on “View/Delete Messages”, we can see the messages in the queue.

SQS using CLI

1. To get the list of the queues type: `aws sqs list-queues`. We may not receive the list sometimes if the CLI is configured in some other region. In this case, we can give the command `aws sqs list-queues --region us-east-1`
2. To send message, we need to specify the queue URL & the message body. We can get the URL using our first command. Again, if you've a different region configured, don't forget to add the region. Note: Don't give a space in between the text of the message.
3. `$ aws sqs send-message --queue-url https://ap-south-1.queue.amazonaws.com/895094500025/myFirstQ --region ap-south-1 --message-body Technoroniicks-here`

4. Or better use inverted commas for white spaces: \$ aws sqs send-message --queue-url https://ap-south-1.queue.amazonaws.com/895094500025/myFirstQ --region ap-south-1 --message-body "Technoronicks here"
5. Go to console, & view the message under the specified queue in the command.
6. To receive the messages, we use receive message command, the parameter under WaitTimeSeconds enables long-poll support which is an important functionality for decreasing the cost. We'll add some for configuration like visibility time-out, max number of messages, wait-time seconds. \$ awssqs receive-message --region ap-south-1 --queue-url https://ap-south-1.queue.amazonaws.com/895094500025/myFirstQ --max-number-of-messages 10 --visibility-timeout 30 --wait-time-seconds 20. In response to the command, we get a MessageID, ReceiptHandle which will be used at the time of delete message API, the MD5 of the body & the body itself.
7. If we check the console, the last message will be in “Messages in Flight” column and after 30 seconds, it’ll be in “Messages available” section.
To delete the messages, we need a receipt handle, the queue URL and region: \$ aws sqs delete-message --receipt-handle AQEBoJJ2f2Bm4CFMpW0bBoms722bNsVaCz5CN93/FFH3d6br09suyxhGWgpVTagZ+h3Fjv86+XAKyIKx4W62m+MyoiXBV7liRrt/rjz2hyGzkgXKGQJIf e3xY6tgftokoxJshryHzGFJqtOtQ/EvD+EZyTB6umjn9vE3z2XKrN3M267pFWaSkGkKP7WPw9XdyK9dhfeZqls7eM816vncEBS9ayxoezuy6EYLE1RNDyM1LMpL9OWuistcIpwl1f/wB3HMjEdJ3iRiMAjFbtj4N8picdhKg/+jm0fdPxml8ya6bE3xoCLdtLaCK15uUyNPC+kS5lPEFKDsxEFI81ReheAgKor/+Bu9TQjJeR1kD8YpluM2mWFc8Alh4nDv+ofQ6RCQV --queue-url https://ap-south-1.queue.amazonaws.com/895094500025/myFirstQ --region ap-south-1
8. Refresh the console, now we can see the value of “Message Available” column has been decreased by one.

FIFO Queue

- It is a new offering so available in few of the region.
- Name of the queue must end with .fifo
- The throughput is lower, if we enable batching throughput reaches to 3000 per second & if it not enabled, it is 300 per second. If we decrease the throughput, it guarantees the order with FIFO queues.
- Messages are processed in the same order by the consumer.

- Message are sent exactly once.
- No per message delay (only per queue delay) because, if we configure it as a per message delay, the FIFO guarantee would be wrong.
- We can have as many consumers as GroupID for your FIFO queues

Features

- Deduplication (not to send the same message twice). If we send the same message twice with the same body, then the second message will be deactivated.
 - When we send a message to FIFO, we need to provide a field called MessageDeduplicationId which should be unique & if we send the message twice, then it'll de-duplicate on it.
 - Working: There's a de-duplication which is 5 minutes & if the SQS sees the same message within that interval of 5 minute with the same message deduplication ID, then it'll de-duplicate & discard the other message.
 - Content based deduplication: If we just want to de-duplicate on payload & if we're not having an ID in mind then content based deduplication is quite handy. The MessageDeduplicationId is generated as the SHA-256 of the message body, hashing algorithm of the message body (not the attributes).
- Sequencing:
 - To ensure strict ordering between messages, we need to specify a MessageGroupId
 - Messages having different group ID may be received out of order.
 - E.g.: To order message from a user we could use the "user_id" as a group id.
 - Messages with same Group ID are will be delivered to the same consumer and will be processed with strict ordering.

LAB

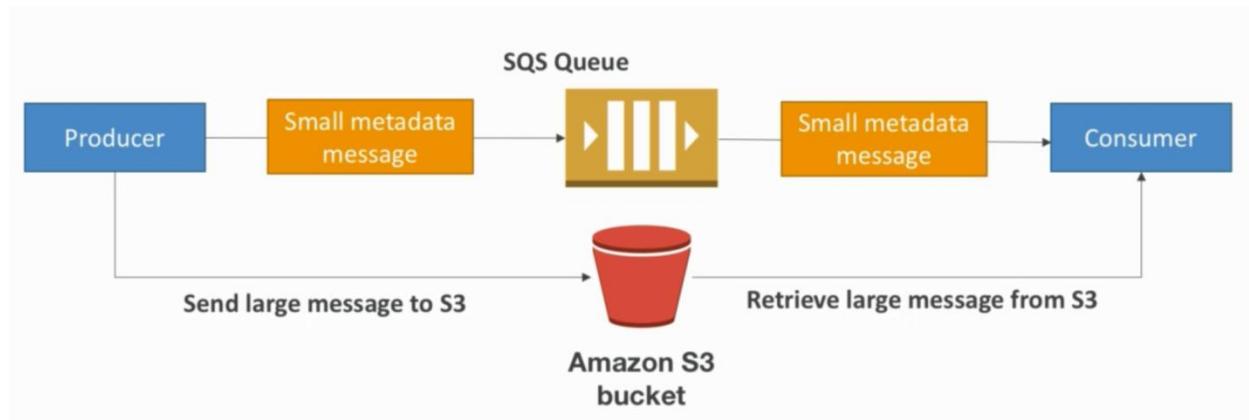
1. Go the SQS under Services.
2. Click on "Get started".
3. Give the queue a name, and it must end with a .fifo extension.

4. Right click on it & let's send a message. In the bottom, we need to specify the message ID & deduplication ID
5. If we send two different messages with same deduplication ID, we can see the message count under "Messages Available" is one.
6. If we send same message with different deduplication ID, then we will be able to see the message count changing under "Messages Available" list.
7. We can configure Content Based Deduplication, it allows us to use SHA 256 of the body message to generate the content based deduplication ID.
8. Right click on the queue & click on "Send Message".
9. Type a message & define the group ID. The Message deduplication ID is optional as it'll be auto-generated.
10. Click on "Send Message", the message gets accepted.
11. If we try to send the same message again following the steps 7-10, then it'll not be accepted as the queue is content-based.

SQS Advanced

Extended Client

- We know that size of the message is 256KB, but how to send larger messages.
- It uses SQS extended client (a Java library)
- It leverages S3 on the top of our SQS.



- If the Producer wants to send large message, then instead of sending it to SQS, it sends to S3
- In the S3 bucket, it knows where the message is put in S3. So, it sends small metadata to the SQS queue indicating a larger message was sent to S3.

- Using extended library, the consumer understands it need to pull a larger message out of S3.
- Hence, it goes to S3 & retrieves the large message.

Security

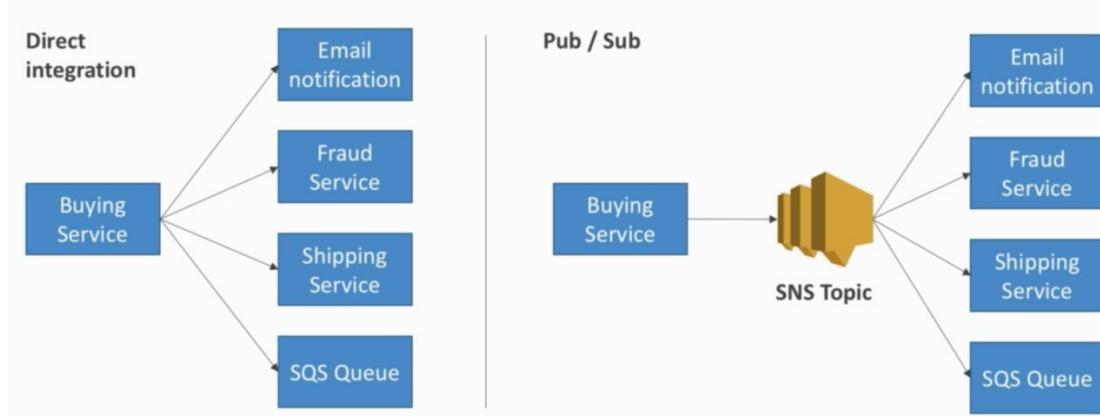
- Encryption in flight is done using HTTPS endpoint.
- Can enable SSE (Server side encryption) using KMS.
 - Can set CMK (Customer master Key) we want to use.
 - Can set the data key reuse period (between 1 min & 24hr). The lower we keep the reuse value, more the API will be used & hence we need to pay more. The default value is 5 minutes.
 - SSE will only encrypt the message ID, not the metadata, timestamp or attributes. So it is advised not to store sensitive information in the body.
- IAM policy must allow the usage of SQS.
- SQS queue access policy
 - Finer grained control over IP.
 - Control over the time the request comes in.
- Since there are no VPC endpoint, we must have internet access to access SQS. So, we can't access SQS privately within our VPC for now.

API's

- CreateQueue, DeleteQueue,
- PurgeQueue: To delete all messages in the queue (takes 60 sec).
- SendMessage, DeleteMessage, ReceiveMessage
- ChangeMesaageVisibility: To change the timeout.
- Batch API's for SendMessage, DeleteMessage&ChangeMessageVisibility which helps in decreasing the costs.
- Note: ReceiveMessage can receive up to 10 message only, so there's no Batch ReceiveMessage.

SNS

- If we want to send one message to many receivers.
- We've the possibility of direct integration.



- Let's take an example of Buying Service and needs to send email, talk to their fraud service, talk to their shipping service may be talking to another SNS queue.
- If we want to integrate all the service together, it'd be quite painful. The other approach to sing Pub/Sub.
- In this approach, the buying service publishes data to the SNS topic & SNS topic has many subscribers.
- These subscribers get data in real-time as a notification to multiple services.
- The “event producers” only sends messages to one SNS topic.
- We can have as many event receivers as we want to listen to the SNS topic notifications.
- Each subscriber to the topic will get all the messages (Note: a new feature to filter all the messages).
- We can have up to 10,000,000 subscription per topic.
- We've 100,000 topics limit.
- Subscribers can be:
 - SQS
 - HTTP/HTTPS (with delivery retries – how many times)
 - Lambda
 - Emails
 - SMS messages
 - Mobile Notifications

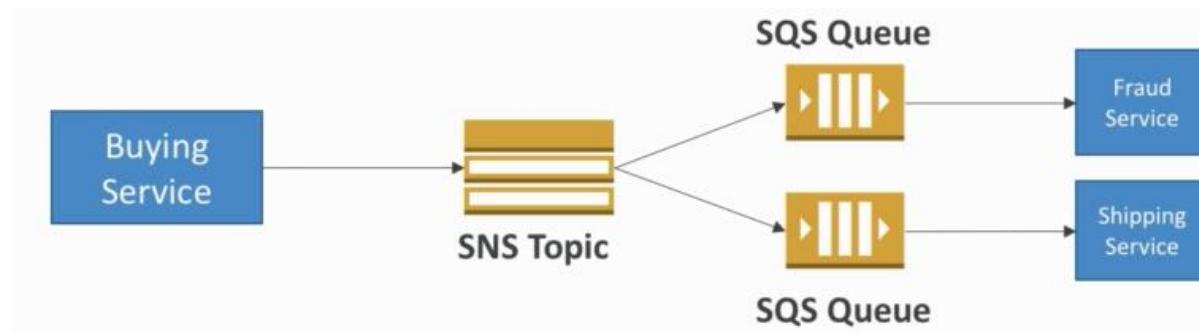
SNS Integration

- Services can send data directly to SNS for notifications.
- CloudWatch (for Alarms).
- Auto Scaling group notifications
- Amazon S3 (on bucket events)
- CloudFormation (upon state changes → failed to build, etc.)
- CloudWatch Event rules.

How to publish

- Topic Publish (within 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, and other apps receive notification.
 - Works with Google GCM, Apple APNS, Amazon ADM.

SNS + SQS: Fan Out



- If we want to publish data to many SQS queues, then we need to push it once in SNS & it will be received in many SQS.
- Fully decoupled.
- No data loss (if the message is not being processed, then the data is not lost as in the case of SNS).

- Ability to add receivers of data layer.
- SQS allows for delayed processing & retries of work.
- May have many workers on one queue & one worker on another queue.

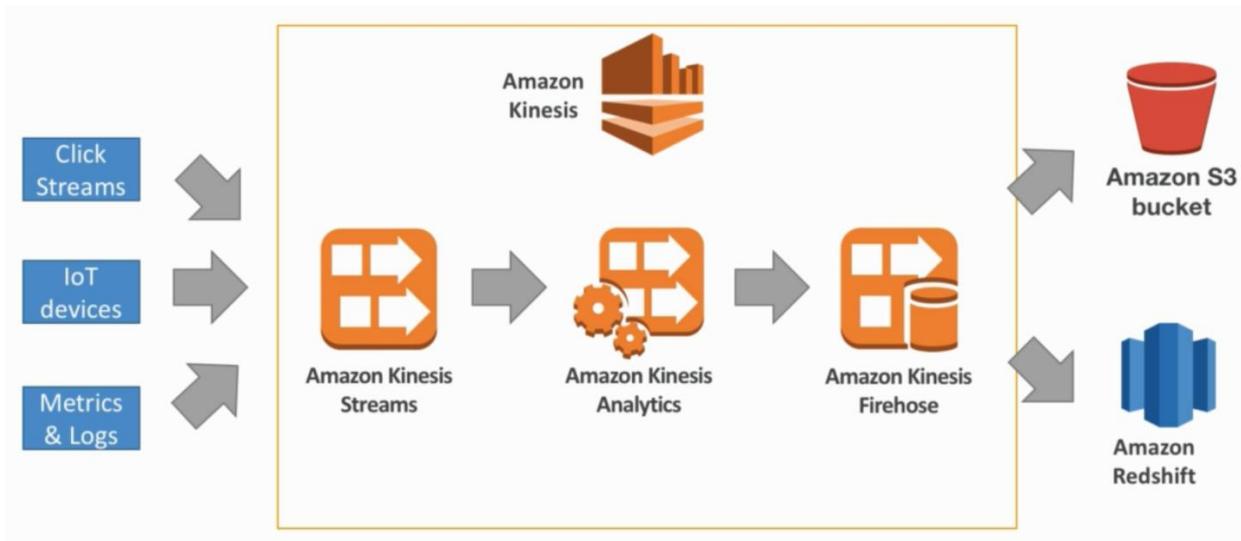
LAB

1. Go to SNS under Services.
2. Give the topic a name.
3. Hit on “Create topic”.
4. To listen to the message on the topic, create a subscription.
5. Select the type of protocol as Email for now. (Protocols -> HTTP/ HTTPS/ Lambda / Email /Email-JSON / Amazon SQS/ Platform Application endpoint/ SMS).
6. Write the Email ID of your choice.
7. Click on “Create Subscription”.
8. Confirm the email from the inbox.
9. Go back to the topic created in step3. Click on Publish message.
10. Write down the Subject & Message body.
11. Keep the default settings & hit on “Publish message”.
12. Check the inbox!

Kinesis

- Kinesis is an alternative to Apache Kafka.
- It is a big-data streaming tool. Allows us to collect application logs, metrics, IoT, clickstreams.
- Great for streaming process frameworks (Spark, NiFi, etc.)
- Data is automatically replicated to 3 AZ.
- The 3 Kinesis products are:
 - **Kinesis Streams:** Low latency streaming ingest at scale.
 - **Kinesis Analytics:** Perform real-time analytics on streams using SQL.
 - **Kinesis Firehose:** Load streams into S3, RedShift, ElasticSearch, etc.

Working



- The Click Streams, IoT devices, metrics & Logs will be producing data into Kinesis streams.
- After having the data, kinesis wants us to process the data & perform fraud computations, metrics, monitoring, alerting, or whatever we want. For this we need to perform some computation in real time. Here it is done by “Amazon Kinesis Analytics”.
- After the computations are done, it is good to store them somewhere like S3, database, RedShift, etc. Here, Amazon Kinesis Firehose comes to play.
- Overall, we can say that kinesis is in the middle for the streaming real-time data & it allows us to quickly onboard data in mass in real-time from our big-data use-case all the way to the analytics to the final store.

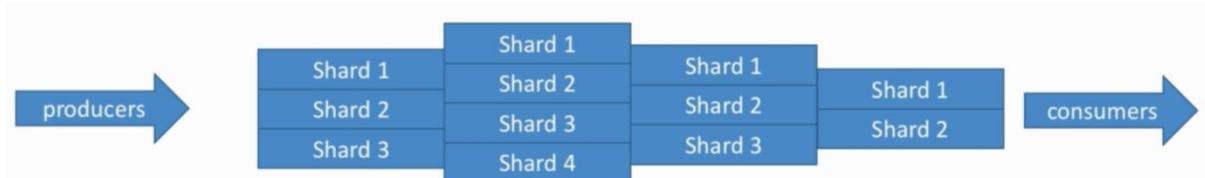
Kinesis Stream Overview

- Streams are divided into ordered Shards/ Partitions. Shard is a uniquely identified sequence of data records in a stream.
- We've producers & it is going to produce to Kinesis streams.
- We can have any number of shards & the data can go to either of the shard.
- The consumer can consume from either of the shard.
- When we want to scale up our streams, we just add shards. Or if we want to increase the number of throughput, we need to increase the number of shard.

- Data is for a period of 1 to 7 days in the shard.
- Default value is 7 for the data to be stored in the shard.
- The retention period is so less because the Kinesis is a massive platform, so we need to process the data, do something & keep somewhere else ASAP.
- Kinesis has the ability to reprocess/ replay the data. So, it has advantage over SQS as data once processed in SQS, it's gone.
- Multiple application can consume the same stream like SNS.
- It provides us a feature of real-time processing with real scale of throughput as we can add shards.
- Since Kinesis is not a database, we can delete the data once it is inserted (immutability).

Shards

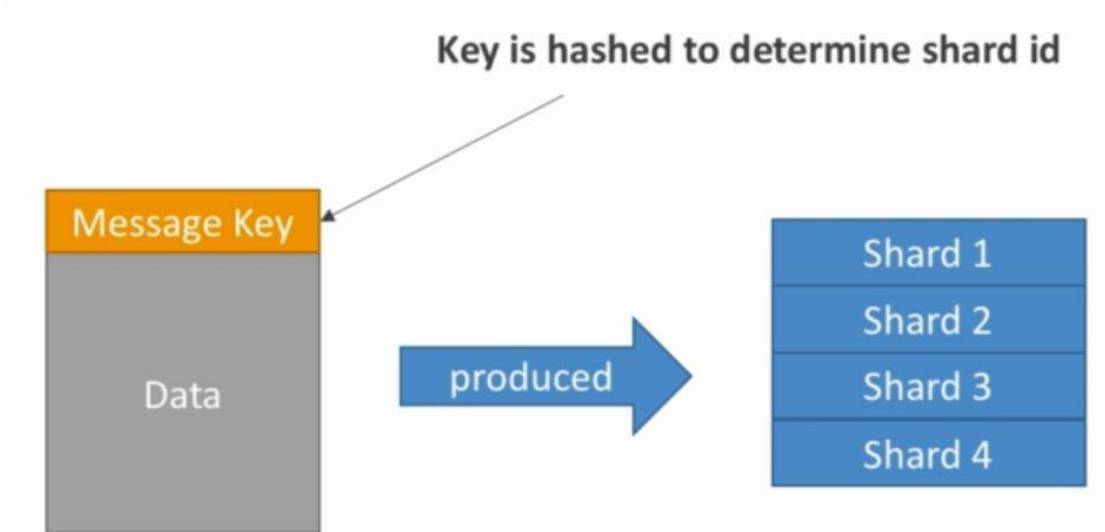
- One stream is made up of many shards.
- Shard represents 1MB/s or 1000 messages per second at the write side.
- We've 2MB/s at read per SHARD.
- Billing is per Shard provisioned, can have as many shards as we want.
- If we've more throughput than the Shards, we are going to have throughput issues.
- Batching feature is available.
- The number of shards can evolve over time (re-shard /merge).
- Records are ordered per Shard.



- For increasing the shards, and merging the shards with a producer, it will have three shards. Maybe someday, we get a higher throughput so we add a shard. This is called re-sharding. And someday we get less throughput, we decrease the number of shard. It is known as merging.

Kinesis API – Put records

- It is a way to send data to the Kinesis. For this, we need to send data & a partition key.



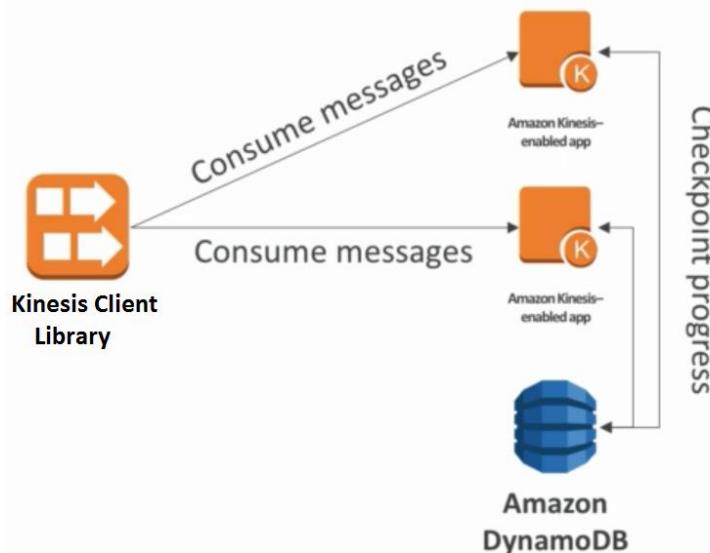
- The data is the grey box & message key is the orange box. Message key is in string, & this key will get hashed to determine the shard ID. So, key is a way to route the data to a specific shard.
- The same key goes to the same partition (helps with ordering for a specific key). So, if we want to all the data for the same key then we need to provide the key to every data point and they'll be in order for us.
- When the data is produced, the messages knows where to go, which shard because of the message key. So our data will be produced to either of the shard(s), but not all of them at the same time.
- The message when sent to a shard get a sequence number & that sequence number is always increasing.
- If we need to choose a partition key, we need to choose the one which is highly distributed. We need to know the partition key must be very distributed, it prevents the hot partition. If the key is not distributed then all the data will go to the same shard & one shard will be overwhelmed.
 - If we've an application with 1 million users, user ID is a great key.
 - If we've country ID as a field, & it turns out that 90% of users are from same country then all our country ID will go to one Shard & that

shard will be overwhelmed. So, we need to choose that is going to be well spread & bottom range of data.

- We can use batching with PutRecords to reduce costs and increase throughput.
- If we get an exception ProvisionedThroughputExceeded, when we go over the limit, it makes our Shard hot.
 - The exceptions happens when we get more data (exceeding MB/s or Transaction per Second for a shard).
 - The solution is:
 - Retries with backoff
 - Increasing Shards.
 - Ensuring that partition key is a good one.
- To produce messages, we'll use CLI, AWS SDK, or producer libraries from various frameworks.

Kinesis API – Consumers

- We can use normal consumers using the CLI, AWS SDK, etc.



- We can also use Kinesis Client Library (in Java, Node, Python, Ruby, .Net)
 - KCL uses DynamoDB to checkpoint offsets).
 - KCL use DynamoDB to track other workers & share the work among shards.

- In the above image we see that KCL is trying to share the work amongst the Shards using Amazon DynamoDB.

LAB

Note: It is a paid service (does not come under Free Tier).

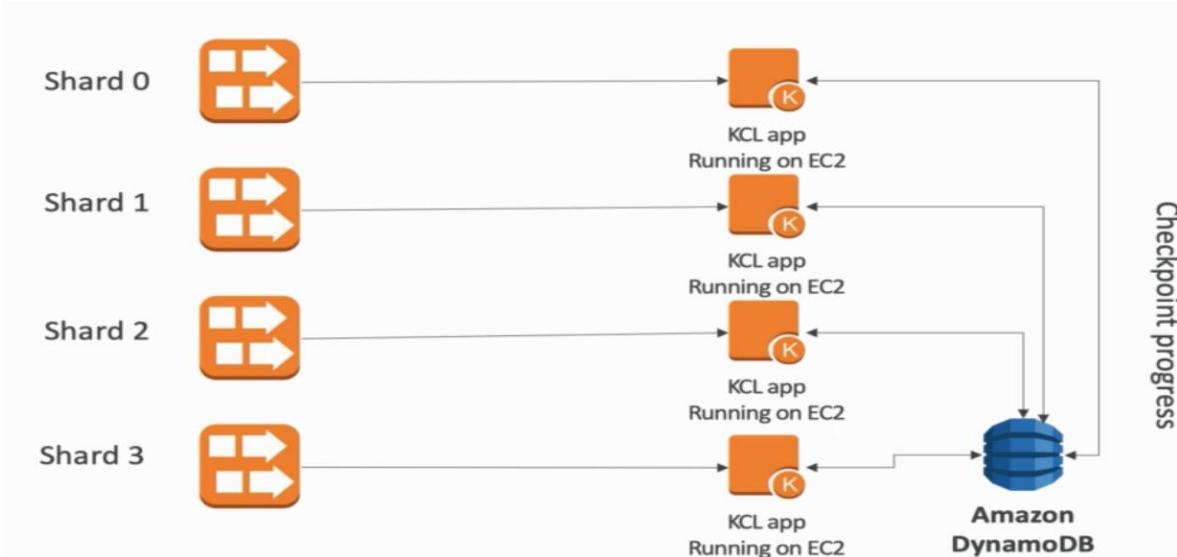
- Click on Kinesis under Services.
 - Hit on “Get started”.
 - There’re different use cases:
 - Kinesis data streams
 - Kinesis Data Firehose
 - Kinesis data Analytics
 - Kinesis Video streams
 - Let us start with Creating a data stream
1. Give your Kinesis stream a name.
 2. There’s even a Shard calculator to compute the number of Shard required.
The maximum number that can be provisioned is 200.
 3. Hit on “Create Kinesis Stream”.
 4. We can even enable Server-side encryption.
 5. Data retention period can be increased from 24 hours to 168 hours (7 days).
 6. Shard-level metrics features for identifying the distribution of the data throughput like incomingBytes, incomingRecords, outgoingBytes, OutGoingRecords, etc.
 7. Monitoring allows us to view the metrics graphically. The red line in the graph indicates the limit.
 8. Go to CLI:
 - a. To list the streams: `aws kinesis list-streams`
 - b. To get information on the stream: `aws kinesis describe-stream --stream-name stream_name`. Note: `stream_name` can be known from our first command. It returns the information about shard, stream details, enhanced monitoring, encryption, etc.
 - c. To send data to shard: `aws kinesis put-record --stream-name stream_name --data "user's message" --partition-key user123`. Partition key determines which

shard in the stream, the data record is assigned to. It returns the shard name & sequence number.

- d. To retrieve the records, firstly we need the shard iterator & later then get-records. To get iterator: aws kinesis get-shard-iterator get-shard-iterator --stream-name value --shard-id value --shard-iterator-type value. Shard ID can be retrieved from 2nd command. shard iterator types is used to start reading data records from the shards, it's types are:
 - i. AT_SEQUENCE_NUMBER
 - ii. AFTER_SEQUENCE_NUMBER
 - iii. AT_TIMESTAMP
 - iv. TRIM_HORIZON: starts reading the last untrimmed records in the shard.
 - v. LATEST: helps start reading the most recent record in the shard.
 - e. To retrieve the records: aws kinesis get-records --shard-iterator value. In response, we get back all the records with sequence number, approximate arrival timestamp, Data & partition key. The data received is base64 encoded data.
9. Hence we need to decode. In the browser, type in base64 decoder & type in the encoded message.
10. Delete the stream.

KCL

- Kinesis Client library is a Java library that help read records from Kinesis Stream with distributed applications sharing the read workload between the different instances of our application.



- Rule: each shard is to be read by only one KCL instance which means if we've an application containing 4 shard, then we'll have maximum 4 KCL instances.
- Progress of the KCL application is check pointed, means how far it's reading into Kinesis is check pointed from time to time in DynamoDB. For this we need access to IAM to write to DynamoDB.
- KCL can run on EC2, Elastic BeanStalk, on Premise Application.
- Records are read in order at the shard level. Each shard is read in order but across shard, the order is not guaranteed.
- E.g.: We've 4 shard in our Kinesis stream & we've 2 application instances which are running on 2 EC2 instances & 2 KCL instances. The first two shard is going to read by 1st shard while the other two are going to be read by the 2nd shard. To know this, they're going to check their progress into DynamoDB & from their make assumptions how they'd share the work.

Kinesis Security

- We get IAM policies to control access & authorization to Kinesis.
- We get encryption in flight using HTTPS endpoint.
- Encryption at rest using KMS (at server side).
- Harder possibility to encrypt/decrypt data client side (harder).

Kinesis Data Analytics

- Perform real-time analytics on Kinesis Stream using SQL.
- Kinesis data analytics is:
 - Auto-scaling.
 - Managed: we don't need to provision any servers.
 - Continuous: It is real time.
- Pay for actual consumption rate.
- Can create real stream out of real-time queries.

Firehose

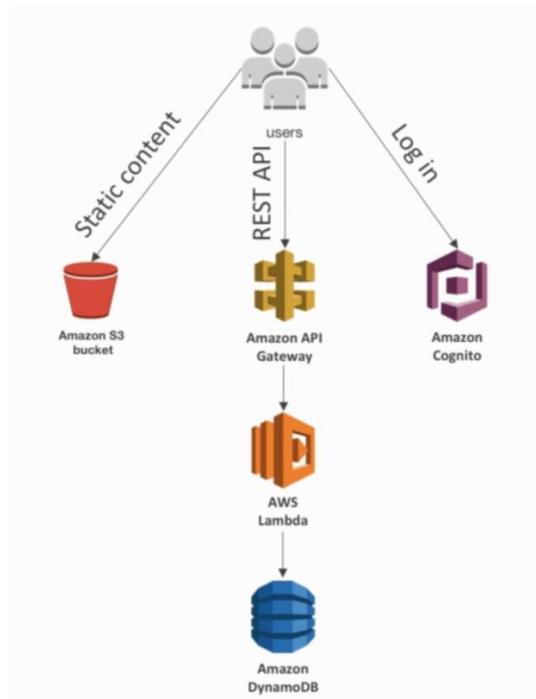
- It is a fully managed service without administration.
- Latency is near time (60 seconds latency).
- Load data in RedShift/ Splunk/ S3/ ElasticSearch
- Automatic Scaling.
- Many data formats supported (but we need to pay for conversion).
- Pay for amount of data going through Firehose.

Difference between SQS, SNS & Kinesis

SQS:	SNS:	Kinesis:
<ul style="list-style-type: none">• Consumer "pull data"• Data is deleted after being consumed• Can have as many workers (consumers) as we want• No need to provision throughput• No ordering guarantee (except FIFO queues)• Individual message delay capability	<ul style="list-style-type: none">• Push data to many subscribers• Up to 10,000,000 subscribers• Data is not persisted (lost if not delivered)• Pub/Sub• Up to 100,000 topics• No need to provision throughput• Integrates with SQS for fan-out architecture pattern	<ul style="list-style-type: none">• Consumers "pull data"• As many consumers as we want• Possibility to replay data• Meant for real-time big data, analytics and ETL• Ordering at the shard level• Data expires after X days• Must provision throughput

16. AWS Lambda

- Serverless is a new paradigm in which developers don't need to manage servers, they just deploy code or specifically, functions! It does not mean there are no servers.
- Initially, serverless meant to be FAAS (Function as a Service).
- Serverless was pioneered by AWS Lambda, but now includes anything that's managed, Databases, Messaging, Storage, etc.
- It is a happy thing for developers, because the less we've to manage the better we sleep at night.
- Serverless in AWS has many different forms:
 - AWS Lambda & Step functions.
 - DynamoDB
 - AWS Cognito
 - AWS API gateway
 - Amazon S3
 - AWS SNS & SQS
 - AWS Kinesis
 - Aurora Serverless
 - IoT
-



In terms of architecture, say we've a website & we've static content we need to serve such as HTML files, CSS files, etc. Our users get it straight from a S3 bucket or a CloudFront in between. Then our user would may use a log in, email or password log in using Amazon Cognito. Then they use Rest API, so they communicate to our server using a REST API managed by API gateway. In the backend, our API Gateway might talk to AWS Lambda function. And, our Lambda function might talk to DynamoDB to store state & have a database that scales for us, and that is serverless.

Overview

Why Lambda?

- In EC2,
 - There're servers in the cloud.
 - These are limited by RAM & CPU
 - It is continuously running.
 - Scaling them requires intervention to add /remove servers.
- With AWS Lambda,
 - There're virtual functions, so we don't need to manage.
 - They're short executions one.
 - They run on demand. It means when we are not running our functions, we are not going to be billed.
 - It scales within milliseconds if we need more Lambda functions.
 - Scaling is automated.

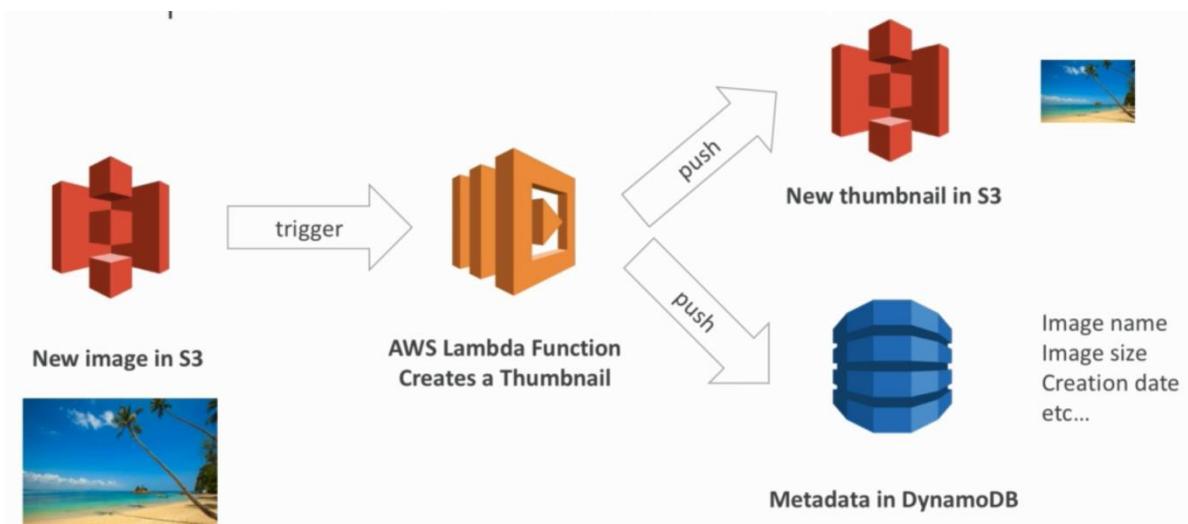
Benefits

- Easy pricing
 - Pay per request & compute time (if our function is called once a month, then we're just going to pay for once a month invocation while if we had EC2 server, we'd have to pay for the whole month).
 - We've a free tier of 1,000,000 AWS requests & 400,000 GBs of compute time.
- Integrated with the whole AWS stack.
- Can be integrated with many programming language.
 - Python

- Java
- Node.js
- C#
- Golang
- Powershell
- Docker is not for Lambda, it's for ECS/Fargate.
- Easy to monitor as it can be integrated with AWS CloudWatch.
- Easy to get more resources per function (like if we've a small function so we can provision just 128 MB of RAM & if we need function for some heavy lifting then it is going to need up to 3GB of RAM).
- The more RAM we get, the more CPU & network we get for our function as well.

Example

1. Serverless Thumbnail creation



When a user uploads an image, we want to generate a smaller version of that image called thumbnail to be more efficient & better functioning website.

The user uploads image & land in S3. Since Lambda is integrated with S3, our Lambda function will be triggered & role of that Lambda function will be to run, perform stuffs & create thumbnail. The thumbnail may be pushed to the same S3 bucket or to a new one as the beach image is little bit smaller.

And maybe we want to push the metadata in DynamoDB such as image name, image size, creation date, etc. the main idea is that there're no servers to manage as all the services are serverless & it can scale infinitely.

2. Serverless CRON Job

CRON is a command to OS or server for a job that is to be executed at a specific time.

We can use CloudWatch events to emit events on the CRON schedule & trigger every one hour & AWS Lambda function will perform a task of our choice like a script, DB cleanup task. Usually, CRON Jobs are run on EC2 machine running Linux. But here, CRON Jobs are triggered by CloudWatch Events as is performed by AWS Lambda function.

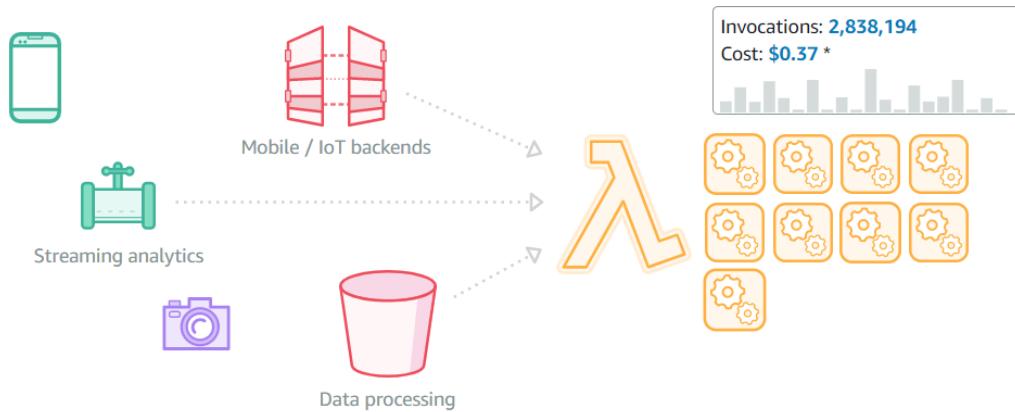
Pricing

- Pay per calls
 - First 1,000,000 requests are free.
 - \$0.20 per 1 million requests thereafter (\$0.0000002 per requests).
- Pay per duration: (in increment of 100ms)
 - 40,000 GB-seconds of compute time per month if FREE (it means we get 400,000 seconds of execution if our function is using 1GB RAM).
 - Or say, we're getting less RAM like 128MB then we get 3,200,000 million seconds of execution.
 - After that we pay \$1 for 600,000 GB-second.
- It is very cheaper to run Lambda, so it's popular.
- There's no charge when our code is not running.

LAB

1. Go to Lambda under Services.
2. Replace the 'function' from the URL with 'begins' (e.g.:<https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/functions> to <https://ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/begins>)
3. Click on 'Run' under "How it works" section.
4. Now click on "Next: Lambda responds to events".

5. If we click on any of the event (Streaming analytics, Mobile or Camera) there are more cog wheels on the right hand side which represents many Lambda functions are being invoked. The more we click on any of these events, the Lambda function scales more.
6. Click on “Next: Scale seamlessly”



7. We can see the cost emulator on based on invocations.
8. Click on “Create a function”.
9. For now, click on “Use a blueprint”
- 10.Under info section type “hello” & hit enter.
- 11.Click on “hello-world-python” & hit “Configure”.
- 12.Write a function-name of your choice.
- 13.Under “Execution Rule”, it tells us to define an IAM Role which allows us to execute stuffs for Lambda function like interacting with EC2, or S3, etc. so hit on “Create a new role with basic Lambda permissions”. It creates a role with permissions to upload logs to CloudWatch.
- 14.Hit on “Create function”.
- 15.We can see that the function has been created & we can test the event after hitting on the “Test” function.
- 16.For now, choose template to be “Hello World” & give your event a name.
- 17.Hit on “Create”.
- 18.Click again on “Test”.
- 19.It says that execution result succeeded. Click on details, we can see that Resource configured for this function is the minimum which is 128 MB, the max memory used was 47MB (it can vary).
- 20.Click on “Click here” to view the logs in the CloudWatch log group.

- 21.Let's modify the code, by commenting out the line11 which says to return event and uncomment line12 which raises an exception and hit 'Save'.
- 22.Click on 'Test' & now we can see that our execution result fails and we've the error message that something went wrong. See the log output in CloudWatch in CloudWatch logs.
- 23.Under IAM, we can see that a role has been created.
- 24.Click on that role and we can note that AWSLambdaBasicExecutionRole policy has been attached to it. This policy allows 2 things, firstly to create Log group for CloudWatch Logs and secondly, it allows to create a LogStream& put log event to the Log group it created.

Configuration parameters

- The default timeout or how long the function runs before failing, by default it is 3 seconds and maximum of 15 minutes.
- We can also send environment variables.
- We can allocate memory from 128MB RAM to 3GB.
- We're able to deploy within a VPC + assign security groups. For e.g.: To RDS instance deployed in the VPC by deploying a Lambda function in the same VPC.
- IAM execution role with correct permission must be attached to the Lambda function.
- We're able to add trigger to the left, so that it can trigger Lambda functions.
- We are able to edit the code, upload a .zip file or we can upload a file from S3.
- If we look in the code, Lambda function is basically a Lambda function which is the name of file and under Lambda_handler we write the part of the code which needs to be invoked/ triggered.
- Environment variables are key pairs that can be accessed from our code. These can be too be encrypted.
- We can even add tag to our Lambda function.
- Under basic settings,
 - We can choose CPU proportional to the memory configured.

- We can even configure the IAM Role attached to this Lambda function but it should be noted that the IAM role must have the permission to create logs to the CloudWatch.
- Timeout can be set which defines how much the function is expected to run before failing.
- Under VPC, we can select a custom VPC (for now, don't select any VPC).
 - If we don't select a VPC, we'll have a better performance to start a function.
 - If we select a VPC, it'll be a little bit slower.
 - Click on "Custom VPC".
 - Select a subnet under that & a security group.

25. Since our code ran within the limit of Timeout Duration. So, to check with default timeout of 3 seconds we'll add a code in python to sleep for 4 seconds. Add a library to add time library & sleep function ->

```
import time//time library
time.sleep(5) //to delay for 5 seconds
```

26. Save & test the function, we can see the execution fails giving an error message that "Task timed out after 3.00 seconds".

27. To print environment variable (for e.g.: key = environment_name & value = dev), Click on environment variables & manage.

28. Add a Key & a value.

29. Add a library to imports "os" to use operating system dependent functionality.

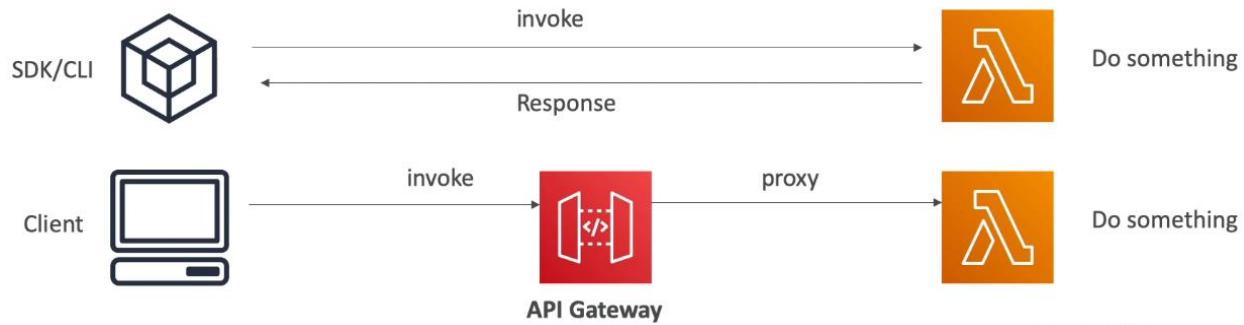
```
import os//os library
envName = os.getenv('environment_name') //to get the var.
print("environment name is: " + envName)// to print the value
```

Output: environment name is: dev

Synchronous Invocation

- We do synchronous invocation when we use the CLI, SDK, API Gateway, ALB.

- Results are returned right away.
- Error handling must happen on the Client side (retries, exponential backoff, etc.)

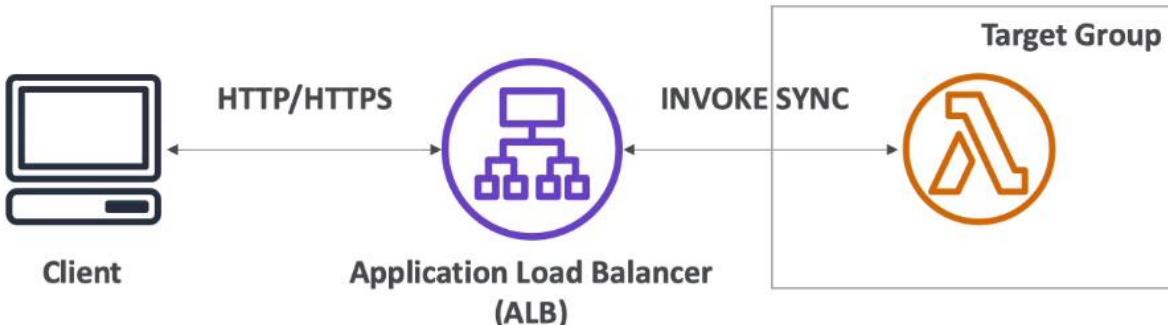


- The CLI/SDK invoke the Lambda function, and Lambda function does something & return some response.

Services

- User invoked
 - ELB
 - API Gateway
 - CloudFront
 - S3 batch
- Service Invoked
 - Cognito
 - Step Function
- Other Services
 - Lex
 - Alexa
 - Kinesis Data Firehose

Integration with ALB



- Lambda functions can be invoked with CLI/SDK.
- But if we want to expose them through the internet, we need to allow them to be used through the HTTP/HTTPS endpoint.
- We can do it by using ALB and we need to register it to a target group.
- The client invokes request in HTTP/HTTPS form to the ALB which will be invoking Lambda function synchronously in the target group. The Lambda function gets back to ALB which in turn returns a response.

Request Payload for Lambda Function

```
{
  "requestContext": {
    "elb": {
      "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:12345678901234567890:targetgroup/Lambda-Target-Group/49e9d65c45c6791a"
    }
  },
  "httpMethod": "GET",
  "path": "/lambda",
  "queryStringParameters": {
    "query": "1234ABCD"
  },
  "headers": {
    "connection": "keep-alive",
    "host": "lambda-alb-123578498.us-east-2.elb.amazonaws.com",
    "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.102 Safari/537.36",
    "x-amzn-trace-id": "Root=1-5c536348-3d683b0b04734faae651f476",
    "x-forwarded-for": "72.12.164.125",
    "x-forwarded-port": "80",
    "x-forwarded-proto": "http"
  },
  "body": "",
  "isBase64Encoded": false
}
```

Annotations pointing to specific fields in the JSON payload:

- ELB information: Points to the `targetGroupArn` field.
- HTTP Method & Path: Points to the `httpMethod` and `path` fields.
- Query String Parameters as Key/Value pairs: Points to the `queryStringParameters` field.
- Headers as Key/Value pairs: Points to the `headers` field.
- Body (for POST, PUT...) & isBase64Encoded: Points to the `body` and `isBase64Encoded` fields.

- So, from ALB to Lambda, the HTTP gets transformed to JSON document.
- There's info about which ELB got invoked ,the target group, the HTTP method (GET here). And hence, for this translation there's a translation of entire HTTPS request to JSON.

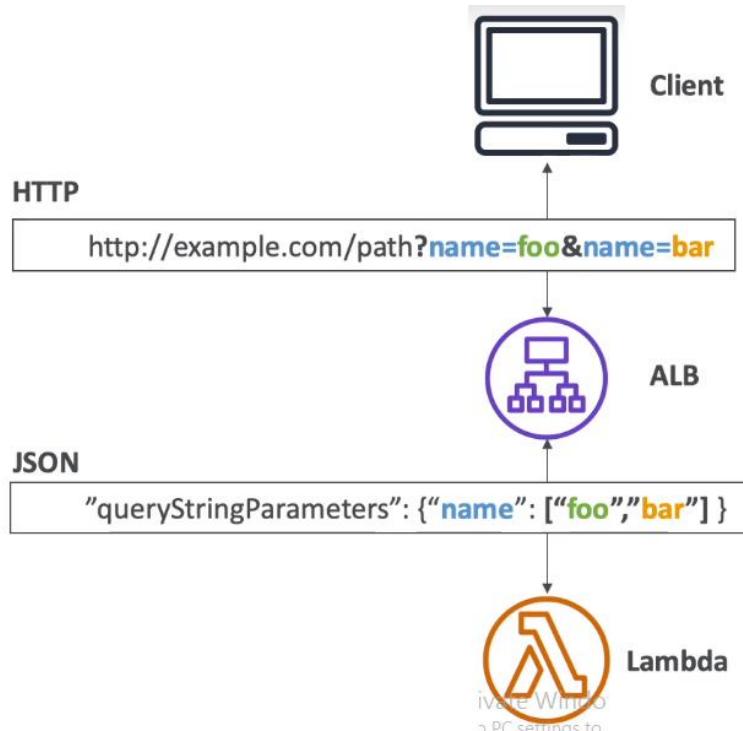
Response from the Lambda Function

```
{
  "statusCode": 200,
  "statusDescription": "200 OK",
  "headers": {
    "Content-Type": "text/html; charset=utf-8"
  },
  "body": "<h1>Hello world!</h1>",
  "isBase64Encoded": false
}
```

← Status Code & Description
← Headers as Key/Value pairs
← Body & isBase64Encoded

- In response, the Lambda function returns JSON document & ALB convert it back to HTTP.

ALB multi-header values

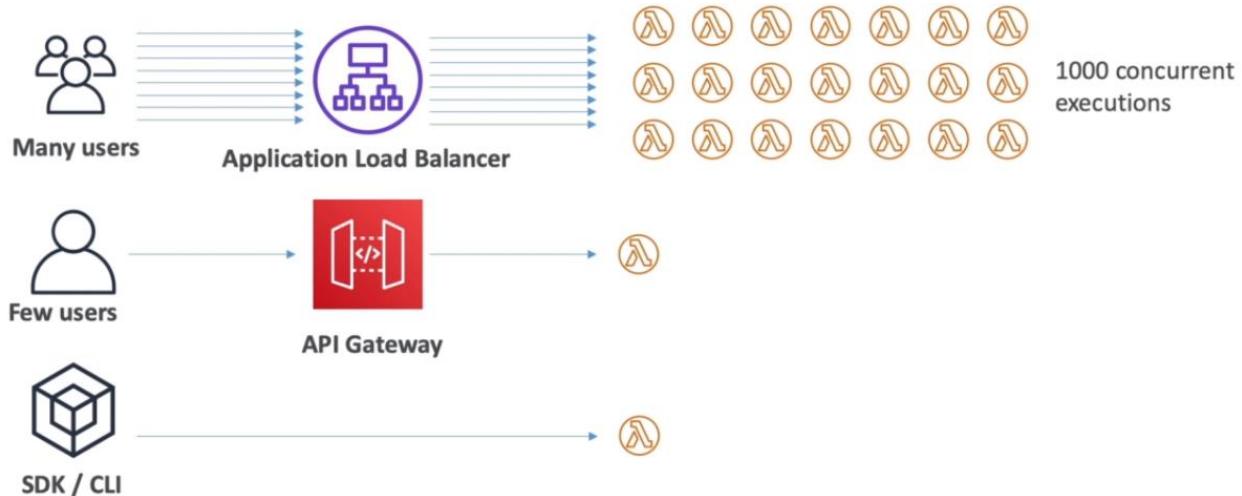


- ALB can support multi-header values and it can be set under ALB settings.
- So, if we pass multiple header/query strings with the same value, then the query string/header will be converted to an array into the Lambda function.
- So, when ALB invokes the Lambda function, the values will be converted to a JSON array.

Concurrency & Throttling

- Lambda function can have 1000 simultaneous execution across our entire region, (though it can be increased through tickets).
- Let's suppose we've 2 Lambda functions running at the same time & if we've a huge load we'll have a lot of invocations.
- It is possible to set a “reserved concurrency” at the function level basically to say that the function should have a reserved capacity & be able to run up to at least 100 function concurrently at the same level.
- Each invocation over the concurrency limit, will trigger a throttle. Like say we can only have 100 Lambda function executing at the same time t if we've a new request coming in, we'd need to invoke 101 lambda function & that doesn't work, since we're over the limit so it needs to trigger a throttle.
 - If we invoke the throttle synchronously, for e.g.: from AWS console, then it returns a throttle error 429
 - If it is asynchronous invocation, it retries automatically & if it retries too many times then it goes to DLQ.

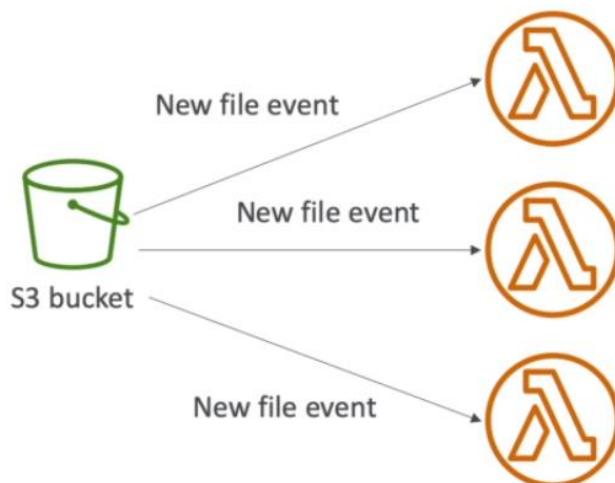
Concurrency Issue



- We've an ALB connected to Lambda function. We've another application where few users are connected to the API Gateway connected to another Lambda function. And we've another application using the SDK/CLI to invoke a Lambda function. When everything is running at low level of invocation, it seems fine. But suppose there're many users hammering the

ALB, hence our Load Balancer will be invoking many Lambda function. But the problem here is all the concurrent execution went to the first application. So, application users of API Gateway & CLI/SDK will be throttled. The concurrency limit applies to all the functions in our account, so we've to be careful that if one function goes over the limit, the other function gets throttled.

Concurrency and Async Invocations



- We upload files to S3 bucket and create New File event that'll invoke our Lambda functions and say we're putting many files at the same time, hence many concurrent executions happening. If the function don't have enough concurrency available then the additional request are throttled. But since this an Asynchronous event having throttling errors (429) and system errors (500 series), Lambda returns event to the queue and attempts to run the function again for up to 6 hours.
- The retry interval increases exponentially from 1 sec to max of 5 minutes.

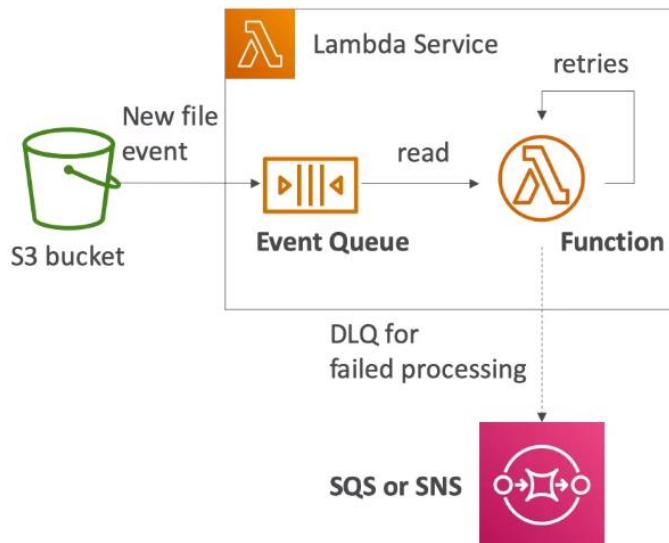
Cold Start & Provisioned Concurrency

- Cold Start
 - When we start a new instance, code is loaded outside the handler run (init).

- If the init is large (consisting of lots of code dependencies, SDKs, ...) this process can take lots of time.
- Hence, first request is served by new instances has higher latency request than the rest.
- For the users to prevent this bad experience, we can choose Provisioned concurrency.
- Provisioned concurrency
 - We allocate concurrency before the function is invoked.
 - This prevent Cold start and all the invocation has low latency.
 - We can use Application auto scaling to manage concurrency (e.g.: schedule or target utilization so that we've enough reserved Lambda functions ready to be used and minimize Cold start issue).

Lambda Retries & DLQ

Since our Lambda function can be invoked synchronously or asynchronously.



- If we perform synchronously and if it fails, then we need to retry and we can use exponential-backoff.
- In Asynchronous invocations, the events are placed in Event Queue.
- If it is asynchronously & fails, AWS retries automatically twice & if after all the retries (in total Lambda function retries for 3 times), it still does not work then the unprocessed events goes to Dead Letter Queue if we set it up.

- For AWS Lambda, DLQ can be SNS topic (if we want to send notification) or SQS queue.
- Asynchronous invocation allows us to speed up the processing if we don't need to wait for the result (like 1000s of files getting processed in parallel).
- Services - Asynchronous invocation involve:
 - S3
 - SNS
 - CloudWatch/EventBridge
 - CodeCommit
 - CodePipeline
 - CloudWatch Logs
 - Simple Email Services
 - CloudFormation
 - Config
 - IoT, IoT Events
- E.g.: let's talk about thumbnail creation service, S3 is an asynchronous trigger. So we upload an image, and it turns out that our Lambda function cannot process our image. And so it goes into a retry loop so we'll try twice again to process the image and may it still doesn't work and we don't have time to change the code. We can just set up a DLQ and go straight to SNS or SQS understand what's happening.

We must have a correct IAM execution role because Lambda function sends data to SQS or SNS otherwise it'll not be able to write events. It may happen that there is wrong with the image. Our Lambda function can send a messages straight to the SNS topic as DLQ or SQS. The original event payload is sent to the DLQ. These is an easy way to debug what's wrong with our function in production without changing the code.

LAB

- 30.On the same page, selecting the function created under Lambda.
- 31.Go to “Edit asynchronous configuration”
- 32.We can select SQS or SNS from DLQ. For now select Amazon SQS.
- 33.Choose a queue created earlier & hit save. While saving, it shows an error “the role does not have permission to send message to SQS”.

34. Go to IAM under services.
35. The role we created in step 13, we need to change its policy. Click on the role & hit on “Attach policy” -> SQS full access.
36. Hit ‘Save’ & we can see it saved without any errors.
37. We can have 1000 concurrency limit but we can either use the unreserved concurrency limit to run our function or we can use Reserve concurrency to tell our function should max run at concurrency of 20. And if it goes above 20, we want to throw errors & to throttle.
38. Hit Save.
39. Click on throttle & we can see that the Reserve concurrency is set to 0.
40. If we test it back, the function throws an error saying-> calling the invoke API failed with message: Rate exceeded.
41. If we set the value again to 20 Test after saving, then it runs without error.

CloudWatch Events/EventBridge

- It can be done through 2 methods:



- Serverless CRON/rate: we create an EventBridge Rule which triggers our Lambda function every hour to perform a task.

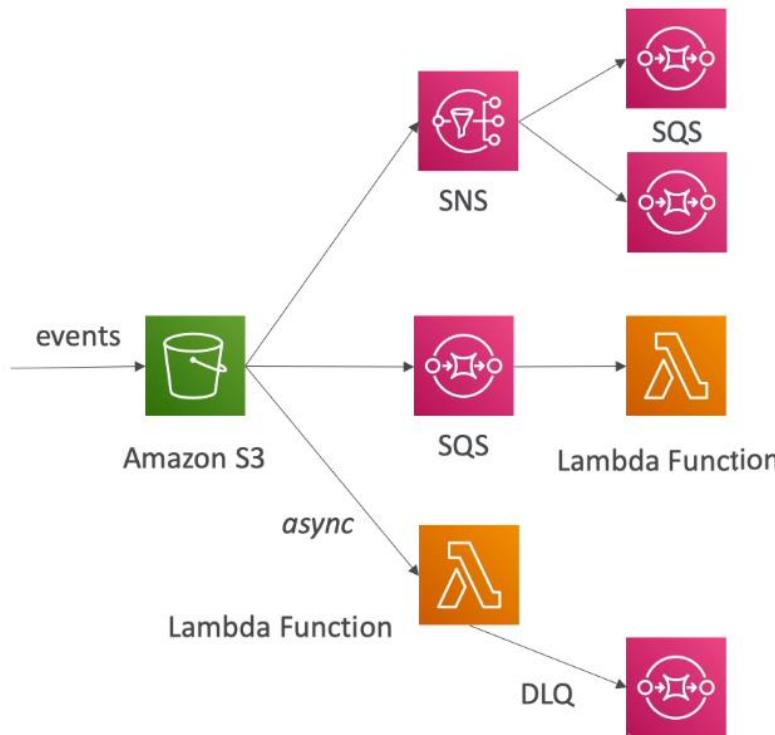


- We can also create a CodePipeline EventBridge rule to detect every time state changes & on state changes, we are going to invoke our Lambda function to perform a task.

S3 event notification

- It is a way to get notified whenever an object is created, removed, replicated or restored.
- We can filter the object name like (*jpg)
- Use case: Generating thumbnail of images uploaded to S3.

The images are uploaded to S3. S3 can send it to 3 things:



- SNS: we can later do a fan-out pattern to send it to multiple SQS.
- SQS: Directly sending to SQS and Lambda function directly read off that SQS queue.
- Lambda function: we can have S3 event notification which directly invokes our Lambda function. In case something goes wrong, we can set-up dead letter Queue (like SQS).
- S3 event notifications typically deliver events in seconds but sometimes may take minute or longer.
- If 2 writes are made to single non-versioned object at the same time, it is possible that only a single notification will be sent. Hence if we want to ensure that event notification is sent for every successful write, we can enable versioning.

Logging Monitoring & X-Ray Tracing

- CloudWatch Logs
 - AWS Lambda Execution logs are stored in AWS CloudWatch logs.
 - For this, we need to make sure our Lambda function is authorized to write to CloudWatch using IAM
- CloudWatch Metrics
 - AWS Lambda metrics are displayed in CloudWatch Metrics
 - It displays information about the invocation, durations, and concurrent executions, error counts, success rates, throttles, Asynchronous delivery charges.
 - If we're reading from Kinesis or DynamoDB streams, the iterator age which defines how lagging we are reading of these strings.
- X-Ray
 - It is possible to trace Lambda with X-Ray.
 - To enable it, we just need to enable it under Lambda configuration.
 - Then we need to use AWS SDK in code.
 - We need to ensure Lambda function has correct IAM Execution Role.
 - The managed policy is called AWSXRayDaemonWriteAccess.
 - These are the environment variables to communicate with X-Ray:
 - _X_AMZN_TRACE_ID: contains the tracing header.
 - AWS_XRAY_CONTENT_MISSING: by default, LOG_ERROR
 - AWS_XRAY_DAEMON_ADDRESS: the X-Ray Daemon_IP

LAB

42. Go to Monitoring tab, selecting the function created under Lambda.
43. We can see the number of times our function got invoked, max & Min duration for which function was running, Throttle, DeadLetter errors. The duration tab can help in calculating 'Timeout' values.
44. Go to AWS X-Ray.

45. We can enable Active Tracing. It shows a message the function's execution role does not have the permission to send trace to X-Ray. AWS attempts to add permission for us.
46. Hit on 'Save'.
47. Go to IAM under the role created in step13.
48. Check for the policies, we can see that a new permission has been added.
49. Go to X-ray under AWS X-Ray.
50. Click on the "Active tracing" check-box. When we save our function, Lambda automatically adds the permissions "xray:PutTraceSegments" & "xray:PutTelemetryRecords" to function's current role.
51. Now, let's test the function. Click on test button.
52. Go to X-Ray, click on Service map. We can see the Lambda function was invoked.

Limits to know

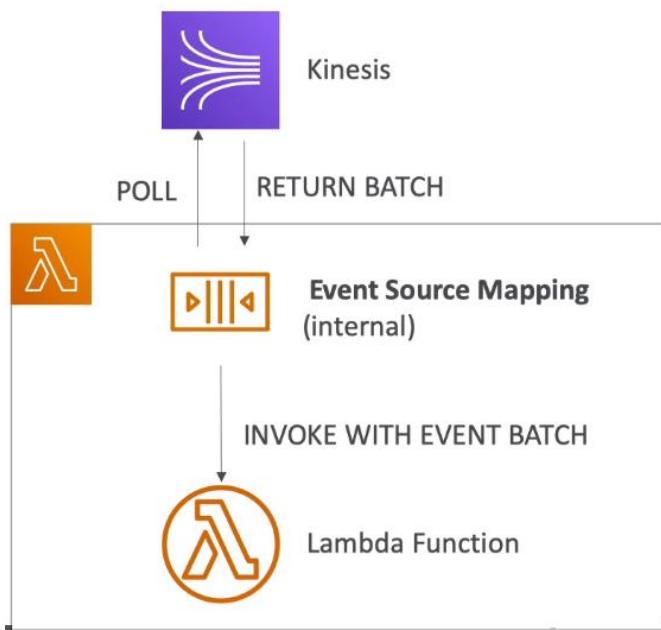
- Execution
 - Memory allocation: 128MB to 3008MB (64MB increments).
 - The maximum execution time is 15 minutes
 - Disk capacity in the function container is 512MB. Like if we need to pull images, databases, data sites, we can write up to $\frac{1}{2}$ GB into the /tmp directory.
 - The max number of lambda function that can execute concurrently is 1000 which can be later increased through support ticket.
- Deployment
 - Lambda function is deployed by uploading a .zip file whose max size is 50MB
 - When we un-compress the zip file which contains the code & dependency, the max size can be 250MB.
 - If we want to go over the limit, we could load files or dependencies at startup & write to our temp directory, & it gives us potentially bit more to work with Lambda.
 - Size of the environment variables are pretty small: 4KB. So we can't store whole files in our environment variables.

Event Source Mapping

It is one of the way to process events in AWS. It applies to:

1. Kinesis Data Streams
2. SQS & SQS FIFO queue.
3. DynamoDB streams.

One thing common in them is Lambda needs to poll from their source. Our Lambda function is invoked synchronously.

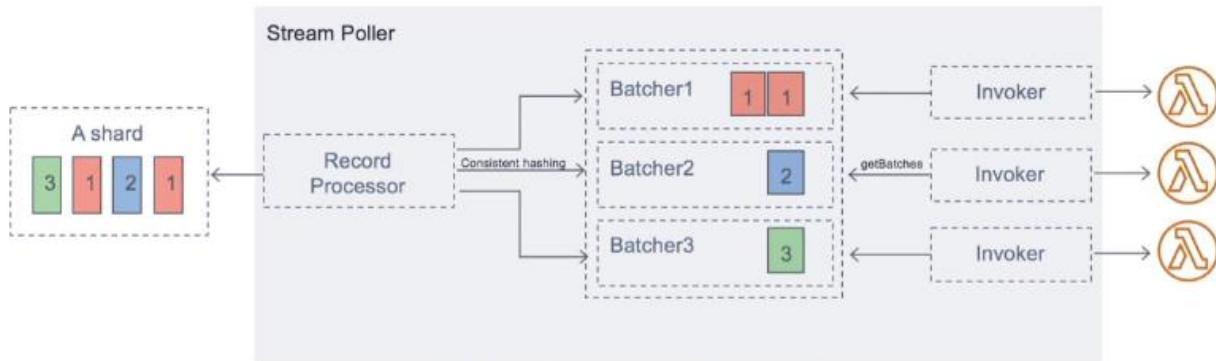


We need to configure Lambda to read from Kinesis. Automatically event source mapping is created which is responsible for polling Kinesis. It is also useful for returning the result from Kinesis and Kinesis returns a batch to us. Once the event source mapping has data to process, it invokes our Lambda function synchronously with an event batch.

There are 2 sources of event source mapper:

1. Streams - Kinesis Data Streams, DynamoDB streams.
2. Queues -

Streams



- In case of streams, there's an event source mappings. It'll create an iterator for each shard (Kinesis shard/DynamoDB shard). Items will be processed at the shard level.
- Can start with new items, from the beginning of shard or from a specific timestamp.
- Processed items are not removed from the stream so that other consumers can use them.
- Low traffic: we can use the batch window to accumulate records before processing.
- If we've high throughput stream and we want to speed up processing, we can setup Lambda to process multiple batches in parallel at the shard level.
- We can have up to 10 batch processes per shard and we have in-order processing for each batch at partition key level, it'll not be read in order entirely for the shard but each key in the shard will be read in order.

Errors

- By default, if function returns an error, entire batch gets reprocessed until the function succeeds or items in the batch expire.
- To ensure in-order processing, processing for the affected shard is paused until the error is resolved.
- We can configure event source mapping to:
 - Discard old events
 - Restrict the number of retries.
 - Split the batch on error (to work around Lambda timeout issues).
- Discarded events can go to Destinations.

SQS & SQS FIFO



The SQS is polled by Lambda Event Source mapping and whenever a batch is returned, Lambda function will be returned synchronously with the event batch.

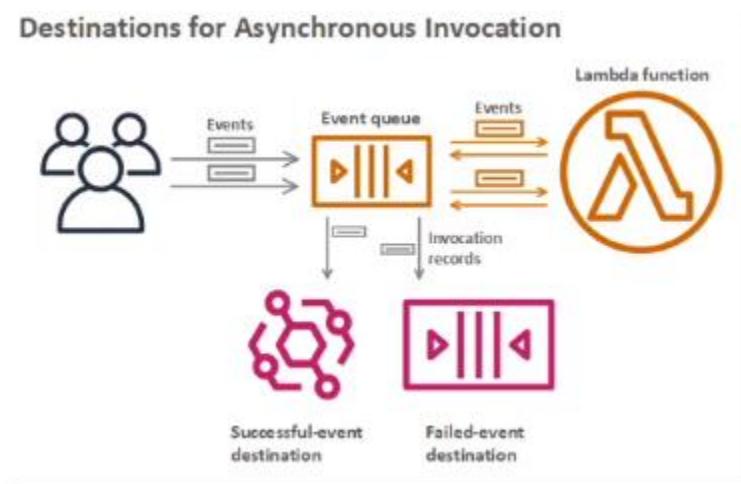
- Event Source Mapping will poll SQS (using Long polling)
- The batch size which is specified is of 1-10 messages.
- It is recommended to specify the queue visibility timeout to 6 times the timeout of Lambda function.
- To use a DLQ
 - We need to set-up the DLQ on the SQS queue not on Lambda (DLQ on Lambda is only for asynchronous invocation).
 - We can use a Lambda destination for failures.
- Lambda supports in-order processing if we have a FIFO queue.
- Number of Lambda function that'll be scaling to process the queue equals number of active message groups.
- For standard queue, items will not be processed in order.
- For standard queue, Lambda scales as fast as possible to read all the messages.
- If errors happens, batches are returned to the queue as individual items and might be processed in a different grouping than original batch and might be processing in a different grouping than the original batch.
- Sometimes, the event source might receive the same item from the queue twice, even if no function error occurred.
- Lambda deletes items from the queue after they're processed successfully.

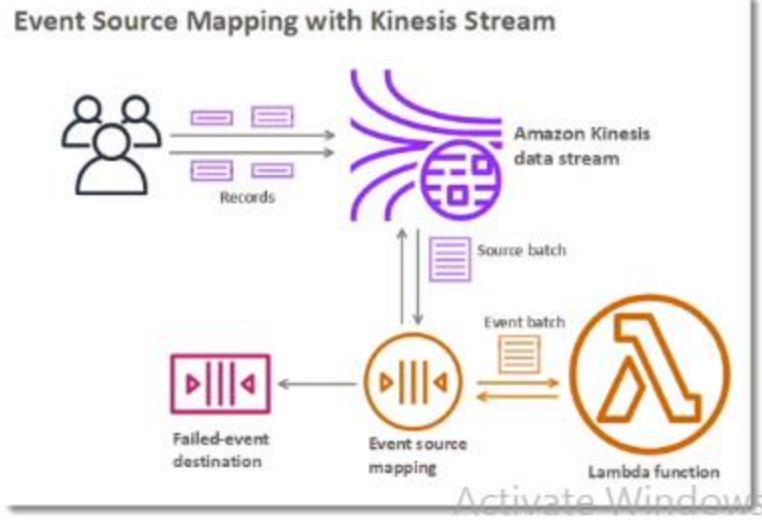
- Source queue can be configured to send items to Dead-letter queue if they can't be processed.

Lambda Event Mapper Scaling

- Kinesis Data streams & Dynamo DB Streams:
 - One Lambda invocation per stream shard.
 - If we use parallelization, we get up to 10 batches processed per shard simultaneously.
- SQS Standard:
 - Lambda adds 60 more instances to scale up.
 - Up to 1000 batches of messages processed simultaneously.
- SQS FIFO:
 - Message with same group ID will be processed in order.
 - The Lambda function scales to the number of Active Message Groups.

Destination





1. When we used Asynchronous invocation or Event Mappers, it was hard to see whether it failed or succeeded with its data.
2. So, Destination sends the data of asynchronous invocation/failure.
3. For Asynchronous invocation, we can define destination for successful & failed events:
 - SQS
 - SNS
 - Lambda
 - EventBridge bus
4. AWS recommends to use Destination instead of DLQ (both can be used at the same time) because Destination allow for more targets.
5. Event Source Mappings are used for discarded events batches because we cannot process it. So, we can send that event batch to:
 - SQS
 - SNS

Lambda Resource Execution Role

- It grants Lambda function permission to AWS Services/resources.
- Sample Managed policies for Lambda:
 - AWSLambdaBasicExecutionRole: Uploads logs to CloudWatch.
 - AWSKinesisBasicExecutionRole : Reads from Kinesis.
 - AWSLambdaDynamoDBExecutionRole: Reads from DynamoDB Streams.

- AWSLambdaSQSQueueExecutionRole: Reads from SQS.
- AWSLambdaVPCExecutionExecutionRole: Deploy Lambda function in VPC.
- AWSXRayDaemonWriteAccess: Uploads trace data to X-Ray.
- When we use an event source mapping to invoke our Lambda function, Lambda uses the execution role to read event.

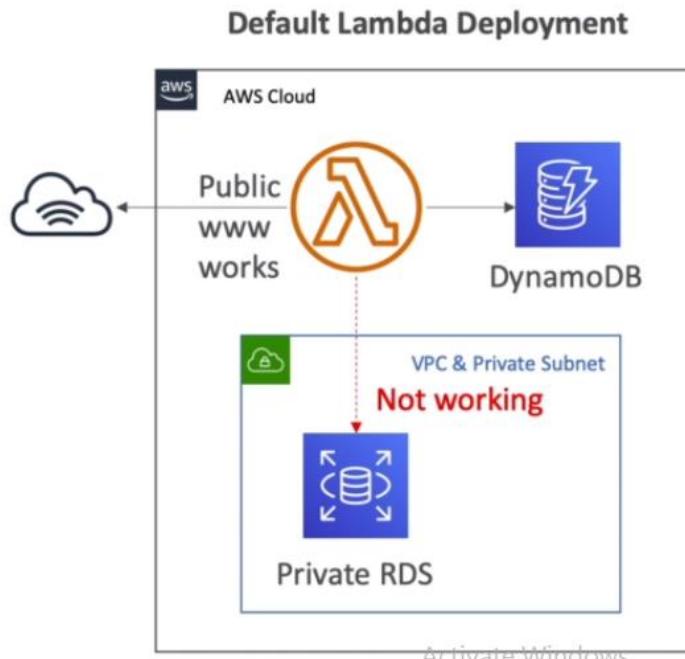
Resource Based Policies

- It is used to give other accounts/AWS services to use our resources.
- It is similar to S3 bucket policies.
- An IAM policy can access Lambda function:
 - If the IAM policy attached to the principal authorizes it.
 - If we have resource based policy which authorizes it.
- When resources like S3 calls our Lambda function, the resource based policy gives it access.

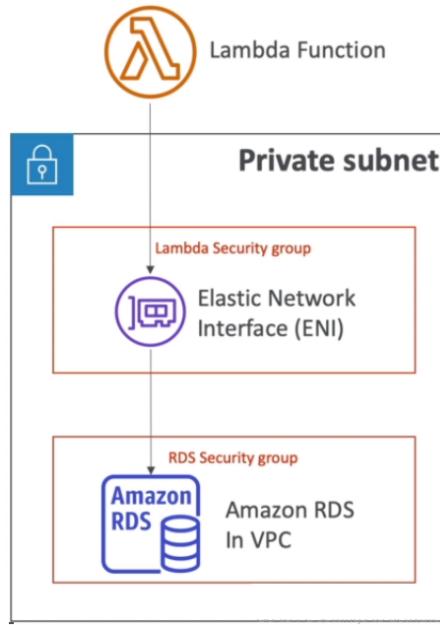
Environment Variables

- It is the key/value pair in the "string" form.
- It help adjusting the function behavior without adjusting the code.
- The environment variables are available to our code.
- Lambda variables adds its own system variables as well.
- These environment variables can be encrypted using KMS which help us to store secret values.
- These secrets can be encrypted by the Lambda service key, or our own CMK.

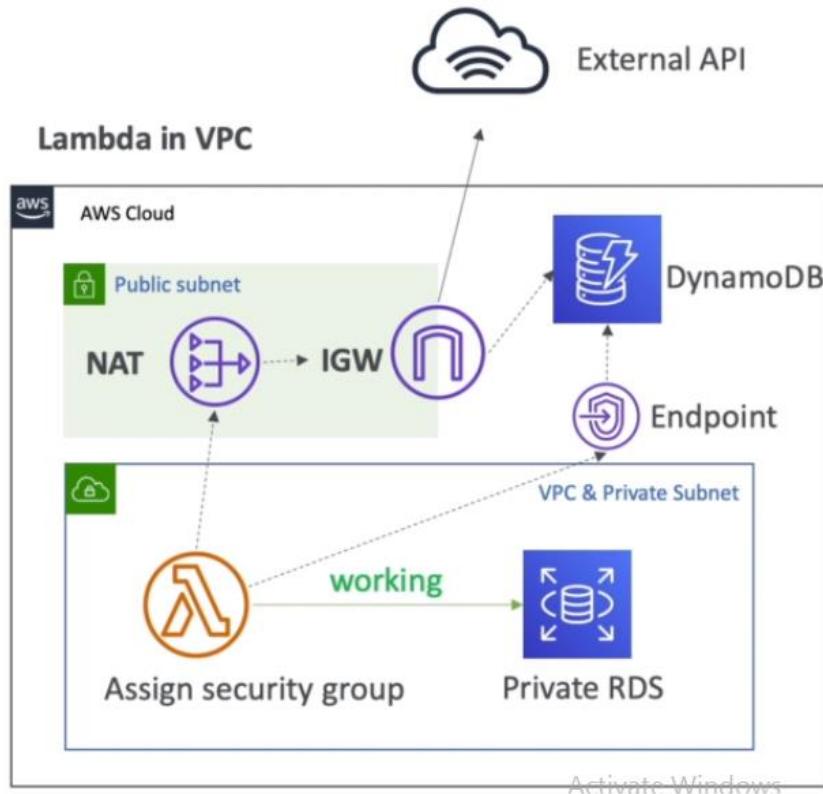
Lambda in VPC



- By default, our Lambda function are launched outside our own VPC (in AWS-owned VPC).
- Hence, it can't access resources in our VPC (RDS, ElastiCache, internal ELB).
- In Cloud, the Lambda function can access any public websites and services such as DynamoDB. But if we've our own VPC in private subnet having a private RDS, then Lambda cannot access RDS.
- To make it accessible, we need to define the VPC ID, the subnets and the security groups.
- Behind the scenes Lambda creates an ENI (Elastic Network Interface) in our subnets and we need to give AWSLambdaVPCAccessExecutionRole.



- Now, we've RDS security group on RDS in VPC. We give VPC access to Lambda function, after setting up, it creates an ENI alongside the Lambda security Group. To access our RDS, the Lambda passes ENI and to make it work, RDS security group does allow network access from Lambda security group.
- The Lambda function inside VPC does not have access to the internet.
- Deploying the Lambda function in a public subnet does not give it access to public internet or a public IP.
- Hence we can deploy the Lambda function in a private subnet & give it internet access if we've a NAT Gateway/Instances.



- In the cloud, we've Lambda function in VPC which is deployed in the private subnet having access to RDS. To access a public API we need to go through the public subnet through the NAT device (NAT Gateway/NAT instance) and it'll be talking to the Internet Gateway of our VPC & Internet Gateway gives access to external API.
- We can access DynamoDB through public route and through Internet Gateway and this works once NAT is put in place or if we want to access DynamoDB privately, we can use VPC endpoints (used to access AWS services privately within the cloud). Hence, we'll create VPC endpoint for DynamoDB and Lambda function will be talking to the endpoint, hence accessing our DynamoDB service.
- If we deploy Lambda function in private subnet, the CloudWatch logs work even if we do not have any endpoints or NAT Gateway.

Performance

- RAM
 - From 128MB to 3008MB (3GB) in 64MB increments.
 - The more RAM we add, the more vCPU credits we get.

- At 1792MB, a function has equivalent to one full vCPU.
- After 1792MB, we get more than one CPU and we need to use multi-threading in our code to get benefit from it.
- If our application is CPU-bound (computation-heavy), increase RAM.
- Timeout: default is 3 seconds, maximum is 900 seconds (15 minutes).
- Anything above 15 minutes is not a good use case for Lambda and it will be better if we use Fargate, ECS or EC2.
- The execution context is temporary runtime environment that initializes any external dependencies of our Lambda Code.
- It is great for DB connections, HTTPS clients, SDK clients.
- If we invoke Lambda function multiple times in a row, then the invocation context can be reused and can reuse all these existing DB connection, HTTP clients, etc. or we can say the execution context is maintained for some time in anticipation of another Lambda function invocation.
- The lambda function uses the /tmp directory.

BAD!

```
import os

def get_user_handler(event, context):
    DB_URL = os.getenv("DB_URL")
    db_client = db.connect(DB_URL)
    user = db_client.get(user_id = event["user_id"])

    return user
```

The DB connection is established
At every function invocation

GOOD!

```
import os

DB_URL = os.getenv("DB_URL")
db_client = db.connect(DB_URL)

def get_user_handler(event, context):
    user = db_client.get(user_id = event["user_id"])

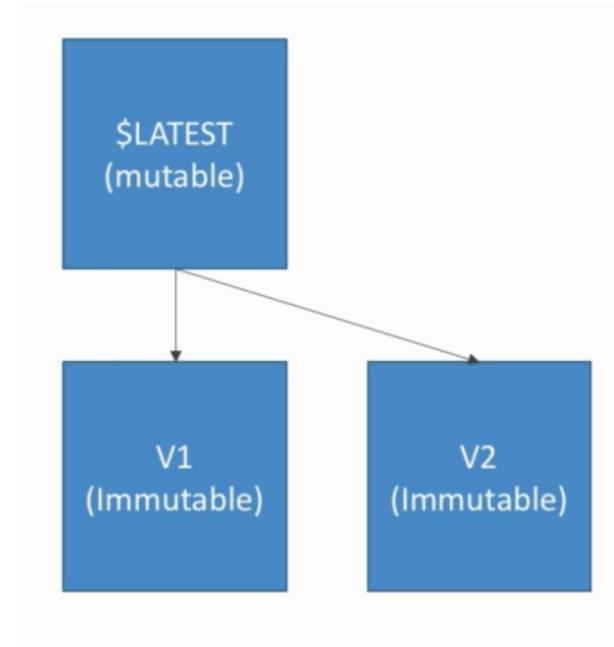
    return user
```

The DB connection is established once
And re-used across invocations

- The good practice is to initialize the DB connection outside of handler, so that it'll be initialized once and can be re-used across function calls and can greatly improve function performance rather than connecting to DB multiple times and then getting the user.

[Qualifiers](#)

[Lambda Versions](#)

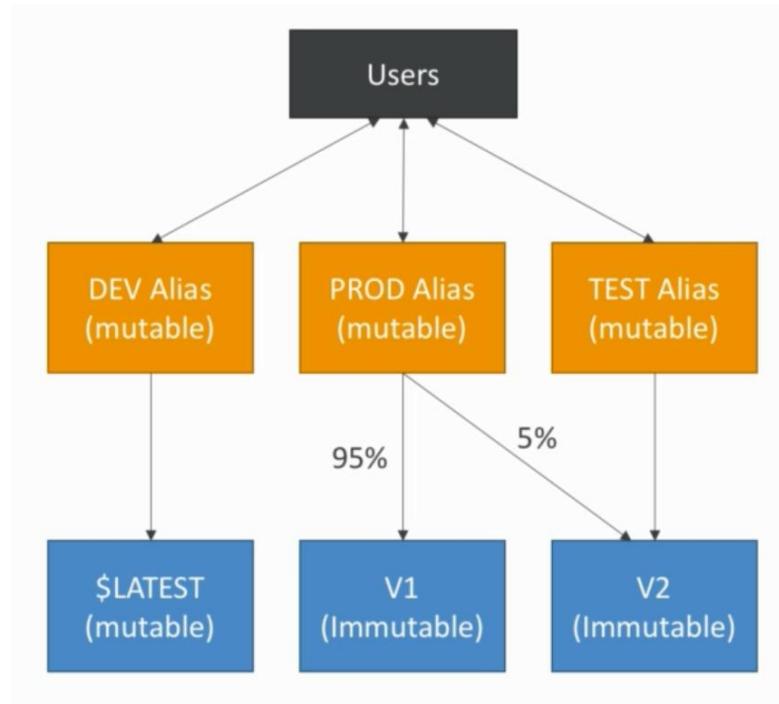


- When we work on lambda functions, we work on \$LATEST which is mutable meaning it can change.
- When we're ready to publish a Lambda function, we create a version. The first version that'll be created will be known as V1. Now, that version is immutable. V1 cannot be changed ever, it's a snapshot.
- After V1, we'll have V2 that is they have increasing version numbers. Version get their own ARN (Amazon Resource Name).
- Version is the code & configurations (so, it means nothing can be changed or everything is immutable).
- So, each version of the Lambda function can be accessed using the ARNs.

Lambda Aliases

We create versions in order to have a DEV, Test & Prod. If we've new versions of the time, we need to re-wire everything in Dev, Test & prod.

- Aliases are pointers to Lambda function versions
- We can define a Dev, Test & Prod aliases & have them point at different Lambda versions and we can change these pointers over tie since they're mutable.
- Aliases are mutable.



- Here we have our \$LATEST Lambda function & we've 2 different versions, V1 & V2. We'll create our Dev Alias which represents our Development environment which'll point to the latest function we have. Users don't point to latest directly, they interact with Dev alias. Now, we've Test alias which'll be testing our V2 function. We may have a prod function, & the prod alias is pointed to V1 directly. So, we can see that all Aliases are mutable while versions are immutable.
- If we want to do Blue\Green deployment or Update our Prod environment to migrate it from V1 to V2 because let's say, the code was great. Now we want to do deployment and assign events. We've prod Alias to be 95% pointing to V1 & 5% of it is going to point to V2.
- Since our users are interacting with Aliases, they don't know what's happening at all.
- In the backend, Aliases enable stable configuration of our event triggers/destination.
- Aliases have their own ARN.

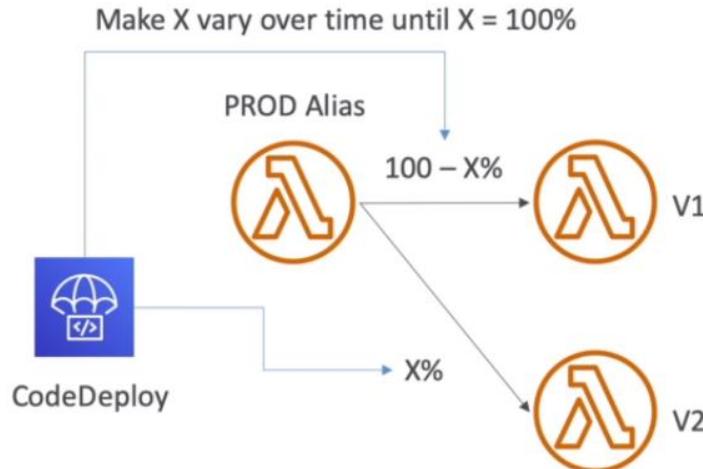
LAB

1. Click on Qualifier tab in your Lambda function created.

2. We can see there are 2 sub-tabs, Version (which show latest) & Aliases (we don't have any, so nothing shows up right here).
3. Click on 'Action', & select 'Publish New Version'.
4. Enter the version name/description & hit on 'Publish'. A Snapshot is been taken & named as our first version.
5. It can talk to all our Services that we attached with our Latest version.
6. We can see the function code but it cannot be edited. So V1 is immutable.
7. Let's go back to Latest & modify the code a little & remove the environment variable & hit 'Save'.
8. Go to action & select "Publish new version".
9. The goal is to expose our user is a Dev, Test, & Prod endpoints and ARNs.
10. Click on "Create Aliases" under Actions. We can see that an Alias is a pointer to one or 2 version.
11. Let's create a Dev Alias. Give it a name & description.
12. Select the version to which'll be pointing to. Let's select \$LATEST for now.
13. Hit on 'Create'. We can see the ARN (Amazon Resource Name).
14. Similarly, create Alias for PROD environment pointing to first version.
15. We've 3 different versions of our function while the Prod environment is pointing to version one & Dev environment is pointing to Version Latest. Let's try to update our website our PROD at version two.
16. Go to Alias & hit on PROD which is pointing to Version one.
17. One way is to directly select the newer version. We'll try to shift traffic between the 2version. Under weight, select 50% to be Additional version's weight.
18. Hit on 'Save'
19. Click on 'Test'
20. This way, we the traffic sometimes goes to V1 or sometimes V2.

Lambda & CodeDeploy

- It can help us to automate the traffic shift for Lambda Aliases.
- This feature is integrated within the SAM framework.



- Let's suppose we've PROD alias which we want to upgrade it from version 1 to v2. We want to shift 100% of traffic from v1 to v2. CodeDeploy makes the 'x' variable vary over time making 'x' vary over time until 'x' = 100%.
- Types of variation:
 - Linear: growing traffic every N minutes until 100%.
 - Linear10PercentEvery3Minutes.
 - Linear10PercentEvery10Minutes.
 - Canary: Try X percent then 100%.
 - Canary10Percent5Minutes.
 - Canary10Percent30Minutes.
 - AllAtOnce: Immediate. It might be dangerous since we've not tested our V2 function, then things can fail if any error comes.
- We can create Pre & Post traffic hooks to check the health of our Lambda function. So, if anything goes wrong, then traffic hooks can fail or CloudWatch alarm can fail, hence CodeDeploy will know, something is going wrong and it'll do a rollback and put the traffic back to previous version.

External Dependencies

- Let's suppose our Lambda function is dependent on external libraries: for e.g.: AWS X-Ray SDK, Database Clients, etc. then we need to install the packages along with our code & zip it together.
 - If we use Node.js, use npm&node_modules directory.
 - For python, use pip --target options.

- For Java, we can include the relevant .jar files.
- Now, we can upload the zip file straight to Lambda if less than 50MB, else we need to upload to S3.
- Native libraries work: They just need to be compiled on Amazon Linux. Native libraries are those which contains native code or the code that has been compiled for the specific hardware architecture or operating system.

LAB

1. Run the code with external dependencies locally on your machine.
2. Zip the code.
3. Go to Lambda under services.
4. Create a function.
5. Select “Author from scratch”.
6. Select the language.
7. Choose to create a new IAM Role.
8. Hit on “Create function”.
9. Under “Function Code”, choose “Upload a .zip file” from Code entry type tab.
10. Hit upload.
11. Click “Save”.
12. Go to IAM under services.
13. Go to the specific Roles, select “Attach policy”.
14. Attach “S3ReadOnlyAccess” so that it can list the buckets and “AWSXrayWriteOnlyAccess” so that we can publish X-Ray traces.
15. Under AWS X-Ray, enable the checkbox of “Active Tracing”.
16. Click on Test & create a Test event.
17. Under Execution result’s details, we can see the list of all our buckets.
18. Go to X-Ray under Services.
19. Go to Service Map

Lambda & CloudFormation

```

AWSTemplateFormatVersion: '2010-09-09'
Description: Lambda function inline
Resources:
  primer:
    Type: AWS::Lambda::Function
    Properties:
      Runtime: python3.x
      Role: arn:aws:iam::123456789012:role/lambda-role
      Handler: index.handler
      Code:
        ZipFile: |
          import os

          DB_URL = os.getenv("DB_URL")
          db_client = db.connect(DB_URL)
          def handler(event, context):
              user = db_client.get(user_id = event["user_id"])
              return user

```

- Inline functions are very simple.
- Use the code.zip file property.
- We cannot include dependencies with inline function.

```

AWSTemplateFormatVersion: '2010-09-09'
Description: Lambda from S3
Resources:
  Function:
    Type: AWS::Lambda::Function
    Properties:
      Handler: index.handler
      Role: arn:aws:iam::123456789012:role/lambda-role
      Code:
        S3Bucket: my-bucket
        S3Key: function.zip
        S3ObjectVersion: String
      Runtime: nodejs12.x

```

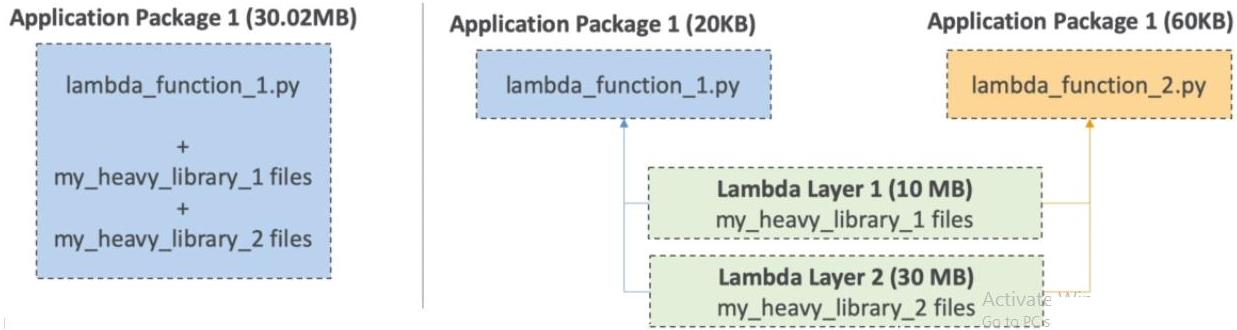
- The other way to do it is by using it as a zip file and do it through S3.
- We need to refer the S3 zip location in the CloudFormation code. It can be
 - S3Bucket attribute.
 - S3Key: full path to the zip.
 - S3ObjectVersion: if bucket is versioned.
- If we update the code in S3, but do not update the S3 bucket, S3 key or S3ObjectVersion, then CloudFormation won't update our function.

How to use Lambda with CloudFormation:

1. First, we need to store the Lambda zip file in S3.
2. We must refer S3 zip location in the CloudFormation code. Just to keep in mind, all the regions must be same.
3. Code the template for CloudFormation.
4. Go to S3 & Create a bucket.
5. Upload the zip file in the bucket.
6. Go to CloudFormation & create new stack.
7. Choose Template is Ready & Upload a template file.
8. Click on choose a file & upload the .yaml file.
9. Define the stack name.
10. In the bucket parameter, copy the bucket name.
11. Copy the zip file name with its extension under S3keyParam.
12. Set other setting to be default & click on Next.
13. Enable the check-box which acknowledges about creating an IAM Role.
14. Create the stack. The Lambda function will get the code directly from the .zip file.
15. Go to lambda & navigate to functions & we can see that our function has been created.
16. When we click on the function, a message is displayed "This function belongs to AWS CloudFormation stack **LambdaThroughCloudFormation**.
17. Click on "Test" & "Create".
18. Under Execution result, we can see the result.
19. Let's go back to CloudFormation & we can see the all the process has been completed - CREATE_COMPLETE.

Lambda layers

- It is used to create custom runtimes for Lambda. A Like the languages like C++, Rust, etc. which are not meant for Lambda function initially but the community as decided to support through Lambda layers.
- It is also used to customize external dependencies in order to re-use them.



- If we look at a zipped application package, it could be pretty big consisting of Lambda function, some heavy libraries, heavy dependencies and high size in MB. And, in order to update the Lambda function, we need to re-upload the zip file multiple times but it may happen the external dependencies may remain constant.
- So the goal is having application package whose code can change too often. We've also created layers for heavy libraries having first layer of 10MB, layer 2 of 30MB. These function can be referencing our layers and hence we've a faster way to deploy application, hence we do not need to re-package our application dependencies.
- Since our layers are externalized, another function, another application package could create another function and reference these same layers.

/tmp space

- If our Lambda function needs to download a big file to work, or
- If our Lambda function needs disk space to perform operations, then
- We can use the `/tmp` directory.
- Its max size is 512 MB.
- The advantage of temp directory is that the directory content remains even if the execution context is frozen. So, it gives us some cache that can be used if our Lambda function is invoked multiple times. It can be helpful if we want to checkpoint our work.
- For permanent persistence of our object (non-temporary), S3 can be used.

Best Practices

- Perform heavy-duty work outside our function handler.

- Connect to the DB outside the function handler otherwise every time when our function handler will be invoked, we're going to reconnect to our DB.
- Initialize the AWS SDK outside our function handler.
- Pull in dependencies or datasets should be kept out of function handler.
- Environment variables should be used for:
 - Database connection string, S3 bucket, etc. should not be put directly in to out code, that's why environment variables are for.
 - Password, sensitive values can be used with environment variables, they can be encrypted using KMS.
 - Environment variables have maximum size of 4KB.
- Minimizing our deployment package size & we need to make sure that the only runtime necessities and the dependencies are put in our code.
 - If we've too big function, break it down into several Lambda functions.
 - The size of the file must be less than 250MB uncompressed & 50MB when compressed.
- Never have a recursive code in Lambda function, other the Lambda function may invoke itself and the bill may get much higher.
- Don't put the Lambda function inside our VPC, the reason is if we put in our VPC it'll take longer to be initialized whereas if you don't put in our VPC, it'll be very quick to initialize.

LAB

1. Got to the function created in Lambda under services.
2. Select the \$LATEST version code.
3. Modify the code under function code.

```

import time
importos

deflambda_handler(event, context):
    print("Connecting to the database...") //Simulation connect to
    DB
    time.sleep(2)                      // DB connection takes 2 sec.

```

```

print("Connected to the database!!!")

print("Doing some work")      //Doing a piece of work
return event['key1']          //Return the event key.

```

4. Save the code & test it.
5. So, every time we run our code & every time it takes 2 seconds to connect to our DB.
6. We're going to modify our code: Connecting to the DB outside our event handler function.

```

import time
importos

print("Connecting to the database...") //Simulation connect to DB
time.sleep(2)                      // DB connection takes 2 sec.
print("Connected to the database!!!!")

deflambda_handler(event, context):

    print("Doing some work")      //Doing a piece of work
    return event['key1']          //Return the event key.

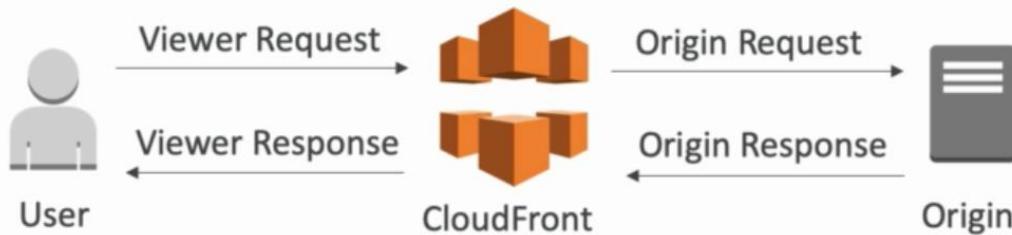
```

7. Save & test it.
8. We can note that the execution time changed from 2004ms to nearly 10ms.
9. If we test it again, it only takes nearly 1ms. This is because our DB initialization work has been done outside the Lambda function and it was done only first time & every time we called the handler & we can do some work & have some blazing quick execution.

Lambda@Edge

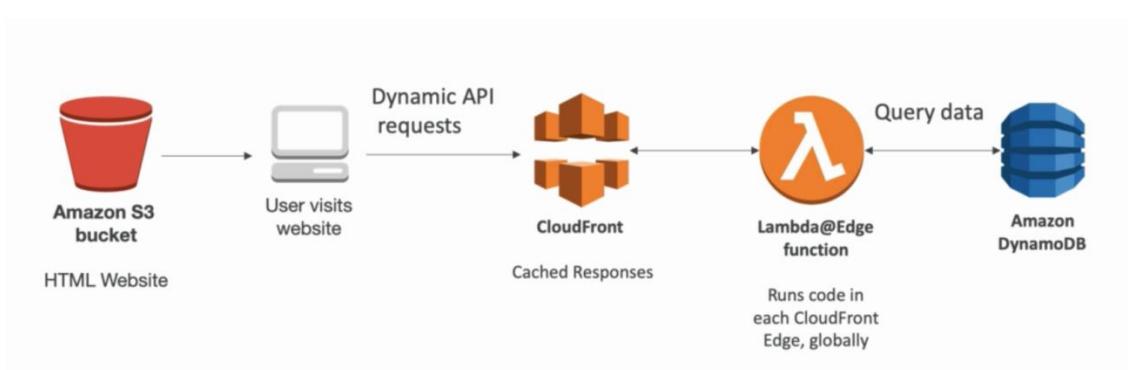
- If we want to run global AWS lambda alongside CloudFront, we'd use Lambda@Edge.
- It can be used when we want to implement request filtering before reaching to our application.
- Lambda@Edge is what we deploy our Lambda function not in a region but in every single CloudFront Edge location. So we can interpret Lambda@Edge as like a global Lambda function.

- It builds more responsive application.
- We don't need to manage servers as Lambda function is deployed globally as there's a central way of managing that.
- CDN contents can be customized.
- We need to pay only what we use.
- We can use Lambda to change CloudFront requests & responses. There're 4 types of request & response:



- After CloudWatch receives a request from a receiver (viewer request).
 - Before CloudFront forwards the request to the origin (origin request).
 - After CloudFront receives the request from origin (origin response).
 - Before CloudFront forwards the request to viewer (viewer response).
- So, Lambda@Edge can impact all of these & is a powerful tool because we can modify every single request that goes to CloudFront.
- There's no need even to go to the origin, Lambda itself can be quote-on-quote origin & return a response right away. So, it's completely to run a global serverless application if we use CloudFront & Lambda@Edge

Global Application



- Let's say we've a static website/ HTML website, uploaded into S3.
- User visits the website & pulls the HTML & other contents.
- For all the dynamic aspects of our websites, clients issue API requests to CloudFront.
- CloudFront caches our responses. It invokes Lambda@Edge function which runs locally, so it global, it runs at the edge & hence very quick & does not go to a different region to run our Lambda function.
- Our Lambda function can query data into Amazon Dynamo DB & return back to CloudFront if needed.

Use Cases

- Website security & privacy.
- Dynamic web application at the edge.
- Search engine optimization (SEO).
- We can intelligently route across Origins & Data centers.
- We can do Bot mitigation at the Edge by filtering out bad requests.
- Real-time image transformation.
- A/B testing (It is a method of comparing two version of a webpage or app against each other to determine which performs better).
- User authentication & Authorization.
- User prioritization.
- User tracking & analytics.

17. DynamoDB

It is a serverless database, which is managed by AWS and AWS scales it for us. It is really well integrated with AWS Lambda & other AWS services. It is also advertised as NoSQL Serverless.

Traditional architecture



- According to traditional architecture, the clients talk to our API layer (composed of EC2, ASG & ELB) & these API layer talks to our DB layer.
- Traditional application leverage RDBMS.
- These DB use traditional SQL Query language.
- DB should have strong requirements about how data should be modeled. It should be in tables, rows & should have joins, indexes & the benefit of these we're able to do within the DB.
- We've the ability to do join, aggregations, computation.
- If we want to scale our application, or we need to vertical scaling (getting a powerful CPU, RAM, disc)

NoSQL Databases

It is a new type of architecture. It means not only SQL databases.

- These're non-relational DB & are distributed & hence **we can't do joins**& since they're distributed there's going to be some horizontal scaling.
- It includes MongoDB, DynamoDB, etc.
- All the data needed in a query is present in 1 row.
- NoSQL databases don't perform any computations such as "SUM", all they do is providing data efficiently.
- They scale horizontally, this is why they are good fit for serverless. This is simply because the design of the DB.
- There's no right/wrong for NoSQL vs SQL, they just require to model the data differently & think about user queries differently.

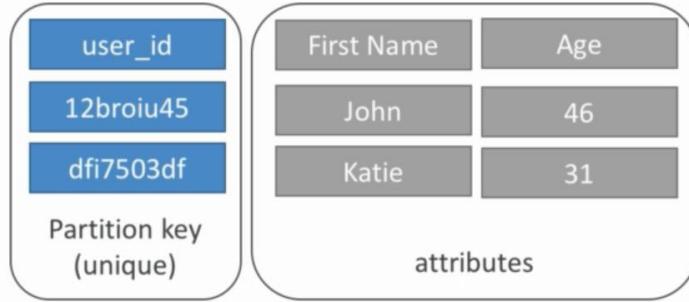
Features

- It is fully managed, highly available with replication across 3AZ.
- NoSQL DB – not a relational DB

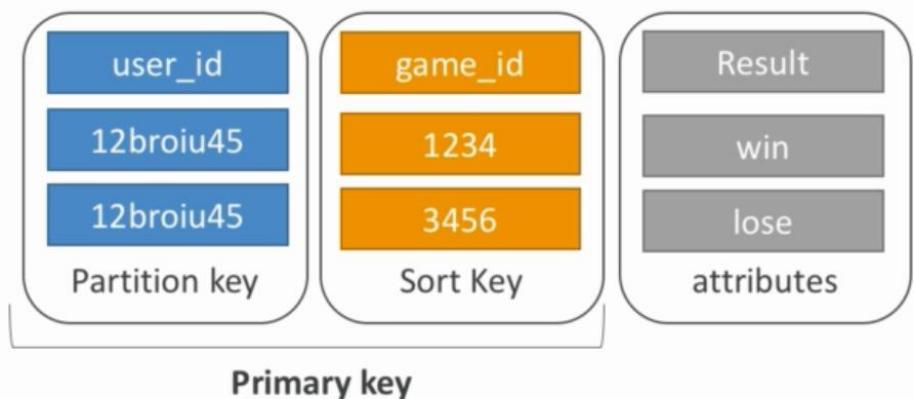
- Scales to massive workloads, distributed DB
- Scale up to millions of requests per second, trillions of rows, 100s TB of storage.
- Fast & consistent in performance (low latency on retrieval)/
- Integrated with IAM for security, authorization & administration.
- Since it's serverless, we don't have to manage. We just need to provision tables.
- Event driven programming (a programming paradigm in which the flow of the program is determined by events such as user's actions such as mouse clicks, keyboard inputs, sensor outputs) with DynamoDB Streams.
- Low cost & auto-scaling capabilities.
- It is made up of tables.
- Each table has a primary key which should be decided at the creation time.
- Each table can have infinite number of item (item = rows).
- Each item has attributes which can be added over time & they're basically equivalent to columns. These attributes can be nested & some attributes can be null).
- Maximum size of an item is 400KB.
- Databases types supported are:
 - Scalar types: String, Number, Binary, Boolean, Null.
 - Document list types: List, Map
 - Set types: String set, Number set, and Binary set.

Primary Keys

- **Option I:** Partition key only (HASH)
- Partition key must be unique for each item.
- Partition key must be “diverse”, so the data is distributed.



- Example: user_id for user table (it is a partition key so must be unique). For attribute we can have FirstName, Age, etc. So, the above table looks like a traditional one.
- **Option II:** Partition key + Sort Key (This makes Dynamo DB more specific).
- Here, primary keys can be a partition of Partition key & a sort key which must be unique.
- Data is grouped by a partition key, so all the data sharing the same partition key will be together and then they'll be sort by the sort key.
- Sort key is also called the range key. It allows us to perform some efficient queries.
- Example: users-game table



- user_id for the partition key
- game_id for the sort key
- The combination of user_id&game_id is our primary key. A user can have many games but a user can play only one game_id.
- In case of attribute, we can have the result of the game.

- We can note that the user_id is same but the game_id is different so we've a unique primary key.
- Example: Building a movie database – Our exercise is to form the best partition key to maximize the data distribution. So we've a
 - movie_id
 - producer_name
 - lead_actor_name
 - movie_language
 - movie_id has the highest cardinality (uniqueness of the data values contained in the column).
 - movie_language doesn't take many values & all the data may be lead towards English, so not a great partition.

LAB

1. Go to DynamoDB under services.
2. Hit on “Create table”.
3. Specify a table name. Here, we do not need to create a DB as AWS manages it for us.
4. Define a partition key. We can add a sort key. (Not for now).
5. Under Auto-Scaling, un-check the read & write capacity, make it 2-2 in both the case. It gives an estimated cost how much we're going to be billed for the month. A DynamoDB Write capacity unit can handle 1 read & write per second.
6. Leave the other settings to be default.
7. Hit on Create.
8. Overview tab gives the basic details about the table like Table name, Partition key & sort key, Encryption details, ARN, etc.
9. Items us gives the provision of adding data to our table.

	user_id	first_name	last_name	age
	234fdsfds4	Kelly		41
	iuyser874	Stephane	Maarek	

10. Click on “Create item”.
11. Click on ‘+’ & create an item for firstName, lName.

- 12.Hit on save.
- 13.We can see that the data is been written to the Table.
- 14.Similarly, create a 2nd item. While adding a new item, create a new column.

Provisioned Throughput

- In DynamoDB we need to provision Read & Write Capacity units.
- Read Capacity Units (RCU): Throughput for reads.
- Write Capacity Units (WCU): throughputs for writes.
- Option to set-up auto scaling of throughput to meet the demand.
- Throughput can be exceeded temporarily using “burst credit”.
- But if burst credits are empty, we get “ProvisionedThroughputException” error.
- It is advised to do exponential back-off retry.

Write Capacity Units

- One WCU represents one write per second for an item up to 1KB in size.
- If we go over 1KB size, then more WCU are consumed.
- Example:
 - Let's suppose we write 10 objects per second of 2KB each.
So we need $10 \times 2 = 20$ WCU
 - Let's suppose we write 6 objects per second of 4.5KB each.
So we need $6 \times 5 = 30$ WCU (4.5 gets rounded to upper KB)
 - Let's suppose we write 120 objects per minute of 2KB each.
So we need $120 / 60 \times 2 = 4$ (since it is per minute)

Strongly Consistent Reads vs. Eventually Consistent Reads

In DynamoDB we get to choose an option between Strongly Consistent Read & Eventually Consistent Read

- **Eventually Consistent Read:** if we read just after a write. It is possible that we'll get an old data or we may get no data. If we keep on asking eventually (few hundred milliseconds), we'll get the right data.
- **Strongly Consistent Read:** If we read data just after write, we'll get the correct data.

- By default, DynamoDB will use eventually consistent reads for GetItem, Query & Scan. But there's a “ConsistentRead” parameter which we can set to true & if we do so, we'll get a “Strongly Consistent Read”. We should not choose Strongly Consistent Read all the time because it'd create impact to our RCU.
- Example: let's suppose we've an application & our DynamoDB is replicated across 3 AZs. We've a distributed DynamoDB System with 3 servers. Our application will do write to our DynamoDB & maybe it'll reach to first server & then our server confirms write is correct. In background, data gets replicated to other two DynamoDB. Strongly consistent reads are little bit slower.

Read Capacity Units

- One read capacity units represents one Strongly Consistent Read per second or 2 Eventually Consistent Reads per seconds for an item up to 4KB in size.
- If item's size are larger than 4KB, more RCU are consumed.
- Example:
 - 10 Strongly Consistent Reads per second of 4KB
 $10 * 4KB / 4KB = 10 \text{ RCU}$
 - 16 Eventually Consistent Reads per seconds of 12KB each
 $(16/2) * (12/4) = 24 \text{ RCU}$
 - 10 Strongly Consistent Reads per second of 6KB each
 $10 * 8KB / 4 = 20 \text{ RCU} \text{ (we have to round up to 6KB to 8KB)}$

LAB

1. Go to the table created.
2. Navigate to the capacity tab.
3. Under “Provisioned capacity”, click on Capacity calculator.
4. Enter the value which we want to provision for Read/Write Capacity & its size.
5. We can see the “Estimated cost”.
6. We can enable auto-scaling by clicking the check-box of Read Capacity & Write Capacity, then we can't provision any RCU or WCU.

- “Target utilization is used to set how much % of the RCU/ WCU to be used all the time. We can even set the Minimum & Maximum provisioned units.

Partition Internal

- Data is divided in partitions, it is divided internally into different partitions then distributed.
- Partitions keys go through a hashing algorithm to know to which partition they go to.
- To compute the number of partitions:
 - By capacity: $(\text{TOTAL RCU}/3000) + (\text{Total WCU}/1000)$
 - By size: $(\text{Total size}/10 \text{ GB})$
 - Total partitions = $\text{Ceiling}(\text{Max}(\text{Capacity}, \text{Size}))$
- WCU & RCU are evenly spread between partitions.

Throttling

- In simple terms, throttling means intentionally slowing or speeding of an Internet Service by Internet Service Provider.
- If we exceed our RCU or WCU, we get ProvisionedThroughputExceededExceptions.
- Reasons:
 - Hot keys: one partition key is read too many times. E.g.: we are selling some stuff on a store & this is a very popular item.
 - Hot partitions.
 - Very large items: Remember RCU & WCU depends on size of item.
- Solutions:
 - Exponential backoff when exception is encountered.
 - Distribute the partition key as much as possible.
 - If it is an RCU issue, we can use DynamoDB Accelerator.

API’s

Writing Data

- PutItem - write Data to DynamoDB (Create or fully replace). It consumes WCU since perform write operation.

- UpdateItem – Update Data in DyanmoDB (Partial updates of attributes). It only updates the attribute we indicate. In UpdateItem, we can use Atomic Counters & increase them if we wanted to count. For E.g.: Number of page hits.
 - Conditional writes: since we are in distributed system & different applications may be accessing the same row at the same time, we can write a condition that “write/ update is only valid if the conditions are accepted otherwise reject.
 - E.g.: if 2 application run at the same time, to same row & one of them write before the other one, other one will get its write denied since it has specified condition. This helps in concurrent access to items.
 - No performance impact.
- So, Conditional Write is basically to deal with concurrency.

Deleting Data

- DeleteItem
 - It'll delete only individual row or item.
 - Ability to perform conditional delete.
- DeleteTable
 - Deletes entre table & all it items.
 - Much quicker deletion than calling DeleteItem on all items. Even it is cost-efficient to use DeleteTable rather than DeleteItem.

Batching Writes

It is always efficient within network.

- We can do up to 25PutItem or DeleteItem In one call.
- There's no UpdateItem for Batching Writes.
- Limit of 16MB of data written or 400KB per item
- Batching allows us to save latency since we're making one call to DynamoDB instead of 25. Hence it reduces number of API calls done against DynamoDB.
- Operations are done in parallel for better efficiency.

- It is possible for a part of batch to fail, so we can retry the failed items using exponential backoff algorithm.

Reading Data

- GetItem – Read based on primary key.
 - Primary key is either a Hash or Hash-Range. It can be partition key alone or a combination of partition key & sort key.
 - We get “Eventually consistent Read” by default.
 - We’ve an option to use Strongly Consistent Reads (It takes more RCU or longer response time).
 - If we want to get only certain attributes in the result, ProjectionExpression can be used. It helps to retrieve particular items of the DynamoDB which helps to save our Network bandwidth.
- BatchGetItem
 - Up to 100 items, or
 - Up to 16MB of data.
 - The items are retrieved in parallel to minimize latency.
 - If we combine BatchGetItem, then we can get the batch of only particular items which helps to get only particular attribute.

Querying Data

- Query returns item based on:
 - Partition key value (must be = operator -> (an equal operator)).
 - Sort key value (=, <, <=, >, >=, Between, Begin) & this is optional in our query.
 - Further, we can use FilterExpression to further filter (client side filtering).
- Returns:
 - Up to 1MB of data.
 - Or number of items specified in the limits.
- Able to do pagination on the result.
- We can query a table, secondary index or a global secondary index.

Scanning Data

- It is the inefficient way of querying DynamoDB.
- It scans the entire table & then it filters out data.
- By default it returns up to 1MB of data & we need to use pagination to keep on reading.
- Since it reads an entire table so it consumes a lot of RCU.
- To limit the size of the scan, we may limit or reduce the size of the result & pause from now & then.
- For faster performance on scans, we can use parallel scans.
 - Multiple scans scan multiple partition at the same time.
 - Increases the RCU consumed & throughput at a large scale.
 - We can limit the parallel scans in terms on MB or number of rows returned.
- We can use combination of ProjectionExpression + FilterExpression (No change to RCU).

LAB

1. Go to DynamoDB under Services.
2. Click on the table already created (if not create a table).
3. Under ‘Items’ tab, click on “Create Item”.
4. Add items to list & hit on Save.
5. Now, if we update the row, we either get PutItem or UpdateItem.
6. Click on the particular row, under ‘Actions’, go to ‘Duplicate’, it calls PutItem API.
7. Again, go to ‘Action’, click on ‘Delete’, it calls DeleteItem API.
8. Clicking on any of the Primary key performs GetItem call. It returns the content for us.
9. To get the whole table contents, many different rows, we’ve to do ‘Scan’ from the console under ‘Items’ tab.
10. To get the item, we can also perform Query. Under the console, go to ‘Items’ tab & hit on ‘Scan’ dropdown & click on ‘Query’.
11. Specify the query parameters (partition key which needs equal operator & sort key which contains multiple comparison parameters).
12. Sort the result in either ascending or descending order.

13. Select the attributes if we want to filter it out.

14. Click on “Start search”.

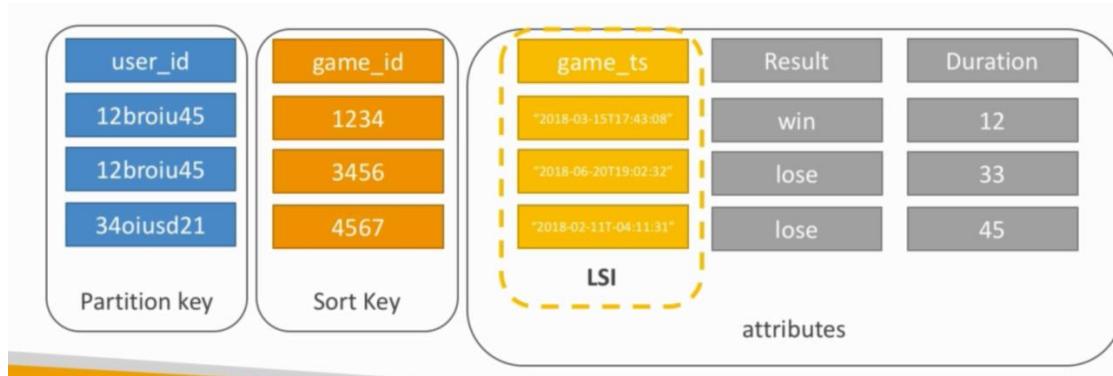
Note: On scanning, we don't get the above options.

15. If we select all the rows & click on ‘Actions’ & click on ‘Delete’, Batch Delete operation is performed.

16. We can also delete entire table, by clicking on “Delete Table”.

Local Secondary Index

- It is an alternate range key for our table. It is local to the Hash key or the Partition key.
- We can have up to 5 local secondary index per table.
- The sort key consist of exactly one scalar attribute.
- The attribute we choose must be a Scalar string, Number or Binary.
- The LSI must be defined at the time of creation.

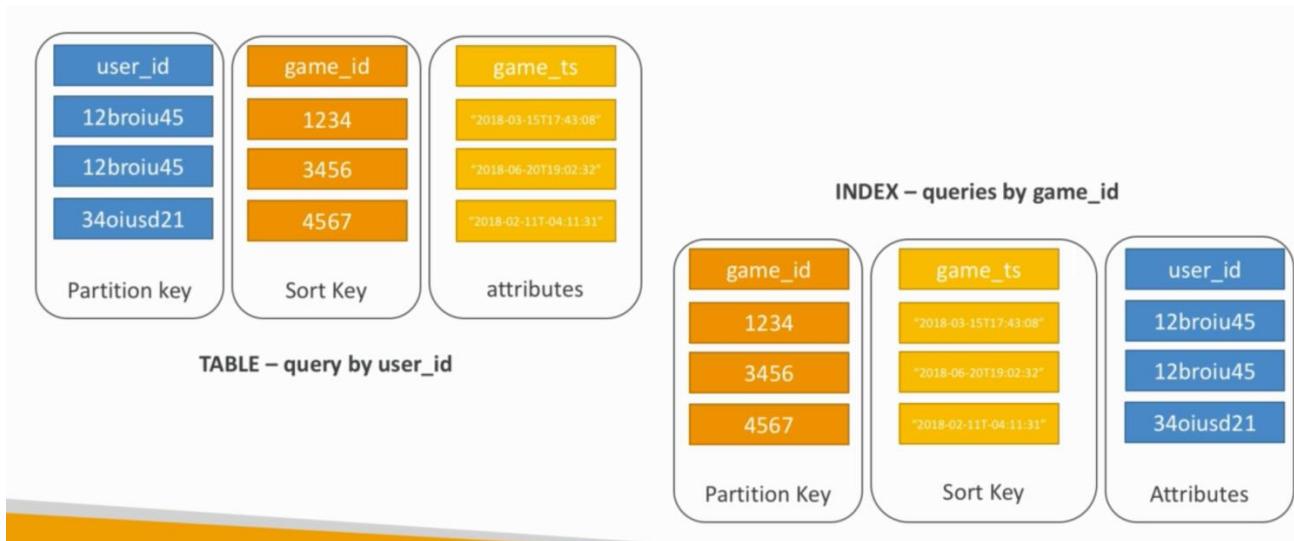


- In this table, we can query using user_id, game_id, but we can't really query by time stamp. Time stamp we've to retrieve during scan and then filter it out. If we want to have an efficient query by timestamp, we can have a LSI on timestamp column. And now, our data will be ordered by our user_id& timestamp.
- It is called Local, because it is local to the partition key.

Global Secondary Index

- It is used to speed up queries on non-key attributes

- GSI = partition key + optional sort key. Hence it allows to define a new index.
- This index is a new table & we can project attributes on it.
 - The partition key & sort key of the original table are always projected (KEYS_ONLY).
 - Can specify extra attributes to the project (INCLUDE parameter).
 - Can use all the attributes from the main table (ALL).
- Must define RCU & WCU for the index.
- Possibility to add / modify GSI.



- E.g.: In the above table, we've the user_id, game_id & game timestamp. This table allows us to do very efficient queries. We can do query by user_id & game_id.

But say, we want to query by game_id. Now the game_id needs to be the partition key now. For this, we need to define an index. In that index, the GSI will have the queries by game_id, so the game_id will be the partition keys.

In terms of Sort key, we can use the game timestamp. The Attributes will be data from the table, which in our case is user_id. We can see, the columns are flipped, and the orders are different. By defining Global Secondary indexes we're able to perform more efficient types of queries.

Note: Local Secondary Index is local to the partition key whereas GSI is a whole new different table.

Indexes & Throttling

- GSI
 - If the writes are throttled on GSI, then the main table will be throttled.
 - It is even true if there's enough WCU on the main table.
 - So, we need to choose our GSI partition key carefully
 - Assign our WCU capacity carefully.
 - If we don't follow the above steps carefully, throttled write on our GSI will be a throttled write on main table.
- LSI
 - Since LSI uses the WCU & RCU of the main table.
 - There's no special throttling consideration. It means if LSI is throttled, our main table is throttled as well.

LAB

Table name*	UserGames	
Primary key*	Partition key	
	user_id	String  
	<input checked="" type="checkbox"/> Add sort key	
	game_id	Number  

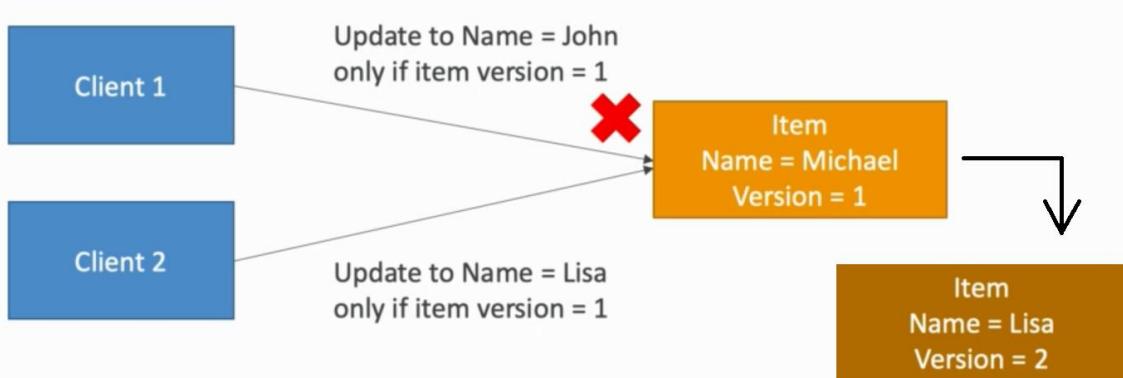


1. Go to DynamoDB under Services.
2. Create a table with Primary key as Partition key & Sort key.
3. For “Table settings”, un-check the “Use default settings”.
4. Under “Secondary indexes”, click on “Add index”. We can add both LSI & GSI.
5. Check the checkbox of “Create as Local Secondary Index” since the name of the partition key of Secondary index is same as the Table’s partitionkey. If it is not same, then the option “Create as Local Secondary Index” will no longer be available.
6. Click on LSI.
7. Uncheck the check boxes of Read Capacity & Write capacity & set it to 1-1.
8. Hit on ‘Create’.
9. Let’s insert a little bit of data, then we’ll define a GSI.
10. Enter the data after clicking on “Create item” & then hit “Save”.
11. Navigate to ‘Indexes’ tab.
12. We can see the name of the index is combination.
13. Go back to ‘Items’ tab.
14. On Scan/Query option, we can select to search from either [Table] or [Index].
15. When we select [Index], we can see that we can see that we can search by “Sort key” equals game_ts.
16. Go to ‘Index’ tab & click on the index created in step5.

17. We can see the ‘Type’ to be LSI.
18. Now, click on “Create index”, and now we can create GSI type of index.
19. Now, we need to input the Read & Write Capacity in advance.

Concurrency

- As we've seen, DynamoDB has a feature of ‘Conditional Update/ Delete’.
- It means that we can ensure that when we Delete or Update the item, it hasn't changed. This is helpful since we access stuff over the network, it is possible that multiple client might be accessing the same object at the same time and we want to make sure that we delete something only if a certain condition is validated.
- It makes DynamoDB an optimistic locking/ concurrency Database.
- E.g.:

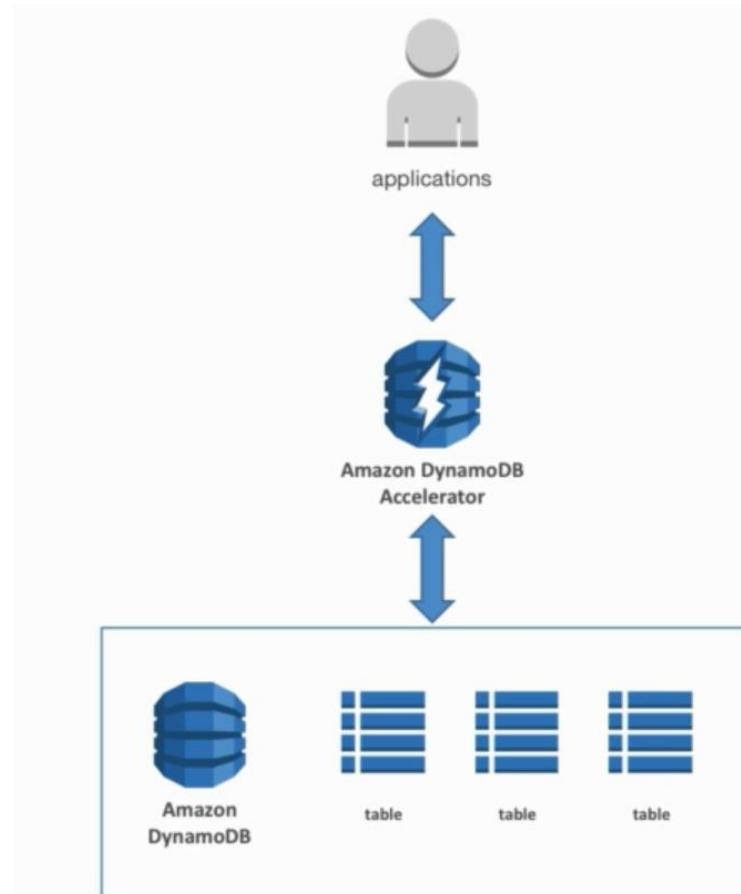


There's an item in our DynamoDB table & name is Michael and the version is 1. So, there're 2 attributes of our item. It could be different apps or web clients & they want to access the same items at the same time.

They want to update the name. Client 1 wants to update the name to 'John' only if the item version is 1. Client 2 wants to update the name to Lisa only if the version is 1. It may happen that when item gets updated by either of the client. So let's suppose the client 2 updates the 'Name' & hence the version gets updated to 2. Now, if client 1 wants to update, it'll be failed since the item version is 2.

DAX

- DAX = DynamoDB accelerator.
- It is a seamless cache for DynamoDB, after enabling it we don't need to re-write any of our application
- Writes go through DAX to DynamoDB
- It provides us microsecond latency for cached reads & queries.
- It even solves the hot key problem (too many reads). Let's suppose we've a popular item entering a big sale & the item is being queried a lot, so we get too many reads on that item, we may get Hot Key problem when we exceed the throughput. Then we put DAX in front of our DynamoDB cluster & now the reads are done from the cache & hence the hot key problem is solved.
- By default, the items live for 5 minutes in the cache.
- We can have 10 nodes in our cache cluster. So, we can scale a lot.
- It is Multi AZ (3 nodes are minimum required for production to enhance that availability). So, we can enable multi nodes in multiple AZ's.
- DAX is secure, we get encryption at rest using KMS, VPC, IAM, CloudTrail, etc.)
- E.g.:



Here's our DynamoDB & we've 3 tables in our DynamoDB tables. So, our applications instead of talking straight to DynamoDB, they're going to interact with the Amazon DynamoDB accelerator, so no code change, just straight interaction. The accelerator interacts with our DynamoDB & it is smart enough whatever & whenever it is needed.

- Hence, it improves our application at minimal cost.

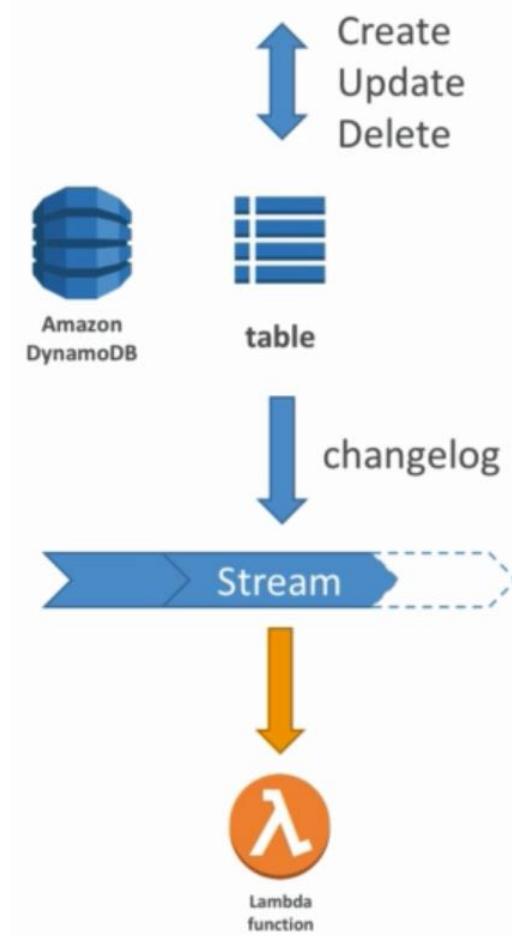
LAB

1. Go to DynamoDB under Services.
2. Go to DAX on the left hand side of our console.
3. Click on 'Dashboard' & we can see that we've no clusters running.
4. Click on "Create cluster".
5. Give it a name.
6. We can specify the Node type according to its size.
7. Select the cluster size from 1-10 nodes.
8. Encryption can be enabled for DAX cluster.

9. We need IAM access to DAX for access to DynamoDB tables.
10. We need to select the subnet group in which our subnet group can be created under VPC.
11. We can assign some security groups as well.
12. For Cluster settings, we can use default settings which gives a TTL of 5 minutes & it does not give any preferences for availability zones or maintenance windows & notifications will be disabled.
13. Hit 'Create'.

Streams

- Any changes in DynamoDB such as Create, Update or Delete can end up in DynamoDB stream.
- E.g.:



Let's suppose we perform any of the operations such as update, create in a DynamoDB table and as a return, the change log of these updates, to create, update & delete will end up in a stream. So we can get a stream of updates as a change log.

- These streams can be further read by AWS Lambda. Then we can do:
 - React to changes in real time (Sending welcome email to users).
 - Perform analytics
 - Create derived tables/ views
 - Insert data to ElasticSearch
- We can implement cross region replications using Streams
- Stream in DynamoDB has just 24 hours of data retention.

LAB

1. Go to DynamoDB under Services.
2. Go to Tables.
3. Under Overview Tab, click on 'Manage Stream' in "Stream Details" sub-section.
4. It has 4 types:
 - a. Keys only - only the key attributes of the modified item
 - b. New image - the entire item, as it appears after it was modified
 - c. Old image - the entire item, as it appeared before it was modified
 - d. New and old images - both the new and the old images of the item (we need to have the data of both & after modification).
5. For now, New & old images.
6. Hit 'Enable'. We can see that the stream is enabled & an ARN has been provided to us.
7. This stream can be integrated with AWS Lambda.
8. Go to triggers.
9. Create a trigger & then choose "New function". Whenever the stream record is written it'll trigger Lambda function.
10. Give the function a name.
11. Choose a role.
12. Choose the DynamoDB table.

- 13.Specify the batch size which indicates the largest number of records that'll be read from out table stream at once.
- 14.'Starting position' specifies the position in the stream to start reading from. For now, choose 'Latest'.
- 15.Enable the trigger.
- 16.Click on 'Create'. But we can see, it throws an error "Your Lambda function "dynamoDB-lambda-function" was successfully created, but an error occurred when creating the trigger: Cannot access stream. Please ensure the role can perform the GetRecords, GetShardIterator, DescribeStream, and ListStreams Actions on your stream in IAM.
- 17.So, go to IAM.
- 18.Go to role & create a Role.
- 19.Choose the 'AWSLambdaDynamoDBExecution role'.
- 20.Under review, specify the role name.
- 21.Hit on "Create role".
- 22.Go back to Lambda function page & refresh the page.
- 23.Under Basic setting, click on the new role created above.
- 24.To add a trigger, go to Designer & select "Add trigger" & select "DynamoDB"
- 25.Fill in the details of table, enter the batch size & the starting window.
- 26.Click on 'Add'.
- 27.Hit on 'Save'. We can see, we don't get errors any more.
- 28.Go to DynamoDB & select the table specified in step25.
- 29.Go to 'Triggers' tab.
- 30.We can see a lambda function created.
- 31.Go back to item, create 3 items.
- 32.Go to CloudWatch under Services.
- 33.Go to 'Logs' and then navigate to 'Log groups'.
- 34.We can see the log of our function created, click on it.
- 35.Under Log Streams we can see that all our logs (including those of items creation) are present.
- 36.We can see the StreamViewType, it is logged as "NEW_AND_OLD_IMAGES" as was defined in step 5.
- 37.On performing CRUD operation on the table, the logs are updated & can be viewed.

TTL

- Items automatically gets deleted after an expiry date/time.
- It is provided at zero extra cost & it does not require any extra RCU / WCU.
- It is a periodic task & DynamoDB does it periodically.
- TTL helps in reducing the storage & managing the table size over time. It may happen that we don't need a row in DynamoDB because it may have expired.
- It helps to sticks to regulatory norms like if we want to keep Personal info data for a period of only 7 days.
- TTL is defined per row.
- Normally the deletion happens just after the specified TTL time but it deletes within 48 hours of expiration.
- Deleted items due to TTL are also deleted in GSI / LSI.
- DynamoDB Streams can help recover expired items.

LAB

1. Go to DynamoDB under Services.
2. Create a table with just a Partition key as a Primary key.
3. Insert items into the table.
4. Add or name a column which defines expiry time. Note that data type should be a 'Number' for this column.
5. Go to the browser & search for "Epoch Convertor".
6. Click on the first URL. It allows us to take a timestamp & returns a number which represents an epoch time which is a number of milliseconds since the year 1970.
7. For now, select the timestamp for 5 minutes later and insert the value in the column created in step3.
8. Likewise, add another item & hit save.
9. Go to 'Overview' tab.
10. Go to "Time to live attribute" & click on "Manage TTL".
11. Enter the name of the column which is going to look for the epoch value.
12. Check the check-box if we want to get the streams that're deleted.
13. We can "Run preview" to check which items are going to expire.

CLI

1. **--projection-expression:** Specifies a list of attributes (or a subset of it) that we want to retrieve from the table.
2. **--filter-expression:** To filter the result.
3. Genera CLI pagination option that includes DynamoDB / S3
 - a. Optimization:
 - i. **--page-size:** Full dataset is received but each API call will request less data to avoid less timeouts.
 - b. Pagination: Used to specify pages.
 - i. **--max-items:** maximum number of results returned by the CLI. It returns NextToken at the end of the command & we pass that token to next command we run to keep on reading the next page, hence named pagination.
 - ii. **--starting-token:** Specify the last received NextToken to keep on reading.

LAB

1. Launch Git.
2. **For projection expression,** run: `aws dynamodb scan --table-name UserGames --projection-expression "game_id, game_ts" --region ap-south-1`
3. **For filter expression:** `aws dynamodb scan --table-name UserGames --filter-expression "user_id = :u" --expression-attribute-values '{':":u": {"S": "123"} }' --region ap-south-1`
 Here,
 “:u” symbolizes the expression which needs to be defined at the end.
 “S” symbolizes String data type (since user_id is of string data type).
4. To run a **scan**, `aws dynamodb scan --table-name UserGames --region ap-south-1`
5. When we want to **do 3 API calls:** This command where we specify the page size to be one and by having the page size being one, we're saying the max number of results, I want from each API call is going to be one. Though we'll get the full data set, it'll run n API sets: `aws dynamodb scan --`

```
table-name UserGames --region ap-south-1 --page-size 1.
```

6. **To limit the number of calls per CLI:** aws dynamodb scan --table-name UserGames --region ap-south-1 --max-items 1
7. From the above command, we get the value of NextToken, which'll be used as a part of the starting token argument i.e., **to retrieve the value of the next n rows** (which is selected under --max-items) in the table: aws dynamodb scan --table-name UserGames --region ap-south-1 --max-items 1 --starting-token eyJFeGNsdXNpdmVTdGFydEtIeSI6IG51bGwsICJib3RvX3RydW5jYXR1X2Ftb3VudCI6IDJ9.
8. We can see, after running the above command, we do not receive any NextToken, it's so because we've reached the end of the table and there's no pages to read.

Transactions

- Transaction = Ability to Create / Update / Delete multiple rows in different tables at the same time.
- It's very similar to what a transaction is in RDBMS such as PostGreSQL.
- It's called transaction because it's all or nothing type of operation.
- Write modes are: Standard, Transactional.
- Read modes are: Eventual Consistency, Strong Consistency, and Transactional.
- It consumes the twice of WCU / RCU.
- E.g.:

Account_id	balance	Last_transaction_ts
Acct_21	230	1562503085
Acct_45	120	1562503085

Let's suppose we've an AccountBalance table. It has account ID, Balance, Last_transaction_ts.

Transaction_id	Transaction_time	From_account_id	To_account_id	value
Tx_12345	1561483349	Acct_45	Acct_21	45
Tx_23456	1562503085	Acct_21	Acct_45	100

Let's have another transaction table such as BankTransaction Table which represents the transaction between the accounts.

If we update BankTransaction Table, we'll have to update the AccountBalancetable at the very same time. So we need either both things to happen, to both the tables or none of it.

- A transaction is a write to both tables, or none.

Session State

- DynamoDB can be used to store data but it can also be used to store the session state as a cache. It is used by the applications to store the sessions on demand and share the user login across all the backend web application.
- vs. ElastiCache
 - ElastiCache is in memory, but DynamoDB is serverless.
 - Both are key/value stores.
- vs. EFS: It is another way in which session state can be stored. The session state is stored on the Disk and we need to share the disk across multiple EC2 instances.
 - EFS must be attached to EC2 instance as a network drive. It cannot be used with Lambda.
- vs. EBS & Instance store
 - EBS & Instance store can be used for local caching, not shared caching coz EC2 instance store & EBS volumes are attached only to one EC2 instance.
- vs. S3
 - S3 has higher latency, it is meant for big files not for smaller objects.

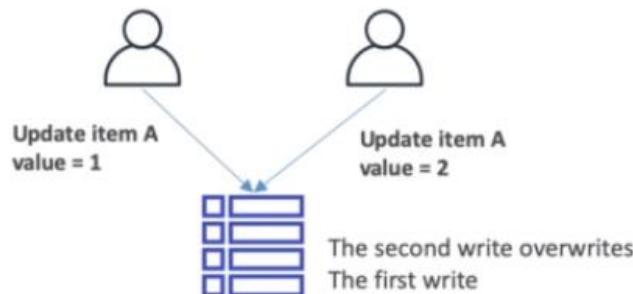
Partition strategies

Partition Key	Sort_Key	Attributes
CandidateID + RandomSuffix	Vote_date	Voter_id
Candidate_A-1	2016-05-17 01.36.45	235343
Candidate_A-1	2016-05-18 01.36.30	232312
Candidate_A-2	2016-06-15 01.36.20	098432
Candidate_B-1	2016-07-1 01.36.15	340983

- Imagine, we've a voting application with 2 candidates, A & B.
- If we use partition key of candidate_id, we'll run into partition issues as we've only 2 partitions.
- So the solution will be to add a suffix (usually it is a random suffix, but sometimes it is a calculated suffix) into our partition key and therefore we can scale our writes across many shards.
- Another example we can take is of pets, like we've lots of records of dogs. If we query about dogs, all the record will hit the same partition and cause the request to be throttled. To design the table efficiently, we can change the partition key to be the pet's breed which will balance the different SSD rather putting load entirely on one SSD.

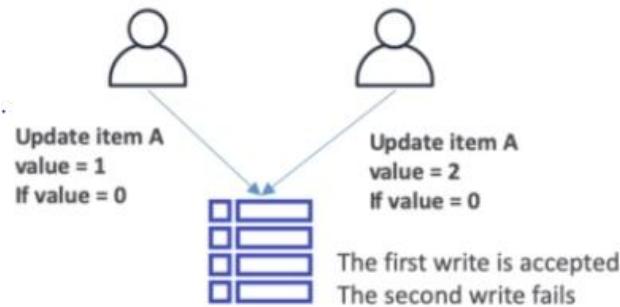
Write Types

- Concurrent write



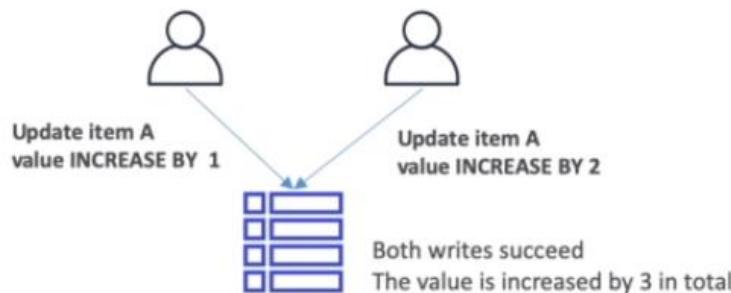
- We've a DynamoDB table and 2 users try to write at the same time. First user want to update item A with value equals one. Meanwhile, second user want to update it with value two. The first write is overwritten by the person whoever the second write. It is known as concurrency.

- Conditional write



- We'll overcome it using conditional write. So know, First user updates value = 1 only if value = 0. And Second user also says, that he'll update only if the value equals 0.
- In this condition first write will succeed but second fails coz when the first write is successful, the value equals one which makes the condition false for the second user to write it.

- Atomic write



- We are writing to DyanmoDB table and update the item A by increasing the value by 1.
- The other user says he want to increase item's A value by 2.
- In this case both writes succeeds and value is increased by 3.

- Batch writes

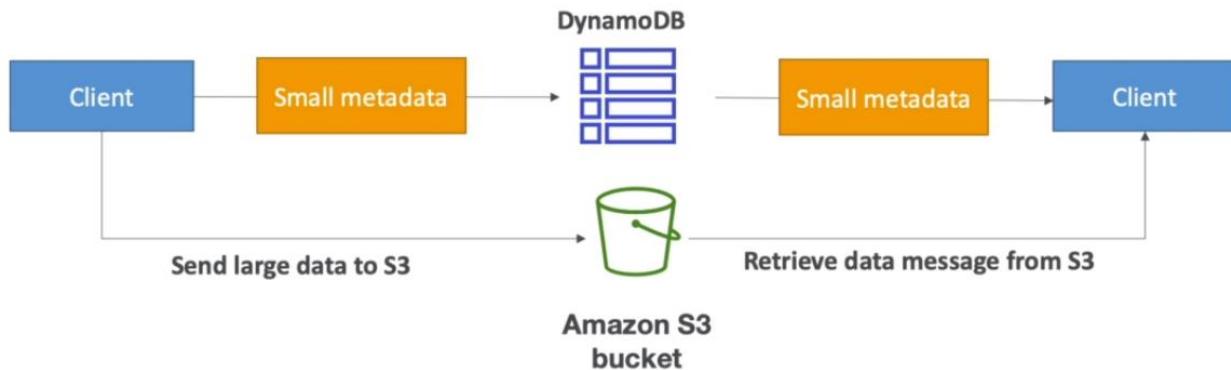


- It is used to update multiple items at a time.

DynamoDB patterns with S3

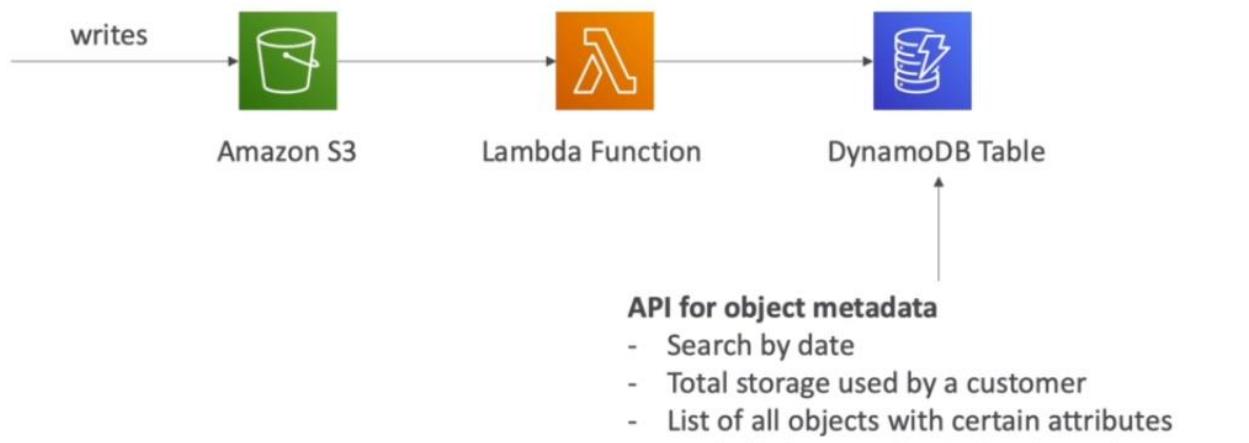
- Since DynamoDB is expensive & maximum object size is 400KB. So it is not recommended to store large items on DynamoDB.

Large Objects Pattern



- But what if we want to store large objects on DynamoDB. If clients want to write on DynamoDB. So, he'll upload the large data to S3. Then metadata is inserted into DynamoDB.
- The metadata information consists of object key, ID, and location of the object in S3 bucket.
- The client who wants to retrieve it, will retrieve the metadata from DynamoDB and get to know from where to pick the files from S3.

Indexing objects with S3

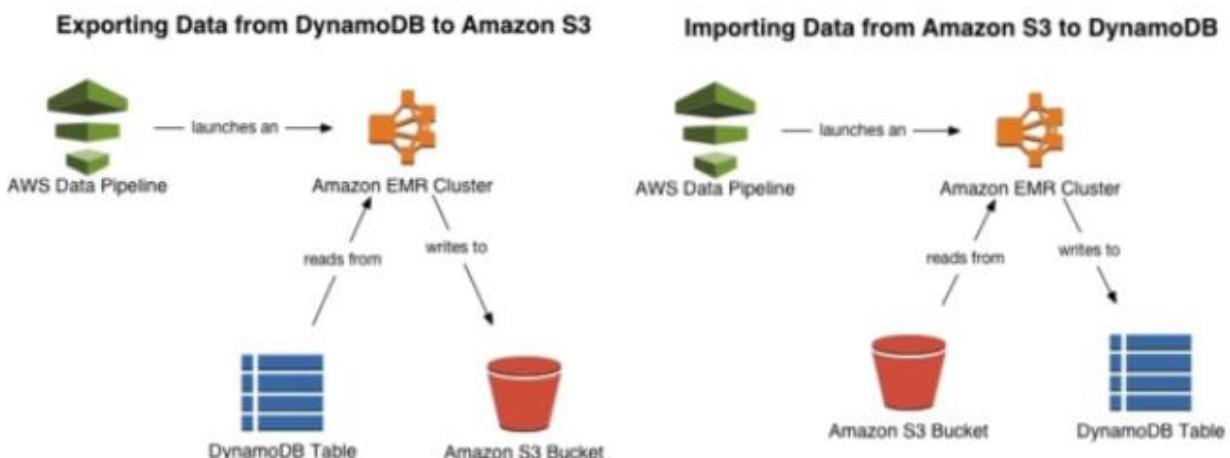


- When we write to Amazon S3 buckets, we cannot search if we do not have the key in advance.

- Now, when we start write in Amazon S3 bucket, but now we're going to have S3 event notification triggering a Lambda function.
- The Lambda function takes all metadata from S3 file & write it into DynamoDB table coz DynamoDB is much better way to store data for indexing, querying & searching.
- On top of DynamoDB table, we create an API where we can search by date, total storage used by customer, etc.
- Using this API, we're able to go back to S3 and find the files needed.

Operations

- Table Cleanup: To delete the entire table, we can:
 - Scan the entire table & delete the item one by one. It is very slow & expensive since it consumes lots of WCU & RCU.
 - We can even drop the table and recreate it. It is fast cheap & efficient.
- Copying DynamoDB table:



- We can use AWS DataPipeline (it uses EMR, a big data tool). It takes the DynamoDB table, put it into S3. After, it takes it back into S3 and finally put it back into DynamoDB table.
- Also, we can copy by creating a backup & restore the backup into a new table (it can take some time but an efficient technique).
- The other way us to scan our first table and then write the data into a new table and for this, we need to write our own code. It is the least efficient technique.

Security & Other features

- Security
 - VPC endpoints available to access DynamoDB without internet.
 - Access fully controlled by IAM.
 - We get encryption at rest using KMS.
 - Encryption in transit using SSL / TLS.
- Backup and Restore feature available.
 - We can do point in time restore like RDS.
 - Enabling this feature has no performance impact.
- Global Tables
 - We've the ability to have a global table which is going to be available for DynamoDB if our application is in many different regions.
 - DynamoDB is multi-region, fully replicated & high performance and this again, this relies on strings.
 - Amazon DNS can be used to migrate to DynamoDB (from Mongo, Oracle, MySQL, S3, etc.)
 - We can launch a local DynamoDB on our computer for development purpose.

18. Serverless API gateway & Cognito

API gateway enables us to spread our application, expose it as a REST API. API Gateway is a way to build, deploy & manage API's. API's are interfaces that can be exposed to other people within our company or in the outside world.



As we've seen, Lambda as a compute source which can automatically scale, it is serverless & runs function as a service. Also, we can use DynamoDB as a

serverless database, provision tables, & interact directly with AWS Lambda to perform operations such as Create, Read, Update, & Delete.

So, we've a computing side & we've a serverless Database side. If the Client needs to interface with Lambda & DynamoDB & Client is our application. Amazon API Gateway takes the client's request as a REST API & then translate & pass them on to the Lambda function. The API gateway helps us to give control what the client can or cannot do. We can defer the business logic & how these request should be handled to AWS Lambda.

API Gateway

- It is going to have integration with AWS Lambda.
- We don't need to manage infrastructure, it gives us a serverless REST API.
- We can handle versioning (v1, v2, etc.)
- We can handle different versions (Dev, Test, Prod, etc.).
- It has tight integration with security (Authentication & Authorization).
- We can create API Keys, handle request throttling.
- It has a tight & neat integration with Swagger / Open API. It allows us to import some files & define quickly our API's and we can even export them as SDK.
- It helps us transform & validate responses.
- It helps us generate SDK & API specifications.
- It is embedded with caching layer so we can cache our responses and limit the load that happens on our Lambda function.

Integration

- We've to separate our API Gateway inside & outside our VPC.
- Outside our VPC, we can run:
 - AWS Lambda function (most popular / powerful combination).
 - EC2 instances
 - Load Balancers
 - Any AWS Service
 - Proxy the request from API Gateway all the way to HTTP endpoints.
- Inside VPC, we can enable:

- AWS Lambda in our VPC
- EC@ endpoints in our VPC.

LAB - I

1. Switch to N. Virginia region, it has most of the features.
2. Go to API Gateway under Services.
3. Under WebSocket API, click on Build.
4. Choose REST & then select “New API”.
5. Since, we want our API to be available from the outside, so we’ll choose a ‘Regional’ Endpoint type.
6. Click on ‘Create’.
7. The Blue & Orange dot on the console represent hints.
8. The forward slash ‘/’ on top of the list represents, it is the root of the API.
9. Click on ‘Actions’, choose ‘Create Method’.
10. Choose method to be ‘GET’ for now.
11. For now select integration type to be “Lambda function”.
12. Select the region to be “us-east-1”, & create a Lambda function.
13. Test the function.
14. Go back to API service, step 12.
15. Enter the Lambda function name created in step 12.
16. Hit ‘Save’.
17. Click ‘Ok’ to invoke permission on Lambda function.
18. We can view the whole flow of our GET request. Client invokes our function, then we can define any type of authorization we want to. Our request goes to Lambda. Then it goes to Lambda API, and we get a response and finally we can map it to method response.
19. Click on ‘Test’ & again click on ‘Test’ and we get the same result as we get after executing the ‘Test’ in Lambda function.
20. Click on ‘Action’ & choose ‘Resources’.
21. Give your Resource a name and then we’ve a Resource path ‘/houses’.
22. Click on “Create Resource”.
23. Within our API, there we can see /house resource.
24. Underneath ‘houses’ create a new GET method.
25. Create a new Test Lambda function & specify it under API method created in above step.

26. Test the function. The response is same as we get from Lambda console.
27. Note: In one API, we've integrated 2 Lambda function. Resources are the path in our API and Methods are type of technique to retrieve data from Servers.

Deployment Stages

- When we make changes in the API Gateway, it doesn't mean they're effective right away.
- We need to make a deployment for them to be in effect.
- Changes are deployed to 'Stages'. We can have as many as we want.
- We can name the stages as per our naming conventions (dev, test, prod).
- Each stage has its own configuration parameter.
- Stages can be rolled back as a history of deployment is kept.

Stage Variables

- We have stages because we have stage variables.
- Stage variables are like environment variables for API gateway. Using these stage variables, we're going to be able to change them & they'll drive changes into configuration values.
- They can be used in:
 - Lambda ARN
 - HTTP endpoint
 - Parameter mapping templates.
- Use cases:
 - We can configure HTTP endpoints that our stages talk to (Dev, test, prod). Like if we have a Dev stage then we can talk to Dev HTTP API.
 - We can pass parameters to AWS Lambda function through mapping templates as the Lambda function, whether it is in Dev, Test or Prod.
 - Stage variables are passed to the "context" object in AWS Lambda.

Stage Variables & Lambda Aliases

- Lambda aliases point to our Lambda Versions and so we can use stage variables to point to specific Lambda alias.

- Our API gateway automatically invoke the right Lambda function.
- E.g.:



We've Lambda function & it has 7 versions. 6 version has been snapshotted & we're working on the current version. We've created Dev, test & prod aliases where 'Dev' points to latest, 'Test' points to version 5 & 'Prod' points to version 2.

We want to expose these aliases to API so we'll have an API Gateway. We'll have 3 stages & in parenthesis we've the variables. So, the Dev stage will have a variable called alias. After configuring the API Gateway correctly, we can have Dev stage pointing to Dev alias in AWS Lambda. Similarly, we can have the Test stage, the alias equals Test pointing to Test Alias Lambda. So on, for prod. Using these stages & stage variables we're able to modularize how we want our stages to be & in which Lambda function to invoke.

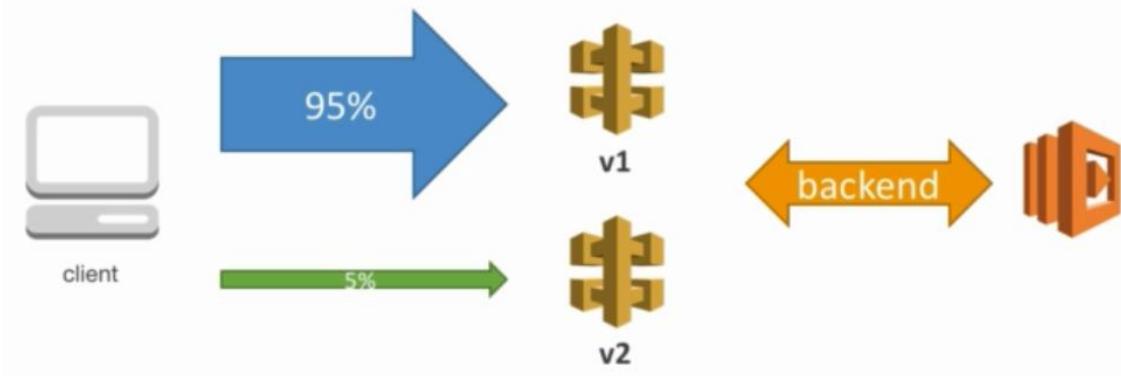
- Environment variables can change the code behavior.
- Stage variable can change what the API Gateway does.

Canary Development

When we deploy an API, we want to test a little bit the improvements of the API before going all in. This is called Canary development.

- Canary stage can be used at any stage but mainly used when we go for production.
- We can choose the % of traffic the canary channel receives.

- E.g.:



We've v1, v2 of our API. They all talk to backend in different ways. A client directs 90% of the traffic to v1 API & only 5% of the traffic to canary channel.

Using this, we're able to just have 5% of the channel through.

- We're able to have logs & metrics separate for better monitoring.
- Using these logs & metrics, we can ensure that our API Gateway is not having any error responses and everything works great with our v2 API. When we're ready, we can increase the version of our API gateway & shift all the traffic to v2. Hence, we can test the 2 versions at the same time.
- We've the possibility to override the stage variables in the canary stage.
- This is a blue green deployment in AWS Lambda & API Gateway.

LAB – II

1. Go to API Gateway under Services.
2. Go to resources (continue with the one in previously created LAB).
3. Click on 'Actions', choose "Deploy API".
4. When we deploy an API, we need to choose a stage. Click on "New stage".
5. Give a name for "Stage Name", "Stage Description", and "Deployment description".
6. Click on 'Deploy'.
7. In the console, we can see the whole 'Resources' is under Stage under the Stage name created in stage 5.
8. On clicking on the Stage Name, we can see a URL which invokes our stage.
9. On clicking on the URL, we get a response from our Lambda function.

10. On clicking on the next resource's method, we get a response from other Lambda function.
11. Under settings tab, we can "Enable API Cache", we can throttle some methods.
12. Under Logs / Tracing, we can enable CloudWatch logs.
13. Under Stage variables tab, we can create stage variables.
14. We can generate SDK for Android, JavaScript, IOS, etc. under SDK generation tab.
15. We can export our API as Swagger, Open API, Postman, etc. under Export tab.
16. We can view the deployment history under "Deployment History" tab.
17. Document History helps us to provide with documentation.
18. Canary is used for testing new API deployments and/or changes to stage variable.
19. We can deploy one more time & we would create a new stage. Go to Resources, click on "Actions", click on "Deploy API".
20. Give the deployment stage a name, description. Etc. and click on 'Deploy'.
21. Now, we've 2 stages in our production API & it can be invoked using the URL mentioned under 'Stages' and navigating to our new stage. It gives the same URL as in step 7 because we've deployed using the same API.

Mapping Templates

- Mapping templates can be used to modify the request/responses.
- We can
 - Rename parameters
 - Modify body content
 - Add headers
 - Map JSON to XML for sending data back to the client.
 - For this, we use a language called VLT: it can be used to write for loops, if statement, etc.
 - Filter output result & remove unnecessary data.

LAB

Let's suppose we don't want JSON response on clicking on our stage's URL. Instead we want an XML response. One way of doing it is going to Lambda function, & where we wrote the code, we can write a XML string in response instead of a JSON. Another way of doing it is mappings.

1. Go to API gateway under services.
2. Navigate to Resources. Click on the Resource's Method's name we want to modify.
3. Click on its Integration Response.
4. Click on the arrow under the table.
5. Go to "Mapping Templates".
6. Click on Application/JSON. Under "Generate template", choose 'Empty'. The 'Empty' model sets the base for us to change the outputs.
7. Here, 'inputRoot' is the variable what we get the response from client.
8. Hence, we'll write:

```
<xml>
  <response>
    <body>$inputRoot.body</body>
    <statusCode>$inputRoot.statusCode</statusCode>
  </response>
</xml>
```

9. Click on 'Save' & save the "Mapping Templates" again.
10. Go to 'Resource' page and navigate to the method for which we changed the response type.
11. Click on 'Test' & hit on 'Test' again.
12. If we click on the URL, we can't see any changes, since we've not deployed the changes.
13. Go to 'Actions'.
14. Click on 'Deploy'.
15. Choose the stage where we want to deploy our change.
16. Now click on the same URL for which we changed the response type & we can see the response has been modified to XML type.
17. Note: We can even change the response type in "Integration request" under 'Resources'.

Swagger / Open API Spec

Swagger is the new name of Open API.

- It is a new way of defining REST API & we can use API definition as code.
- To API gateway, we can import an existing swagger / Open API 3.0 spec to API Gateway. It includes:
 - Methods
 - Method Request
 - Integration Request
 - Method Response
 - +AWS extensions for API Gateway & setup every single option.
- Can export current API as Swagger / Open API Spec.
- Swagger can be either in YAML or JSON.
- We can even generate SDK for our application using Swagger.

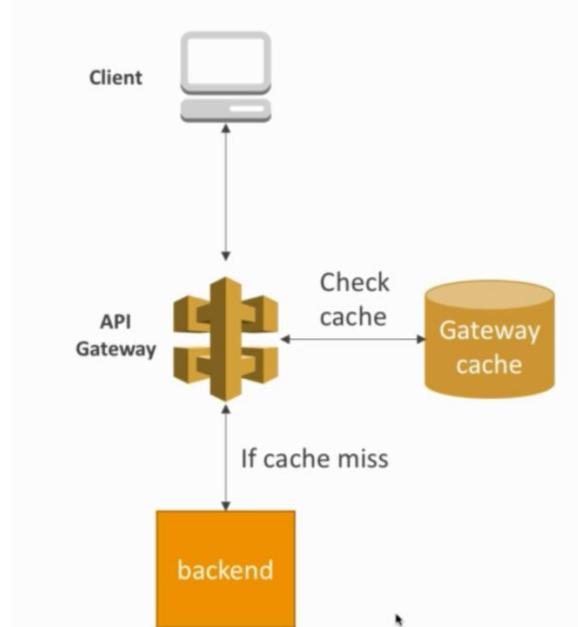
LAB

1. Go to API Gateway under services.
2. Go to Stages tab.
3. Go to any of the stage.
4. Navigate to the ‘Export’ tab.
5. Click on “Export as Swagger”.
6. On clicking on JSON/YAML, we can view the file which our API represents.
7. Click on “API” in the main console.
8. Hit on “Create API”.
9. Under “WebSocket”, click on ‘Build’.
10. Choose “REST API” & click on “Example API”.
11. Click on ‘Import’. We can see we’ve a new API called PetStore.
12. Under ‘Resource’ tab, we can see a lot of different endpoints. We’ve GET, POST methods. On clicking on either of the methods, we can view that we’ve dummy endpoints.
13. When we click on ‘Models’ which were created using the Swagger file. It helps with the documentation.
14. Go to ‘Resources’, click on ‘Action’ & choose “Deploy API”.
15. Create a new stage & enter the name of the stage, etc.
16. Click on ‘Deploy’.

- 17.Click on the URL generated, we are moved to the new page which says “Welcome to your Pet Store API”.
- 18.Under Resource, click on different method types, & we can see the response.
- 19.If we enter the pet ID in the URL “/pets/{pet ID}”, we get the nth part of the response.

Caching

- Caching reduces the number of calls made to the backend.
- It could be ElasticCache, DynamoDB tags, etc.
- E.g.:



Let's suppose our client is interacting with our API Gateway. When we get a GET request on API Gateway, the API Gateway is going to check its own cache, if it is enabled. If the cache has something in Cache, it is returned to the client. If the cache doesn't have the data we need then, we'll invoke our backend, get the result, send it back to our client and put it in cache.

Hence, by using cache, we reduce the number of calls made to backend & hence we reduce the latency.

- Default time TTL is 300 seconds & max is 3600 seconds.
- Caches are defined per stage, so we can have a Dev cache, Prod cache, and we can have the cache being bigger in Prod than in Dev.

- We can encrypt our cache, and we can have a cache size between 0.5GB to 237GB.
- Provide 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). This'll bypass the cache & will help in getting the straight from the backend.

LAB

1. Go to API Gateway under Services.
2. Go to Stages (PetStore) we lastly created.
3. Go to any of the stage like Dev.
4. Under ‘Settings’, we can enable cache by checking the check-box “Enable API Cache”.
5. Set the cache capacity.
6. We can encrypt the cache data by checking the “Encrypt cache data”.
7. We can set TTL for cache.
8. Per-key cache invalidation helps to set the validation with header.
9. Later, we can set how to handle unauthorized request if the request is unauthorized, we can ignore or fail the request.
10. We can even override it at a Resource level.

Monitoring

- CloudWatch Logs
 - We can enable CloudWatch logging at Stage level (with log level).
 - Can override settings on a per API basis. E.g.: We can enable at error logs, debug logs, info logs, etc.)
 - Log can contain information about the request / response body which can help in debugging.
- CloudWatch Metrics
 - Metrics are by stage.
 - We can enable detailed metrics.
- X-Ray integration

- We can enable tracing at Gateway level to get extra information about the requests made from API Gateway to Lambda function to DynamoDB (for e.g.)
- X-Ray API Gateway + AWS Lambda gives us the full picture about how the request goes through the entire system.

LAB

1. Go to API Gateway under Services.
2. Go to Stages.
3. Go to Logs/Tracing tab.
4. We can enable CloudWatch logs, Detail CloudWatch Metrics, access logging.
5. On checking the box for CloudWatch Logs, we can set it the Log level to 'ERROR' or 'INFO'.
6. We should enable "Log full request/response data" only if there's no tricky or sensitive information.
7. To make these changes work, go on to Settings on the left hand console page & we need to provide CloudWatch log ARN.
8. Go to IAM under Services.
9. Create a Role. Select "API Gateway". We can see it says "Allows API Gateway to push watch to the CloudWatch Logs".
10. Click on Next.
11. Choose "AmazonAPIGatewayPushToCloudWatchLogs" & click on Next.
12. Give the Role a Name.
13. Click on "Create Role".
14. Copy the Role ARN & paste it to Settings under API gateway.
15. Click on 'Save'. Now API Gateway can send logs to CloudWatch.
16. Go back to Stages.
17. Select the root of the Stage created.
18. Go to Logs/Tracing tab and enable the logs of your choice.
19. Click on "Save Changes". Click on Invoke URL.

Cross-Origin Resource Sharing (CORS)

- CORS must be enabled when we receive API calls from another domain.

- For this to enable, the OPTIONS pre-flight request must contain the following headers:
 - Access-Control-Allow-Methods
 - Access-Control-Allow-Headers
 - Access-Control-Allow-Origin
- CORS can be enabled from the console.

LAB

1. Go to API Gateway under Services.
2. Navigate to Resources.
3. Go to the root & click on ‘Actions’.
4. Choose “Enable CORS”.
5. We can choose the type of response, how do we want to set-up the GET, OPTIONS, where are the headers, and whether the origin.
6. Click on the button “Enable CORS & replace existing CORS headers”.
7. We can see that “OPTIONS” gets added below the method function.

Usage plans & API Keys

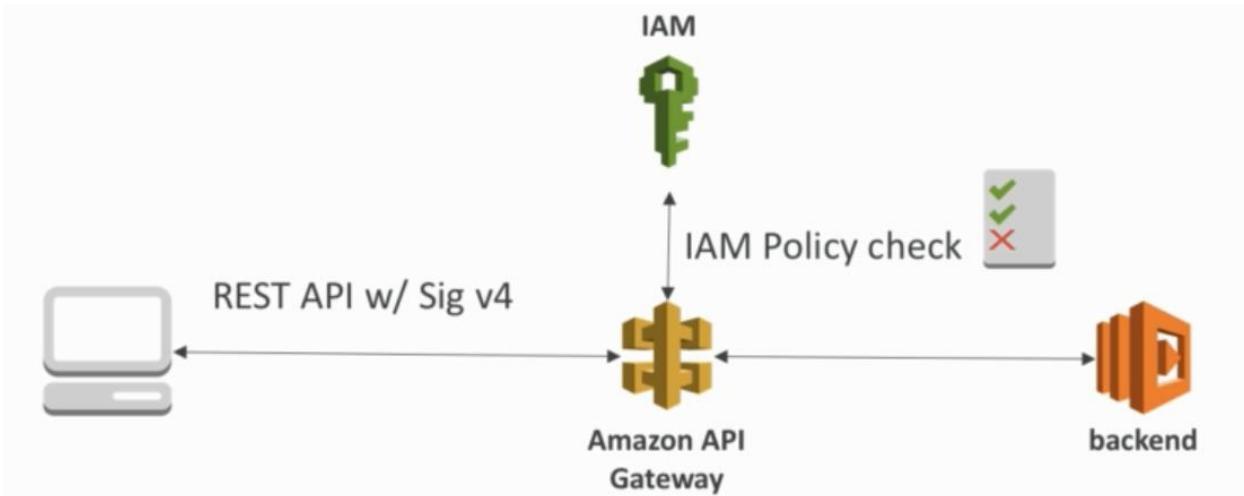
- We’re able to set usage plans & API keys.
- E.g.: we’ve customers & we want to limit their API usage.
We can have usage plans:
 1. Throttling: We can get some throttling, so how fast we can use our API so we can send overall capacity & then burst capacity.
 2. Quotas: Number of requests made per day / week / month.
 3. Associate with desired API Stages.
- API Keys: When we now have usage plan, we’ll need to associate it with a API key.
 - So we need to generate one key per customer.
 - Then we need to associate these keys with usage plans.
- We’re ability to track usage for API keys and even we can bill our clients for these API keys. Using the usage plans and API keys, we’re able to share and sell our APIs to people if we wanted to.

Security

There are 3 aspects, there is IAM Permissions, Lambda authorizers, and theirs is Cognito user pool.

IAM Permissions

- Create an IAM policy authorization and attach to User / Role. If we want to give a role access to one of our user to one of API that makes sense to attach an IAM policy to our user and our role.
- API Gateway verifies IAM permissions passed by the calling application (when we call our REST API).
- It's good to provide API access within our own infrastructure.
- Sig v4, or signature v4 capability is leveraged where the IAM credentials are under one header and the header is passed on to the API Gateway. Sigv4 is the process to add authentication information to AWS requests sent by HTTP.
- E.g.:



We've our client which is calling API Gateway with Sig v4, and then API Gateway calls IAM, verifies the policies, makes sure it all checks out, and if it is happy, goes to the back end.

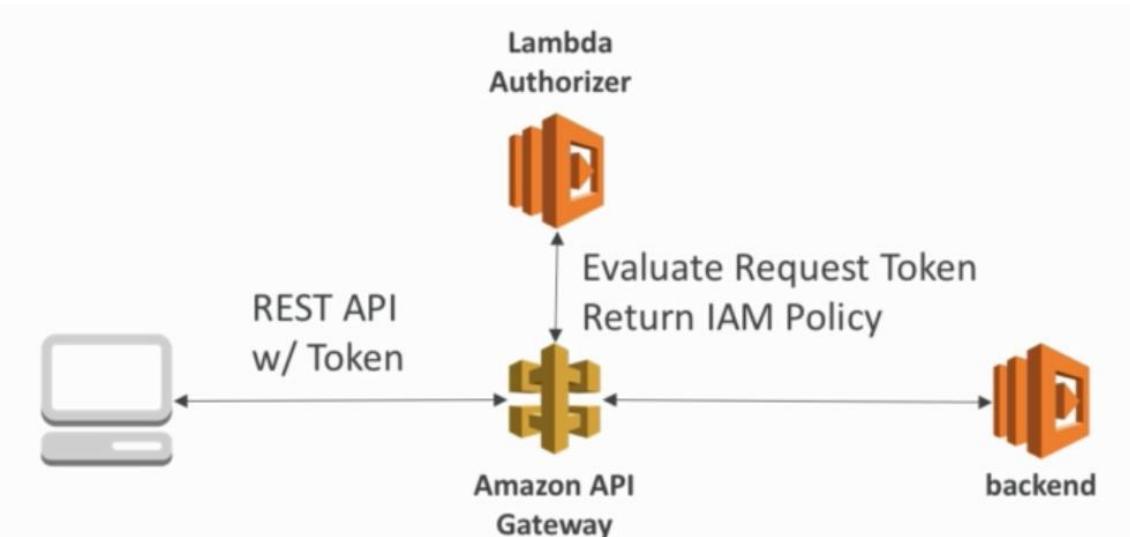
Advantages

1. There's no addition cost to this solution.

2. Note: When we see Sig4, we need to think of IAM permission for API Gateway.
3. It is a pretty easy solution, but if we give access to users outside of our AWS then we can't use IAM for missions.

Lambda Authorizer (formally Custom Authorizers)

- As the name indicates, Lambda authorizers uses AWS Lambda to validate the token in header being passed in the header of our request.
- We've an option to cache result of authentication
- We can even use OAuth / SAML / 3rd party type of authentication.
- As a result of authorization, Lambda must return an IAM policy for the user, later the IAM policy defines whether the user can call IAM policy.
- E.g.:

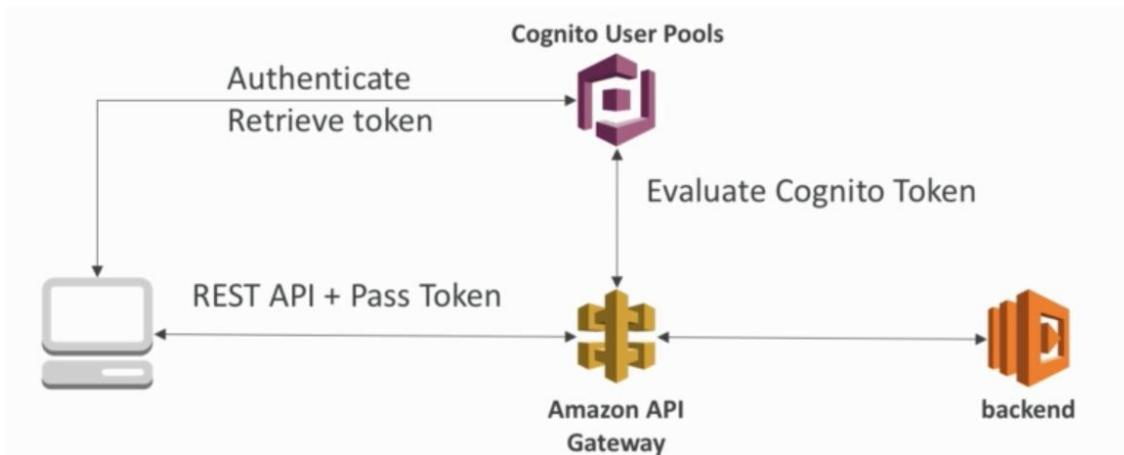


Our clients calls the REST API with a 3rd party token, the API Gateway calls the Lambda Authorizer and pass the token to Lambda authorizer and Lambda returns an IAM policy and if everything runs proper, then API Gateway talks to the backend and we're good.

Cognito User Pools

- It manages full user lifecycle.
- API gateway will automatically verify the identity from AWS Cognito.
- We don't need to implement any custom Lambda function

- Cognito only helps with authentication not with authorization.
- E.g.:



The client calls the Cognito user pools to authenticate, then the pool gives back the token to the client. The client calls our API Gateway as REST API and pass the token that it just received from the Cognito user pool. The API Gateway will then make sure that the Cognito token is correct by talking to Cognito directly. When it succeeds, we can talk to backend.

- The backend must ensure that, we're authorized to make the call. In this type of solutions, we manage our own user pools. It helps us to Facebook, Google authentication.

Summary

- IAM:
 - When we already have users / Roles ready in our AWS account.
 - It handles authentication & authorization through IAM policies.
 - Leverages Sig v4
- Custom Authorizer
 - Great for 3rd party tokens that we don't control, and we're very flexible in terms of what IAM policy is going to be returned.
 - We can handle authentication and authorization because we return an IAM policy.
 - We're going to pay per Lambda invocation, but we can use caching to limit the number of calls to our Lambda function for authorizing. But

if we've 1 million users then we need to call our Lambda function one million time every time the cache gets invalidated.

- Cognito User Pools:
 - We manage our own user pools (can be backed by Facebook, Google login, etc.)
 - No need to write any custom code.
 - Must implement authorization in the backend. As Cognito only provides authentication pattern not authorization pattern.

Cognito

- When we want to give users an identity, so that they can interact with our application, we use Cognito.
- Cognito User Pools:
 - Sign-in functionality for app users.
 - Integrated with API Gateway
- Cognito Identity Pools (Federated Identity)
 - Provide AWS Credentials to users so that they can access AWS resources directly.
 - They can integrate with Cognito User Pools as an identity provider.
- Cognito Sync
 - It is used to synchronize data from Device to Cognito.
 - It may be deprecated and replaced by AppSync

Cognito User Pools (CUP)

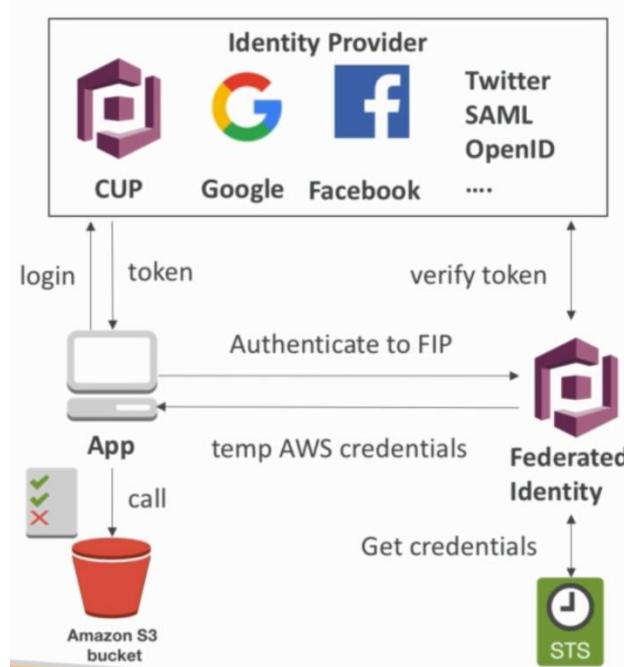
- It creates a serverless database of our users for our mobile apps.
- Simple login: Username (or email) / password combination
- Possibility to verify emails / phone numbers and add MFA
- Can enable Federated identities (Facebook, Google, SAML, etc.)
- In return, we get back a JSON Web Token (JWT). This token can be used to verify the identity of someone.
- It can be integrated with the API Gateway for authentication.
- E.g.:



We've an app & it wants to authenticate to CUP. It is going to register our login using a password, and CUP. After verifying login, it says “Okay, here is our JSON Web Token”.

Federated Identity Pools

- Provides direct access to AWS resources from the Client Side.
- It is done by logging into Federated Identity provider – or remain anonymous
- From this, we get temporary AWS credentials back from the identity Pool.
- These credentials comes with an IAM Policy attached to it.
- E.g.: Let's suppose we want to provide temporary access to provide write to an S3 bucket using Facebook login.



We've an app & our app has the ability to provide login to identity provider. The identity provider can be Google, FB, etc. So, from there our app gets login & get a token. Using this token, we're going to pass it on to Federated Identity Pool. Again, Federated Identity verifies token with our identity provider. Once the token has been verified the token will talk to STS Service to get temporary credentials for AWS. Once it has that, it passes the temporary credentials back to our application, and once our application has these temporary AWS credentials, it is able to interact directly with S3 bucket. Also, we've an IAM policy which allows us to do certain things. The idea behind it is we trade in a login for an identity provider and at the end we get temporary AWS credentials allowing us to interact with AWS for our app users.

AWS Cognito Sync

- It is deprecated and we should use AWS SyncApp now.
- It allows us to store user preferences, configuration & the state of the app.
- It has a cross-device synchronization capability (like Android, iOS, etc.)
- Offline capability (Syncroniztion back when online), so if we were to change our preferences offline and when we go back online, they're synchronized automatically.
- Requires Federated Identity Pool in Cognito (not User Pool).
- The data is stored in data sets & each data set can store 1MB and we can have up to 20 data sets to synchronize.

Cognito user pools vs. identity pools

Use a user pool when you need to:

- Design sign-up and sign-in webpages for your app.
- Access and manage user data.
- Track user device, location, and IP address, and adapt to sign-in requests of different risk levels.
- Use a custom authentication flow for your app.

Use an identity pool when you need to:

- Give your users access to AWS resources, such as an Amazon Simple Storage Service (Amazon S3) bucket or an Amazon DynamoDB table.
- Generate temporary AWS credentials for unauthenticated users.

19. Serverless Application Model (SAM)

SAM allows us to write YAML files and templates, the way our application should behave and deployed. It is basically a shortcut to do CloudFormation. SAM is a squirrel and also a serverless development framework which allows us to deploy our serverless application onto AWS.

- It is a Framework used for deploying Serverless application.
- All the Serverless application is configured in YAML code.
- It generates complex CloudFormation code from simple SAM YAML file.
- Since our SAM Code is CloudFormation code, it supports anything that CloudFormation supports such as Output, Mappings, Parameter, Resources, etc.
- There're are only 2 commands to deploy to AWS.
- SAM can use CodeDeploy to deploy our Lambda function.
- SAM can help running our Lambda, API Gateway, DynamoDB locally.

Overview

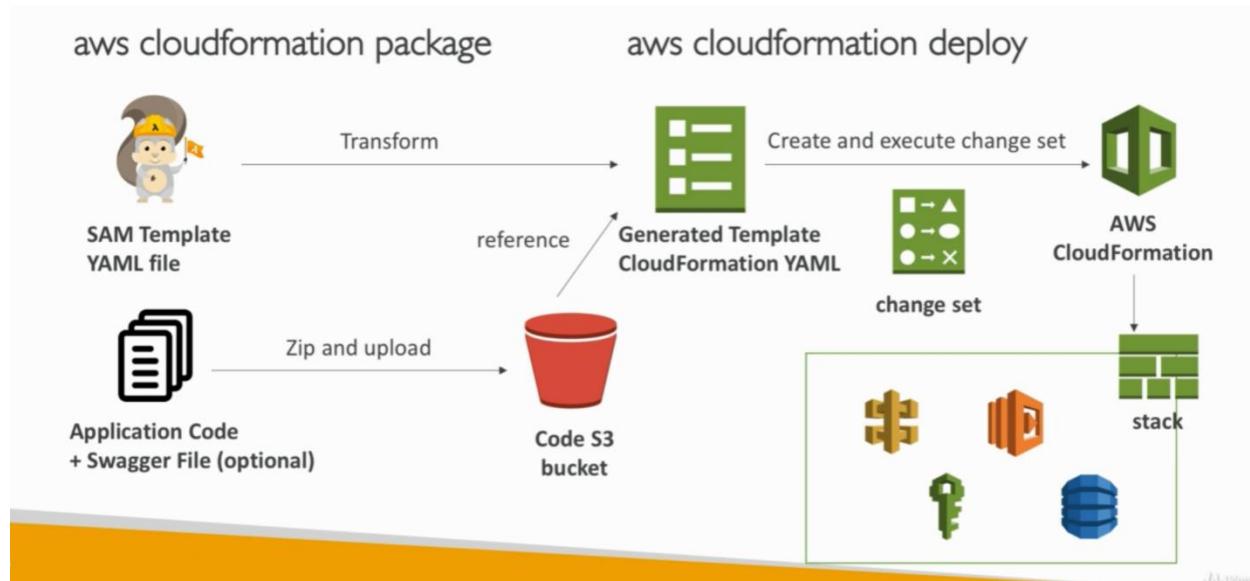
Recipe

- There is a transform header which indicates it is a SAM template.
 - Transform: 'AWS::Serverless-2016-10-31'
 - So, whenever we see this kind of header, we should get a hint that it is a SAM template.
- Whenever we write code, we can use these 3 Resources that're brought in, especially by the SAM format which is:
 - AWS::Serverless::Function
 - AWS::Serverless::API
 - AWS::Serverless::SimpleTable
 - The function is going to be Amazon Lambda, the API is going to be API Gateway, and the simple table is going to be DynamoDB.

- Package & Deploy

- We can use AWS CloudFormation package / sam package as CLI.
- When we package our application, we use AWS CloudFormation Deploy / SAM Deploy.

SAM Deployment



Let's run our first command, AWS CloudFormation Package. Let's suppose we've got our SAM template, it's a YAML file. And we also have our application code and an optional Swagger file (optional - if we want to have an API gateway). By doing AWS CloudFormation package, it uploads the ZIP file of our code to the S3 bucket therefore S3 buckets holding all our deployments and our packaging.

The function AWS CloudFormation package will also transform our SAM template into a CloudFormation template. This transform indicates how CloudFormation should transform their templates into much more complex CloudFormation YAMLS. We upload the code to S3 because the generated templates will have a reference to the uploaded code in S3.

Now, we're going to run our 2nd command, AWS CloudFormation Deploy and it does deployment of the generated template. In particular, it'll create & execute the change-set. Change-set is basically figuring out how CloudFormation should take its existing state and move it to next state based on the modifications we've generated. Here, there's a change-set and CloudFormation applies to our

stack and our stack may be comprised of all the things we know from before like API Gateway, Lambda function, etc.

SAM CLI

- It helps us to develop & test our functions locally with `sam local`.
- It invokes functions from known event resources: Amazon S3, DynamoDB, etc.
- It validates SAM templates.
- We can generate Serverless service using `sam init`.

Installation

1. Open the URL <https://github.com/awslabs/aws-sam-cli>
2. Under “Get Started” section, click on the installation link.
3. The prerequisites for installation are:
 - a. Python 3.6 / 3.7
 - b. Docker
 - c. AWS CLI
4. Run the .exe, it will create 3 shortcuts.
5. If you have, <win10 download oracle toolbox from Github from the URL: <https://github.com/docker/toolbox/releases>
6. Run “Oracle VM VirtualBox” & click on ‘New’, and then click on “Run” (Create Docker VM if we’re running for the first time).
7. Start “Docker Quickstart Terminal” and we can see the Docker Ship is available displaying “Start interactive shell”.
8. Once the above steps has been completed, run the AWS CLI
9. Run the command to check Python is installed: “`python --version`”. If not installed, install python and add its path to system variables.
10. To check whether pip is installed, run: `pip --version`. If not installed, download & install pip.
11. Install AWS-SAM-CLI using the following command: `pip install –user aws-sam-cli`.
12. To initialize the project, we can run: `sam-init --help`. `sam-init` creates a project for us. We can change its python runtime version. `sam-init` creates a lot of files, so we’re going to create the files one by one.

LAB

1. Let's create a source folder named src.
2. We'll create an app.py file in that folder. It will contain our application.
3. We'll also need a template file named template.yaml file.
4. We'll need some commands, so we'll create commands.sh file.
5. Open & copy the lambda_function.py & template.yaml from the URL:
<https://github.com/aws-samples/serverless-app-examples/tree/master/python/hello-world-python3>
6. Modify the file contents (for now navigate to the folder: "C:\Users\Raunak\Documents\Python Scripts\SAM_CLI-src").
7. Go to commands.sh file & edit the files.
8. After coding the app.py, template.yaml & command.sh file & running them, go to CloudFormation under 'Services'. We can see that a new stack has been created.
9. Navigate to the 'Resources' tab, we can see 2 resources being created. It created a Lambda function & an IAM Role.
10. Navigate to the events tab, it shows the list of event being executed.
11. If we go to Lambda under Services, we can see that a new lambda function has been created.
12. When we see the function code, we can see that it is the same code as we wrote in app.py file (local environment).
13. Test the code & we can see the execution result has been succeeded.
14. We'll integrate API Gateway in front of our Lambda function. Go to URL:
<https://github.com/aws-samples/serverless-app-examples/tree/master/python/microservice-http-endpoint-python3>
15. Copy the python file & YAML file and modify it.
16. Run the commands, & we can see there's a YAML file generated in the specified path.
17. We can see the function generated & an API gets created.
18. Make the code changes & package & deploy.
19. Go to the DynamoDB table created & add some items to it.
20. Refresh the Lambda function page on console.
21. We can see that the environment variables' value were automatically set.

- 22.Under “Basic settings”, we can see a role has been created under Existing role.
- 23.Under IAM Role, we can see that a role has been created and it has AWSLambdaBasicExecutionRole policy attached to it.
- 24.It has also an inline policy "microservicehttpenddpointpython3RolePolicy0" to perform multiple task like “GetItem”, “DeleteItem”, “PutItem”, etc. al on our table.
- 25.Go to API Gateway, and navigate to the API created in step 17.
- 26.’Test’ the resources and we can get the response which are written in the DynamoDB Table.
- 27.Navigate to CloudFormation under Services and choose the stack we created in step 18.
- 28.Under ‘Template’ tab, we can see the template we designed in the YAML file.
- 29.If we click on “View processed template”, it gives the detailed information of our template.
- 30.We can even view the template in designer file.
- 31.If we navigate to ‘Lambda’ under ‘Services’, and click on “Create function”, we can choose “Browse serverless app repository”.

Policy Templates

- These’re the list of templates to apply permissions to our Lambda Functions.
- Important example:
 - S3ReadPolicy: Gives read only permissions to objects in S3.

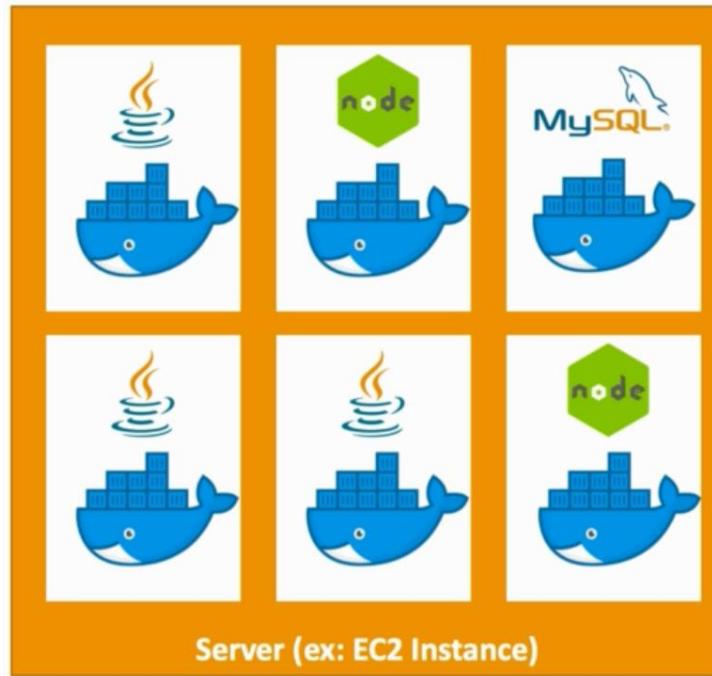
```
MyFunction:
  Type: 'AWS::Serverless::Function'
  Properties:
    CodeUri: ${codeuri}
    Handler: hello.handler
    Runtime: python2.7
  Policies:
    - SQSPollerPolicy:
        QueueName:
          !GetAtt MyQueue.QueueName
```

- SQSPollerPolicy: Allows to poll an SQS queue.
- DynamoDBCrudPolicy: From our Lambda function, we can Create, Read, Update, &Delete operation to our DynamoDB table.

20. Docker, ECS, ECR, Fargate

Docker

- Docker is a software development platform used for deploying applications.
- Apps are packaged in containers that can be run on any OS.
- They run the exact same way on all machine, it allows us to:
 - Run any application on any machine.
 - No compatibility issues.
 - Predictable behavior, no surprises.
 - Less work.
 - Easier to maintain & deploy since it is standardized.
 - Works with any OS, any language & any technology.



- Suppose we've an OS, for example EC2 Instance running Linux, we'll be able to deploy a Docker container on it which runs Java, NodeJS, SQL Database, etc. These containers can talk to each other, be linked, and networked. Even they run on the same machine, they don't interact & impact with each other until we tell them to do so. Even if one application is going crazy, another one is not going to be infected.
- Since, they run on the same server, we can scale in & out (we can have more Java containers)

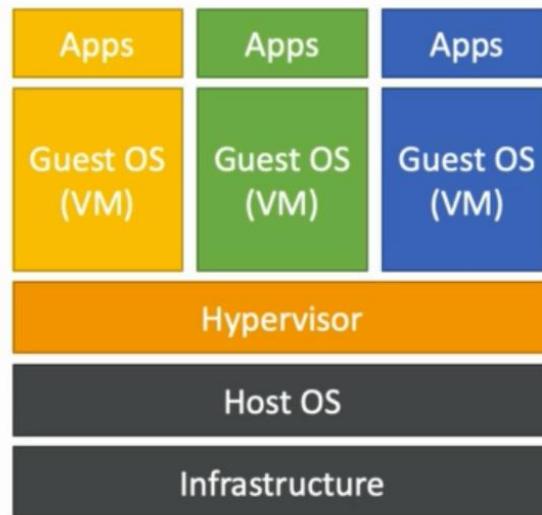
Storage

- Docker images are stored in Docker Repositories.
- The 2 famous repositories are:
 - Public: Docker Hub at <https://hub.docker.com/>. In this we can find base images for many technology or OS like Ubuntu, MySQL, Java, NodeJS, etc.
 - Private: This is where we store our personal images, and one of them is Amazon ECR (Elastic Container Registry).
- Go to <https://hub.docker.com/>
- In the search bar, type the images you're looking for like MySQL, Ubuntu, Java, DynamoDB, etc.

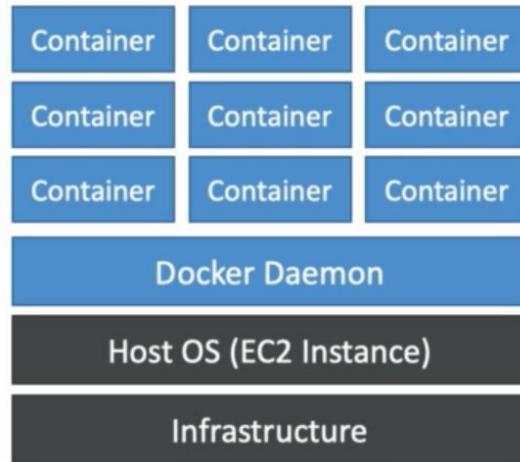
- Download & install Docker.
- Create a Docker file, after building them we'll have Docker images & when we run Docker images, these creates Docker containers. We can push our images to Docker Hub to make them public or to Amazon ECR to make them private and later on it can even be pulled.

Docker vs. Virtual Machine.

- Docker is a sort of virtualization technology, but not exactly.
- With Docker, all the resources like RAM, CPU, they're shared with the hosts which means we can run many containers on one servers.



- In case of VM, we have the infrastructure, the actual hardware, then comes the host OS, and then we've the hypervisor (They allow to run multiple OS run simultaneously on a computer system). This is how, EC2 is built. There's a giant server in Amazon's Data center and they have a host OS and they run Hypervisor on it). When we request for EC2, we get the guest OS and an app that could be run on it and same happens when multiple request happens. These guest-OS are logically & securely separated. If we want to run more VMs on these guest OS, we'll have to run another hypervisors, and hence we can stack up very quickly.
- Docker:



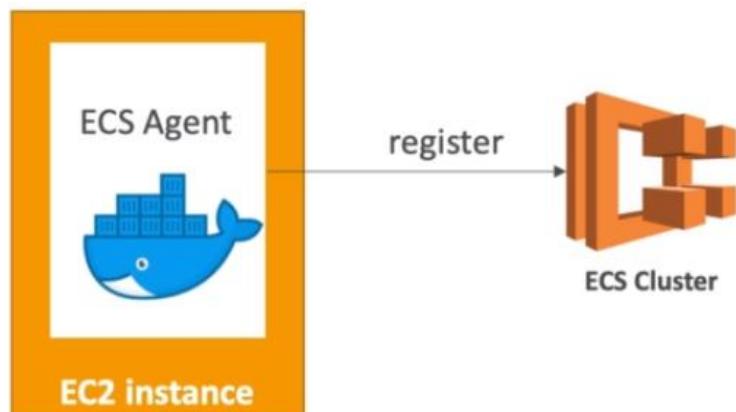
- In case of Docker, we've the infrastructure, the Host OS. It may be our EC2 instance, and there's no Hypervisor. There's something called Docker Daemon and after installing Docker Daemon, we can run multiple container for whatever technology we want.

Docker Container Management

- To manage containers, we need container management platform.
- There're 3 choices for this:
 - ECS: Amazon's own platform.
 - Fargate: Amazon's own Serverless platform.
 - EKS: Amazon's managed Kubernetes (open source).

ECS (Elastic Container Service)

Introduction



- ECS clusters are logical group of logical EC2 instances.

- EC2 instances run the ECS agent (Docker container).
- The EC2 instances agents registers the instance to the ECS clusters.
- The EC2 instances run a special AMI, made specifically for ECS (not the plain Amazon Linux 2 AMI).
- We've our EC2 instance which is running an ECS Docker agent which'll register our instance into the ECS cluster.

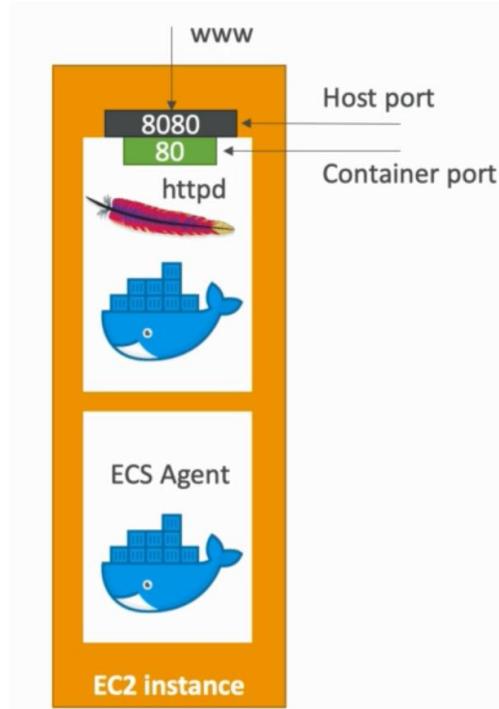
LAB

1. Go to ECS under Services.
2. Click on “Create Cluster” from the ‘Clusters’ menu.
3. Click on EC2 Linux + Networking.
4. Click on “Next step” button.
5. Give your cluster a name.
6. For now, select instance size to be t2.micro.
7. Leaving other settings to be default, select a key pair.
8. Under the network settings, we'll select the old VPC under VPC.
9. Select all the 3 subnets so that our ASG uses 3 subnets.
10. Select to create a new security group.
11. Under Container instance IAM role, we need to create a new role because it needs to be provisioned an IAM Role so that EC2 instance can register to ECS.
12. Hit on ‘Create’.
13. The ECS instance is linked to an EC2 instance. The “Agent Connected” is true says that EC2 instance is running on the machine.
14. When we create an ECS Cluster, it came with an ASG. If we navigate to EC2 under Services, and then go to Auto Scaling Groups, under ‘Details’ tab, we can see it is multi AZs.
15. We can click on ‘Edit’ & can add capacity to it.
16. Further, if we go to “Launch Configuration” under Auto-Scaling Group, under the ‘Details’ Tab, we can see the User data. It is echoing ECS_Cluster data which equals cluster-demo into ecs.config file & this is how our instance starts to register to the cluster.
17. It also created a Security group which allows port 22.

18. An IAM Role is also being created which contains AmazonEC2ContainerServiceforEC2Role.
19. Let's SSH into the machine with the public IP being generated. We can see that it is an Amazon ECS-Optimized Amazon Linux AMI.
20. If we run `/etc/ecs/ecs.config`, we can know how our instance gets registered to the cluster-demo. It is registered by Docker Agent.
21. We'll type in `dockerps`, which means docker processes. We can view the container ID, the IMAGE Name, the command is "/agent", & this agent is registering our instance to ECS service.
22. If we copy-paste the container ID with command, `docker logs container_id`. We can see the logs of the ECS agent of what happens.

ECS Task definition

- Task 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 & host.
 - Memory & CPU required.
 - Environment variables.
 - Networking information.
-



There's an EC2 instance running an ECS agent and we're going to run HTTPD (Apache server) in a Docker container & we'll pull it from Docker Hub. The Docker image is created inside the container. Port 80 represents the Apache server. We're going to map this host (port 80) onto port 8080 (just an arbitrary number). When some traffic comes from the internet into our EC2 instance on port 8080, then Docker maps port 8080 to port 80 within our Apache container and will be directly talking to Apache.

LAB

1. Go to ECS under Services.
2. Click on Task Definition and click on “Create Task Definition”.
3. For now, click on EC2.
4. Give your name for Task-definition.
5. Task role is an optional IAM Role that tasks can use. If a Task Role fails to performs its task, like pulling an image from ECR it might be because it is missing a task role. (For now, leave it empty).
6. We will leave the Network Mode to default. The Network mode is Bridge on Linux & NAT on windows.
7. Task Execution Role is basically allowed to create a task.

8. If we go back to our ECS LAB, we created a Cluster, where the EC2 instance has CPU availability equals 1024, and Memory available equal to 985. Now, when we go to Task size section under ECS and when we navigate to task Memory, we set it 300 and CPU and Task CPU to be 250. This is the amount the RAM we're going to assign to our Task.
9. Now, we're going to add container definition.
10. Give your container a name.
11. Now, we need to choose the image. To select Docker as an image, select 'httpd'. The tag we're going to select the latest one, say 2.4. It looks like 'httpd:2.4'.
12. Under Memory Limits, we'll add a Hard Limit say, "300". We can add Soft limit even, but, we'll leave it for now. Hard Limit & Soft Limit corresponds to the 'memory' and 'MemoryReservation' parameters.
13. Set the Host port to 8080 & container port (here the Docker's port) to be 80.
14. Leave the many other options to be default and click on add and click on create.
15. We can see the name as task_name:1, where '1' represents the version number.
16. Under JSON tab, we get the JSON form of task definition.

ECS Service

- ECS service help define how many task should run and how they should run.
- They ensure that number of task desired is running across our fleet of EC2 instances.
- They can be linked ELB / NLB / ALB if needed.

LAB

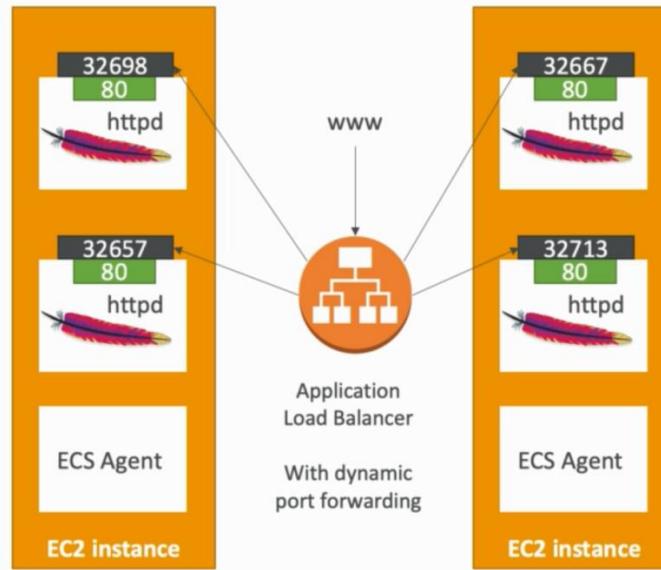
1. Go to ECS under services.
2. Go to Cluster sub-menu.
3. Go to the cluster created previously.
4. Under Services tab, click on Create.
5. Select launch type to EC2.

6. Task definition comes from the one we previously created.
7. Select the revision ('1' for now).
8. Assign the service a name.
9. Select Service type to be 'Replica' for now. Replica means we're going to run as many task as possible.
10. So, let's say "Number of tasks" to be 5. But if we run Daemon, the "Number of Task" is changed to 'automatic' and it's going to run one task on each ECS instance and ECS cluster. Daemon is like when we want to run monitoring across our instances and gather metrics, but for our case since our application will just run 'Replica', setting Number of Task to be '1'.
11. Setting minimum health percent to be '0'. It allows us to do rolling restarts.
12. Setting the maximum health percentage to be 2.
13. We're going to select the Deployment type to be "Rolling type".
14. Deployments helps us decide how our service is going to be deployed.
Select it as "Rolling update". There's a new rolling update called "Blue-Green deployment" that is powered by "CodeDeploy". So, CodeDeploy can be used to deploy ECS as well.
15. Select the task placements to be "AZ Balance Spread". It defines the strategy how to place task on ECS instances.
16. Click on Next Step.
17. We'll select Load Balancer type to be 'None'.
18. Under Service discovery, we'll disable the Service discovery integration.
19. Click on Next step.
20. We'll not set the Auto-scaling but it is possible to set-up auto scaling for our ECS Service and click on Next Step.
21. Hit on "Create Service". When the service status is in Pending state, which means we're going to launch our httpd task.
22. Click on the container instance and we can see it has created a Public & Private IP. We can see the Memory details under Details tab.
23. Open a new tab, copy the public IP and add port number (8080) to its end.
To make it work, we'll go to our EC2 instance, and click on our Security group and add a custom inbound rule where the type will be "Custom TCP" & port to be 8080. Click 'Save'.

24. Go back to the new tab opened, & now search for public_ip:8080. It says “It works”. We can see now, our first Docker container is running our EC2 instance.
25. If we type “dockerps” in our CLI, we can see that on the top of our ECS agent, there’s httpd 2.4 service running.
26. We can curl this container from our localhost. Type “curl localhost:8080” in CLI and hence it works. (CURL is used to transfer data using various network protocols).
27. We can scale our service and run multiple task. Go to ‘Services’ tab under the Cluster created.
28. Click on ‘Update’ button and we can update the number of task (for now change to 2). Click on next & finally click on “Update Service”.
29. If we go to ‘Task’ tab under Cluster section, we can see only one task running. When we navigate to ‘Events’ tab, we can see, no other task pending and moreover it says “unable to place a task because no container instance met all of its requirements... The container is already using a port required by yourtask”. This is because we have set that we can have only one task per ECS instance and if we try to launch a second task, it’ll again use port 8080 which is already used by our instance. Hence, it’ll not work. To resolve this, we’ll go back to our cluster.
30. Under ECS instance tab, click on ‘Actions / Scale ECS instance’ and we’ll set it to ‘2’ and click on ‘Scale’. This will create another t2.micro instance and it joins our cluster and then run our task.
31. Click on ‘Refresh’ and we can see both our instances.
32. Go back to Auto-scaling group in EC2 under Services. We can see that “Desired capacity” has been set to ‘2’, and ‘Instances’ tab represents ‘2’ instances of total.
33. If we go back to ECS, under Cluster we can see that both task under ‘Task’ tab is running.
34. If we ping the public_ip:8080 of our new task, it says “It works”.

ECS with Load Balancer

The idea now is that we want to run our ‘httpd’ service, but this time on different EC2 instances.



This is our instance, and we're going to change task definition but not going to include any host ports. We'll just specify container port and host port become random and we'll repeat this step many times. Every time we'll launch a task, we'll have a random port number. It'll be difficult to direct the traffic to these port number. But Load Balancer will balance this and it'll receive our traffic from internet. This is called dynamic port forwarding. It'll route the traffic to random port & spread our load to different containers. So, Dynamic Port Forwarding is how we run multiple same task on the same EC2 instance.

LAB

1. Go to the Clusters in ECS.
2. Click the Service tab.
3. Choose your service & select the number of task to be 4.
4. We can see that 2 task is running under Task tab.
5. Go back to task definition.
6. Select your tsk created & click on “Create new version”.
7. Leave all settings as default except under “Port mappings”, clear the Host port tab & set it to null.
8. Click on ‘Update’ and ‘Create’. A new task with “task_name:2” revision number gets created.
9. Under ‘Service’ tab, we’ll set the Revision to be 2.
10. Click on Next step(s)& update.

11. We can check the deployment, and we can see the count of “Desired count” is 4. Later on, it creates 4 task.
12. If we go to ECS instances tab under Cluster, we can see running task is 2-2 and things like CPU & memory availability go split.
13. Type dockerps in CLI. It shows 2 ports running.
14. If we curl local host, we can say, it says “it works”.
15. Go to Service under cluster, & if we update its Load balancing, it says “ELB cannot be configured. It can only be set at time of service creation. So we need to create a new service.
16. Give your task a name.
17. Select the latest revision.
18. Keep the cluster name same.
19. Give a new service name and service type to be replica.
20. Number of task is set to 4.
21. Min-Max percentage to be 0-200.
22. Click ‘Next’.
23. We’ll choose the Load Balancer type to be “Application Load Balancer”. It allows containers to do Dynamic Host Port Mapping.
24. Click on “Create a new Role”.
25. Also, we need to create Load Balancer, select ALB.
26. Give your ALB a name.
27. Choose other settings to be default.
28. Select all the AZs.
29. Click on Next.
30. Create a new security group which listens on Port 80 and click on Next.
31. Give your target a name, click on next 7 review it.
32. Since our Load Balancer is going to talk to our EC2 instance, we need to allow the Load Balancer to talk to our instance on any port & it has to be done through security groups.
33. Go to the security group created in step 30.
34. We will remove the port 80 traffics & add type to be “All traffic”.
35. We’ll add a security group to it say ecs-asg& hit on save.
36. Go back to service and refresh and we’ll have the Load Balancer.
37. Under Service IAM Role, choose a role created before, or else Create a new role.

38. We'll add container to Load Balancer.
39. The Production listener port will be 80:HTTP.
40. Under Path pattern , we'll leave it just '/' and evaluation order to be 1.
41. The health-check path will also be '/'.
42. We'll disable the service discovery & click on Next and hit on "Create Service".
43. When we go to Cluster demo, we can see 2 services running and it starts to run some task. The older one can be deleted now before updating it as Number of task to be 0.
44. When we see the details of the service, we can see there's 4 task running.
45. When we click on any of the Task, under Containers section we can check the Host ports.
46. Go to Load Balancers under EC2.
47. Copy the DNS name & paste it on the browser, it says "It works!"

ECR (Elastic Container Registry)

Introduction

- So far, we've been using Docker images from Docker Hub (public).
- ECR is a private Docker image repository.
- Access is controlled through IAM (if permission error, check policy).
- We need to run push & pull commands.

LAB

1. Go to Git command prompt after installing Docker.
2. Type docker --version.
3. Go to the directory where the Docker file is located.
4. Run the command: "docker build -t my-httpd-image ." It installs a bunch of task and get our image ready. When it is completed, we get our Docker image build and we're 'going to run in ECS, and before that we need to push it in ECR.
5. Go to ECR under Services and create a new repositories.
6. Give it a name & create one.

7. When we click on the repositories name, we can see that there's no image present.
8. Click on view push commands
9. On a Linux machine, to authenticate our Docker client to registry, we need run: `aws ecr get-login --no-include-email --region region_name`. It generates a temporary Docker login that we can use to login our Docker against ECR repository & start pushing and pulling images from it. But in Linux, Mac machine, the command do not perform any operation unless we surround the command by “\$ (command_name)”. It returns with “Login Succeeded”. On windows, we need to run: “Invoke-Expression -Command (Get-ECRLoginCommand -Region region_name)”.
10. To build our Docker image, run “`docker build -t demo`”.
11. After the build gets competed, we tag our image so that we can push the image to this repository, run “`docker tag demo: docker_url`”.
12. To push the image to our newly created AWS repository in ECR, run “`docker push docker_url`”.
13. If we go back to our repository, we can see our latest image.
14. To download the image on our machine, we need to run the same command that we used to login in step 9”.
15. And then to pull images, run “`docker pull docker_url`”.
16. Now, we're going to update our running definition to run the task we just pushed.
17. Go to “Task Definitions” and then click on the task created and select create new revision. (ECR under services)
18. Go to the service created under “Cluster” in previous lab.
19. In “Add container” section, click on ‘Add’
20. Under ‘Image’, write the full image name and hit ‘Create’.
21. Go back to ‘Cluster’ and under the cluster name, go to Services and then click on ‘Update’.
22. Change the ‘Revision’ under ‘task definition’ to be the latest number.
23. Click on next step and “Update Service”.
24. Under ‘Cluster’, go to ‘Task’ tab & we can see the new task definition are being launched from the new revision and the older ones are being shut down.

- 25.Meanwhile when we go to EC2 & click on ‘Targets’, we can see that the previous tasks are draining. It takes time as we can see when we scroll down under ‘Description’ tab to ‘Attributes’ section, there’s a “Deregistration delay” of 300 seconds.
- 26.To make the above task faster, go to Cluster and under ‘Task’, choose the previous version task and click on ‘Stop’.
- 27.When all the task are up, we’ll test it with ELB.
- 28.Go to load Balancers under EC2.
- 29.Under ‘Description’ tab, copy the DNS name and paste the URL in a new tab.
- 30.We can see “Hello world from custom Docker image”. We can view the information like Docker name, Image ID, CPU, etc.
- 31.If we refresh the page, we can see some of the values changing, as the Load Balancer is balancing the task.
- 32.One of the point to be noted was that our EC2 instance was able to pull of these task and was able to pull our data from ECR because of the correct IAM permissions.
- 33.Go to instances under EC2, look at the IAM Role under description (ECS instance role). Under policies, the Elastic container Registry, it has the “Read access” level to all our resources.

Fargate

- When we launch an ECS cluster, we’ve to create our EC2 instances.
- In order to scale, we need to add EC2 instances and then scale our service which was hard to manage.
- We need to manage the infrastructure.
- With Fargate, it’s all Serverless.
- Here we don’t provision EC2 instances, we just create task definitions and AWS runs container for us.
- To scale, we just need to increase the task number.
- So, if we need to run Docker container in the cloud, just we need to think of Fargate.

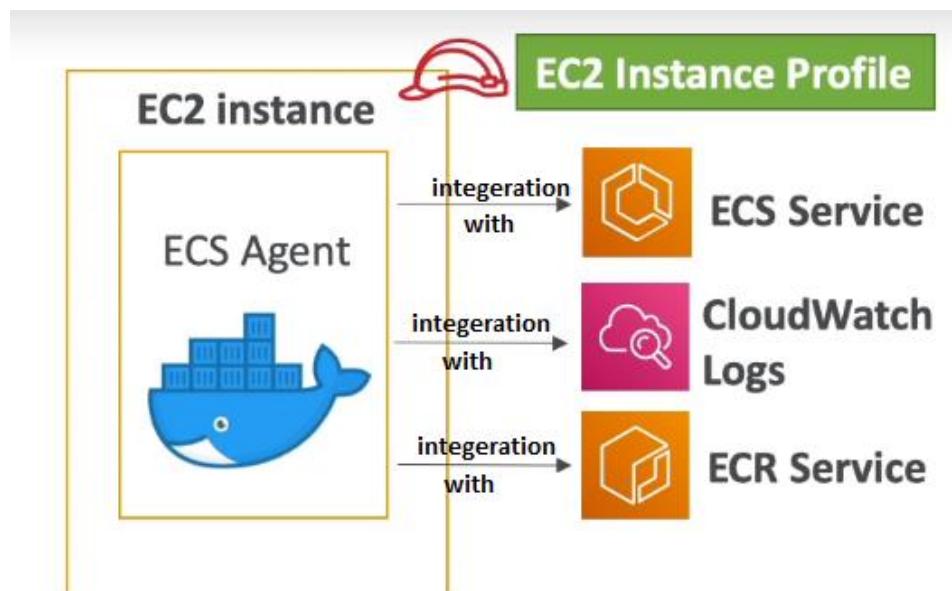
LAB

1. Go to “Elastic Container Service” under Services.

2. Click on Cluster.
3. Select the template to be “Networking only”.
4. Click on Next step.
5. Choose a cluster name of your choice.
6. Create or choose an old VPC.
7. Hit on Create.
8. Go to Service, and click on Create.
9. Select the Launch type to be Fargate.
10. Go to “Task Definition”, create a new Task definition.
11. Select the Launch Compatibility as Fargate, hit on next.
12. Enter a Task definition name.
13. Select the task memory & vCPU.
14. Click on “Add container”.
15. Give your container a name & choose an image.
16. Select Hard Limit under memory limit to be 512.
17. Under Port Mapping, add container port to be 80. There’s no need to do Host port mappings as these are not valid. It is smart enough and it does dynamic routing on its own.
18. Click on ‘Add’ and hit ‘Create’.
19. Go back to Cluster.
20. Create a service under ‘Services’ tab.
21. Select the Launch type to be ‘FARGATE’.
22. Choose the task definition to be the one created in step 18 and select the revision to be latest.
23. Write a Service name of your choice.
24. Select number of tasks to be 2 now.
25. Minimum healthy percentage to be 0 and max to be 200.
26. Select deployment type to be “Rolling update”.
27. Click on next step.
28. Select a VPC and choose all the subnets in that region for now.
29. Choose a security group if it is not created earlier.
30. If we want to assign private IP, then we need to disable “Auto-assign IP”, or vice-versa.
31. Select the Load balancer to be application type.
32. Select the Load Balancer created earlier.

33. Click on “Add to Load balancer”.
 34. Select Production Listener Port to be 80:HTTP.
 35. Create Target Group to be new.
 36. Select Target Group Protocol to be “HTTP”.
 37. Select path pattern to be ‘/’ and Evaluation order to be ‘1’.
- Note: This might give errors, so go back to the Application Load balancer & under listener tab, edit the rules & delete the rule which says "IF Path is /"
38. Select Health Check Path to be ‘/’.
 39. Disable Service Discovery Integration and click on Next.
 40. Choose “Do not select the service’s desired count” under “Service auto-scaling”.
 41. Hit on Create.
 42. The point to be noted here that we've not provisioned any EC2 instance in a Fargate Service, but behind the scenes AWS will provision the Docker container for us in a serverless fashion.
 43. Go to the DNS, and we can see Docker name to be ECS Fargate & Network mode to be ECS Fargate.

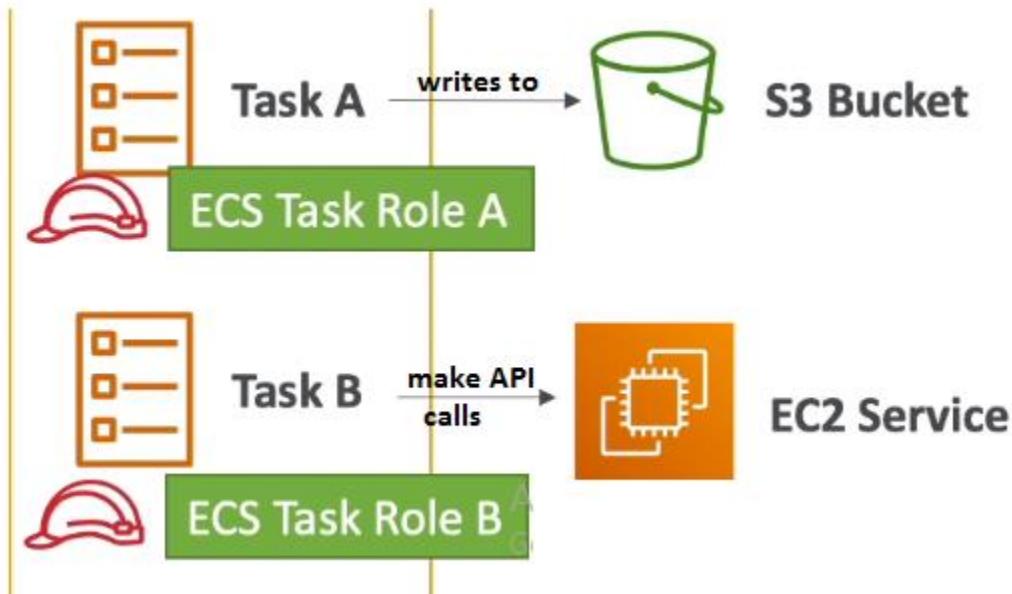
ECS IAM Role Deep Dive



ECS agent has integration with ECS Service, CloudWatch logs, & ECR Service through EC2 instance profile.

How IAM Roles work: Let's suppose we've an EC2 instance and it is running an ECS agent. To make it work, we need to attach an

1. EC2 instance profile
 - a. It is attached at an EC2 instance level. Later, it is used by the ECS agent.
 - b. ECS agent will use that role to make API calls to the ECS service to register the instance with their ECS cluster.
 - c. It sends container logs to CloudWatch logs.
 - d. It pulls Docker images from ECR.



2. ECS task Role: On our EC2 instances, we can run some task.

- a. These task may interact with other AWS Services, so it needs its own role. So, we need to create an ECS Task role.
- b. It allows each task to have a specific role with minimum permissions.
- c. In this example, we'll create an ECS Task role A, & we'll attach to the task A, & for example this task role may allow us to write us to an Amazon S3 bucket.
- d. We use different task role for different services we run. For example, Task B makes API calls to EC2 service if our task interact with the EC2 service.

Note: The main idea is that we want to have an IAM role at the EC2 instance level to help the ECS agent and then we've task role at each task

level so that each task does have the right permissions to do what it needs to do.

- e. Task Role is defined in the Task definition.

LAB

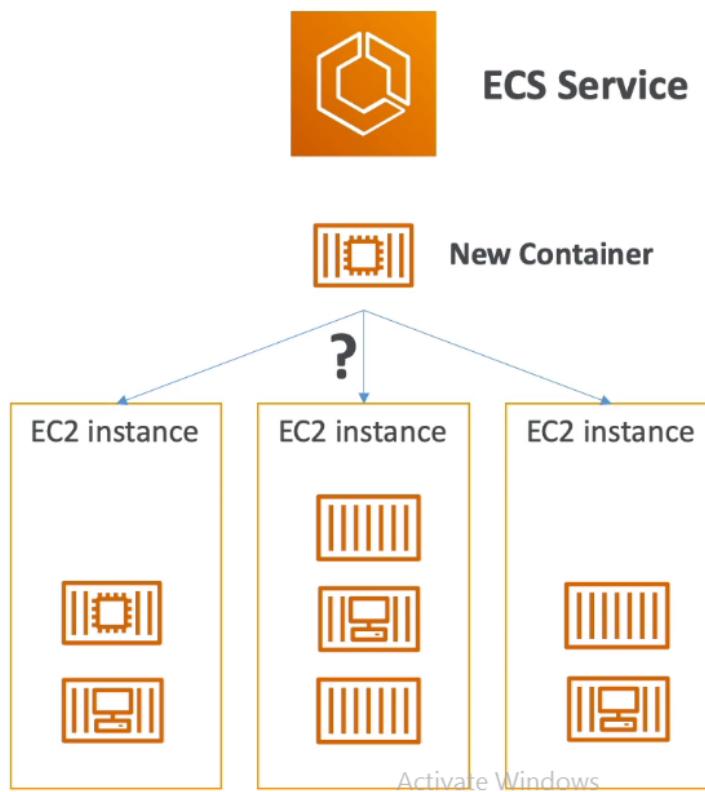
1. Go to IAM and navigate to roles.
2. Search for "ECS", we'll get a list of following roles:
 - a. AWSServiceRoleForECS
 - b. ecsInstanceRole
 - c. ecsServiceRole
 - d. ecsTaskExecutionRole
3. Go to Create Role and select "Elastic Container Service".
4. Under Use case, select Elastic Container Service Task (It represent the Task role which help in creating a separate task role).
5. Click "Next: Permission"
6. Search for "AmazonS3ReadOnlyAccess", select it & click next.
7. Click Next & assign your role a name.
8. Hit on Create Role. Now, this role can be endorsed task & service.
9. EC2 Instance role can be attached to our Instance coz it has Instance Profile ARN. If we look into its policy, it allows us to connect to ECS service, so we can create Cluster, we can do a bunch of API calls to register an Instance, etc. Also, we can do stuffs with ECR, like we can do BatchGetImage, GetDownloadUrlForLayer i.e., we can pull images directly from ECR. The logs help us to allow EC2 instances to log CloudWatch events to ECR using ECS service.
10. ECS Service Role is attached to the ECS Service which can be seen under "Trust relationship" tab which allow ECS service to perform action on our behalf. So basically, when new task are created, it'll allow these task to be registered with Elastic Load Balancer. When new task are created, they can be automatically reserved for our ALB or our CLB for example.
11. ECSTaskExecutionRole is a task role coz if we go to the trust relationship, it can be endorsed by ECS task. So, this is a role that is assigned to specific task & service. If we move to policy, it allows to get images from ECR and send logs to CloudWatch logs.

12. AWSServiceRoleForECS allows ECS service to do action on our behalf. these action allow us to create network interface. de-register Load balancer, route53 & so on.

ECS Tasks

Placement

When we create a task of type EC2, ECS must figure out where to place the task based on the available memory, CPU & ports on our target EC2 instances.



For e.g.: We've an ECS cluster made up of 3 EC2 instances and some task are placed on the instances in some ways. now, if our ECS service has a new container, a new task that it wants to place on our instances.

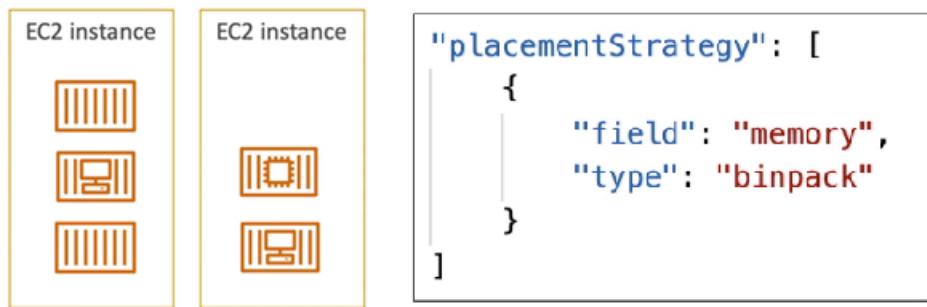
Similarly, a service scales in that is we remove an ECS Task, the ECS Service needs to determine which ECS Task to terminate. To assist with this, we define a Task Placement strategy and Task placement constraints and this guides where the container will be added or removed from. This only works when we use ECS launched on EC2 instance but not for Fargate, coz AWS figures out for us where to start the container and we don't manage any backend instances.

Process

- Task placement strategies are a best effort.
- When Amazon ECS places task, it uses the following process to select the container instances:
 - Identify the instances that satisfy the CPU, memory and port requirements in the task definition.
 - Identify the instances that satisfy the task placement constraints.
 - identify the instances that satisfy the task placement strategies.
 - Select the instances for task placement and place the task.

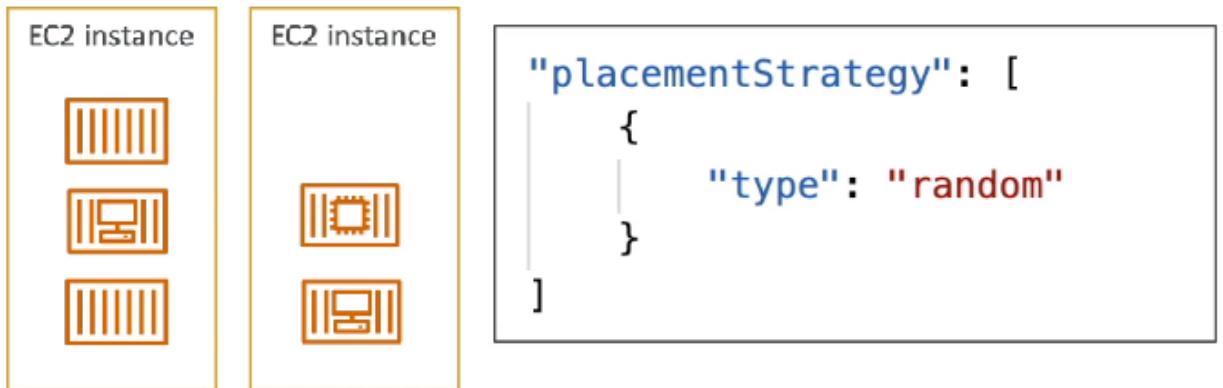
Strategies

1. Binpack:



- a. It place the task based on the least available amount of CPU or memory.
- b. It minimize the number of instances in use (cost savings).
- c. So, if there's one EC2 instance, it tries to fill up that EC2 instance and once its fully filled, then it places the EC2 instances on other ECS instance. Hence, we place as many containers as possible on one instance before moving on to the other, minimizing the number of EC2 instance in use and try to maximize the utilization of the EC2 instance.

2. Random:



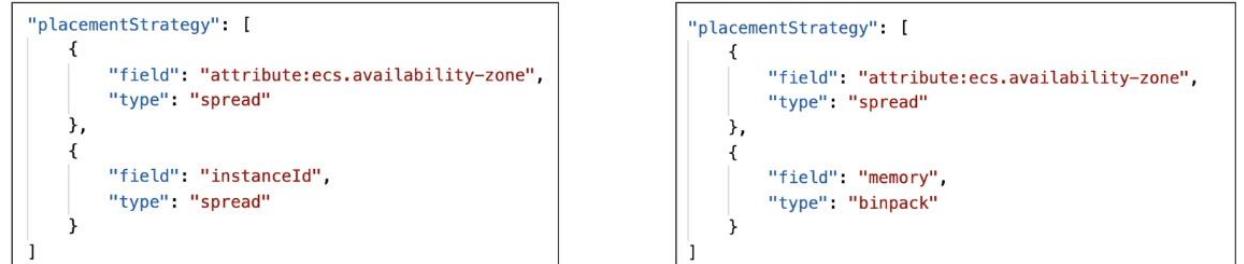
- a. Task is placed randomly.
- b. Say, we've 2 EC2 instance and tasks are being added, then they'll be placed randomly.
- c. It is not an optimal strategy.

3. Spread:



- a. Say, we've 3 EC2 instances and they're in 3 different availability zones.
- b. It places the task evenly based on specified value. Hence, our first task will be on AZ-A, then AZ-B, then AZ-C.
- c. It maximizes the high availability of ECS services by spreading the task on EC2 instances.
- d. Example: the value can be InstanceID, attribute: ecs.availability-zone

4. These placement strategies can be mixed together.



Like, we can have a spread on availability zone, and spread on instance id. Or, spread on availability zone and then binpack on memory.

Constraints

These are used to bring constraints (limitation) how task are being placed. these are of 2 types:

1. **distinctInstance**:

```
"placementConstraints": [
    {
        "type": "distinctInstance"
    }
]
```

We're seeing through the ECS service that each task should be placed on a different container instance. So, we never have 2 task on the same instance.

2. **memberOf**:

```
"placementConstraints": [
    {
        "expression": "attribute:ecs.instance-type =~ t2.*",
        "type": "memberOf"
    }
]
```

We place the task on instances that satisfy an expression that can be defined using Cluster query language, which is pretty advanced. Ex: We would like to place a placement constraint here, keeping the instance type as "t2". it means we want to keep all the task only on t2 instances.

LAB

1. Go to ECS under Services.

2. Navigate to previously created cluster: cluster-demo
3. Go to Services tab & hit on "Create".
4. Select Launch type as EC2.
5. Select task definition to be my-httppd.
6. For now, keep the cluster name as it is.
7. Assign your service a name.
8. Keep the Service type to be replica.
9. Put the number of task as 2.
10. Keep the minimum & maximum healthy percentage to 0-200.
11. Select the deployment type to be rolling update.
12. In the section, Task Placement, we can choose the placement type to be binpack which binpacks our memory. Either, we can select "One task per host" which is a distinctInstance. Either, we can do a mix, like "AZ Balanced Spread" which spread across the availability zones & instances too or "AZ Balanced BinPack" which spread across availability-zone & binpack memory. Else, we can do a custom which allows us to do 2 things where we can define the strategy and the constraints. Select "Custom" for now.
13. Under Strategy, select the type to be Binpack & Field to be memory. Under Constraint, select the type to be "DistinctInstance".

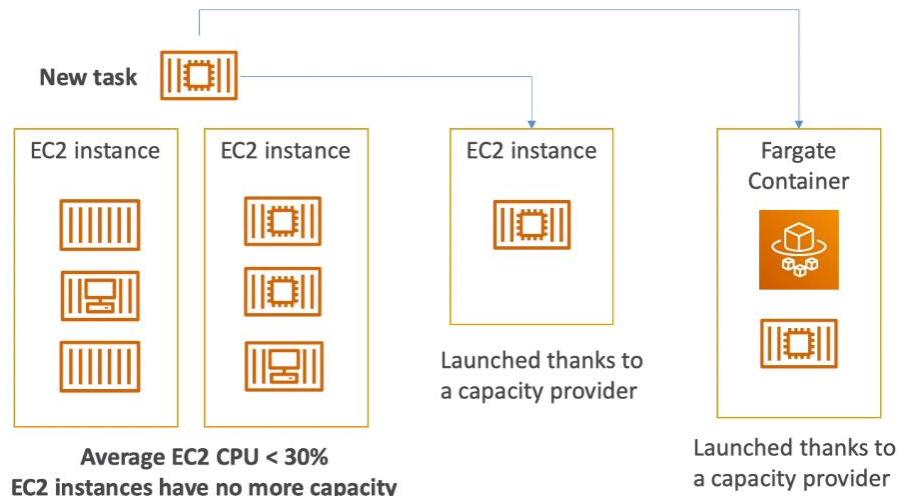
ECS - Service Auto Scaling

- CPU & RAM is tracked in CloudWatch at the ECS Service level and hence we can define Service Auto-scaling based on these CloudWatch Metrics.
- For Service auto-scaling, we can do:
 - Target Tracking: If we want to target a specific average of a CloudWatch Metric, for e.g.: CPU Utilization should be at 60% across the ECS Service.
 - Step Scaling: It is to scale based on CloudWatch Alarms.
 - Scheduled scaling: The scaling is based on predictable changes.
- This looks like auto-scaling policies we've for our auto-scaling groups, and is in fact the same because it is using the same auto-scaling service in the back-end.
- ECS service scaling at the task level is **not** equal to EC2 Auto-Scaling at the instance level.

- So, if we scale up/down our ECS service, it does not mean that our EC2 auto-scaling group at the instance level.
- This makes ECS auto-scaling pretty difficult at the Service level.
- Fargate Auto-scaling is much easier to setup coz if we setup a Fargate ECS service with service auto-scaling, it is going to be serverless and the service can scale on its own, and we don't need to worry about the infrastructure that it relies on.
- But let's say, we want to do ECS service auto-scaling & EC2 auto-scaling at the same time, we've a new feature called the Cluster Capacity Provider.

ECS - Cluster Capacity Provider

- A capacity Provider is used in association with a cluster to determine the infrastructure that a task runs on.
 - For ECS and Fargate users, the FARGATE and FARGATE_SPOT capacity providers are added automatically and this provisions us Fargate infrastructure.
 - For Amazon ECS on EC2, we need to associate the capacity provider with an auto-scaling group. Hence, Auto-Scaling group can automatically add EC2 instances when needed.
- So, when we run a task or a Service, we define a capacity provider strategy, to prioritize in which provider to run.
- This allows the capacity provider to automatically provision infrastructure for us.



- Let's take an example, we've 2 EC2 instances, and they're packed with ECS task. The average CPU utilization is <30%, but our EC2 instances have no more capacity and it is full, so we're going to create a new task. So we're going to create a new task and that task is assigned to a Cluster Capacity Provider, the Capacity provider automatically launches an ECS instance, a new EC2 instance running on ECS and it put the task there as well. Or, if we select a capacity provider of type Fargate, then this task will go on the Fargate container.

LAB

- Go to ECS under Services.
- Navigate to previously created cluster: cluster-demo
- Switch to "Capacity Providers" tab and hit on Create.
- Assign a name to your Capacity Provider.
- Select an auto-scaling group (It is created by ECS Service earlier for us).
- Enable the managed scaling.
- Keep the target capacity to be 70%, which says as the 70%capacity of our EC2 instances gets reached, then it launches new EC2 instance for us.
- Disable the Managed termination protection for now.
- Hit on Create.
- Go to your Auto scaling groups and edit the max capacity to 4 and Update.
- Go back to our cluster demo and under Services, Create a service and this time select the launch type to "Switch to capacity provider strategy". and "Add another provider". The provider will be the one we created earlier in step 9. So. these task are going to be launched based on Capacity provider.
- Select task definition to be my-httppd
- Select Cluster as cluster-demo.
- select Service type to be Replica.
- For now, keep the number of task to be 5.
- Keep the min-max healthy percent to be 0-200.
- Select deployment type to be rolling update.
- Select the Templates to be AZ Balances spread click on Next Step.
- For now, no load balancer.
- Keep unchecked Enable Service discovery integration.

21. Select Do not adjust the Service's desired count for Service auto scaling for now. Though, we can configure the auto-scaling policies, keeping Minimum number of task to be 1, desired number of task to be 10, maximum number of tasks to be 20. We can assign IAM role for Service Auto Scaling which will allow the ECS Service to scale for us, So AWSServiceRoleForApplicationAuto can be selected for now. Further, we can select Scaling Policy type to Target tracking, to track a specific metric like CPU or Memory utilization or Step scaling if we want to scale in our out based on some CloudWatch Alarms.

22. Click on Next and Review.

23. Hit on Create Service.

24. Go to Service & then move to Task Tab & there we can see 5 task are running.

25. Now, go onto tab ECS instances, 2 more instances are up and they're created by the capacity provider.

26. If we now go to the EC2 management console & go to Activity History tab, there we can see that 2 new instances have been created & they've launched by Amazon ECS auto scaling plan. Finally, go to Services under your Cluster in ECS and delete both the services and scale back the EC2 auto-scaling group back the Desired & Max capacity to 1 (or 2). this scales down our ECS cluster to 1 instance.

Summary

- ECS is used to run Docker Containers & has 3 flavours:
 - ECS "Classic": provision EC2 instances to run container onto these EC2 instances but we need to manage our own infrastructure.
 - Fargate: It is a serverless offering for ECS, so we don't need to manage any EC2 instances and these are managed for us.
 - EKS: Managed Kubernetes by AWS.
- ECS Classic
 - EC2 instances must be created.
 - We must configure the file /etc/ecs/ecs.config with a cluster name.
 - EC2 instance must run an ECS agent which allows us to register ECS agent with an ECS cluster.
 - EC2 instances can run multiple container of the same type. For this:

- We must not specify the host port, only the container port (as the host port will be dynamic).
 - We should use an Application Load balancer with the dynamic port mapping feature to direct traffic to these containers.
 - The EC2 instance must allow the traffic from the ALB on all ports.
- ECS tasks must have IAM Roles to execute actions against AWS.
- Security groups operate at the instance level not at the task level.
- ECR is used to store Docker images.
 - ECR is tightly integrated with IAM.
 - We've 2 option to do the ECR login:
 - AWS CLI v1 login command - `$aws ecr get-login --no-include-email --region eu-west-1`). And this command generates a Docker-login command and we need to run the output of the command to make it work.
 - AWS cLI v2 login command - This uses a pipe, so we run AWS ECR get login password command & we pipe it into docker login command.
 - `aws ecr get-login-password --region eu-west-1 | docker login --username AWS --password-stdin 123456.dkr.ecr.eu-west-1.amazonaws.com`
 - For Docker Push n Pull:
 - `docker push 123456.dkr.ecr.eu-west-1.amazonaws.com/demo:latest`
 - `docker pull 123456.dkr.ecr.eu-west-1.amazonaws.com/demo:latest`
 - In case, an eC2 instance (or we) cannot pull a Docker image, we need to check IAM permission.
- Fargate
 - Fargate is Serverless (no EC2 instances to manage) so if we want to run our Docker containers on serverless offerings, we can use Fargate.

- AWS provision containers for us and will assign them and in ENI (Elastic network interface). These containers are provided on the container spec we defined like no. of CPU/RAM we define.
- Fargate tasks can have IAM Roles to execute actions against AWS, so if we need a Docker container running under S3 for example, like we need to provision correct IAM Role that has access to S3.
- ECS
 - ECS does have integration with CloudWatch logs.
 - We need to setup logging at a task definition level.
 - Each container will have a different log stream.
 - The EC2 instance profile should have the correct IAM permissions.
 - We can use IAM task roles for our tasks.
 - We've different task placement strategies: Binpack, Random, Spread.
 - Service auto scaling has 3 different ways: Target tracking, Step scaling, or scheduled.
 - Cluster auto-scaling can be done through Capacity providers.

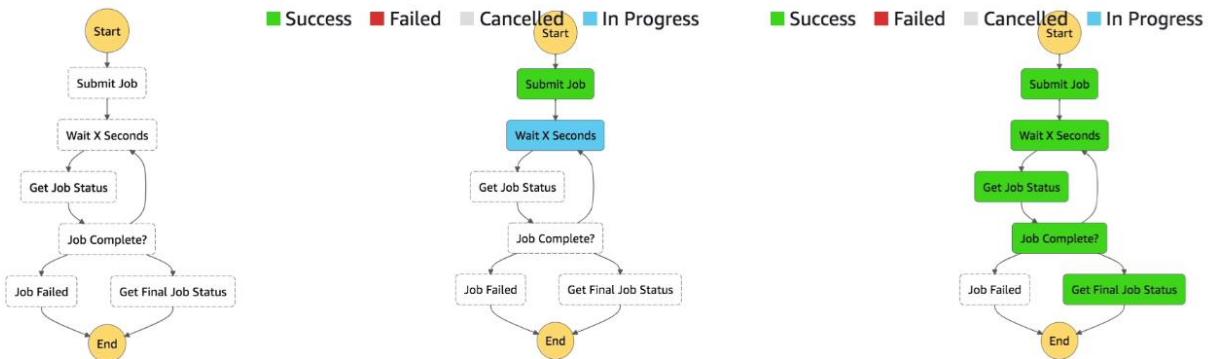
21. Other Serverless: Step Functions & AppSync

AWS Step functions

We know how Lambda function works. But sometimes we're unable to queue the Lambda function in proper order i.e., this Lambda function should go first and then this one and if error comes, we need to go to this Lambda function. So we are looking for a Workflow. So,

- Step function builds serverless visual workflow to orchestrate Lambda functions.
- To build this flow, this state machine, we represent it as a JSON document and hence it's called a JSON state machine.
- Features: Sequence (one Lambda function above the other), parallel (Doing 2 Lambda functions at the same time), conditions (if loops & so on), timeouts (if this last for more than 'x' minutes), error handling (retry & catch).

- Step functions can also integrate with EC2, ECS, On premise servers, API Gateway, SQS queues, etc.
- Maximum execution time at least for standard version is 1 year. We've the possibility to implement human approval feature.
- Use cases:
 - Order fulfillment in a factory
 - Data processing
 - Web application (like payment processing)
 - Any workflow.
- The interesting part of Step function is that we get an aspect of visualization



- Like, we design our state machine in JSON, we get a graph similar to this and hence it shows the steps that're going to happen.
- By looking in the graph, we can say what happens when a workflow starts in Step function. Once, the execution is complete, we can see the steps directly what it goes through. It helps in debugging as we can know, where the process got failed.

Error handling

- Any state can encounter runtime errors for various reasons.
 - State machine definition issues (no machine rule in a Choice state).
 - Task failures (like, an exception in Lambda function).
 - Transient issues (for example, network partition events).

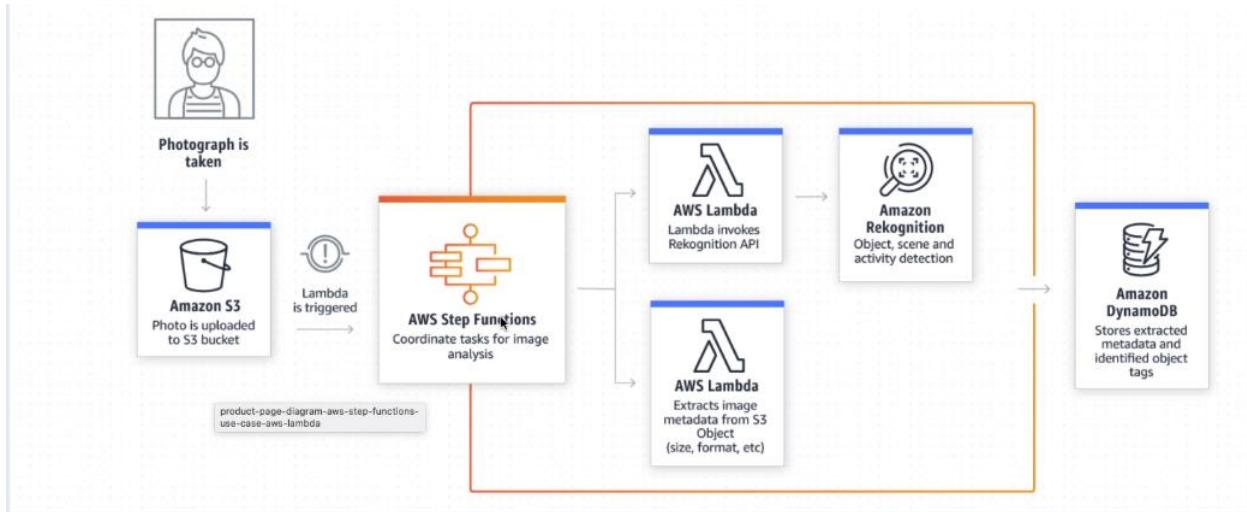
- By default when a state reports an error, AWS step functions causes the execution to fail entirely.
- We can handle those errors like by Retrying failures as in some cases error can be transient (temporary) - We can retry by defining the number of intervals we want, number of maximum attempts or the BackoffRate.
- If we know a specific type of error, we can just move on & catch it and move on.
- Best practice is to include data in the error message.

Step Functions - Standard vs. Express

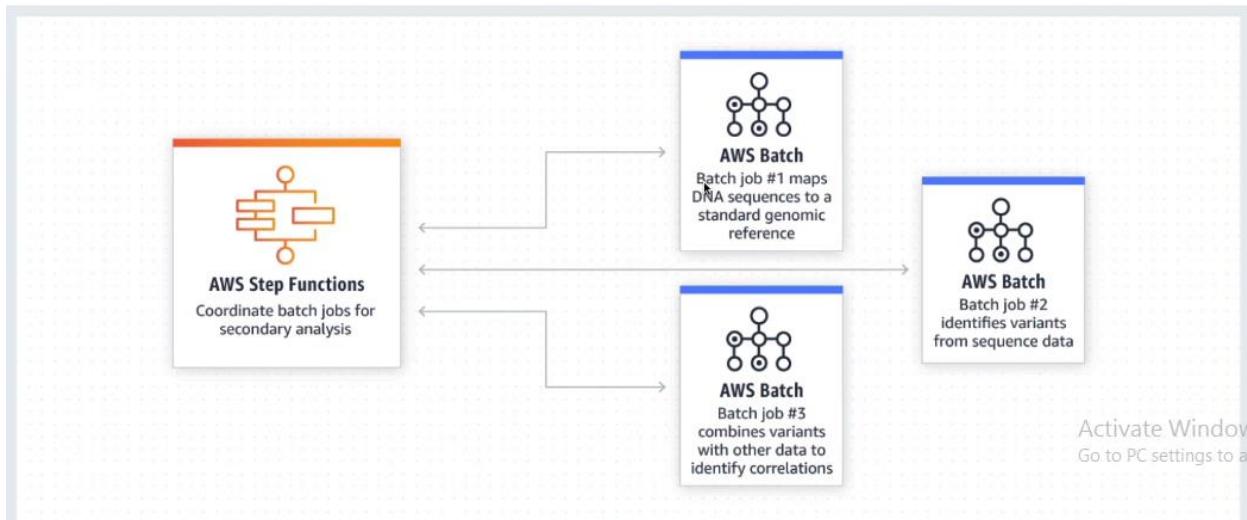
	Standard Workflows	Express Workflows
Maximum duration	1 year.	5 minutes.
Supported execution start rate	Over 2,000 per second	Over 100,000 per second
Supported state transition rate	Over 4,000 per second per account	Nearly unlimited
Pricing	Priced per state transition. A state transition is counted each time a step in your execution is completed (more expensive)	Priced by the number of executions you run, their duration, and memory consumption (cheaper)
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.	Executions can be inspected in CloudWatch Logs by enabling logging on your state machine.
Execution semantics	Exactly-once workflow execution.	At-least-once workflow execution.

So, the idea is that Express workflows are going to be much cheaper. They're intended for a large number of workflow and with small duration whereas the Standard workflow are much longer workflows and with the slower execution rate.

LAB



Using Step functions, we can transcode media files, e.g.: A photograph is taken which is uploaded into S3 bucket. A Lambda is triggered to trigger a step function workflow and this workflow contains Lambda functions and some AI tools to look at the picture and store some metadata into a DynamoDB table.



We can also do sequence batch processing jobs. this is about ETL (extract, transform & load) where our step functions says this batch job happens first and then the next one.



We can send messages from Automated workflows. The API gateway invoked the Lambda function, which in turn invokes a step function. Later, the Admin verifies & approves it. It goes into a SQS queue and then to a Lambda function & then to a S3 bucket.

1. Go to Step functions under Services.
2. Click on Get Started and choosing a Hello World example. We've a JSON document which is a State Machine definition. On the Left hand side, we've the data and on the right hand side we've visualization of data.
3. Click on Next.
4. Specify your state machine a Name.
5. We need to have an IAM role which gives permissions to our State machine to access the resources.
6. We can enable CloudWatch loggings but we'll keep it in OFF state for now.
7. Under Review functions, we can see that we gave access to create an IAM Role to access the resources & under Policy templates, we can see StepFunctionsExecutionRoleWithXRayAccess policy is included in our role's access policy.
8. Hit on Create State Machine.
9. A new state machine is created and we can start with the new execution.
10. We'll keep everything as it is (just an Input which will be initial data that'll be passed on to the state machine) and click on Start Execution.
11. If we scroll down to Graph Inspector, we can see whole path of the data into my execution of the step function running. When we further scroll down, we can see the event execution history. Even we can see the input data for every state on clicking on the triangle symbol.

12. Let's start a new execution and changing the value of IsHelloWorld example to false. If we start the execution, our Execution fails within a second. If we navigate to Step functions > State machines > HelloWorld, then we can see the role name which has been created and the type to be standard.
13. This time we'll create a different state machine defining it in another way. So go to State machines page & click on Create on Create State machine
14. Under author with code snippets, we can choose the type to be Standard or Express. Standard are long duration workflows and can run up to an year having use cases like Machine Learning, order fulfillment whereas Event driven workflows are for streaming data processing, Microservices orchestration, IoT data ingestion and other short duration, high-rate event workflows.
15. If we define machine state by selecting a template, we can see templates like "Retry failure", "Catch failure". In Retry failure, we need to replace the Lambda function with that of ours. We can set the retry parameters such as name of the error, seconds between the interval, number of attempts and the BackoffRate. In Catch failure, we set the error name and the next state of the machine which it'll move to in case it encounters that exception.

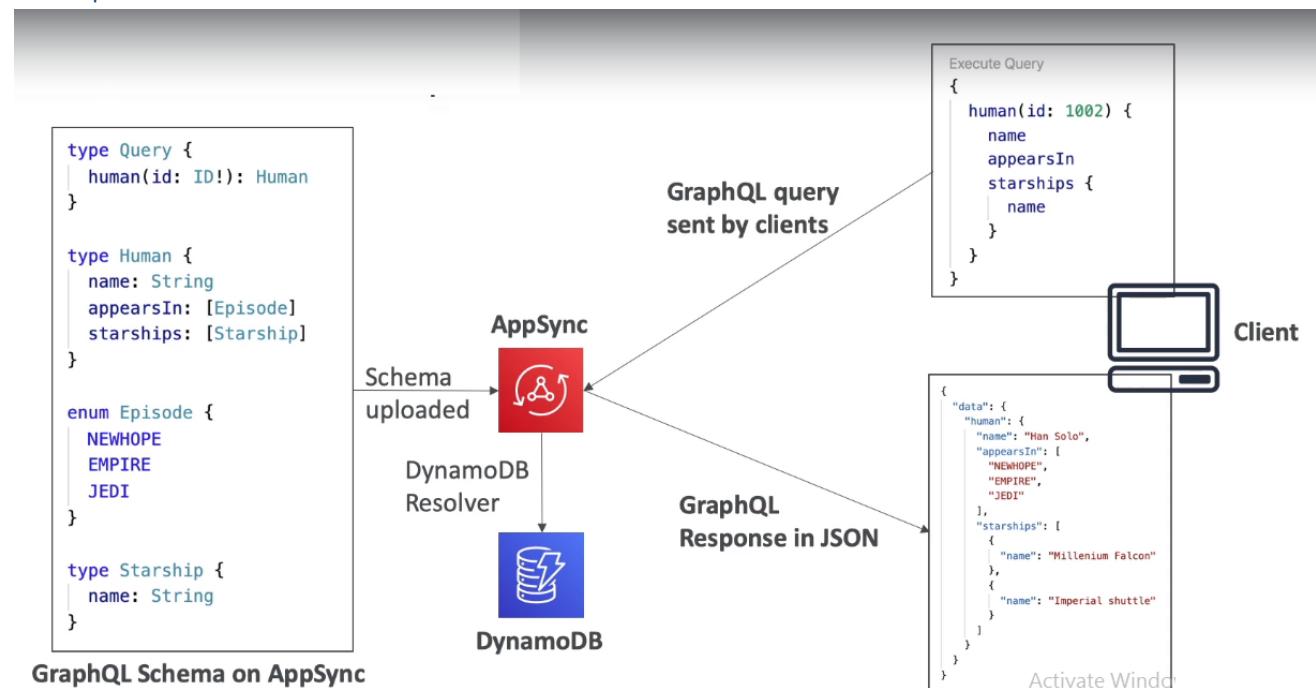
AppSync

Overview

- AppSync is a managed service that uses GraphQL.
- GraphQL makes it easy for application to get exactly the data they need. GraphQL is an open source data query & manipulation language for APIs and a runtime for filling queries with existing data. It is not specified to any specific DB or storage engine. The idea is that we need to ask for the fields we want and GraphQL returns that.
- This includes combining data from one or more sources into the graph. Hence, data sets behind GraphQL can include:
 - NoSQL data stores, Relational databases, HTTP APIs.
 - GraphQL in AppSync has direct integration with DynamoDB, Aurora, Elastic search, other sources.
 - We can get any data from anywhere and we can define Custom sources with AWS Lambda.

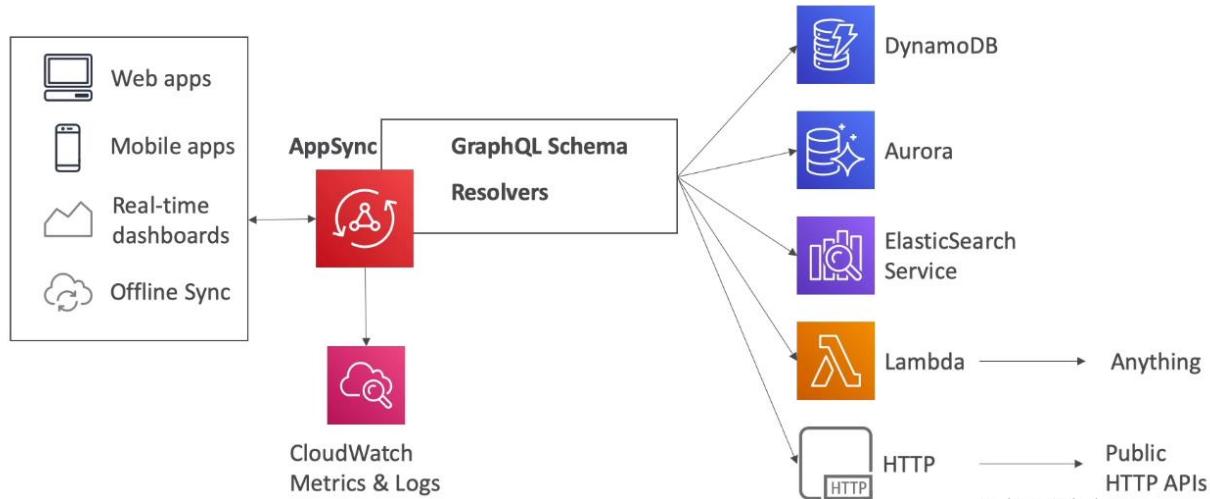
- We can use AppSync in AWS is having real-time WebSockets integration or MQTT on Websockets with real-time retrieved data. The idea is that if we're building a real-time web application that needs to access a data feed, we can use AppSync with WebSocket, ALB, or the API Gateway.
- For mobile application, if we need to have local data access & data synchronization then AppSync can be a good option in replacement with cookie to sync.
- To get started with AppSync, we just need to upload one GraphQL schema.

Example



We've AppSync in the middle, and to start with AppsSync as we know, we need to create a GraphQL schema. Then we've the client which says I want to do this query on AppSync (like here the name of the human, and the name of the fact we wanna know which movie he appears in and the Starships. Then AppSync runs its own resolver and one of the resolver may be DynamoDB i.e., fetching the data from the DynamoDB. The AppSync knows how to return the data in JSON form to the clients to comply exactly with the exact query that was asked.

Diagram



At high level, AppSync has integrations for our Web Applications, Mobile Applications, Real-time dashboard, or applications that needs offline data synchronizations. For AppSync at its core, we've GraphQL schema for upload & resolvers to tell how to fetch data. For these resolvers, we've direct integrations with DynamoDB, Aurora, ElasticSearch service or anything we want to have through our Lambda function or already existing public HTTP endpoints with HTTP integrations. AppSync is also integrated with CloudWatch Metrics & Logs to get logs/informations if anything happens in AppSync.

Security

There're 4 ways to authorize applications to interact with our AWS AppSync GraphQL API:

1. **API_KEY:** We generate these keys just like those of API Gateway and give them to users.
2. **AWS_IAM:** Using this, we can allow IAM users roles or cross account access to our absent API.
3. **OPENID_CONNECT:** If we wanted to have an integration with an OpenID connect provider and a JSONWeb Token.
4. **AMAZON_COGNITO_USER_POOL:** It is to integrate with already existing User pools and through Cognito User pools we can federate through other social login providers.

If we want to get HTTPS security on AppSync with a custom domain, the recommended solution is using CloudFront in front of AppSync.

LAB

AppSync is going to be used to get a GraphQL APIs on AWS & many other.

1. Hit on Create API.
2. For now under sample project section, hit on Event App.
3. Give a name for the AppSync API. Here, our template will automatically deploy an API and provision DynamoDB tables and IAM roles on our behalf.
4. Hit on Create.
5. Once the API is created, navigate to "Schema" which acts as the source of everything in GraphQL.
6. Next moving to Data sources which represents the DynamoDB tables that AppSync created for us. One of these tables will be containing the comments & other one will be connecting the events. So, if we click on the table & view the "Items" tab, they're empty but they get filled up once we get using the API.
7. Queries tells us how to use our APIs. We'll use a query and we've different queries here. We can do create events or list events. Click on play button  and select ListEvents and we can see the data is empty as we don't have any events yet. Click on the play button & this time select "CreateEvent" and it's created. It was a very simple query and we can also change the name, when, where and the description section.
8. Edit the description, say it to be "My Second event" and click on play button to create a new event. Hence, a new event is created.
9. Now, click on Play button & select List Events, we can see that we are returning 2 different events.
10. Go to event table & refresh on Items tab, we can see that 2 events has been created. Hence this shows the AppSync API directly insert API into our DynamoDB table. Similarly, comments can be added too (not for now).
11. Caching: It helps us reducing the number of queries we've on backend.
12. Under Settings, we get the public API that can be shared, API ID as well as the API key that people must have to use our API.
13. We can change our API name.

14. We can even change the default authorization mode. Here, we can use API keys, IAM (for user enrolls or Cross account access into our AppSync API), OpenID Connect and Amazon Cognito User Pool.
15. Further, we can add Additional authorization providers which will again give us the above 4 options.
16. We can use Logging to send CloudWatch logs.
17. We've X-Ray to have tracing in the extra console and finally Tags.
18. Monitoring provide us with Metrics information for our APIs such as number of errors, total number of request, Latency which we get for our API.
19. If we delete the API, it does not delete the DynamoDB table. So we need to delete these tables manually.

22. Advanced Identity

AWS STS - Security Token Service

- It allows us to grant limited and temporary access to AWS resources (up to 1 hour).
- Some of the important API's are:
 - AssumeRole: To assume roles within our account or cross account.
 - AssumeRoleWithSAML: return temporary credentials for users logged with SAML.
 - AssumeRoleWithWebIdentity: It returns roles/credentials for users logged in with Identity Providers (like Facebook, Google Login, etc). We don't use these & AWS recommends against using this by using Cognito Identity pool instead.
 - GetSessionToken: It is used in MFA, from a user or AWS root account user.
 - GetFederationToken: To obtain temporary credentials for a federated user.
 - GetCallerIdentity: It returns the details about the IAM user/role used in the API call. If we don't know who we're in AWS, then just we need to call STS GetCallerIdentity to get information about who we're and what's our account number.

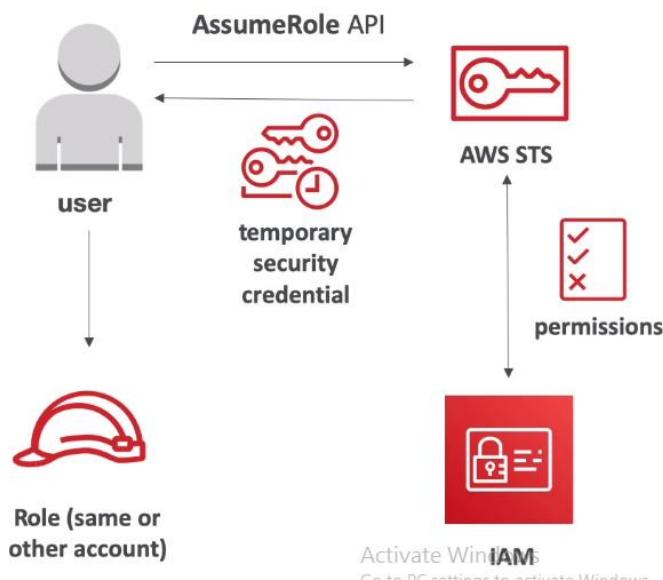
- `DecodeAuthorizationMessage`: It is used to decode an error message when an AWS API is denied.

Using STS to Assume Role

Steps

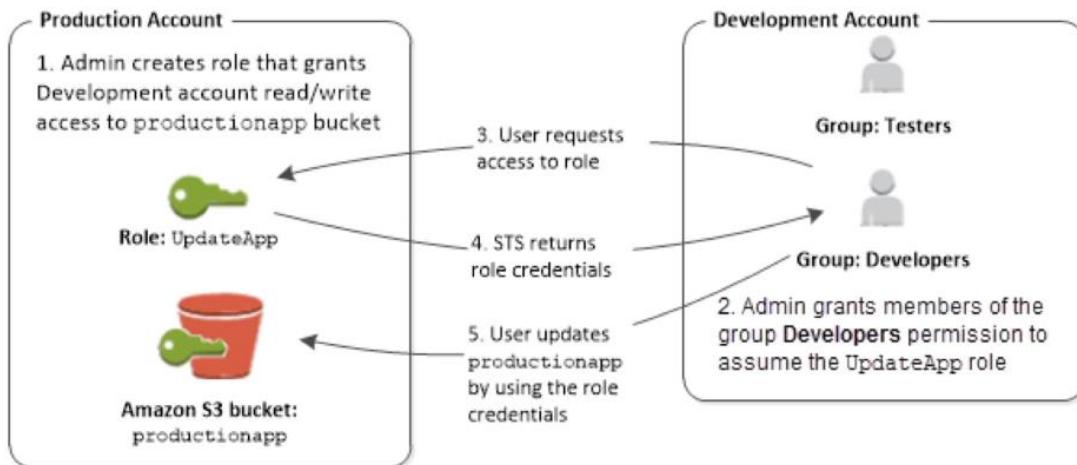
1. Define an IAM Role within our account or Cross-account.
2. Define which principal can access these IAM Role and we authorize everything with IAM policies.
3. We use STS API to do `AssumeRole` API call to impersonate (pretend) the IAM Role we've access to.
4. Temporary credentials can be valid between 15 minutes to 1 hour.

Diagram



Our user wants to access a role within the same or another account. To do so, it is going to do the `AssumeRole` API onto STS. STS checks whether the permissions are correct or not and then we return the temporary AWS credential which would allow us to act as if we were the role.

Cross account access with STS



In this, we create a role in another account and then we would write the correct permission into our own account and the target account, and finally run the AssumeRole API to access the target account. For ex: If role allows us to access S3 bucket, then we can access S3 bucket in our account.

STS with MFA

- For this, we use the `GetSessionTokenAPI` from STS to get a session token. once we are logged in with an API with MFA device.
- We then need an IAM Policy with proper IAM conditions. We then need to add `aws:MutliFactorAuthPresent:true`, which is very clear and it looks like the diagram above. In the above diagram, it depicts, this role allows us to terminate/stop instances only if we've MFA on, so `MutliFactorAuthPresent:true`.
- Reminder, GetSession Token is the API we use because it returns
 - an Access ID
 - Secret Key
 - Session Token
 - Expiration date of our credentials, so we know when to renew them.

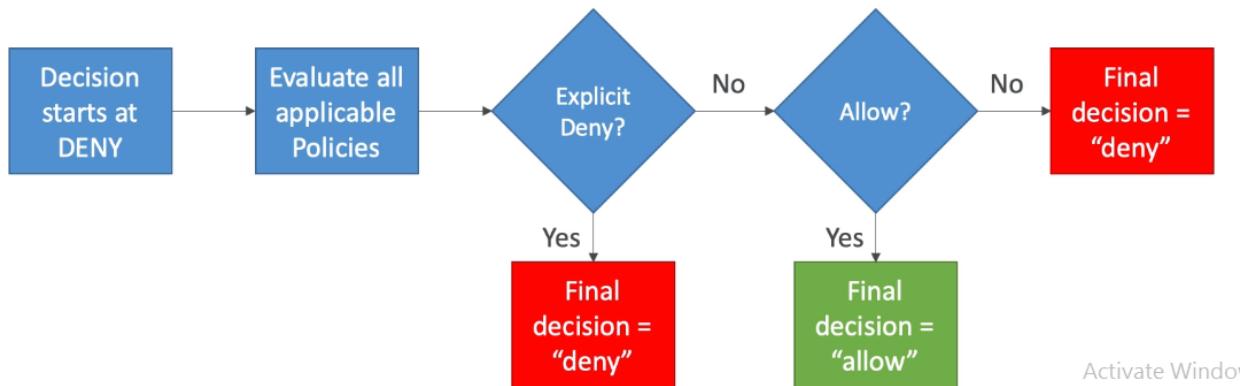
LAB

- Got to IAM under Services.
- Go to Users.

3. Under your user, navigate to security credentials, assign an MFA device.
4. Copy the assigned device ARN.
5. Go to the console.
6. `aws sts get-session-token --serial-number yourSerialNumberFromStep4 --token-code codeFromAuthenticatorApp`
7. Using these credentials, we're going to configure new profile. Type `aws configure --profile mfa` and hit enter.
8. Copy paste the Access Key, Secret Access key ID from the credentials obtained in step 6.
9. Navigate to credentials folder using `cd .aws`
10. Copy the contents of the credentials folder and add `aws_seesion_token = yourSessionTokenFromStep6`.
11. Save this.
12. Now if we type `aws s3 ls --profile mfa`, it makes an API calls against S3 using our profile.

Authorization Model

1. If there's an explicit DENY in our policy, then we end the decision with a DENY.
2. If there's an ALLOW, we end the decision with ALLOW.
3. Else, we DENY.



Let's suppose the user wants to create a DynamoDB table and the decision starts with DENY, so by default it can't do it. Then all the policies attached to the user will be evaluated. Then we look in the policy, that is there an explicit DENY in the policy or is there somewhere in the policy which says user cannot create DynamoDB table, so the final decision is DENY. Then, only then, the allow will be

evaluated. So, we'll look if there's any allow statement which allow the user to create the DynamoDB table. If Yes, then the final decision is ALLOW and if No, then the final decision is DENY.

So, if there's both an explicit DENY & explicit allow in the policy, then the DENY will win as the explicit DENY condition is evaluated first.

IAM Policies & S3 Bucket policies

- IAM Policies are attached to users, role & groups whereas
- S3 bucket policies are attached to bucket.
- When we evaluate an IAM principle, we can put an object into a S3 bucket. What is going to be evaluated is the union of IAM policy and that of S3 bucket policy.



- So, if we take an action in S3, we're going to look at IAM policy & the S3 bucket policy and all these rules are added together & it gives us the total policy which will be evaluated from the security standpoint. For ex: If we've an EC2 instance & we remove the S3 policies in the IAM policy but they're still in the S3 bucket policy authorizing the EC2 instance to do something, then we'll be still able to do stuff in S3.

Example I

- There's no IAM Role attached to EC2 instance, and it authorizes Read & Write to our S3 bucket.
- There's no S3 bucket policy attached.
- In this case, EC2 instance can read & write to S3 bucket coz the Union of these 2 policy says that our EC2 instance has the right to read & write to my bucket.

Example II

- We've an IAM Role attached to our EC2 instance, and it authorizes Read & Write to our S3 bucket.
- S3 bucket policy attached, saying explicit Deny to the IAM Role.
- In this case, EC2 instance cannot read & write to S3 bucket coz the explicit Deny and Explicit DENY has higher priority than explicit ALLOW.

Example III

- We've an IAM Role attached to our EC2 instance, with no S3 bucket permissions.
- In S3 bucket policy which is attached to the S3 bucket, there's explicit Read Write Allow to the IAM Role used by the EC2 instance.
- This time the EC2 instance can Read & Write to our bucket because of the Union policy.

Example IV

- We've an IAM Role attached to our EC2 instance which explicitly deny S3 bucket permissions.
- S3 bucket policy allows Read & Write to the IAM Role.
- In this case, EC2 instance cannot read & write to S3 bucket coz the explicit Deny and Explicit DENY has higher priority than explicit ALLOW.

Dynamic Policies with IAM

- We need to assign each user access to their /home/<user> folder in S3 bucket.
- Option I:
 - Create an IAM policy, allowing George to have access to /home/George.
 - Create an IAM policy, allowing Sarah to have access to /home/ Sarah.
 - ... This way, we'll have one policy per user and we need to create IAM policy every time we add a user.
 - Hence, this is not scalable.
- Option II:

```
{
  "Sid": "AllowAllS3ActionsInUserFolder",
  "Action": ["s3:*"],
  "Effect": "Allow",
  "Resource": ["arn:aws:s3:::my-company/home/${aws:username}/*"]
}
```

- Creating Dynamic Policy with IAM:
- We'll leverage a special policy variable \${aws:username}. At runtime, it'll be replaced by the value of the username.

Inline vs. Managed Policy

There are 3 kind of AWS Policy.

1. AWS Managed Policy
 - a. They are maintained by AWS.
 - b. Good for power users and administration
 - c. Will be updated by AWS in case of new services/new APIs.
2. Customer Managed Policy
 - a. These policies are created by us.
 - b. These are the best practices if we want to have granular control. They're reusable & can be applied to many principle.
 - c. We can have version control + Rollback, there's a central change management to see who did what.
3. Inline Policy
 - a. They're directly within a principle having strict relationship between the policy & the principle.
 - b. They don't have version control, neither can be rollback. We can't alter them very easily.
 - c. If we delete the IAM principle, the policy will be deleted.

LAB

1. Go to IAM under Services.
2. Navigate to policies.
3. Under Filter policies click on "AWS managed", we get the list of all AWS managed policies.

4. Similarly, check for Customer managed policy. These are the ones which we've created.
5. If we click on any of them, we can check the permissions, Policy versions and the Policy usage means what is using the policy.
6. To see the Inline policies, go to Users section in IAM Dashboard.
7. Click on the User name and we can see the policies (If we've created earlier. Also, there's a limit to create policy as the max size of the policy should be 2KB. Suppose if we choose EC2 service, if we add all the Access level, the size of the policy increases and at the end it restricts us to create the policy).

Granting User Permission

Granting User Permission to pass a Role to AWS Service

- When we configure AWS Service, we must pass an IAM Role to the service (this happens during the setup).
- The service will later assume the role and perform actions.
- Example:
 - When we created an EC2 instance role & assigned it to EC2 instance, we actually passed the role.
 - Passing a role to Lambda function, where we created an IAM role, and then passed it to Lambda function, so that it can call Amazon S3, etc.
 - Passing a role to an ECS task.
 - Passing a role to CodePipeline to allow it to invoke other services.
- When we want to pass a role to another AWS Service, we need an IAM Permission called iam:PassRole
- The above permission comes with another permission, iam:GetRole to view the role being passed..

Can any role be passed to any service? No. Roles can only be passed to what their trust allows. Hence, trust policy for the role is an indication to which service can assume that role.

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Sid": "TrustPolicyStatementThatAllowsEC2ServiceToAssumeTheAttachedRole",  
        "Effect": "Allow",  
        "Principal": {  
            "Service": "ec2.amazonaws.com"  
        },  
        "Action": "sts:AssumeRole"  
    }  
}
```

Here's a trust policy for an IAM role. It says only the EC2 service has the trust to allow to assume that role. As if we navigate to IAM then to Roles, we can see a column defined as role name & other as trusted entities which allows specific service to assume the role. On clicking on any of the role like "AmazonS3ReadOnlyAccessForCodeDeploy", then going to Trust Relationship tab and then clicking on "Show policy document", we can see the trust policy which allow the EC2 service to assume that role.

Example

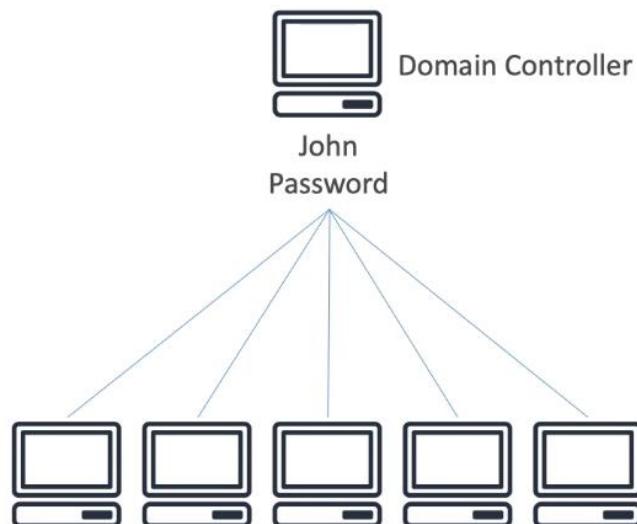
```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:*"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": "iam:PassRole",  
            "Resource": "arn:aws:iam::123456789012:role/S3Access"  
        }  
    ]  
}
```

This IAM policy allows us to do anything on EC2 such as creating instances, terminating them, etc. The second part of the statement allows us to pass a role,

and that role is called S3Access. It means, only role that can be assigned to our EC2 instances is S3Access role. And for this, we need an action, IAM pass role to do it.

Microsoft Active directory (AD)

- It is a software that is found on any windows software with AD Domain services.
- It is a Database of objects, and the objects can be User Accounts, Computers, Printers, File Shares, Security Groups. So, all our users managed within the Microsoft ecosystem on premise is going to be managed by Microsoft Active Directory.
- There's a centralized security management, create accounts, assign permission, etc.
- All the objects will be organized into a tree and a group of tree is called a forest.



Let's suppose, we've a domain controller, we're going to create an account on it with a username & password. The idea is that, all the other Windows Machines that's going to be connected to our network are going to be connected to our domain controller. So, if we're using username ABC & password XYZ on the first machine, it is going to look up into the controller and say, we've that login and hence it allows to login from that machine. Hence all these machines are going to

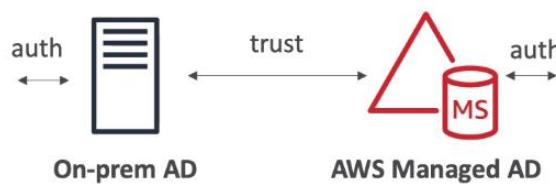
be connected to our domain controller and then allow us to have the users accessible on any single machine.

AWS Directory Services

It provides us a way to create an Active directory on AWS. It is of 3 types:

1. AWS Managed Microsoft AD:

- a. It is used to create our own Active directory in AWS. We can manage users locally & it supports MFA.



- b. We can establish "trust" connections with our on-premise AD where we have our own users there as well. It means our AD can be trusted with on-premise AD and vice-versa. It means if our users use an account that is not managed by AWS, it can go & look up the accounts into on-premise active directory. Similarly, if an on-premise directory user goes and authenticate to our on-premise Active Directory using the AWS accounts, it can be trusted to go & look it up. So the users are going to be shared between On-premise Active Directory and AWS.

2. AD connector:

- a. It is a Directory Gateway (proxy) to re-direct to on premise AD.
- b. Users are solely managed into the o-premise AD.



In this example our AD connector is just acting as a proxy. So if the user is going to authenticate with the AD connector, it is going to proxy the request back to our on-premise AD. In the first case (in AWS Managed Microsoft AD), users were sitting on the on-premise

AD whereas in AD connector, it connects, it proxies the queries, the connection request back to our on-premise Active Directory. The only place where we are going to manage users is on-premise AD.

3. Simple AD: It is an AD compatible managed directory on AWS. It does not use Microsoft directory and it cannot be joined with an on-premise Active Directory.



So, if we don't have an On-premise AD and we need an active directory for our AWS Cloud, then we can have just a simple AD as a standalone. Using Active Directory, we can create EC2 instances that're going to be running Windows & these Windows instances can join the domain controller for our network & share all the credentials.

LAB

1. Go to Directory Service under AWS.
2. Click on "Set up directory". There's an option called "Amazon Cognito user pool" which redirects to the Cognito Service so we don't count this into Directory Services.
3. So, we've a AWS Managed Microsoft AD where we are going to have an Active Directory where it is going to be integrated with AWS Cloud and can have trust relationship with our On-premise directory.
4. Click on Next. We can see 2 editions. The standard edition is optimized for 30,000 objects whereas the Enterprise edition is for 50,000 objects.
5. If we choose AD connector under Directory types, it is a proxy for redirecting proxy request to our existing Microsoft Active Directory on-premise. It is designed for connector up to 500 users and similarly large AD connector can support up to 5,000 users.

23. AWS Security & Encryption

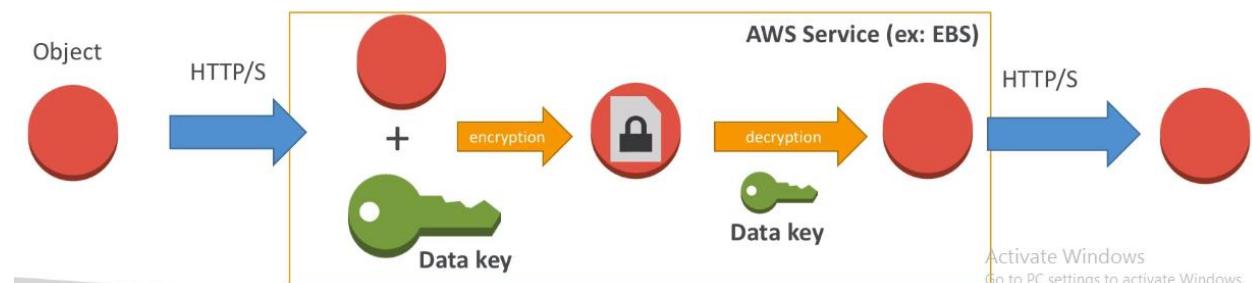
Encryption

Encryption in Flight (SSL)



- It is required because if we sent credit card details to the server, we don't want our credit card number to be shown on network . So, in Encryption in flight, all our data is encrypted an send to server and decrypted on server then.
- SSL certificates help with the encryption (HTTPS).

Server Side Encryption at REST



- Data is encrypted after being received by the server.
- Data is sent before being sent.
- Data is stored in the form of a key usually known as Data Key.
- The encryption/decryption keys must be managed, usually called KMS and server must have access to KMS.
- The object is getting transferred to EBS. EBS uses a data key to perform encryption and its stored in encrypted form. Once we need to retrieve the data, SWS decrypts the data using the data key and we get the decrypted data over HTTP/HTTPS.

Client Side Encryption

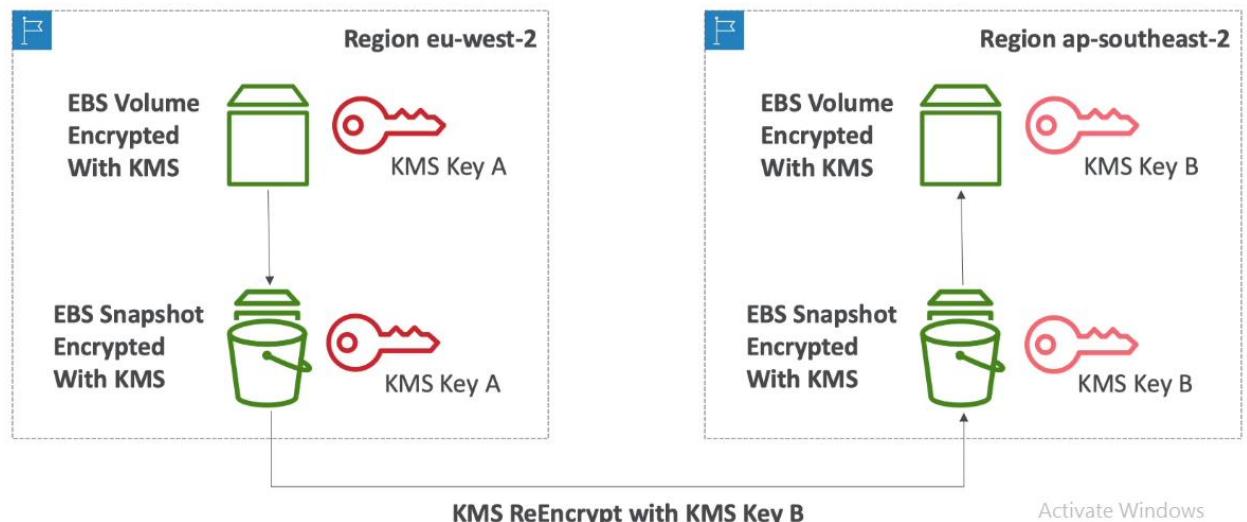


- Data is encrypted by the client and never decrypted by the server.
- We've an object & client side data key and we encrypt the data client side. We store the data on AWS (like S3, FTP, etc.). when client receives it, it gets encrypted data. If the client retrieves the data key, then it decrypts the object.

KMS

- With KMS, we can manage who can access what data and AWS manages all the encryption keys for us.
- We can fully manage the keys & the policies. So we can:
 - Create keys
 - Define rotation policies
 - Disable/Enable them.
- We can audit the key usage from CloudTrail.
- It is fully managed with IAM for Authorization.
- It is integrated with:
 - EBS: to encrypt volumes
 - S3: server side encryption of objects.
 - Redshift: Encryption of data
 - RDS: encryption of data
 - SSM: Parameter store
- KMS can be used with CLI/SDK
- There's 3 types of CMK:
 - AWS Managed Service Default CMK: free
 - User Keys created in KMS: \$1 per month

- User Keys imported (must be 256-bit symmetric key): \$1 per month
- Also, we need to pay for API calls to KMS (\$0.03 / 10,000) calls.
- The significance of KMS is that, CMK is used to encrypt the data & can never be retrieved by the user. CMK can be rotated for extra security.
- It is advised never to store password in plain text, especially in code.
- Encrypted secrets can be stored in code/environment variables.
- KMS has a limit, so we can encrypt up to 4KB of data per call.
- If data > 4KB, we need to use Envelop encryption
- To give access to KMS to someone:
 - Make sure the key policy allows the user.
 - Make sure the IAM policy allows the API calls.
- KMS keys are limited to region. It means if we create a KMS key in region A, it cannot be transmitted B.



- Say, we've encrypted EBS volume, with KMS in region eu-west-2. We need to copy the volume across new region. So, firstly we'll create a snapshot of our volume and then we'll copy the encrypted snapshot to a new region but we need to specify new KMS key. Finally, when we recreate the volume from snapshot then the volume will be encrypted with new KMS Key.

```
{
  "Sid": "Allow use of the key with destination account",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::TARGET-ACCOUNT-ID:role/ROLENAMESPACE"
  },
  "Action": [
    "kms:Decrypt",
    "kms>CreateGrant"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": "ec2.REGION.amazonaws.com",
      "kms:CallerAccount": "TARGET-ACCOUNT-ID"
    }
  }
}
```

KMS Key Policy

- If we need to copy snapshots across accounts,
 - We need to create a snapshot, encrypted with our own CMK.
 - We need to attach a KMS Key policy to authorize cross-account access.
 - Share the encrypted snapshot.
 - In the target account, we'll create a copy of the snapshot, encrypt it with a KMS key in our account.
 - Create a volume from snapshot.

Customer Master Key

It is of 2 types:

- Symmetric (AES-256 Keys)
 - It is the first offering of KMS.
 - It is a single encryption key that's used to Encrypt & Decrypt.
 - AWS Services that're integrated with KMS uses Symmetric CMK.
 - Necessary for envelop
 - In this type, we never get access to the keys unencrypted, we need to call KMS API to use.
- Asymmetric (RSA & ECC key pair)
 - We've public key for encryption and a private key for decryption
 - It is used for Encrypt/Decrypt, or sign/Verify

- The public key can be downloaded but access to the private keys are impossible.
- Use Case: Encryption outside of AWS by users who cannot call the KMS API.

Uses

1. Anytime we need to share sensitive information
 - a. Database Password
 - b. Credentials to external service.
 - c. Private key of SSL certificate

Key Policies

- If we do not specify the Key policy, no-one can access the key.
- Default KMS Key policy:
 - It is created if we do not provide a specific KMS Key policy.
 - It gives complete access to the root user = entire AWS account. It means entire account has right to access KMS key on top of that IAM policy is available to it.
 - Gives access to the IAM policies to the KMS key.
- Custom KMS Key Policy
 - We need to define the users, roles, that can access the KMS key.
 - We need to define who can administer the key.
 - It is helpful when we do cross-account access of KMS Key.

Request Quotas

- When we exceed the request quotas, we get a throttling exception:

You have exceeded the rate at which you may call KMS. Reduce the frequency of your calls.
 (Service: AWSKMS; Status Code: 400; Error Code: ThrottlingException; Request ID: <ID>)

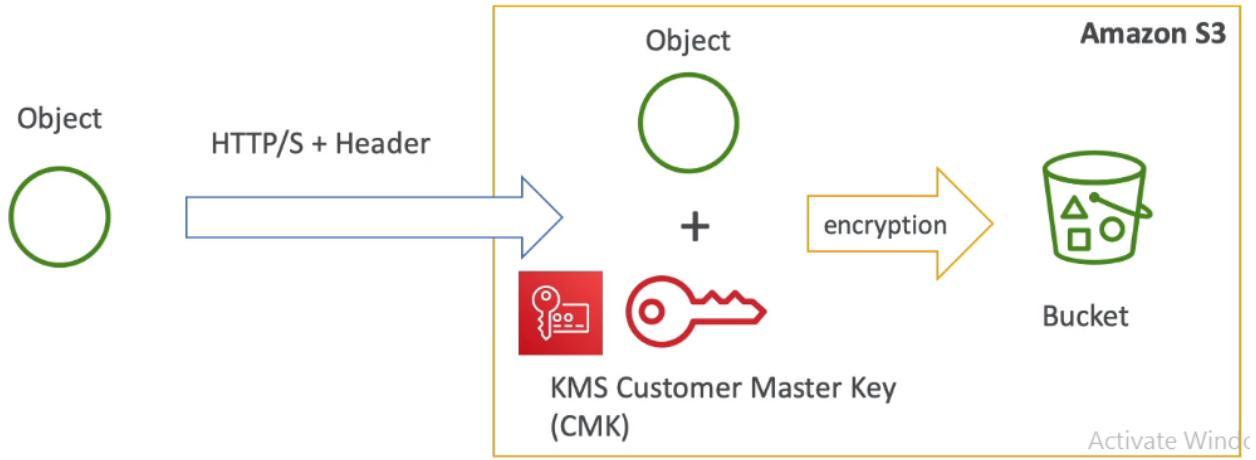
- To respond to these exception, we use exponential backoff.

API operation	Request quotas (per second)
Decrypt Encrypt GenerateDataKey (symmetric) GenerateDataKeyWithoutPlaintext (symmetric) GenerateRandom ReEncrypt Sign (asymmetric) Verify (asymmetric)	<p>These shared quotas vary with the AWS Region and the type of CMK used in the request. Each quota is calculated separately.</p> <p>Symmetric CMK quota:</p> <ul style="list-style-type: none"> • 5,500 (shared) • 10,000 (shared) in the following Regions: <ul style="list-style-type: none"> • us-east-2, ap-southeast-1, ap-southeast-2, ap-northeast-1, eu-central-1, eu-west-2 • 30,000 (shared) in the following Regions: <ul style="list-style-type: none"> • us-east-1, us-west-2, eu-west-1 <p>Asymmetric CMK quota:</p> <ul style="list-style-type: none"> • 500 (shared) for RSA CMKs • 300 (shared) for Elliptic curve (ECC) CMKs

- For cryptographic operations (Encryption & Decryption), they share a quota.
- This includes any service made by AWS service on our behalf (Ex: SSE-KMS), every time AWS will use that key for us and then it'll be part of that quota.
- In case of Throttling exception, if we're using GenerateDataKey API, we can use DEK caching from the encryption SDK to reduce the number of API calls done to AWS.
- We can also request a Request Quotas increase through API or AWS support.

SSE - KMS

- Allows us to do server side encryption in S3.
- Encryption keys are handled and managed by KMS.
- Advantages of KMS are:
 - We've control over the Key Policy and IAM that who can use that key.
 - We also get audit trail into CloudTrail.
- Object will be encrypted Server side.
- For SSE-KMS to work, we must set the header "x-amz-server-side-encryption":"aws:kms" when we do a HTTP/HTTPS request to S3.



The object is uploaded to S3, we reference the KMS CMK that we want to use to encrypt the object with, & finally the object is encrypted into the bucket.

Deep Dive

We are uploading some big files into S3, but KMS encrypts only up to 4 KB data. So,

- SSE-KMS leverages the GenerateDataKey API & Decrypt KMS API calls.
- As soon as these API are called by S3, then it can be shown up in CloudTrail which helps in security logging.
- To perform SSE-KMS, we need:
 - A KMS Key policy that authorizes the user/role to use the key.
 - An IAM policy that authorizes access to KMS.
 - Else, we get an Access Denied error as we need access to the GenerateDataKey and Decrypt API calls.
- For every S3 calls to KMS and encrypts/decrypts an object, then it counts against our SSE-KMS limits.

S3 Bucket policies

Force SSL

```
{  
    "Id": "ExamplePolicy",  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowSSLRequestsOnly",  
            "Action": "s3:*",  
            "Effect": "Deny",  
            "Resource": [  
                "arn:aws:s3:::awsexamplebucket",  
                "arn:aws:s3:::awsexamplebucket/*"  
            ],  
            "Condition": {  
                "Bool": {  
                    "aws:SecureTransport": "false"  
                }  
            },  
            "Principal": "*"  
        }  
    ]  
}
```

- We can force SSL connection to our S3 bucket using a condition called aws:SecureTransport = false so that we can deny anything that has Secure Transport = false or anything that is using HTTP.
- Or, we can allow only HTTPS stuffs i.e., aws:SecureTransport = True. It has little security flaws as it'd allow anonymous GetObject if using SSL.

Force encryption of SSE-KMS

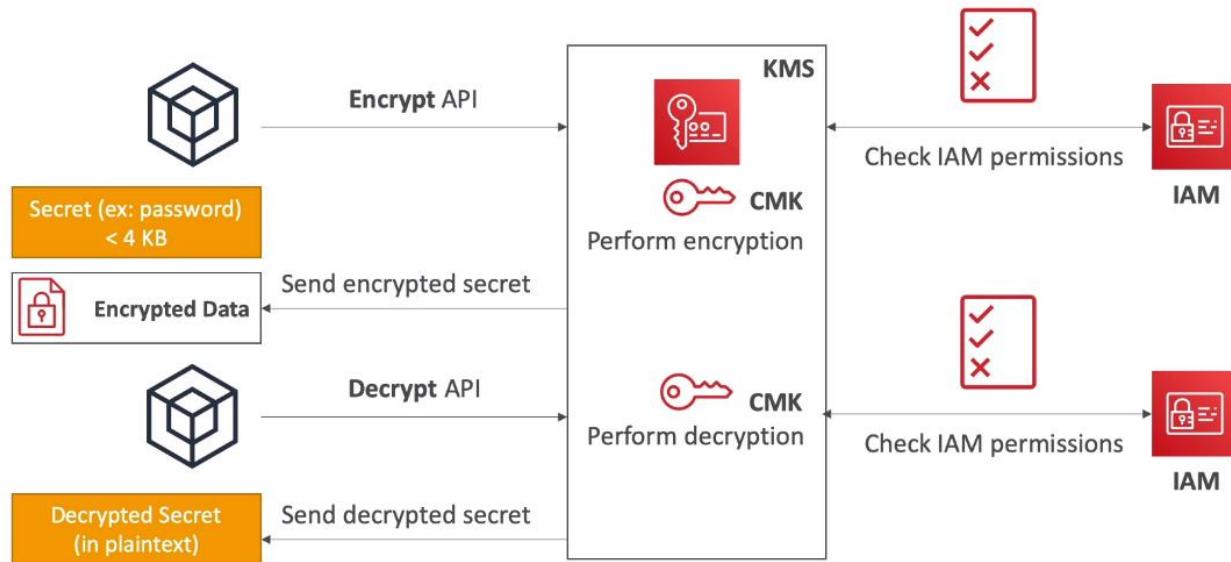
```
{  
    "Version": "2012-10-17",  
    "Id": "PutObjPolicy",  
    "Statement": [  
        {  
            "Sid": "DenyIncorrectEncryptionHeader",  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::<bucket_name>/*",  
            "Condition": {  
                "StringNotEquals": {  
                    "s3:x-amz-server-side-encryption": "aws:kms"  
                }  
            }  
        },  
        {  
            "Sid": "DenyUnEncryptedObjectUploads",  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::<bucket_name>/*",  
            "Condition": {  
                "Null": {  
                    "s3:x-amz-server-side-encryption": true  
                }  
            }  
        }  
    ]  
}
```

Activate Window

Guru PC settings itrac

- Deny Any incorrect encryption header. We need to check if the encryption header is set to aws:kms
- We need to deny whenever there's no encryption header and we need to ensure that encryption header are not uploaded un-encrypted.

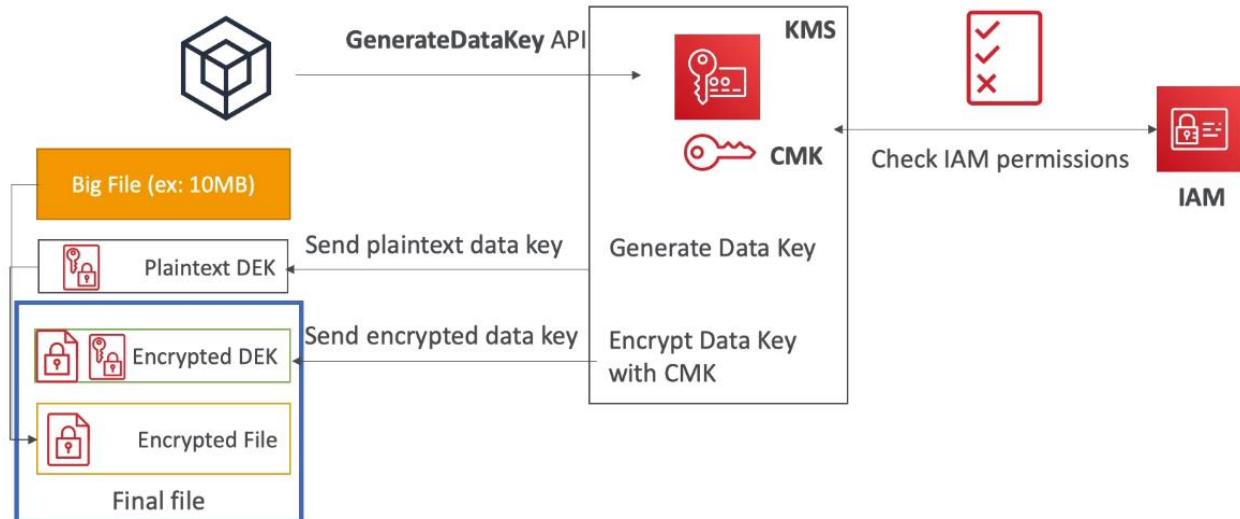
API - Encrypt & Decrypt



Let's say we've a password of less than 4 KB. We send this to KMS using Encrypt API using SDK/CLI. We also specify the CMK we used in KMS. Then, it needs to check IAM if it has the right permission and if it has, then Encryption is done successfully. Hence KMS sends us back the encrypted Data.

To decrypt it, we use the CLI/SDK and then KMS automatically understands which CMK was used for encryption and look for it for performing Decryption. It checks with IAM that if it has correct permission to perform decryption, and then it sends the decrypted secret in plain text.

- KMS Encrypt API call has a limit of 4 KB.
- If we want to encrypt >4KB, then we use Envelope Encryption.
- The API which is used to generate large amount of data is `GenerateDataKey` API.



We've a KMS service, and we want to encrypt a big file (say 10MB). Using the SDK, we call the `GenerateDataKey` API and we'll specify a CMK. IAM checks the permission if we can generate the data Key and then it generates the data key for us and sends us back the plain text version of it. Using the plain text Datakey, we can encrypt the file at the client side. Then we build an envelope around it which is the final file. It contains the encrypted version of DEK and the Encrypted File and since both the file are encrypted under one SDK, it is called envelope encryption.

To decrypt the envelope, we'll call the `Decrypt` API on the Envelope file. We can pass only 4 KB of data & will check the IAM permission. Later, we decrypt the data key, hence we get the plain text DEK and encrypted file and we decrypt the file together and hence we have the decrypted file.

API Summary

- `Encrypt` API - Encrypt up to 4 KB of data through KMS.
- `GenerateDataKey` API - Generates unique symmetric data key
 - Returns plain text of data key
 - Returns encrypted version of data key with the CMK that we specify.
- `GenerateDataKeyWithoutPlainText` -
 - Generates data key that can be used in near future (not immediately)
 - DEK that is encrypted under the CMK that we specify must be decrypt later.

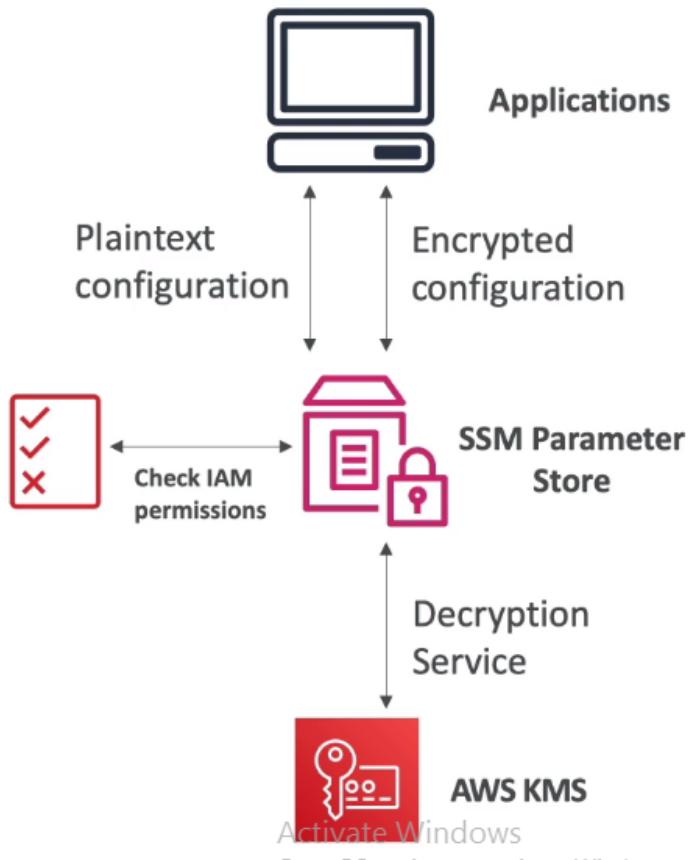
- Decrypt API: Decrypt up to 4 KB of data (including Data Encryption Keys).
- GenerateRandom API: it returns a random byte string.

Encryption SDK

- AWS Encryption SDK implemented Envelop encryption for us.
- Encryption SDK also exists as a CLI tool we can install.
- SDK encryption exist for Java, Python, C & JavaScript
- It has a feature of Data Key Caching
 - We can re-use the data caching key instead of creating new ones for each encryption.
 - Helps with reducing the number of calls to KMS with a security trade-off.
 - If we implement Data Key caching, we use LocalCryptoMaterialsCache which indicate the maximum size of cache. (max age, max bytes, max number of messages that can be encrypted by DEK before moving on to next DEK).

SSM Parameter Store

- It securely stores configuration and secrets.
- We've optional seamless encryption with KMS. It means we've our secrets stored and have them KMS encrypted directly from SSM parameter store.
- It is a serverless service, scalable, durable, and SDK is super easy to use.
- We've versioning of our configuration and secrets.
- All the security & configuration management is done using path & IAM.
- We can use notification with CloudWatch events.
- Integration with CloudFormation.



We've an application, and the parameter is stored in the parameter store. If we request that configuration then the Parameter Store will check IAM permission checks the correct permission. Or, we can ask for the encrypted configuration in which case the Parameter Store will also ask with IAM and will check the KMS permission and if so, then calls the Decrypt API from the KMS Service to give our Decrypted Secret.

Hierarchy

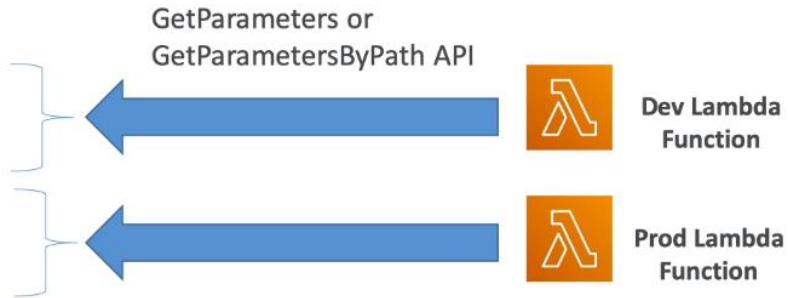
- This is the way to store the parameters in the Parameters Store. We can create a hierarchy like:

- /my-department/
 - my-app/
 - dev/
 - db-url
 - db-password
 - prod/
 - db-url
 - db-password
 - other-app/
- /other-department/

- It is sort of a folder structure like that of a file system.
- Using the parameter store, we can reference secrets from Secrets Manager like:
[/aws/reference/secretsmanager/secret_ID_in_Secrets_Manager](https://aws/reference/secretsmanager/secret_ID_in_Secrets_Manager).
- We can also reference parameter from the parameter store directly from AWS like if need to retrieve the latest AMI ID:
[/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2](https://aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2).

- /my-department/

- my-app/
 - dev/
 - db-url
 - db-password
 - prod/
 - db-url
 - db-password
- other-app/



- We've Lambda function and if we want to access the Dev parameters, then we set an environment variable and then the Lambda function will get the parameters and if we've a Prod Lambda function with another environment variable, then it automatically retrieves the Prod values.

Standard vs. Advanced Parameter Tier

	Standard	Advanced
Total number of parameters allowed (per AWS account and Region)	10,000	100,000
Maximum size of a parameter value	4 KB	8 KB
Parameter policies available	No	Yes
Cost	No additional charge	Charges apply
Storage Pricing	Free	\$0.05 per advanced parameter per month
API Interaction Pricing (higher throughput = up to 1000 Transactions per second)	Standard Throughput: free Higher Throughput: \$0.05 per 10,000 API interactions	Standard Throughput: \$0.05 per 10,000 API interactions Higher Throughput: \$0.05 per 10,000 API interactions

Parameter Policies (for Advanced Parameters)

- Allow us to assign TTL to a parameter (expiration date) to force updating/deleting sensitive data such as passwords.
- We can assign multiple policies at a time.

Expiration (to delete a parameter)

```
{
  "Type": "Expiration",
  "Version": "1.0",
  "Attributes": {
    "Timestamp": "2020-12-02T21:34:33.000Z"
  }
}
```

- Expiration (To delete a parameter): The parameter expires on Dec 2020

ExpirationNotification (CW Events)

```
{
  "Type": "ExpirationNotification",
  "Version": "1.0",
  "Attributes": {
    "Before": "15",
    "Unit": "Days"
  }
}
```

- ExpirationNotification: to send a notification to CloudWatch Events 15 days before the expiration happens.

NoChangeNotification (CW Events)

```
{
  "Type": "NoChangeNotification",
  "Version": "1.0",
  "Attributes": {
    "After": "20",
    "Unit": "Days"
  }
}
```

- NoChangeNotification: If the parameter does not change in 20 days, then send a notification through CloudWatch Events.

Secret Manager

- It helps us store secrets in AWS.
- Capability to force rotation of secrets every X days.
- We can automate generation of secrets on rotation using Lambda.
- We can integrate Secret Manager with RDS to synchronize secrets between DB and Secrets Manager.
- The secrets are encrypted using KMS.
- It is mostly meant for RDS integration.

SSM Parameter Store vs. Secret Manager

- Secret Manager

- It is an expensive service.
- Automatic rotation of secrets using Lambda.
- Direct integration with RDS, Redshift, DocumentDB.
- KMS encryption is mandatory as we cannot have plain texts in secrets manager.
- Can integrate with CloudFormation.
- Parameter Store
 - It is a simple API.
 - There's no secret rotation built-in.
 - KMS encryption is optional as we can store plain texts in Parameter Store.
 - It can be integrated with CloudFormation.
 - We can call API to get secrets from Secrets Manager through SSM Parameter store API.

CloudWatch logs - Encryption

- We can encrypt CloudWatch logs with KMS keys.
- Encryption is enabled at the log group level. We can either associate a CMK with an existing Log Group or we can create a new Log Group with a CMK.
- We cannot associate a Log Group with CMK using CloudWatch console.
- We've to use CloudWatch Logs API for the CLI & SDK.
 - associate-kms-key: if the log group already exist.
 - Create-log-group: if the log group doesn't exists yet and directly associate it with a KMS key.

CodeBuild Security

- Since CodeBuild is outside the VPC, but we can launch CodeBuild inside VPC to access VPC resources.
- Do not store secrets in CodeBuild as plain text in Environment Variables.
- Instead,
 - we can have Environment Variables that reference Parameter Store's parameters.
 - Or, we can have Environment Variables that can reference secrets manager secrets.

24. Other Services

Simple Email Service

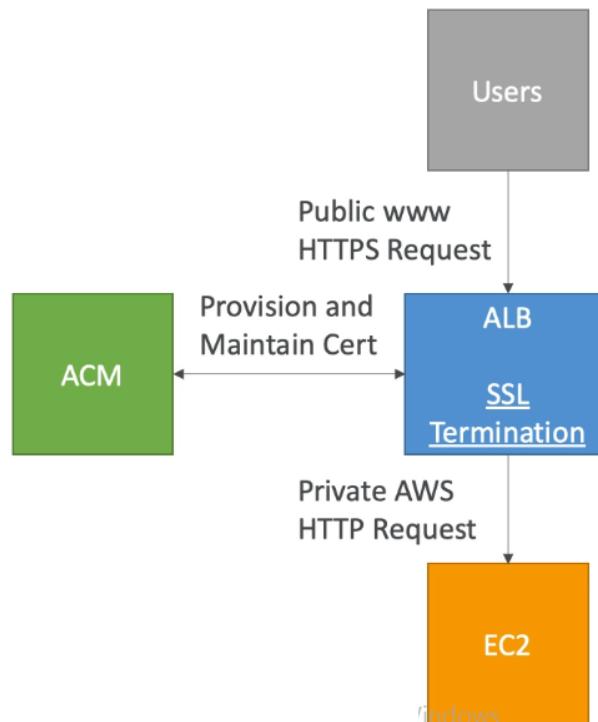
- It is used to send emails using:
 - SMTP interface
 - AWS SDK
- We can receive emails and it can be integrated with:
 - S3
 - SNS
 - Lambda
- To send/receive emails, we need IAM Permission.

Summary of Databases

- RDS: it is a relational DB which is good for online transaction processing.
 - PostGRE SQL, MySQL, Oracle, etc.
 - Aurora+Aurora Serverless.
 - These DB needs to be provisioned, hence we cannot scale horizontally rather vertically.
- DynamoDB: It is a NoSQL Database.
 - It is a managed DB having key-Value store.
 - We can store documents, objects of up to 400KB.
 - It is serverless and we can provision RCU/WCU
 -
- ElastiCache: it is an in-memory DB.
 - Redis/Memcached
 - It is used for caching the data coming from other DB, caching queries, or caching session data.
 - We need to provision it.
- Redshift: It is used for OLAP-Online Analytics Processing.
 - Data Warehousing/Data Lake
 - Analytics queries
 - It needs to be provisioned in advance.
- Neptune: It is a Graph Database.
- DMS: It is a Database Migration Service
- DocumentDB: it is a managed MongoDB on AWS.

ACM - AWS Certificate Manager

- It is used to host public SSL certificates in AWS. It can be done in 2 ways:
 - We can buy our own certificates and upload them using CLI.
 - We can have ACM provision & renew public SSL certificates for us (free of cost).
- ACM loads SSL certificates on following integration. It loads them onto:
 - Load Balancers including the Load Balancer that we create using Elastic Beanstalk.
 - CloudFront Distributions.
 - API or API Gateway.
- SSL certificate is painful to manage coz we need to renew them when they expire and we need to load them correctly, need to ensure security. So, ACM is great to leverage in our AWS infrastructure.



- Say, we've our users connected to ALB and that ALB has SSL certification. It means the users are connected to public internet talking and they're talking via HTTPS but after termination these request will be transformed to HTTP. So, SSL certificate gets loaded to ALB using ACM. ACM loads and maintains the certificate on ALB. Now, when HTTPS request is made to ALB, then ALB

makes private AWS network (HTTP request not HTTPS). So only ALB deals with HTTPS not our EC2 instances. It decreases our CPU cost on EC2 instances coz they don't have to decrypt & encrypt payload. Also, we don't need to manage SSL certificates for EC2 instances.