

**developer-associate-notes**

**Tina Bu**

# **AWS Developer Associate Notes**

- AWS Developer Associate Notes
  - Exam Details
  - 1 IAM
  - 2 EC2
    - 2.1 EC2 101
    - 2.2 Security Group
    - 2.3 EBS Volume
    - 2.4 Command Line CLI
    - 2.5 CloudWatch
    - 2.6 EFS - Elastic File System
    - 2.7 Spread placement groups
    - 2.8 Load Balancer
  - 3 Databases
    - 3.1 Relational Database
    - 3.2 RDS (OLTP)
    - 3.3 Non-relational databases
    - 3.4 DynamoDB
    - 3.5 Data Warehousing - Redshift (OLAP)
    - 3.6 ElastiCache
    - 3.7 Aurora
  - 4 S3
    - 4.1 S3 101
    - 4.2 Tiered Storage
    - 4.3 Security
    - 4.4 Encryption
    - 4.5 Performance Optimization
  - 5 Networking & Content Delivery
    - 5.1 DNS 101
    - 5.2 Route53
    - 5.3 CloudFront
    - 5.4 API Gateway
  - 6 Serverless Computing
    - 6.1 Serverless 101
    - 6.2 Lambda

- 6.3 Alexa Skill
- 6.4 Step Functions
- 6.5 X-Ray
- 7 Other AWS Services
  - 7.1 SQS - Simple Queue Service
  - 7.2 SNS - Simple Notification Service
  - 7.3 SES - Simple Email Service
  - 7.4 Kinesis
  - 7.5 Elastic Beanstalk
  - 7.6 Systems Manager Parameter Store
- 8 Developer Theory
  - 8.1 CI/CD
  - 8.2 CodeCommit
  - 8.3 CodeDeploy
  - 8.4 CodePipeline
  - 8.5 Docker and CodeBuild
  - 8.6 CloudFormation
- 9 Monitoring

## Exam Details

- Exam Domains
  - 22% Deployment (CI/CD, Beanstalk, prepare deployment package, serverless)
  - 26% Security (make authenticated calls, encryption, application authentication and authorization)
  - 30% Developing with AWS Services (code serverless, functional requirements -> application design, application design -> code, interact with AWS via APIs, SDKs, AWS CLI)
  - 10% Refactoring (optimize application to best use AWS)
  - 12% Monitoring and Troubleshooting (write code that can be monitored, root cause analysis on faults)
- Exam Format
  - Multiple choice or multiple response questions
  - 130 min in length, 65 questions
  - scenario based questions
  - Score 100 - 1000, passmark is 720
  - \$150
  - certification valid for 2 years
- serverless focused, less on EC2

## 1 IAM

- Always use roles, never use secret or access keys
- Manage users and their level of access to AWS
- capabilities
  - centralized control: universal, doesn't apply to region
  - shared access to your AWS account
  - granular permissions
  - Identity Federation
    - Active Directory: log in with Windows login password
    - Facebook / LinkedIn / ...
    - Using SAML (Security Assertion Markup Language 2.0), you can give your federated users single sign-on (SSO) access to the AWS Management Console.
  - Multifactor authentication
    - multiple independent authentication mechanisms
  - provide temporary access for users/devices and services
  - set up password rotation policy
  - integrates with many different AWS services
  - supports PCI DSS compliance
- Key terminology
  - *users*: end users
    - root account
      - account created when first setup AWS account
      - full admin access
      - always setup Multifactor authentication on root account
      - shouldn't be using day-to-day
    - new users have NO permissions when created (least privilege)
    - new users are assigned access key ID and secret access keys when created
  - programmatic access (CLI)
    - access key ID
    - secret access key (only visible once after creation)
    - never share access key among developers
    - don't upload to GitHub

- console access (browser)
  - password
- *groups*
  - a collection of users
  - under one set of permissions
  - Users inherit the permissions of their group
  - easier to collectively manage users who need the same permissions
- *policies*
  - a JSON document that defines permissions

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```

- give permissions as to what a user/role/group can do

- *roles*
  - a set of defined access
  - create roles and assign them to AWS resources
  - a secure way to grant permissions to entities
    - IAM user
    - application code
    - AWS service
    - users with identity federation
  - example: EC2 talk to S3
    - instead of assigning user access keys, give them roles
    - delete saved credentials `rm ~/.aws/credentials` and `rm ~/.aws/config`
    - create role -> EC2 -> AmazonS3FullAccess
    - go to EC2 -> actions -> instance settings -> attach/replace IAM role
  - preferred from security perspective
    - allow you to not use access key id and secret access key

- controlled by policies
  - if you change a policy it takes effect immediately
  - can attach/detach roles to running EC2 without having to stop or terminate the instances
- ARN
  - Amazon resource name
  - qualified: the function ARN with the version suffix
    - e.g. arn:aws:lambda:aws-region:acct-id:function:helloworld:\$LATEST
  - unqualified: the function ARN without the version suffix
    - e.g. arn:aws:lambda:aws-region:acct-id:function:helloworld
- KMS
  - key management service
  - multi-tenant solution
  - customer master key (CMK)
    - alias
    - creation date
    - description
    - key state
    - key material (customer/AWS provided)
    - can NEVER be exported
  - regional
  - create and control the encryption keys used to encrypt your data
  - integrated with
    - EBS
    - S3
    - Redshift
    - Elastic Transcoder
    - WorkMail
    - RDS
  - Envelope encryption
    - master key -> encrypts envelope key
    - envelope key (data key) -> encrypts the data
    - during decryption:
      - use master key to decrypt encrypted data key
      - use plain text data key to decrypt encrypted data
  - Lab
    - IAM -> create 2 users -> assign to a group with administrator access
    - Encryption keys ->

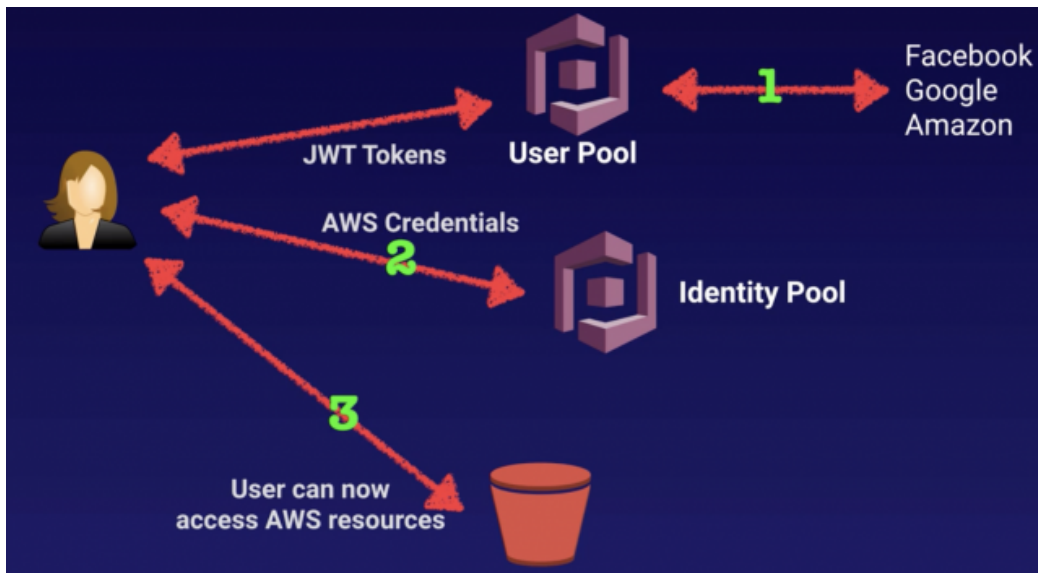
- choose region ->
- create alias and description ->
- choose material option
  - KMS generated key material
  - your own key material
- define key administrative permissions (manages but can't use) ->
- define key usage permissions (uses the key to encrypt and decrypt data)
- deletion
  - key actions -> disable -> schedule key deletion
  - 7 ~ 30 days
  - can't delete immediately, minimum wait a week
- API calls
  - create an EC2 instance

```
# to encrypt a file
aws kms encrypt --key-id YOURKEYIDHERE --plaintext fileb://secret.txt
--output text --query CiphertextBlob | base64 --decode >
encryptedsecret.txt
# to decrypt a encrypted file
aws kms decrypt --ciphertext-blob fileb://encryptedsecret.txt --
output text --query Plaintext | base64 --decode > decryptedsecret.txt
# to decrypt and encrypt a file with a new CMK without exposing the
plaintext
aws kms re-encrypt --destination-key-id YOURKEYIDHERE --ciphertext-
blob fileb://encryptedsecret.txt | base64 > newencryption.txt
# rotate the key
aws kms enable-key-rotation --key-id YOURKEYIDHERE
```

- Cognito - web identity federation
  - AWS web identity federation services (identity broker)
  - give user access with web-based identity provider like Amazon / Facebook / Google
  - User first authenticates with web ID provider -> authentication token
  - exchange authentication token for AWS credentials -> IAM role
    - sign up and sign in to your apps
    - access for guest users
    - synchronizes user data across devices
      - push synchronization



- User pools
  - user based
    - user registration
    - authentication
    - account recovery
    - sign up and sign in functionality (username, password)
- Identity pools
  - provide temporary credentials and authorization to access AWS services



- Cognito user pools
- STS
- Cross Account Access

## 2 EC2

### 2.1 EC2 101



- Elastic Compute Cloud
- resizable compute capacity (virtual machines) in the cloud
- Instance purchasing options
  - *on-demand*
    - fixed rate, low cost, flexibility
      - no up-front payment
      - no long-term commitment
    - applications with short term, spiky, unpredictable workloads
    - Linux: by the second
    - Windows by the hour
  - *reserved instances*
    - contract 1 year / 3 years
    - upfront charge with discounted hourly price
      - standard RI: up to 75% off on-demand
      - convertible RI: up to 54% off on-demand, have capacity to change your instance types
      - scheduled RI: launch within the time window you reserved
    - applications with steady state or predictable usage
    - can't change region
  - *spot*
    - bid price
    - if your application have flexible start and end times
    - applications only feasible at very low compute prices
      - genomics, pharmaceutical, chemical companies
    - if your spot instance is terminated by AWS, you won't be charged for a partial hour of usage
    - if you terminate the instance yourself you will be charged for the complete hour
    - pay for what you bid on
  - *dedicated hosts*
    - physical EC2 servers dedicated for your use
    - useful for regulatory requirements or licenses that may not support

multi-tenant virtualization or cloud deployments

- can be purchased on-demand or as a reservation
- Instance Types - F1 G1 I3 H1 M1 P1 R1 T1 X1

Family	Speciality	Use Case	Memorize
F1	field programmable gate array	Genomics research, financial analytics, realtime video processing, big data	
I3	high speed storage	NoSQL DBs, data warehousing	IOPS
G3	graphics intensive	video encoding/3D application streaming	graphics
H1	high disk throughput	MapReduce, distributed file systems (HDFS, MapR-FS)	high
T2	lowest cost, general purpose	web servers, small DBs	
D2	dense storage	file servers, data warehousing, Hadoop	density
R4	memory optimized	memory intensive apps/DBs	RAM
M5	general purpose	applications servers	main choice
C5	compute optimized	CPU intensive apps/DBs	compute
P3	graphics/general purpose GPU	ML, bitcoin mining	pics
X1	memory optimized	SAP HANA/Apache Spark	extreme memory

- How I remember them now;
  - F for FPGA
  - I for IOPS
  - G - Graphics
  - H - High Disk Throughput
  - T cheap general purpose (think T2 Micro)
  - D for Density
  - R for RAM
  - M - main choice for general purpose apps
  - C for Compute
  - P - Graphics (think Pics)
  - X - Extreme Memory

- created from Amazon Machine Image (AMI)
  - marketplace & community has pre-built AMI for purchase
- key pair
  - to login to your EC2
  - `chmod 400 key.pem`
  - save in ~/.ssh
  - `ssh ec2-user@public_ip -i key.pem`
- to using the instance as a web server
  - `sudo su` to elevate access to root
  - `yum update -y` update the OS and relevant packages
  - `yum install httpd -y` to install Apache
  - `service httpd start` to start the server
  - `chkconfig httpd on` to turn Apache on automatically if instance restarts
  - `service httpd status` to check if Apache is running
  - `cd /var/www/html` go to root directory for web server
  - `vim index.html` to add content in webpage
  - `<html><body><h1>Hello</h1></body></html>`
  - go to web browser with the public ip, you should see the content you put
- 1 subnet == 1 availability zone
- Bootstrap Script
  - Advanced details
  - runs at root level, executes commands in chronological order
  - automating software installs and updates

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

```

yum install httpd php 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.amazonaws.com/acloudguru-production/connect.php

```

### **connect.php**

```

<?php
$username = "acloudguru";
$password = "acloudguru";
$hostname = "yourhostnameaddress";
$dbname = "acloudguru";

//connection to the database
$dbhandle = mysql_connect($hostname, $username, $password) or
die("Unable to connect to MySQL");
echo "Connected to MySQL using username - $username, password -
$password, host - $hostname<br>";
$selectd = mysql_select_db("$dbname",$dbhandle) or die("Unable
to connect to MySQL DB - check the database name and try again.");
?>

```

- to get information about an instance
  - `curl https://169.254.169.254/latest/meta-data/`
  - `curl https://169.254.169.254/latest/user-data/` bootstrap scripts
- Termination protection
  - prevent your instance from being accidentally terminated from the console, CLI, or API.
  - disabled by default
  - The `DisableApiTermination` attribute does not prevent you from terminating an instance by initiating shutdown from the instance (using an operating system command for system shutdown) when the `InstanceInitiatedShutdownBehavior` attribute is set.

## **2.2 Security Group**

- Updated rule take effect immediately
- Inbound
  - everything blocked by default

- white list
  - allow rules only, no deny rules
  - use Network Access Control Lists to block specific IP
- Outbound
  - all outbound traffic is allowed
- Stateful
  - If you create an inbound rule allowing traffic in, that traffic is automatically allowed back out again
- EC2: Security Groups = m:m

## 2.3 EBS Volume

- block storage
  - With block storage, files are split into evenly sized blocks of data, each with its own address but with no additional information (metadata) to provide more context for what that block of data is.
  - Another key difference is that block storage can be directly accessed by the operating system as a mounted drive volume, while object storage cannot do so without significant degradation to performance.
- EC3 block storage
  - Elastic Block Storage (EBS) volume
  - instance store volume
- Elastic Block Storage
  - root device volume
    - or boot volume
    - where Linux/Windows is installed (like C://)
  - additional drive
    - attached to an EC2
    - create a file system on it
    - run a database on it
  - need to be in same availability zone as the EC2 instance
  - automatically replicated to avoid failure
- Types

EBS Types	AWS Name	Disk	Feature	Use Case
general purpose SSD	GP2	SSD	balances price & performance (3 ~ 10,000 IOPS per GB)	most workloads

provisioned IOPS SSD	IO1	SSD	high performance (> 10,000 IOPS)	I/O intensive application; DB (relational or NoSQL)
throughput optimized HDD	ST1	HDD	frequent access, throughput intensive; can't be a boot volume	Big data, data warehouse, log processing
cold HDD	SC1	HDD	less frequently accessed workloads, lowest cost; can't be a boot volume	file server
EBS magnetic (standard)		HDD	legacy generation; only magnetic volume that is bootable	cheap, infrequent data access

---

\*HDD: magnetic storage volumes

- EBS lab
  - add an additional volume to an EC2
  - `sudo su` then `lsblk` to show all volumes
    - `xvda` has `MOUNTPOINT: /` means that's your root volume
    - `xvdf` is the EBS you attached, with no `MOUNTPOINT`
  - `file -s /dev/xvdf` if returns `/dev/xvdf: data` then it means there is no data on the volume
  - create a file system on the additional volume: `mkfs -t ext4 /dev/xvdf`
  - `file -s /dev/xvdf` should return `Linux rev 1.0 filesystem data` now
  - mount the volume to a directory: `cd` then `mkdir filesystem` then `mount /dev/xvdf /filesystem`
  - `lsblk` should now show `MOUNTPOINT: /filesystem` for the additional volume
  - use the additional volume: `cd filesystem` then `echo "hello" > hello.txt`
  - unmount the volume: `cd` then `umount -d /dev/xvdf`
  - detach the volume in AWS console

- `lsblk` should only show `xvda` now
- `cd filesystem` then `ls` should return nothing, `hello.txt` was deleted
- EBS volume -> actions -> create snapshot (encrypted)
- snapshot -> create volume
- EBS volume -> actions -> attach volume to EC2
- `lsblk` should show `xvdf` now, `file -s /dev/xvdf` should say there is a filesystem on it
- `mount /dev/xvdf /filesystem` then `cd /filesystem` then `ls` you should see `hello.txt`
- volume size can be changed on the fly
  - size
  - storage type
- When instance terminated
  - root EBS volume
    - deleted by default
    - can turn termination protection on
  - additional ECS volumes persists
- Encryption
  - can't encrypt EBS root volume of default AMI
  - to encrypt root volume
    - use OS (bitlocker if Windows)
    - take a snapshot -> create a copy and select encrypt -> create image from the encrypted copy -> launch a new instance from the AMI (only larger instances available)
  - can encrypt additional volume
- Snapshot
  - exist on S3
  - time copies of the volumes
  - Incremental
  - can be taken when instance is running, but stop the instance first if the EBS serves as root device to ensure consistency
  - Encryption
    - snapshots of encrypted volumes are encrypted automatically
    - volumes created from encrypted snapshots are encrypted automatically; volumes created from unencrypted snapshots are unencrypted automatically
    - you can share snapshots, but only if they are unencrypted
      - with other AWS accounts



- public, sell in marketplace
  - can be used to move EC2 volume to another availability zone
- to move EC2 to another AZ
  - take a snapshot
  - create an AMI from the snapshot
  - use the AMI to launch a EC2 in a new AZ
- to move EC2 to another region
  - take a snapshot
  - create an AMI from the snapshot
  - copy AMI to another region
  - use the AMI to launch a EC2 in the new region
- Can create AMI (Amazon Machine Image) from EBS-backed instances and snapshots
- EBS vs Instance Store
  - instance store (ephemeral storage)
    - The data in an instance store persists only during the lifetime of its associated instance
    - can't be stopped, can reboot or terminate
      - will lose data if stopped, or underlying disk drive fails or instance terminates
      - won't lose data if reboot (intentionally or unintentionally)
      - ROOT volumes will be deleted on termination
    - You can attach additional instance store volumes to your instance. You can also attach additional EBS volumes after launching an instance, but not instance store volumes.
    - the root device for an instance launched from the AMI is a instance store created from a template stored in S3 (can take a little longer than EBS backed volumes)
  - EBS backed volumes
    - the root device for an instance launched from the AMI is a EBS volume created from a EBS snapshot
    - can be stopped and snapshotted
    - won't lose data if stopped
    - won't lose data if reboot
    - by default, ROOT volumes will be deleted on termination, but can change setting to keep it

## 2.4 Command Line CLI

- prerequisite
  - setup access in IAM:
    - add user -> programmatic access (access key ID and secret access key)
- configure credentials
  - `aws configure` to put in access key ID and secret access key
- example commands
  - `aws s3 ls` list all buckets
  - `aws s3 mb s3://test-bucket` create a new bucket
  - `echo "test" > hello.txt` create a new file
  - `aws s3 cp hello.txt s3://test-bucket` copy file to bucket
  - `aws s3 ls s3://test-bucket` list files in bucket
- install on your laptop

## 2.5 CloudWatch

- monitoring performance
- CloudTrail is about auditing API calls
- AWS services and user applications
- Interval
  - every 5 minutes by default
  - can have 1 minute interval if turn on detailed monitoring
- Features
- Alarms
- Dashboards
- Events
- Logs
- Cloud Watch vs Access Logs
  - access logs: requests or connections, the time it was received, the client's IP address, latencies, request paths, and server responses

## 2.6 EFS - Elastic File System

- supports the Network File System version 4 (NFSv4) protocol
- only pay for the storage you use, no pre-provisioning
- can scale to TB
- can support thousands of concurrent NFS connections
- can't share EBS with multiple EC2
- can create EFS mount
- stored multi AZ within a region

- read after write consistency

## 2.7 Spread placement groups

- EC2 attempts to place instances so that they spread out across underlying hardware to minimize correlated failures by default
- Spread placement groups configures the placement of a group of instances
- 2 types
  - Cluster
    - packs instances close together inside 1 AZ
    - low-latency network performance
    - high throughput
    - tightly-coupled node-to-node communication that is typical of HPC applications
  - spread placement group
    - critical instances
    - different hardware
- unique namespace within AWS account
- can't merge placement groups
- can't move existing instance to a placement group
- maximum of 7 running instances per Availability Zone

## 2.8 Load Balancer

- Balance load across web servers
- Types of Load Balancers
  - *Application load balancer*
    - layer 7 (application layer): application aware
    - intelligent, advanced request routing
    - best suited for balancing HTTP and HTTPS traffic
  - *Network load balancer*
    - layer 4 - connection level
    - TCP traffic
    - extreme performance: millions of requests per second
    - most expensive, usually used in production
  - *classic load balancer / elastic load balancer*
    - balancing HTTP/HTTPS applications use Layer 7-specific features, e.g. x-forwarded and sticky sessions
      - Public IP -> load balancer -> private IP to server, the x-forwarded-for

header contains the public IP (IPv4 address)

- also Layer 4 load balancing for applications that rely purely on the TCP protocol
- low cost
- legacy
- 504 Error
  - gateway timed out
  - application not responding
  - -> trouble shoot the application
    - web server / db
    - scale it up or out if necessary

## **3 Databases**

### **3.1 Relational Database**

- since the 70s
- database
- tables
- row
- fields (columns)

### **3.2 RDS (OLTP)**

- relational database service
- runs on VM
  - but you can't log into these VMs
  - no OS level access
  - AWS patches the OS and DB
  - NOT serverless
  - Aurora Serverless IS serverless
- 6 Engines
  - MySQL
  - PostgreSQL
  - Oracle
  - SQL Server
  - MariaDB (fully compatible with MySQL)
  - Aurora (not available for free tier yet)
- Instance Types
  - optimized for memory
  - optimized for performance
  - optimized for I/O
- Storage
  - general purpose SSD
  - provisioned IOPS SSD: for I/O-intensive workloads, particularly database workloads
  - magnetic: backward compatibility
  - You can create MySQL, MariaDB, and PostgreSQL RDS DB instances with up to 32 TiB of storage. You can create Oracle RDS DB instances with up to 64

TiB of storage. You can create SQL Server RDS DB instances with up to 16 TiB of storage. For this amount of storage, use the Provisioned IOPS SSD and General Purpose SSD storage types.

- 2 types of Backups on AWS

- *automated backups*

**Backup**

**Backup retention period**  
The number of days for which automated backups are retained. Setting this parameter to a positive number enables backups. Setting this parameter to 0 disables automated backups.

7 days ▼

**Backup window**  
The daily time range (in UTC) during which automated backups are created if automated backups are enabled.

**Start Time**      **Duration**

02 ▼ : 16 ▼ UTC      0.5 ▼ hours

- Recover to any point in time within a retention period
    - 1 ~ 35 days
  - daily snapshot + transaction logs throughout the day
    - choose the most recent daily snapshot, then apply the logs
    - allows point in time recovery down to a second within the retention period
  - enabled by default
  - backup data stored in S3
  - free storage space equal to the size of your DB
  - taken during a defined window, you may experience latency
    - change to backup window takes effect immediately
  - deleted after deletion of original RDS instance
- *snapshots*
  - done manually, user initiated
  - stored even after deletion of original RDS instance
- when restore a backup/snapshot, the restored version of the DB will be a new RDS instance with a new DNS endpoint
- Encryption
  - supporting all 6 engines
  - done using the KMS service (Key Management Service)
  - If the RDS instance is encrypted, the data stored, the automated backups, read replicas and snapshots are also going to be encrypted
- Multi-AZ
  - When you provision a Multi-AZ DB Instance, Amazon RDS automatically creates a primary DB Instance and synchronously replicates the data to a

- standby instance in a different Availability Zone (AZ)
- for disaster recovery only
  - synchronous replications
  - exact copy of main instance in another AZ
  - not for improving performance (use read replicas for performance improvement)
  - can't use secondary DB instances for writing purposes
- auto failover
  - DB maintenance/DB instance failure/availability zone failure
  - AWS will point the endpoint to the secondary instance
  - Reboot from failover: force change AZ
  - Loss of availability in primary Availability Zone
  - Loss of network connectivity to primary Compute unit
  - failure on primary Storage
  - failure on primary
- same DNS routed to secondary DB
  - CNAME changed to standby DB
- supporting all 5 other engines besides Aurora
  - Aurora is fault tolerant itself so no need for multi-AZ
- Read replica
  - performance, scaling
  - spread load across multiple read replicas
  - for read-heavy DB workload
  - must have automatic backups turned on
  - read-only copy of your production DB
  - asynchronous replications
    - up to 5 of any database
    - can have read replicas of read replicas (latency)
    - each read replica has its own DNS end point
    - can have multi-AZ
    - can be in another region
    - can be created for multi-AZ DB
    - can be Aurora or MySQL
  - can be promoted to master as their own database, breaks the replication though
  - read from different DB but only write to 1 primary
    - read directed first to primary DB and then sent to secondary DB
  - no auto failover

- supporting
  - MySQL
  - PostgreSQL
  - MariaDB
  - Aurora
- no charge for replicating data from your primary RDS instance to your secondary RDS instance
- Security Group
  - Technically a destination port number is needed, however with a DB security group the RDS instance port number is automatically applied to the RDS DB Security Group.
- RDS Reserved instances
  - reserve a DB instance for a one or three year term
  - discount compared to the On-Demand Instance pricing
  - available for multi-AZ deployments
- database on EC2
  - EBS
- endpoint
  - DNS address
  - never given a public IPv4 address
  - AWS manages the mapping from DNS to IPv4
- security group
  - edit inbound rule to allow MYSQL/Aurora traffic from web server
  - add in the web server security group instead of an IP address

### **3.3 Non-relational databases**

- collection
- document
- key-value pairs
- JSON

### **3.4 DynamoDB**

- NoSQL database
- consistent, single-digit millisecond latency at any scale
- fully managed
- serverless
- auto-scale



- e.g. Use DynamoDB for session storage alleviates issues that occur with session handling in a distributed web application by moving sessions off of the local file system and into a shared location
- integrates well with Lambda
- 2 data models
  - document
    - JSON, HTML, XML
  - Key-value
    - key: name of the data
    - value: the data itself
- Components
  - *Tables*
  - *Items*: row
  - *Attributes*: column
- use case
  - mobile
  - web
  - gaming
  - ad-tech
  - IoT
- Stored on SSD storage -> fast
- Spread across 3 geographically distinct data centers
  - avoid single point of failure
- partitioned into nodes -> sharding
- 2 consistency models
  - eventual consistent reads
    - default
    - consistency across all copies usually reach within 1 sec
    - best read performance
  - strongly consistent reads
    - returns a result that reflects all writes that received a successful response prior to the read
    - needs to read an update within 1 sec
- Primary keys
  - stores & retrieves data based on a primary key
  - 2 types
    - partition key
      - unique attribute, e.g. user ID

- put to a internal hash function -> determines the partition or physical location on which the data is stored
- composite key
  - == partition key + sort key
  - use if partition key is not necessarily unique
  - e.g. user ID + timestamp
  - all items with the same partition key are stored together, then sorted according to the sort key value
- Access Control
  - managed with IAM
  - you can create an IAM user to access & create DynamoDB tables
  - or an IAM role that enables you to obtain temporary access keys to DynamoDB
  - or use a special *IAM condition* parameter `dynamodb:LeadingKeys` to restrict user access to only their own records
    - add a condition to an IAM policy to allow access only to items where the partition key value matches their user ID

```

"Condition": {
  "ForAllValues:StringEquals": {
    "dynamodb:LeadingKeys": [
      "${www.mygame.com:user_id}"
    ],
    "dynamodb:Attributes": [
      "UserID",
      "GameScore"
    ]
  }
}

```

- Billing
  - within a single region
    - read and write capacity
    - incoming data transfer
  - cross regions
    - data transfer
- Index
  - a data structure that allows you to perform fast queries on specific columns in SQL databases
  - run your searches on the index, rather than on the entire dataset

	local secondary index	global secondary index
creation	can only be created when table is created, can't add/remove/modify later	you can create it when create your table, or add it later
partition key	same as your original table	different
sort key	different	different
view	a different view of your data, organized according to an alternative sort key	a completely different view of the data
Example	partition: user id; sort: account creation date	partition: email address; sort: last log-in date

- Scan vs Query API call
  - query
    - a query operation finds items in a table based on the primary key attribute (partition key) and a distinct value to search for
  - scan
    - examines every item in the table, then applies the filter

	Query	Scan
finds items with	primary key and a distinct value to search for	examines every item in a table
refine results	use an optional sort key to refine the results	then applies a filter
attributes returned	by default, returns all attributes for the items	by default, returns all attributes for the items
only return specific attributes	use the <code>ProjectionExpression</code> parameter	use the <code>ProjectionExpression</code> parameter (use filter in console)
sorted	by sort key (ascending, set <code>ScanIndexForward</code> to False for descending)	
consistency	by default, eventually consistent, can explicit set it to be strongly consistent	

performance

generally more efficient

scan dumps the entire table, then filters out the values to provide the desired result, which adds an extra step and as the table grows, the scan operation takes longer. Scan operation on a large table can use up the provisioned throughput for just in a single operation.

- 
- to improve performance
    - set a smaller page size which uses fewer read operations
      - larger number of small operations
      - allows other requests to succeed without throttling
    - avoid scan if you can
    - isolate scan operations to specific tables and segregate them from your mission-critical traffic
    - design your table in a way that you can use `Query`, `Get`, or `BatchGetItem` API
    - use parallel scans
      - by default, a scan operation processes data sequentially in returning 1MB increments before moving on to retrieve the next 1MB of data
      - it can only scan one partition at a time
      - you can configure DynamoDB to use parallel scans instead of logically dividing a table or index into segments and scanning each segment in parallel
      - best to avoid parallel scans if your table or index is already incurring heavy read/write activity from other applications
  - Pricing models
    - provisioned throughput
    - on-demand capacity option
  - Provisioned Throughput
    - measured in Capacity Units
    - 1 Write Capacity Unit = 1 \* 1KB Write per second
    - 1 Read Capacity Unit = 1 \* Strongly Consistent Read of 4KB per second OR 2 \* Eventually Consistent Reads of 4KB per second (default)
    - Calculate how many capacity units each read/write need separately and then round up to the nearest whole number. Finally multiply by the number of read/write per second.

- e.g. If each row is 6KB, then it needs 2 Read Capacity Units. If you read 100 rows per second, then it requires 200 Read Capacity Units.
- If you only need eventual consistency, divide the result by 2 (100 Read Capacity Units)
- need to specify Read Capacity Units and Write Capacity Units requirements when create your table
- `ProvisionedThroughputExceededException`
  - your request rate is too high for the read/write capacity provisioned on your DynamoDB table
  - SDK will automatically retries the requests until successful
  - if you are not using the SDK, you can
    - reduce the request frequency
    - use Exponential Backoff
  - raise a ticket to AWS support to increase the provisioned throughput
- Exponential Backoff
  - many components in a network can generate errors due to being overloaded
  - progressively longer waits between consecutive retries for improved flow control
  - e.g. 50 ms, 100 ms, 200 ms...
  - If after 1 minute this doesn't work, your request size may be exceeding the throughput for your read/write capacity
  - all AWS SDKs use exponential backoff besides simply automatic retry
    - S3, CloudFormation, SES
- On-Demand Capacity Pricing Option
  - charges apply for: reading, writing and storing data
  - you don't need to specify your capacity requirements
  - DynamoDB instantly scales up and down based on the activity of your application
  - can change provisioned capacity to on-demand once a day

On-Demand Capacity	Provisioned Capacity
Unknown workloads	You can forecast read and write capacity requirements
Unpredictable application traffic	Predictable application traffic
Spiky, short lived peaks	Application traffic is consistent or increases gradually

you want to pay for only what you use (pay per request)

---

- Accelerator (DAX)
  - a in-memory cache for DynamoDB
  - delivers up to a 10x read performance improvement
  - eventually consistent reads only, not for applications that require strongly consistent reads
  - microsecond performance for millions of requests per second
  - ideal for read-heavy and bursty workloads
  - e.g. auction, gaming and retail sites during black Friday promotions
  - a write-through caching service: data is written to the cache as well as the back end store at the same time
  - allows you to point your DynamoDB API calls at the DAX cluster instead of the table
    - If the item you are querying is in the cache (cache hit), DAX will return the result.
    - If the item is not available (cache miss), then DAX performs an eventually consistent `GetItem` operation against DynamoDB. Also write the data to cache.
- Advantages
  - Retrieval of data from DAX reduced read load on DynamoDB
  - may be able to reduce Provisioned Read Capacity
- Not for
  - not for write-intensive applications and applications that don't perform many read operations
  - not for applications that don't require microsecond response times
- Transactions
  - supports mission-critical applications which need an all-or-nothing approach
  - ACID transactions
    - Atomic: transactions are treated as a single unit, all or nothing operation, can't be partially completed
    - Consistent: transactions must leave the DB with a valid status, prevents corruption and integrity issues
    - Isolated: no dependency between transactions, can happen in parallel or sequentially
    - Durable: when committed, remain committed, even after system or power failure (written to disk not in memory)

- read or write multiple items across multiple table as an all or nothing operation
- check for a pre-requisite condition before writing to a table
- TTL
  - defines an expiry time for your data
  - expressed as epoch time (UNIX time)
    - number of seconds since 1970-01-01 2 am
  - expired items are marked for deletion
    - within the next 48 hours
    - you can filter out expired items from your queries and scans
  - used for removing irrelevant/old data
    - session data
    - event logs
    - temporary data
  - reduces cost by automatically removing data which is no longer relevant
  - Lab: create a DynamoDB table and configure TTL

```
# check IAM user has permission for DynamoDBFullAccess in console
aws iam get-user

# create a new table SessionData
aws dynamodb create-table --table-name SessionData --attribute-
definitions \
AttributeName=UserID,AttributeType=N --key-schema \
AttributeName=UserID,KeyType=HASH \
--provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5

# download data JSON file
# populate SessionData table
aws dynamodb batch-write-item --request-items file://item.json

{
  "SessionData": [
    {
      "PutRequest": {
        "Item": {
          "UserID": {"N": "5346747"},
          "CreationTime": {"N": "1544016418"},
          "ExpirationTime": {"N": "1544140800"},
          "SessionId": {"N": "6734678235789"}
        }
      }
    }
  ]
}
```

```

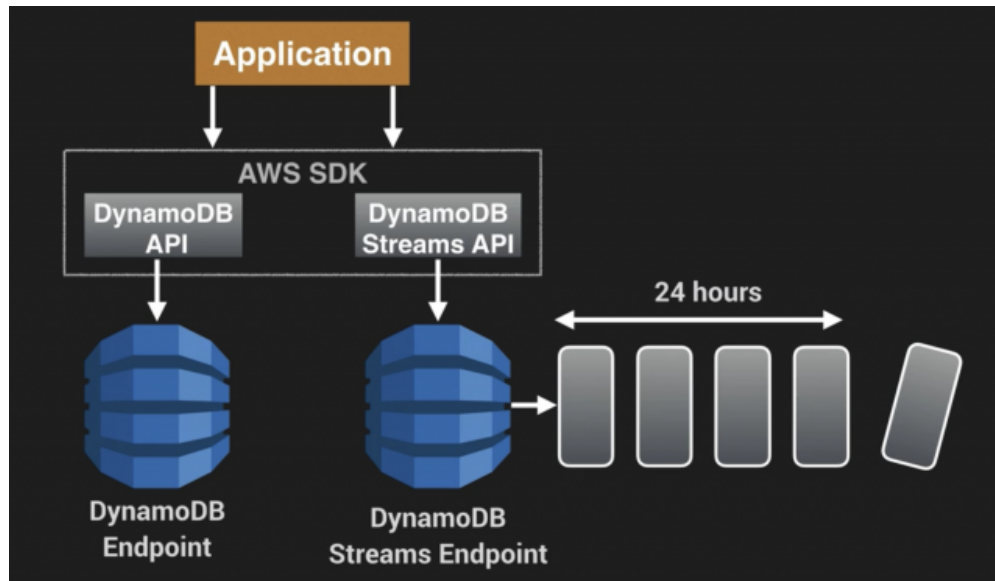
    }
  },
  {
    "PutRequest": {
      "Item": {
        "UserID": {"N": "6478533"},
        "CreationTime": {"N": "1544013196"},
        "ExpirationTime": {"N": "1544140800"},
        "SessionId": {"N": "6732672579220"}
      }
    }
  },
  {
    "PutRequest": {
      "Item": {
        "UserID": {"N": "7579645"},
        "CreationTime": {"N": "1544030827"},
        "ExpirationTime": {"N": "1544140800"},
        "SessionId": {"N": "7657687845893"}
      }
    }
  }
]
}

```

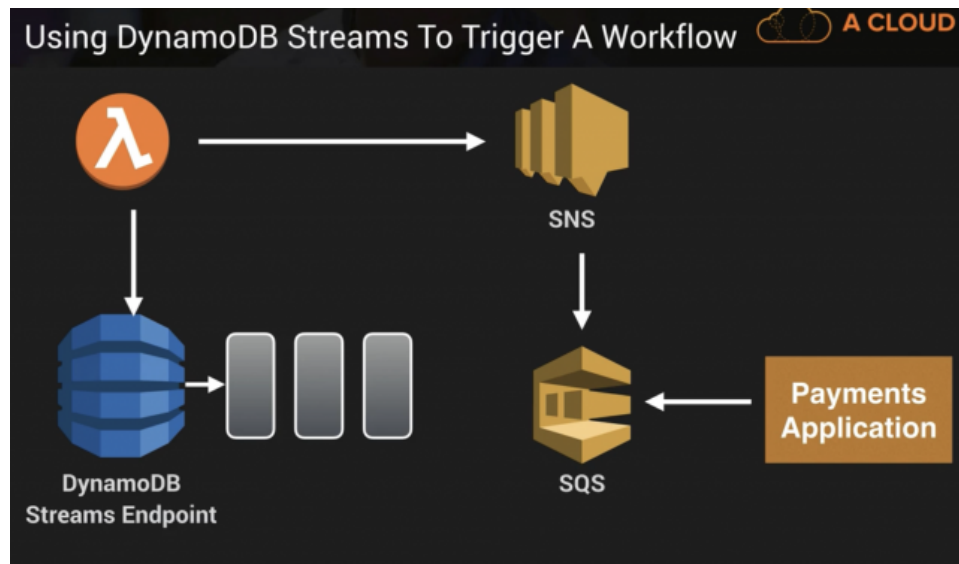
- then you should be able to check for the table and items in the console
- **Actions** -> **Manage TTL** -> TTL attribute put name of the column (ExpirationTime) -> **Continue**
- **Run preview** will show you which records will expire with your TTL



- Streams



- time-ordered sequence (stream) of item level modifications
  - insert, update, delete
  - recorded in near real-time
- logs are encrypted at rest and stored for 24 hours
  - auditing
  - archiving of transactions
  - replaying transactions to a different table
  - trigger events based on a particular change ->
  - event source for Lambda: create applications that takes actions based on events in your DynamoDB table
    - Lambda polls the DynamoDB Streams, executes Lambda code based on a DynamoDB Streams event
  - applications can take actions based on contents
- accessed using a dedicated endpoint
- by default the primary key is recorded
- before and after images can be captured



- Create DynamoDB tables with PHP files on an EC2
  - Create an IAM role
    - IAM -> Roles -> Create Role -> AWS service -> EC2 -> AmazonDynamoDBFullAccess
  - EC2
    - launch instance -> select IAM role -> Advanced Details

```

#!/bin/bash
yum update -y
yum install httpd24 php56 git -y
service httpd start
chkconfig httpd on
cd /var/www/html
echo "<?php phpinfo();?>" > test.php
git clone https://github.com/acloudguru/dynamodb

```

- security group allows: HTTP via port 80 and SSH access
- Check bootstrap script worked

```

ssh -i mykeypair.pem ec2-user@publid-ip
sudo su
cd /var/www/html
ls # should have a dynamodb repository and `test.php` file

```

- Install the AWS SDK for PHP via Composer

```

sudo su

```

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"

php -r "if (hash_file('sha384', 'composer-setup.php') ===
'a5c698ffe4b8e849a443b120cd5ba38043260d5c4023dbf93e1558871f1f07
f58274fc6f4c93bcfd858c6bd0775cd8d1') { echo 'Installer
verified'; } else { echo 'Installer corrupt'; unlink('composer-
setup.php'); } echo PHP_EOL;"

php composer-setup.php

php -r "unlink('composer-setup.php');"

php -d memory_limit=-1 composer.phar require aws/aws-sdk-php
```

- cd into the dynamodb folder, update createtables.php and uploaddata.php with your EC2 region
- Create tables and upload data in DynamoDB
  - in browser, go to public-ip/dynamodb/createtables.php and it will run the script to create some tables in DynamoDB
  - go to public-ip/dubamodb/uploaddata.php and enter to upload data to the tables created
  - go to DynamoDB to check the data we uploaded
- DynamoDB portal: Scan (shows everything in the table) / Query
- query DynamoDB programmatically in CLI

```
aws dynamodb get-item --table-name ProductCatalog --region eu-west-2
--key '{"Id":{"N":"205"}}' # primary key
# because your EC2 has the DynamoDB role
```

### 3.5 Data Warehousing - Redshift (OLAP)

- business intelligence, reporting
- tools: Cognos, Jaspersoft, SQL Server Reporting Services, Oracle Hyperion, SAP NetWeaver
- PB scale, large and complex datasets
- use a separate data store to not affect production DB performance
- different architecture of DB and infrastructure layer
  - single Node (160 GB)

- multi-node
  - leader node
    - manages clients connections and receives queries
  - compute node
    - stores data and perform queries
    - up to 128
- Advanced Compression
  - compression based on columns instead of rows
    - data of same type
    - stored sequentially on disk
  - doesn't require indexes or materialized views
    - use less space
  - auto samples data and selects most appropriate compression scheme when loading data
  - massively parallel processing (MPP)
- Backups
  - enabled by default with 1 day retention period
    - Max retention period 35 days
  - always attempts to maintain at least 3 copies of your data
    - original on compute node
    - replica on compute node
    - backup in S3
  - asynchronously replicate your snapshots to S3 in another region for disaster recovery
- Billing
  - billed for compute node hours
    - 3 node cluster run for 1 day will incur 72 billable hours
    - only compute node, not leader node
  - billed for backup
  - billed for data transfer (only within a VPC, not outside it)
- Security
  - encrypted in transit using SSL
  - encrypted at rest using AES-256 (advanced standard)
  - by default redshift takes care of key management
    - can manage your own key through
      - HSM
      - KMS key management service
- Availability

- currently only available in 1 AZ
- can restore snapshots to new AZ in the event of an outage

### **3.6 ElastiCache**

- in-memory caching (frequently-accessed data)
- improves web applications performance
  - caching frequent queries
  - pull from in-memory cache instead of disk-based DB
  - takes off load from production DB
  - works with DynamoDB as well, but DAX is specifically designed for DynamoDB and it only supports the write through strategy
- sits between your application and the database, takes the load off your DB
- for read-heavy DB workload & data is not changing frequently
  - social networking
  - gaming
  - media sharing
  - Q&A portals
- or compute-intensive workload
  - recommendation engine (store results of I/O intensive DB queries or output of compute-intensive calculations)
- 2 engines
  - Memcached
    - object caching
    - simple cache to offload DB
    - scale horizontally (scale out)
    - multi-threaded performance
    - no multi-AZ capacity
    - no persistence
    - managed as a pool that can grow and shrink, similar to EC2 auto scaling group
    - individual nodes are expendable
    - automatic node replacement
    - auto discovery
  - Redis
    - open-source in-memory key-value store
    - supports complex data structures like sets and lists
    - support master/slave replication
    - support cross AZ redundancy with fail-over

- ranking & sorting: leaderboards
- pub/sub capabilities
- persistence
- Backup & restore capabilities
- managed more as a relational database
- Redis clusters are managed as stateful entities that include failover, similar to RDS
- Caching Strategies
  - Lazy Loading
    - loads the data into the cache only when necessary
    - if requested data is in the cache, Elasticache returns the data
    - If it's not in the cache or has expired, Elasticache returns a `null`
    - Your application then fetches the data from the DB and writes the data received in to the cache so that it is available next time
    - Add TTL (Time to live): number of seconds until the key (data) expires to avoid keeping stale data in the cache
    - Lazy loading treats an expired key as a cache miss and causes the application to retrieve the data from the database and subsequently write the data in to the cache with a new TTL
    - doesn't eliminate stale data, but helps to avoid it
  - Write-Through
    - adds or updates data to the cache whenever data is written to the DB

	Advantages	Disadvantages
Lazy Loading	only requested data is cached, avoids filling up cache with useless data; Node failures are not fatal, a new empty node will just have a lot of cache misses initially	cache miss penalty: initial request, then have to query the DB and write data to cache; stale data - if data is only updated when there is a cache miss, it can become out-of-date. Doesn't automatically update if the data in the DB changes.
Write-Through	data in cache is never stale; Users are generally more tolerant of additional latency when updating data than when retrieving it	write penalty: every write involves a write to the cache as well as to the DB; If a node fails and a new one is spun up, data is missing until added or updated in the DB; wasted

resources if most of the data is  
never read

---

- if a DB is having a lot of load, you should use ElastiCache if DB is read-heavy and not prone to frequent changing; if it's because management keep running OLAP transactions, use Redshift

### **3.7 Aurora**

- MySQL & PostgreSQL compatible
- open source
- cost effective
- Scalability
  - auto scaling storage
  - start with 10 GB, scales in 10GB increments to 64 TB
  - compute resources: scale up to 32vCPUs and 244GB of memory
- Availability
  - 2 copies of your data is contained in each AZ
  - minimum of 3 AZ
  - -> 6 copies of your data
  - designed to handle the loss of
    - up to 2 copies of data without affecting DB write availability
    - up to 3 copies without affecting read availability
  - Self-healing
  - automated backups always enabled
    - doesn't impact performance
  - snapshots
    - doesn't impact performance
    - can be shared with other AWS accounts
- Performance
  - read replicas
    - Aurora replicas: up to 15
    - MySQL replicas: up to 5
  - Automated failover from Aurora to Aurora read replica
  - No Automated failover from Aurora to MySQL read replica
- Endpoints
  - cluster: primary instance for write operations
  - reader: read replicas
  - instance: individual instance

## 4 S3

### 4.1 S3 101

- Simple Storage Service
- object-based storage: files
  - files, images, web pages,
  - not for OS or DB
- capacity
  - total volume of data you can store is unlimited
  - don't need to predict how much storage AWS does capacity planning on your behalf
  - for a single object: 0 byte to 5 TB
  - to upload with a `PUT` operation, limit is 5 GB
- object
  - key: file name
  - value: data (sequence of bytes)
    - stored in buckets (folder)
    - all newly created buckets are private by default
    - receive a HTTP 200 status code if uploaded successful, when you use CLI or API
    - can setup logs
- version ID
  - for versioning
- metadata
  - data about data you are storing
- subresources
  - bucket-specific configuration
  - bucket policies, Access Control List
  - cross origin resource sharing (CORS)
    - allows resources in one bucket to access files in another bucket (by default they can't)
    - e.g. S3 hosted website accessing javascript or image files located in another S3 bucket
    - need to set this up even if the files are publicly accessible
    - bucket -> properties -> static website hosting -> use this bucket to



- host a website
  - bucket -> permissions -> CORS configuration
  - put in the endpoint website URL not the bucket URL
  - transfer acceleration
    - data transfer between user and S3 bucket (upload)
- universal name space
  - bucket name needs to be unique globally
  - similar to a DNS address
  - `https://s3-eu-west-1.amazonaws.com/bucket_name`
- Charges
  - storage per GB
  - requests (GET, PUT, COPY)
  - storage management
    - inventory
    - analytics
    - object tags
  - data management pricing
    - data transferred out of S3
    - free to transfer in S3
  - transfer acceleration
- durability
  - safe
  - data is spread across multiple devices and facilities
  - designed to sustain the loss of 2 facilities concurrently
  - guarantees 11 x 9s durability (amount of data you can expect to lose a year)
  - if you store 10 million object, you will lose 1 object every 10 thousand years
- availability
  - built for 99.99%
  - guarantee 99.9%
- data consistency
  - read after write consistency: PUTS of new objects
    - after you upload a file, you'll be able to access it
  - eventual consistency: overwrite PUTS and DELETES
    - update or delete of a file can take time to propagate
- The Storage Gateway service is primarily used for attaching infrastructure located in a Data center to the AWS Storage infrastructure. The AWS documentation states that; "You can think of a file gateway as a file system mount on S3." Amazon Elastic File System (EFS) is a mountable file storage

service for EC2, but has no connection to S3 which is an object storage service. Amazon Elastic Block Store (EBS) is a block level storage service for use with Amazon EC2 and again has no connection to S3.

#### 4.2 Tiered Storage

S3 Type	Availability	Durability	Features	Retrieval Time
S3 standard	99.99%	11 * 9	stored redundantly across multiple device in multiple facilities; sustain loss of 2 facilities concurrently	ms
S3 - IA	99.9%	11 * 9	rapid but infrequent access; charged a retrieval fee	ms
S3 One Zone - IA	99.5%	11 * 9	20% lower cost compared to S3 IA , infrequently access; no multi-AZ	ms
reduced redundancy (legacy)	99.99%	99.99%	data that can be recreated if lost, e.g. thumbnails	
S3 - Intelligent Tiering	99.9%	11 x 9	unknown or unpredictable access patterns; both frequent & infrequent access tiers, automatically move your data to the most cost-effective tier based on your access frequency	ms
S3 Glacier		11 * 9	data archive, infrequently accessed; very	3 ~ 5 hours

	cheap	
S3 Glacier Deep Archive	lowest cost	12 hours

---

- Lifecycle management
  - automates moving your objects between the storage tiers
  - can be used in conjunction with versioning
  - can be applied to current or previous versions

### 4.3 Security

- private/public buckets
  - by default, all newly created buckets are PRIVATE
  - only the owner can upload/read/delete
  - no public access
- access control
  - bucket policies
    - applied at a bucket level
    - JSON
    - can't make an object public if it's in a private bucket
    - Permissions -> bucket policy -> policy generator
    - S3 bucket policy -> allow ->
  - access control list
    - applied at an object level
    - fine-grained access control
    - add account/make public
    - different permissions for objects within a bucket
  - server access logging: S3 buckets can be configured to create access logs that log all requests made to a S3 bucket, and be written to another bucket
- MFA delete
- pre-signed URLs
- 2 ways to open a file
  - open: authorizes user with AWS credentials
  - object https link: viewed as a unknown traffic
- Versioning
  - keep all versions of an object
  - billed for total size

#### 4.4 Encryption

- in transit (uploading to/downloading from S3)
  - SSL/TLS (transport layer security): HTTPS
- at rest
  - server side encryption
    - SSE-S3: S3 managed keys, each object is encrypted with its own unique key using strong multi-factor encryption, also encrypt the key with a master key
    - SSE-KMS: AWS key management service, managed keys, envelope key that encrypts the data's encryption key, also get audit logs
    - SSE-C: customer provided keys
  - client side encryption
    - user encrypt data -> upload to S3
    - client library such as Amazon S3 Encryption Client
- to enforce encryption on S3 when uploading a file
  - every time a file is uploaded, a PUT request is initiated
  - x-amz-server-side-encryption parameter will be included in the PUT request
    - x-amz-server-side-encryption: AES256: SSE-S3
    - x-amz-server-side-encryption: kms:kms: SSE-KMS
  - update bucket policy to deny any S3 PUT request that doesn't include the x-amz-server-side-encryption parameter in the request header
  - bucket -> permissions -> bucket policy -> policy generator
  - S3 bucket policy -> deny -> \* (principal) -> PutObject (Actions) -> bucket ARN -> add conditions -> StringNotEquals & s3:x-amz-server-side-encryption & aws:kms -> add condition -> add statement
  - copy generated policy JSON to bucket policy
  - add a wildcard to resource ARN: "Resource": "arn:aws:s3:::files/\*"

```
PUT /my-file HTTP/1.1
Host:bucket-name.s3.amazonaws.com
Date: Thu, 10, 2019 15:28:34 GMT
Authorization: authorization string
Content-Type: text/plain
Content-Length: 72424
x-amz-meta-author: Tina Bu
x-amz-server-side-encryption: AES256
Except: 100-continue (S3 not send request body until receive an
```

acknowledgement)  
[72424 bytes of object data]

#### 4.5 Performance Optimization

- S3 is designed to support very high request rates
  - 3,500 PUT requests per second
  - 5,500 GET requests per second
- if you are routinely receiving > 100 PUT / LIST / DELETE or > 300 GET requests per second,
- GET-intensive workloads
  - use CloudFront to cache your most frequently accessed objects
- random key names no longer helps with faster performance
- [LEGACY] mixed request type workloads
  - [LEGACY] key names impacts the performance
  - [LEGACY] AWS uses key names to determine which partition an object should be stored in
  - [LEGACY] use sequential key names (prefixed with a time stamp/alphabetical sequence) increases the likelihood of having multiple objects stored on the same partition, thus can cause I/O issues and contention
  - [LEGACY] thus, use a random prefix forces S3 to distribute your keys across multiple partitions, distributing the I/O workload
  - [LEGACY] e.g. add a prefix of 4-character hexadecimal hash
- - The following sequential Key Names are not optimal:
    - mybucket/2018-03-04-15-00-00/cust1234234/photo1.jpg
    - mybucket/2018-03-04-15-00-00/cust3857422/photo2.jpg
    - mybucket/2018-03-04-15-00-00/cust1248473/photo2.jpg
  - For optimal performance, introduce some randomness into the key name, e.g. prefix the key name with a 4-character hexadecimal hash:
    - mybucket/7eh4-2018-03-04-15-00-00/cust1234234/photo1.jpg
    - mybucket/h35d-2018-03-04-15-00-00/cust3857422/photo2.jpg
    - mybucket/o3n6-2018-03-04-15-00-00/cust1248473/photo2.jpg
- Read the FAQ!

## 5 Networking & Content Delivery

### 5.1 DNS 101

- Naming
  - The "route" part comes from the historic "Route 66" of the USA
  - 53 is port 53 for TCP and UDP
- DNS
  - convert human friendly domain names -> Internet Protocol (IP) address
  - IP addresses are used by computers to identify each other on the network
  - IPv4 or IPv6
    - IPv4
      - 32 bit field
      - 4 billion addresses
      - running out
    - IPv6
      - 128 bits
      - 340 undecillion addresses
- top level domains
  - string of characters separated by dots
  - last word is top level domain
    - .com
    - .edu
    - .gov
  - second level domain name
    - .co.uk: .uk is top level, .co is second level domain
  - Controlled by the Internet Assigned Numbers Authority (IANA)
  - domain registrars
    - assign domain names directly under 1 or more top-level domains
    - Amazon
    - GoDaddy.com
    - 123-reg.co.uk etc.
- SOA record
  - start of authority record
- NS record
  - name server record

- used by top level domain servers to direct traffic to the content DNS server
- "acloudguru.com" -> .com server -> NS records -> SOA (DNS records)
- different types of DNS record
  - A record (A = address)
    - points to a IPv4 address
  - AAAA record
    - IPv6 address
  - CName
    - Canonical Name
    - resolve 1 domain name to another
    - "m.acloudguru.com" -> "mobile.acloudguru.com" -> "IPv4"
    - BATMAN -> "Bruce Wayne" -> 412-412-4121
  - Alias Records
    - map resource records to ELB, CloudFront, or S3 buckets websites
    - DNS -> target name
    - can't be used for naked domain name (without a www.)
    - has to be an A record or an alias
    - always choose an alias record over a CName in an exam scenario
    - Provides a route53-specific extension to DNS functionality
  - MX record
    - Mail exchange
  - PTR record
    - reverse of an A Record
    - address -> domain name
- TTL
  - time to live
  - length a DNS record is cached in the resolving server or user local PC
  - lower TTL, faster changes to DNS records take to propagate throughout the internet
- naked domain name
  - domain.com instead of www.domain.com
  - use A record with an Alias
  - Alias target: S3, ELB, CloudFront, Elastic Beanstalk, etc

## 5.2 Route53

- Domain Registration
  - can buy domain names directly in AWS
  - and map your domain name to

- EC2
- Load Balancer
- S3 bucket
- takes up to 3 days to register
- Routing policies
  - simple routing
    - 1 record with multiple IP address
    - Route53 returns a random value
    - hosted zones -> create record set -> naked name -> A record -> put 3 IPs for EC2 in
    - access tinabu.com and the server will change
    - can modify TTL
  - weighted routing
    - split traffic based on weights
    - e.g. 20% Paris 80% N. Virginia
  - latency-based routing
    - route your traffic based on the lowest network latency for your end user
    - fastest response time
  - failover routing
    - create an active/passive set up
    - e.g. primary site in EU-WEST-2 and secondary disaster recovery site in AP-SOUTHEAST-2
    - failover to passive if active is down
  - geolocation routing
    - choose where your traffic will be sent based on the user's geographic locations
    - continent / country based
  - geoproximity routing
    - let Route53 route traffic based on user geographic location and your resources
    - must use Route53 traffic flow
    - you can influence it with "bias"
    - traffic policy tab
  - multivalue answer routing
    - like simple routing: configure Route53 to return multiple values
    - but plus health checks and return only healthy resources
- Health check
  - create on individual record sets

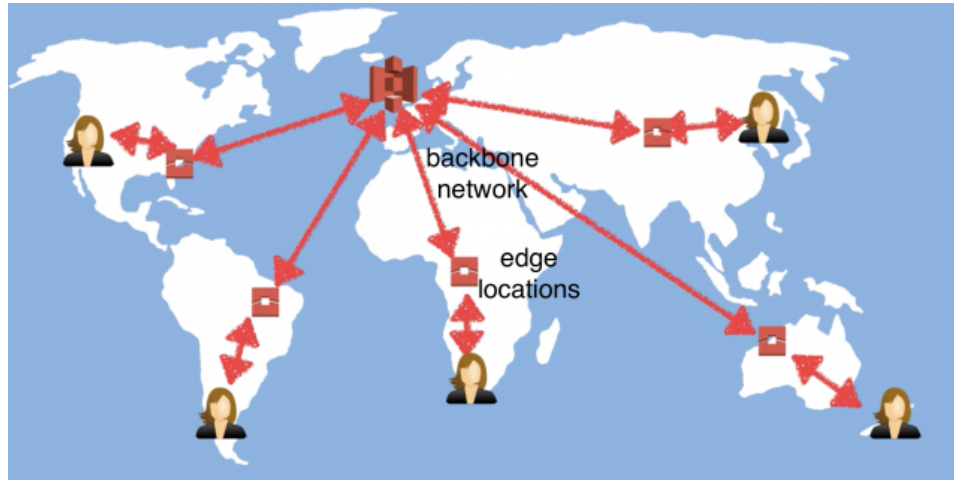


- Add health check to A record
- if a record fails a health check, it will be removed from Route53 until it passes the health check again
- can set SNS notification for health check failures
- Lab
  - Register a domain
    - choose a domain name: .com, .co, etc
    - fill registrant contact information
  - Hosted zones -> Create Record Set
  - Create 3 EC2 in 3 regions
    - N. Virginia 3.87.24.95
    - Seoul 54.180.113.112
    - Paris 35.181.48.173
  - Create a load balancer
    - Application load balancer: internet-facing, IPv4
    - configure target group and health check
    - add EC2s to target group
    - ELBs don't have pre-defined IPv4 addresses, you resolve to them using a DNS name
  - Go to Route53, hosted zones
    - create record set, A record
    - add ELB as the target
    - go to `domain.com` instead of public-ip in browser

### 5.3 CloudFront

- CDN
  - Content Delivery Network
  - a system of distributed servers that deliver content based on the geolocation of the user and the origin of the webpage
  - dynamic, static, streaming, and interactive content
  - global network of edge locations
  - access content at `http(s)://domain-name/filename`
- Edge location
  - where content is cached
  - requests are automatically routed to the nearest edge locations, delivered with the best possible performance
  - separate to region or AZ, many more edge locations than regions/AZs actually (100 edge locations in 25 countries)

- NOT read only
- you can WRITE to them, AWS transfers the files to S3
- cached for TTL (time to live) in seconds
  - default is 24 hours
  - max is 365 days
  - if your files are updated more than once every 24 hours, reduce the TTL to a fraction (1/2, 1/3, 1/4) of the frequency, depending on how critical it is for it to be up-to-date
- can manually clear cached objects with charge (invalidation)
  - on an object basis
- Origin
  - origin of the files that CDN will distribute
    - S3 bucket
    - EC2 instance
    - ELB
    - Route53
    - your own server on premise
    - you can have multiple origins
  - origin access identity
    - You can optionally secure the content in your Amazon S3 bucket so users can access it through CloudFront but cannot access it directly by using Amazon S3 URLs.
    - This prevents anyone from bypassing CloudFront and using the Amazon S3 URL to get content that you want to restrict access to.
- Distribution
  - name given to the CDN
  - a collection of edge locations
  - Web distribution
    - used for websites
    - HTTP/HTTPS
  - RTMP distribution
    - Adobe's realtime messaging protocol
    - used for Flash/media streaming
- S3 transfer acceleration
  - data arrives at an edge location, then routed to Amazon S3 over an optimized network path
  - much faster upload
  - amazon backbone network



- - cross region replication
    - sync a bucket to another region
    - versioning must be enabled on both source and destination
    - regions must be unique
    - files in an existing bucket are not replicated automatically
    - all subsequent updated files will be replicated automatically
    - Delete markers are not replicated
    - deleting individual versions or markers will not be replicated
  - To serve paid content
    - signed cookies
    - signed URLs
  - Security
    - origin access identity
    - WAF
      - web application firewall, protects you at the application layer
      - blocks IP addresses based on rules
      - protects from Cross-site Scripting attacks
    - geo-restriction: whitelist/blacklist countries for your content
    - invalidations: invalidate object in cache
    - Shield
      - protects from Distributed Denial-of-Service attacks
    - Macie
      - uses Machine Learning to protect sensitive data

## 5.4 API Gateway

- API

- application programming interface
- REST API
  - Representational State Transfer 17%

```
{
  "_id": "375bc02dk02r93yb72",
  "firstname": "Tina",
  "lastname": "Bu",
}
```

- SOAP APIs
  - Simple Object Access Protocol
  - can use API gateway as a SOAP web service passthrough

```
<?xml version="1.0"?>
<quiz>
  <qanda seq="1">
    <question>
      Who was the 42 president of the US?
    </question>
    <answer>
      William Jefferson Clinton
    </answer>
  </qanda>
</quiz>
```

- developers can publish, maintain, monitor, and secure APIs at any scale
- "front door": for applications to access your backend services
  - applications: EC2, Lambda, any web application
  - backend: data, business logic, functionality
- expose HTTPS endpoints to define a RESTful API
- serverless-ly connect to services like Lambda, DynamoDB
- send each API endpoint to a different target
- scales effortlessly
- runs efficiently with low cost
- track and control usage by API key
- throttle requests to prevent attacks
  - The Throttling filter can protect a Web Service or Service Oriented Architecture (SOA) from message flooding. You can configure the filter to only allow a certain number of messages from a specified client in a given time frame through to the protected system.

- by default, limits the steady-state request rate to 10,000 requests per second (rps) (10 requests per millisecond)
- max concurrent requests is 5,000 requests across all APIs within an AWS account
- if go over either of the 2 limits, you will receive a *429 Too Many Request* error
- connect to CloudWatch to log all requests for monitoring
  - Access logging
    - customized
    - log who has accessed your API and how
  - Execution logging
    - API Gateway manages the CloudWatch Logs
    - errors or execution traces (such as request or response parameter values or payloads), data used by Lambda authorizers (formerly known as custom authorizers), whether API keys are required, whether usage plans are enabled, and so on.
- maintain multiple versions of an API
- endpoints
  - hostname of the API
  - HTTPS for RESTful API
  - edge-optimized: geographically distributed clients, access across regions
  - regional: clients in the same region
  - private: only accessible from VPC
- Configuration Steps
  - define an API (container)
  - define resources and nested resources (URL Paths)
  - for each resource
    - select supported HTTP methods (PUT, GET, POST, DELETE)
    - set security
    - choose target
      - EC2, Lambda, DynamoDB
    - set request and response transformations
  - Deploy API to a stage
    - dev, prod, etc
    - uses API gateway domain by default
    - can use custom domain
  - supports AWS certificate manager: free SSL/TLS certs!
- API caching
  - cache your endpoint's response

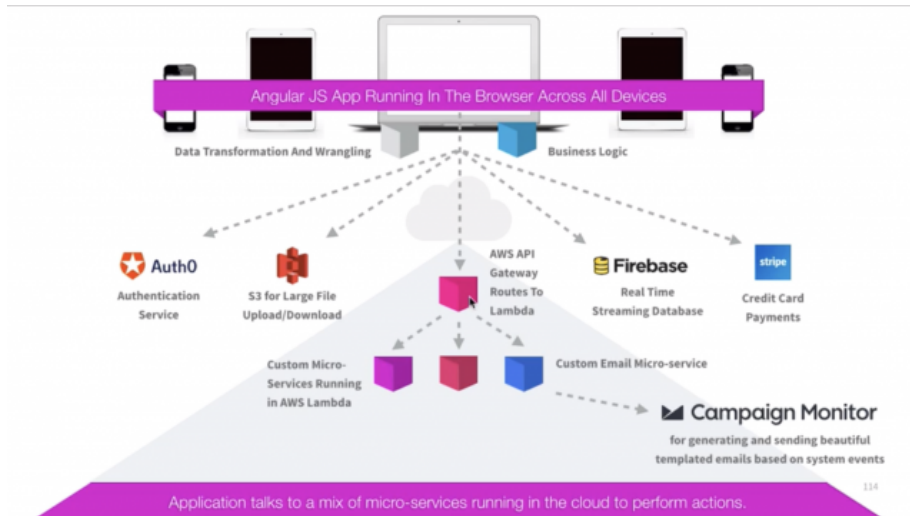
- reduce the number of calls made to your endpoint
- improve the latency
- cache responses from your endpoint for time-to-live (TTL) period in seconds
- like pre-made burger at McDonald's
- Same-origin policy
  - a web browser permits scripts contained in a first web page to access data in a second web page, but only if both have the same origin (domain name)
  - to prevent cross-site scripting (XSS) attacks
  - enforced by browser
  - ignored by tools like PostMan and `curl`
- CORS (cross origin resource sharing)
  - one way the server end (not client end in the browser) can relax the same-origin policy
  - allows restricted resources (e.g. fonts) on a web page to be requested from another domain outside the domain from which the first resource was served
  - browser makes an HTTP `OPTIONS` call for a URL
  - server returns a response says: "these other domains are approved to `GET` this URL"
  - ERROR: "origin policy cannot be read at the remote resource", you need to enable CORS on API gateway
  - if you are using Javascript/AJAX that uses multiple domains with API gateway, ensure that you have enabled CORS on API gateway
  - enforced by the client
- API Keys
  - to identify an API client and meter their access
  - put quota limits on client
- Import API
  - import an API
  - from an external definition file into API Gateway
  - e.g. Swagger v2.0 definition files
  - `POST` request that includes the Swagger definition in the payload and endpoint configuration
  - or `PUT` request with the Swagger definition in the payload to update an existing API
  - update an API by overwriting it with a new definition
  - merge a definition with an existing API
  - mode query parameter in the URL
  - Lab

- **API Gateway -> Create API -> Import from Swagger -> Import**

## 6 Serverless Computing

### 6.1 Serverless 101

- Some History of Cloud
  - Data Center (physical hardware, provisioned, in racks, assembly code/protocols, high level languages, OS, Application layer) ->
  - IAAS (EC2 launched in 2006, something can still go wrong, getting hacked, broke down) ->
  - PAAS (Elastic Beanstalk, only need to upload code without worrying about infrastructure configuration, still have to manager the server) ->
  - Containers (2014 Docker, lightweight, isolated, deployed to a server) ->
  - Serverless (2016 Lambda, Code)
    - ACloudGuru Architecture



- Traditional vs Serverless Architecture
  - Traditional
    - user -> route53 -> load balancer -> web server -> backend DB server -> response to user
  - Serverless
    - user -> API gateway -> Lambda -> backend DB server -> response to user
    - serverless services: Lambda, API gateway, S3, Dynamo DB, Aurora
    - RDS is not serverless except Aurora Serverless
    - data centers, hardware, assembly code/protocols, high level languages,



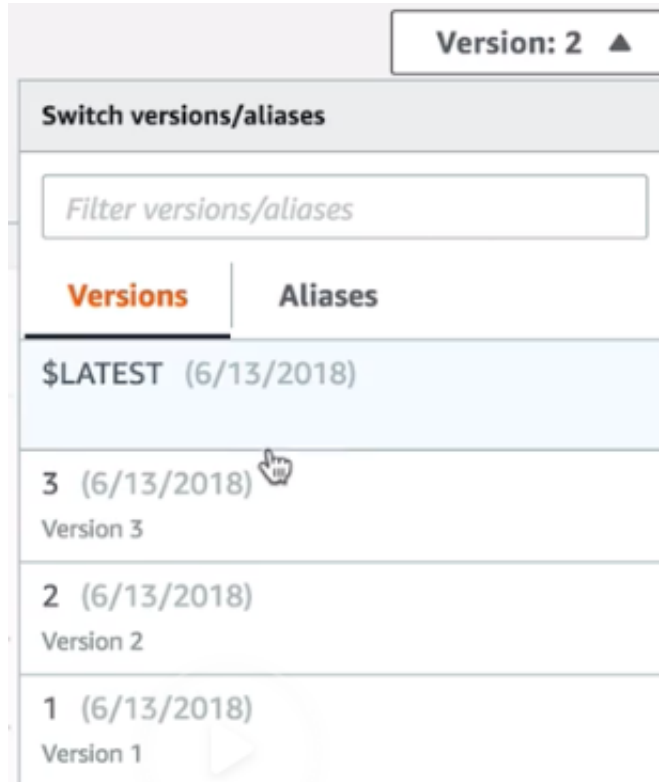
operating systems, application layer/AWS APIs, AWS Lambda

## 6.2 Lambda

- Lambda
  - a compute service where you can upload your code and create a Lambda function.
  - Lambda takes care of provisioning and managing the servers you use to run the code.
  - You don't need to worry about OS, patching, scaling, etc
  - independent
    - 1 event = 1 function
  - lambda functions can trigger other lambda functions
    - 1 event = x functions (fan out)
  - debug: AWS X-ray
  - can do things globally, can use it to back up S3 buckets
- 2 use case
  - event-driven compute service in response to events, e.g. change to data in S3, change to data in Dynamo
  - compute service in response to HTTP requests with API Gateway or API calls made with AWS SDKs. e.g. Amazon Alexa
- Languages
  - Node.js, Java, Python, Ruby, Go, .NET
- Price
  - number of requests
    - first 1 million requests are free per month
    - \$0.20 per 1 million requests thereafter
  - duration
    - time your code is executed
    - and memory usage
- Benefits
  - no servers
  - continuous scaling
    - scales out automatically
    - not scale up
  - super cheap
- Triggers
  - API Gateway, AWS IoT, Alexa Skills Kit, Alexa Smart Home, CloudFront, CloudWatch Events, CloudWatch Logs, CodeCommit, Cognito Sync Trigger,

DynamoDB, Kinesis, S3, SNS

- S3 events can trigger
- RDS can't trigger
- version control
  - you can publish one or more versions of your Lambda function
  - each version has a unique ARN
  - once published, it's immutable
  - the latest function code has version \$LATEST
  - qualified version (ARN) will use \$LATEST , unqualified will not have it
  - you can also create aliases for versions (like a pointer)
  - In the portal, under Qualifiers are the versions and aliases
  - Actions -> publish new version -> immutable now
  - can only update code under the \$LATEST version



- Actions -> Create Alias -> point it to a version
- create splits between traffic for A/B testing
  - can't have \$LATEST in there, need to create an alias first
  - only 2-ways

Create a new alias

×

An alias is a pointer to one or two versions. Select the version(s) you would like the alias to point to.

Name\*

MySplitTraffic

Description

Version\*

3

Weight: 75%

You can shift traffic between two versions, based on weights (%) that you assign. Click [here](#) to learn more.

Additional Version

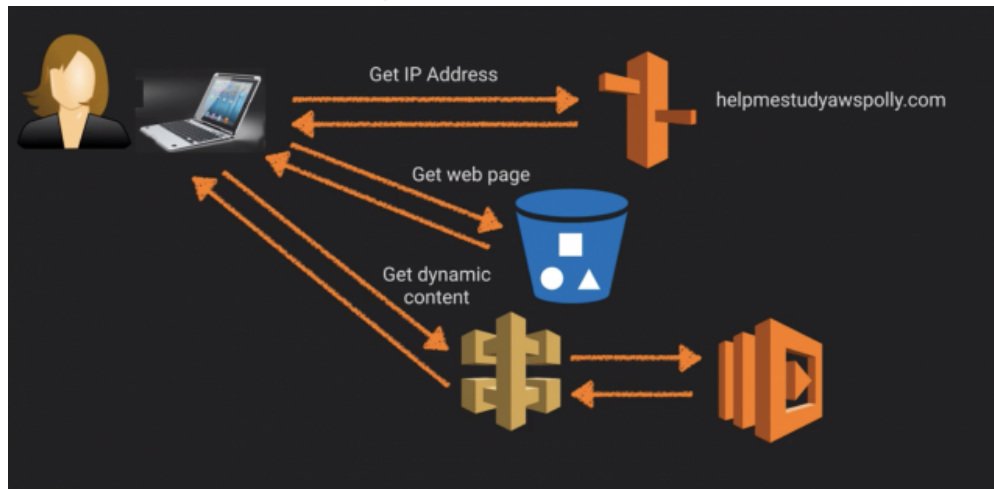
2

Weight

25 %

Cancel Create

- Lab: Build a serverless webpage



- Register a domain name and host a static website with S3
  - check if domain name is available
  - check if bucket name is available
  - domain name has to be the same as bucket name
  - bucket name will include the .com or other domain
  - Create a S3 bucket, choose static website hosting
  - edit public access settings: make it public by uncheck everything so "Objects can be public"
  - with Route53 register a domain
- Create a lambda function
  - Author from scratch
  - put in name

- select Python 3.6
- create new role from template (simple microservice permissions policy template)
- create function
- modify function code with following:

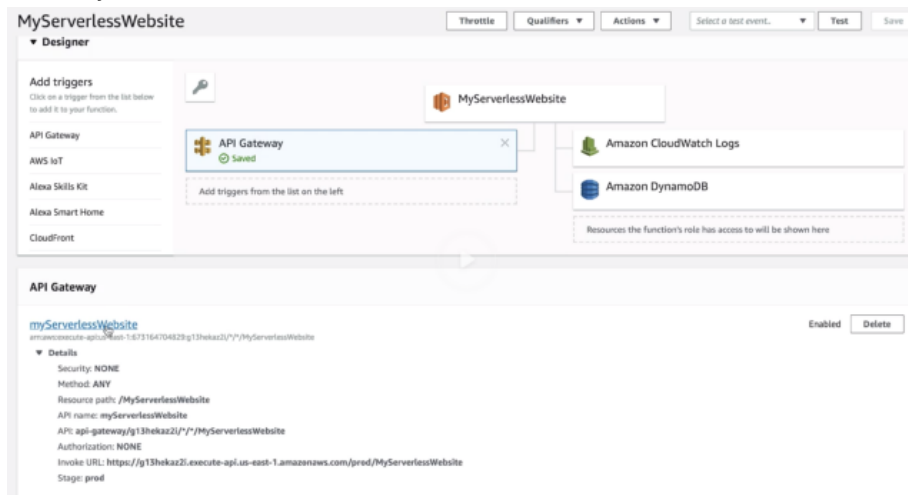
```
import json

def lambda_handler(event, context):
    print('In Lambda handler')

    resp = {
        'statusCode': 200,
        'headers': {
            "Access-Control-Allow-Origin": "*",
        }
        'body': json.dumps('Hello from Tina\'s Lambda function!')
    }

    return resp
```

- Add a trigger
  - Select **API Gateway** -> **Configure triggers** -> **Create a new API**
  - Security: AWS IAM, ADD



- Configure API Gateway to invoke your Lambda function
  - click on the blue name hyperlink, not the Invoke URL
  - Actions: delete the ANY method
  - Create a GET method

- integration type: Lambda function
- select Lambda Proxy integration
- choose a region, and add Lambda function name -> **save** -> **OK**
- **Actions** -> deploy API with default stage
- left bar Stages -> GET method -> click on Invoke URL -> should see message in `json.dumps()`
- Open `index.html` and update the API link
  - line 11 `xhttp.open("GET", "YOUR-API-GATEWAY-LINK", true)`
  - function on a button, that's going to go to the API gateway and display the response of Lambda function
- upload `index.html` and `error.html` to S3
  - make them public from More -> make public
- Configure a domain name for your S3 hosted website
  - Route53 -> hosted zones
  - create record set
  - A-record, yes to alias, choose your S3 bucket name as your alias target
- Click on "click me" which triggers our lambda function, should see the text change

### 6.3 Alexa Skill

- Different components
  - speech recognition
  - Natural language understanding
  - text to speech
  - skills
  - learning
- Lab: Create an Alexa Skill that calls a Lambda Function
  - Create a S3 bucket
    - edit public access settings: make it public by uncheck everything
    - give all objects in the bucket public access by:
    - **Permissions** -> **bucket policy** -> copy and paste with the correct S3 ARN

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
```

```

        "Action": "s3:GetObject",
        "Resource": "arn:aws:s3:::1bigthing-alexa-skill/*"
    }
]
}

```

- Polly
  - Add some plain text
  - choose the language and voice -> **Synthesize to S3**
  - put in your S3 name -> **synthesize**
  - this will create a mp3 file and save it in S3 bucket
- Lambda Function
  - **Create a function -> From AWS Serverless Application Repository -> alexa-skills-kit-nodejs-factskill -> deploy**
  - go into the function, trigger should be Alexa Skills Kit
  - copy your ARN
- Create an Alexa skill
  - Create a Alexa skill from <https://developer.amazon.com/alexa/>, use the same email you registered your Alexa Echo
  - In Developer Console: **Amazon Alexa -> Your Alexa Consoles -> Skills -> Create Skill -> choose the default language as your Echo's language -> Create Skill**
  - Choose a template: **Fact Skill -> Choose**
  - Invocation: change skill invocation name -> **Save Model**
  - Endpoint: paste Lambda ARN to Default Region -> **Save Endpoint**
  - Intents: add utterances -> **Save Model -> Build model**
  - Test: choose **Development** -> In Alexa Simulator, enter some speech
- Make the Alexa skill play the mp3 file in our S3 bucket
  - S3: bucket -> copy the object URL
  - Lambda:

```

const data = [
    '<audio src = \'https://s3.amazonaws.com/bucket-name/object-hash.mp3\' />',
]

```

## 6.4 Step Functions

- allows you to visualize and test your serverless applications
- graphical console

- automatically triggers and tracks each step
- retries when there are errors
- logs the state of each step
- JSON based amazon states language (ASL)
- **Application Integration -> Step Functions -> Create state machine -> Sample Projects -> Job Poller** (asynchronize job with Lambda and Batch) -> **Next -> Deploy resources -> Start execution**
- To delete all resources: **CloudFormation -> Actions -> Delete stack**

### 6.5 X-Ray

- collects data about requests that your application servers
- identify issues and opportunities for optimization
- X-Ray SDK stored in your application -> JSON to X-Ray Daemon -> X-Ray API -> X-Ray Console
- SDK provides
  - *Interceptors* to add to your code to trace incoming HTTP requests
  - *Client handlers* to instrument AWS SDK clients that your application uses to call other AWS services
  - An *HTTP client* to use to instrument calls to other internal and external HTTP web services
- Integration with
  - Elastic Load balancing
  - Lambda
  - API Gateway
  - EC2
  - Elastic Beanstalk
- Languages
  - Java, Go, Node.js, Python, Ruby, .NET
  - same as Lambda
- Lab
  - <https://console.aws.amazon.com/elasticbeanstalk/#/newApplication?applicationname=scorekeep&solutionStackName=Java>
  - **Generate Sample Traffic** -> X-Ray console
  - IAM -> Roles -> aws-elasticbeanstalk-ec2-role, Add policies: AWSXrayFullAccess, AmazonS3FullAccess, AmazonDynamoDBFullAccess
  - **Developer Tools** -> **X-Ray**: should see a service map
  - **View traces** to debug

## 7 Other AWS Services

### 7.1 SQS - Simple Queue Service

- Concept
  - message queue that stores messages while waiting to be processed
  - decouple infrastructure
  - buffer (temporary repository) between components
  - can configure auto-scaling groups (producer & consumer has different workload)
  - distributed queue system
  - decouple components
  - oldest AWS service
- message
  - default 256 KB of text in any format
  - up to 2GB (stored in S3 instead)
  - persistence
    - 1 min to 14 days
    - default retention period 4 days
  - visibility timeout
    - amount of time the message is invisible in the SQS after a reader picks up that msg
    - deleted if message is processed after visibility timeout expires
    - visible again if message is not processed so in case the first EC2 died, another one can pick it up after the time out (fail safe)
    - may result in a message being delivered twice
    - default 30 sec, max 12 hours
    - update this if your task takes >30 sec
    - with the `ChangeMessageVisibility` parameter
- pull-based
  - short polling
    - keeps polling, even if queue is empty
    - returns immediately
  - long polling
    - doesn't return until a message arrives in the queue or the long poll times out



- maximum 20 seconds for a long-poll timeout
- 2 types
  - standard queue
    - default
    - nearly-unlimited number of transactions per second
    - at least once
    - occasionally more than 1 copy of a message might be delivered out of order
    - generally delivered in same order as they are sent, but no guarantee
      - best-effort ordering
  - FIFO queue
    - first-in-first-out: order strictly preserved
    - allow multiple ordered message groups within a single queue
    - exactly once: no duplicates
    - 300 transactions per second (TPS)

## 7.2 SNS - Simple Notification Service

- web service for message publications
  - publish messages from an application
  - deliver them to subscribers or other applications
  - instantaneous push notifications (push based)
- Publish-subscribe (pub-sub)
- multiple transport protocol
  - mobile: Apple, Google, Fire OS, Windows, Android
  - SMS text messages or email to SQS queues
  - HTTP endpoint
  - trigger Lambda functions
    - message payload as input parameter
    - Lambda manipulate the information in the message, publish the message to another SNS topic or send the message to other AWS services, like Polly or S3
- group multiple recipient using *topics*
  - topics: access point for subscribers
  - 1 topic - M endpoint types, e.g. you can group iOS, Android and SMS recipients
- stored redundantly across multiple AZ
- pricing
  - pay-as-you-go model with no upfront costs

- \$0.5 per 1 million SNS requests

	<b>SNS</b>	<b>SQS</b>
common	messaging services	messaging services
difference	pull (poll)	push

### 7.3 SES - Simple Email Service

- send automated emails
  - marketing, notification, and transactional
  - e.g. purchase confirmation, shipping notification, order status update
- receive emails
  - delivered automatically to an S3 bucket
  - can be used to trigger Lambda functions & SNS notifications
- pricing
  - pay-as-you-go

	<b>SES</b>	<b>SNS</b>
name	simple email service	simple notification service
messages	email only	email, SMS, SQS, HTTP (multiple formats)
can trigger	Lambda or SNS notification	Lambda
email	incoming & outgoing email	push notifications only
prerequisite	user's email address, not subscription based	user needs to subscribe to a topic, you can fan out messages to large number of recipients

### 7.4 Kinesis

- accepts streaming data
  - generated continuously
  - by thousands of data sources
  - send in the data records simultaneously
  - small sizes KB
  - e.g. amazon purchases, stock prices, game data (as the gamer plays), social network data, geospatial data like Uber, IoT sensor data

- Kinesis
  - a platform on AWS to send your streaming data to
  - load and analyze streaming data
  - build your custom application
- 3 core Kinesis services
  - Kinesis Streams
    - store streaming data in shards
      - 5 transactions per second for reads
      - up to a max of total data read rate of 2 MB per second
      - up to 1,000 records per second for writes
      - up to a max of total data write rate of 1 MB per second (including the partition keys)
      - the data capacity of your stream is the sum of the capacities of its shards
    - persistent: 24 hours by default, up to 7 days
  - Kinesis Firehose
    - no shards management
    - no data consumers management
    - no data persistent
    - optional lambda function
    - output immediately (to S3, Redshift, Elasticsearch Cluster)
  - Kinesis Analytics
    - analyze data in Streams/Firehose with SQL queries
    - store results in S3/Redshift/Elasticsearch Cluster
- Lab
  - CloudFormation
    - create a new stack `myKinesesStack` with template in S3
    - EC2 URL: both data producer and consumer
  - Kinesis
    - Streams
  - DynamoDB
    - tables created for data storage

## 7.5 Elastic Beanstalk

- deploy and scale web applications
  - language: Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker
  - server platforms: Apache Tomcat, Nginx, Passenger, PUMA, IIS
  - developer don't need to worry about the underlying infrastructure needed to

- run the application
  - similar to CloudFormation but not JSON based, with GUI and clicks
- under Compute services
- Steps
  - upload the code in .zip
  - Elastic Beanstalk handles the deployment, capacity provisioning, load balancing, auto-scaling and application health check
- Deployment policies options

	<b>All at Once</b>	<b>Rolling</b>	<b>Rolling with Additional Batch</b>	<b>Immutable</b>
Deploy new version	to all instances simultaneously	in batches	launches an additional batch of instances, and deploys new version in them	to a fresh group of instances in their own autoscaling group, moved to your existing autoscaling group, finally old instances are terminated
Instances while deployment	out of service	each batch is out of service	on	on
performance	outage, downtime	reduced performance	maintains full capacity	maintains full capacity
Failed update	roll back by re-deploying the original version to all instances	perform an additional rolling update to roll back the changes	perform an additional rolling update to roll back the changes	only need to terminating new auto scaling group
Use case	not for mission-critical systems	not for performance sensitive systems	performance sensitive systems	mission-critical production systems

- you only pay for the AWS resources
- you have full administrative control of the underlying AWS resources
  - or Elastic Beanstalk can do it for you
- *Managed Updates*

- automatically applies updates to your OS, Java, PHP, Node.js, etc
- monitor and manage via a dashboard
- integrated with CloudWatch and X-Ray for performance metrics
- customize environment with configuration files
  - e.g. define packages to install, create Linux users and groups, run shell commands, specify services to enable or configure your load balancer
  - YAML/JSON
  - save with `.config` extension and inside folder `.ebextensions` in the top-level directory of your source code bundle
  - example: configures the health check URL for load balancer
  - **my-healthcheck.config**

```
{
  "option_settings":
    [
      {
        "namespace": "aws:elasticbeanstalk:application",
        "option_name": "My Application Healthcheck URL",
        "value": "/healthcheck"
      }
    ]
}
```

- RDS and Elastic Beanstalk
  - launch the RDS instance from within the Elastic Beanstalk console
    - in your Elastic Beanstalk environment
    - for dev & test deployments
    - not for production as the lifecycle of the database is tied to the lifecycle of the application environment. If environment terminated, the database instance will be terminated too.
  - launch the RDS instance in the RDS console
    - decouple the RDS instance from your EBS environment
    - production environments
    - connect multiple environments to the same database
    - more choice of database types
    - can tear down your application environment without affecting the database instance
    - add an additional security group to your environment's auto scaling group
    - provide connection string configuration to your application servers

(endpoint, password)

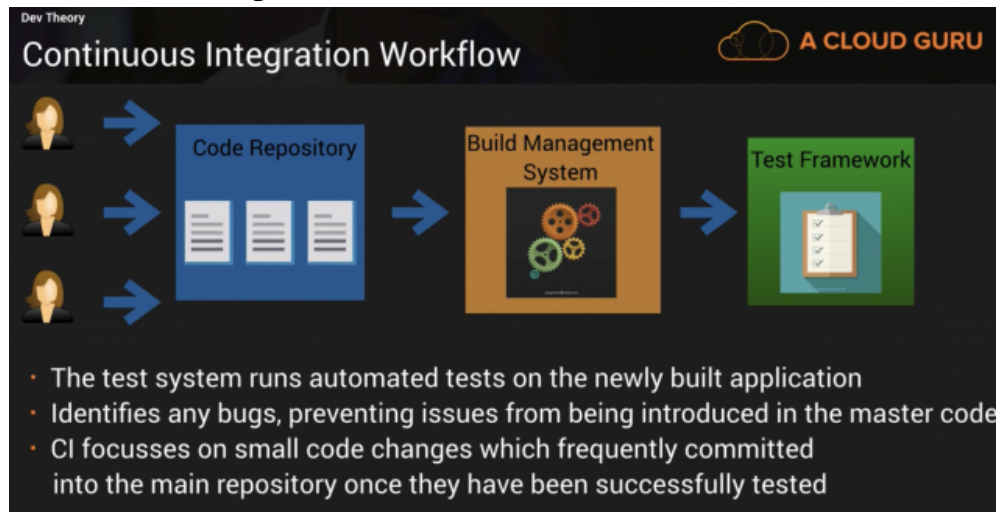
### **7.6 Systems Manager Parameter Store**

- under EC2 -> **System Manager Shared Resources** -> **Parameter Store** -> save a secret (passwords, database connection strings, license codes)
  - plain text
  - encrypt
- parse to CloudFormation or Lambda or EC2
- reference the values by using their names

## 8 Developer Theory

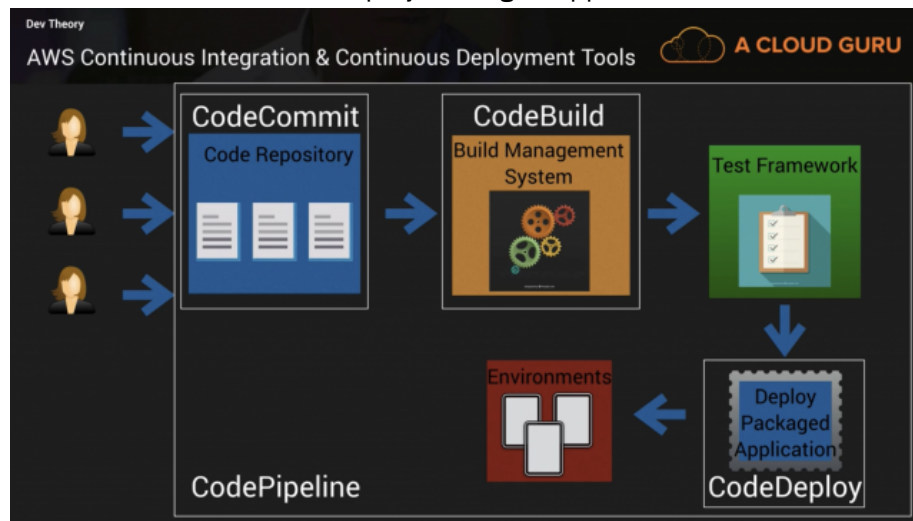
### 8.1 CI/CD

- enable frequent software changes to be applied whilst maintaining system and service stability
- CI
  - Continuous Integration



- Code Repository ->
  - multiple developers working in on different features or bug fixes
  - all contributing to the same application
  - sharing the same code repository e.g. git
  - frequently pushing their updates into the shared repo - at least daily
- build management system ->
  - code changes trigger an automated build, every commit
  - we need a way to ensure that any code change does not break the build or introduce new bugs in the application
- Test framework ->
  - the test system runs automated tests on the newly built application
    - unit tests
    - integration tests
    - functional tests
  - identifies any bugs, preventing issues from being introduced in the master code

- CI focuses on small code changes which frequently committed into the main repository once they have been successfully tested
- CD
  - Continuous Delivery/Deployment
  - Continuous Delivery
    - a development practice where merged changes are automatically built, tested, and prepared for release into staging and eventually production environments
    - a manual decision process to initiate deployment of the new code
    - Test Framework -> (user) Deploy Packaged Application -> Environments
  - Continuous Deployment
    - takes the idea of automation one step further and automatically deploys the new code following successful testing, eliminating any manual steps (automates the release process)
    - the new code is automatically released as soon as it passes through the stages of your release process (build, test, package)
    - small changes are released early and frequently, rather than grouping multiple changes into a larger release
    - Test Framework -> (auto) Deploy Packaged Application -> Environments



## 8.2 CodeCommit

- source control service
- secure and highly scalable private Git repositories
  - industry-standard open source distributed source control system
  - centralized repository for all your code, binaries, images, and libraries



- tracks and manages code changes
- maintains version history
- manages updates from multiple sources
- enables collaboration
- new branch -> commits -> (test/peer review) -> merge back into master by creating a pull request
- encrypted in transit and at rest
  - HTTPS
  - SSH
- **Settings -> Notifications** -> SNS topic / CloudWatch event

### 8.3 CodeDeploy

- automated deployment service
  - EC2 instances
  - on premises systems
  - Lambda functions
- auto-scales with your infrastructure
- integrates with various CI/CD tools
  - Jenkins, GitHub, Atlassian, AWS CodePipeline
- integrates with config management tools
  - Ansible, Puppet, Chef
- 2 deployment approaches

	In-Place/Rolling update	Blue/Green
process	the application is stopped on each instance in turn, the latest revision will be installed on the stopped instance (if the instance is behind a load balancer, you can configure the LB to stop sending requests to the instances which are being upgraded)	new instances are provisioned and the latest revision is installed, registered with an elastic load balancer, traffic is then routed to them and the original instances terminated
performance	the instance is out of service during this time and your capacity will be reduced	no reduction in performance/capacity during deployment (new instances can be created ahead of time, code can be released to production by simply switching

		all traffic to the new servers (just a change on your load balancer))
deployment on	EC2, on-premise	EC2, on-premise, and Lambda
roll back	the previous version need to be re-deployed	faster and more reliable. Just routing the traffic back to the original servers (as long as you haven't already terminated them)
notes		blue: active deployment (original), green: new release

---

- Terminology
  - Deployment Group: a set of EC2 instances/Lambda functions to which a new revision of the software is to be deployed
  - Deployment: the process and component used to apply a new revision
  - Deployment Configuration: a set of deployment rules as well as success/failure conditions used during a deployment
  - AppSpec File: defines the deployment actions you want AWS CodeDeploy to execute
  - Revision: everything needed to deploy the new version, AppSpec file, application files, executables, config files
  - Application: unique identifier for the application you want to deploy. Ensures the correct combination of revision, deployment configuration and deployment group are referenced during a deployment
- AppSpec File
  - defines the parameters that will be used for a CodeDeploy deployment
  - file structure
    - Lambda deployment: YAML/JSON
      - version: reserved for future use, currently only allows 0.0
      - resources: name and properties of the Lambda function
      - hooks: specifies Lambda functions to run at set points in the deployment lifecycle to validate the deployment. e.g. validation tests to run before allowing traffic to be sent to your newly deployed instances
        - BeforeAllowTraffic: specify the tasks or functions you want to run before traffic is routed to the newly deployed Lambda function e.g. test to validate the function has been deployed correctly

- AfterAllowTraffic: specify the tasks or functions you want to run after the traffic has been routed to the newly deployed Lambda function e.g. test to validate the function is accepting traffic correctly and behaving as expected
- example

```
version:0.0
resources:
  - myLambdaFunctionName:
      Type: AWS::Lambda::Function
      Properties:
        Name: "myLambdaFunctionName"
        Alias: "mylambdaFunctionAlias"
        CurrentVersion: "1"
        TargetVersion: "2"
hooks:
  - BeforeAllowTraffic:
      "LambdaFunctionToValidateBeforeTrafficShift"
  - AfterAllowTraffic:
      "LambdaFunctionToValidateAfterTrafficShift"
```

- EC2/on-premises: YAML
  - version: reserved for future use, currently only allows 0.0
  - os: the operating system version you are using
  - files: the location of any application files that need to be copied and where they should be copied to
  - hooks: lifecycle events allow you to specify scripts that need to run at set points in the deployment lifecycle
    - e.g. to unzip application files prior to deployment, run functional tests on the newly deployed application
    - run order of hooks



- Deregister instance from load balancer
  - BeforeBlockTraffic: run tasks on instances before they are deregistered from a load balancer
  - BlockTraffic: deregister instances from a load balancer
  - AfterBlockTraffic: run tasks on instances after they are deregistered from a load balancer
- Upgrade an application
  - ApplicationStop: gracefully stop the application in preparation for deploying the new revision
  - DownloadBundle: CodeDeploy agent copies application revision files to a temporary location
  - BeforeInstall: pre-installation scripts, e.g. backing up the current version, decrypting files
  - Install: CodeDeploy agent copies application revision files from temporary location to the correct location
  - AfterInstall: post-installation scripts, e.g. configuration tasks, change file permissions, add executable access to a file
  - ApplicationStart: restarts any services that were stopped during ApplicationStop
  - ValidateService: tests to validate the service
- Register instance to load balancer
  - BeforeAllowTraffic: run tasks on instances before they are registered from a load balancer
  - AllowTraffic: register instances with a load balancer
  - AfterAllowTraffic: run tasks on instances after they are registered from a load balancer
- example

```

version: 0.0
os: linux
files:
  - source: Config/config.txt
    destination: /webapps/Config
  - source: Source
    destination: /webapps/myApp
hooks:
  BeforeInstall:
    - location: Scripts/UnzipResourceBundle.sh
    - location: Scripts/UnzipDataBundle.sh
  AfterInstall:
    - location: Scripts/RunResourceTests.sh
      timeout: 180
  ApplicationStart:
    - location: Scripts/RunFunctionTests.sh
      timeout: 3600
  ValidateService:
    - location: Scripts/MonitorService.sh
      timeout: 3600
      runas: codedeployuser

```

- `appspec.yml` must be placed in the root directory of your revision, otherwise the deployment will fail
- typical folder setup

```

appspec.yml
/Scripts
/Config
/Source

```

- Lab:
  - Create roles
    - Create a role for EC2 with `AmazonS3FullAccess` policy
    - Create a role for CodeDeploy with `AWSCodeDeployRole` policy
  - Create user
    - programmatic access
    - with access to `CodeDeployFullAccess` and `AmazonS3FullAccess`

```

aws configure
# put in the access key and secret access key

```

- EC2
  - launch an instance with the EC2 role created above
  - add tag: AppName: mywebapp
  - set security group to allow SSH and HTTP

```
ssh -i keyname.pem ec2-user@public-ip

# install CodeDeploy agent on EC2
sudo yum update
sudo yum install ruby -y
sudo yum install wget
cd /home/ec2-user
# change this to the correct region
wget https://aws-codedeploy-us-east-1.s3.amazonaws.com/latest/install
# add execute permission to the installation file
chmod +x ./install
sudo ./install auto
sudo service codedeploy-agent status
```

- Create Code repo
  - **appspec.yml**
    - define all the parameters needed for a CodeDeploy deployment

```
version: 0.0
os: linux
files:
  - source: /index.html
    destination: /var/www/html/
hooks:
  BeforeInstall:
    - location: scripts/install_dependencies.sh
      timeout: 300
      runas: root
    - location: scripts/start_server.sh
      timeout: 300
      runas: root
  ApplicationStop:
    - location: scripts/stop_server.sh
      timeout: 300
      runas: root
```

- **install\_dependencies.sh**

```
#!/bin/bash
yum install -y httpd
```

- **start\_server.sh**

```
#!/bin/bash
service httpd start
```

- **stop\_server.sh**

```
#!/bin/bash
service httpd stop
```

- Put code into a S3 bucket

- create a bucket

```
# create your application.zip
aws deploy create-application --application-name mywebapp

# load it into CodeDeploy and S3 bucket
aws deploy push --application-name mywebapp --s3-location
s3://bucket-name/webapp.zip --ignore-hidden-files
```

- CodeDeploy

- click on app mywebapp -> **Create deployment group** -> select service role created earlier -> in-place -> **EC2 instances** for environment configuration -> put in the EC2 tags -> **Create deployment group**
  - **Create deployment** -> select the deployment group -> select S3 bucket as **Revision location** -> choose **Additional deployment behavior settings** -> choose **Rollback configuration overrides** settings

## 8.4 CodePipeline

- CI/CD workflow tool
- model, visualize, and automate the entire release process
- orchestrate build, test, deployment every time there is a change to your code
- all based on a user defined software release process
- allows frequent releases in a reliable way
- code update -> build -> test -> deploy
- modeled with GUI or CLI
- Integrates with

- CodeCommit, CodeBuild, CodeDeploy, Lambda, Elastic Beanstalk, CloudFormation, Elastic Container Service
- GitHub, Jenkins
- every code pushed to your repo (CodeCommit/S3) automatically enters the workflow and triggers the set of actions (with a CloudWatch event trigger)
- the pipeline automatically stops if one of the stages fails, changes will roll back
- Lab Step 1: Create an EC2 with CloudFormation, upload code v1.0 to S3, deploy application to EC2 with CodeDeploy.
  - everything needs to be in same region
  - CloudFormation
    - create a new S3 bucket
    - save CF\_Template.json to your bucket
    - creates EC2 instance, tag it, and associate the key pair, set a security group, select instance regions

```
aws cloudformation create-stack --stack-name
CodeDeployDemoStack \
--template-url http://s3-us-east-1.amazonaws.com/<bucket-
name>/CF_Template.json \
--parameters ParameterKey=InstanceCount,ParameterValue=1 \
ParameterKey=InstanceType,ParameterValue=t2.micro \
ParameterKey=KeyPairName,ParameterValue=<pem-key-name> \
ParameterKey=OperatingSystem,ParameterValue=Linux \
ParameterKey=SSHLocation,ParameterValue=0.0.0.0/0 \
ParameterKey=TagKey,ParameterValue=Name \
ParameterKey=TagValue,ParameterValue=CodeDeployDemo \
--capabilities CAPABILITY_IAM
```

- IAM
  - give your user permission to access CloudFormation
  - add AmazonS3FullAccess, AWSCodeDeployFullAccess policy
  - create this policy, and attach it to the user

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "cloudformation:*",
      "Resource": "*"
    }
  ]
}
```



```

    },
    {
      "Effect": "Allow",
      "Action": "iam:*",
      "Resource": "*"
    },
    {
      "Action": "ec2:*",
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}

```

- Check we are good to go

```

# Verify that the Cloud Formation stack has completed
aws cloudformation describe-stacks --stack-name
CodeDeployDemoStack --query "Stacks[0].StackStatus" --output
text

# Log in to your instance and check that the CodeDeploy agent
has correctly installed
ssh -i key-name.pem ec2-user@public-ip
sudo service codedeploy-agent status

```

- S3
  - create a bucket
  - enable versioning
  - upload v1.0 code mywebapp.zip
    - need to be a compressed file
- CodeDeploy
  - **Create application** -> select EC2
  - **Create deployment group** -> select the service role -> in-place -> **EC2 instances** for environment configuration -> put in the EC2 tags -> uncheck load balancer -> **Create deployment group**
  - **Create deployment** -> select the deployment group -> select S3 bucket as **Revision location**
    - check by opening the EC2 public-ip in browser
- Lab Step 2: Build CodePipeline and manually trigger CodeDeploy to update to code v2.0.

- S3
  - upload v2.0 code `mywebapp.zip`
- CodePipeline
  - create a pipeline
  - **Source provider:** S3, put in bucket name, file name
  - detection options: **CloudWatch Events**
  - skip build stage
  - **Deploy:** CodeDeploy
  - check by opening the EC2 public-ip in browser
- Lab Step 3: Finally configure automated pipeline, which triggers a new deployment when we upload v3.0 to our S3 bucket.
  - S3
    - upload v3.0 code `mywebapp.zip`
  - check by opening the EC2 public-ip in browser

## 8.5 Docker and CodeBuild

- Docker
  - a lightweight standalone executable software packages
  - includes everything the software needs to run the code
    - runtime environment
    - libraries
    - environment settings
  - Elastic Container Service
- CodeBuild
  - a build service that runs a set of commands that you define
    - compile code
    - run tests
    - produces artifacts ready to deploy
- Lab: CodeBuild takes source code from CodeCommit, then build a Docker image.
  - Install Docker

```
# check Docker is installed by
docker --version
docker run hello-world
```

- Create Dockerfile

```
FROM ubuntu:12.04
```

```

# Install apache
RUN apt-get update -y
RUN apt-get install -y apache2

# Create a simple web page
RUN echo "Hello Cloud Gurus!!! This web page is running in a
  Docker container in AWS Elastic Container Service." >
  /var/www/index.html

# Configure Apache, set a few variables
RUN a2enmod rewrite
RUN chown -R www-data:www-data /var/www
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2

# expose port 80 and start apache
EXPOSE 80

CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]

```

- Store the Dockerfile in CodeCommit
  - give user CodeCommitFullAccess & AmazonEC2ContainerRegistryPowerUser permission
  - **CodeCommit -> Create repository** -> configure credentials -> clone the repo

```

# set up the credential helper
git config --global credential.helper '!aws codecommit
  credentialhelper $@'
git config --global credential.UseHttpPath true

# clone the repository
git clone https://git-codecommit.us-east-
  1.amazonaws.com/v1/repos/<myreponame>

```

- put your Dockerfile into the repository

```

cp Dockerfile ./<myreponame>
cp buildspec.yml ./<myreponame>
git add .
git commit -m "Added Dockerfile and buildspec.yml"

```

```
git push
```

- Build a Docker image and create a ECS cluster to run it
  - **Elastic Container Service -> Clusters -> Create cluster** -> choose the **EC2 Linux + Networking** template -> create a cluster of 1 instance in the VPC you want
  - **Elastic Container Service -> Repositories -> Create repository** -> build and push your docker image to the repo

```
aws ecr get-login --no-include-email --region us-west-1
# then authenticate with the returned credentials
# build an image
docker build -t mydockerrepo .
# tag it
docker tag mydockerrepo:latest 757250003982.dkr.ecr.us-west-1.amazonaws.com/mydockerrepo:latest
# push it to AWS
docker push 757250003982.dkr.ecr.us-west-1.amazonaws.com/mydockerrepo:latest
```

- **Clusters -> Task definition -> Create new task definition** -> choose **EC2** as the launch type -> **Add container** -> put in the **Port mappings** you need e.g. 80 80
- **Actions -> Create Service**
- Use CodeBuild to auto-build the image

```
version: 0.2
#env
#variables:
# key: "value"
# key: "value"
#parameter-store:
# key: "value"
# key: "value"

phases:
  pre_build:
    commands:
      - echo Logging into Amazon ECR...
      - aws --version
      - $(aws ecr get-login --no-include-email --region us-east-1)
```

```

build:
  commands:
    - echo Build started on `date`
    - echo Building the Docker image...
    - build -t mydockerrepo .
    - docker tag mydockerrepo:latest 757250003982.dkr.ecr.us-west-1.amazonaws.com/mydockerrepo:latest
  post_build:
    commands:
      - echo Build completed on `date`
      - echo pushing to repo
      - docker push 757250003982.dkr.ecr.us-west-1.amazonaws.com/mydockerrepo:latest

```

- commit the updated `buildspec.yml` to CodeCommit repo
- **CodeBuild -> Create project** -> put in your repo name -> select **Privileged** flag -> put in your buildspec file name/add your own build commands -> **Create build project**
- you can override the settings in `buildspec.yml` by adding your own commands in the console when you launch the build
- IAM -> Role -> `codebuild-hello-cloud-gurus-service-role` attach `AmazonEC2ContainerRegistryPowerUser` policy
- **Start build**
- If your build fails, check the build logs in the CodeBuild console or view the full log in CloudWatch

## 8.6 CloudFormation

- manage, configure and provision your AWS infrastructure as code
- a way to completely scripting your cloud environment
- Template
  - YML, JSON
  - upload it to CloudFormation with S3
  - CloudFormation interprets it
  - then makes API calls to create the resources you defined
  - the resulting resources are called a *Stack*
  - example

```

AWSTemplateFormatVersion:"2010-09-09"
Description:"Template to create an EC2 instance"
Metadata:

```

```

Instances:
  Description: "Web Server Instance"

Parameters: #input values
  EnvType:
    Description: "Environment type"
    Type: String
    AllowedValues:
      - prod
      - test

Conditions:
  # only create production resources is EnvType is set to prod
  CreateProdResources: !Equals[!Ref EnvType,prod]

Mappings: #e.g. set values based on a region
  RegionMap:
    us-west-1:
      "ami": "ami-04681a1dbd79675a5"

Transform: # include snippets of code outside the main template
  Name: 'AWS::Include'
  Parameters:
    Location: 's3://MyAmazonS3BucketName/MyFileName.yaml'

Resources: #the AWS resources you are deploying
  EC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: t2.micro
      ImageId: ami-83hy37g2qk943hrd83jd

Outputs:
  InstanceID:
    Description: The Instance ID
    Value: !Ref EC2Instance

```

- Template components
  - *Parameters*: input custom values
  - *Conditions*: e.g. provision resources based on environment
  - *Resources*: AWS resources to create, the only mandatory section
  - *Mappings*: create custom mappings like Region:AMI

- *Transforms*: reference additional code stored in S3, allowing for code re-use, e.g. Lambda code or template snippets / reusable pieces of CloudFormation code. specifying the use of the Serverless Application Model (SAM) for Lambda deployments.
- *Outputs*
- across all regions and accounts
- benefits
  - consistent, fewer mistakes
  - less time and effort
  - version control and peer review your templates
  - free to use
  - can be used to manage updates & dependencies
  - can be used to rollback and delete the entire stack as well
- Lab: Create a CloudFormation Stack
  - **Management Tools -> CloudFormation -> Create new stack -> Upload a template to Amazon S3** -> upload CFTemplate.yml

```
AWS::CloudFormation::Template
AWSTemplateFormatVersion: 2010-09-09
```

```
Description: Template to create an EC2 instance and enable SSH
```

```
Parameters:
```

```
  KeyName:
```

```
    Description: Name of SSH KeyPair
```

```
    Type: 'AWS::EC2::KeyPair::KeyName'
```

```
    ConstraintDescription: Provide the name of an existing SSH key pair
```

```
Resources:
```

```
  EC2Instance:
```

```
    Type: 'AWS::EC2::Instance'
```

```
    Properties:
```

```
      InstanceType: t2.micro
```

```
      ImageId: ami-04681a1dbd79675a5
```

```
      KeyName: !Ref KeyName
```

```
      Tags:
```

```
        - Key: Name
```

```
          Value: My CloudFormation Instance
```

```
  InstanceSecurityGroup:
```

```
    Type: 'AWS::EC2::SecurityGroup'
```

```
    Properties:
```

```
GroupDescription: Enable SSH access via port 22
SecurityGroupIngress:
  IpProtocol: tcp
  FromPort: 22
  ToPort: 22
  CidrIp: 0.0.0.0/0
```

Outputs:

```
InstanceID:
  Description: The Instance ID
  Value: !Ref MyEC2Instance
```

- rollback on failure: rollback if any step has error. choose no if you want to debug what's wrong
- CloudFormation will create a S3 bucket, and stores the template there
- delete all resources, then delete your S3 bucket
- Serverless Application Model (SAM)
  - an extension to CloudFormation
  - define and provision serverless applications
  - simplified syntax for defining serverless resources
    - API, Lambda Functions, DynamoDB
- SAM CLI
- SAM Lab

```
# install Python 3
# install pip
# install AWS SAM CLI
pip install aws-sam-cli

# create a S3 bucket
aws s3 mb s3://<bucket-name> --region us-east-1
```

- **index.js**

```
exports.handler = (event, context, callback) => {
  const reponse = {
    statusCode: 200,
    body: JSON.stringify('Hello Cloud Gurus, this Lambda
function was deployed using SAM.')
  };
  callback(null, reponse);
};
```



- **lambda.yml**

```
AWSTemplateFormatVersion: '2010-09-09'
# this tells CloudFormation this is SAM
Transform: AWS::Serverless-2016-10-31
Resources:
  TestFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs6.10
      Environment:
        Variables:
          S3_BUCKET: <bucket-name>
```

- package your code and deploy it

```
# package your deployment code and upload it to S3
sam package \
  --template-file ./lambda.yml \
  --output-template-file sam-template.yml \
  --s3-bucket <bucket-name>

# deploy your serverless application with CloudFormation
sam deploy \
  --template-file sam-template.yml \
  --stack-name mystack \
  --capabilities CAPABILITY_IAM
```

- **Configure test events -> Test**

- Nested Stacks

- stacks that create other stacks
- allow re-use of CloudFormation code for common use cases
- e.g. standard configuration for a load balancer, web server, application server
- create a template for each common use case (need to be in S3) and reference from within your CloudFormation template
- use the `Stack` resource type
- example

```
Resources:
  Type: AWS::CloudFormation::Stack
  Properties:
```

```
NotificationARNs:
  - String
Parameters:
  AWS CloudFormation Stack Parameters
Tag:
  - Resource Tag
# mandatory
TemplateURL: https://s3.amazonaws.com/.../template.yml
TimeoutInMinutes: Integer
```

- Read the CI/CD whitepaper

## **9 Monitoring**

